Fachhochschule Aargau
Nordwestschweiz

Semesterarbeit

ABOVE – Autonomous Battle Over Endor

Gerstendörfer Thomas, Ia02
Hellstern Thomas, Ia02
Kellenberger Lukas, Ia02
Mühlebach Michael, Ia02

Fachhochschule Aargau
Departement Technik
Studiengang Informatik
Betreuende Dozenten: Josef A. Huber, Dr. R. Käser

Windisch, 23. Juni 2004

Danksagung

Folgende Personen waren auf die eine oder andere Art an unserem Projekt beteiligt und wir, das Projektteam, möchten ihnen allen herzlich danken: **Prof. Dr. Ernst Gutknecht** (Dokumentationsdruck), **Max Hinder** (Tester), **Prof. Josef A. Huber** (Betreuung), **Prof. Dr. Rudolf Käser** (Betreuung), **Hans Kellenberger** (Korrektur), **Christoph Meier** (Tester), **Christine Mühlebach** (Korrektur), **Stefan Mühlebach** (Tester), **Major A. Payne** (3D Models), **Prof. Dr. Jean-Daniel Tacier** (Mathematische Unterstützung), **Arnaud de Wolf** (Tester), **John Wright** (3DS-Format Loader).
Zusätzlich möchten wir auch Craig Reynolds für sein Boids-Modell und George Lucas für seine einmalige Idee danken.

Inhaltsverzeichnis

1 Einleitung **7**

2 Ziel **9**
2.1 Funktionale Anforderungen 9
2.2 Qualitative Anforderungen 10

3 Planung, Dokumente zum Entwicklungs-Prozess **11**
3.1 Milestones 11
3.2 Rollen im Projekt 11
3.3 Vorgehensmodell 11
3.3.1 Projektphasen 11
3.3.2 Programmierstil 12
3.3.3 Release-Builds 12
3.3.4 Dokumentation 13
3.4 Aufwand 14

4 Fachliche Inhalte **15**
4.1 Sprache/Plattform 15
4.2 Architektur 16
4.3 3D/Model/J3D 18
4.3.1 Scene Graph 18
4.3.2 SkyBox 18
4.3.3 Texturkoordinaten 20
4.3.4 Sprites 20
4.3.5 Positionierung der Agenten 21
4.4 Simulationsmodell 23
4.4.1 Obstacle Avoidance 23
4.4.2 Schwarmverhalten 27
4.4.3 Gegnersuche 29
4.4.4 Angriff/Abschuss 29
4.4.5 Starfighter-Verhalten 30

4.4.6 Capital Ship-Verhalten 31
4.4.7 Implementation 31
4.5 Performance der Vektoroperationen 32
4.5.1 Testvorgehen 32
4.5.2 Erkenntnisse 33
4.6 Deployment 34
4.6.1 Java Web Start 34

5 Schlussfolgerungen **37**
5.1 Stand der Entwicklungen 37
5.2 Zukunft 38
5.3 Lerneffekt 38

A Benutzerhandbuch **41**
1 Installation 41
2 Start der Applikation 41
3 Einstellungen 42
4 Navigation in der 3D-Welt 44

B Grössenordnungen und Einheiten in Star Wars **45**

C Glossar **47**

Literaturverzeichnis **49**

Zusammenfassung **51**

Abbildungsverzeichnis

1.1 Szene aus dem Film The Empire Strikes Back 7

4.1 UML Klassendiagramm 17
4.2 Scene Graph von ABOVE 19
4.3 Vektoren des ViewObjects 21
4.4 ABOVE Agent 24
4.5 Lokale Umgebung eines Agenten 25
4.6 Obstacle Avoidance 26

A.1 Settings-Dialog 42

Kapitel 1

Einleitung

Im vierten Semester des Informatikstudiums an der Fachhochschule Aargau steht im Fach Software Engineering jeweils eine Projektarbeit auf dem Programm, deren Gegenstand und Umfang die Studierenden, innerhalb gewisser Grenzen, im Team selbst festlegen können.

In der Filmindustrie ist der Einsatz von Computersimulationen zur Animation von Massenszenen bereits Alltag. Man braucht hier nur an die gigantische Schlacht um Helms Deep im Film *Lord of the Rings: The Two Towers* [7] zu denken. Angesichts dieser Tatsache entschied sich unsere Gruppe, die aus dem Film *Star Wars: Episode IV – Return of the Jedi* [16] bekannte finale Raumschlacht zwischen Vertretern der Rebellenallianz und den Truppen des Galaktischen Imperiums zu simulieren, und dabei gleich noch einige Erfahrungen mit agentenbasierter Künstlicher Intelligenz, wie auch mit der 3D Animation zu machen.

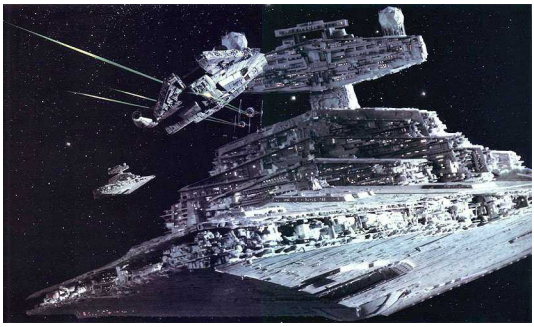

Abbildung 1.1: Szene aus dem Film Star Wars: Episode V – The Empire Strikes Back [14]

Kapitel 2

Ziel

Ziel des Projekts ABOVE ist die Entwicklung eines Programms, das die Grundlage für die Computersimulation der finalen Weltraumschlacht aus dem Film *Return of the Jedi* (1983) zwischen Vertretern der Rebellenallianz und des Imperiums legt. Alternativ soll das Programm auch für den Einsatz als 'intelligenter' Gegner in einem Computerspiel dienen können (das Spiel selbst ist allerdings nicht Gegenstand dieser Projektarbeit).

Auch wenn die Simulation im dreidimensionalen Raum stattfindet, und dies visualisiert werden muss, so liegt der Fokus des Projekts auf der Simulation des Verhaltens von Schwärmen von Raumjägern.

2.1 Funktionale Anforderungen

1. Das Programm simuliert den Flug mindistens einer Gruppe von Jägern im dreidimensionalen Raum als Schwarm von situierten Agenten.

2. Die folgenden Anforderungen stellen optionale Verbesserungen des Modells dar:

   2.1 Die simulierten Raumjäger können Hindernisse erkennen und diesen ausweichen.

   2.2 Die simulierten Raumjäger können zwischen Mitgiedern der eigenen Staffel (des eigenen 'Schwarms'), sowie ihnen feindlich, freundlich und neutral gesinnten Raumschiffen unterscheiden und dementsprechend reagieren.

   2.3 Die simulierten Raumjäger greifen als feindlich erkannte Ziele an.

   2.4 Die Raumjäger folgen innerhalb ihrer Staffel einer Befehlshierarchie: z.B. Leader/Wingman.

3. Die Animation soll auf den Rechnern der Teammitglieder durchschnittlich mindestens 20 FPS erreichen.

4. Das Programm läuft auf mindestens den von den Projektteammitgliedern eingesetzten Plattformen: Windows 2000/XP und Mac OS X 10.3.

5. Optional soll das Programm direkt aus dem Web-Browser gestartet werden, vorausgesetzt ist eine kompatible Runtime-Engine, alle anderen benötigten Bibliotheken werden automatisch installiert.

6. Optional sollen Agenten (Verhaltens-Typ, Aussehen), deren Startbedingungen (Position, Geschwindigkeit) und die Definition der Umgebung (Aussehen, Hindernisse) als *Szene* geladen werden können. Idealerweise werden solche Szenendateien im XML-Format gespeichert.

7. Einstellungen des Benutzerinterfaces sollen zur Laufzeit verändert werden können und persistent sein.

8. Optional kann die Animation für die Nachbearbeitung (z.B. manuelle Kameraführung) in einem 3D Animationsprogramm oder für das Rendern in hoher Qualität in noch zu bestimmender Weise (z.B. RenderMan Interface Bytestream [11]) abgespeichert werden.

2.2 Qualitative Anforderungen

1. Das Projekt soll zeigen, ob sich das vorgeschlagene Modell der autonomen situierten Agenten für eine Spiele-KI eignet.

2. Es soll geprüft werden, wie *stabil* die implementierte KI ist, d.h. ob das erwünschte Verhalten nur in einem sehr engen Parameter-Rahmen auftritt.

3. Die Teammitglieder lernen den Umgang mit der Programmierung von 3D-Applikationen.

4. Die Teammitglieder lernen die speziellen Probleme der KI-Programmierung anhand eines einfachen Beispiels kennen.

Kapitel 3

Planung, Dokumente zum Entwicklungs-Prozess

3.1 Milestones

26.02.2004	Projektidee, Zusammenschluss des Projektteams
12.03.2004	Kick-Off Meeting, Beginn Implementation
26.04.2004	Abgabe eines Zwischenberichts
30.06.2004	Präsentation der Arbeit

3.2 Rollen im Projekt

Projektleiter	Lukas Kellenberger
Infrastruktur	Thomas Gerstendörfer
Dokumentation	Thomas Hellstern
Architektur	Michael Mühlebach und Thomas Gerstendörfer
Modell-Entwicklung	Thomas Gerstendörfer
3D-Entwicklung	Michael Mühlebach
Tuning	Lukas Kellenberger
Codequalität	Thomas Gerstendörfer
Codereview	Thomas Gerstendörfer

3.3 Vorgehensmodell

3.3.1 Projektphasen

1. Inception, Bereitstellung der Infrastruktur und des Glossars
2. 3D Spielwiese
3. Schwarm nach den Original Boids-Regeln

4. Hindernis umfliegen (Sternenzerstörer / Asteroid)

5. Freund-Feind Erkennung / Angriffsmodi

6. Befehlshierarchie (Leader / Wingman)

3.3.2 Programmierstil

Da ABOVE in der Programmiersprache Java geschrieben ist, stützen wir uns hauptsächlich auf den von Sun vorgeschlagenen Programmierstil [17], zusammengefasst und ergänzt in [3]. Variablen, Klassen und sämtliche Kommentare werden ausschliesslich englisch formuliert.

3.3.3 Release-Builds

Im einfachsten Fall, in welchem jeder Projektteilnehmer gleich auf seinem Rechner eine Releaseversion der Software herstellen kann, entstehen oft die folgenden Probleme:

- Durch die Verwendung unterschiedlicher Versionen von Compilern und anderen Dienstprogrammen sind die Versionen kaum vergleichbar.

- Es schleichen sich sehr leicht neue Abhängigkeiten von Bibliotheken Dritter ins Programm, die unbemerkt bleiben, bis sich das Programm beim Kunden oder Anwender partout nicht installieren lassen will.

- Nicht eingecheckte Dateien, die durch Unachtsamkeit Teil des Releases werden, erhöhen die Wahrscheinlichkeit von Fehlern und erschweren das Reproduzieren und Beheben von Fehlern. Falls ausserdem die Möglichkeit vieler Versionskontroll-Werkzeuge, die Versionsnummer automatisch in die Dateien zu schreiben, genutzt wird, ermöglichen die Versionsnummern nicht mehr, den Stand der Software zum Releasezeitpunkt wiederherzustellen.

Um diesen Gefahren zu begegnen entschlossen wir uns bereits zu Beginn des Projekts, alle Releases direkt aus dem CVS, das heisst ausschliesslich mit bekannten, markierten und wiederherstellbaren Dateiversionen, und ausserdem nur auf dem speziellen Build-System zu erstellen. Die Nachvollziehbarkeit wird noch zusätzlich erhöht, indem die Releaseerstellung vollständig durch Skripten kontrolliert wird, und damit

auch alle nötigen Schritte inklusive dem Kopieren auf den Distributionsserver[1] zu jeder Zeit und für alle Projektteilnehmer nachvollziehbar sind.

Mit dieser Infrastruktur ist es möglich, jeden Tag um 12:30 Uhr automatisch einen *Daily Snapshot Release* mit den neusten Änderungen zu erzeugen. Damit werden nicht nur die Build-Skripten ständig getestet, sondern es wird auch laufend geprüft, dass ABOVE ständig Integriert wird, also dass zu jedem Zeitpunkt alle Module soweit zueinander passen, dass das Programm kompiliert. Gleichzeitig wird es einfacher, die aktuellste Version von ABOVE Dritten zu präsentieren und weiterzugeben, ohne dass diese die Entwicklungsumgebung reproduzieren müssen.

3.3.4 Dokumentation

Aufgrund unserer Erfahrungen aus vorangehenden Projekten ist die Dokumentation in die folgenden drei Teile aufgeteilt:

Projektbericht Der Projektbericht dokumentiert den Projektablauf aus der Sichtweise *nach* Ablauf des Projekts. Hier sind unter anderem die Ergebnisse unserer Nachforschungen wie auch alle Designentscheidungen dokumentiert.

Projekt Web Site Die Projekt Web Site[4] dient auch als Visitenkarte des Projekts, ermöglicht in erster Linie die sehr einfache, formale und gut dokumentierte Kommunikation der einzelnen Teammitglieder. Dokumente der Homepage sollten typischerweise zu Projektende entweder leer sein, (z.B. Liste der Fehler) oder in den Projektbericht aufgenommen werden.

API-Dokumentation Die API Dokumentation richtet sich in erster Linie an die anderen Teammitglieder, die die entsprechenden Klassen/Methoden benutzen möchten, oder sich in einen Teil des Projekts einarbeiten möchten. Implementationsdetails werden nur aufgeführt, wenn sie für den Aufrufer von Bedeutung sind. Um die Dokumentations-Kommentare klar und deutlich zu formulieren, stützen wir uns an die in [18] aufgestellten Regeln.

Um nicht am Ende der Projektlaufzeit noch tagelang Dokumentation verfassen zu müssen, notieren wir fortlaufend alle Entscheide, grösseren Designänderungen und erreichten Teilziele im Projektbericht.

[1] `http://www.cs.fh-aargau.ch/~ia02gers/above-releases/`

3.4 Aufwand

Bereits vor Beginn des Projekts war klar, dass der um später die in Kapitel 2 ausformulierten Ziele und den darauf aufbauenden Anforderungen zu erfüllen notwendige Aufwand den Rahmen der üblichen Laborübungen deutlich sprengen würde. Die tatsächlich geleisteten Aufwände pro Projektmitglied pro Periode zwischen zwei Labornachmittagen, und im Total sind in Tabelle 3.1 aufgeführt.

	TG	TH	LK	MM	
8.3 – 14.3	50	5	15	50	120
15.3 – 28.3	84	8	6	84	182
29.3 – 18.4	84	4	15	84	187
19.4 – 25.4	20	4	6	14	44
26.4 – 9.5	24	4	6	40	74
10.5 – 23.5	40	8	10	20	78
24.5 – 6.6	40	4	12	25	81
7.6 – 20.6	30	8	15	30	83
21.6 – 30.6	60	14	35	50	159
Total:	432	59	120	397	**1008**

Tabelle 3.1: Geleistete Arbeitsstunden der Projektmitglieder, in Stunden.

Kapitel 4

Fachliche Inhalte

4.1 Sprache/Plattform

Die erste Aufgabe nach dem Festlegen der Ziele war die Wahl der Entwicklungsplattform, was sowohl das Festlegen der Programmiersprache als auch des verwendeten Toolkits beinhaltete. Tabelle 4.1 liefert eine Übersicht der wichtigsten 3D-Plattformen.

	OpenGL	Direct3D	Java3D
Programmiersprache	C/C++	C/C++	Java
Plattform	fast alle	Windows	Windows, OS X
Komplexität	mittel	hoch	mittel
Zukunft	sicher	sicher	ungewiss
Performance	hoch	hoch	ok

Tabelle 4.1: 3D-Plattformen

Da unser Projekt sowohl unter Windows als auch Mac OS X entwickelt werden soll, kann Direct3D nicht berücksichtigt werden. Nun lässt sich die Frage der Plattform anders formulieren: Gesicherte Performance unter erhöhtem Aufwand für die plattformübergreifende Entwicklung (OpenGL mit C/C++) oder das Risiko, nicht genügend Leistungsreserven für eine aufwändige KI zu haben?

Aufgrund der Tatsache, dass noch nicht alle Teammitglieder mit C oder C++ vertraut sind, und nachdem erste Abklärungen gezeigt haben, dass Java3D genügend nahe an der darunterliegenden Plattform (wieder OpenGL oder Direct3D) liegt, um eine akzeptable Performance zu erreichen, entscheiden wir uns schliesslich für den Einsatz von Java3D.

Um bei Bedarf das Programm auf eine andere Plattform portieren zu können ohne erst das Modell im Anzeige-Code suchen zu müssen,

beschliessen wir, die Simulation so weit wie möglich und sinnvoll von der 3D-Darstellung zu trennen,

4.2 Architektur

Wie bereits in obenstehendem Abschnitt angesprochen war eines der Hauptziele der ABOVE Architektur die möglichst vollständige Enkoppelung des Modells von der 3D-Anzeige. Ein weiteres Ziel war es, die Architektur durch Verwendung der aus [2] bekannten Designpattern zu vereinfachen. Andererseits erforderten die Performanceanforderungen von sowohl der ständigen Neuberechnung des Modells als auch der dreidimensionalen Anzeige einige Kompromisse, insbesondere konnten die verschiedenen Teile nicht so stark entkoppelt werden, wie wir uns das gewünscht hatten. Abbildung 4.1 zeigt die wichtigsten Klassen der resultierenden Architektur und ihre Beziehungen.

Eintrittspunkt der Applikation ist die Klasse `Application`, die als typischer Singleton den Zugriff auf die wichtigsten Systemressourcen verwaltet. Die wichtigsten dieser Ressourcen sind a) das Hauptfenster mit der 3D-Anzeige, und b) die Verwaltung der Benutzereinstellungen.

Die Benutzereinstellungen werden von der Klasse `Settings` verwaltet, welche die eigentliche harte Arbeit wiederum an den mit Java 1.4 eingeführten *Preferences*-Mechanismus delegiert, diesen aber nicht direkt der Anwendung zur Verfügung stellt. Für jede Einstellung steht eine Zugriffsmethode zur Verfügung, welche nicht nur die Typsicherheit garantiert, sondern auch Standardwerte bereitstellt, falls die entsprechende Einstellung nicht vorhanden ist. Eine Liste der unterstützten Einstellungen findet sich ab Seite 3. Um die Einstellungen nicht jedesmal mühsam in der Registry bzw. einer Konfigurationsdatei editieren zu müssen, stellt `Settings` einen Konfigurationsdialog zur Verfügung, was während der Entwicklung das Fokussieren auf unterschiedliche Aspekte deutlich vereinfachte.

Das Hauptfenster der Applikation beinhaltet eine Referenz auf das globale `Universe`-Objekt, welches die Wurzel des für die dreidimensionale Darstellung benötigten Scene Graphs, welcher in Abschnitt 4.3 genauer beschrieben ist, bildet. Des weiteren besitzt das `Universe` eine Referenz auf das `Model`-Objekt, welches die eigentliche Simulationsmodell kapselt und in Abschnitt 4.4 genauer beschrieben ist.

Jeder `Agent` im Simulationsmodell wird in der 3D-Ansicht durch ein `ViewObject` repräsentiert. Die verschiedenen Typen von Agenten, und damit auch Eigenschaften wie das für die Anzeige eingesetzte 3D-

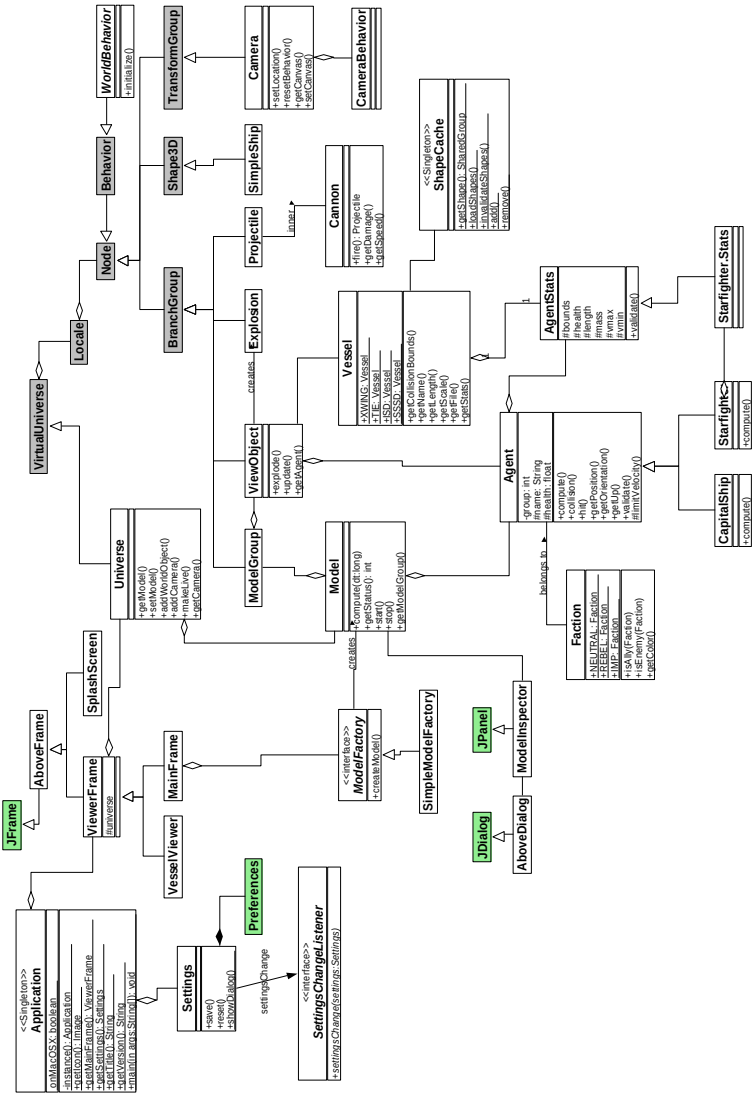

Abbildung 4.1: UML Klassendiagramm

Modell, oder die in `AgentStats` zusammengefassten Parameter welche das Verhalten von Agenten des selben Typs charakterisieren, werden als statische Objekte vom Typ `Vessel` abgelegt.

Da das Laden der 3D Modelle, insbesondere ihrer Texturen, sehr Zeit- und Speicheraufwändig ist, speichert der Singleton der Klasse `ShapeCache` alle bereits geladenen 3D-Objekte zwischen.

4.3 3D/Model/J3D

4.3.1 Scene Graph

Der *Scene Graph* repräsentiert grösstenteils unsere Animation. Genauer genommen ist es weniger ein Graph als viel mehr ein Baum, welcher aus Nodes besteht. Diese Nodes sind in Gruppen und Blätter aufgeteilt. Eine detailliertere Ansicht des *Scene Graph* zeigt die Abbildung 4.2.

Unter Blättern versteht man die eigentlichen Polygone bzw. Geometrien, die in der Szene sichtbar sind. In unserem Fall sind dies die *Shapes* der Agenten und statische Objekte wie Planeten.
Folgende Gruppen werden in ABOVE verwendet:

TransformGroup Dieser Gruppe kann eine Transformationsmatrix übergeben werden, mit welcher all ihre Elemente transformiert werden.

BranchGroup Ein eigener Untergraph wird von dieser Gruppe repräsentiert. Dieser kann während der laufenden Animation aus dem *Scene Graph* entfernt werden, was mit allen andern Gruppen nicht möglich ist.

SharedGroup Um Elemente unter mehrere Gruppen zu hängen, muss diese Gruppe verwendet werden. Sie erlaubt es als einzige *Nodes* unter mehreren Gruppen zu teilen.

4.3.2 SkyBox

Nach dem Einbauen der detailierten Schiffsmodelle sowie der Kameraperspektive, wurde es wichtig eine optisch bestmögliche Orientierungshilfe einzubauen. Die naheliegendste Wahl lag bei einer SkyBox.

Die *SkyBox* umschliesst die Welt in der sich die Simulation bewegt, und definiert ihr Aussehen. Das perfekte Aussehen bei ABOVE ist natürlich ein Sternenhintergrund.

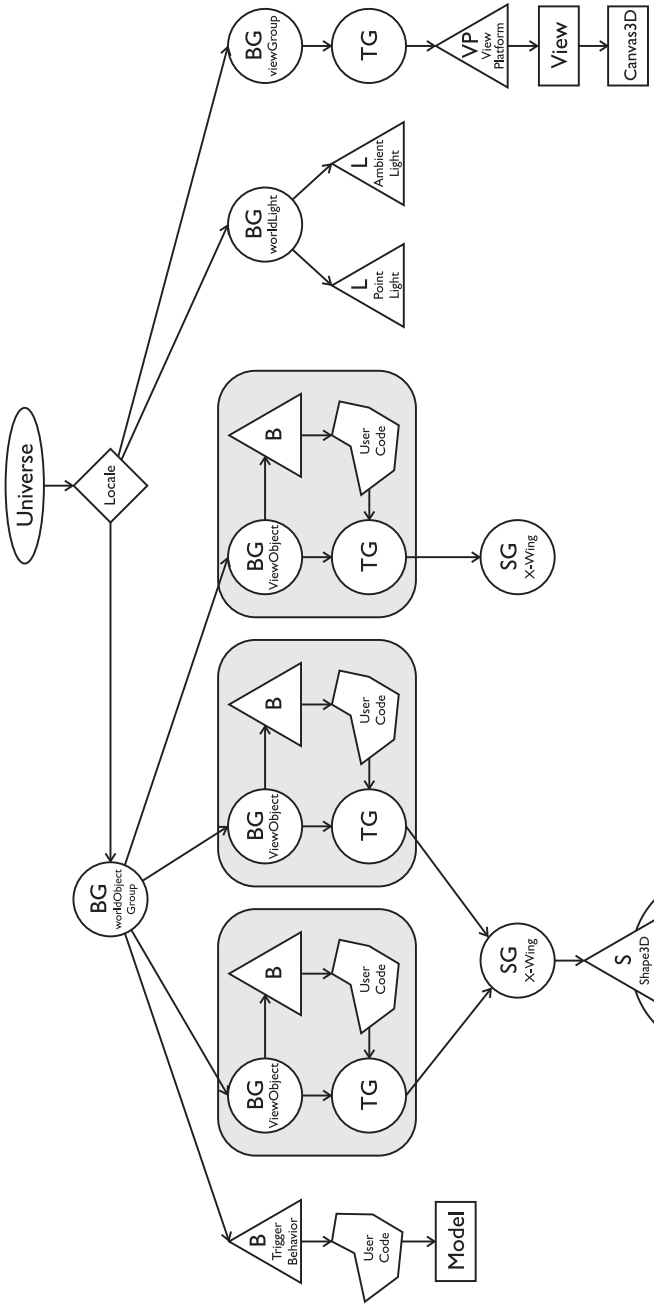

Abbildung 4.2: Scene Graph von ABOVE

Das erste Problem war, dass die von Java3D vorgegebenen Geometrieformen nicht einwandfrei funktionieren. Es war unmöglich die Inversen der Flächennormalen eines Würfels (in Java3D `Box`) berechnen zu lassen. Diese werden jedoch benötigt, damit die Texturen auf der Innenseite des Würfels sichtbar sind.

Aus einem Beispiel von Claude Schwab[21] war zu sehen, dass auch dieser es nicht fertig brachte, die vorgefertigten Geometrien für diesen Zweck zu verwenden. Das naheliegendste war, eigene Klassen für einen `Cube` und eine `SkyBox` zu schreiben, die diesen verwendet.

4.3.3 Texturkoordinaten

Wenn man für eigene Geometrieformen Texturen verwenden will, muss man ihnen entsprechende Texturkoordinaten angeben. Die den Texturkoordinaten entsprechenden Punkte auf dem Textur-Bild, welches beispielsweise im TGA-Format vorliegt, werden danach auf das zuvor definierte Polygon gespannt. Die Werte sind normalerweise zwischen 0 und 1 und stehen somit für das ganze Bild. Hat das Bild eine Grösse von 512×512, so umschliessen die vier Koordinaten: $(0, 0)$, $(0, 1)$, $(1, 1)$ und $(1, 0)$ das ganze Bild.

Im Falle der SkyBox gab es eine spezielle Eigenheit: Die Textur müsste unendlich gross sein, da sich der Sternenhimmel unendlich wiederholt. Dies kann man jedoch ein wenig austricksen: Gibt man an, dass die Werte für die Texturkoordinaten nicht von 0 bis 1 gehen, sondern bis 2, verkleinert man die Grösse des Bildes um den Faktor vier. Dies hat zur Folge, dass die Textur wiederholt wird, da sie nicht auf das Polygon passt.

4.3.4 Sprites

Nach einem ersten Versuch mit texturierten Kugeln beschlossen wir, grosse, weit entfernte Objekte wie Planeten oder den Todesstern als Sprite zu implementieren, denn um einigermassen glatte Kugeln zu erhalten, stieg die benötigte Anzahl Polygone ins Unermessliche.

Hierfür ist die Klasse `OrientedShape3D` sehr hilfreich, denn sie richtet sich immer so aus, dass ihre z-Achse auch bei mehreren Kameraperspektiven in Richtung des Betrachters zeigt. Für Objekte, die keine eigene Geometrie mitbringen, erstellten wir deshalb ein Rechteck mit Seitenverhältnis des Bildes, und fügten das Bild als Textur hinzu.

Eine kleinere Schwierigkeit war, dass Java3D die Ränder der Planeten nicht transparent anzeigte, obwohl die Texturen im PNG Format mit

Transparenz abgespeichert wurden und als `RGBA` (die drei Grundfarben plus Alphakanal) geladen wurden. Die Lösung fanden wir schliesslich in J. Miyamotos Java3D Tutorial [9]: Nur wenn das Transparenzattribut `BLENDED` aktiviert ist, wird die vorhandene Transparenzinformation auch von Java3D ausgewertet.

4.3.5 Positionierung der Agenten

Die im Simulationsmodell enthaltenen Agenten besitzen folgende für die Anzeige relevanten Datenkomonenten: Position $\vec{p}$, *up*-Vektor $\vec{u}$, und die Orientierung $\vec{o}$, wie gezeigt in Abbildung 4.3.5.

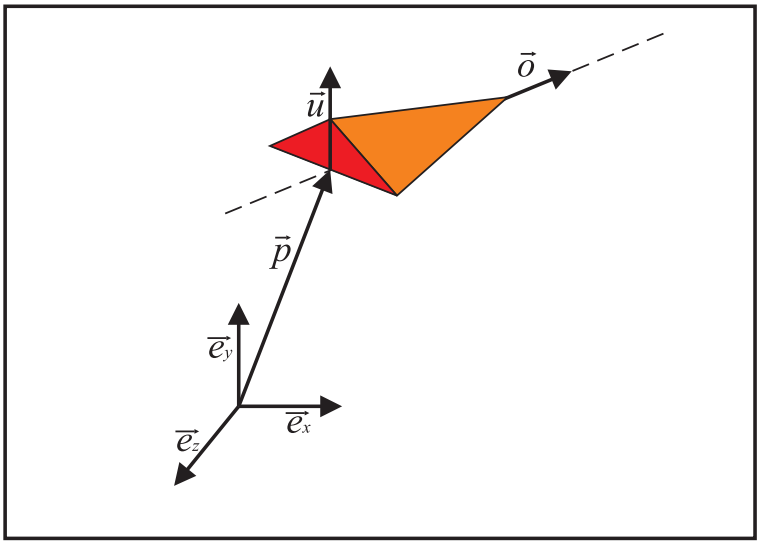

Abbildung 4.3: Vektoren des ViewObjects

Das im Scene Graph eingefügte `Vessel` muss entsprechend diesen Daten transformiert werden, um an der richtigen Position mit der richtigen Orientierung zu sein. Dabei wird davon ausgegangen, dass die Vessels so in die Welt eingefügt werden, dass ihre Orientierung der y-Achse entlangläuft und ihr up-Vektor der z-Achse. Dies geschieht in vier Schritten, welche alle in Matrizen gespeichert werden.

Schritt 1: Rotation um eigene Achse Dazu wird nur der *up*-Vektor benötigt.

$$\delta = \arccos \frac{|u_y|}{\|\vec{u}\|} \tag{4.1}$$

$$\alpha = \begin{cases} \pi - \delta & : \quad \vec{u} \times \vec{o} > 0 \cap u_y < 0 \\ \delta - \pi & : \quad \vec{u} \times \vec{o} \leq 0 \cap u_y < 0 \\ \delta & : \quad sonst \end{cases} \tag{4.2}$$

$$M_1 = \begin{pmatrix} \cos\alpha & 0 & -\sin\alpha \\ 0 & 1 & 0 \\ \sin\alpha & 0 & \cos\alpha \end{pmatrix} \tag{4.3}$$

$$\tag{4.4}$$

Schritt 2: Höhe der Orientierung Hier wird eine Rotation um die x-Achse durchgeführt.

$$\delta = \arcsin \frac{|o_y|}{\|\vec{o}\|} - \frac{\pi}{2} \tag{4.5}$$

$$\beta = \begin{cases} \pi - \delta & : \quad o_y < 0 \\ \delta & : \quad sonst \end{cases} \tag{4.6}$$

$$M_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\beta & -\sin\beta \\ 0 & \sin\beta & \cos\beta \end{pmatrix} \tag{4.7}$$

$$\tag{4.8}$$

Schritt 3: Rotation um die y-Achse Die Rotation um die y-Achse repräsentiert die Ausrichtung des *Vessels* in der Welt.

$$\delta = -\arcsin \frac{|o_x|}{\sqrt{o_z^2 + o_x^2}} \tag{4.9}$$

$$\omega = \begin{cases} \pi - \delta & : \quad o_z > 0 \\ \delta & : \quad sonst \end{cases} \tag{4.10}$$

$$\gamma = \begin{cases} -\omega & : \quad o_x < 0 \\ \omega & : \quad sonst \end{cases} \tag{4.11}$$

$$M_3 = \begin{pmatrix} \cos\gamma & 0 & -\sin\gamma \\ 0 & 1 & 0 \\ \sin\gamma & 0 & \cos\gamma \end{pmatrix} \tag{4.12}$$

$$\tag{4.13}$$

Schritt 4: Positionierung und Zusammenfügen Mit den *Transform-Groups* in Java3D ist es möglich Transformationen in zwei Komponenten auszuführen. Die Rotation wird als Matrix übergeben welche folgendermassen aus den vorherigen Schritten zusammengesetzt wird:

$$M_r = M_3 \cdot M_2 \cdot M_1 \tag{4.14}$$

Die Translation kann als einfacher Verschiebungs-Vektor $\vec{p}$ übergeben werden.

4.4 Simulationsmodell

Das für ABOVE entwickelte Modell simuliert den Kampf, indem in jedes Cockpit ein Agent gesetzt wird, der nur einen Teil der anderen Jäger sieht, und aufgrund dieser Information entscheidet, was als nächstes zu tun ist.

In unserer Simulation bezeichnen wir mit dem Begriff *Verhalten* ein Modell, wie diese Entscheidungsprozesse der Agenten aussehen könnten, um das von aussen sichtbare Verhalten zu erzielen. Das Ergebnis jedes Verhaltens ist die zugehörige 'Steuerkraft', die als gewünschte Richtungsänderung aufgefasst werden kann, und zur Ermittlung der tatsächlichen Richtungsänderung noch an die simulierte Leistungsfähigkeit der Raumjäger angepasst wird (minimale und maximale Geschwindigkeit und Beschleunigung, Simulation von Trägheit).

In den folgenden Abschnitten werden die einzelnen im Modell implementierten Verhalten kurz beschrieben. Dabei ist $\vec{p}$ der Ortsvektor des aktuellen Agenten, $\vec{v}$ sein Geschwindigkeitsvektor, und $\vec{d} := \vec{p}_k - \vec{p}$ die Entfernung des k-ten Agenten vom betrachteten (aktuellen) Agenten. Des weiteren ist α der Winkel zwischen dem Geschwindigkeitsvektor des aktuellen Agenten und $\vec{d}$. Ausserdem werden konfigurierbare, aber für einen Agenten zur Laufzeit konstante Gewichtsparameter allgemein mit μ_x bezeichnet, wie auch die bei Verhalten x berücksichtigte lokale Umgebung durch die maxmimale Entfernung d_x vom Agenten und den maximalen Winkel α_x begrenzt ist. Allgemein ist für ein Verhalten x die Menge K der Agenten in der lokalen Umgebung wie folgt definiert:

$$K = \{\text{agent} \mid \alpha \leq \alpha_x \wedge \|\vec{d}\| \leq d_x\} \tag{4.15}$$

4.4.1 Obstacle Avoidance

Ziel dieses Verhaltens ist es, anderen Objekten auszuweichen. Gegenüber der Beschreibung von Reynolds [10] konnten wir die Komple-

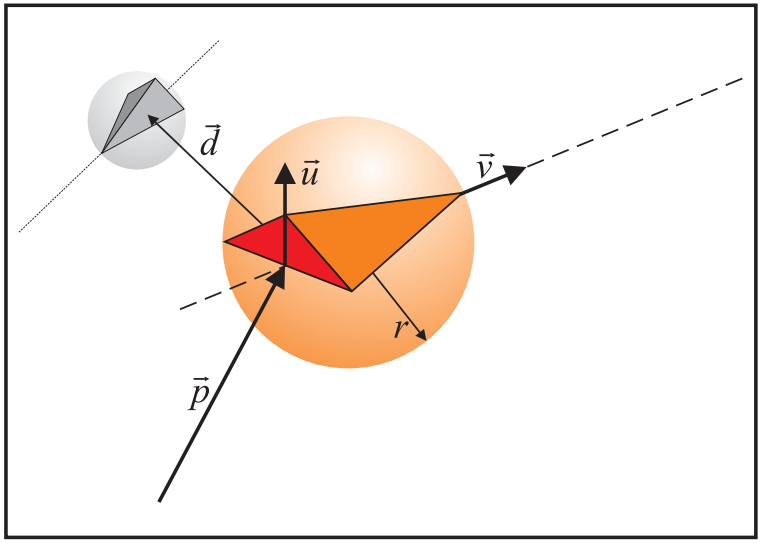

Abbildung 4.4: ABOVE Agent

xität deutlich verringern, da in unserem Modell die Geschwindigkeiten von Jägern und Capital Ships sehr unterschiedlich sind, und die Rechenschrittlänge Δt gering genug, dass Capital Ships, die die grössten Hindernisse stellen, praktisch als statische Objekte betrachtet werden können.

Der verwendete Algorithmus konzentriert sich nur auf das nächste, also am wenigsten weit entfernte, Objekt, welches in der ungefähren Flugbahn des Agenten liegt (beschränkt durch den Winkel α_{thres} und die Entfernung σ).

σ hängt von der aktuellen Geschwindigkeit des Agenten ab (der Agent muss um so weiter voraus planen, je schneller er fliegt), und wird mit Hilfe der Länge des Agenten (der doppelte Radius r der Hüllkugel) und dem festen Parameter μ_{d_o} wie folgt berechnet:

$$\sigma = (\|\vec{v}\| + 2r) \cdot \mu_{d_o} \tag{4.16}$$

Da der Winkel α zwischen dem Richtungsvektor des Agenten und $\vec{d} := \vec{p}_{obstacle} - \vec{p}$ nicht viel darüber aussagt, ob Hindernisse unterschiedlicher Grösse tatsächlich im Weg liegen, ermitteln wir als nächstes den Winkel ϕ, unter welchem die Kugel mit Radius $r + r_{obstacle}$ erscheint, wobei r

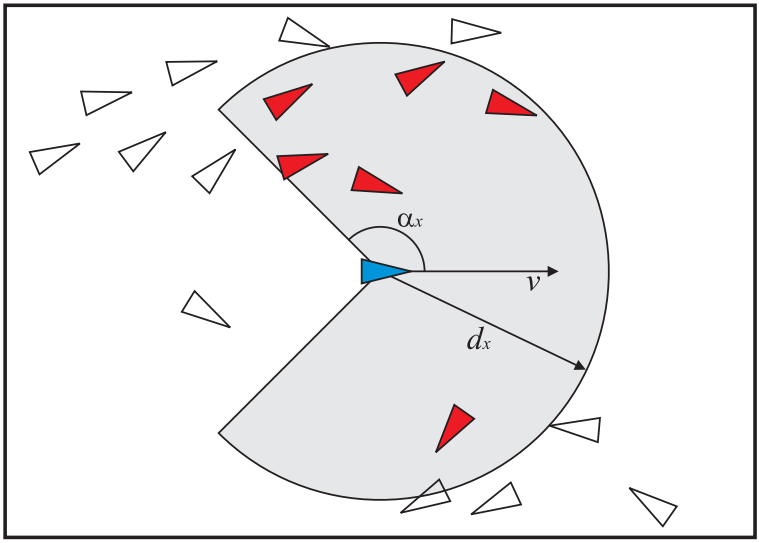

Abbildung 4.5: Lokale Umgebung, beschränkt durch den Winkel α_x und die Entfernung d_x

der Radius der Hüllkugel des betrachteten Agenten und r derjenige der Hüllkugel des Hindernisses ist. Die Addition der beiden Hüllkugelradien transformiert das Problem in ein einfacheres, bei dem nur mehr ein Punkt an einer Kugel vorbeikommen muss, anstelle von zwei Kugeln unterschiedlicher Grösse (Abbildung 4.6).

$$\phi = \arcsin \frac{r + r_{obstacle}}{\|\vec{d}\|} \tag{4.17}$$

Die Taylor-Reihenentwicklung von arcsin beginnt wie folgt:

$$\arcsin x = x + \frac{x^3}{6} + \frac{3x^5}{40} + \frac{5x^7}{112} + \dots \tag{4.18}$$

Da die Randbedingungen dafür sorgen, dass die Winkel klein sind, und ausserdem an ϕ nur geringe Anforderungen bezüglich Genauigkeit gestellt werden, genügt in unserem Fall das erste Glied der Taylorreihe von arcsin. Also approximieren wir ϕ wie folgt:

$$\phi \approx \tilde{\phi} = \frac{r + r_{obstacle}}{\|\vec{d}\|} \tag{4.19}$$

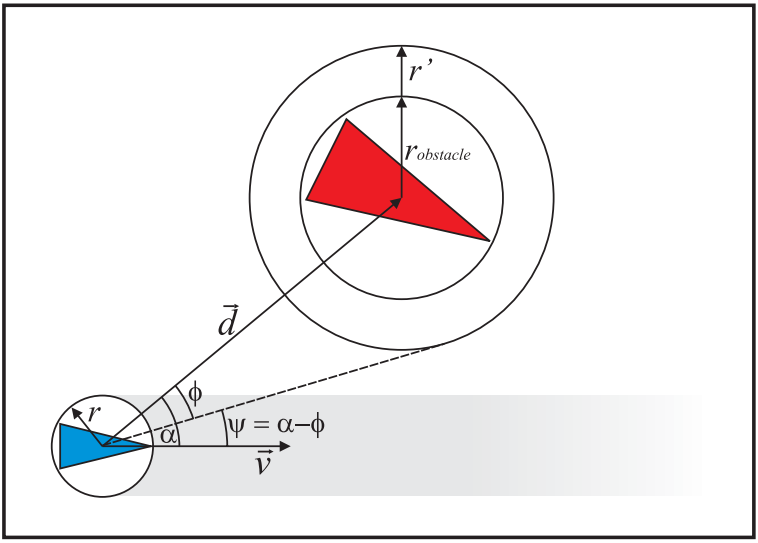

Abbildung 4.6: Obstacle Avoidance

Durch Subtraktion dieser Grösse vom Winkel α zwischen der Bewegungsrichtung des Agenten $\vec{v}$ und $\vec{d}$ erhalten wir einen Winkel, der ausdrückt, wie stark wir von der aktuellen Richtung abweichen müssen, um dem Hindernis auszuweichen. Ist dieser Winkel $\psi := \alpha - \tilde{\phi}$ kleiner als der von der Agilität des Agenten abhängige Schwellwert ψ_{thres}, so muss der Agent das Hindernis berücksichtigen.

In unserem Modell soll die Stärke des Ausweichens von der Kollisionsgefahr, ausgedrückt durch die Entfernung zum Hindernis, abhängig sein. Die Ausweichrichtung $\vec{o}$ wird bestimmt durch Subtraktion der normalisierten Entfernung zum Hindernis von der normalisierten Flugrichtung (*heading*), normalisiert, und mit einem situationsabhängigen Skalierungsfaktor und dem Gewichtsparameter μ_o multipliziert.

$$\vec{f_o} = \frac{\frac{\vec{v}}{\|\vec{v}\|} - \frac{\vec{d}}{\|\vec{d}\|}}{\left\| \frac{\vec{v}}{\|\vec{v}\|} - \frac{\vec{d}}{\|\vec{d}\|} \right\|} \cdot \frac{\mu_o}{\psi^2 + \frac{\|\vec{d}\|^2}{\rho} + 1} \tag{4.20}$$

Um das Ausweichverhalten möglichst natürlich aussehen zu lassen, nimmt der Skalierungsfaktor sowohl mit dem Quadrat der Entfernung zum Hindernis als auch mit dem Quadrat des Gefährdungswinkels ψ ab,

was unnatürlich wirkende Sprünge beim Überschreiten der Schwellwerte verhindert, und gleichzeitig dafür sorgt, dass ein Agent, der in die Nähe eines grossen Hindernisses gedrängt wurde, auch genügend Gegensteuer gibt.

Der Parameter ρ gleicht die unterschiedlichen Wertebereiche von ψ und $\|\vec{d}\|$ einander an und wird mit σ aus Gleichung 4.16 wie folgt berechnet:

$$\rho = \frac{\sigma^2}{\alpha_{thres}} \tag{4.21}$$

4.4.2 Schwarmverhalten

Das in unserem Modell eingesetzte Schwarmverhalten basiert auf dem *Boids*-Modell von Craig W. Reynolds [8][10]. Dieses Modell eigent sich für die Simulation von Vogelschwärmen, da sich der Schwarm beim Umfliegen von Hindernissen aufteilen kann, wie dies auch bei Vogelschwärmen zu beobachten ist, und bei anderen Modellen wie beispielsweise bei *Follow the Leader* nicht berücksichtigt wird.

Die Komplexität des resultierenden Verhaltens eines Schwarmes solcher Boids ist emergent. Es gibt beispielsweise keine Regel für das Aufteilen des Schwarms, sondern das ganze Verhalten beruht auf nur drei einfachen Regeln:

Cohesion, alle Teilnehmer des Schwarms wollen nahe bei anderen Mitgliedern des Schwarms sein, *Alignment*, alle Teilnehmer wollen in die selbe Richtung fliegen wie die sie umgebenden Schwarmteilnehmer, und schliesslich *Separation*, alle Teilnehmer des Schwarms wollen genügend Abstand voneinander halten, um eine Kollision zu vermeiden.

Cohesion beschreibt, wie die einzelnen Schwarmteilnehmer nahe bei den anderen Teilnehmern sein wollen. Dazu genügt es, den Massenmittelpunkt M des lokalen Schwarms zu ermitteln, und anschliessend in die entsprechende Richtung zu steuern.

Sei k die Anzahl Agenten im lokalen Schwarm, und sei $\vec{p}_k$ der Ortsvektor des k-ten Agenten, dann wird der Ortsvektor $\vec{p}_M$ des Mittelpunktes des lokalen Schwarms wie folgt berechnet:

$$\vec{p}_M = \frac{1}{k} \cdot \sum_k \vec{p}_k \tag{4.22}$$

Zusammen mit dem Ortsvektor $\vec{p}$ des betrachteten Agenten, kann nun

die “Kohäsionskraft” $\vec{f_c}$ ermittelt werden:

$$\vec{f_c} = \frac{\vec{p}_M - \vec{p}}{\|\vec{p}_M - \vec{p}\|} \tag{4.23}$$

Alignment beschreibt das Bestreben der Agenten mit gleicher Geschwindigkeit in die selbe Richtung zu fliegen. Hierfür genügt es, die Geschwindigkeitsvektoren aller Agenten des lokalen Schwarms zu mitteln, und schon ist die gewünschte Geschwindigkeit bestimmt. Um die Animation flüssiger zu gestalten berücksichtigt unser Modell nur die Richtung, nicht aber den Betrag der Geschwindigkeit.

Sei $\vec{v}$ der Geschwindigkeitsvektor des betrachteten Agenten, und $\vec{v}_k$ derjenige des k-ten Agenten, dann berechnet sich die “Alignmentkraft” $\vec{f_a}$ wie folgt:

$$\vec{f_a} = \frac{\vec{v}}{\|\vec{v}\|} - \frac{\sum_k \vec{v}_k}{\|\sum_k \vec{v}_k\|} \tag{4.24}$$

Separation (Aligned Collision Avoidance) beschreibt, wie die einzelnen Teilnehmer des Schwarms versuchen, genügend Abstand voneinander zu wahren, um nicht zusammen zu stossen. Dies ist deutlich einfacher als das in Kapitel 4.4.1 beschriebene allgemeine Ausweichen, da sich alle beteiligten Agenten bereits mit ähnlicher Geschwindigkeit in die gleiche Richtung bewegen. Folglich muss nur mehr die Entfernung zu den nächsten Agenten berücksichtigt werden.

Sei k die Anzahl von Agenten des selben Schwarms im Blickfeld des betrachteten Agenten. Und sei $\vec{d_k}$ der Distanzvektor zum k-ten Agent, dann wird die “Separationskraft” $\vec{f_s}$ wie folgt ermittelt:

$$\vec{f_s} = \sum_k -\frac{1}{\|\vec{d_k}\|^2} \cdot \frac{\vec{d_k}}{\|\vec{d_k}\|} \tag{4.25}$$

Flocking: Bereits durch einfache Addition der Steuervektoren aus *Separation*, *Alignment* und *Cohesion* entsteht ein erstaunlich komplexes und lebensähnliches Verhalten. Um aber eine noch feinere Kontrolle über das Verhalten zu erreichen, schränken wir für jedes Teilverhalten den lokalen Schwarm gesondert ein (bestimmt durch je einen Winkel und eine max.Ẽntfernung) und gewichten ausserdem jedes Teilverhalten bei der Addition, wodurch alleine für Flocking neun Parameter nötig sind.

$$\vec{f}_{flocking} = \vec{f_c} \cdot \mu_c + \vec{f_a} \cdot \mu_a + \vec{f_s} \cdot \mu_s \tag{4.26}$$

4.4.3 Gegnersuche

Ähnlich dem oben beschriebenen *Cohesion*-Aspekt des Schwarmverhaltens bestimmen wir auch bei der Suche nach Gegnern den Mittelpunkt M aller gegnerischen Agenten innerhalb des Radarbereichs d_{radar}. Da allerdings die Gegner in den meisten Fällen relativ weit entfernt sind, beziehen wir ausserdem die aktuelle Geschwindigkeit der Agenten in Abhängigkeit ihrer Entfernung mit ein, berechnen also den Mittelpunkt $\vec{p}_M$ zum Abfangzeitpunkt, und daraus die 'Such-Steuerkraft' $\vec{f}_{seek}$.

$$\vec{p}_M = \frac{\sum_k \left(\vec{p}_k + \vec{v}_k \cdot \frac{\|\vec{d}_k\|}{d_{radar}}\right)}{k} \tag{4.27}$$

$$\vec{f}_{seek} = \frac{\vec{p}_M - \vec{p}}{\|\vec{p}_M - \vec{p}\|} \cdot \mu_{seek} \tag{4.28}$$

Sobald mindestens einer der gegnerischen Agenten innerhalb des Angriffsbereichs, der durch den Winkel α_{attack} und die Entfernung d_{attack} bestimmt wird, ist, wird für jedes dieser potentiellen Ziele der Wert ρ ermittelt, und dasjenige mit dem kleinsten Wert ρ als bestes Ziel ausgewählt.

$$\rho = \|\vec{d}\| + \alpha \cdot \mu_{FrontalAttackPriority} \tag{4.29}$$

4.4.4 Angriff/Abschuss

Wurde gemäss des in Kapitel 4.4.3 beschriebenen Verhaltens ein Ziel ausgewählt, versucht der Agent diesem Ziel zu folgen, sich in optimale Schussposition zu bringen und das Ziel abzuschiessen.

$$\vec{f}_{attack} = \frac{\vec{d}}{\|\vec{d}\|} \cdot \mu_{attack} + \frac{\vec{v}_{target}}{\|\vec{v}_{target}\|} \cdot \mu_{attack-align} \tag{4.30}$$

Ist das Ziel nun innerhalb des durch den Winkel α_{fire} begrenzten Schussfeldes, so wird ein Schuss auf das Ziel abgefeuert. Da die maximale Feuerrate limitiert ist, kann erst wieder ein Schuss abgefeuert werden, wenn genügend Zeit verstrichen ist.

Sobald das Ziel den Angriffsbereich verlässt, gilt es als verloren, und das Gegnersuche-Verhalten muss erst wieder ein neues Ziel ermitteln, ehe dieses angegriffen werden kann.

4.4.5 Starfighter-Verhalten

Im Starfighter-Verhalten bilden wir das komplexe Verhalten eines Raumjägers nach. Dazu trafen wir die folgenden Annahmen über das Verhalten:

1. Die Piloten sind in Staffeln organisiert. Das in Kapitel 4.4.2 beschriebene Flocking-Modell ist geeignet, diese Staffeln zu beschreiben.

2. Das komplexe Verhalten kann durch geschickte, situationsbezogene Auswahl beziehungsweise durch Kombination einfacher Grundverhalten beschrieben werden.

3. Es sollen primär Schiffe der selben Kategorie angegriffen werden. Mit anderen Worten, gegnerische Jäger, nicht Sternenzerstörer, sind das primäre Ziel.

Bereits zu Beginn war klar, dass die resultierenden Steuerkraftvektoren der einzelnen Grundverhalten nicht einfach gewichtet und addiert werden können, sondern dass bestimmte Verhalten wichtiger sind als andere. Wenn das aktuelle Ziel sich hinter einem Hindernis befindet, muss dieses Hindernis erst umflogen werden, ehe das Ziel angegriffen werden kann. Vergleiche hierzu auch [10] und [1].

Durch geschickte Wahl der Parameter μ_{d_o} aus Gleichung (4.16) und μ_o aus (4.20) wird es allerdings möglich, genau diesen Aspekt nicht mehr getrennt zu behandeln, sondern nur mehr zu unterscheiden, ob ein Ziel vorhanden ist oder nicht. Wie aus Gleichung (4.31) ersichtlich wird beim Verfolgen eines Ziels nur mehr der *Separation*-Aspekt des Schwarmverhaltens berücksichtigt, um Kollisionen zu vermeiden.

Die Simulation von Trägheit hat sich in Versuchen entgegen den Erwartungen kaum auf den visuellen Eindruck der Simulation ausgewirkt, so dass die aktuelle Version nur mehr die Geschwindigkeit gemäss Gleichung (4.32) auf den durch v_{min} und v_{max} begrenzten Bereich beschränkt.

Zu guter Letzt werden die neue Position des Jägers wie auch der neue up-Vektor $\vec{u}$ berechnet, wobei der up-Vektor nur sehr schwach geändert wird, um relativ langsame Rollbewegungen zu erreichen, was den visuellen Eindruck verbessert.

$$\vec{f} \quad = \quad \begin{cases} \vec{f}_{attack} + \vec{f}_s \cdot \mu_s + \vec{f}_o & \text{, falls Ziel bekannt} \\ \vec{f}_{seek} + \vec{f}_{flocking} + \vec{f}_o & \text{, sonst} \end{cases} \tag{4.31}$$

$$\begin{aligned}
\vec{v} &= \vec{v}_t + \vec{f} \cdot \mu_{agility} \\
\vec{v}_{t+\Delta t} &= \begin{cases} \vec{v} \cdot \frac{v_{max}}{\|\vec{v}\|} & ,\text{für} \|\vec{v}\| > v_{max} \\ \vec{v} \cdot \frac{v_{min}}{\|\vec{v}\|} & ,\text{für} \|\vec{v}\| < v_{min} \\ \vec{v} & ,\text{sonst} \end{cases} \qquad (4.32) \\
\vec{p}_{t+\Delta t} &= \vec{p}_t + \vec{v}_{t+\Delta t} \cdot \Delta t \qquad (4.33) \\
\vec{u}_{t+\Delta t} &= \vec{u}_t + \mu_{roll} \cdot \left(\frac{\vec{u}_t}{\|\vec{u}_t\|} - \frac{\vec{f}}{\|\vec{f}\|} \right) \qquad (4.34)
\end{aligned}$$

4.4.6 Capital Ship-Verhalten

In der aktuellen Version beschränkt sich ABOVE darauf, die Kampfhandlungen zwischen Raumjägern zu simulieren, wodurch Schlachtschiffe zu dekorativen Hindernissen reduziert werden. Dementsprechend einfach ist das Verhalten dieser *Capital Ships*: sie bewegen sich stur in der einmal eingeschlagenen Richtung weiter, schliesslich muss ein Sternenzerstörer von 1.6km Länge (Tabelle B.1) auch niemand ausweichen. Also wird der neue Ortsvektor $\vec{p}_{t+\Delta t}$ wie folgt mit Hilfe des vorangehenden Ortsvektors $\vec{p}_t$, der Geschwindigkeit $\vec{v}$ und dem verstrichenen Zeitabschnitt Δt berechnet, $\vec{u}$ und $\vec{v}$ bleiben invariant.

$$\vec{p}_{t+\Delta t} = \vec{p}_t + \vec{v} \cdot \Delta t \qquad (4.35)$$

4.4.7 Implementation

Aufhängepunkt für das Modell ist die `Model`-Klasse, die alle Agenten der Simulation enthält, und die Methode `compute(dt)` anbietet, die den nächsten Simulationsschritt von *dt* Millisekunden ausführt, und dazu im Wesentlichen auf allen Agenten die entsprechende `compute()`-Methode aufruft.

Zu Beginn war geplant, die unterschiedlichen Verhalten im Sinne des State oder Strategy Design Patterns [2, S. 305-324] zu kapseln, um eine hohe Modularisierung und Wiederverwendbarkeit zu erreichen.

Allerdings muss jedes Verhalten als erstes die Agenten der lokalen Umgebung auswählen, wozu für jeweils n Agenten die Entfernung und der Winkel α aufwändig berechnet werden müssen, der Aufwand also in der Grössenordnung n^2 liegt. Auch wenn es für die meisten Verhalten möglich ist, einen Grossteil der aufwändigen Berechnungen nur einmal pro Agent und Rechenschritt auszuführen, so muss doch meistens ein signifikanter Teil der Operationen $n \cdot k$ Mal pro Rechenschritt

durchgeführt werden, wobei $k \leq n$ die Anzahl Agenten in der lokalen Umgebung ist.

Da die Berechnung des Modells neben der dreidimensionalen Anzeige limitierender Faktor für die Simulationsperformance ist, und nicht wie diese zu einem grossen Teil von der Grafikhardware übernommen wird, entschieden wir uns, das gesamte Verhalten jeweils in einer speziellen Subklasse von `Agent` vollständig, und in aufwandoptimierter Form zu implementieren. Es ist allerdings klar festzuhalten, dass bei diesem Vorgehen die Wartbarkeit wesentlich erschwert wird: Die Implementation für die `Starfighter`-Klasse besteht aus rund 150 Zeilen, gespickt mit Abhängigkeiten, um doppelte Berechnungen auszuschliessen.

Eine Möglichkeit, dieses Problem zu umgehen, wäre der Einsatz räumlich organisierter Datenstrukturen, wie beispielsweise einer Adaption der in [13] vorgestellten Suchbäume für die mehrdimensionale Bereichssuche, weiterentwickelt zu sogenannten BSP (Binary Space Partitioning) Bäumen [15]. Bei unserem Modell kann sich die Lage auf Grund der relativ hohen Geschwindigkeit der Jäger innert 10 Schritten komplett ändern. Damit verursacht das Aufrechterhalten der räumlich organisierten Struktur einen grossen Aufwand. Da auch die für ABOVE zur Verfügung stehende Zeit beschränkt ist, beschlossen wir, auf solche spezialisierten Datenstrukturen zu verzichten.

4.5 Performance der Vektoroperationen

Da unsere Simulation grosse Rechensourcen beansprucht, war es uns ein Anliegen, häufig gebrauchte Vektoroperationen auszumessen. Zeitintensive Operationen sollen durch schnellere Algorithmen ersetzt werden.

Die Messungen basieren auf den Vektoroperationen aus der Klasse Vector3f, welche in Java3D verteilt ist und in der von Sun gelieferten Java-Erweiterung javax.vecmath zu finden ist. Alle Operationen wurden mit Float (einfacher) Genauigkeit durchgeführt und die dafür benötigte Zeit in Nanosekunden festgehalten.

Alle Messungen wurden auf einem Intel Pentium Pro 4 mit 1800 MHz, 512MB RAM, Windows 2000 SP4 und Java 1.4.2_03-b02 (HotSpot Client VM) durchgeführt. Ähnliche Resultate wurden auch auf anderen Rechnern mit verschiedenen Plattformen erzielt.

4.5.1 Testvorgehen

Für die Realisierung wurde vorgänging ein Testprogramm entwickelt, welche alle Operationen nacheinander ausführt. Gemessen wurden die von uns häufiggebrauchten Operationen Addition, Subtraktion, Multiplikation, Zwischenwinkel und Kreuzprodukt zweier Vektoren, sowie Skalierung, Normalisierung und Längenbestimmung eines Vektors. Die erhaltenen Resultate wurden mit Hilfe des Computer-Algbra-System Mathematica 4.2 ausgewertet.

1. Testversuch Für den ersten Testversuch wurde ein Array mit 500 Vektoren mit Float-Zufallszahlen erstellt. Dieser Array wurde jeweils 100 Mal auf eine Operation angewandt. So ergaben sich pro Durchlauf 50000 Berechnungen. Damit eine repräsentative Statistik entstand, wurde ein Durchlauf 100 Mal wiederholt. So ergaben sich pro Operation 100 Messwerte, welche für die Testauswertung in einer Datei gespeichert wurden. Für den ganzen Testversuch wurden 85 Mio. Operationen ausgeführt.

Probleme Bei der graphischen Darstellung der Werte wurden Schwankungen sichtbar in den Messungen. Einige Messwerte wiesen eine unakzeptable Abweichung von 40% vom Erwartungswert auf. Trotzdem gab uns dieser erste Versuch einen ersten Überblick über die Relationen unter den einzelnen Operationen. Mögliche Ursache für diese Ausreisser waren, dass wir unter der Genauigkeit der Zeitmessung der einzelnen Operationen gemessen hatten. Als Konsequenz wurde ein weiterer Testversuch angesetzt. Siehe 4.5.1

2.Testversuch Wiederum wurde ein Array von 500 Vektoren angelegt. Diesmal wurde der Array 2000 Mal auf eine Operation angewandt. Jetzt ergaben sich 1 Mio. Berechnungen pro Operation. Für die Auswertung wurde der Durchlauf 50 Mal wiederholt und wieder in Mathematica ausgewertet. Eine Übersicht ist in Tabelle 4.2 zu finden. Diesmal entsprachen die Messergebnisse den Vorstellungen. Somit sind auch die Annahmen der Ursachen über die schlechten Werte im ersten Testversuch bewiesen.

4.5.2 Erkenntnisse

Die Testversuche haben gezeigt, dass unter den verschiedenen Methoden grosse Zeitdifferenzen bestehen. Interessant ist der Unterschied von fast

Operation	Messwert [ms]	Index
`set(v1)`	35.99	1.00
`lengthSquared()`	37.00	1.03
`scale(float)`	44.40	1.23
`dot(v1)`	45.62	1.27
`scale(float, v1)`	46.28	1.29
`add(v1)`	64.66	1.80
`sub(v1)`	65.83	1.83
`add(v1, V2)`	68.61	1.91
`sub(v1, v2)`	70.12	1.95
`set(v1) + add(v2)`	75.17	2.09
`cross(v1, v2)`	86.17	2.39
`Math.abs(float`	90.55	2.52
`length()`	234.90	6.53
`normalize()`	294.40	8.28
`normalize(v1)`	315.79	8.77
`angle(v1)`	867.39	24.10

Tabelle 4.2: Messwerte des 2.Testversuchs mit 50 Durchläufen und einer Arraylänge von 500 Elementen.

10% zwischen den beiden verschiedenen `add`-Operationen. Eine weitere interessante Erkenntnis ist, dass die Verwendung von der Zwischenwinkelbestimmung von zwei Vektoren möglichst vermieden werden sollte, da diese Operation mit Abstand am meisten Rechenzeit benötigt. Häufig genügt für die Simulation nur eine Annäherung des Winkels, was mit einem viel schnelleren Algorithmus bestimmt werden kann. Ebenfalls zeigen die Versuche, dass bei der Längenbestimmung eines Vektors wenn möglich die Methode `lengthSquared()` verwendet werden sollte, da diese im Gegensatz zu der `length()` Methode fast 10 Mal schneller ist. Oft kann es sogar von Nutzen sein, das Quadrat der Länge für weitere Berechnungen zu gebrauchen.

4.6 Deployment

Der Einfachheit halber packen wir alle Klassen wie auch die zur Laufzeit benötigten Resourcen wie z.B. 3D-Modelle, Texturen und Bilder in eine startbare Jar-Datei, und packen das Ganze anschliessend zusammen mit dem kompletten Quelltext und der Dokumentation in ein Zip-Archiv.

4.6.1 Java Web Start

Um das Testen neuer Versionen auf verschiedenen Rechnern einfacher zu gestalten, wollten wir ursprünglich ermöglichen, ABOVE mittels Java Web Start direkt aus dem Web-Browser zu starten, und nicht zuletzt, um die lokale Installation zu vereinfachen. Leider traten beim Testeinsatz vom JNLP die folgenden Probleme auf:

1. Die JNLP-Spezifikation [19] verlangt, dass der Application Descriptor die sog. *codebase* als voll qualifizierte URI enthält. Es ist folglich nicht möglich, den Application Descriptor mit den Jar-Dateien in eine Zip-Datei zu packen, und anschliessend die ausgepackten Verzeichnisse auf dem Webserver der Wahl zu speichern, ohne entweder den Deployment Deskriptor manuell anzupassen, oder den Deskriptor vom HTTP-Server dynamisch erstellen zu lassen.

2. Die Installation der für Java3D benötigten, nativen Bibliotheken ist zwar prinzipiell möglich, aber mangels direkter Unterstützung von Sun äusserst aufwändig und schwierig zu testen. Man müsste für jede unterstützte Plattform erst die entsprechende Java3D-Version auf einem Testgerät installieren, anschliessend die benötigten Bibliotheken indentifizieren und ein Jar packen und das entsprechende JNLP-File schreiben. Zusätzlich müsste das Jar signiert werden, und zwar mit einem Schlüssel, den man bei einer CA zertifiziert hat, deren Root-Zertifikat mit Java Web Start ausgeliefert wird. Dieser Spass ist nun leider nicht nur teuer, sondern erzeugt auch einigen administrativen Overhead, und automatische Builds sind praktisch unmöglich – ausser man wirft all die mühsam erkaufte Sicherheit über Bord und speichert den Schlüssel im Source-Repository.

3. Erfahrungen einiger Projektmitglieder aus Projekten in der Industrie zeigen, dass Java Web Start im besten Fall in nur rund 80% der Fälle funktioniert, und bekannte, gravierende Fehler nicht behoben werden.

Diese Schwierigkeiten wiegen die Vorteile des Einsatzes von Java Web Start für unser Projekt bei weitem auf, weshalb wir uns entschlossen, JNLP nicht zu unterstützen.

Kapitel 5

Schlussfolgerungen

5.1 Stand der Entwicklungen

Keines der Projektmitglieder hatte vor Beginn des Projektes Erfahrungen mit der 3D-Programmierung oder der KI-Entwicklung sammeln können, unsere Erwartungen waren deshalb auch entsprechend gering. Wir waren umso erstaunter, als bereits kurz nach Projektbeginn das Boids-Modell nach Reynolds [8] stabil lief, woraufhin wir die Anforderungen nach oben korrigierten.

Zu Projektende sind alle Muss-Anforderungen implementiert, ein grosser Teil der optionalen Anforderungen ist ebenfalls implementiert oder zumindest vorbereitet. Einzig die Schnittstelle für den Export der berechneten Simulationsdaten zu externen 3D-Visualisierungsprogrammen ist noch nicht vorhanden.

Zur Zeit sind die folgenden Fehler bekannt und noch offen:

Restart-Bug Wird die Simulation zurückgesetzt, bricht in rund 5% der Fälle das Programm tief im Java3D-Code mit einer `NullPointerException` ab. Wir vermuten, dass dieses Problem aufgrund mangelnder Thread-Synchnronisation auftritt, konnten es aber noch nicht genauer analysieren, da es nur sehr selten auftritt.

Model Inspector Werden die Daten der aktuellen Simulation im *Model Inspector* angezeigt, und die Simulation zurückgesetzt, zeigt der Model Inspector weiterhin den Endstand des alten Modells an. Dieses Problem liesse sich sehr elegant mit dem Observer Design-Pattern [2] lösen, wurde aber aufgrund anderer Prioritäten auf einen späteren Release verschoben.

5.2 Zukunft

Beim Abschluss der Implementationsphase hatten wir noch eine Reihe von Ideen, welche noch nicht in die Applikation eingebaut sind. Da die meisten von ihnen schon seit Projektbeginn bestanden, sind die Ansätze dafür meistens schon vorhanden. Deshalb wäre eine Weiterentwicklung an diesen Punkten nicht unmöglich. Konkret hatten wir noch folgende Ideen bereit:

Verbesserte Kamerasteuerung Die momentan implementierte Kamerasteuerung beherscht eigentlich nur die Rotation um einen festen Punkt und die Vergrösserung des Ausschnittes. Es wäre jedoch wünschenswert wenn man auch einem Agenten automatisch Folgen könnte, wie auch in einen Agenten *sitzen*.

Head Up Display Oder kurz HUD wäre Ideal geeignet um Daten der laufenden Applikation direkt anzuzeigen. Beispiele wären ein FPS-Zähler oder Daten über die Kamera in der Welt um die Orientierung zu vereinfachen. Zudem könnte man auch Steuerungelemente für die Simulation einbauen wie Pause oder Geschwindigkeit.

Parametriesierbares Modell Es hat sich im Projektverlauf gezeigt, dass es gut wäre, wenn man verschiedene Modelle speichern und laden könnte. Dies würde auch die Daten für die Szenerie von den eigentlichen Simulationsalgorithmen entkoppeln.

Verbesserung/Erweiterung der KI Die momentane Implementation der Agenten berücksichtigt nur die grundlegenden Algorithmen des Boids-Verhalten und die simplen Angriffsalgorithmen. Befehlshierachien, Primärziele und spezielles Angriffsverhalten der Capital Ships sind nur einige Beispiele für mögliche Erweiterungen.

Output für Renderprogramme Da unser Fokus von Anfang an auf autonomen Simulationen war, ist nicht nur die visuelle Präsentation möglich. Eine Idee ist, den Output des Modells in eine für Renderprogramme wie zum Beispiel RenderMan[11] lesbare Form zu schreiben.

5.3 Lerneffekt

Was sich als nützlich erwiesen hat, ist im Vorfeld des Projekts ein Meeting durchzuführen, um das Team zu bilden und sich auf ein Thema zu

einigen. So haben die Beteiligten bereits die Möglichkeit, sich bis zum eigentlichen Projektstart erste Gedanken zu machen.

Das stetige Nachführen der Dokumentation und der Protokolle hilft die Übersicht zu wahren und vermeidet, dass alle Schreibarbeiten am Schluss anfallen. Ebenfalls ist das abzuhandelnde Thema viel präsenter.

Das Programmieren künstlicher Intelligenzen hat sich als einfacher erwiesen als anfangs angenommen. Wir mussten jedoch feststellen, dass je komplizierter die KI wird, desto enger grenzt sich der Parameterbereich für eine vorhersagbares Verhalten ein. Dieses Phänomen zeigte sich sehr schön beim Flocking. Für einen sehr grossen Bereich der Parameter gab es befriedigende Resultate, doch nachdem weitere Verhaltensmuster zugefügt wurden, war die Wahl der Parameter entscheidend für das Verhalten.

Im Verlaufe des Projekts ergaben sich vermehrt Schwierigkeiten mit der Performance. Einerseits beansprucht die 3D- als auch die KI-Programmierung enorme Rechenleistung. Bis zu einem gewissen Grad konnte mit Hilfe von Approximationen rechenintensiven Operationen durch schnellere Algorithmen an Effizienz gewonnen werden. Um eine wirklich optimierte Lösung der KI zu erhalten, müsste in einem weiteren Schritt der ganze Code nochmals überarbeitet werden und Operationen durch Annäherungen beschrieben werden.

Erfreulicherweise hatten sich die anfangs befürchteten Probleme bezüglich Plattformunabhängigkeit nicht bestätigt. Einzig treten bei der 3D-Anzeige unter Mac OS X manchmal sichtbare Clipping-Artefakte auf, was durch den Beta-Status von Java3D 1.3.1 auf dieser Plattform erklärbar ist.

Anhang A

Benutzerhandbuch

1 Installation

Die folgenden Programme müssen installiert sein, um ABOVE starten zu können:

- J2SE 1.4 kompatible VM (Sun J2SE 1.4.2 empfohlen)
- Java3D Version 1.3.1, erhältlich von Sun's Java3D Website[1]

Um ABOVE zu installieren, muss lediglich der Inhalt des Zip-Files `above-xxx.zip` (wobei *xxx* für die Versionsnummer steht) in ein frei wählbares Verzeichnis entpackt werden. Dabei ist einzig zu beachten, dass die Datei `StarfireExt.jar` ins selbe Verzeichnis wie `above-xxx.jar` zu liegen kommt.

2 Start der Applikation

Auf den meisten Systemen genügt es, die Datei `above-xxx.jar` per Doppelklick zu starten. Falls dies nicht funktionieren sollte, kann das Programm gestartet werden, indem man auf der Konsole ins Verzeichnis wechselt, in welches die Dateien entpackt wurden, und folgenden Befehl eintippt: `java -jar above-xxx.jar`

Vessel Viewer

Die Hilfsapplikation VesselViewer, welche die zur Verfügung stehenden Schiffe anzeigt, kann wie folgt gestartet werden:
`java -cp above-xxx.jar ch.fha.ia02.above.VesselViewer`

[1] `http://java.sun.com/products/java-media/3D/`

Benchmark

Das ABOVE-interne Benchmark-Programm lässt das Modell mit nur minimalen 3D-Überbau laufen, um zu ermitteln, wie aufwändig die ständige Neuberechnung des Modells ist. Es kann wie folgt gestartet werden:

```
java -cp above-xxx.jar ch.fha.ia02.above.ModelBenchmark
```

Zusätzlich zu den ABOVE-typischen Statusmeldungen gibt das Programm eine Zeile mit den Messergebnissen aus. Hier als Beispiel das Ergebnis einer Messung von 10'000 Schritten mit einem Modell von 39 Agenten (3 Staffeln zu 12 Jägern plus 3 Star Destroyers):

```
39 agents, 10000 steps, 25047 ms (2.5047 ms/step)
```

3 Einstellungen

Das in Abbildung A.1 dargestellte Dialogfesnter zur Verwaltung der Einstellungen kann im Menü unter *Simulation | View Settings...* geöffnet werden, und bietet die Möglichkeit folgende Einstellungen zu verändern:

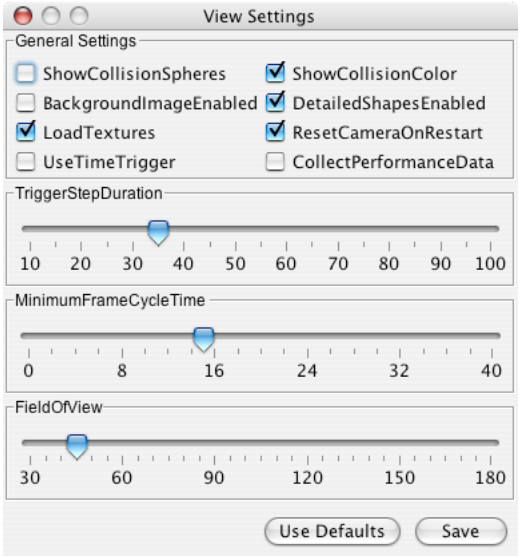

Abbildung A.1: Settings-Dialog

ShowCollisionSpheres Aktiviert die Anzeige der im Modell für das Vermeiden von Kollisionen verwendeten Sphären.

ShowCollisionColor Zeigt Kollisionen von Agenten an, indem ihre *CollisionSpheres* farblich markiert werden.

BackgroundImageEnabled Aktiviert die Anzeige eines Hintergrundbilds, kann je nach Plattform zu stockender Anzeige führen.

DetailedShapesEnabled Falls aktiv werden die Agenten durch komplexe 3D-Modelle dargestellt.

LoadTextures Aktiviert das Laden von Texturen für 3D-Modelle, kann die Ladezeiten deutlich erhöhen.

ResetCameraOnRestart Falls aktiv wird die Kameraposition beim Neustart der Simulation auf die Anfangseinstellung zurückgesetzt.

CollectPerformanceData Stellt ein, ob Performancedaten gesammelt werden oder nicht.

UseTimeTrigger Entkoppelt Berechnung und Darstellung des Modells: das Modell wird mit konstanten Zeitschritten der Länge *TriggerStepDuration* berechnet.

TriggerStepDuration Stellt die Zeitschrittlänge für die Berechnung des Modells in Millisekunden ein. Falls die eingestellte Zeit kürzer ist als die für die Berechnung eines Schritts benötigte Zeit, stimmt die Anzeige nicht mehr mit der Modellzeit überein und alle Bewegungen werden scheinbar langsamer.

MinimumFrameCycleDuration Stellt die minimale Zeit, in Millisekunden, zwischen der Berechnung zweier Bilder ein, limitiert also die pro Sekunde maximal angezeigten Bilder (FPS).

FieldOfView Stellt den Öffungswinkel der Kamera in Grad ein.

Zum Speichern der Einstellungen wird der *Preferences*-Mechanismus von Java verwendet, der versucht, die Einstellungen abhängig von der Plattform in kanonischer Art und Weise zu speichern. Unter Windows werden die ABOVE-Einstellungen in der Registry unter `HKEY_CURRENT_USER\Software\JavaSoft\Prefs\ch\fha\ia02\above` abgelegt.

4 Navigation in der 3D-Welt

Die Applikation bietet einige Steuerungmöglichkeiten für die Kamera bzw. ihr virtuelles Auge. Die Möglichkeiten sind mit dem Blick durch ein Fernglas zu vergleichen. Man kann sich um sich selbst drehen, hinauf und hinunter schauen und vergrössern bzw. verkleinern. Das letztere hat natürlich Einfluss auf den Sichtbereich. Was nicht möglich ist, ist die Bewegung des eigenen Standpunktes: Man kann zwar den Blickwinkel frei verändern mit dem Fernglas umherschwenken, jedoch nicht die eigene Position.

Folgend sind nun die Navigationsmöglichkeiten etwas genauer beschrieben:

Zoom Vergrössern und Verkleinern kann man auf zwei Arten: Entweder man benützt das Mausrad, wobei nach vorn vergrössert und nach hinten verkleinert wird. Oder man fährt mit der gedrückten mittleren Maustaste vorwärts und rückwärts.

Hoch- und Runtersehen Die Kamera wird nach oben bzw. unten gedreht, indem man mit der gedrückten linken Maustaste nach hinten bzw. nach vorn fährt.

Rotation um eigene Achse Die Rotationsachse ist immer die Vertikale. Dies bedeutet, sie ist unabhängig von Zoom und Rauf- bzw. Runterdrehen der Kamera. Nach links und rechts kann man sich drehen, indem man mit gedrückter linker Maustaste nach rechts oder links fährt.

Anhang B

Grössenordnungen und Masseinheiten im Star Wars-Universum

Auch wenn die von ABOVE simulierte Star Wars-Welt ein Werk der Fiktion ist, so orientieren wir uns trotzdem möglichst nahe an den veröffentlichten Grössenordnungen, und wenn es nur um das Erreichen einer möglichst *echt aussehenden* Simulation geht. Ausserdem erleichtert dieses Vorgehen den eventuellen Einsatz des Programmes für die Simulation von Luftkämpfen auf der Erde.

Allerdings wird dieses Vorhaben dadurch erschwert, dass die verfügbaren Quellen nicht nur unvollständig sind, sondern sich auch deutlich widersprechen. Im Folgenden stürtzen wir uns hauptsächlich auf die von C. Saxton in [12] zusammengetragenen Angaben.

Ein besonderes Problem ist, dass für die meisten Raumschiffe keine Maximalgeschwindigkeit bekannt ist, sondern nur die (vermutlich unkompensierte) maximale Beschleunigung in der Einheit MGLT angegeben wird, wobei 1 MGLT in etwa 400 m/s^2 entspricht. Allerdings wird MGLT dann als eine Art maximale Unterlichtgeschwindigkeit betrachtet, im speziellen verwenden einige Computerspiele MGLT als Geschwindigkeit, wobei dort 1 MGLT = 1 m/s gilt! Allerdings ist für einige Schiffe die maximale Geschwindigkeit beim Flug in standardisierter Atmosphäre bekannt, ebenfalls wird die relative Maximalgeschwindigkeit verschiedener Schiffe, wie auch Entfernungen und Reaktionszeiten im Kampf, in der Star Wars Literatur ausführlich diskutiert.

Aus diesen Angaben, validiert und ergänzt mit extrapolierten Performancedaten realer Flugzeuge und Schiffe der U.S. Navy [20], ermittelten wir die in Tabelle B.1 aufgeführten, für die Simulation verwendeten Daten.

Bezeichnung	Länge	Masse	v_{max}
Incom T-65c A2 Starfighter (*X-Wing*)	12.5 m	13 t	290 m/s
TIE-Fighter [weiss]	6.08 m	10 t	210 m/s
TIE-Interceptor	6.23 m	10 t	250 m/s
Imperator-Class Star Destroyer (*ISD-I*)	1'600 m	30'000 t	–
Sovereign-Class Command Ship	15'000 m	150'000 t	–
F/A-18 E/F *Super Hornet*	18.5 m	30 t	600 m/s
Nimitz-Class Flugzeugträger	333 m	88'000 t	15 m/s

Tabelle B.1: Die für die Simulation verwendeten Daten und Vergleichswerte der U.S. Navy. Bei den simulierten Jägern ist v_{max} die maximale Geschwindigkeit während dem Kampf, und deshalb ohne Bedeutung für die beiden Star Destroyer.

Anhang C

Glossar

CA [Abk. für "Certificate Authoroty"] Eine Organisation, die Zertifikate für die Verwendung mit Asymmetrischen Verschlüsslungsverfahren ausstellt, und dabei i.a. bestätigt, dass der Zertifikatsbesitzer wirklich die Person oder Organisation ist, die er behauptet.

FPS [Abk. für "Frames per Second"] Bezeichnet die durchschnittliche Anzahl pro Sekunde neu berechneter und angezeigter Bilder, insbesondere bei der Darstellung von 3D Objekten.

Java Web Start JNLP-Referenzimplementation von Sun Microsystems. Java Web Start wird seit J2SDK/JRE 1.2 mit den VMs des selben Herstellers gleich ausgeliefert und seit J2SDK/JRE 1.4 auch gleich bei der Installation der VM mit installiert. `http://java.sun.com/products/javawebstart/`

JNLP [Abk. für "Java Network Launching Protocol"] Ein Protokoll, um in Java geschriebene Applikationen ohne manuelle Installation direkt aus dem Internet starten zu können. JNLP beinhaltet unter anderem Mechanismen für die Versionierung, und eingeschränkten Zugriff auf das lokale System.

HUD [Abk. für "Head Up Display"] Dieser Begriff stammt eigentlich aus der Militäraviatik und bezeichnet dort einen halbdurchlässigen Spiegel im Blickfeld des Piloten, auf den wichtige Daten der Ziel- und Flugcomputer eingeblendet werden. Im Kontext von Computerspielen bezeichnet HUD die Gesamtheit aller ins Hauptfenster eingeblendeten Informationen für den Spieler, wie z.B. Karten oder die Anzeiger der Lebenspunkte.

KI [Abk. für "Künstliche Intelligenz"] Im Kontext dieses Projektes definieren wir Intelligenz über das Verhalten der simulierten Piloten:

Beim Betrachter soll der Eindruck entstehen, der Pilot verhalte sich intelligent, und folge nicht nur einem vordefinierten Pfad oder einfachen Regeln.

Scene Graph Der *Scene Graph* besteht aus einem Überbau aus einem *Virtual Universe* mit einer beliebigen Anzahl von `BranchGroup` bzw. untergeordneten Graphen.

Virtual Universe Die Wurzel eines oder mehrerer *Scene Graphs*, enthält Informationen über Java3D (wie zum Beispiel: Version, ob OpenGL oder DirectX eingesetzt wird.)

Literaturverzeichnis

[1] Brooks, R. A.: *A Robust Layered Control System for a Mobile Robot*, IEEE Journal of Robotics and Automation, Vol. 2, No. 1, March 1986, pp. 14-23; also MIT AI Memo 864, September 1985.

[2] Gamma E. et al.: *Design Patterns: elements of reusable object-oriented software*, 1995 Addison-Wesley.

[3] Gerstendörfer T.: Java Code Conventions, `http://wanda.fh-aargau.ch/i/ia02gers/codeconv.html` [20. März2004]

[4] Kellenberger, L. et al.: *ABOVE Projekt Website* `http://wanda.fh-aargau.ch/i/ia02gers/above/`

[5] Kopka, H.: *LaTeX*, Band 1: Einführung, 3., überarbeitete Auflage, 2000 Addison-Wesley.

[6] Klingener, F.: *Java 3D Simulation Clocks*, 27. Oktober 2001, `http://www.brockeng.com/VMech/Time/PlanJ/Clocks.htm` [17. März 2004]

[7] *Lord of the Rings: The Two Towers*, Regie: Jackson, P., Buch: Tolkien, J.R.R., Walsh, F., Boyens, P., Sinclair, S., Jackson, P., 179 min, NZ/USA 2002.

[8] Reynolds, C. W.: *Flocks, Herds, and Schools: A Distributed Behavioral Model, in Computer Graphics*, 21(4) (SIGGRAPH '87 Conference Proceedings) pages 25-34. `http://www.red3d.com/cwr/papers/1987/boids.html`

[9] Miyamoto, J.: *Java3D Tutorials, Lesson 7: Transparency* `http://www.vrupl.evl.uic.edu/VRLabAcc/about_vrlab/java3d/lesson07/` [22. Mai 2004]

[10] Reynolds, C. W.: *Steering Behaviors For Autonomous Characters*, in the proceedings of Game Developers Conference 1999 held in

San Jose, California. Miller Freeman Game Group, San Francisco, California. Pages 763-782. http://www.red3d.com/cwr/papers/1999/gdc99steer.html

[11] Pixar: *The RenderMan Interface*, Version 3.2, Juli 2000

[12] Saxton, C.: *STAR WARS Technical Commentaries* http://www.theforce.net/swtc/ [1. April 2004]

[13] Sedgewick, R.: *Algorithmen in C*, 1992 Addison Wesley.

[14] *Star Wars: Episode V – The Empire Strikes Back*, Regie: Kershner, I., Buch: Lucas G., Brackett, L., Kasdan, L., 124 min, USA: Lucasfilm Ltd. 1980.

[15] Torres, E.: *Optimization of the Binary Space Partition Algorithm (BSP) for the Visualization of Dynamic Scenes*, Sept. 1990, Eurographics '90

[16] *Star Wars: Episode VI – Return of the Jedi*, Regie: Marquand, R., Buch: Lucas G., Kasdan, L., 134 min, USA: Lucasfilm Ltd. 1983.

[17] Sun Microsystems: *Code Conventions for the Java Programming Language*, http://java.sun.com/docs/codeconv/index.html [20. März 2004]

[18] Sun Microsystems: *How to Write Doc Comments for the Javadoc Tool*, http://java.sun.com/j2se/javadoc/writingdoccomments/ [20. März 2004]

[19] Sun Microsystems: *Java Network Launsching Protocol & API Specification*, JSR-56, Version 1.0.1, 21. Mai 2001 http://java.sun.com/products/javawebstart/download-spec.html

[20] The United States Navy: *Fact File*, http://www.chinfo.navy.mil/navpalib/factfile/ffiletop.html [2. April 2004]

[21] University of applied sciences, Biel School of Engineering and Architecture: *Project Demo*, http://www.hta-bi.bfh.ch/~swc/DemoJ3D/Demo/index.html [10. Juni 2004]

Zusammenfassung

Der Bericht beginnt mit einer Beschreibung der Problemstellung, ehe im zweiten Kapitel die Ziele wie auch der Anforderungskatalog an die Software definiert werden. Im darauffolgenden Kapitel wird der von uns eingesetzte Entwicklungsprozess kurz umrissen, ausserdem werden dort die geleisteten Aufwände aufgeführt. Kapitel 4 beschreibt die eigentliche Implementation beginnend bei der Wahl von Sprache und Plattform, übergehend in eine kurze Beschreibung der gewählten Architektur ehe die dreidimensionale Anzeige und das zugrundeliegende Simulationsmodell beschrieben werden. Eine kurze Betrachtung der Performance von Vektor-Operationen und möglicher Schwierigkeiten bei der Distribution der Software runden dieses Kapitel ab. Schliesslich endet dieser Bericht mit einer Diskussion des Entwicklungsstands, einer möglichen zukünftigen Weiterentwicklung wie auch des aufgetretenen Lerneffekts.