Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Um ein Daumenkino zu erstellen, muss man eine Reihe von Bildern, deren Inhalt sich nur geringfügig unterscheidet, anfertigen. Beim schnellen Durchblättern schließlich werden diese Bilder vom menschlichen Auge als eine flüssige Bewegung wahrgenommen. Diese Eigenschaft macht sich unter anderem auch der Trickfilm zunutze: Für eine fließende Abfolge einer Animation werden meist 15 bis 25 Bilder pro Sekunde (engl. *frames per second*) abgespielt, die jeweils nur geringe Bewegungsänderungen enthalten. Dabei wurden die Schlüsselbilder, sogenannte *key frames* von einem kleinen Teil der Zeichner angefertigt, während die Übergangsbilder dazwischen von vielen deutlich schlechter bezahlten Mitarbeitern gezeichnet wurden.
Die Technik der *Keyframe-Animation* nimmt einem heutzutage diesen aufwändigen Prozess bei der Anfertigung flüssiger Bewegungsabläufe am Computer ab, indem lediglich Start- und Endpunkt der Animation entlang einer Zeitachse definiert werden müssen. Diese Schlüsselszenen bilden damit nach wie vor die Wendepunkte der Animation, zwischen deren Einzelbildern interpoliert wird, ohne jedoch jedes Bild von Hand anfertigen zu müssen. Dafür muss der Computer numerische Zwischenwerte berechnen können, um insbesondere zwischen Translationen, Rotationen, Positionsangaben, Farben und Farbintensität zu interpolieren. Diesen Prozess der Interpolation nennt man auch *Inbetweening*.

Im Rahmen dieser Studienarbeit soll eine Software entwickelt werden, die dem Benutzer eine interaktive Benutzeroberfläche zur Kontrolle von Pfaden für die Animation verschiedener Objekte zur Verfügung stellt. Diese Keyframe-Technik wird heute schon von verschiedenen Programmen bereitgestellt, in denen üblicherweise Geschwindigkeit und Art der Bewegung zwischen den Keyframes entlang einer Zeitachse definiert werden kann. Im folgenden Kapitel sollen daher zur Anforderungserhebung zunächst drei dieser 3D-Editoren untersucht werden.

2 Anforderungserhebung des zu entwickelnden Systems

2.1 Untersuchung bestehender Systeme

Um eine genauere Vorstellung der zu implementierenden Keyframe-Animation und der hierfür nötigen Benutzerinteraktion zu bekommen, wurden zu Beginn der Arbeit kleinere Animationen mit drei verschiedenen Programmen erstellt. Hierbei handelte es sich um die Software *Cinema 4D* der MAXON Computer GmbH, *Maya* von Alias Systems und den *3d-Modeleditor* von Torsten Richter. Diese unterschiedlich mächtigen Programme boten mit ihrer Vielzahl von Funktionen zur Erstellung komplexer Animationen zunächst einen Einblick in die Funktionsweise der Keyframeanimation an sich, aber auch in die Möglichkeiten der Benutzerinteraktion sowie in die mögliche Gestaltung der Benutzerschnittstelle.

2.1.1 Cinema 4D

Cinema 4D erlaubt die zeitgleiche Animation verschiedener Objekte einer Szene, deren Eigenschaften in einem Zeitfenster festgelegt werden. Die Animation der Objekte kann dabei in horizontalen Spuren festgelegt werden, welche mehrere Sequenzen enthalten können, die die Animation zeitlich begrenzen können. Mithilfe von Keyframes kann der Benutzer schließlich diesen Sequenzen ein Weg-Zeit-Verhalten zuweisen.

Für eine einfache Animation muss lediglich ein Objekt in dem perspektivischen Editorfenster erzeugt und auf die gewünschte Startposition gebracht werden. Ebenso muß auch in der Zeitleiste die Anfangszeit der Animation festgelegt werden. Damit besteht in *Cinema 4D* stets eine Verbindung zwischen dem zu definierendem Pfad und der Weg-Zeit-Relation. Drückt man auf den Aufnahmebutton, so wird der Keyframemodus aktiviert. Der Benutzer kann den Endpunkt in der Zeitleiste festlegen, sowie das Objekt zu dessen Endposition verschieben. Der Keyframepfad wird in dem Editorfenster dabei automatisch angezeigt. Die Aktion kann nun durch ein erneutes Klicken des Aufnahmebuttons beendet werden. Um die erstellte Animation abzuspielen,

verfügt *Cinema 4D* über einen in die Software integrierten Player, der eine sofortige Anschauungsmöglichkeit bietet. Das Programm erlaubt mit diesen Funktionen die intuitive Erstellung einer Raumkurve mit gleichzeitiger Festlegung, wann sich das Objekt an welcher Position dieser Kurve befinden soll.

2.1.2 Maya

Das von Alias Systems entwickelte Animationsprogramm *Maya* unterscheidet sich von dem im vorherigen Abschnitt vorgestellten Programm durch eine umfangreichere Funktionsauswahl. Dagegen ist die Vorgehensweise bei der Erstellung eines Pfades mit Hilfe von Keyframes sowie die Aufteilung in der Benutzerschnittstelle ähnlich derer von *Cinema 4D*.
Um einen Keyframe an einer bestimmten Position hinzuzufügen, kann ebenfalls der direkte Weg über die Mausinteraktion in dem Animationsfenster gewählt werden, oder aber die x, y und z-Koordinaten werden über dafür vorgesehene Eingabeboxen hinzugefügt. Dabei liegt der Default-Wert immer bei $(0, 0, 0)$. Der Verlauf der Raumkurve kann durch die Änderung der Keyframekoordinaten jederzeit geändert werden; zusätzlich kann bei der Interpolation durch Hermite-Kurven auch die Tangente manipuliert werden und damit die Kurvenexpansion in beliebige Richtung verlaufen.

2.1.3 3d-Modeleditor

Von den drei untersuchten Programmen ist der *3d-Modeleditor* als einziges eine Shareware und unterscheidet sich von den anderen durch einen weitaus geringeren Funktionsumfang. Mit der Auswahlmöglichkeit zwischen der Animation durch Keyframes oder inverser Kinematik stellt es dennoch einen breiteren Funktionsumfang zur Verfügung als die Anforderungen des zu entwickelnden Animationseditors und bietet damit einen guten Überblick über Aussehen und Funktionsweise der zu implementierenden Funktionalitäten in einem kleineren Rahmen. Durch die relativ geringen Auswahlmöglichkeiten der Interaktion ist die Benutzeroberfläche sehr übersichtlich und erlaubt den schnellen Zugriff auf alle Aktionen, die zur Erstellung einer Animation notwendig sind. Ist ein Pfad definiert worden, kann die Kurve lediglich durch Änderung der Position eines Keyframes verändert werden, nicht jedoch durch eine Manipulation der Tangente, wie es in beiden oben beschriebenen Programmen möglich ist. Ebenso kann ein Objekt nur in konstanter Geschwindigkeit entlang des Pfades bewegt werden; ein unterschiedliches Geschwindigkeitsverhalten mehrerer Objekte kann demnach nur durch die Abstände zwischen den Keyframes auf der Raumkurve erreicht werden. Das Beschleunigen oder Abbremsen zwischen zwei Keyframes ist dabei nicht möglich.

2.2 Anforderungsdefinition des zu entwickelnden Systems

Aufbauend auf die vorherige Analyse der drei bestehenden Animationsprogramme wurde eine Anforderungsdefinition erstellt, die notwendige Funktionalitäten und Datenstrukturen umfasst. Die zu entwickelnde Software soll demnach eine interaktive Benutzersteuerung ermöglichen, die das Hinzufügen beliebig vieler Objekte erlaubt. Jedes Objekt soll weiterhin mit einer Liste von Keyframes verknüpft sein, die den jeweiligen Animationspfad definieren. Die einzelnen Keyframes dieses Pfades sollen zusätzlich durch spätere Manipulation ihrer Position eine Änderung der Animation erlauben, wobei ebenfalls für jeden Keyframe eine Änderung des Geschwindigkeitsverhaltens zur Anpassung der Position verschiedener Objekte zu einem gleichen Zeitpunkt t möglich sein muss. Änderungen, die das einzelne Objekt betreffen, sind dagegen von der Implementierung ausgeschlossen; das Objekt ist also in seiner Form und Farbe statisch. Hauptziel der Arbeit ist demnach die Erstellung einer Software, die die gleichzeitige Animation verschiedener Objekte entlang selbst definierbarer Pfade vornimmt. Dem Benutzer sollen dabei über ein *Graphical User Interface* (GUI) die verschiedenen Interaktionsmöglichkeiten zur Verfügung gestellt werden, wobei neben der eigentlichen Animationsfläche ein Bereich für die Navigation in der Objekt- und Keyframeliste mit den Änderungsmöglichkeiten dieser Keyframes, sowie ein weiterer für die Menüleiste und die Toolbar mit Shortcut-Buttons und Player reserviert werden muss. Das GUI des entwickelten Animationseditors besteht daher aus drei Modulen, deren Funktionen in den folgenden Abschnitten genauer erläutert werden.

2.2.1 Das Animationsfenster

In dem Animationsfenster können Objekte hinzugefügt und durch Translation oder Rotation bewegt werden. Um sich diese Bewegungen ’merken’ zu können, muss der Benutzer über die Keyframe-Toolbar einen neuen Key erzeugen und mit Hilfe mehrerer Keyframes schließlich einen Pfad definieren, entlang dessen sich das jeweilige Objekt bewegen soll. Für jeden Keyframe können Position und Orientierung während der gesamten Zeit verändert werden, wodurch automatisch die angesprochene Schlüsselposition des Bewegungsablaufs in dem Animationsfenster verschoben wird.

Die Reihenfolge der Keyframes kann zu keinem Zeitpunkt verändert werden; vielmehr wird ein neuer Keyframe immer in einer Liste hinter dem aktuellen Keyframe angefügt. Die Position des Keyframes zu einem bestimmten Zeitpunkt entspricht dabei stets den aktuellen Werten des Objekts. Sind mindestens zwei Keys gesetzt worden, so werden diese in dem Animationsfenster durch eine Interpolationskurve verbunden. Bei n Keyframes werden also immer $n - 1$ verbin-

dende Kurven angezeigt, die den Pfad der Bewegung beschreiben.

Hat der Benutzer die Voraussetzung erfüllt und mindestens zwei Keys gesetzt, so kann die Animation abgespielt werden. Die Ausgabe erfolgt dabei im gleichen Fenster, wobei sich der Zeitablauf an den gespeicherten Werten im Keyframefenster (siehe 2.2.2) orientiert.

Abbildung 2.1 zeigt das Animationsfenster mit mehreren Objekten und jeweils einer durch Keyframes definierten Kurve, die den Objektpfad bestimmt.

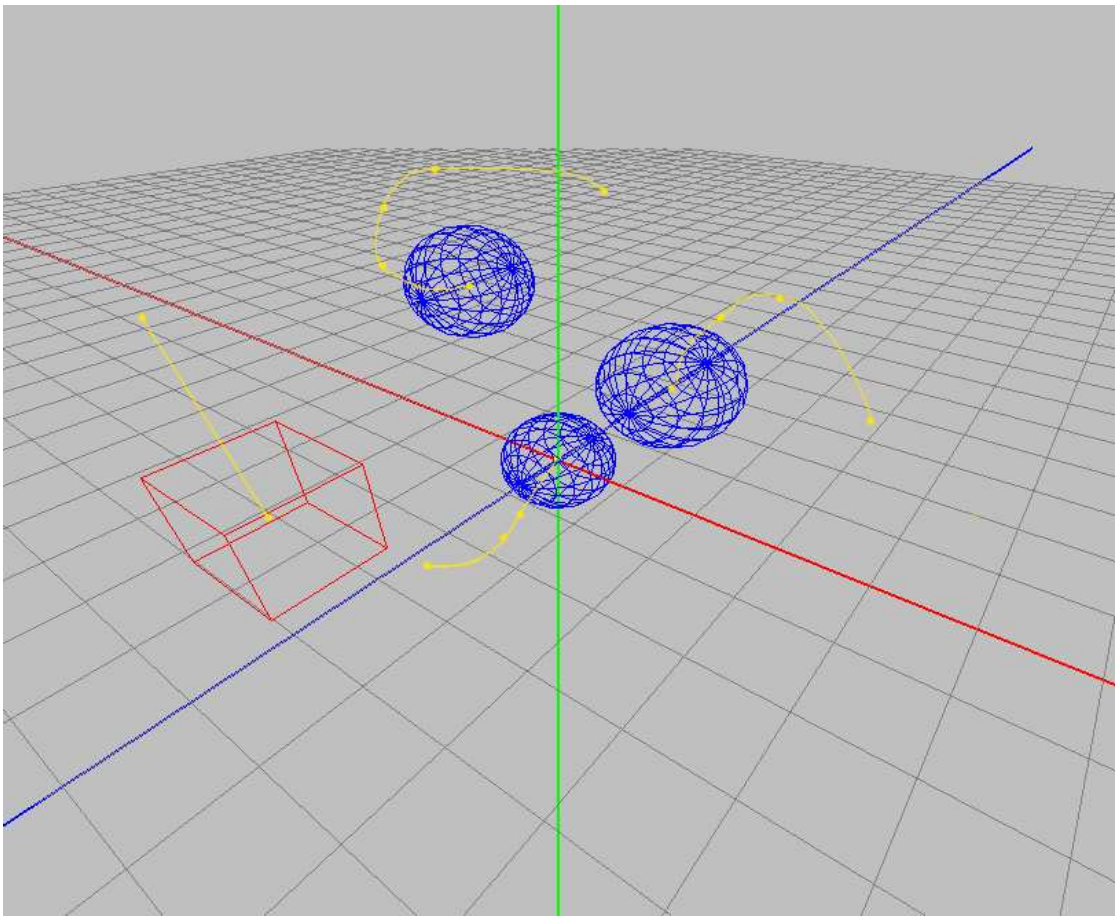

Abbildung 2.1: Objektpfade im Animationsfenster

Um dem Benutzer die Möglichkeit der Rundum-Ansicht zu geben, kann das Animationsfenster weiterhin mit der Maus oder den Cursortasten der Tastatur bewegt werden. Diese Bewegungen basieren auf eine Veränderung der Kameraposition in Bezug auf das Weltkoordinatensystem, welches in Abbildung 2.2.1 dargestellt ist. Eine Links-/Rechtsbewegung mit der Maus oder

den Cursortasten bedeutet gleichermaßen die Bewegung der Kamera auf der x-Achse. Dagegen bedeutet eine Bewegung nach oben ein ’Zoom-In’, bei der die Kamera näher an das Objekt rückt und die Szenerie größer erscheint. Umgekehrt resultiert eine Bewegung nach unten in ein ’Zoom-Out’, so dass der Benutzer einen größeren Blickwinkel auf die (nun kleineren) Objekte hat.

Die Tastatur bietet zudem weitere Interaktionsmöglichkeiten, die eine Immersion in die 3D-Welt erlauben. So dienen die Tasten 1-4 zur Veränderung der Kamerablickrichtung, welche im voreingestellten Zustand immer eine Linie zu dem Koordinatenursprung bildet. Hierdurch erhält der Benutzer die Möglichkeit, ein beliebiges Objekt im Raum zu fokussieren. Die Tasten 5-0 dienen weiterhin zur Veränderung des Kamerakoordinatensystems, welches die Kamera in verschiedene Richtungen ’kippen’ lassen kann.

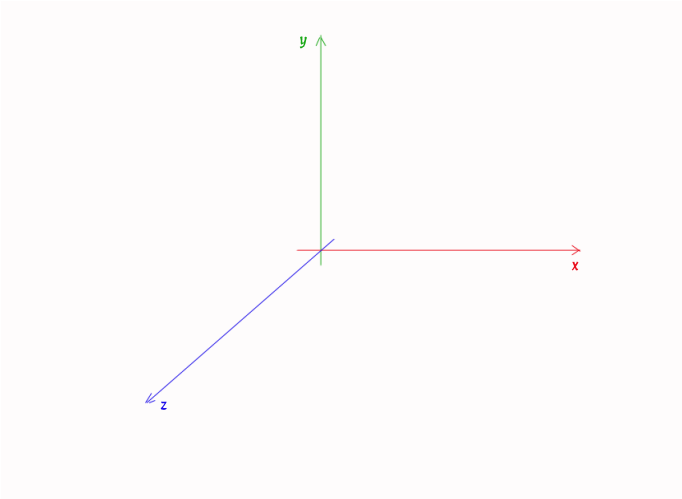

Abbildung 2.2: Das Weltkoordinatensystem in dem Animationsfenster

2.2.2 Das Keyframefenster

In dem Keyframefenster wird eine Liste der vom Benutzer hinzugefügten Objekte erstellt. Es kann jeweils nur das aktive Objekt mit seinen Werten angezeigt werden. Um zwischen den Objekten zu wechseln, muss der Benutzer erst ein anderes Objekt in der Liste auswählen, damit die zugehörigen Attributwerte erscheinen. Diese Werte können durch folgende Eingabemöglichkeiten manipuliert werden:

1. Ein beliebiger Keyframe kann aus der gespeicherten Keyframeliste des aktiven Objekts ausgewählt werden

2. Die ’aktive’ Keyframeposition kann durch Eingabe neuer x-, y- und z-Koordinaten verändert werden

3. Jedem Keyframe wird zusätzlich eine Geschwindigkeit zugeordnet, die das Objekt an der eingegebenen Position haben soll

4. Die Länge der Tangente kann angepasst werden und bietet damit eine hohe Einflussmöglichkeit auf den Kurvenverlauf

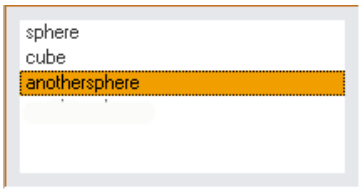

Abbildung 2.3: Die Objektliste

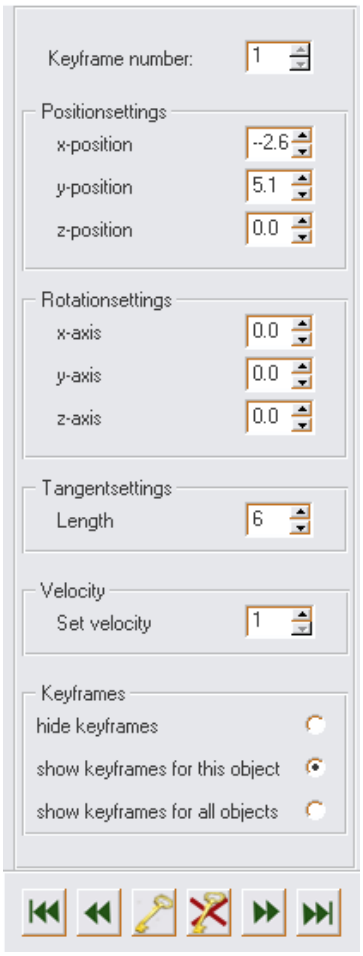

Abbildung 2.4: Das Keyframefenster

Die Werte sind zunächst durch die voreingestellte Keyframeposition (0.0, 0.0, 0.0) gespeichert. Durch das Keyframefenster bietet sich dem Benutzer nun eine Möglichkeit, die Attribute jedes einzelnen Keyframes zu ändern, bzw. einen unerwünschten Keyframe nachträglich zu löschen. Die Festlegung der Geschwindigkeit wirkt sich zusätzlich auf die Animation des Objektes aus:

Wird ein Keyframe n mit einer anderen Geschwindigkeit als der Nachfolger $n + 1$ gesetzt, so muss der Computer zwischen den beiden Geschwindigkeitswerten interpolieren. Die Implementierung dieser Funktion mit ihren mathematischen Voraussetzungen wird in Abschnitt 3.2 erläutert.

2.2.3 Die Menüleiste und die Toolbar

Im oberen Bereich des GUI hat der Benutzer die Möglichkeit, über verschiedene Shortcut-Buttons Objekte hinzuzufügen, sowie eine der Manipulationsmöglichkeiten bezüglich der Position und Orientierung einzelner Objekte auszuwählen. Gleiches kann auch über die herkömmliche Menüleiste erfolgen; mit der Toolbar bietet sich dem Benutzer lediglich eine Alternative, häufig vorkommende Aktionen schneller auszuführen.

Abbildung 2.5: Die Menüleiste und Toolbar in *The Animator*

Ein weiterer Bestandteil der Toolbar ist der Player, der über verschiedene Funktionen - play, play backward, stop and pause - verfügt. Der Button ’Play’ bewirkt die Ausführung der Animation über den Pfad aller gespeicherten Keyframes für jedes Objekt in der Liste. Grundsätzlich erfolgt der Verlauf dabei von den jeweils ersten Keyframes in der Liste jedes Objektes an; bewirkt der Benutzer jedoch über den Button ’Pause’ ein Anhalten der Animation, so wird der auf dem Animationspfad aktuelle Keyframe, bzw. falls das Objekt sich zwischen zwei Keyframes befindet, der nachfolgende, gespeichert. Ein weiterer Klick auf den Button ’Play’ führt anschließend zu einer erneuten Ausführung der Animation, die nun ab den zu den für jedes Objekt gespeicherten Keyframes gehörenden Positionen für jedes Objekt fortgesetzt wird. Der Stop-Button bewirkt hingegen den Abbruch der Animation ohne Speicherung der schon erreichten Position auf dem Pfad. Bei erneuter Ausführung des Play-Buttons beginnt die Bewegung dementsprechend wieder ab dem ersten Keyframe.

2.3 Umsetzung der Anforderungsdefinition in geeignete Datenstrukturen

Der Animationseditor erfordert zunächst die Implementierung einer Liste, die dem Benutzer erlaubt, Objekte vom Typ Cube, Cone oder Sphere hinzuzufügen. Gleichzeitig ermöglicht diese Datenstruktur die interne Navigation zwischen den Objekten, so dass dem aktiven Objekt die

jeweils dazugehörige Keyframeliste zugewiesen werden kann. Die Objektliste besteht demnach aus mehreren vom Benutzer spezifizierten Objekten, die mit ihrem Typ sowie ihrem Namen übergeben werden. Die Klasse *Objekt* vererbt dabei alle Attribute an die Unterklassen *Sphere*, *Cone* und *Cube*. Dies ermöglicht eine interne Identifizierung und damit die Zuordnung der Objekte innerhalb des Quellcodes.
Jedes Objekt muss weiterhin auf eine Liste von Keyframes zeigen, die die Festlegung des Animationspfades erlauben. Die Klasse *Objekt* ist demnach eine Liste, die aus Elementen der Klasse *Keyframe* besteht. Letztere besitzt schließlich verschiedene Attribute, die die Speicherung der vom Benutzer im Keyframefenster festgelegten Daten ermöglicht. Die Verkettung der Datenstrukturen mit ihren Parametern sind innerhalb des UML-Diagramms in Abbildung 2.3 dargestellt.

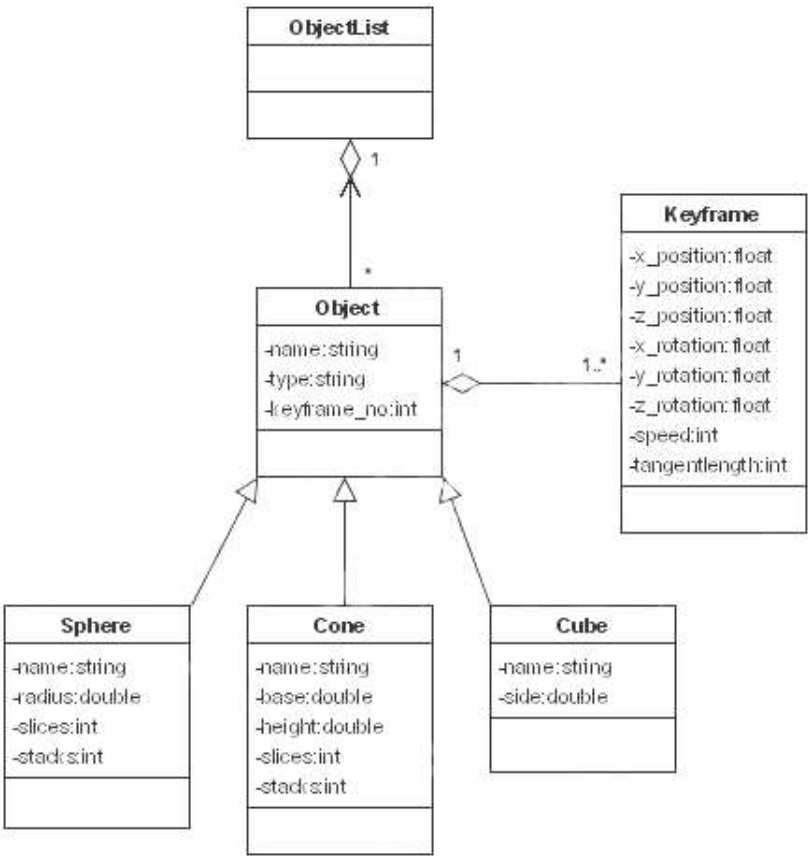

Abbildung 2.6: UML-Modellierung der Zusammenhänge von Objekt- und Keyframelisten

3 Mathematische Grundlagen

Für die Umsetzung der Interpolation zwischen den Keyframes und den einzelnen Rotationswinkeln sowie für die Darstellung des Geschwindigkeitsverhaltens müssen die nötigen mathematischen Formeln implementiert werden, deren Grundlagen an dieser Stelle erklärt werden sollen.

3.1 Interpolation der Schlüsselpositionen

Die erstellten Keyframes bilden die Basis für den Kurvenverlauf, der den Animationspfad begründet. Für diesen Pfad gilt es bei der Implementierung jedoch zwischen drei Fällen zu unterscheiden, die jeweils einen anderen mathematischen Ansatz zur Interpolation verfolgen:

- Hat der Benutzer nur einen Keyframe definiert, so kann folglich keine Interpolation ausgeführt werden. Voraussetzung für eine Animation sind also mindestens zwei Keyframes unterschiedlicher Position.
- Hat der Benutzer zwei Keyframes definiert, so wird das Inbetweening durch eine *lineare Interpolation* (3.1.1) vorgenommen.
- Bei mehr als zwei vorhandenen Keyframes wird mittels *Hermite* (3.1.3) interpoliert, wobei für das erste sowie das letzte Segment der Kurve jeweils der Sonderfall in der Berechnung der Tangente zu beachten ist.

3.1.1 Lineare Interpolation

Wurden für ein Objekt nur zwei Keyframes definiert, wird das Inbetweening durch eine lineare Interpolation vorgenommen. Das Objekt bewegt sich in der Animation demnach auf einer Geraden zwischen den beiden Schlüsselpositionen. Sind also zwei Punkte P_0 und P_1 bekannt, dann können alle Zwischenbilder für einen Parameter u in Relation zu der Zeit t durch

$$C(u) = (1 - t) * P_0 + t * P_2 \tag{3.1}$$

berechnet werden [3].

3.1.2 Interpolation durch Kurven

Der durch mehr als zwei Keyframes definierte Pfad ist im mathematischen Sinne eine Kurve im dreidimensionalen Raum. Grundlage für die Implementierung solcher Kurven ist die Auseinandersetzung mit parametrischen Gleichungen, die die Kurve formen: Eine Kurve im dreidimensionalen Raum kann durch ein Set aus Koordinaten x, y und z beschrieben werden. Jede dieser Koordinaten wird durch eine eigene parametrische Gleichung beschrieben, deren generelle Form durch

$$x = x(u), y = y(u) \text{ und } z = z(u)$$

ausgedrückt werden kann. u ist dabei eine unabhängige Variable aus dem Intervall $[a, b]$ und $x(u) = a^u + bu + c$ ist die parametrische Form für die Koordinate x, die in gleicher Weise für y und z dargestellt werden kann. Die Werte der x-, y- und z-Koordinaten hängen von dem parametrischen Wert u ab, sind jedoch untereinander unabhängig.

Die darzustellende Kurve basiert auf Punkten, die jeweils durch einen Vektor $\vec{p}$ de£niert sind. $\vec{p}$ besteht dabei aus den Bestandteilen $x(u)$, $y(u)$ und $z(u)$, so dass der Vektor $\vec{p}$ eine Funktion des parametrischen Wertes u ist: $\vec{p} = p(u)$. Die Funktionen, die die Vektorbestandteile von $\vec{p}$ ausmachen, bestimmen gleichzeitig die Form der Kurve. Um nun eine Kurve im Raum zu de£nieren, müssen kubische Polynome als parametrische Funktionen verwendet werden. Für $x(u)$ gilt dann

$$x(u) = a_x u^3 + b_x u^2 + c_x u + d_x \tag{3.2}$$

Für $y(u)$ und $z(u)$ lässt sich die Gleichung äquivalent darstellen. Die drei Gleichungen für $x(u)$,$y(u)$ und $z(u)$können nun in einer einzigen Vektorgleichung zusammengefasst werden:

$$Q(u) = au^3 + bu^2 + cu + d \tag{3.3}$$

Zur Darstellung einer Raumkurve können einerseits Hilfspunkte zwischen den zu interpolierenden Punkten gefunden werden, so dass eine Interpolation der Kurve durch mehr als zwei Punkte kontrolliert wird. Ein anderer Ansatz ist jedoch, die Tangente der Kurve an Start- und Endpunkt zu berechnen, wie es für die Konstruktion von Hermite-Splines gebraucht wird.

3.1.3 Interpolation durch Hermite-Splines

Die Anforderungen an die Kurve zur Erzeugung eines Animationspfades sind durch eine Interpolation der Punkte sowie durch eine einfache Berechnung des Kurvenverlaufs mit lokaler

Kontrolle, also eine alleinige Berechnung der beiden an den veränderten Punkt angrenzenden Kurvensegmente, gegeben. Die Interpolation zwischen n Keyframes bedeutet im ganzheitlichen Fall die Konstruktion eines Polynoms $(n-1)$-ten Grades. Schon bei einer geringen Anzahl von Keyframes würde die Komplexität der Berechnung daher enorm ansteigen, weshalb für diese Arbeit eine Kurve aus zusammengesetzten Kurvensegmenten zwischen jeweils zwei Keyframes verwendet wurde. Diese sogenannte *Spline* sollte zudem durch Berechnung einer Start- und Endtangente für jeden Punkt kontrolliert werden können, wobei die beiden Tangenten die gleiche Orientierung haben sollten. Die Endtangente eines Segments ist also gleich der Starttangente des nächsten. Damit ist gewährleistet, dass die Kurve an allen Punkten einen fliessenden Übergang hat.

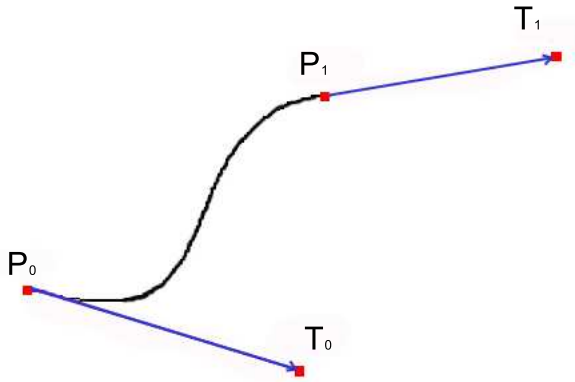

Abbildung 3.1: Interpolation zweier Punkte P_0 und P_1 mittels Hermite-Splines

Die hier aufgeführten Anforderungen lassen sich mit der Hermite-Interpolation erfüllen, bei der jeweils zwei Punkte durch ein kubisches Polynom (siehe 3.3) verbunden werden. Neben den zwei Keyframes, die den Start- und Endpunkt eines Segments bilden, müssen auch die Start- und Endtangente bekannt sein, bevor die Kurve generiert werden kann. Diese können aus den jeweils ersten Ableitungen der parametrischen Funktionen xu, yu und zu berechnet werden:

$$Q'(u) = 3au^2 + 2bu + c. \tag{3.4}$$

Die vier Unbekannten a, b, c und d können durch die Angabe zweier Kurvenbegrenzungspunkte P_0 und P_1 sowie der beiden an diesen Punkten anliegenden Tangenten P'_0 und P'_1 berechnet werden. Wenn $u = 0$ ist, ist $Q(0) = P_0$ und $Q'(0) = P'_0$. Ebenso ist für $u = 1$ $Q(1) = P_1$ und $Q'(1) = P'_1$. Dann erhält man

$$Q(0) = d = P_0$$
$$Q(1) = a + b + c + d = P_1$$
$$Q'(0) = c = P'_0$$
$$Q'(1) = 3a + 2b + c = P'_1$$

und daraus

$$a = 2(P_0 - P_1) + P'_0 + P_1 \; b = 3(P_1 - P_0) - 2P'_0 - P'_1.$$

Die nun bekannten Werte a, b, c und d können jetzt in die Ausgangsgleichung 3.3 eingesetzt werden:

$$Q(u) = [2(P_0 - P_1) + P'_0 + P'_1]u^3 + [3(P_1 - P_0 - 2P'_0 - P'_1]u^2 + P'_0 u + P_0 \tag{3.5}$$

Hieraus erhält man nach Umformen die finale Gleichung der Hermite-Kurve:

$$Q(u) = (2u^3 - 3u^2 + 1)P_0 + (-2u^3 + 3u^2)P_1 + (u^3 - 2u^2 + u)P'_0 + (u^3 - u^2)P'_1. \tag{3.6}$$

Jedes Segment ist damit eine seperate Funktion mit einem Gültigkeitsbereich über dem Intervall $[0, 1]$. Damit der kontinuierliche Verlauf der Kurve gegeben ist, müssen sich nicht nur zwei benachbarte Segmente Q_1 und Q_{i+1} in Positionen überschneiden, so dass $Q_i(1) = Q_{i+1}(0)$, sondern, wie oben beschrieben, müssen auch die Start- und Endtangente mit $Q'_i(1) = Q'_{i+1}(0)$ übereinstimmen.

Die Tangentenvektoren werden in dieser Arbeit zusätzlich zur Bewahrung des kontinuierlichen Verlaufs der Kurve auch zur Manipulation des Kurvenverlaufs benutzt. Die Interaktionsmöglichkeit durch den Benutzer lässt dafür eine Manipulation der Tangentenlänge zu, wodurch auf die Kurve Einfluss genommen werden kann. Eine weitere Einflussmöglichkeit würde die Richtungsbestimmung der Tangentenorientierung durch den Benutzer bieten; im Rahmen dieser Arbeit werden der Einfachheit halber jedoch automatische Hermite-Kurven generiert, bei der für jeden Punkt die Tangenten automatisch festgelegt werden. Eine Möglichkeit hierfür bietet sich durch das Generieren von Tangenten, die an einem innenliegenden Keyframe kontinuierlich sind, in dem Sinne, dass auch die zweite Ableitung der Funktion über einem Intervall $[a, b]$ kontinuierlich ist. Um dies zu erreichen, ist die Berechnung eines linearen Gleichungssystems nötig, in dem die Keyframes als bekannte Einheiten und die Tangenten als Unbekannte angenommen werden. Zunächst werden die erste und zweite Ableitung der Hermite-Kurve Q berechnet:

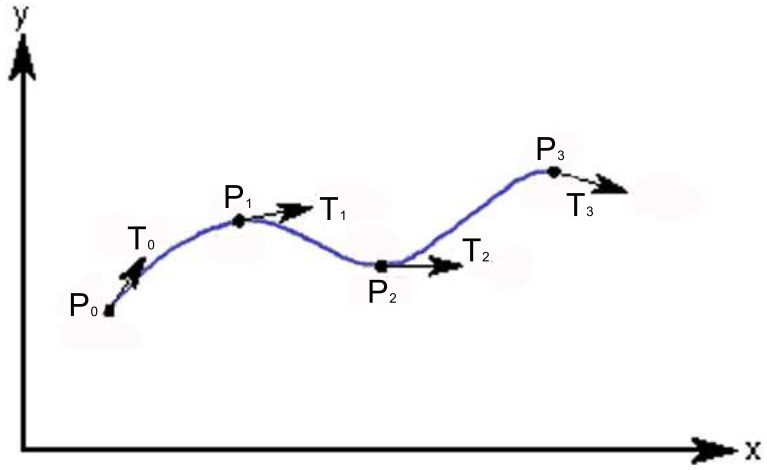

Abbildung 3.2: Kontinuierlicher Kurvenverlauf

$$Q_i'(u) = (6u^2 - 6u)P_i + (-6u^2 + 6u)P_{i+1} + (3u^2 - 4u + 1)P_i' + (3u^2 - 2u)P'i + 1 \quad (3.7)$$

$$Q_i''(u) = (12u - 6)P_i + (-12u + 6)P_{i+1} + (6u - 4)P_i' + (6u - 2)P_{i-1}'. \quad (3.8)$$

An einem gegebenen innenliegenden Punkt der gesamten Kurve P_{i+1} soll die abgehende zweite Ableitung des Kurvensegments Q_i gleich der eingehenden zweiten Ableitung des Kurvensegments Q_{i+1} sein. Weiterhin wird angenommen, dass jedes Kurvensegment eine gültige Parametrisierung von 0 bis 1 hat. Zusammengefasst ergeben sich also folgende weitere Voraussetzungen:

$$Q_i''(1) = Q_{i+1}''(0) \quad (3.9)$$

und

$$\begin{aligned} 6P_i - 6P_{i+1} + 2P_i' + 4P'i + 1 &= -6P_{i+1} + 6P_{i+1} + 4P_{i+1}' - 2P_{i+2}' && (3.10)\\ \Leftrightarrow 2P_i' + 8P_{i+1}' + 2P_{i+2}' &= 6[(P_{i+2} - P_{i+1}) + (P_{i+1} - P_i)] && (3.11)\\ \Leftrightarrow P_i' + 4P_{i+1}' + P'i + 2 &= 3(P_{i+2} - P_i) && (3.12) \end{aligned}$$

Angewendet auf die gegebenen Punkte $P_0, ..., P_n$ erhält man $n - 1$ lineare Gleichungen, die als Matrixprodukt folgendermaßen dargestellt werden können [2]:

$$\begin{bmatrix} 1 & 4 & 1 & \cdots & \cdots & 0 & 0 \\ 0 & 1 & 4 & 1 & \cdots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0\cdots & 1 & 4 & 1 & 0 & \\ 0 & 0 & \cdots & 0 & 1 & 4 & 1 \end{bmatrix} \begin{bmatrix} P'_0 \\ P'_1 \\ \vdots \\ P'_{n-1} \\ P'_n \end{bmatrix} = \begin{bmatrix} 3(P_2 - P_0) \\ 3(P_3 - P_1) \\ \vdots \\ 3(P_{n-1} - P_{n-3}) \\ 3(P_n - P_{n-2}) \end{bmatrix} \tag{3.13}$$

Die $n-1$ Gleichungen mit $n+1$ Unbekannten können nur durch zwei weitere Gleichungen gelöst werden, welche durch die Tangenten, die nun neben der bereits beschriebenen Kondition des ¤iessenden Übergangs zwischen zwei Segmenten durch die zweite Ableitung auch die Position des Extremwerts der Kurve erfüllen müssen, erlangt werden können. Die Tangenten können daher den gegebenen Werten v_0 und v_1 zugewiesen werden, so dass

$$Q'_0(0) = P'_0 = v_0 \text{ und } Q'_{n-1}(1) = P'_n = v_1.$$

Mit Hilfe der zwei zusätzlichen Gleichungen kann das endgültige Gleichungssystem hergeleitet werden, dessen Lösung die passenden Tangentenvektoren liefert:

$$\begin{bmatrix} 1 & 0 & 0 & 0 & \cdots & 0 & 0 \\ 1 & 4 & 1 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 4 & 1 & \cdots & 0 & 0 \\ \vdots & & & & & & \\ 0 & 0 & \cdots & 1 & 4 & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 & 4 & 1 \\ 0 & 0 & \cdots & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} P'_0 \\ P'_1 \\ \vdots \\ P'_{n-1} \\ P'_n \end{bmatrix} = \begin{bmatrix} v_0 \\ 3(P_2 - P_0) \\ 3(P_3 - P_1) \\ \vdots \\ 3(P_{n-1} - P_{n-3}) \\ 3(P_n - P_{n-2}) \\ v_1 \end{bmatrix} \tag{3.14}$$

3.1.4 Berechnung von Catmull-Rom Splines

Eine Alternative zu der mit der 'second degree continuity' umgangenen Tangenteneingabe durch den Benutzer bieten die sogenannten *Catmull-Rom Splines*, die sich von den Hermite-Kurven lediglich durch eine Generierung der Tangenten durch einfache geometrische Prozeduren unterscheiden [3].

Der Vorteil der Catmull-Rom-Splines liegt in dem geringen Verbrauch von Rechenzeit bei der Berechnung einzelner Tangenten. Ändert der Benutzer jedoch die Position eines Keyframe, reicht nicht mehr, wie bei der Hermite-Interpolation, eine Neuberechnung der Nachbarsegmente; vielmehr müssen nach einer Veränderung eines Kontrollpunktes auf der Kurve vier Segmente

neu berechnet werden [3]. Eine Manipulation der Tangente in Bezug auf ihre Länge wirkt sich dagegen wieder nur auf die beiden benachbarten Segmente aus.

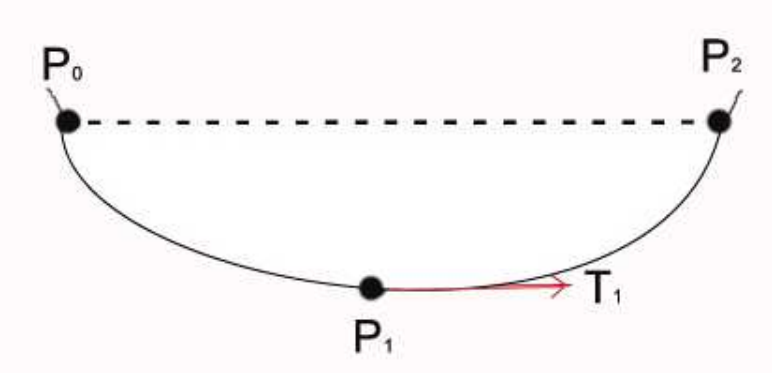

Abbildung 3.3: Die Tangentenberechnung mit Catmull-Rom

Für jeden inneren Keyframe P_i auf dem festgelegten Pfad kann die Tangente P_i' dieses Punktes, wie in Abbildung 3.1.4 dargestellt, durch die Hälfte des Vektors vom vorherigen Keyframe P_{i-1} bis zum nächsten Keyframe P_{i+1} berechnet werden:

$$P_i' = \frac{1}{2}(P_{i+1} - P_{i-1}) \tag{3.15}$$

.
Eingesetzt in die Matrix der Hermite-Kurve zwischen P_i und P_{i+1} ergibt das

$$Q_i(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_i \\ P_{i+1} \\ \frac{1}{2}(P_{i+1} - P_{i-1}) \\ \frac{1}{2}(P_{i+2} - P_i) \end{bmatrix} \tag{3.16}$$

und umgeformt

$$Q_i(u) = \begin{bmatrix} u^3 & u^2 & u & 1 \end{bmatrix} \frac{1}{2} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 2 & -5 & 4 & -1 \\ -1 & 0 & 1 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix} \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix} \tag{3.17}$$

Hieraus kann eine Kurve von Keyframe P_1 bis Keyframe P_{n-1} berechnet werden. Da bei dieser Formel jedoch die Endkeyframes P_{-1} und P_{n+1} nicht einbezogen wurden, bilden die Kurvensegmente Q_0 und $Qn-1$ einen Sonderfall in der Berechnung. Anstelle einer benutzerdefinierten Eingabe der zugehörigen Tangenten kann auch hier eine automatische Generierung ver-

wendet werden. Dabei werden der zweite und dritte Keyframe genutzt und der Vektor zwischen diesen beiden Punkten von dem zweiten Punkt subtrahiert. Es ergibt sich ein neuer Punkt $V = -(P_2 - P_1)$, der die Richtung für die abgehende Tangente des ersten Keyframes vorgibt (siehe Abbildung 3.1.4). Die Tangente kann also wie folgt berechnet werden:

$$P'(0) = \frac{1}{2}(P_1 - (P_2 - P_1) - P_0) = \frac{1}{2}(2P_1 - P_2 - P_0) \tag{3.18}$$

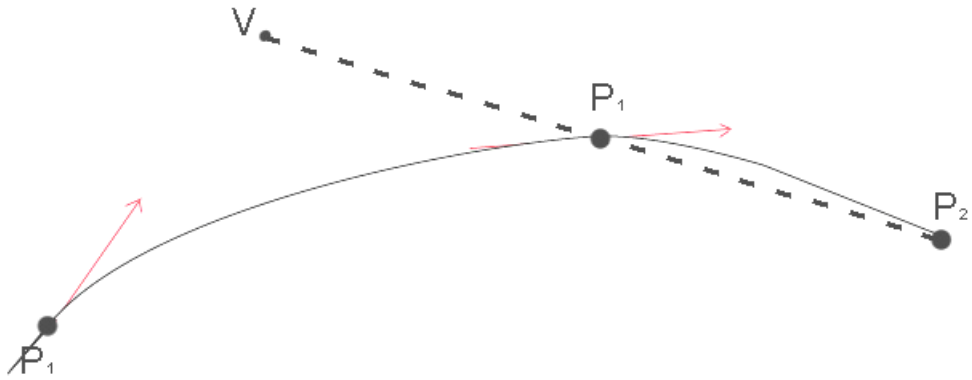

Abbildung 3.4: Tangentenberechnung für den ersten Keyframe

Damit kann der komplette Animationspfad des Objektes bestimmt und das einzelne Objekt durch Unterteilung jedes Kurvensegments in kleinere Teilintervalle mit Hilfe geeigneter OpenGL-Befehle entlang der Kurve bewegt werden.

Die Kurve bezeichnet dabei die *Raumkurve*, die jedes Objekt entlang des durch den Benutzer bestimmten Pfades durch den $3D$-Raum führt. Die Position des Objektes entspricht zunächst zu einem festgelegten Zeitpunkt t jeweils einem Punkt $\vec{P}$ auf der Kurve. Dieses Verhalten wird durch die *Weg-Zeit-Funktion* begründet, die jedem Zeitpunkt t die bis dahin zurückgelegte Kurvenstrecke zuordnet. Diese Kurvenstrecke bezeichnet man auch als *Bogenlänge s* und beschreibt damit die Länge des Weges auf der Kurve bis zu einem Punkt $\vec{P}$ in Abhängigkeit von der Zeit t und dem Kurvenparameter u. Mit Hilfe der Bogenlänge können wir jetzt im folgenden Abschnitt die Bewegung des Objektes auch als zeitliches Problem betrachten.

3.2 Interpolation der Zeit

In Abschnitt 2.2.2 wurde bereits kurz auf die Manipulationsmöglichkeit der Geschwindigkeit durch den Benutzer eingegangen. Jeder Keyframe wird zunächst mit einer voreingestellten Geschwindigkeit von 1 erstellt, welche jederzeit durch den Benutzer auf einer Skala von 1 - 10 verändert werden kann. Dabei ist es wichtig, die Geschwindigkeit nicht im Zusammenhang mit dem Keyframe zu sehen; vielmehr bedeutet die Veränderung der Geschwindigkeitswerte eine Veränderung der Geschwindigkeit, mit der ein Objekt den Keyframe erreicht, bzw. passiert. Daher gilt es, zwischen drei Fällen zu unterscheiden:

1. Zwei benachbarte Keyframes k_0, k_1 haben den gleichen Geschwindigkeitswert, so dass das Objekt mit konstanter Geschwindigkeit zwischen k_0 und k_1 bewegt wird.

2. Zwei benachbarte Keyframes k_0, k_1 unterscheiden sich in ihrem Wert in der Art, dass für k_0 eine geringere Geschwindigkeit eingestellt gilt als für k_1.

3. Für k_0 wurde eine höhere Geschwindigkeit eingestellt als für k_1.

Im zweiten und dritten Fall muss zwischen den beiden Keyframes zeitlich interpoliert werden, so dass eine fliessende Bewegung des Objektes entsteht. Hat k_0 eine geringere Geschwindigkeit als k_1, so muss das Objekt beschleunigen, umgekehrt muss es abbremsen (siehe 3.2.1). Diese Eigenschaften bezeichnet man als *ease in* und *ease out* [4].
Zunächst soll allerdings auf den ersten Fall eingegangen werden, der durch seine mathematischen Berechnungsvoraussetzungen die Lösung für die beiden Fälle der Interpolation zwischen unterschiedlichen Geschwindigkeitswerten einleitet. Das Objekt soll sich demnach vorerst bei gleichem Geschwindigkeitswert zwischen zwei Keyframes in konstantem Tempo entlang der Kurve bewegen. Im mathematischen Sinne muss sich bei unterschiedlicher Krümmung der Kurve auch die erste Ableitung unterscheiden, weshalb die Bogenlänge, die bei konstanter Geschwindigkeit durchlaufen wird, vom Ausgangspunkt auf der Kurve abhängt. Abbildung 3.2 zeigt eine Kurve, die in äquidistante Teilintervalle des Parameters u unterteilt ist. Deutlich zu sehen ist jedoch, dass sich die einzelnen Kurvensegmente in ihrer Länge unterscheiden. Damit würde das Objekt stark gekrümmte Kurvenabschnitte schneller passieren als weniger oder gar nicht gekrümmte Intervalle der Kurve.
Die Idee ist nun, eine Distanz s auf der Kurve mit Hilfe einer gegebenen konstanten Geschwindigkeit r sowie einer Zeit t zu bestimmen, so dass $s = rt$. Der Ausgangspunkt auf der Kurve entspricht hierbei u_1, gesucht wird der Parameter u_2. Die Bogenlänge zwischen $Q(u_1)$ und $Q(u_2)$ muss dann dieser Distanz s entsprechen [2].

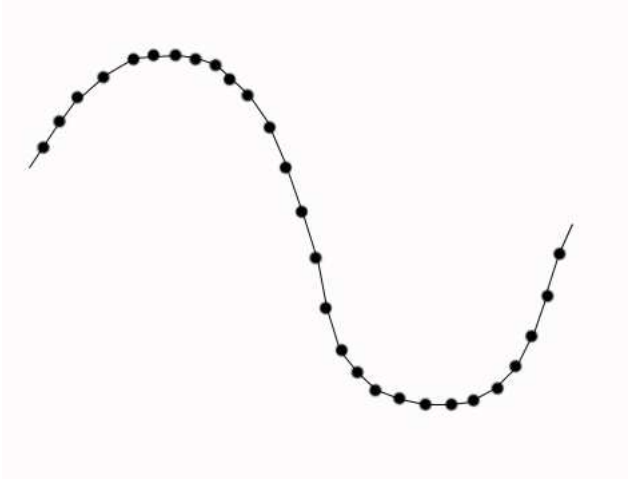

Abbildung 3.5: Geschwindigkeitsverhalten entlang einer Kurve

Um die Zeit kontrollieren zu können, die ein Objekt beim Passieren eines gewissen Kurvenabschnitts braucht, muss demnach eine Beziehung zwischen den parametrischen Werten und der Bogenlänge aufgebaut werden. Angenommen, eine Funktion $s = G(u)$ berechnet die Bogenlänge vom Startpunkt $Q(0)$ bis zu dem Punkt, der mit dem parametrischen Wert $Q(u)$ übereinstimmt. Dann kann für den Fall $u_1 = 0$ die inverse Funktion $G^{-1}(s)$ bei gegebener Länge s zur Bestimmung des Parameters u_2 eingesetzt werden. Diesen Vorgang bezeichnet man als *Reparameterisierung* [2].

Um die Kurve neu zu parameterisieren, können verschiedene Verfahren eingesetzt werden. Die analytische Berechnung der Länge eines Kurvenabschnitts erfolgt mittels Integration, kann jedoch für die meisten kubisch polynomiellen Kurvenformen nicht eingesetzt werden [4]. Eine Alternative bietet sich durch die numerische Abschätzung des Integrals, was den Vorteil der Minimierung der Funktionsevaluierungen hat. Die genaueste Methode ist hierbei die 'Gaussian quadrature', bei der ein endliches Integral $[-1, 1]$ durch $\int_{-1}^{1} f(x)dx \approx \sum_{i=1}^{n} a_i f(x_i)$ angenähert wird [2]. c_i und x_i hängen dabei von n ab, so dass das Integral durch die gewichtete Summe aus verschieden verteilten Funktionsberechnungen approximiert werden kann.

Eine einfachere Methode bietet jedoch die Annäherung der Kurve durch Geraden an verschiedenen parametrischen Werten: Jeweils zwei benachbarte Punkte P_0 und P_1 auf einer Kurve $P(u)$ mit $u = 0.00, 0.05, 0.10, ..., 1.0$ werden durch eine Gerade verbunden. Die hieraus berechnete Distanz zwischen diesen zwei Punkten wird in einer Tabelle, der *Bogenlängentabelle* (siehe Abbildung 3.2), dem Parameterwert der Kurve zugeordnet. Die Bogenlänge der Kurve $P(u)$ bis

zu einem Punkt $P(0)$ berechnet sich demnach aus der Summe der Geraden bis zu diesem Punkt [5].

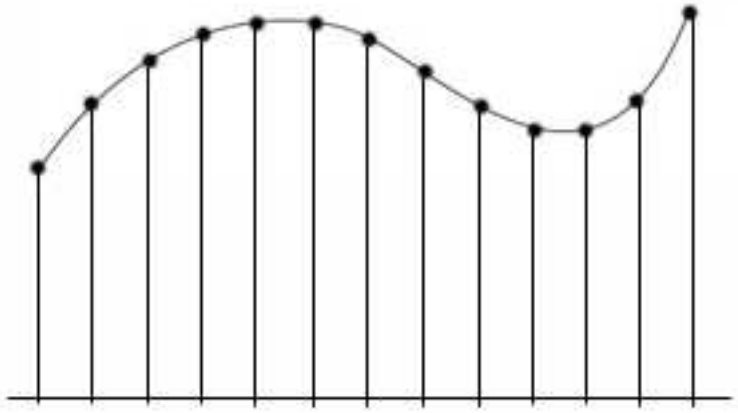

Abbildung 3.6: Approximation der Bogenlänge

Jeder Bogenlänge s kann mithilfe dieser Tabelle schließlich ein Parameterwert u zugeordnet werden, so dass die Distanz, die ein Objekt in einem bestimmten Zeitintervall zurücklegen soll, einfach gefunden werden kann.

3.2.1 Ease-in und Ease-out

Die oben beschriebene Berechnung der Tabelle dient zur Kontrolle der Geschwindigkeit, mit der das Objekt das Kurvensegment passieren soll. Wird das Objekt demnach entlang äquidistanter Teilintervalle über die Bogenlänge bewegt, so resultiert das in einer Animation mit konstanter Geschwindigkeit. Dagegen kann die Bewegung des Objektes durch eine Bogenlängen-Zeit-Relation beschleunigt (*ease-in*), bzw. abgebremst (*ease-out*) werden. Der Eingabeparameter bezieht sich hierbei auf die Zeit t; gesucht wird die Bogenlänge s. Meist wird die Kurvenparameterisierung 'normalisiert', so dass der Parameterwert zwischen 0 und 1 variiert. Die normalisierte Bogenlänge entspricht dem Bogenlängenparameter geteilt durch die gesamte Bogenlänge der Kurve [4]. Die Beschleunigung, bzw. das Abbremsen eines Objektes kann damit durch eine Funktion $s(t) = ease(t)$ implementiert werden, wobei die Normalisierung von t Werte zwischen 0 und 1 voraussetzt [4]. $s(t)$ ist, wie schon oben beschrieben, die gesuchte Funktion, die jedem Zeitwert t die zurückgelegte Wegstrecke s zuordnet. $s(t)$ drückt damit die gewünschte Distanz aus, die das Objekt zu einem Zeitpunkt t bereits passiert haben soll. Um nun einen Punkt auf der Raumkurve zu erhalten, wird die Kurve an der Stelle u berechnet, nachdem der zu der Bogenlänge korrespondierende Wert u in der Bogenlängentabelle gefunden worden ist [1].

Parameterwert	Bogenlänge
0.0	S0
0.1	S1
0.2	S2
0.3	S3
0.4	S4
0.5	S5
0.6	S6
0.7	S7
0.8	S8
0.9	S9
1.0	S10

Abbildung 3.7: Die Bogenlängentabelle

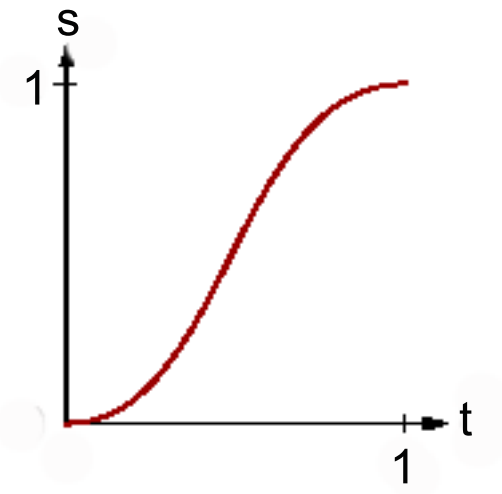

Abbildung 3.8: Definition der Weg-Zeit-Relation mit natürlicher Bewegung des Objektes

3.3 Die Rotation

Die Animation der Objekte in dem Animationseditor wurde bisher lediglich in Bezug auf die Interpolation der Position betrachtet. Obwohl sich auch die Kameraausrichtung verändern lässt, ist es zusätzlich wünschenswert, auch die Orientierung der Objekte beeinflussen zu können. Hierfür ist eine Interpolation der Richtung nötig, so dass die Objekte sich entlang des Pfades um ihre eigenen Achsen drehen können. Die Veränderung der Orientierung bedeutet daher auch die Änderung der Position des Objektkoordinatensystems im Verhältnis zu dem Weltkoordinatensystem.

Während das Problem der Interpolation zwischen den Keyframes die Suche nach einer Funktion, die für einen Zeitwert t eine Position P auf der Kurve liefert, betraf, muss bei der Darstellung der Rotation zwischen einer Reihe von Objektorientierungen interpoliert werden. Die Repräsentation der Orientierung kann durch verschiedene Methoden, wie z.B. der Achse-Winkel-Rotation oder der Rotation durch Quaternionen, implementiert werden. Für diese Arbeit wurde jedoch die Repräsentation durch *Euler Winkel* gewählt, die sich durch eine relativ einfache und intuitive Struktur auszeichnet [4].

3.3.1 Rotation mit Euler-Winkeln

Die Voraussetzung für die Repräsentation einer Rotation im dreidimensionalem Raum sind mindestens drei Werte, welche aus den Winkeln einer sequentiellen Rotation um ein Koordinatensystem aus orthogonalen Achsen bestehen können. Da das lokale Objektkoordinatensystem unabhängig von dem Weltkoordinatensystem ist, kann die Orientierung der Objekte durch Euler-Winkel dargestellt werden [3]. Die Rotationsachsen sind damit die lokalen Achsen, die einzelnen Objekten zugeordnet sind.

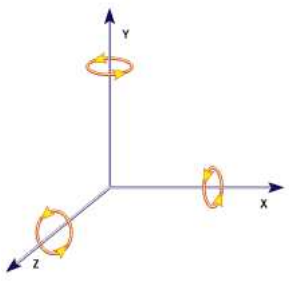

Abbildung 3.9: Lokales Objektkoordinatensystem

In Abbildung 3.3.1 ist ein lokales Koordinatensystem dargestellt, welches wahlweise um die x-, y- oder z-Achse rotieren kann. Ein Tripel von Euler-Winkeln (α, β, γ) bedeutet hierbei immer, dass die Rotation um die x-Achse, von der Rotation um die y-Achse, und diese von der Rotation um die z-Achse gefolgt wird. Jede dieser Rotationen wird dabei durch eine Transformationsmatrix $R_i(\alpha)$ dargestellt, mit $i = x, y, z$. Das Gesamtergebnis folgt aus der Konkatenation dieser Matrizen [6]. Die Achsenrotationen können dabei in beliebiger Reihenfolge ausgeführt werden, d.h. die Rotationssequenz z-x-y ist genauso möglich wie x-y-z oder andere. Es kann ebenso eine Achsenrotation dupliziert werden, z.B. y-x-y. Eine Rotationssequenz mit zwei aufeinander folgenden gleichen Achsen, wie y-x-x ist jedoch nicht erlaubt, denn die daraus entstehende Redundanz trägt zu keinem weiteren Freiheitsgrad bei [2].
Weiterhin werden die Rotationen in umgekehrter Reihenfolge konkateniert: Soll einer Drehung um die z-Achse eine Drehung um die x-Achse folgen, so würde letztere um die schon in neuer Orientierung verlaufende x-Achse erfolgen, was im Normalfall nicht erwünscht ist. Um dies zu vermeiden, wird die gewünschte Drehung mit $R_x R_z$ ausgeführt und entspricht dabei einer Rotation um die lokalen Achsen.

Durch Konkatenation der drei Matrizen der x-y-z Euler-Winkel-Rotationen kann eine verallgemeinerte Rotationsmatrix erstellt werden [2]:

$$R = R_x R_y R_z =$$

$$\begin{bmatrix} cos\theta_y cos\theta_z & -cos\theta_y sin\theta_z & sin\theta_y \\ sin\theta_x sin\theta_y cos\theta_z + cos\theta_x sin\theta_z & -sin\theta_x sin\theta_y sin\theta_z + cos\theta_x cos\theta_z & -sin\theta_x cos\theta_y \\ -cos\theta_x sin\theta_y cos\theta_z + sin\theta_x sin\theta_z & cos\theta_x sin\theta_y sin\theta_z + sin\theta_x cos\theta_z & cos\theta_x cos\theta_y \end{bmatrix}$$

Euler-Winkel sind durch die minimale Anzahl der benötigten Werte gut für eine einfache Darstellung der Objektorientierungen geeignet. Ein großer Nachteil ihrer Verwendung ist jedoch die Möglichkeit, einen Freiheitsgrad zu verlieren, was zu einem sogenannten *gimbal lock* führt. Dieser Fall tritt immer genau dann auf, wenn beispielsweise um 90° um die x-Achse rotiert wird, so dass die vorherige z-Achse des lokalen Koordinatensystems nun mit der negativen y-Achse übereinstimmt. Jede Rotation, die jetzt mit θ_y ausgeführt wird, wird anschließend immer von jeder Rotation mit θ_z abgezogen. Die Kombination der z- und der y-Rotation kann durch einen Wert $\theta_z - \theta_x$ ausgedrückt werden, der ebenso gleich eine Rotation um die x-Achse hätte sein können.

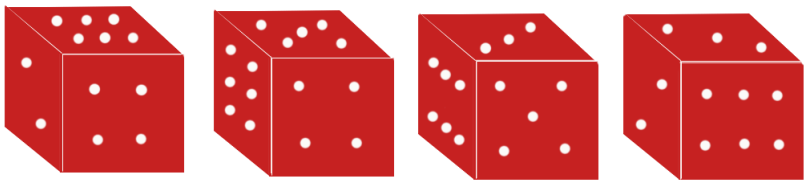

Abbildung 3.10: Beispiel für den *gimbal lock*

Abbildung 3.10 zeigt ein Beispiel des ’gimbal locks’: Ein Würfel, der um die z-Achse, danach um die x-Achse und schließlich um die y-Achse gedreht wird (immer ausgehend von den feststehenden Achsen des Weltkoordinatensystems), ist in seiner Position am Ende gleich einer einzigen Rotation um die x-Achse [2].

4 Implementation der Software

Der Animationseditor wurde in C++ programmiert, wobei die Implementation zusätzlich auf die Klassenbibliotheken *Qt* von Trolltech und *OpenGL* basiert.
Qt liefert ein System zur grafischen Oberflächenprogrammierung mit Kompatibilität zu C++, Objektorientiertheit und sehr hoher Ausführungsgeschwindigkeit.
Das GUI von *The Animator* besteht aus verschiedenen Modulen und Bedienelementen, sogenannte *Widgets*, der QT-Klassenbibliothek, bei deren Implementation speziell auf die intuitive Bedienbarkeit geachtet wurde. Die Zusammensetzung der einzelnen Module im Layout soll an dieser Stelle durch ein Codefragment aus der Klasse AnimationSoftwareWindow.cpp, in der sich die Implementation des Layouts und sämtliche Slots der Bedienelemente befinden, erläutert werden:

Auszug aus der Klasse AnimationSoftwareWindow.cpp:

```
// Anlegen des Hauptfensters
QFrame* mainframe = new QFrame(this);

// Einbettung des Animationsfensters
glwindow = new AnimationWindow(mainframe);

// vertikales Layout
QVBoxLayout *layout = new QVBoxLayout (this);

// menubar wird "layout" zugefügt
QMenuBar* menubar = new QMenuBar (this);
layout->addWidget(menubar);

QVBoxLayout* mainlayout = new QVBoxLayout( layout );
```

```
// Anlegen der Toolbar
QToolBar* toolbar = new QToolBar(this);
// toolbar und mainframe werden dem mainlayout zugefügt
// mit der Gewichtung 1:20
mainlayout -> addWidget(toolbar,1);
mainlayout -> addWidget(mainframe,20);

// horizontales Layout für für mainframe
QHBoxLayout* framelayout = new QHBoxLayout(mainframe);

QFrame* left = new QFrame(mainframe);
framelayout -> addWidget(left,1);
  framelayout -> addWidget(glwindow,15);

// "left" bekommt vertikales Layout
QVBoxLayout* leftlayout = new QVBoxLayout(left);

// die Objektliste wird angelegt
objectbox = new QListBox(left);
objectbox->setMargin( 8 );
leftlayout->addWidget(objectbox);
connect (objectbox , SIGNAL(selectionChanged()) ,
this , SLOT(selectObject()));

 // unter die Objektbox kommt das Fenster für die Keyframesettings
KeyframeWindow* keyframewindow = new KeyframeWindow(left);
keyframewindow->setAnimationSoftwareWindow();
leftlayout ->addWidget(keyframewindow);

// unter das KeyframeWindow kommt die Keyframe-Toolbar
QHGroupBox* keyframebox = new QHGroupBox (left);
leftlayout-> addWidget ( keyframebox);
```

Neben den Widgets stellt die Bibliothek verschiedene Datenstrukturen zu Verfügung, die eine problemlose Integration in den Quellcode erlauben. Die für diese Arbeit benötigten Listen konnten somit als QList implementiert werden. Die Konstruktoren der Klassen Object.cpp und

Objectlist.cpp konnten demnach folgendermaßen implementiert werden:

Auszug aus der Klasse ObjectList.cpp:

```
// Liste aus Zeigern auf die Klasse ’Object.cpp’
ObjectList::ObjectList() : QList <Object>(){
      [...]
}
```

Auszug aus der Klasse Object.cpp:

```
// die Klasse ’Object.cpp’ ist eine Liste aus Zeigern
// auf die Klasse ’Keyframe.cpp’
Object::Object(QString n, QString t) : QList <Keyframe>(){
  name=n;
type=t;
keyframe_no = 0;

  append(new Keyframe());

[...]

}
```

Für die grafische Darstellung der Zeichenroutinen wurde OpenGL unter zusätzlicher Einbindung von GLUT verwendet. Jede Änderung der Ansicht bewirkt das Neuzeichnen eines Frames und damit den Aufruf der Methode paintGL(). Um die Funktionsweise dieser Zeichenroutinen zu verstehen, soll auch hier ein Programmfragment die Iterationsabfolge der zu zeichnenden Objekte verdeutlichen:

Auszug aus der Klasse AnimationWindow.cpp

```
void AnimationWindow::paintGL() {

// lösche den Farb- und Tiefenpuffer
glClear ( GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
// aktivieren der Modelview-Matrix
```

```
glMatrixMode (GL_MODELVIEW);
glLoadIdentity ();
// Lichtquelle setzen, die Position ist dabei
// GLfloat position[4] = {1.0,1.0,1.0,0.0};
glLightfv ( GL_LIGHT0 , GL_POSITION , position );
// Kameraeinstellung
// (kann durch Mausbewegung oder Tastatur verändert werden)
gluLookAt ( ax,ay,az , bx,by,bz , cx,cy,cz );
// setze die Farbe auf Grau
glColor3f(0.5f, 0.5f, 0.5f);
// zeichne Grundmuster
for (float step=-50.0;step<=50.0;step+=2.0){
glBegin(GL_LINES);
  // senkrechte Linien
glVertex3f(step,0.0,-50.0);
glVertex3f(step,0.0,50.0);
// waagerechte Linien
glVertex3f(-50.0,0.0,step);
glVertex3f(50.0,0.0,step);
glEnd();
}

// ruft die Methode zum Zeichnen des Koordinatenkreuzes auf
drawCoordinationCross(-100.0, 100.0);

//////// draw all object in list ////////////

// wenn die Objektliste nicht leer ist
if ( !(listofobjects->isEmpty()) ){
// iteriere durch die gesamte Objektliste
   for (int i=0; i < (listofobjects->count()); i++){
// keylist ist das aktuelle Objekt (und seine Keyframeliste)
   keylist = listofobjects->at(i);

     if ( !(keylist->isEmpty()) ){
// finde den aktiven Keyframe
```

```
    Keyframe* activekeyframe = keylist->current();
// finde die x-, y- und z-Koordinaten des aktuellen Keyframes
float x = (activekeyframe->getPosition(1)/10.0f);
float y = (activekeyframe->getPosition(2)/10.0f);
float z = (activekeyframe->getPosition(3)/10.0f);
// getType() liefert den Objekttyp zurück (Cube, Cone, Sphere)
QString objecttype = (keylist->getType());

if (objecttype == "Sphere") {
glPushMatrix();
// bewege das Objekt an die aktuelle Position
glTranslatef(x,y,z);
glColor3f(0.0f, 0.0f, 1.0f);
// zeichne Objekt (radius,slices,stacks)
glutWireSphere(1.5,18.0,10.0);
glPopMatrix();
glFlush();

}

if (objecttype == "Cone"){
glPushMatrix();
// bewege das Objekt an die aktuelle Position
glTranslatef(x,y,z);
glColor3f(0.0f, 1.0f, 0.0f);
// zeichne Objekt (base, height, slices, stacks)
glutWireCone(2.0, 2.0, 10.0, 2.0);
glPopMatrix();
}

if (objecttype == "Cube"){
glPushMatrix();
// bewege das Objekt an die aktuelle Position
glTranslatef(x,y,z);
glColor3f(1.0f, 0.0f, 0.0f);
// zeichne Objekt (side)
```

```
glutWireCube(2.0);
glPopMatrix();
}
glFlush();
} // end if keylist is not empty

[...]

 } // end for all object in list
  } // end if objectlist is not empty

    [...]

} // end paintGL()
```

Die Methode paintGL() enthält zusätzlich noch ähnliche Implementationen für das Zeichnen der Kurve, der Tangenten und für die Interpolation. Neben dieser Zeichenroutine besteht die Klasse *AnimationWindow.cpp* zudem aus verschiedenen Initialisierungsmethoden sowie Methoden zur Steuerung der Kamerabewegung über die I/O-Schnittstelle.

5 Der Animationseditor

Der Animationseditor basiert auf den oben beschriebenen theoretischen Grundlagen und dient demnach zur Erstellung von Animationen. Mit der implementierten Software kann der Benutzer also Objekte erstellen und diese entlang eigens definierter Pfade bewegen. Die Objekte lassen sich dabei einer Liste hinzufügen und durch späteren Zugriff auf diese Liste nach Belieben löschen. Außerdem kann in der Liste der Objekte frei navigiert werden, so dass ein beliebiges Objekt anwählbar ist.
Jedes Objekt wird mit einem Keyframe initialisiert, der ihm zunächst die Position $(0, 0, 0)$ vorgibt. Der Benutzer kann beliebig viele weitere Keyframes hinzufügen und hat damit die Möglichkeit, einen Pfad zu erstellen. Für jeden Keyframe kann die Position jederzeit geändert werden. Auch erlaubt die Navigationsmöglichkeit zwischen einzelnen Objekten und deren Keyframes eine ständige Änderung der Attribute und somit die Veränderungsmöglichkeit des Objektpfades. Sind mindestens zwei Keyframes für ein Objekt erstellt worden, so werden diese Positionen durch einen Pfad verbunden, der die Wegstrecke des Objektes in einer späteren Animation darstellt. Zwei Keyframes werden durch eine Gerade verbunden, drei oder mehr Keyframes durch eine zusammengesetzte Kurve, die zwischen den Punkten interpoliert. An jedem dieser Keyframes wird zudem eine Tangente angezeigt, die den Grad der Ausbuchtung der Kurve bestimmt. Der Benutzer erhält über das Keyframefenster die Möglichkeit, diese Tangente in ihrer Länge zu verändern und somit auf die Rundung der Kurve an dem aktiven Keyframe Einfluss zu nehmen.
Die Anzeige des Kurvenverlaufs dient als Vorschau, wie die einzelnen Objekte in der späteren Animation bewegt werden. Um eine bessere Übersicht über die sich möglicherweise überschneidenden Pfade zu erhalten, kann der Benutzer mit Hilfe der Maus oder der Tastatur die Ansicht drehen, kippen oder ein- bzw. auszoomen. Dadurch kann ebenfalls eine beliebige Stelle in dem Fenster fokussiert werden. Mit dieser Option ist es möglich, sich die 2D-Abbildung der Szene auch in dreidimensionaler Form vorstellen zu können.
Die Animation kann letztendlich über einen Player gesteuert werden, der neben der Abspielfunktion auch über eine Stop- und Pause-Taste verfügt. Animiert werden immer alle Objekte einer

Szene, was in einer längeren Implementierungsphase durchaus optional gestaltet werden könnte.

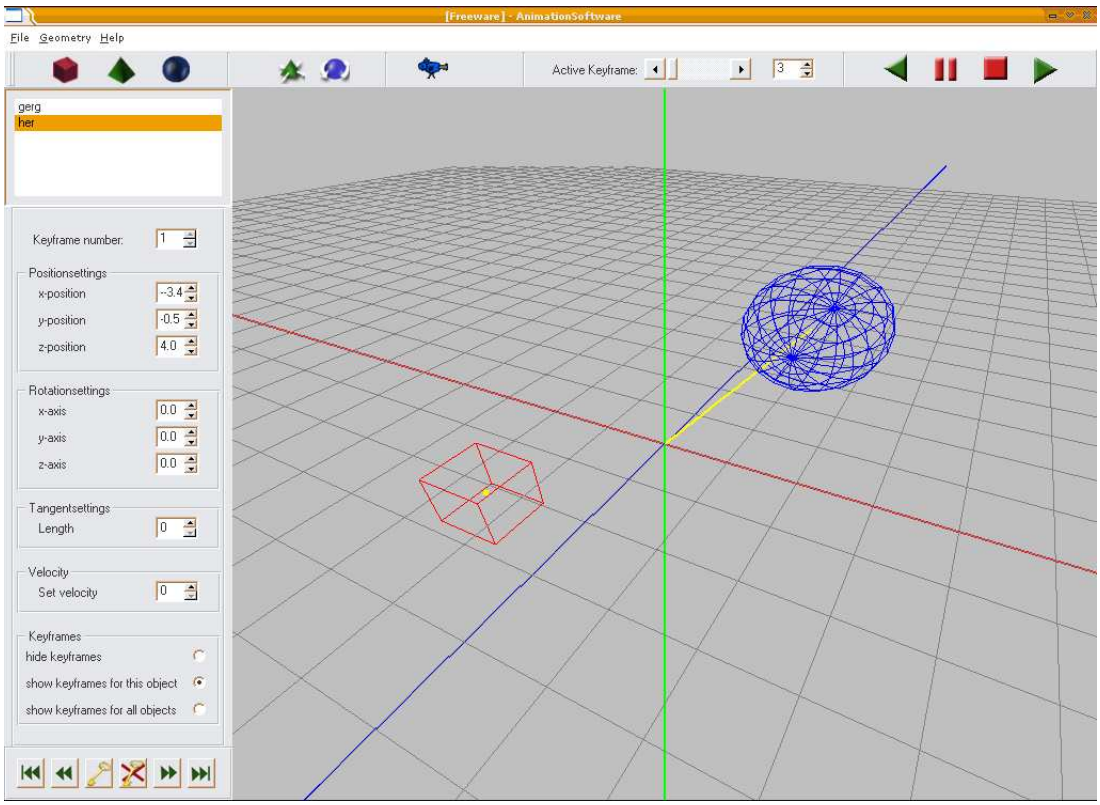

Abbildung 5.1: Die fertige Software

Das System verfügt also über eine einfache Form der Objektanimation, die jedoch über weitere schon implementierte Methoden schnell ausbaufähig sein sollte. Eine Unzulänglichkeit stellt allerdings die Performanz dar, die momentan lediglich ca. 30 Objekte erlaubt, bis das System deutlich langsamer wird (getestet auf einem 2,4 GHz Prozessor mit 512 MB Arbeitsspeicher). Zwar können deutlich mehr Objekte hinzugefügt werden, allerdings wird die Zeitspanne nach dieser Grenze für jede Positionsänderung eines Keyframes deutlich länger. Für die Erstellung größerer Animationen, die eine Reihe von Objekten enthalten, ist die Software daher leider noch ungeeignet. Zur Verbesserung der Benutzerinteraktivität wäre zusätzlich eine Auswahl der Objekte innerhalb des Animationsfensters wünschenswert, ebenso wie eine direkte Manipulationsmöglichkeit der Keyframes und Tangenten. Die Anzeige der lokalen Objektkoordinatensysteme mit gleichzeitiger direkter Rotierungsmöglichkeit der Objekte würde zudem eine intuitivere Erstellung der Animation ermöglichen, war aufgrund der kurzen Implementierungsphase jedoch nicht eingeplant.

6 User Manual

6.1 Introduction

The Animator is an interactive tool for animating objects with the help of key frames. The user has the possibility to add the amount of objects he needs for his scene and to define a curve for each object with a certain number of key frames. The key frames will serve as a path along which the belonging object is going to move when starting the animation. With that the user ist able to build up a whole scene made of basic geometric primitives.

6.2 How to use The Animation

6.2.1 The Graphical User Interface

The Graphical User Interface (GUI) of *The Animator* is divided into three main modules, two of which hold a number of input possibilities. The third module contains the 3D-window with the three coordinate axis x, y and z as well as a grid defining the surface of the plane spanning between the x- and the z-axis. The following sections explain the interactions possibilities in detail.

1. The menubar

   The menubar is situated on top of all other modules holding the File-, Geometry- and Helpmenu. With that it allows the access to the main operations of *The Animator*, though tasks concerning the manipulation of key frames are only accessable through the key frame window which will be explained in section 4. Most importantly, the menubar provides the only access to the file menu to add a new workspace (and with that delete all current data from view), to save an existing workspace, to export the animation to an .avi-file and to exit the whole program. These operations can also be accessed by using shortcuts which are shown next to the menu entry.

2. The toolbar

The toolbar serves for a quick selection possibility of operations that are often used, as for example the adding of objects. There are three different types of objects (cube, sphere and cone) that can be placed into the scene. In addition the toolbar holds the player to show the animation of the objects with the option to rewind, forward pause or stop the movement.

3. The objectlistbox

   Once the user adds an object, its name is shown in the listbox above the keyframewindow. Every name is thereby associated with one object shown in the animation window which allows the user to change the selection of an object and with that to get the active list of keyframes for the manipulation of position and orientation. By right-clicking it is also possible to delete an object from list and from view.

4. The keyframe window

   To define the path an object is supposed to move along, the user is able to add keyframes and change their attributes. In the lower part of the keyframe window a series of keyframe buttons give the possibility to add a new keyframe to the list, to delete a keyframe or to navigate within the list. Note that each object holds exactly one list of keyframes, that is, with adding an object there is also one keyframe added by default to initialize the list. Therefore the path that is defined by one list of keyframes equalizes the curve the belonging object is supposed to move along.

   The user can effect one keyframe in list by choosing the certain keyframe number in the spinbox. By this all attributes will change to the default values or, in case the user has manipulated them before, to the currently saved values of this keyframe. Thereby the position of each keyframe can be changed to define the object's path. In addition, there is the possibility to change the tangent's length for each keyframe which accordingly results in a bigger or smaller expansion of the curve.

   Apart from defining the interpolation of position for each object the user is also able to change the object's orientation so that it might rotate around one of its three axis.

   For an extra possibility of influence on the animation the user is finally able to lay down a value of speed on a scala of 1 to 10 to define the velocity an object is to arrive at this keyframe. With that the object can not only move at constant speed, as it is the default, but also accelerate or slow down between two keyframes.

   Finally, the keyframe window holds the possibiliy of influence on the displayed keyframes in the animation window. By default there are only those keyframes shown that belong to the active object. Toggling between the radio buttons can activate the displaying of the keyframes of all objects or hide all keyframes.

5. The animation window

   The animation window serves as the main module of the software showing all objects and their keyframes. Depending on the number of key frames an object's path consists of, a curve with its belonging tangents on each key is drawn. By this the user can imagine what the animation is going to look like and change the path if it is not what he wants. To get a better insight into the position of different key frames and the built up path it is possible to rotate, tilt or zoom-in and zoom-out of the scene. Moving the mouse to the right and to the left will also result in a movement of the camera view along the positive and negative x-axis. Moving it up or down is equivalent to zooming in and out.

6.2.2 Frequently Asked Questions

- What can I do with *The Animator*?

  The Animator is an interactive application for building up an animation scene. You can add different kinds of basic geometrics and define a path for each of them by adding anchor points, the so-called key frames.

- How can I add an object?

  Adding an object can either be done by a click on the drop down menu 'Object' in the menu bar and choosing the entry 'Cube', 'Cone' or 'Sphere', or by simply using the buttons in the toolbar right below.

- Is there a way to add new key frames?

  Each object comes with one keyframe but additionally you can add as many keyframes you want by clicking on the key button which you will find below the keyframe window. The new key frame will be the active key frame in this module, that is, its number is shown in the spin box labeled 'keyframe number'. Accordingly, all its attributes are shown which will be set to the default values to start with.

- How can I manipulate the position and orientation of an object?

  Once an object is added it will have one key frame at the default position (0,0,0). Changing this position will not only move the keyframe but also move the object to the new values in the animation window. If you add a new key frame the object will again move to the default position.

- How can I preview the animation I created?

You will get a first impression of the object's movement by the path that is drawn between the key frames provided that the specific object has at least two keyframes to connect. To render the scene you can then click on the play button which will result in the animation of all objects from their starting key frame on.

- Do I have the possibility to later influence the position and orientation of the keyframes I created?

  You can change all attributes of the key frames at any time by choosing the specific key frame number that is to be changed and adjust its position and orientation values.

Literaturverzeichnis

[1] D.Eberly. *3D Game Engine Design. A Practical Approach to Real- Time Computer Graphics*. Morgan Kaufmann Publishers, 2000.

[2] L. Bishop J. van Verth. *Essential Mathematics for Games and interactive Applications - A programmer's guide*. Morgan Kaufmann Publishers, 2004.

[3] M. Brill M. Bender. *Computergra£k, Ein anwendungsorientiertes Lehrbuch*. Hanser, 2003.

[4] R. Parent. *Computer Animation Algorithms and Techniques*. Morgan Kaufmann Publishers, 2002.

[5] D. Salomon. *Computer Graphics Geometric Modeling*. Springer, 1999.

[6] P. Shirley. *Fundamentals of Computer Graphics*. AK Peters, 2002.