UNIVERSITÄT
KOBLENZ · LANDAU
Fachbereich 4: Informatik

Entwicklung eines Animationseditors

Studienarbeit

im Studiengang Computervisualistik

vorgelegt von

Stephan Palmer

Betreuer: Prof. Dr. Stefan Müller
Institut für Computervisualistik / Arbeitsgruppe Müller

Koblenz, im Oktober 2006

Erklärung

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☐	☐
Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.	☐	☐

.....

(Ort, Datum) (Unterschrift)

Inhaltsverzeichnis

1 Einleitung

Fristete der Computer vor wenigen Jahrzehnten sein Dasein noch in Rechenzentren weniger, großer Universitäten und Firmen, so sind Computer und deren Rechenleistung heute allgemein verbreitet. Gerade in den letzten Jahren hat dabei auch die computergenerierte Bilderzeugung große Fortschritte gemacht, und die Entwicklung hält rasant an. Auf diese Weise erzeugte Bilder und Bildsequenzen sind aus den visuellen Medien nicht mehr wegzudenken, sie werden in den meisten Film- oder Fernsehproduktionen eingesetzt. Dokumentationen, Wissenssendungen und Nachrichtenformate setzen solche Bilder zur Vermittlung von Inhalten ein. Spezialeffekte werden Filmen mit dem Computer hinzugefügt. Ausschließlich mit dem Rechner erstellte Filme, deren Entwicklung vor ungefähr dreißig Jahren mit einfachen und kurzen Sequenzen begann, stehen heute realen Filmen in Länge, visueller und erzählerischer Qualität kaum noch nach. Doch nicht nur bei der Erstellung von Filmen ist der Computer nicht mehr wegzudenken. Computerspiele stellen in vielen Bereichen die treibende Kraft hinter der Computerentwicklung dar. Sie sind die "Killerapplikation", denn es sind vor allem Spiele, die zur Zeit nach leistungsfähigeren Computerkomponenten für Heimcomputer, wie Prozessoren und Graphikkarten, verlangen. Auch die visuelle Qualität von Computerspielen nimmt stetig zu, und hat ein sehr realitätsnahes Niveau erreicht.

Ob Dokumentation, Film oder Spiel, die Erzeugung computergenerierten Inhalts besteht aus vielen Teilen. Die Erzeugung der Bilder ist nur ein kleiner Teil davon. Der Begriff *Computeranimation* wird oft für den gesamten Prozess der Erstellung eines Films im Computer verwendet. Im Speziellen beschäftigt sich die *Animation* mit der Bewegung. Die Erstellung von Bewegungsabläufen wird für Animationsfilme, Computerspiele und auch für Spezialeffekte in realen Filmen benötigt. Heute existiert eine Vielzahl von Programmen und Werkzeugen, welche die Erstellung von Animationen im Computer ermöglichen. Verbreitete Applikationen wie MAYA[1], LIGHTWAVE[2] oder BLENDER[3] stützen sich dabei auf grundlegende Konzepte, die hinter der Erstellung von Animationssequenzen im Computer stehen. Die Arbeit setzt sich mit diesen Grundlagen auseinander und entwickelt darauf aufbauend einen Animationseditor, der die notwendige Funktionalität für die Erstellung von Animationssequenzen bereitstellt. Die Entwicklung soll neben den theoretischen Grundlagen vor allem Einsicht in die benötigten Programm- und Datenstrukturen geben.

Der folgende Abschnitt geht auf den Begriff der Computeranimation ein. Abschnitt 3 stellt die benötigten mathematischen Grundlagen vor. Darauf aufbauend beschäftigt sich Abschnitt 4 mit den technischen Grundla-

[1] Zugehörige Website im Oktober 2006: http://www.autodesk.com/alias.
[2] Zugehörige Website im Oktober 2006: http://www.newtek.com.
[3] Zugehörige Website im Oktober 2006: http://www.blender3d.org.

gen der Computeranimation. Abschnitt 5 stellt die Anforderungen an das zu entwickelnde Programm auf, und beschreibt die Umsetzung innerhalb dieser Arbeit. Abschnitt 6 schließt mit Zusammenfassung und Ausblick.

2 Begriff der Computeranimation

In diesem Abschnitt wird eingegrenzt, was in dieser Arbeit unter dem Begriff *Computeranimation* verstanden wird. Es soll der Begriff Animation ebenso beleuchtet werden wie der Zusammenhang zwischen technischer und künstlerischer Seite der Computeranimation.

Unter *Animation* versteht man den Vorgang, durch schnell hintereinanderfolgende Darstellung statischer *Einzelbilder* den Eindruck von bewegten Bildern zu erzeugen. Dabei stellen aufeinanderfolgende Einzelbilder dieselbe Szene mit geringfügigen Veränderungen dar. Ergeben die Veränderungen von Anfang bis Ende der Bildsequenz einen sinnvollen Zusammenhang, so erkennt das visuelle Wahrnehmungssystem des Menschen dies als durchgängige Bewegung im Bild. Diese Tatsache ist durch den Umstand zu erklären, das das menschliche Auge nur eine bestimmte Anzahl von Bildern in einem bestimmten Zeitraum unterscheiden kann. Jedes wahrgenommene Bild hinterlässt für eine gewisse Zeit ein *Nachbild* im visuellen System. Bei ausreichender Darstellungsgeschwindigkeit füllen zurückbleibende Bilder dann die Lücken zwischen den Einzelbildern, was den Eindruck kontinuierlicher Bewegung erzeugt. Ist die Darstellung zu langsam, die Anzahl der Einzelbilder zu gering oder sind die Unterschiede zwischen den Einzelbildern zu groß, so wird dies als *Flackern* wahrgenommen. Einzelbilder werden meist als *Frames* bezeichnet, die Anzahl der in einem bestimmten Zeitraum dargestellten Bilder als *Framerate*. Die *Framerate* wird in *Frames pro Sekunde* angegeben.

Klassische, nicht computergestützte Animation verwendet dabei manuell erstellte Einzelbilder, die mit einer Kamera abgefilmt werden. In der Geschichte dieser klassischen Animation machte sich im Besonderen eine Firma einen Namen. Die WALT DISNEY STUDIOS' gehörten zu den Ersten, die animierte Szenen und Filme produzierten, und damit erfolgreich waren. Seit Beginn der zwanziger Jahre des zwanzigsten Jahrhunderts war WALD DISNEY mit seiner Firma maßgeblich an den technischen Fortschritten in der Animation beteiligt, und erfand viele Hilfsmittel für die leichtere und außerdem künstlerisch flexiblere Handhabung des Erstellungsprozesses von handgezeichneten Animationen. Nach vielen Kurzanimationen wie THREE LITTLE PIGS ("Drei kleine Schweine") im Jahr 1933 veröffentlichte Disney im Jahr 1937 seinen ersten, vollen Animationsfilm SNOW WHITE AND THE SEVEN DWARFS '("Schneewittchen und die sieben Zwerge"). Danach beherrschte Disney für viele Jahrzehnte die Animationsbranche, und schuf weltweit erfolgreiche Filme, Serien und Charaktere wie zum Beispiel THE JUNGLE BOOK ("Das Dschungelbuch", 1967, siehe Abbildung 1).

Mit der Entwicklung der Computer wurden diese in der *computergestützten Animation* herangezogen, um den klassischen Animationsprozess zu erleichtern und zu ergänzen. Die grundlegende Herangehensweise

Abbildung 1: Ein Bild aus WALD DISNEYS Animationsfilm THE JUNGLE BOOK.

blieb jedoch gleich. Dies änderte sich spätestens in den Jahren nach 1980. Es wurden Filme produziert, die einen starken Anteil an rein computergenerierten Bildern und Bildfolgen hatten. Filme wie die der STAR WARS TRILOGIE wurden mit computergenerierten Spezialeffekten detailreicher oder gar erst möglich, und die Disney-Produktion TRON kombinierte in weiten Teilen umfangreiche, computergenerierte Umgebungen mit realen Schauspielern. 1985 erstellten die PIXAR ANIMATION STUDIOS dann einen der ersten, komplett im Computer erschaffenen Kurzfilme, LUXO JR. (siehe Abbildung 2). TOY STORY, der erste, rein computergenerierte Film mit voller Länge wurde 1995 ebenfalls von PIXAR produziert. Die Anzahl und Qualität mit *Computeranimation* erschaffener Filme nimmt seither stetig zu[4].

Der Computer übernimmt bei der Computeranimation die Erzeugung der Einzelbilder. Statt von einem Menschen von Hand gezeichnet und illustriert, werden die am Ende in der Bildfolge verwendeten Bilder vom Computer generiert. Dadurch wird jedoch der Mensch in der Produktion einer Animation nicht ersetzt. Stattdessen ändert sich seine Aufgabe. Anstatt die Einzelbilder selbst zu erzeugen, werden nun *Szenenbeschreibun-*

[4]Ein umfangreicher Überblick über die Geschichte der Animation und Computeranimation kann in [Ker04] gefunden werden.

Abbildung 2: Ein Bild aus PIXARS computeranimierten Kurzfilm LUXO JR.

gen vom Künstler in den Computer eingegeben, aus denen dieser dann die entsprechenden Bilder erzeugt. Daher gibt es zwei Aspekte der Computeranimation, die es zu unterscheiden gilt. Auf der einen Seite ist das Erstellen einer Computeranimation immer ein *künstlerischer* Prozess, auch wenn das Eingabemedium nun der Computer, und nicht mehr Papier und Bleistift ist. Auf der anderen Seite müssen die entsprechenden Werkzeuge im Computer geschaffen werden: Werkzeuge für die Eingabe der Szenenbeschreibungen, und Werkzeuge für die Generierung der Bilder aus diesen Beschreibungen. Diese Arbeit legt das Augenmerk auf den zweiten, technisch-informatischen Aspekt der Computeranimation. Die künstlerischen Aspekte, sowie all die anderen wichtigen Seiten einer Produktion von Computeranimationen[5] sollen nur insoweit erwähnt werden, wie sie wichtige *Entwurfsentscheidungen* auf technischer Seite beeinflussen.

Für den Prozess von der Eingabe der Szenenbeschreibung in den Computer bis zum fertigen Einzelbild gibt es verschiedene Typen von *Werkzeugen* beziehungsweise Programmen. Diese Werkzeuge bauen aufeinander auf, d.h. das Ergebnis eines Werkzeugs ist das Ausgangsmaterial für die Anwendung eines nächsten Werkzeugs. Sie bilden also eine Kette von Werkzeugen, die sogenannte *Toolchain*. Diese Kette gliedert also den technischen Realisierungsprozess einer computergenerierten Animation. Eine beispielhafte Beschreibung der einzelnen Schritte ist:

[5]Wie unter anderem das Projektmanagment, welches zu Animationsprojekten gehört. Einen Eindruck dazu gibt [Ker04].

- *Modellierung*: Das Erstellen der äußeren Form oder *Geometrie* der darzustellenden Objekte oder *Modelle*.
- *Rigging*: Entsprechende Modelle werden mit einem *Skelett* versehen, um deren Bewegungsmöglichkeiten zu erweitern.
- *Texturierung*: Die Oberflächenstruktur (Farbe, *Materialien*) der Modelle wird festgelegt.
- *Beleuchtung*: Alle Modelle werden in einer *Szenen* zusammengefügt. In dieser Szene werden *Lichtquellen* hinzugefügt.
- *Animation*: Die Bewegung der Modelle über die Zeit wird festgelegt.
- *Rendering*: Aus der Szeneninformation errechnet der Computer die Einzelbilder der Sequenz. Dazu nutzt er alle Informationen der vorherigen Schritte.
- *Compositing*: Produktion der fertigen Filmsequenz (Hinzufügen von Ton, Zusammenfügen der Einzelbilder, ...). Es können auch weitere Effekte durch Bearbeitung der Einzelbilder hinzugefügt werden.

In dieser Arbeit wird unter *Computeranimation* der entsprechende Teilschritt der *Toolchain* verstanden[6]. Ziel dieser Arbeit ist es, für diesen Animationsteil eine beispielhafte Applikation zu entwickeln. Der Computer kann also aus den erstellten Geometrie-, Oberflächen-, Bewegungs- und Beleuchtungsinformationen ein Bild für einen bestimmten Zeitpunkt in einer Bildsequenz errechnen. Alle Einzelbilder werden dann zu der Sequenz zusammengefügt. Dieser Ablauf ist nochmals beispielhaft in Abbildung 3 gezeigt.

[6]Der Begriff *Computeranimation* wird auch oft für den gesamten Erstellungsprozess verwendet.

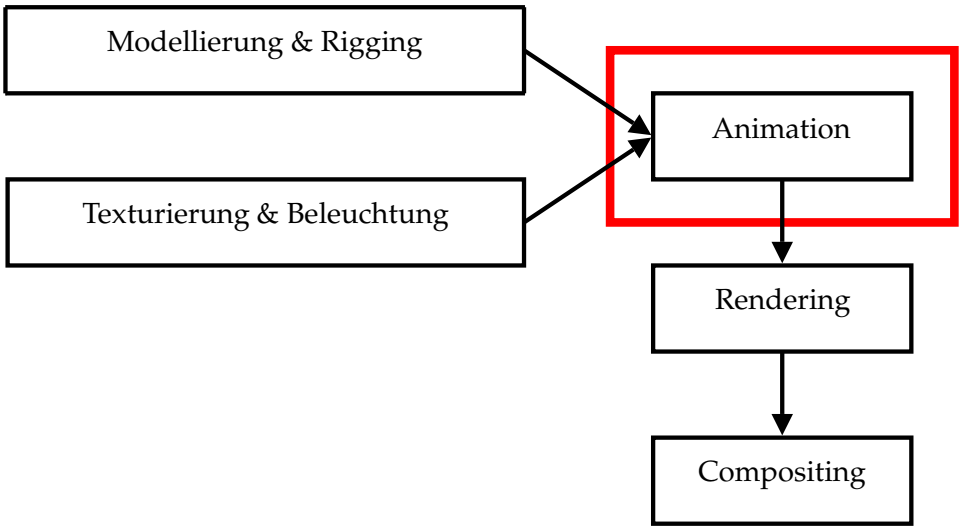

Abbildung 3: Die *Toolchain* zur Erstellung von Computeranimationen in einer beispielhaften Strukturierung. Ein beispielhaftes Werkzeug für den Schritt *Animation* soll in dieser Arbeit entwickelt werden, deshalb ist dieser Schritt in der Graphik hervorgehoben.

3 Mathematische Grundlagen

3.1 Objekteigenschaften

Die Gegenstände einer computergenerierten Animation, d.h. diejenigen Entitäten, die sich bewegen sollen, werden im Folgenden *Objekte* genannt. Die Objekte besitzen eine Vielzahl für den Animationsvorgang wichtige Eigenschaften, deren genaue Anzahl und Auswahl sich in verschiedenen, konkreten Umsetzungen unterscheiden können. Die mathematischen Grundlagen zentraler Eigenschaften sollen in diesem Abschnitt erläutert werden. Dabei wird, ebenso wie in den folgenden Abschnitten, auf eine umfassende Darstellung verschiedener mathematischer Möglichkeiten verzichtet. Stattdessen werden nur für diese Arbeit relevante Lösungen im Detail angesprochen, und an wichtigen Stellen auf weitere Ansätze verwiesen. Eine umfassende Einführung in die mathematischen Grundlagen der Computeranimation kann in [Par02] und [Ben03] gefunden werden.

3.1.1 Position

Die Position eines Objekts wird im Allgemeinen durch einen *Positionsvektor* $\boldsymbol{t} = (t_x, t_y, t_z)$ bestimmt, welcher das Objekt in einem dreidimensionalen *kartesischen Koordinatensystem* lokalisiert. Im Zusammenhang mit der Computeranimation wird t als *Translation* bezeichnet. Die Achsen x, y und z des Koordinatensystems stehen jeweils orthogonal zueinander. Objekte, beziehungsweise genauer deren Positionen, bewegen sich somit in einem dreidimensionalen *Vektorraum*. Für eine genauere Betrachtung der mathematischen Zusammenhänge dieses Raums, wie die Klassifizierung in *Rechtssystem* oder *Linkssystem* und die Operationen des *Skalarprodukts* und *Kreuzprodukts*, sei auf [Shi02] verwiesen.

3.1.2 Orientierung

Die Orientierung eines Objekts wird durch seine Positionierung noch nicht bestimmt, sondern diese muss noch um Rotationsinformationen ergänzt werden. Während im zweidimensionalen Raum für die Beschreibung einer Rotation ein *Rotationsmittelpunkt* und ein *Winkel* ausreicht, so ist dies in einem Raum mit drei Dimensionen aufwändiger. Für die Beschreibung von Rotationen im 3D-Raum gibt es verschiedene, gebräuchliche Techniken, welche im Folgenden kurz erläutert werden. Sie werden außerdem anschaulich und im Detail in [Par02] dargestellt.

Die *Axis-Angle*-Repräsentation (Axis-Angle bedeutet "Achse-Winkel") stellt die dreidimensionale Rotation durch einen Drehwinkel φ um eine beliebige Achse $\boldsymbol{n} = (n_x, n_y, n_z)$ dar. Diese Darstellung ist mächtig genug,

um jede denkbare 3D-Orientierung darzustellen[7]. Mit Hilfe der *Quaternionen*, die eine Erweiterung der *komplexen Zahlen* darstellen, kann die *Axis-Angle*-Darstellung noch verbessert werden. Es ist mit Quaternionen möglich, eine beliebige Achse sowie den Drehwinkel zu kodieren. Die Hintereinanderausführung von Rotationen kann durch Multiplikation entsprechender Quaternionen dargestellt werden. *Axis-Angle* und Quaternionen haben den Vorteil, bei Darstellung und Interpolation von Rotationswerten keine *Singularitäten* aufzuweisen, im Unterschied zu der im nächsten Absatz vorgestellten Repräsentation. Quaternionen bieten zusätzlich in Programmen hohe numerische Stabilität, und es gibt verschiedene Interpolationsmöglichkeiten (zum Thema Interpolation siehe Abschnitt 3.2).

Eine weitere Möglichkeit, Rotationen in drei Dimensionen darzustellen, sind die *Euler-* beziehungsweise *Cardan*-Winkel[8]. Diese wurden aus den in Abschnitt 5.3 beschriebenen Gründen für das entwickelte System verwendet, und werden daher hier genauer betrachtet. Auch mit dieser Repräsentation ist es möglich, jede Orientierung im Raum darzustellen. 3D-Rotationen werden dabei als Kombination von drei Rotationen um drei feste Achsen beschrieben. Die Achsen sind dabei die Koordinatensystemsachsen, und die Reihenfolge der Rotationen ist vorher genau zu bestimmen. Somit sind *Euler-* beziehungsweise *Cardan*-Winkel erst mit der Definition der Reihenfolge der Teilrotationen eindeutig festgelegt. Abbildung 4 veranschaulicht das Prinzip dieser Rotationsdarstellung.

Bei *Euler*-Winkeln werden dabei jeweils nur zwei Koordinatenachsen mit drei Rotationsschritten verwendet. Dies ergibt xyx, xzx, yxy, yzy, zxz und zyz als sinnvolle Reihenfolgen. Im Unterschied dazu verwenden *Cardan*-Winkel alle drei Koordinatenachsen, mit den möglichen Reihenfolgen xyz, xzy, yxz, yzx, zxy und zyx. [Par02] unterscheidet weiterhin zwischen *Fixed-Angle*-Repräsentation (Fixed-Angle bedeutet "Festgelegter Winkel") und *Euler*-Repräsentation[9]. Dabei unterscheiden sich die beiden Darstellung dahingehend, dass bei *Fixed-Angle* stets dieselben Achsen, nämlich die des globalen Koordinatensystems, als Rotationsachsen genutzt werden. Bei der *Euler*-Darstellung werden stets die Achsen des lokalen Objektkoordinatensystems verwendet. Eine Veränderung der Objektposition wirkt sich daher bei diesen Darstellungen unterschiedlich aus.

Alle dargestellten Rotationsdarstellungen, die auf einer Kombination von drei Teilrotationen beruhen, leiden unter einer Singularität, dem Problem des *Gimbal Lock*. Dieser englische Begriff beschreibt das Problem des Verlusts eines Freiheitsgrads bei bestimmten Werten für die Einzelrotatio-

[7] Dies folgt aus *Eulers Rotationstheorem*, welches impliziert, das für jedes Paar zweier Orientierungen gilt: Die eine Orientierung kann aus der Anderen durch eine einzige Drehung um eine bestimmte Achse erzeugt werden.

[8] In der Computergraphik werden *Cardan*-Winkel auch oft als *Euler*-Winkel bezeichnet

[9] [Par02] weist darauf hin, dass diese Begriffe in der Fachliteratur nicht konsistent verwendet werden.

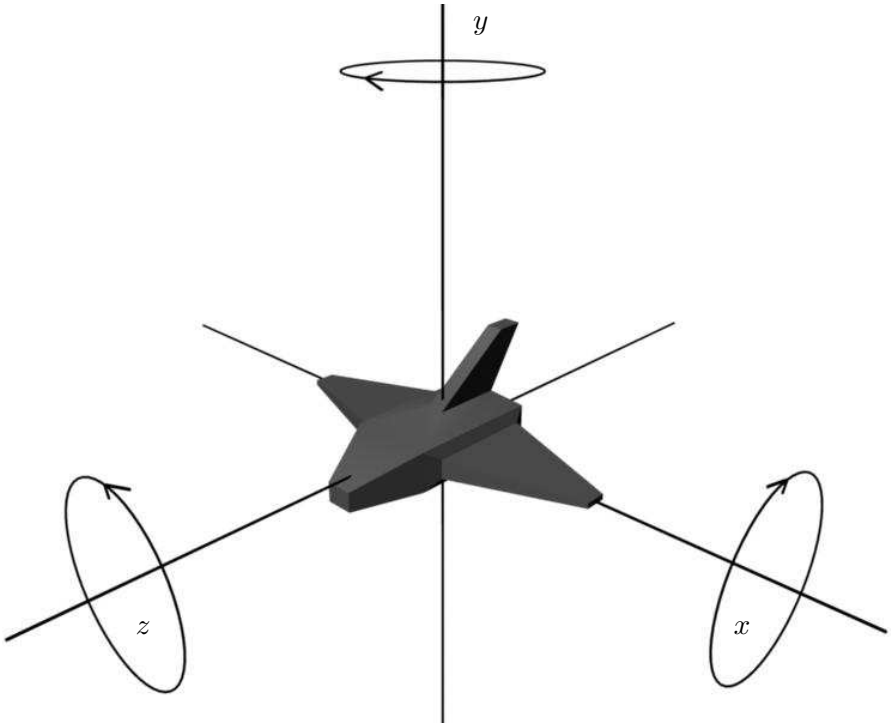

Abbildung 4: Die Drehung um drei Koordinatenachsen in fester Reihenfolge wird bei *Cardan*-Winkeln verwendet, um Rotationen im Raum darzustellen.

nen. So führt z.B. bei *Cardan*-Winkeln mit der Reihenfolge xyz eine Rotation um $(0, 90, 50)$ zum selben visuellen Ergebnis wie die Rotation $(-20, 90, 30)$. Das Problem ist hier intuitiv in der Hintereinanderausführung der einzelnen Rotationen zu suchen, die dazu führen kann, das die erste und die letzte Rotationsachse im globalen Koordinatensystem in die gleiche Richtung zeigen[10]. Also ist es von der Reihenfolge der Rotationsachsen abhängig, welche Achsenrotation zu den Problemen des *Gimbal Lock* führt. Besonders deutlich wird das Problem des *Gimbal Lock* bei der Interpolation von Winkeln in *Euler/Cardan*-Darstellung. Die Rotationswerte werden für jede Achse einzeln interpoliert, und während dieser Interpolation können *Gimbal Lock*-behaftete Winkeldarstellungen durchlaufen werden. Dies wirkt sich in einem visuell höchst unnatürlichen beziehungsweise unerwarteten Verhalten aus.

Ein anschauliches Beispiel für den *Gimbal Lock* gibt [Ben03]. Es ist in Abbildung 5 gezeigt. Das Modell eines Flugzeugs ist entlang der z-Achse ausgerichtet. Dieses soll nun wie in Abbildung 5 im Einzelbild oben rechts ausgerichtet werden. Intuitiv betrachtet ist das Flugzeug dabei um 90 Grad um die y-Achse gedreht, und um 45 Grad um die x-Achse. Eine *Cardan*-Rotation der Reihenfolge xyz mit den Werten $(45, 90, 0)$ erzielt jedoch ein anderes visuelles Ergebnis, welches im unteren Teilbild von Abbildung 5 dargestellt ist. Der Grund ist in der Problematik des *Gimbal Lock* zu suchen. Die gewünschte Konfiguration kann mit den Werten $-90, 135, -90$ erreicht werden, was nicht offensichtlich ist.

3.2 Interpolation und Kurven

3.2.1 Interpolation

Unter *Interpolation* soll hier die Problemstellung verstanden werden, eine beliebige Anzahl von Punkten durch eine Raumkurve so zu verbinden, das die Kurve durch alle Punkte verläuft. Stellen die Punkte den Zustand einer Eigenschaft dar, so ermöglicht es diese Kurve, mit einer variablen Anzahl von Zwischenschritten von einem Zustand in den Nächsten überzugehen. Ein solcher Zustand kann z.B. die Position oder Rotation eines Objekts sein.

In der Mathematik ist das *Interpolationsproblem* verbreitet und lange bekannt. Es lässt sich für den zweidimensionalen Fall wie folgt formulieren: Zu einer Menge von Punkten $\boldsymbol{p_i} = (x_i, y_i)$ wird eine *Funktionskurve* $f(x)$ gesucht, die für alle i die Eigenschaft $f(x_i) = y_i$ besitzt[11]. Die Beschreibung durch mathematische, zweidimensionale Funktionen der Art $f(x) = y$ reicht im Kontext der Computeranimation jedoch nicht aus. Hier ist es notwendig, einen Interpolationsmechanismus zu besitzen, welcher

[10] Mathematisch wirkt sich das Problem des *Gimbal Lock* als Singularität bei der Berechnung von Rotationsmatrizen aus den *Cardan*-Winkeln aus.

[11] Eine mathematisch vollständigere Formulierung befindet sich in [Ben03].

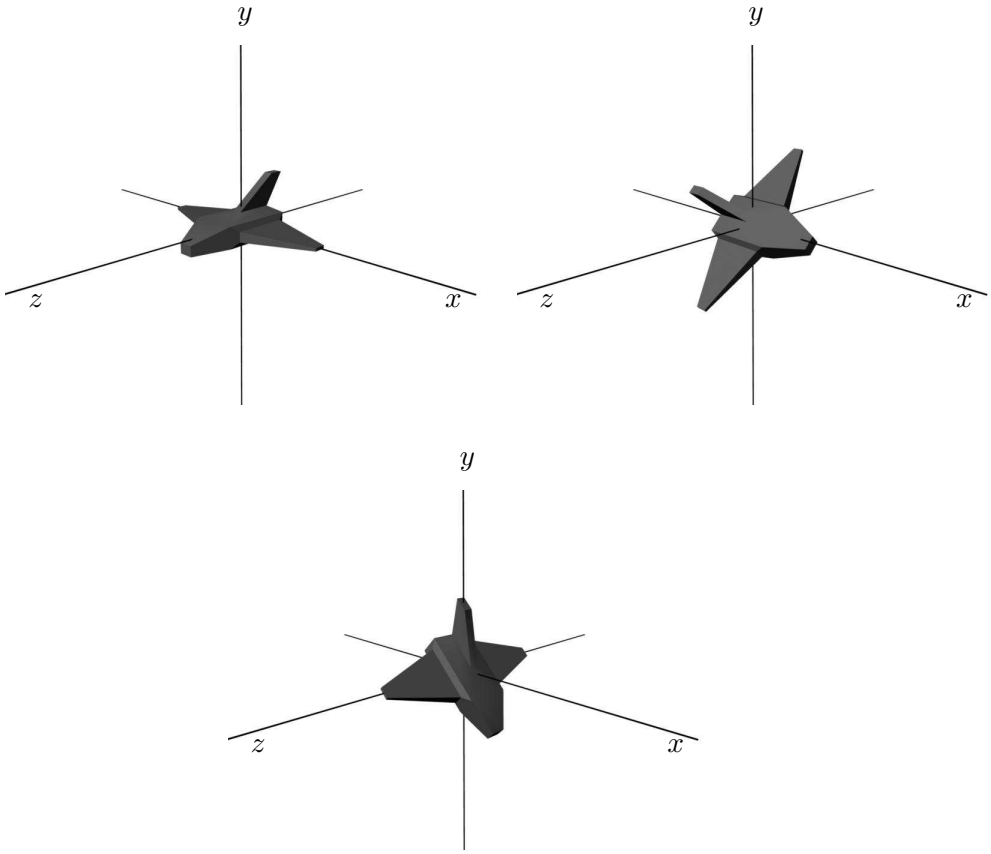

Abbildung 5: Das linke obere Einzelbild zeigt das abstrakte Modell eines Flugzeugs. Mittels einer *Cardan*-Rotation soll es in die Konfiguration der rechten oberen Abbildung gebracht werden. Die nach Betrachtung der Zielkonfiguration intuitiv gewählten Rotationswerte $(45, 90, 0)$ führen jedoch zur Konfiguration im unteren Teilbild. Dies wird durch den *Gimbal Lock* verursacht. Die Zielkonfiguration kann durch die Werte $(-90, 135, -90)$ erreicht werden. Das Beispiel wurde [Ben03] entnommen.

sowohl für Punkte mit zwei als auch mit drei Dimensionen funktioniert (siehe Abschnitt 4).

Daher werden in den nächsten Abschnitten Kurvendarstellungen präsentiert, die sich zur Interpolation von Punkten sowohl im 2D-Raum als auch im 3D-Raum eignen. Es ist weiterhin zweckmäßig, in den Beispielen zunächst auf eine allgemeine Betrachtung der Interpolation einer beliebigen Anzahl von Punkten zu verzichten, obwohl die vorgestellten Verfahren dies beherrschen. Es werden zunächst die Methoden zur paarweisen Interpolation vorgestellt. Diese Betrachtung wird dann um Anmerkungen zum "Zusammensetzen" dieser Einzelinterpolationen ergänzt.

Der einfachste Fall einer Interpolation zwischen zwei Punkten ist die *lineare Interpolation*. Wie bereits der Name andeutet, werden zwei Punkte durch eine Linie verbunden. Die möglichen Zwischenwerte, die dieses Interpolationsverfahren erzeugt, liegen alle auf dieser Linie. Mit einem *Laufparameter* $t \in [0, 1]$ wird also zwischen zwei Punkten p_0 und p_1 interpoliert. Aus Gleichung 1 wird klar, das diese Formulierung sowohl für Punkte mit zwei Elementen, als auch für solche mit drei Elementen funktioniert.

$$\boldsymbol{p}(t) = (1 - t) \cdot \boldsymbol{p_0} + t \cdot \boldsymbol{p_1} \tag{1}$$

Die Dimension des Raums ist für das Funktionieren der Gleichung zur linearen Interpolation nicht bedeutsam.

3.2.2 Kurven

Punkte lassen sich flexibler mit den sogenannten *Bézier*-Kurven verbinden. Dabei ist zu beachten, dass nur der erste und letzte Punkt wirklich auf der Kurve liegen. Das *Bézier*-Verfahren gehöhrt somit zu den *approximierenden* Verfahren, es interpoliert nicht alle Eingabepunkte. Je nach Anzahl der Eingabepunkte, die hier *Kontrollpunkte* genannt werden, unterscheidet sich der *Grad* einer *Bézier*-Kurve, welcher immer um eins unter dieser Anzahl liegt. Kurven mit drei Eingabepunkten sind vom Grad zwei, sie werden *quadratisch* genannt. Von besonderem Interesse für diese Arbeit sind *kubische* Kurven vom Grad drei, die also vier Eingabepunkte approximieren. Beispielhafte Kurven sind in Abbildung 6 gezeigt. *Bézier*-Kurven haben folgende zentrale Eigenschaften, die für die Verwendung in der Computeranimation interessant sind[12]:

- Die Kontrollpunkte bilden eine *konvexe* Hülle, welche die Kurve nicht verlässt.
- Der erste und der letzte Kontrollpunkt liegen auf der Kurve.

[12] Weitere Eigenschaften sind in [Ben03] dargestellt.

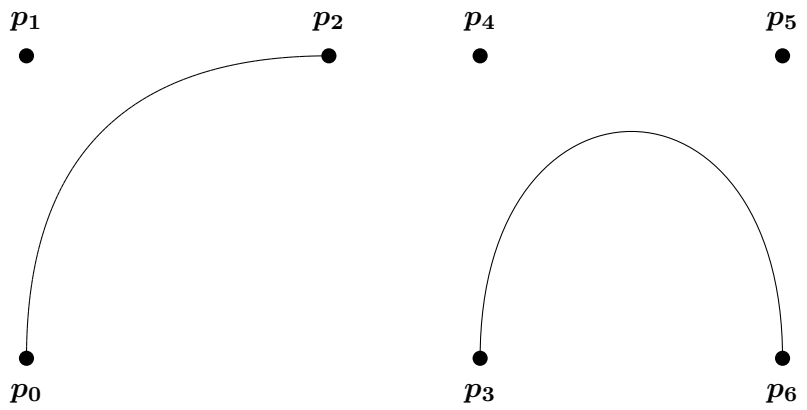

Abbildung 6: Quadratische *Bézier*-Kurve (links) und kubische *Bézier*-Kurve (rechts).

- Die Tangenten der Kurve am Start-/Endpunkt sind durch deren Verbindung mit dem zweiten beziehungsweise vorletzten Punkt gegeben.

Zwei *Bézier*-Kurven können zu einer einzigen stetigen Kurve zusammengesetzt werden, wenn folgende Bedingungen erfüllt sind:

- Der letzte Punkt der ersten Kurve und der erste Punkt der zweiten Kurve sind identisch.
- Die Tangenten am Endpunkt der ersten Kurve und am Startpunkt der zweiten Kurve sind parallel (jedoch nicht unbedingt gleich lang).

Zu *Bézier*-Kurven gibt es eine umfangreiche theoretische Basis. Diese Kurven können mit Hilfe von Basisfunktionen, den sogenannten *Bernstein-Polynomen*, konstruiert werden. Die Anzahl dieser Basisfunktionen, sowie deren Grad, hängt von der Anzahl der verbundenen Punkte ab. Eine ausführlichere Betrachtung der theoretischen Basis ist an dieser Stelle jedoch nicht sinnvoll, denn sie ist für das Verständnis der weiteren Zusammenhänge dieser Arbeit nicht notwendig oder zielführend. Solche Betrachtungen können zum Beispiel in [Ben03] nachgelesen werden. Die Grundidee einer vereinfachten Berechnung von *Bézier*-Kurven basiert auf der wiederholten Hintereinanderausführung der linearen Interpolation. Die Eingabepunkte, die stets in einer festen Reihenfolge vorliegen müssen, werden mit einem fest gewählten Parameter t jeweils paarweise linear interpoliert, um einen neuen Punkt auf deren Verbindungslinie zu erhalten. Die Anzahl der so generierten Punkte ist um eins geringer als die Anzahl der Eingabepunkte. Dies wird nun mit demselben t-Wert solange mit den jeweils neu

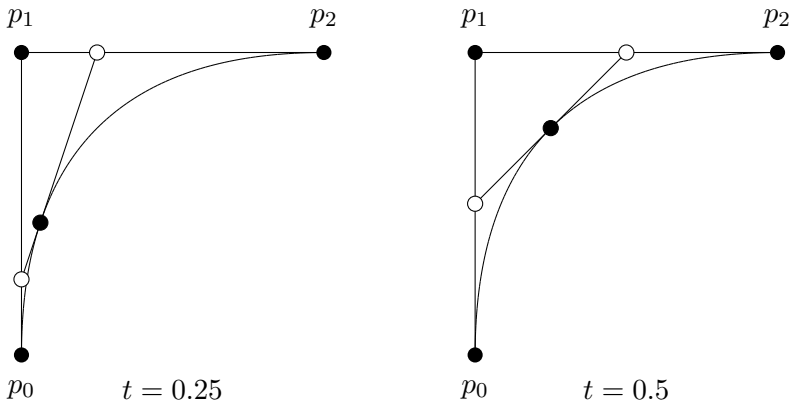

Abbildung 7: Beispiel für die Durchführung des *de Casteljau*-Algorithmus für eine quadratische *Bézier*-Kurve und die Interpolationsparameter $t = 0.25$ und $t = 0.5$.

generierten Punkten wiederholt, bis nur noch ein Punkt erzeugt wird. Dieser stellt den Punkt auf der Kurve für den Parameter t dar. Dieses Verfahren zur Berechnung beziehungsweise Darstellung von *Bézier*-Kurven nennt man den *de Casteljau*-Algorithmus.

Dieser Algorithmus ist für Kurven beliebigen Grads einsetzbar, eine Darstellung in Pseudocode zeigt Codefragment 1. Grafische Beispiele für die Anwendung des *de Casteljau*-Algorithmus zeigt Abbildung 7. Das Prinzip des Algorithmus ist außerdem in Gleichung 2 am Beispiel quadratischer Kurven verdeutlicht, Gleichung 3 zeigt die endgültige Formel. Gleichung 4 gibt die entsprechende Formel für Kurven vom Grad drei wieder.

$$\boldsymbol{p}(t) = (1-t)((1-t)\boldsymbol{p_0} + t\boldsymbol{p_1}) + t((1-t)\boldsymbol{p_1} + t\boldsymbol{p_2}) \tag{2}$$

$$\boldsymbol{p}(t) = (1-t)^2\boldsymbol{p_0} + 2(1-t)t\boldsymbol{p_1} + t^2\boldsymbol{p_2} \tag{3}$$

$$\boldsymbol{p}(t) = (1-t)^3\boldsymbol{p_0} + 3(1-t)^2t\boldsymbol{p_1} + 3(1-t)t^2\boldsymbol{p_2} + t^3\boldsymbol{p_3} \tag{4}$$

Die Beschreibung der *Bézier*-Kurven hat aufgezeigt, das diese sehr intuitiv gestaltbar sind. Allerdings widersprechen sie der Festlegung, die am Anfang dieses Abschnitts getroffen wurde, denn es sollten zunächst Verfahren für die paarweise Interpolation betrachtet werden. Um mit dem approximierenden *Bézier*-Verfahren eine Kurve darzustellen, sind jedoch mindestens drei Kontrollpunkte nötig. *Hermite*-Kurven stellen eine Möglichkeit dar, zwei Punkte ohne zusätzliche Kontrollpunkte durch eine Kurve zu verbinden. Dazu wird dem Start- und Endpunkt jeweils eine *Tangente* hinzugefügt. Diese Tangenten bestimmen dann die Richtung, in der die Kurve

Codefragment 1 : *de Casteljau*-Algorithmus

Parameter : P, geordnete Punkteliste mit n Punkten $\boldsymbol{p_i} = (p_x, p_y)$
Parameter : t, Interpolationsparameter
Rückgabewert : $\boldsymbol{p} = (p_x, p_y)$, der gesuchte Bézier-Punkt

```
Funktion deCasteljau(P,t)
{
    for j = n − 2; n ≥ 0; n-- do
        for k = 0; k ≤ n; k++ do
            p_k = Interpolation(p_k p_{k+1},t);
        end
    end
    Liefere p_0 zurück;
}
```

verläuft. Tangenten wurden bereits bei der Beschreibung der *Bézier*-Kurven erwähnt. Das erste und letzte Punktepaar einer solchen Kurve bestimmen die Tangente der Kurve an Start- und Endpunkt. Tatsächlich ist es zweckmässig, *Hermite*-Kurven mit Hilfe von *Bézier*-Kurven zu beschreiben. Somit können dann bekannte Algorithmen wie der *de Casteljau*-Algorithmus weiterverwendet werden. Die Beschreibung einer *Hermite*-Kurve zum Verbinden der zwei Punkte $\boldsymbol{p_0}$ und $\boldsymbol{p_1}$ besteht neben diesen beiden Punkten aus den Tangenten $\boldsymbol{d_0}$ und $\boldsymbol{d_1}$. Gleichung 5 beschreibt die Umrechnung in vier *Bézier*-Punkte. Dies wird in Abbildung 8 veranschaulicht. Weiterhin verdeutlicht diese Abbildung die Tatsache, das Tangenten von *Hermite*-Kurven in Relation zur Kurve selbst sehr lang werden.

$$\begin{aligned} \boldsymbol{p_0^B} &= \boldsymbol{p_0} \\ \boldsymbol{p_1^B} &= \boldsymbol{p_0} + \frac{1}{3}\boldsymbol{d_0} \\ \boldsymbol{p_2^B} &= \boldsymbol{p_1} - \frac{1}{3}\boldsymbol{d_1} \\ \boldsymbol{p_3^B} &= \boldsymbol{p_1} \end{aligned} \tag{5}$$

Mehr als zwei Punkte lassen sich mit *Hermite*-Kurven leicht interpolieren. Auch hier wird jedem Punkt eine Tangente zugewiesen, um danach aufeinanderfolgende Punkte jeweils paarweise zu interpolieren. Da die Tangenten dabei dieselben bleiben, wird so eine stetige Kurve erzeugt[13] (Abbildung 9). Einen Sonderfall solcher *Hermite*-Kurven mit mehr als zwei Punkten sind die *Catmull-Rom*-Kurven. Die Tangente für die Interpolation wird hierbei automatisch aus der Position des vorherigen und nachfolgenden Punkt errechnet:

[13] Da dies die Bedingung für das stetige Zusammensetzen von *Bézier*-Kurven erfüllt.

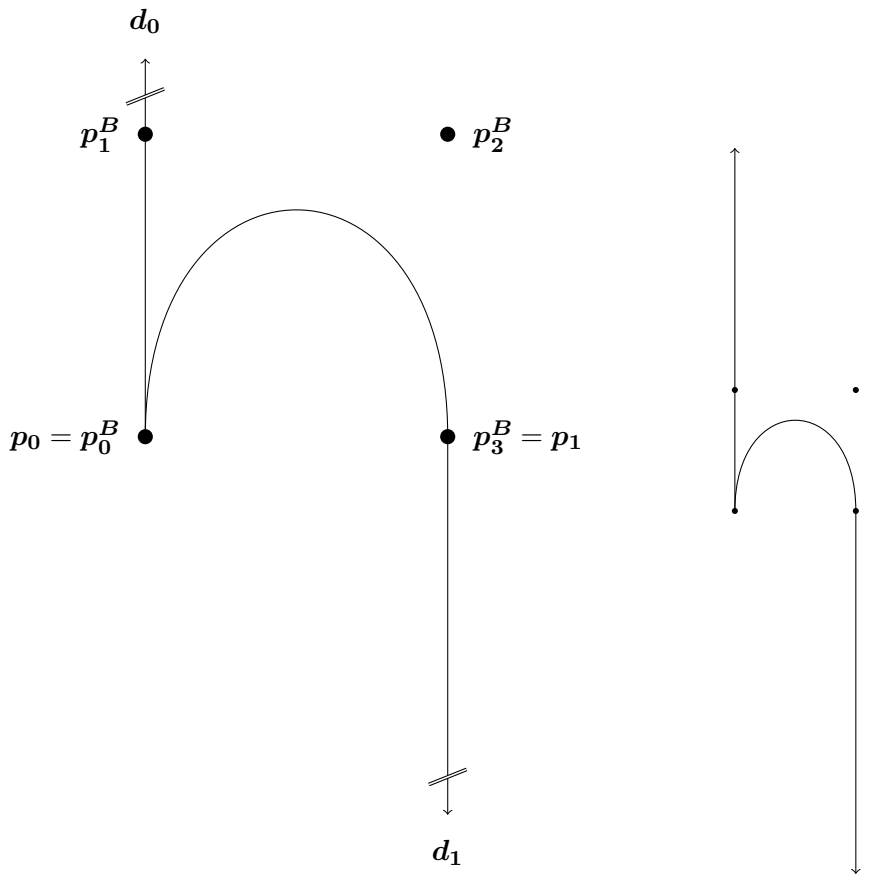

Abbildung 8: *Hermite*-Kurve, dargestellt mit den entsprechenden *Bézier*-Punkten. Diese Kurve entspricht der kubischen *Bézier*-Kurve aus Abbildung 6. Die linke Darstellung zeigt die Kurve mit verkürzten Tangenten, die rechte Darstellung mit korrekten Größenverhältnissen. Man sieht deutlich, das *Hermite*-Tangenten sehr lang werden.

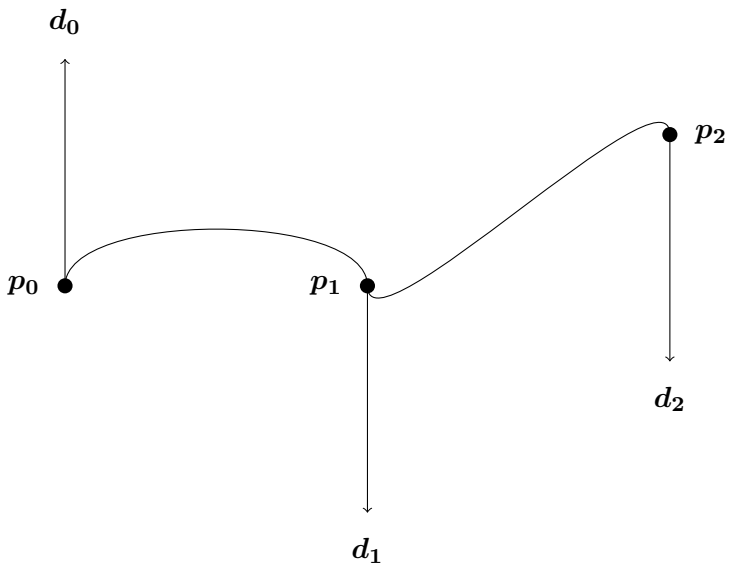

Abbildung 9: *Hermite*-Kurve mit drei Kontrollpunkten.

$$d_i = \frac{p_{i+1} - p_{i-1}}{2} \tag{6}$$

Die Randpunkte müssen anders behandelt werden. *Catmull-Rom*-Kurven eignen sich für die automatische Interpolation von Punkten, in der eine manuelle Kontrolle nicht notwendig oder möglich ist. Ein Beispiel zeigt Abbildung 10.

Neben auf *Bézier*-Kurven basierenden Darstellungen gibt es weitere Techniken, Punkte durch Kurven zu verbinden oder anzunähern. Diese bieten oft mächtigere Darstellungsmöglichkeiten. Auf eine genauere Darstellung solcher Techniken, wie zum Beispiel *B-Splines* oder NURBS, wird an dieser Stelle verzichtet. Die Mächtigkeit von *Hermite*-Kurven reicht für die in dieser Arbeit wichtigen Problemstellungen der Computeranimation aus, so dass deren einfacherer Bedienbarkeit der Vorzug gegenüber den komplexeren Verfahren gegeben wird.

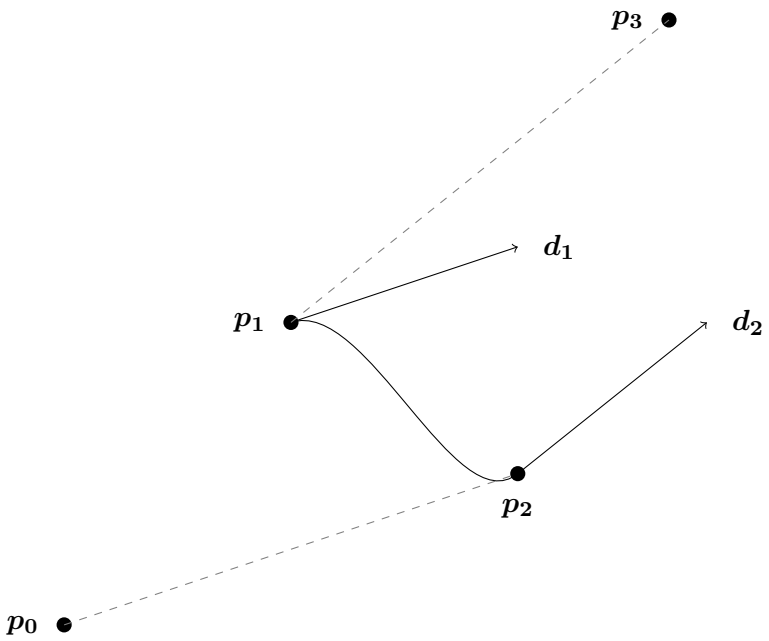

Abbildung 10: *Catmull-Rom*-Kurve mit vier Kontrollpunkten.

4 Technische Grundlagen

4.1 Beschreibung des Grundproblems

Das Grundproblem der Computeranimation ist es, Objekte zu bewegen. Sie beschäftigt sich mit der Bewegung dieser Gegenstände über die Zeit, oder genauer mit der Veränderung von Objekteigenschaften über die Zeit. Diese Objekte haben verschiedene *Eigenschaften*, deren Werte sich über die Zeit ändern können. Neben den bereits in Abschnitt 3.1 genannten Eigenschaften der *Position* und *Orientierung* gibt es eine Vielzahl weiterer Eigenschaften, die sich zum animieren eignen. Im Kontext der Computeranimation wird die Position meist *Translation* genannt, die Orientierung meist *Rotation*. Weitere mögliche Objekteigenschaften sind z.B. die Größe beziehungsweise *Skalierung* des Objekts oder dessen*Transparenz*. Doch auch andere Eigenschaften wie *Texturparameter* oder *Farbinformationen* sind möglich. Generell lässt sich jede Eigenschaft animieren, deren Werte zu verschiedenen Zeitpunkten festgehalten werden kann, und bei der zwischen diesen Werten eine Interpolation möglich ist.

Um die Animation zu realisieren wird eine *Zeitleiste* eingeführt. Diese wird dabei nicht in einer Zeiteinheit wie Sekunden oder Minuten unterteilt, sondern in *Einzelbilder* beziehungsweise *Frames*. Es hängt von der am Ende gewählten Abspielgeschwindigkeit ab, wieviele dieser Einzelbilder pro Sekunde zu sehen sind. Jede Eigenschaft eines Objekts kann zu jedem so festgelegten Zeitpunkt einen anderen Wert annehmen. Sind die Zeitwerte zweier festgelegter Werte nicht benachbart, so berechnet der Computer automatisch die Zwischenwerte. Er interpoliert zwischen den festgelegten Werten. Eigenschaften können *skalar* (zum Beispiel bei Transparenz) oder vektoriell (zum Beispiel bei Translation) sein. Das Festlegen der Werte zu bestimmten Zeitpunkten, und die Beeinflussung der Zwischenwertberechnung sind die grundlegenden Eingabemöglichkeiten, die dem Computeranimationskünstler zur Verfügung stehen.

Im Folgenden sollen die grundlegenden Anforderungen an Ansätze zur Wertfestlegung und Interpolation vorgestellt werden. Danach werden verschiedene Ideen zur Umsetzung angesprochen.

4.2 Anforderungen

Für die traditionelle Produktion von Animationssequenzen haben sich mit der Zeit grundlegende Prinzipien herausgebildet, deren Einhaltung für realistisches und erfolgreiches Animieren von zentraler Bedeutung sind. [Las87] zeigt am Beispiel früher PIXAR-Animationskurzfilme, das diese Grundprinzipien ihre Bedeutung auch im Kontext von 3D-Animation behalten. Einige dieser Prinzipien beschäftigen sich mit der realistischen Produktion von Bildsequenzen. Der Betrachter erwartet, das sich Bewegungen so verhal-

ten beziehungsweise "anfühlen", wie er es aus dem echten Leben gewohnt ist. Andere Prinzipien beschäftigen sich mit Elementen erfolgreicher Umsetzung der Handlung in den Einzelbildern, oder mit der Herangehensweise bei Erstellung der Einzelbilder eines Bewegungsablaufs. Der Aufbau computerbasierter Animationswerkzeuge wird von diesen Grundprinzipien beeinflusst. Sie werden nun kurz vorgestellt:

1. *Squash and Stretch*: Bedeutet so viel wie "Stauchen und Strecken" und weist darauf hin, das sich Objekte bei Bewegung verformen. So bekommt man bei der Betrachtung ein Gefühl für das Gewicht und die Beschaffenheit des Gegenstands.

2. *Timing*: Die "Geschwindigkeit" einer Bewegung hat großen Einfluss darauf, was sie dem Zuschauer vermittelt. So kann zum Beispiel eine Kopfdrehung in langsamer Geschwindigkeit als Umsehen, bei schneller Geschwindigkeit als Geste der Verneinung wirken. Weiterhin vermittelt die Bewegungsgeschwindigkeit, und vor allem deren Änderung, eine Vorstellung von Größe und Gewicht eines Objekt. Schwere Objekte brauchen länger, um ihre Geschwindigkeit zu ändern.

3. *Anticipation*: Schaffung einer "Erwartung" wichtiger Abläufe in der Handlung beim Betrachter. Dies wird erreicht durch geeignete Bewegungen in der Szene. So kann zum Beispiel ein Sprung eines Charakters durch übertriebenes Zurücklehnen zuvor angekündigt werden.

4. *Staging*: Die klare "Inszenierung" einer Idee oder Handlung. Die Aufmerksamkeit des Betrachters muss an diejenige Stelle gelenkt werden, an der die zentralen Elemente der Szene stattfinden.

5. *Follow Through and Overlapping Action*: Bewegungen sind meist nicht unabhängig voneinander, sondern "folgen und überschneiden einander". So wird zum Beispiel eine Hand, die einen Ball wirft, die Wurfbewegung noch weiter fortsetzen, nachdem der Ball die Hand schon verlassen hat.

6. *Straight Ahead Action and Pose-To-Pose Action*: Dieses Prinzip beschäftigt sich mit der Organisation der Erstellung der Einzelbilder. Dies kann geschehen, in dem die Einzelbilder wichtiger, definierender Posen zuerst erstellt, und die fehlenden Zwischenbilder später erstellt werden (*Pose-To-Pose*, siehe Abbildung 12). Andererseits kann man man auch alle Einzelbilder in sequenzieller Reihenfolge erstellen (*Straight Ahead*). Für spontane Animationssequenzen, deren Ausgang nicht unbedingt klar feststehen muss, eignet sich der Ansatz, alles in Reihenfolge zu erstellen. Bei klar definierter Handlung wird meist der Ansatz gewählt, erst wichtige Schlüsselbilder zu erstellen, und deren zeitlichen Ablauf zu definieren.

7. *Slow In and Out*: Beginn und Abschluss von Bewegungen sind mit "Beschleunigung und Abbremsung" verbunden.

8. *Arcs*: Die meisten Bewegungen in der Realität beschreiben im Verlauf "Kurven". Lineare und direkte Bewegungen wirken unnatürlich. Die Hüfte eines bewegten Menschen zum Beispiel ändert beim Laufen die Höhe, und beschreibt somit eine bogenförmige Vorwärtsbewegung (siehe Abbildung 11).

9. *Exaggeration*: "Übertreibungen" können eingesetzt werden, um die hinter einer Handlung stehende Idee unmissverständlich klar zu machen.

10. *Secondary Action*: Sich bewegende Objekte können "sekundäre Bewegungen" verursachen, insbesondere bei heftigen und unkontrollierten Bewegungsabläufen. Andere Objekte werden so in Bewegung gesetzt.

11. *Appeal*: Bedeutet so viel wie "Anziehung". Bei der Gestaltung einer Animationssequenz sollte bedacht werden, dass diese dem Betrachter am Ende auch gefallen muss. Dies kann durch ansprechende Charaktere, überzeugende Handlung und vielerlei andere Möglichkeiten erreicht werden.

Einen guten Überblick über diese Grundprinzipien bietet [Ben03]. Der Computer muss für alle Teile der *Toolchain* Werkzeuge bereitstellen, welche den Benutzer unterstützen, diese Prinzipien zu befolgen. Die Werkzeuge müssen im Hinblick darauf genug Funktionalität bereitstellen. Einige der genannten Prinzipien beeinflussen insbesondere den Animationsteil der Werkzeugkette. Die Stärke computerbasierter Animationswerkzeuge ist die automatische Erstellung der Zwischenbilder zu Schlüsselpositionen, so dass hier meist der *Pose-To-Pose*-Ansatz verfolgt wird (siehe Abbildung 12). Daher werden Schlüsselwerte der Objekteigenschaften zu bestimmten Zeitpunkten festgelegt, und die Zwischenwerte interpoliert. Mit Bezug auf die in Abschnitt 4.1 eingeführte Unterteilung der Zeitleiste in einzelne *Frames* werden Schlüsselpositionen *Keyframes* genannt. Diese Zwischenwertsberechnung muss berücksichtigen, dass natürliche Bewegung Kurven folgt (*Arcs*, siehe Abbildung 11), und dass ihre Geschwindigkeit nicht uniform ist (*Slow In and Out*). Die Zeitpunkte der Schlüsselwerte müssen flexibel anpassbar sein (*Timing*). Um Objekte auch verformen zu können (*Squash and Stretch*), müssen entsprechende Objekteigenschaften animierbar sein (wie zum Beispiel die Größe oder *Skalierung*).

Neben den hier genannten, zentralen Anforderungen gibt es natürlich eine Vielzahl weiterer Anforderungen an Computerwerkzeuge für die Animation. So muss ein gutes Animationsprogramm zum Beispiel problemlos mit den anderen Werkzeugen der Produktionskette zusammenarbeiten,

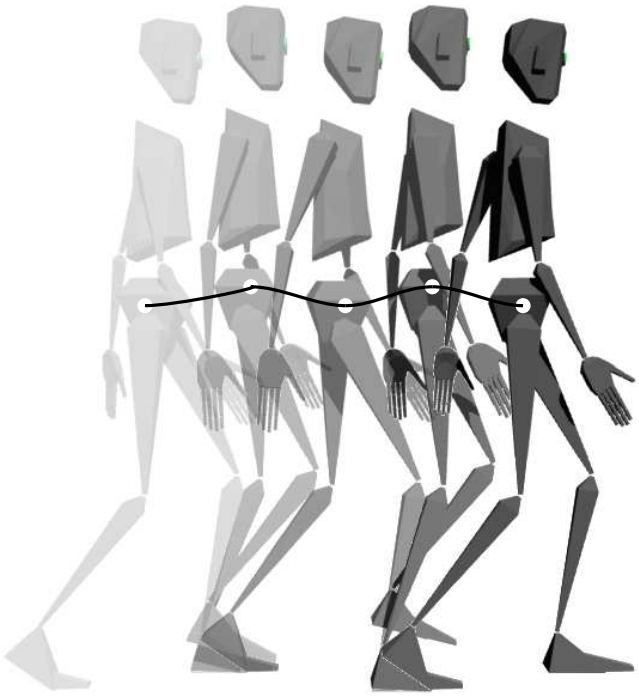

Abbildung 11: Natürliche Bewegungen beschreiben meist Kurven. Dies sollte auch bei animierten Sequenzen berücksichtigt werden. Hier abgebildet sind Einzelschritte der Animation eines gehenden Roboters. Die Vorwärtsbewegung der Hüfte folgt dabei einer Kurve.

stabil sein und mit der Anzahl der bewegten Objekte gut skalieren. Diese Anforderungen sind jedoch nicht zentral für das Problem der Animation, denn ihre Notwendigkeit ist nicht spezifisch für diese Problemstellung.

4.3 Lösungsansätze

Ein grundlegendes Problem der Computeranimation ist es, den *Bewegungspfad* eines Objekts im Raum festzulegen. Es soll möglich sein, auf diesem Pfad das *Weg-Zeit-Verhalten* zu bestimmen. Es geht also darum, den Zeitpunkt zu bestimmen, wann ein Objekt an welcher Stelle eines vorher festgelegten Pfads zu finden ist. Dieses Grundproblem ist leicht erweiterbar auf andere Eigenschaften eines Objekts. Auch die Orientierung oder Farbe eines Objekts sind Werte, die man als Punkte in einem Raum auffassen kann. Eine zeitliche Veränderung dieser Eigenschaften soll ebenso möglich sein. Es werden nun zwei Ansätze vorgestellt, diese Problemstellung zu lösen.

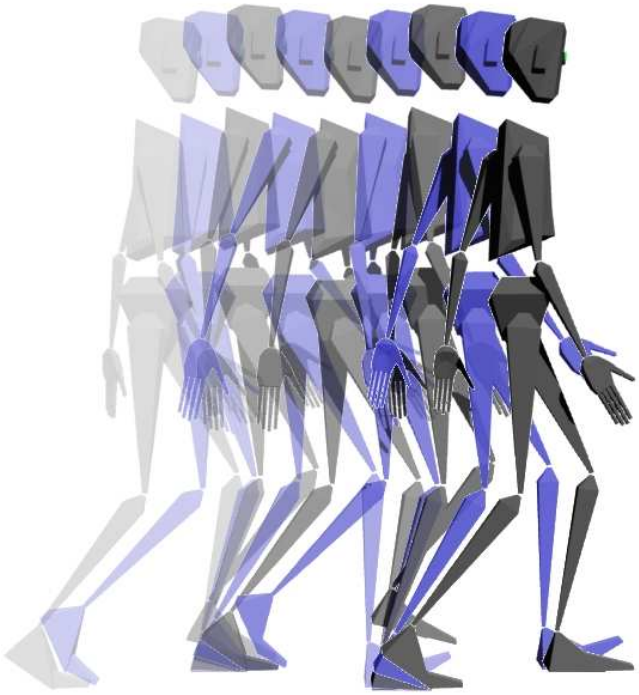

Abbildung 12: Nachdem die Einzelbilder zu Schlüsselpositionen festgelegt wurden, berechnet der Computer die Einzelbilder zu den Zwischenschritten. Beispielhafte Zwischenbilder zur Animationssequenz aus Abbildung 11 sind hier blau dargestellt.

4.3.1 Expliziter Ansatz

Dieser Ansatz trennt die Modellierung des Interpolationspfads zwischen zwei Punkten explizit von der Festlegung des Weg-Zeit-Verhaltens auf diesem Pfad. Er ist besonders anschaulich anhand der 3D-Position eines Objekts zu beschreiben, und ist in gleicher Weise für andere Werte umsetzbar. Diese Werte können sowohl zwei- als auch dreidimensional sein.

Der Bewegungspfad an sich wird hierbei komplett zeitunabhängig festgelegt. Hierzu wird oft die *Hermite*-Interpolation verwendet. Die Schlüsselpunkte werden festgehalten, und ihnen wird eine Tangente zugeordnet. Der Kurvenverlauf wird mit Hilfe dieser Tangenten gesteuert. Schlüsselpunkte und Tangenten werden in einer geeigneten Datenstruktur gespeichert. Abbildung 9 zeigt ein Beispiel für den zweidimensionalen Fall. Abbildung 13 zeigt ein Beispiel für die Interpolation von 3D-Positionsinformationen. Mit Hilfe des Interpolationsparameters t der *Hermite*-Kurve kann nun jede Position auf der Kurve berechnet werden. Im einfachsten Fall kann dieser Parameter als Zeitparameter verwendet werden. Zwischen jeweils zwei Punkten wäre die Zeit somit auf den Bereich $[0, 1]$ normiert. Punkte, deren Parameter t denselben Abstand haben, liegen bei *Bézier*-basierten Kurven jedoch nicht immer gleichweit voneinander entfernt (siehe Abbildung 14). Eine konstante Erhöhung von t führt somit nicht zu einer konstanten Bewegungsgeschwindigkeit auf der Kurve. Zudem kann die Kurvenlänge zwischen verschiedenen Schlüsselpunkten sehr unterschiedlich sein, so dass sich bei jeweils konstanter Erhöhung des Interpolationsparameters die Geschwindigkeit der Interpolation stark ändern kann; je länger die Kurve, desto höher ist die Durchschnittsgeschwindigkeit.

Um eine flexible Kontrolle der Geschwindigkeit entlang der Kurve zu ermöglichen, ist eine äquidistante Parametrisierung der Interpolationskurve notwendig, da die wahrgenommene Geschwindigkeit dem zurückgelegten Weg pro Zeiteinheit entspricht. Es muss also möglich sein, die Entfernung zwischen zwei Punkten *entlang der Kurve* zu ermitteln. Dies ermöglicht die *Bogenlängentabelle* (engl. *Arc length table*), in der für verschiedene Werte des Kurvenparameters t die Entfernung gespeichert wird, die entlang der Kurve zwischen dem zu t gehörenden Punkt und dem Anfangspunkt der Kurve liegt. Die in der Tabelle gespeicherten t-Werte sind dabei äquidistant gewählt. Tabelle 1 zeigt eine Bogenlängentabelle mit zwanzig Werten der in Abbildung 14 gezeigten Kurve. Eine Tabelle für reale Anwendung würde wesentlich mehr Werte enthalten. Der einfachste Weg zur Berechnung einer Bogenlängentabelle ist es, die linearen Distanzen der den jeweiligen Parameterwerten t entsprechenden Punkten aufzuaddieren. Auch für Kurven, die aus mehreren *Hermite*-Teilkurven bestehen, kann eine Bogenlängentabelle erstellt werden. In dieser Tabelle wird dann zusätzlich die Nummer der Kurve aufgeführt, auf die sich der Parameter t in der aktuellen Tabellenzeile bezieht. Die Bogenlängenwerte nehmen dabei stetig zu,

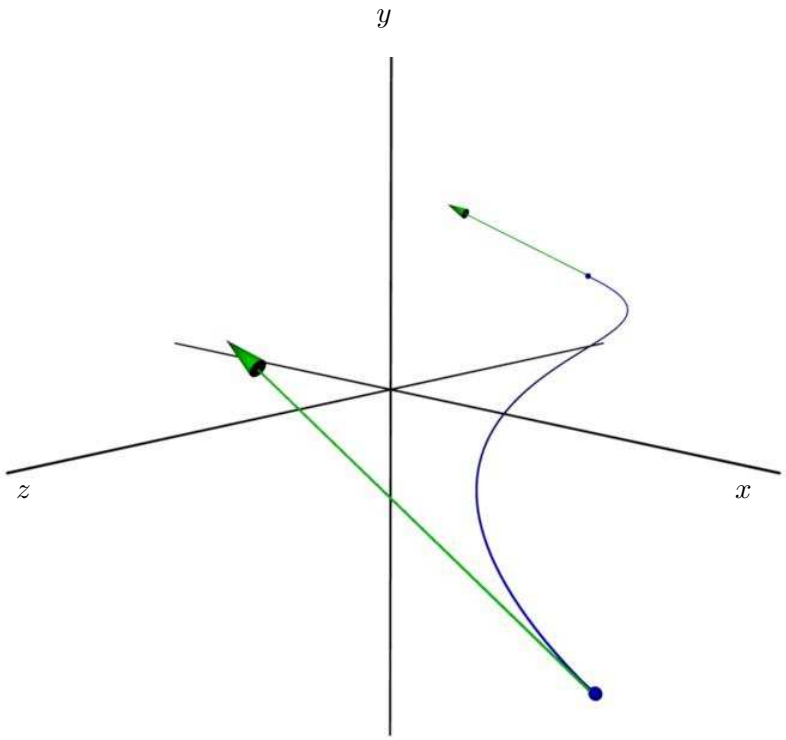

Abbildung 13: Ein Beispiel für *Hermite*-Interpolation von Positionsinformationen im dreidimensionalen Raum. Punkte und Kurven sind in blau, Tangenten in grün dargestellt.

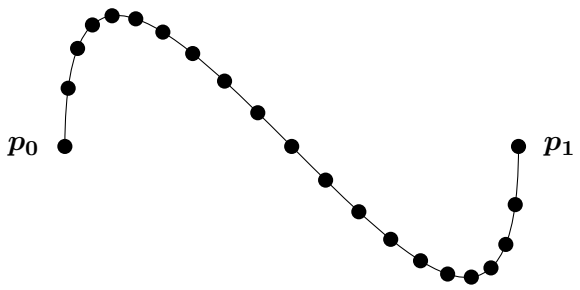

Abbildung 14: Bei einer *Hermite*-Kurve liefert konstante Erhöhung des Interpolationsparameters t Punkte, die nicht gleichweit voneinander entfernt sind.

t	Kurven-Nr.	Bogenlänge	t	Kurven-Nr.	Bogenlänge
0.00	0	0.00	0.55	0	5.60
0.05	0	0.77	0.60	0	6.20
0.10	0	1.31	0.65	0	6.76
0.15	0	1.68	0.70	0	7.25
0.20	0	1.97	0.75	0	7.64
0.25	0	2.28	0.80	0	7.96
0.30	0	2.68	0.85	0	8.25
0.35	0	3.17	0.90	0	8.61
0.40	0	3.72	0.95	0	9.16
0.45	0	4.33	1.00	0	9.93
0.50	0	4.96			

Tabelle 1: Bogenlängentabelle für die in Abbildung 14 gezeigten Punkte und Kurve.

bei einer neuen Teilkurve wird der letzte Bogenlängenwert der vorherigen Kurve weiterverwendet. In der *normalisierten Bogenlängentabelle* werden die Bogenlängenwerte und die Parameter t Einzelkurven-übergreifend auf den Bereich $[0, 1]$ normalisiert. Tabelle 2 zeigt eine normalisierte Bogenlängentabelle und die zugehörigen Kurven.

Eine Bogenlängentabelle ermöglicht es, zu einem beliebig gewählten Punkt mit einer bestimmten Entfernung vom Startpunkt (entlang der Kurve) den zugehörigen Kurvenparameter t zu schätzen. Dazu werden in der Tabelle diejenigen angrenzenden Zeilen gesucht, deren Entfernungswert die gesuchte Entfernung einschließen. Es wird berechnet, wo auf der Strecke zwischen diesen beiden Entfernungswerten die gesuchte Entfernung genau liegt. Es wird also der Parameter der linearen Interpolation zwischen den beiden Entfernungen der Tabelleneinträge berechnet, der den gesuchten Entfernungswert erzeugt. Mit diesem Wert werden die in der Tabelle eingetragenen Kurvenparameter t interpoliert, um den t-Wert für den gesuchten Punkt zu ermitteln.

Das zeitliche Bewegungsverhalten (also das Geschwindigkeits- und Beschleunigungsverhalten) wird beim expliziten Ansatz durch eine separate Kurve festgelegt. Die Zeit wird dazu auf den Bereich $[0, 1]$ normiert. Dieser Zeitbereich spannt mit dem ebenfalls auf $[0, 1]$ normierten Wertebereich der Bogenlänge einen zweidimensionalen Bereich auf, in dem mit einem Graphen für jeden Zeitpunkt ein zugehöriger Entfernungswert festgelegt wird. So kann zu jedem Zeitpunkt einfach bestimmt werden, an welcher Stelle der Kurve man sich gerade befindet. Dieser *Weg-Zeit-Graph* kann also durch eine Funktion beschrieben werden, mit den Zeitwerten als Eingabe, und den Bogenlängenwerten als Ausgabe. Geeignet ist jede mathematische Abbildung, die den Bereich $[0, 1]$ auf den Bereich $[0, 1]$ abbildet

t	K.-N.	B.
0.000	0	0.00
0.125	0	1.41
0.250	0	2.59
0.375	0	3.60
0.500	0	4.45
0.625	0	5.14
0.750	0	5.68
0.875	0	6.17
1.000	0	6.84
0.125	1	7.35
0.250	1	7.69
0.375	1	8.38
0.500	1	9.32
0.625	1	10.35
0.750	1	11.31
0.875	1	12.12
1.000	1	12.85
0.125	2	13.51
0.250	2	14.25
0.375	2	15.18
0.500	2	16.23
0.625	2	17.34
0.750	2	18.46
0.875	2	19.62
1.000	2	20.98

t	K.-N.	B.
0.000	0	0.00
0.042	0	0.07
0.083	0	0.12
0.125	0	0.17
0.167	0	0.21
0.208	0	0.24
0.250	0	0.27
0.292	0	0.29
0.333	0	0.33
0.375	1	0.35
0.417	1	0.37
0.458	1	0.40
0.500	1	0.44
0.542	1	0.49
0.583	1	0.54
0.625	1	0.58
0.667	1	0.61
0.708	2	0.64
0.750	2	0.68
0.792	2	0.72
0.833	2	0.77
0.875	2	0.83
0.917	2	0.88
0.958	2	0.94
1.000	2	1.00

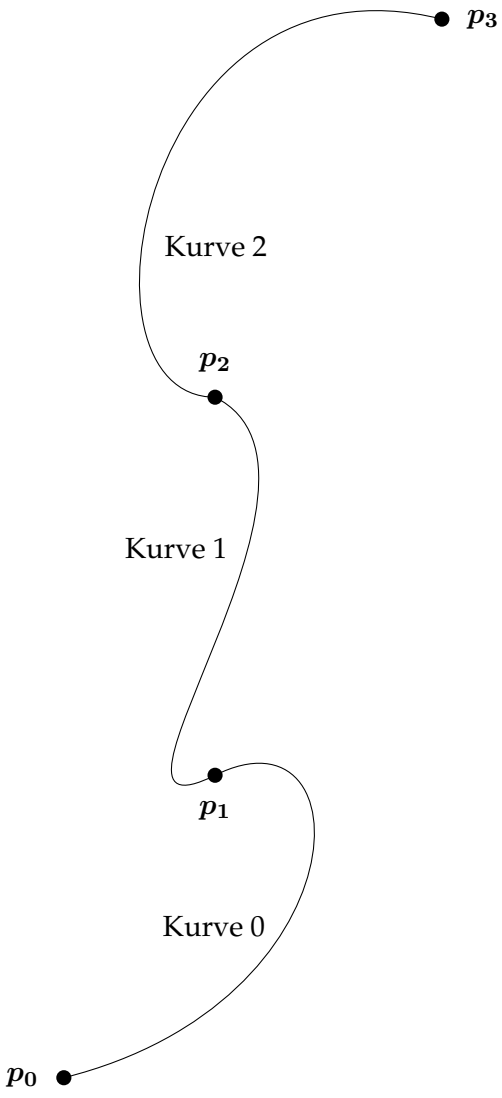

Tabelle 2: Eine Kurve aus mehreren *Bézier*-Einzelkurven mit zugehöriger Bogenlängentabelle. Es sind sowohl die nichtnormalisierte (links) als auch die normalisierte Tabelle (rechts) abgebildet.

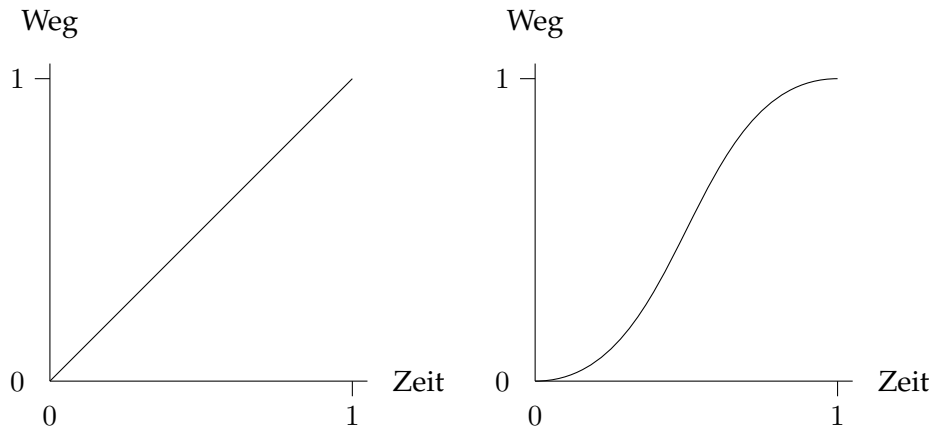

Abbildung 15: Zwei Weg-Zeit-Graphen. Links dargestellt ist ein Graph, welcher eine Bewegung mit konstanter Geschwindigkeit bewirkt. Rechts dargestellt ist ein Graph mit Beschleunigung der Geschwindigkeit am Anfang, und Abbremsen der Geschwindigkeit am Ende (*Ease-in / Ease-out*).

und dabei das Funktionskriterium erfüllt, zu jedem Eingabewert nur einen Ausgabewert zu haben. Mit dem Festlegen des Weg-Zeit-Verhaltens lassen sich Effekte wie das Beschleunigen und Abbremsen am Anfang/Ende einer Bewegung realisieren (*Slow In and Out*). Dieses Geschwindigkeitsverhalten wird auch *Ease-in/Ease-out* genannt (bedeutet soviel wie "behaglich hinein / behaglich hinaus"). Abbildung 15 zeigt zwei Graphen, einen für konstante Geschwindigkeit, und einen für den *Ease-in/Ease-out*-Effekt. Mit den Weg-Zeit-Graphen können viele verschiedene Verhalten realisiert werden. Zusammenhänge und Eigenschaften dieser Graphen werden im Detail in [Par02] dargestellt, und sollen hier nicht weiter vertieft werden.

Mit Hilfe der Pfadfestlegung über *Hermite*-Kurven und der Definition des Geschwindigkeitsverhaltens über Weg-Zeit-Graphen kann der Benutzer also das Bewegungsverhalten eines Objekts gestalten. Dieser Ansatz der expliziten Trennung von Weg und Geschwindigkeit hat den Vorteil, das Veränderungen am Pfad das Geschwindigkeitsverhalten nicht beeinflussen, und umgekehrt der Pfad immer derselbe bleibt, auch wenn der Weg-Zeit-Graph verändert wird. Der Ansatz ist außerdem für die Eigenschaft der Position intuitiv sehr einsichtig und visuell gut erfassbar. Schlüsselpunkte und Tangenten können in einem Editor direkt im dreidimensionalen Raum festgelegt werden, für den Weg-Zeit-Ablauf kann ein eigener Bereich zur Darstellung des Graphen genutzt werden. Obwohl dieser Ansatz neben der Position für andere Eigenschaften ebenso anwendbar ist, sind die Zusammenhänge für Eigenschaften wie Orientierung oder Farbe weniger intuitiv einleuchtend. So kann auch das Festlegen einer dreidimensio-

nalen *Cardan*-Rotation oder RGB-Farbe (Farbdarstellung mit den Kanälen Rot, Grün und Blau) durch Festlegung von Punkten in einem dreidimensionalen Raum realisiert werden. Diese Darstellung ist jedoch für solche Eigenschaften weniger intuitiv, da kein unmittelbarer Zusammenhang zwischen der im Editor festgelegten Kurve und dem visuellen Ergebnis der Objektdarstellung besteht. Daher wird in Systemen, die den expliziten Ansatz verfolgen, oft die Rotation nicht selbst festgelegt, sondern zu jedem Zeitpunkt aus den Positionsinformationen neu berechnet. Solche Berechnungen nutzen Eigenschaften der Positionskurve (zum Beispiel Ableitungen) oder zusätzliche Informationen (wie zum Beispiel eine weitere Kurve von Interessenspunkten, mit denen eine Blickrichtung errechnet werden kann), um die Rotation zu berechnen[14]. So wie hier beschrieben lässt sich der explizite Ansatz nur auf Eigenschaften anwenden, deren Wert aus zwei- oder mehrdimensionalen Punkten besteht. Denn ein Pfad lässt sich zwischen einfachen, eindimensionalen Werten nicht aufspannen. Jede Interpolation zwischen zwei skalaren Werten ist eine lineare Interpolation, für die die Festlegung eines Geschwindigkeitsverhalten jedoch vorstellbar ist. Skalare Eigenschaften müssen also gesondert behandelt werden.

4.3.2 Impliziter Ansatz

Der implizite Ansatz modelliert die Zeit nicht durch einen vom Bewegungspfad unabhängigen Weg-Zeit-Graphen. Der grundlegende Ansatz besteht darin, für jeden skalaren Eigenschaftswert einen *Wert-Zeit-Graphen* zu erstellen. Mehrdimensionale Eigenschaften, wie zum Beispiel Position oder Orientierung, werden komponentenweise behandelt. Der x-Wert der Position wird also in einem eigenen *Wert-Zeit-Graphen* behandelt, ebenso wie der y-Wert und der z-Wert. Die Komponenten mehrdimensionaler Eigenschaften werden also komplett unabhängig behandelt. In dem aus Zeit und Wert aufgespannten zweidimensonalen Raum können nun Schlüsselpunkte festgelegt werden, indem zu einem bestimmten Zeitpunkt ein bestimmter Wert festgehalten wird. Zwischen diesen Werten wird nun mit Hilfe einer *Hermite*-Kurve interpoliert, zu jedem Schlüsselpunkt existiert also eine zusätzliche Tangente, mit der der Verlauf der Interpolationskurve gesteuert wird. Da die Zeit bereits in dem so entstandenen Graphen berücksichtigt ist, ist eine weitere, separate Behandlung der Zeit nicht notwendig. Somit ist auch die Aufstellung einer Bogenlängentabelle unnötig, denn eine Koordinatenachse des Wert-Zeit-Graphen entspricht ja der Zeitachse. Um den Wert der Eigenschaft beziehungsweise Eigenschaftskomponente zu einem bestimmten Zeitpunkt zu ermitteln, muss lediglich der Schnittpunkt der Kurve mit der Geraden bestimmt werden, die parallel zur Werteachse ist und auf der Zeitachse den Wert des gesuchten Zeitpunkts hat (sie-

[14] Weitere Informationen dazu können ebenfalls in [Par02] gefunden werden

he Abbildung 16). Damit auf der Kurve nur ein solcher Schnittpunkt existiert, dürfen die Tangenten nie auf einen Punkt zeigen, der einen kleineren Wert auf der Zeitachse hat (siehe Abbildung 17). Auch mit dem impliziten Ansatz lassen sich vielfältige Verhaltensweisen für die Bewegung realisieren. Abbildung 18 zeigt einen beispielhaften Graph für ein *Ease-in/Ease-out*-Verhalten. Um zum Beispiel die Position mit einem solchen Verhalten zu animieren, müssen alle drei Wert-Zeit-Graphen der Positionskomponenten konsistent gestaltet werden.

Das implizite Verfahren zur Festlegung von Interpolationspfad und Geschwindigkeitsverhalten ist flexibel einsetzbar. Der Benutzer kann zu jeder skalaren Komponente einer Eigenschaft einen eigenen Graphen erstellen, somit sind alle Eigenschaften im Animationsvorgang gleich behandelbar. Der Ansatz hat jedoch den Nachteil, das eine Veränderung im Geschwindigkeitsverhalten immer auch eine Veränderung des Interpolationspfads bedeutet, was zu ungewollten Veränderungen in der dargestellten Szene führen kann. So kann ein Objekt, welches zuvor einen kollisionsfreien Weg zwischen anderen Objekten genommen hat, plötzlich mit einem anderen Gegenstand kollidieren. Auch müssen für konsistentes Verhalten mehrdimensionaler Eigenschaften wie Position oder Orientierung mehrere Graphen einheitlich gestaltet werden. Dies bedeutet hohen manuellen Vergleichsaufwand.

4.3.3 Schlussfolgerung

Es wurden zwei Ansätze vorgestellt, Eigenschaftswerte von Objekten festzulegen und zu interpolieren. Der explizite Ansatz trennt Weg- und Geschwindigkeitsfestlegung voneinander. Er ist für die Positionsfestlegung sehr intuitiv, da der Weg direkt bei der Festlegung visualisiert werden kann. Veränderungen am Geschwindigkeitsverhalten beeinflussen den genommenen Weg nicht. Der Ansatz ist jedoch nur bedingt auf beliebige Eigenschaften übertragbar. Der implizite Ansatz legt Weg- und Geschwindigkeit durch gleichartige Kurven fest. Er bietet den Vorteil, für alle Objekteigenschaften gleich geeignet zu sein. Auf der anderen Seite können Weg- und Geschwindigkeitsverhalten immer nur zusammen festgelegt werden, jede Veränderung des einen Aspekts beeinflusst auch den anderen Aspekt. Der Ansatz selbst funktioniert für alle Eigenschaften gleich, und geht daher auf die speziellen Eigenheiten jeder Eigenschaft nicht ein. Eine intuitive Festlegung von Positionen im dreidimensionalen Editor zum Beispiel wird nicht verwendet. Dies kann jedoch durch geeignete Editor-Werkzeuge kompensiert werden.

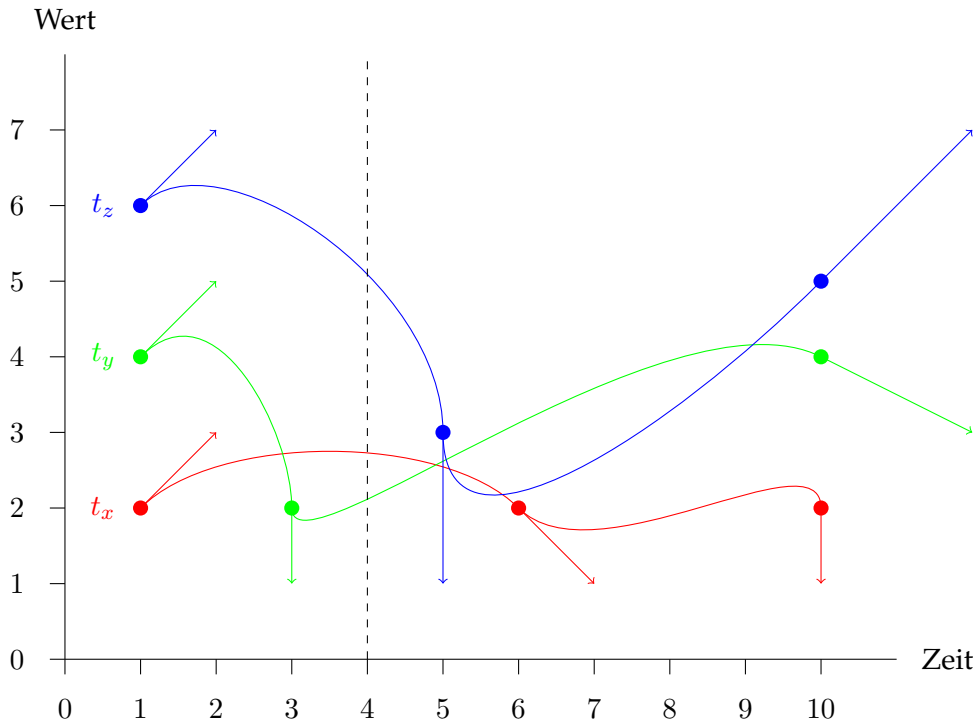

Abbildung 16: Wert-Zeit-Diagramm für die Komponenten t_x (rot), t_y (grün) und t_z (blau) eines Positionsvektors $\boldsymbol{t}$. Es sind die Schlüsselpunkte und die Tangenten der Hermite-Kurven gekennzeichnet. Die Tangenten sind jedoch auf ein Drittel verkürzt dargestellt. Die Komponentenwerte zum Zeitpunkt 4 sind die Schnittpunkte der Kurven mit der eingezeichneten gestrichelten Linie.

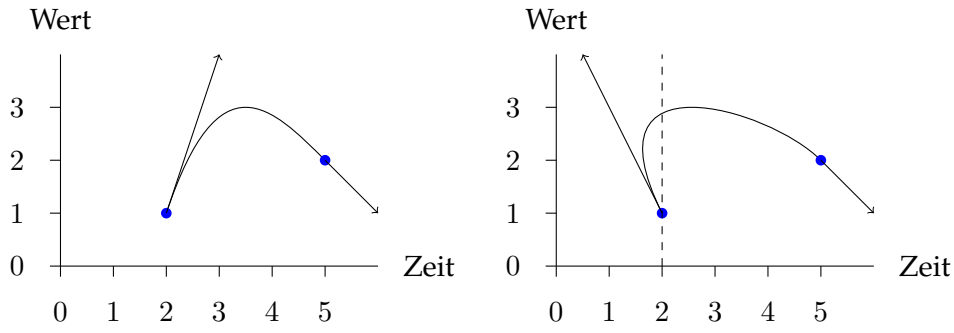

Abbildung 17: Zwei Wert-Zeit-Graphen. Der Rechte ist ungültig, da für einen Zeitpunkt zwei mögliche Werte existieren. Die Tangenten der *Hermite*-Kurven sind auf ein Drittel verkürzt dargestellt.

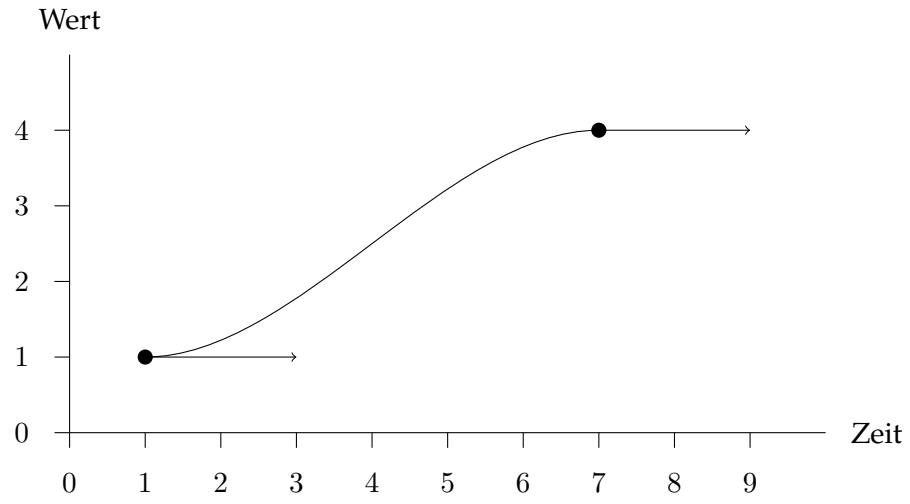

Abbildung 18: Wert-Zeit-Graph, welcher *Ease-in/Ease-out*-Verhalten realisiert. Die Tangenten der *Hermite*-Kurve sind auf ein Drittel verkürtzt dargestellt.

5 Realisierung eines Animationseditors

5.1 Anforderungen

Nachdem in den Abschnitten 3 und 4 die theoretische Basis erläutert wurde, soll nun die konkrete Realisierung des im Rahmen dieser Arbeit entwickelten Animationseditors dargestellt werden. Dabei werden zentrale Programmkomponenten wie Datenstrukturen, Animationsgraphen und Mausinteraktion selbst entwickelt, um so wertvolle Einblicke in die Funktionsweise einer solchen Applikation zu gewinnen. Folgende konkrete Funktionalität soll die zu entwickelnde Software bieten:

- Erstellen und Entfernen von Objekten.
- Verschiedene Objekteigenschaften wie Position, Orientierung und Skalierung können manipuliert werden.
- Eigenschaften können mit Schlüsselwerten versehen werden, und deren Interpolationsverhalten im Sinne von Abschnitt 4 geeignet beeinflusst werden.
- Da es sich um einen grafischen Editor handelt, sollen alle wichtigen Teile der Applikation mit der Maus bedienbar sein.
- Die erstellte Animation soll abspielbar und in Einzelbildern exportierbar sein.
- Es soll möglich sein, die Arbeit im Editor speichern und laden zu können.

Im Folgenden wird zuerst auf die verwendeten Hilfsmittel eingegangen, und anschließend der Gesamtaufbau der Applikation erläutert. Diese Betrachtung wird durch einen Überblick über Implementierungsdetails und beispielhafte Ergebnisse ergänzt.

5.2 Verwendete Hilfsmittel

Für zentrale Teile der Applikation soll in dieser Arbeit nicht auf existierende Lösungen zurückgegriffen werden. Es ist jedoch unmöglich und darüber hinaus nicht sinnvoll, alle für die Entwicklung eines solchen Editors notwendigen Bestandteile selbst zu entwickeln. Dieser Abschitt beschreibt, auf welche Bibliotheken und Hilfsmittel bei der Umsetzung zurückgegriffen wurde.

Die Applikation wurde mit der Programmiersprache C++ realisiert. Die weite Verbreitung der Sprache sowie die Verfügbarkeit vieler qualitativ hochwertiger Bibliotheken auf Basis von C++ machen diese Sprache

zu einer flexiblen Grundlage für die Programmentwicklung in dieser Arbeit. Mit der STANDARD TEMPLATE LIBRARY[15] (STL, "Standard Template Bibliothek") stellen moderne C++-Compiler eine Sammlung von effektiv implementierten Algorithmen und Datenstrukturen zur Verfügung. Die STL wurde für die flexible Datenhaltung im entwickelten Animationseditor verwendet. Weiterhin wurde auf die Bibliothek TINYXML[16] zurückgegriffen. Sie ermöglicht das Erstellen von Objekthierarchien nach dem XML-Standard[17] (EXTENSIBLE MARKUP LANGUAGE), sowie das Lesen und Erstellen von XML-Textdateien. Diese Bibliothek wurde verwendet, um das Speichern und Laden der Animationsdaten zu ermöglichen.

Ein Programm mit einer Benutzeroberfläche (*Graphical User Interface*, GUI) benötigt Funktionalität zur Darstellung und Nutzung von Knöpfen, Menus, Dialogen, und den vielen anderen Elementen einer graphischen Schnittstelle. Die Entwicklung einer Bibliothek zur Erstellung einer solchen Oberfläche stellt eine umfangreiche Aufgabe in sich selbst dar, die in verschiedenen Projekten erfolgreich umgesetzt wurde. Die APIs (*Application Programming Interface*, "Schnittstelle zur Applikationsentwicklung") von kommerziellen Betriebssystemen wie MIRCOSOFT WINDOWS[18] oder APPLE MAC OS X[19] ermöglichen es, Programme mit den Oberflächenbausteinen des Betriebssystems zu erstellen. Daneben gibt es freie Bibliotheken wie zum Beispiel GTK (GIMP TOOL KIT[20]), welche auf verschiedenen Plattformen jeweils ihr eigenes Aussehen definieren. Andere Projekte stellen eine Abstraktionsebene zu Verfügung, mit der Benutzeroberflächen unabhängig von konkreten Bibliotheken definiert werden können. So kann eine Applikation auf verschiedenen Plattformen die jeweils spezifischen Oberflächenbibliotheken nutzen. Obwohl diese Abstraktion natürlich ganz eigene Probleme aufwirft, wurde im Sinne späterer Flexibilität zur Realisierung der Editoroberfläche ein solcher Ansatz gewählt.

Das Projekt WXWIDGETS[21] stellt eine ausgereifte Bibliothek zur Verfügung, die es erlaubt, Oberflächen für eine Vielzahl von Betriebssystemen zu entwickeln (inklusive MICROSOFT WINDOWS, APPLE MAC OS X und LINUX). WXWIDGETS wurde gewählt, da der Programmcode unter der Lizenz LGPL (LESSER GNU PUBLIC LICENSE[22], eine Lizenz für quelloffenen, nichtkommerzielle Software) steht, und somit ohne Probleme verwendbar ist. Weiterhin existiert eine umfangreiche Dokumentation mit

[15] Zugehörige Website im Oktober 2006: http://www.sgi.com/tech/stl/.
[16] Zugehörige Website im Oktober 2006: http://tinyxml.sourceforge.net/.
[17] Zugehörige Website im Oktober 2006: http://www.w3.org/XML/.
[18] Zugehörige Website im Oktober 2006: http://www.microsoft.com.
[19] Zugehörige Website im Oktober 2006: http://www.apple.com/macosx/.
[20] Zugehörige Website im Oktober 2006: http://www.gtk.org/.
[21] Zugehörige Website im Oktober 2006: http://www.wxwidgets.org.
[22] Zugehörige Website im Oktober 2006: http://www.gnu.org/licenses/lgpl.html.

Programmbeispielen sowohl im Internet[23] als auch im Form eines eigens veröffentlicheten Buchs [wxWidgets05]. In dieser Arbeit wurde die Version 2.6 der WXWIDGETS-Bibliothek verwendet.

Die Entwicklung einer kompletten Benutzeroberfläche auf Ebene des Programmcodes ist eine aufwändige und unübersichtliche Aufgabe. In Form eigener Editoren existieren Werkzeuge, mit deren Hilfe Benutzeroberflächen graphisch zusammengestellt werden können. Diese Werkzeuge erstellen dann automatisch den notwendigen Programmcode zur erzeugten Oberfläche. Weiterhin ist dieser Code in einer bewährten Art angeordnet[24]. Dieses sogenannte *Rapid Application Development* (RAD, "Schnelle Applikationsentwicklung") erlaubt es, komplexere Oberflächen in kurzer Zeit zu erstellen. Der Nutzer kann sich zu grossen Teilen auf die Funktionalität hinter der Oberfläche konzentrieren. Der Editor DIALOGBLOCKS[25] zur Erstellung von WXWIDGETS-Oberflächen ist eng verbunden mit dem WXWIDGETS-Projekt selbst, und steht für viele Plattformen zur Verfügung. Da es sich jedoch um ein kommerzielles Projekt handelt, wurde auf den für MIRCOSOFT WINDOWS verfügbaren Editor WXDEVCPP[26] zurückgegriffen, welcher quelloffen und frei (GPL, GNU PUBLIC LICENSE[27]) verfügbar ist. Beispiele einer mit WXWIDGETS realisierten Applikation zeigt Abbildung 19[28]. Abbildung 20 zeigt beispielhaft die Oberfläche des Editors WXDEVCPP.

Zur Darstellung der Objekte, Erstellen der Animationseinzelbilder (*Rendering*) sowie für Teile der Benutzerschnittstelle wurde die OPENGL-Bibliothek (OPEN GRAPHICS LIBRARY[29], "Offene Graphik-Bibliothek") verwendet. Sie ermöglicht das effiziente Darstellen von räumlichen *Szenen*, erstellt also zweidimensionale Bilder aus dreidimensionalen Daten. OPENGL ist ein weit verbreiteter Standard, zu dem umfangreiche Dokumentation existiert, wie zum Beispiel in [OpenGL05].

5.3 Aufbau der Applikation

Die im Rahmen dieser Arbeit erstellte Applikation trägt den Namen ANIMATIONPLAYGROUND ("Animationsspielplatz"). In diesem Abschnitt wird ein Überblick über den Aufbau dieses Animationseditors gegeben, auf Details der Implementierung wird im nächsten Abschnitt eingegangen. Die Benutzeroberfläche teilt sich in zwei Fenster, dem *Szeneneditor* und dem *Interpolationseditor*, wobei der Szeneneditor das Hauptfenster des Programms darstellt.

Alle Objekte und Einstellungen einer Animation werden in einer *Szene*

[23] Zugehörige Website im Oktober 2006: http://www.wxwidgets.org/manuals/2.6.3/wx_contents.html.
[24] Auch bei der Verwendung einer Bibliothek kann man ungeschickt vorgehen.
[25] Zugehörige Website im Oktober 2006: http://www.anthemion.co.uk/dialogblocks.
[26] Zugehörige Website im Oktober 2006: http://wxdsgn.sourceforge.net.
[27] Zugehörige Website im Oktober 2006: http://www.gnu.org/copyleft/gpl.html.
[28] Gezeigt sind Abbildungen der Applikation AUDACITY. Zugehörige Website im Okto-

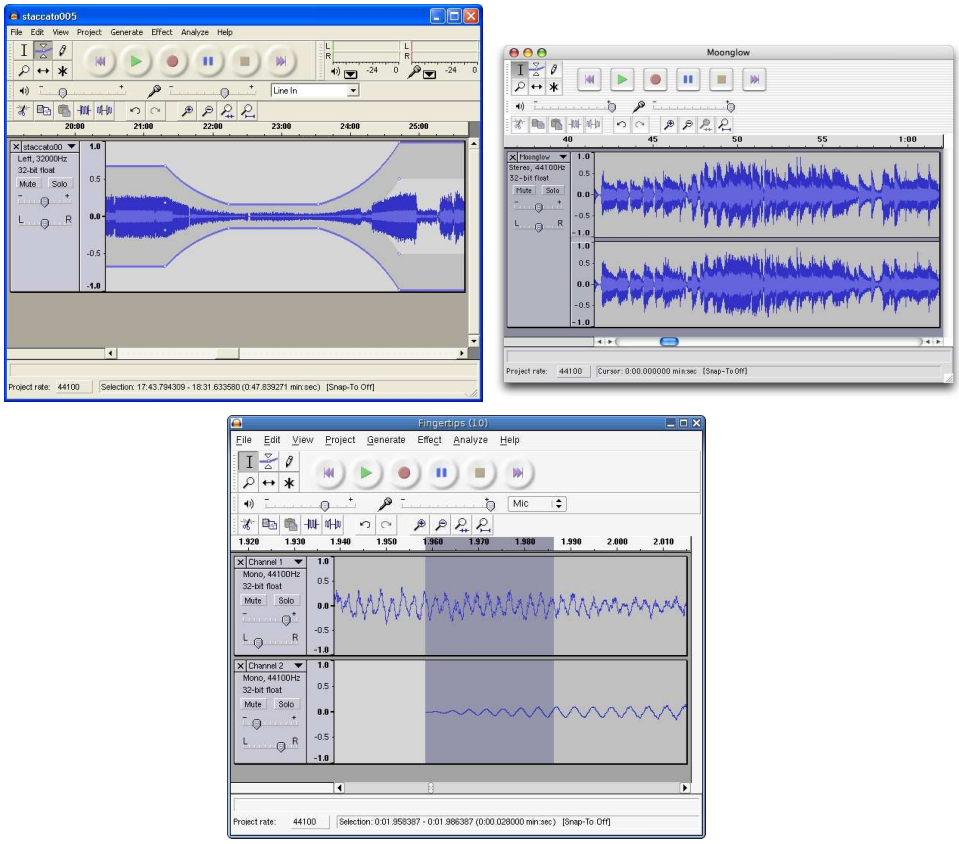

Abbildung 19: Eine WXWIDGETS-Applikation auf verschiedenen Betriebssystemen. Gezeigt ist die frei Applikation AUDACITY.

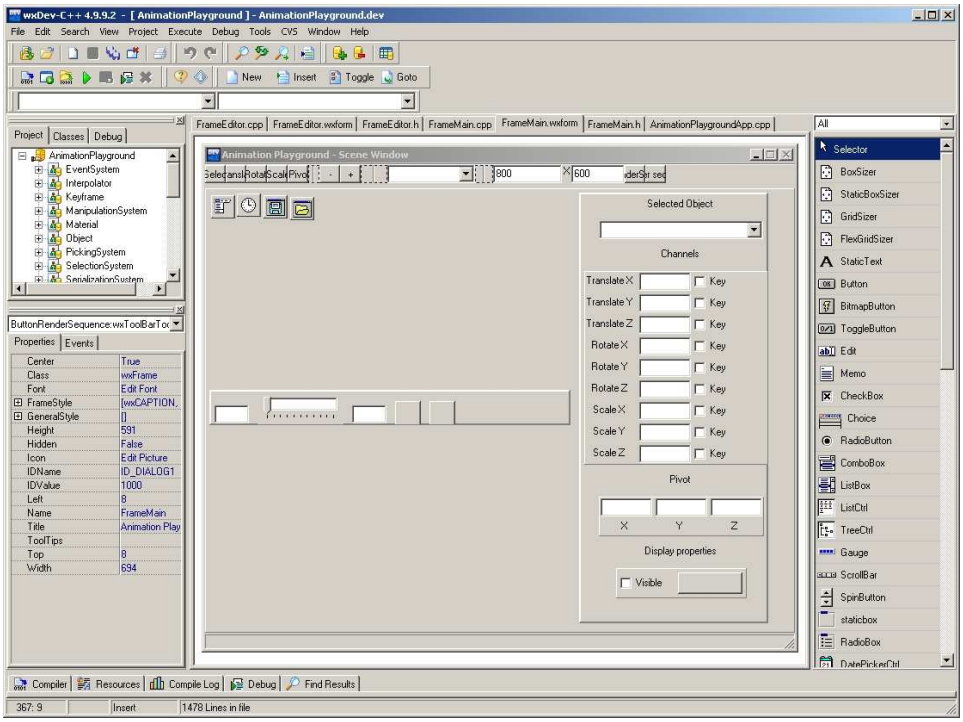

Abbildung 20: Beispielhafte Abbildung des WXDEVCPP-Editors zur Erstellung von WXWIDGETS-Applikationen.

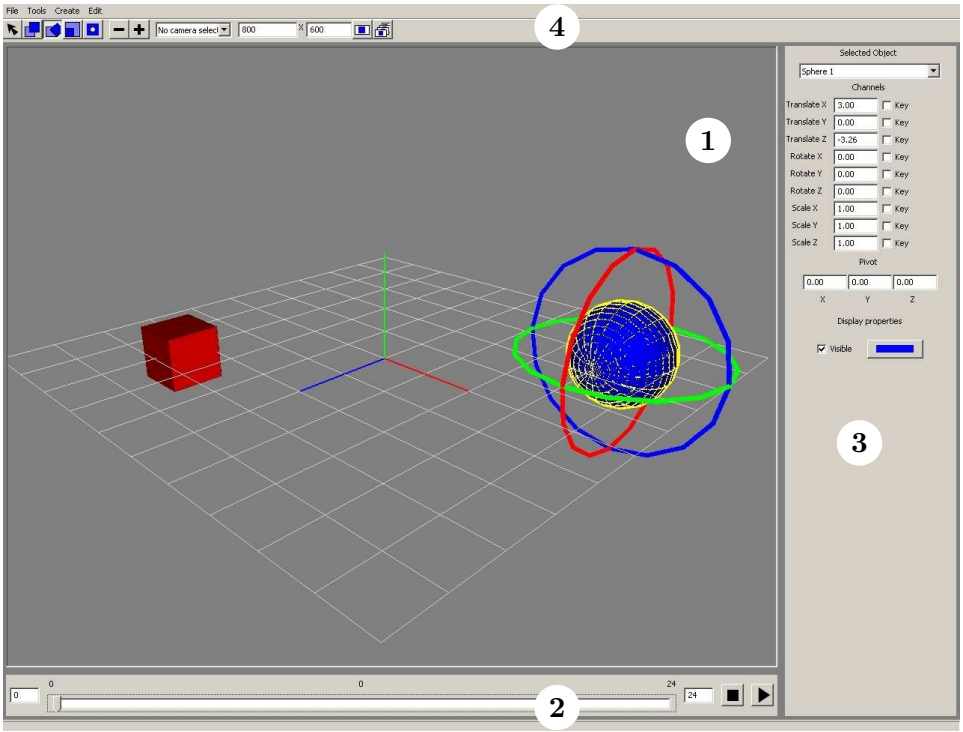

Abbildung 21: Der *Szeneneditor* der Applikation ANIMATIONPLAYGROUND. Er besteht aus dem zentralen Sichtfenster zur Darstellung des Objektraums (1), Zeitleiste (2), Objektfeld (3), Werkzeug- und Menüleiste (4).

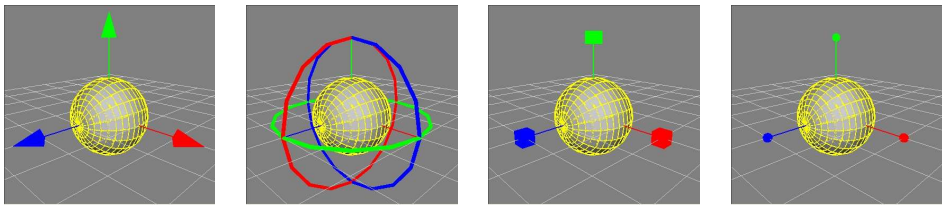

Abbildung 22: Manipulationswerkzeuge im Szeneneditor (von links nach rechts): Translation, Rotation, Skalierung, Verschiebung des *Pivot*-Punkts.

zusammengefasst, die in Form einer XML-Datei abgespeichert und geladen werden kann. Der Szeneneditor (Abbildung 21) ermöglicht Einsicht in die Szene, sowie deren Manipulation. Das zentrale Sichtfenster gibt Einsicht in den dreidimensionalen Raum, und stellt die Objekte aus einer vom Benutzer steuerbaren Position und Orientierung dar. Hier ist es möglich, Objekte zu *selektieren* und durch *Manipulatoren* direkt zu beeinflussen (Abbildung 22). Unter dem Sichtfenster ist das Zeitfeld zu finden, in dem sich die Zeitleiste und Knöpfe zum Abspielen der Animation befinden. Der Zeitverlauf einer Animation wird in Einzelbilder unterteilt. In der Zeitleiste kann das aktuelle Einzelbild sowie der durch die Leiste abgedeckte Zeitraum gewählt werden. Veränderungen von Objekteigenschaften, die nicht durch diesen Zeitausschnitt abgedeckt werden, gehen dabei nicht verloren. Im Objektfeld, welches sich rechts vom Sichtfenster befindet, kann ebenfalls ein Objekt ausgewählt werden. Dort können alle Objekteigenschaften durch entsprechende Eingabefelder beeinflusst werden. Hier ist es auch möglich, die Kanäle zu einem bestimmten Zeitpunkt mit einem *Keyframe* zu versehen, das heisst den Wert zu diesem Zeitpunkt festzulegen. Die Auswahl der Manipulationswerkzeuge, sowie die Kontrolle der Erstellung von Einzelbildern aus der Animation sind in der *Werkzeugleiste* über dem Sichtfenster angebracht. Auf die Erstellung der Objekte sowie das Speichern und Laden kann über die Menüleiste des Szenenfensters zugegriffen werden.

Als Animationseditor erlaubt es ANIMATIONPLAYGROUND, Objekte zu erstellen. Diese haben folgende Eigenschaften:

- *Pivot* ("Drehpunkt"): Der Mittelpunkt, auf den sich die anderen Eigenschaften beziehen.
- *Translation*: Ein Positionsvektor entsprechend Abschnitt 3.1.1.
- *Rotation*: Eine *Cardan*-Rotation entsprechend Abschnitt 3.1.2.

ber 2006: http://audacity.sourceforge.net/.

[29] Zugehörige Website im Oktober 2006: http://www.opengl.org.

- *Skalierung*: Ein dreidimensionaler Vektor, dessen Komponenten die Streckung oder Stauchung des Objekts entlang der drei Koordinatenachsen bestimmen.
- Farbe: Die Farbe des Objekts.
- Sichtbarkeit: Objekte können unsichtbar gemacht werden.

Bei der Rotation wurden trotz der *Gimbal Lock*-Problematik die *Cardan*-Winkel gewählt, da sie einfach zu bedienen sind. (vergleiche Abschnitt 3.1.2). Neben den Objekttypen *Würfel, Kugel, Teekanne* und *Ebene* gibt es noch die Spezialobjekte *Kamera* und *Positionszeiger* (engl. *Locator*). Eine Kamera ist mit einem Positionsanzeiger verbunden, welcher das Blickziel und somit eine Blickrichtung zwischen Kamera- und Zielposition definiert. Rotation und Skalierung können bei Kamera und Positionsanzeiger nicht verwendet werden, die Translation funktioniert normal. Die Eigenschaften Translation, Rotation und Skalierung können animiert werden, die anderen sind über die Zeit statisch. Die neun Komponenten dieser Vektoren werden dabei *Kanäle* beziehungsweise *Channel* genannt. Jeder dieser Kanäle kann für jedes Objekt unabhängig animiert werden. Es existieren also folgende animierbaren Kanäle:

- Translation X
- Translation Y
- Translation Z
- Rotation X
- Rotation Y
- Rotation Z
- Skalierung X
- Skalierung Y
- Skalierung Z

Es wurde der in Abschnitt 4.3 vorgestellte *implizite Ansatz* zur Animation umgesetzt. *Keyframes* speichern also für einen bestimmten Zeitpunkt (ein bestimmtes Einzelbild) den Wert eines Kanals. Die Animationsgraphen stellen somit ein Wert-Zeit-Diagramm dar. Die Definition der zur Interpolation der Schlüsselwerte verwendeten Kurven ist eng an die Definition von *Hermite*-Kurven angelehnt. Zwei Kontrollpunkten $\boldsymbol{p_0}$ und $\boldsymbol{p_1}$ werden ebenfalls zwei Tangenten $\boldsymbol{d_0}$ und $\boldsymbol{d_1}$ zur Seite gestellt, die jedoch genau auf die entsprechenden mittleren Punkte einer kubischen *Bézier*-Kurve zeigen (siehe Gleichung 7).

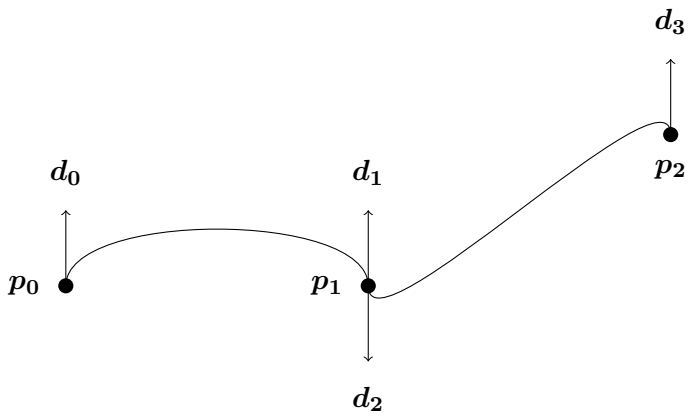

Abbildung 23: Beispielhafte Interpolationskurve zum entwickelten Animationseditor. Die Kurve entspricht der in Abbildung 9 gezeigten *Hermite*-Kurve.

$$\begin{aligned} \boldsymbol{p_0^B} &= \boldsymbol{p_0} \\ \boldsymbol{p_1^B} &= \boldsymbol{p_0} + \boldsymbol{d_0} \\ \boldsymbol{p_2^B} &= \boldsymbol{p_1} + \boldsymbol{d_1} \\ \boldsymbol{p_3^B} &= \boldsymbol{p_1} \end{aligned} \tag{7}$$

Desweiteren besitzen Kontrollpunkte, an denen zwei Kurven zusammenstossen, für jede Kurve eine eigene Tangente. Abbildung 23 zeigt ein Beispiel einer solchen Kurve. Die *Keyframes* stellen die Kontrollpunkte der Interpolationskurven dar, benachbarte Schlüsselwerte eines Kanals werden mit einer Kurve verbunden. Für jeden *Keyframe* kann eingestellt werden, ob die zwei möglichen Tangenten verbunden sein sollen oder nicht (*Locked tangents*). Die Richtung verbundener Tangenten wird abgeglichen, so dass eine *stetige Kurve* sichergestellt ist. Statt einer Kurveninterpolation ist auch eine lineare Interpolation zwischen zwei Schlüsselwerten einstellbar. Die Applikation verbindet zwei *Keyframes* standardmäßig mit einer Kurve, die *Ease-in/Ease-out*-Verhalten realisiert.

Die Interpolationskurven zu den im Szeneneditor angelegten *Keyframes* können im *Interpolationseditor* bearbeitet werden (Abbildung 24). Im zentralen Sichtfenster werden *Keyframes*, Kurven und Tangenten dargestellt. Der hier angezeigte Raum ist zweidimensional, die x-Achse entspricht der Zeitachse, die y-Achse der Werteachse. Positionen von Schlüsselwerten und Tangenten können hier angepasst werden. Weiterhin ist es in diesem Teil der Applikation möglich, Keyframes für einen Kanal anzulegen oder zu löschen. Dies ist eine redundante Funktionalität zum Szeneneditor, die je-

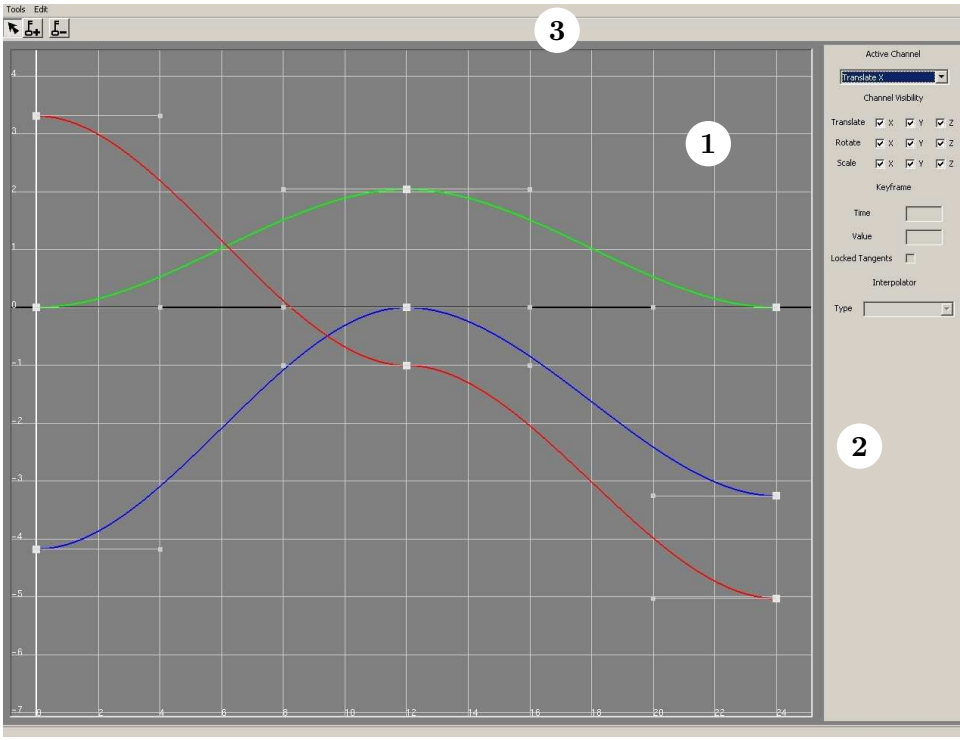

Abbildung 24: Der *Interpolationseditor* der Applikation ANIMATIONPLAYGROUND. Er besteht aus dem zentralen Sichtfenster zur Darstellung der Objektkanäle (1), Wertefeld für *Keyframes* und Interpolationskurven (2), Werkzeug- und Menüleiste (3).

doch auch im Interpolationseditor benötigt wird. So kann einem Graphen schnell ein weiterer Kontrollpunkt hinzugefügt werden. Im Wertefeld an der rechten Seite des Interpolationseditors kann der Abgleich zweier Tangenten an einem Schlüsselwert ermöglicht oder verhindert, sowie die Interpolationsart zwischen zwei *Keyframes* eingestellt werden. Auch die numerischen Werte eines *Keyframes* können hier angepasst werden. Werkzeuge und Funktionen werden auch hier über eine Werkzeug- und eine Menüleiste zur Verfügung gestellt.

In den zentralen Sichtbereichen der einzelnden Editorfenster kann mit Maus und Tastatur navigiert werden, Elemente wie Objekte, Keyframes oder Tangenten können mit der Maus manipuliert werden. Viele Werkzeuge und Funktionen sind mit Tastaturkürzeln versehen. Die Farbgebung von Manipulatoren und Kurven folgt dem Schema:

- Rot: Anzeige oder Manipulation von Kanälen mit Komponente X.
- Grün: Anzeige oder Manipulation von Kanälen mit Komponente Y.

- Blau: Anzeige oder Manipulation von Kanälen mit Komponente Z.

Bei der Gestaltung von Elementen wie Navigation, Farbgebung oder Form der Manipulatoren wurde sich an verbreiteten Programmen wie MAYA und LIGHTWAVE orientiert.

5.4 Implementierung

5.4.1 Gesamtstruktur

In den folgenden Abschnitten werden Implementierungsdetails vorgestellt. Durch Beschreibung zentraler Elemente der *Programmcode*- und *Klassenstruktur* sollen zentrale Zusammenhänge der Applikation ANIMATIONPLAYGROUND vorgestellt werden. Die Programmbestandteile lassen sich den drei Gruppen *Fenster*, *Szene* und *Komponentensammlung* zuordnen. In der Gruppe *Fenster* befinden sich die OPENGL-basierten Sichtfenster (*Viewports*) sowie die Bestandteile der Benutzeroberfläche, realisiert mit WXWIDGETS. In dieser Gruppe wird also die Oberfläche von Szeneneditor und Interpolationseditor erstellt. Die Gruppe *Szene* beinhaltet Klassen für Objekte, Schlüsselwerte und Interpolationsgraphen. Somit befindet sich in dieser Gruppe der Programmcode für die zentrale Animationsfunktionalität. Die Gruppe *Komponentensammlung* ist die Umfangreichste, denn hier befinden sich viele Bausteine für unterschiedliche Funktionalität. Diese Bausteine werden in den anderen Gruppen, aber auch untereinander, für die Realisierung der eigenen Aufgabe benötigt. Abbildung 25 gibt einen Überblick über die Bestandteilsgruppen der Applikation.

In der Gruppe Komponentensammlung sind zwei Arten von Komponenten zu unterscheiden. Zum einen existieren eine Vielzahl von Klassen, die Funktionalität kapseln, die an vielen Stellen Verwendung finden. Dies sind zum Beispiel neue Datentypen wie Vektoren, aber auch *Objektfabriken* oder *Manager*. Fabriken übernehmen die Erzeugung wichtiger Objekttypen wie Objekte oder *Keyframes*. Manager übernehmen Vermittlungsaufgaben, wie sie zum Beispiel beim Selektieren anfallen. Die andere Gruppe von Komponenten besteht aus Basisklassen zur Definition von Eigenschaftsprofilen, die teilweise *abstrakte* Methoden beinhalten. Es existieren folgende Basisklassen zur Eigenschaftsdefinition:

- `Drawable`: Das Objekt kann OPENGL-Geometrie erzeugen, also im *Viewport* gezeichnet werden.
- `EventHandler`: Empfängt und verarbeitet Ereignisse.
- `EventThrower`: Erzeugt Ereignisse.
- `Pickable`: Kann mit der Maus angeklickt werden.
- `Selectable`: Kann selektiert werden.

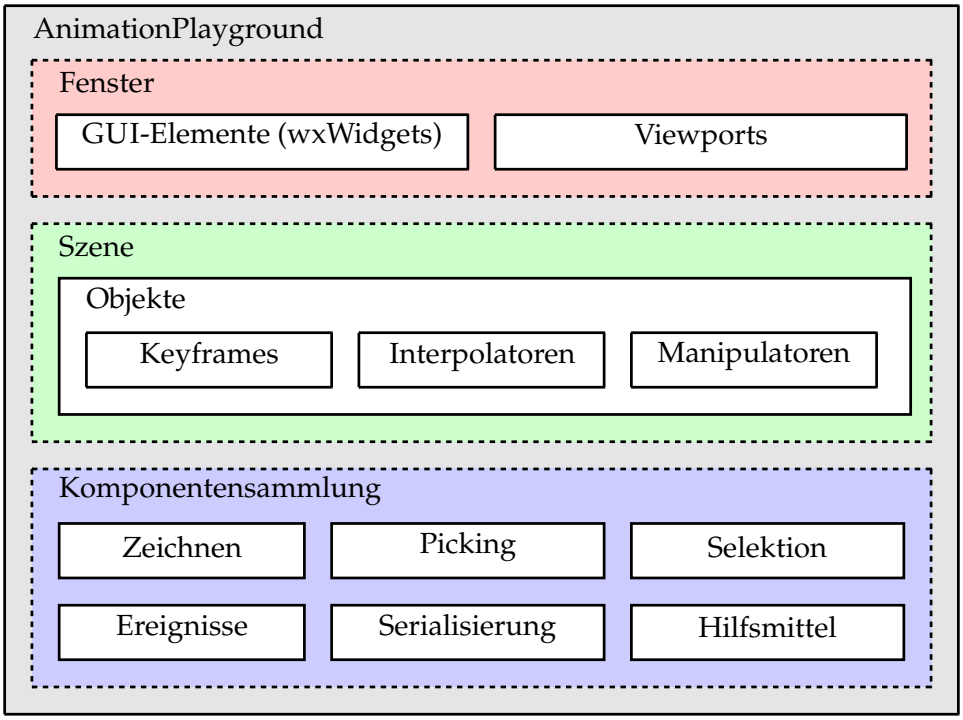

Abbildung 25: Grundlegende Komponenten der Applikation ANIMATION-PLAYGROUND.

- `Serializable`: Kann relevante Daten in einen XML-Datenbaum schreiben, und davon lesen.

Da diese Basisklassen jedoch nicht alle rein abstrakt sind, handelt es sich nicht um reine *Interfaces*. Vielmehr wird umfangreich *Mehrfachvererbung* eingesetzt, um aus den Basiskomponenten Objekte mit verschiedenen Eigenschaften zu erzeugen. So kann durch Mehrfachvererbung zum Beispiel ein Objekt erstellt werden, welches sowohl *selektierbar* als auch *zeichenbar* ist. Abgeleitete Objekte greifen auf Funktionalität und Daten der Basisklassen zurück, und implementieren Teile der Funktionalität selbst. Dies ermöglicht den Zugriff auf Objekte über einen Basisklassenzeiger, sowie die schnelle Wiederverwendung von Programmcode. Die Entscheidung für den sicherlich nicht ganz unproblematischen Einsatz umfangreicher Mehrfachvererbung wurde aufgrund dieser Vorteile getroffen.

5.4.2 Fenster

Alle Bestandteile der Benutzeroberfläche, also die Fenster von Szeneneditor und Interpolationseditor und die dort enthaltenen Menus und Schaltknöpfe wurden mit WXWIDGETS-Klassen oder davon abgeleiteten Klassen

realisiert. Die Strukturierung des WXWIDGETS-Programmcodes übernimmt dabei die Entwicklungsumgebung WXDEVCPP. Für einen Großteil der GUI wurde die vorhandene Funktionalität von WXWIDGETS weiterverwendet. Nur für die OPENGL-Sichtfenster mussten umfangreichere Anpassungen vorgenommen werden. Von der Klasse `wxGLCanvas`, die einen OpenGL-Kontext bereitstellt, wurden dazu über die Basisklasse `Viewport` die Klassen `Viewport3D` und `ViewportEditor` abgeleitet. `Viewport3D` stellt die Steuerung und Darstellung der dreidimensionalen Ansicht im Szeneneditor zur Verfügung, `ViewportEditor` realisiert Darstellung und Kontrolle der zweidimensionalen Graphendarstellung im Interpolationseditor. Auch Hilfsmittel wie das *Grid* oder die Koordinatenachsen werden in diesen Klassen implementiert. Jedes Objekt, welches in einem *Viewport* dargestellt werden soll, wird von `Drawable` abgeleitet. Die Basisklasse stellt Funktionalität für *Materialien* und Oberflächen bereit, die abgeleitete Klasse implementiert die dazustellende Geometrie mit üblichen OPENGL-Funktionen. Oft ist es notwendig, bei einem Maustastendruck im *Viewport* die Geometrie unter dem Mauszeiger zu identifizieren. Nur so können Objekte mit der Maus anwählbar ("anklickbar") sein. Dazu wird eine Klasse zusätzlich von `Pickable` abgeleitet. Eine `PickingManager`-Instanz verwaltet die Farbe, mit der alle Instanzen der von `Pickable` abgeleiteten Klassen in dem OPENGL-Hintergrundpuffer gezeichnet werden, und weist jedem Objekt eine eindeutige Farbe zu. Nach einem Maustastendruck kann das Manager-Objekt gefragt werden, welches Objekt zu der Farbe unter dem Mauszeiger gehört. Auch Verdeckungen mehrerer Objekte werden so automatisch behandelt, da immer nur die Farbe des vordersten Objekts sichtbar ist.

Im Allgemeinen sollen sichtbare und anwählbare Objekte auch selektierbar sein. Es soll also möglich sein, ein oder mehrere Objekte den besonderen Zustand zu geben, gerade das aktiv bearbeitete Objekt zu sein. In der erstellten Applikation ist es dazu möglich, von der Klasse `Selectable` abzuleiten. Instanzen solcher Klassen können entweder *selektiert* oder *nicht selektiert* sein. Ein Objekt der Klasse `SelectionManager` verwaltet dabei eine Menge von selektierbaren Objekten. In der implementierten Version kann nur ein Objekt pro Manager selektiert sein, Mehrfachselektion ist nicht möglich.

5.4.3 Szene und Objekte

Animierbare Objekte haben alle die Basisklasse `Object`, und werden von einer Instanz der Klasse `ObjectIndex` verwaltet. Die Szene, also eine Instanz der Klasse `Scene`, greift auf die Objekte immer über diesen Objektindex zu. Die Klasse `Object` und die dort verwalteten Daten stellen das Herzstück des Animationseditors dar. Hier werden Schlüsselwerte und Interpolationskurven verwaltet. Die Klasse `Object` selbst ist ein gutes Bei-

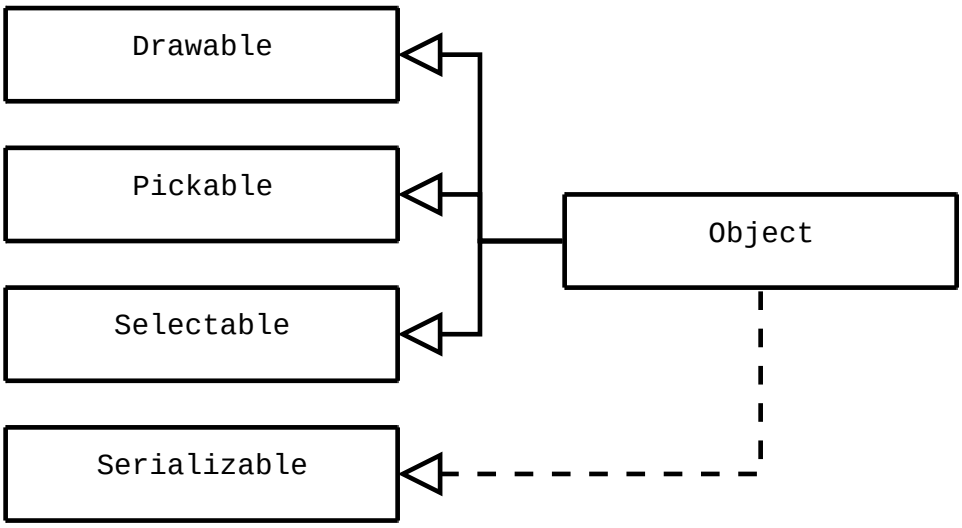

Abbildung 26: Basisklassen der Klasse `Object`. Dabei stellt nur die Basis `Serializable` ein echtes *Interface* dar, die anderen Klassen vererben auch Programmcode.

spiel für die Mehrfachvererbung von Basiskomponenten (vergleiche Abschnitt 5.4.1). Sie leitet sich von vier Basisklassen ab, um Darstellbarkeit, Anwählbarkeit mit der Maus, Selektierbarkeit und Serialisierbarkeit zu ermöglichen (siehe Abbildung 26). Zu einer Instanz der Klasse `Object` gehört jeweils eine Instanz der Klassen

- `KeyframeIndex` und
- `InterpolatorIndex`

welche die zum Objekt gehörenden *Keyframes* und Interpolatoren den neun Kanälen zu Translation, Rotation und Skalierung zuordnen. Zu benachbarten *Keyframes* eines Kanals, also Instanzen der Klasse `Keyframe`, existiert jeweils eine Instanz einer von `Interpolator` abgeleiteten Klasse. Interpolatoren übernehmen die Zwischenwertberechnung zwischen Schlüsselwerten. Die Klasse `Object` selbst übernimmt das Anlegen, Aktualisierung und Löschen von *Keyframes* und Interpolatoren, sowie die daraus resultierenden Auswirkungen. So müssen nach Entfernung eines Schlüsselwerts auch Interpolatoren entfernt werden, die mit diesem Keyframe verbunden waren. Abbildung 27 gibt einen Überblick über die für Objekte wichtigen Klassen.

Zur Interpolation zwischen Schlüsselwerten gibt es zwei implementierte Interpolator-Typen. `InterpolatorLinear` realisiert eine lineare Interpolation der verbundenen `Keyframe`-Instanzen. `InterpolatorBezier`

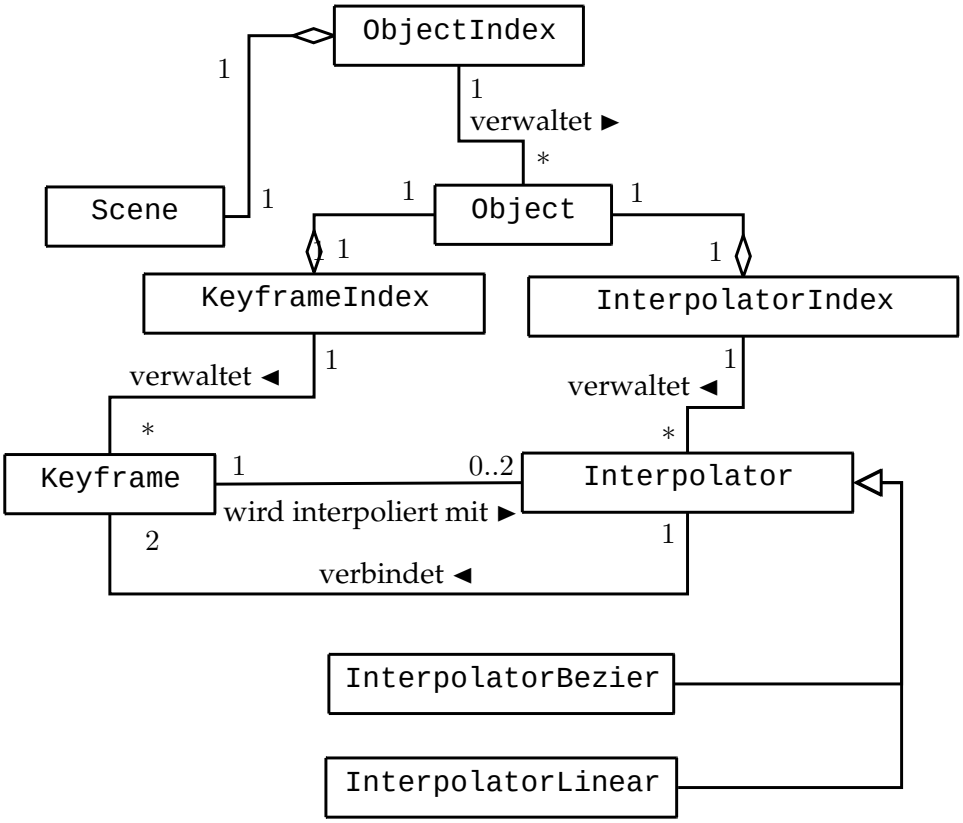

Abbildung 27: Klassendiagramm zu den zentralen Kompontenten Objekt, *Keyframe* und Interpolator.

realisiert die zentrale Funktionalität der Interpolation mit einer kubischen *Bézier*-Kurve. Sie nutzt Instanzen der Klassen

- `BezierSpline2D` und
- `InterpolationTangent2D`

um mit den verbundenen `Keyframe`-Instanzen eine kubische *Bézier*-Kurve zu erstellen. Dazu wird aus Schlüsselwerten und Tangenten die Kurve erstellt (siehe Abschnitt 5.3), welche dann den Wert-Zeit-Graph realisiert. Wie in Abschnitt 4.3.2 beschrieben muss zur Bestimmung des Werts eines Kanals zu einer bestimmten Zeit der Punkt auf der Interpolationskurve bestimmt werden, dessen x-Wert dem Zeitpunkt entspricht. Bei korrekten Interpolationsgraphen kann es nur einen solchen Punkt geben. Dieser Punkt ist daher unter Verwendung von *Intervallteilung* ermittelbar. Errechnet man mit Hilfe des *de Casteljau*-Algorithmus (siehe Codefragment 1) den Punkt für den mittleren Wert eines Intervalls des Kurvenparameters t, so erhält man einen Punkt mit x- und y-Wert. Es ist sichergestellt, das alle Punkte der so entstandenen einen Intervallhälfte kleinere x-Werte haben als der ermittelte Punkt, und ebenso ausschließlich größere x-Werte im anderen Intervallteil vorliegen. Durch wiederholtes Ausführen dieses Vorgehens kann ein Punkt auf der Kurve ermittelt werden, dessen x-Wert hinreichend Nahe am gesuchten Wert liegt. Dieser Algorithmus ist in der Funktion `getYIntersection` der Klasse `BezierSpline2D` implementiert, und wird in Codefragement 2 beschrieben.

Da die Klassen zu Objekten, *Keyframes*, Interpolatoren und Tangenten jeweils von `Drawable` ableiten, können diese im entsprechenden *Viewport* direkt dargestellt werden. Sie haben auch `Pickable` und `Selectable` als Basis, so dass die Instanzen dieser Klassen direkt im Sichtfenster manipuliert werden können. So ist es zum Beispiel möglich, eine Tangente im Interpolationseditor zu verschieben. Die zugehörigte Interpolator-Instanz greift beim nächsten Darstellungszyklus auf die Tangenten-Instanz zu, und die veränderten Werte werden übernommen. Die Benutzerschnittstelle wird mit echtzeitfähigen Raten neu gezeichnet, so dass sich Veränderungen der im Sichtfenster dargestellten Entitäten für den Betrachter unmittelbar auswirken. Für die Manipulation selektierter Gegenstände stellen die *Viewports* der zwei Editoren jeweils geeignete Werkzeuge zur Verfügung. Weiterhin existieren im Sichtfenster des Szeneneditors *Manipulatoren* für die Objekteigenschaften Translation, Rotation, Skalierung, und *Pivot*-Position. Diese sind selbst auswählbar, und erlauben die Veränderung der Eigenschaften des aktuell selektierten Objekts. Manipulatoren sind von der Basisklasse `Manipulator` abgeleitet.

Codefragment 2 : getYIntersection

Parameter : x, der x-Wert des gesuchten Bézier-Punkts $\boldsymbol{p} = (p_x, p_y)$
Parameter : t_{start}, Startwert des Parameterintervalls
Parameter : t_{stop}, Endwert des Parameterintervalls
Rückgabewert : y, der y-Wert des gesuchten Bézier-Punkts

```
Funktion getYIntersection(x, t_start, t_stop)
{
    if t_start − t_stop < Abbruchkriterium then
        Breche Berechnung ab;
    end
    Parameter t = Mittelwert von t_start und t_stop;
    Bézier-Punkt p = deCasteljau(t);
    if |p_x − x| < Schwellwert then
        y = p_y;
    else
        if p_x < x then
            y = getYIntersection(x,t,t_stop);
        else
            y = getYIntersection(x,t_start,t);
        end
    end
    Liefere y zurück;
}
```

5.4.4 Weitere Komponenten

Bisher wurden die Code-Gruppen *Fenster* und *Szene* genauer erläutert. Bei dieser Betrachtung wurden für diese Gruppen wichtige Teile der *Komponentensammlung* beschrieben. Zusätzlich gibt es neben Klassen zur Darstellung von Vektoren noch weitere wichtige Komponenten.

Alle Objekte, *Keyframes* und Interpolatoren haben eine in der Szene eindeutige Identifikationsnummer, mit der sie angesprochen werden können. Das Erstellen der Instanzen dieser Klassen wird daher von *Fabriken* übernommen, damit eine eindeutige ID-Vergabe gewährleistet ist. Die Fabriken ermöglichen auch den Zugriff auf Instanzen über die Identifikationsnummer. Die Klassen der Fabriken sind:

- `ObjectFactory`
- `KeyframeFactory`
- `InterpolatorFactory`

Die Fabriken sind in der Lage, die verschiedenen Objekt- und Interpolatortypen zu erzeugen. Eigene Fabriken für cnObject oder `Interpolator` abgeleitete Klasse existieren nicht.

Es wurde ein einfaches *Ereignissystem* implementiert. Ereignisempfänger können dazu von der Basisklasse `EventHandler` ableiten, zum Erzeugen von Ereignissen kann eine Klasse `EventThrower` als Basis besitzen. Ereignisverarbeitende Instanzen melden sich bei einem Ereignismanager an, welcher vom Typ `EventManager` ist. Die Ereignistypen selbst sind fest implementiert, zur Behandlung eines bestimmten Typs überschreibt der Ereignisempfänger die entsprechende *virtuelle Funktion*. Jeder Ereignisproduzent kann jeden Typ von Ereignis auslösen. Ereignisse werden verwendet, um notwendige Aktualisierungen im System auszulösen. So sind Sichtfenster auch Ereignisempfänger und stellen die Szene neu dar, wenn ein Event eine Änderung des Inhalts signalisieren.

Zur Umsetzung des Ladens und Speicherns der Szenendaten wurde die Basisklasse `Serializable` erstellt. Sie enthält zwei virtuelle Methoden `read` und `write`, die beim Ableiten implementiert werden. Beim Erstellen der XML-Daten erzeugt eine Instanz zuerst ein `TiXmlElement`-Objekt der TINYXML-Bibliothek, welches ihr selbst entspricht. Relevante Daten werden in diesen Datenknoten geschrieben. Das XML-Element selbst wird als Kind an den Knoten angehängt, der der Serialisierungsfunktion `write` als Parameter übergeben wurde. Für komplexere Daten einer Instanz, die selbst von `Serializable` abgeleitet wurden, wird ebenfalls die `write`-Funktion aufgerufen, mit dem neu erstellten XML-Knoten der Instanz als Parameter. Ausgehend von der Szene wird so ein Baum aus `TiXmlElement`-Knoten aufgebaut, welcher alle relevanten Daten der Animation enthält.Die Lesefunktion `read` geht den umgekehrten Weg, um die Daten aus einem

Datenbaum auszulesen. Die TINYXML-Bibliothek ermöglicht weiterhin das Erstellen einer XML-Datei aus einem solchen Baum, und das Erstellen eines XML-Datenbaums aus einer Datei. Abbildung 28 zeigt einen Teil einer XML-Szenendatei. Beispieldateien sind auf der beigefügten CD zu finden.

5.5 Weitere Dokumentation

Neben den Ausführungen in dieser Arbeit sind weitere dokumentierende Materialen auf der beigefügten CD enthalten. Es wurden drei kurze Filmsequenzen erstellt. FEATUREOVERVIEW zeigt das Programm ANIMATIONPLAYGROUND in Benutzung und veranschaulicht zentrale Funktionskomponenten. SIMPLEANIMATION1 und SIMPLEANIMATION2 sind kurze Animationssequenzen, die mit der entwickelten Applikation erstellt wurden. Die zugehörigen XML-Dateien sind ebenfalls enthalten. Weiterhin sind die genutzten Bibliotheken und Programme (siehe Abschnitt 5.2), jeweils als Archiv, auf der CD enhalten. Der komplette erstellte Quellcode ist inklusive eines Projekts für WXDEVCPP mit beigefügt.

Die Einzelbilder von SIMPLEANIMATION1 sind ebenfalls im Anhang zu finden.

```
<?xml version="1.0" ?>
<AnimationPlaygroundFile>
  <Scene>
    <Time FrameStart="0" FrameEnd="24" FrameCurrent="24" />
    <ObjectIndex>
      <Object Type="Plane" Id="1">
        <CurrentTransformation>
          <Vector3D x="0.000000" y="0.000000" z="0.000000" />
        </CurrentTransformation>
        <CurrentRotation>
          <Vector3D x="0.000000" y="0.000000" z="0.000000" />
        </CurrentRotation>
        <CurrentScale>
          <Vector3D x="8.000000" y="8.000000" z="8.000000" />
        </CurrentScale>
        <Pivot>
          <Vector3D x="0.000000" y="0.000000" z="0.000000" />
        </Pivot>
        <KeyframeIndex>
          <Channel Type="TRANSLATE_X" />
          <Channel Type="TRANSLATE_Y" />
          <Channel Type="TRANSLATE_Z" />
          <Channel Type="ROTATE_X" />
...
      </Object>
    </ObjectIndex>
  </Scene>
</AnimationPlaygroundFile>
```

Abbildung 28: Ausschnitt einer XML-Szenendatei von ANIMATIONPLAYGROUND.

6 Fazit

6.1 Zusammenfassung

In dieser Arbeit wurden die mathematischen und technischen Grundlagen erläutert, welche bei der Entwicklung eines Animationseditors zu beachten sind. Es wurde aufgezeigt, das bei der Repräsentation der zu animierenden Eigenschaften eine Auswahlmöglichkeit besteht. Insbesondere die Orientierung eines Objekts kann auf unterschiedliche Weise realisiert werden. Anhand der Anforderungen, die auch bei nicht computergestützer Animation an die Ergebnisse gestellt werden, wurde die Notwendigkeit zentraler Fähigkeiten eines Animationsprogramms aufgezeigt. Die korrekte Abbildung natürlicher Bewegung, welche entlang von Kurven geschieht, ist die zentrale Anforderung. Daher muss es im computergestützten Editor möglich sein, die Bewegung der Objekte beziehungsweise die Veränderung der Objekteigenschaften mit Hilfe von Kurven zu beschreiben. Nach der Einführung geeigneter Kurvenbeschreibungen wurden dazu zwei Ansätze vorgestellt, welche die Komponente Zeit implizit beziehungsweise explizit modellieren.

Mit Hilfe dieser theoretischen Grundlage und weiterer Erwägungen, die die praktische Nutzung des Editors im Blickpunkt hatten, wurden die Anforderungen an das zu entwickelnde Programm aufgestellt. Auf Basis des ausgewählten impliziten Ansatzes wurde ein Animationseditor entwickelt, der diese Anforderungen umsetzt. Objekte können im Szeneneditor erstellt, und deren Eigenschaften Translation, Rotation und Skalierung animiert werden. Zur Steuerung der Kurven steht ein separater Interpolationseditor zu Verfügung. Erstellte Animationen können in einer XML-Datei gespeichert werden, soweit in Einzelbildern exportiert werden. Es wurde ein Überblick über den Aufbau der Applikation gegeben. Verwendete Hilfsmittel und Grundzüge der Implementierung wurden aufgezeigt. Ausgewählte, mit dem entwickelten Editor erstellte Animation wurden als Beispiel präsentiert.

6.2 Ausblick

Die erstellte Applikation realisiert in kleinem Maßstab die Prinzipien, die auch in verbreiteten Applikationen wie MAYA oder LIGHTWAVE zur grundlegenden Animation Verwendung finden. Die Funktionsvielfalt dieser kommerziellen Programme ist enorm. Das Potential für Erweiterungen des erstellten Programms ist entsprechend umfangreich. Insbesondere mächtigere Werkzeuge zur Bearbeitung von Schlüsselwerten würden einen ergonomischen und schnellen Umgang mit dem Editor fördern (zum Beispiel Mehrfachselektion).

Interessant erscheint die Auseinandersetzung mit der Frage, welcher

Ansatz zur Animation am geeignetsten erscheint. Beide hier vorgestellten Ansätze haben Vor- und Nachteile, und verlangen vom Nutzer umfangreiche Handarbeit, um realistische Bewegungsabläufe zu erstellen. In neueren Ansätzen spielt die Verwendung von *Physiksimulation* eine immer größere Rolle. Mit Hilfe physikalisch basierter Prozesse lassen sich schnell umfangreiche Animationen erstellen, die dennoch realistisch wirken. Es ist absehbar, das in Zukunft realistische, computergenerierte Animationen in immer kürzerer Zeit erstellt werden.

Abbildungsverzeichnis

Tabellenverzeichnis

Liste der Algorithmen

Literatur

[Ben03] **M. Bender, M. Brill:** Computergrafik: Ein anwendungsorientiertes Lehrbuch, Carl Hanser Verlag 2003.

[Ker04] **I. V. Kerlow:** The Art of 3D Computer Animation and Effects, John Wiley & Sons, Inc. 2004.

[Las87] **J. Lasseter:** Principles of traditional animation applied to 3D computer animation, Computer Graphics (Proceedings of SIGGRAPH 87), 21(4), S. 35-44, 1987

[OpenGL05] **D. Shreiner, M. Woo, J. Neider, T.Davis:** OpenGL Programming Guide, 5. Auflage, Addison Wesley 2005.

[Par02] **R. Parent:** Computer Animation: Algorithms and Techniques, Morgan Kaufmann Publishers 2002.

[Shi02] **P. Shirley:** Fundamentals of Computer Graphics, A K Peters, Ltd. 2002.

[wxWidgets05] **K. Hook, J. Smart:** Cross-Plattform GUI Programming with wxWidgets, Pearson Education 2005.

Anhang

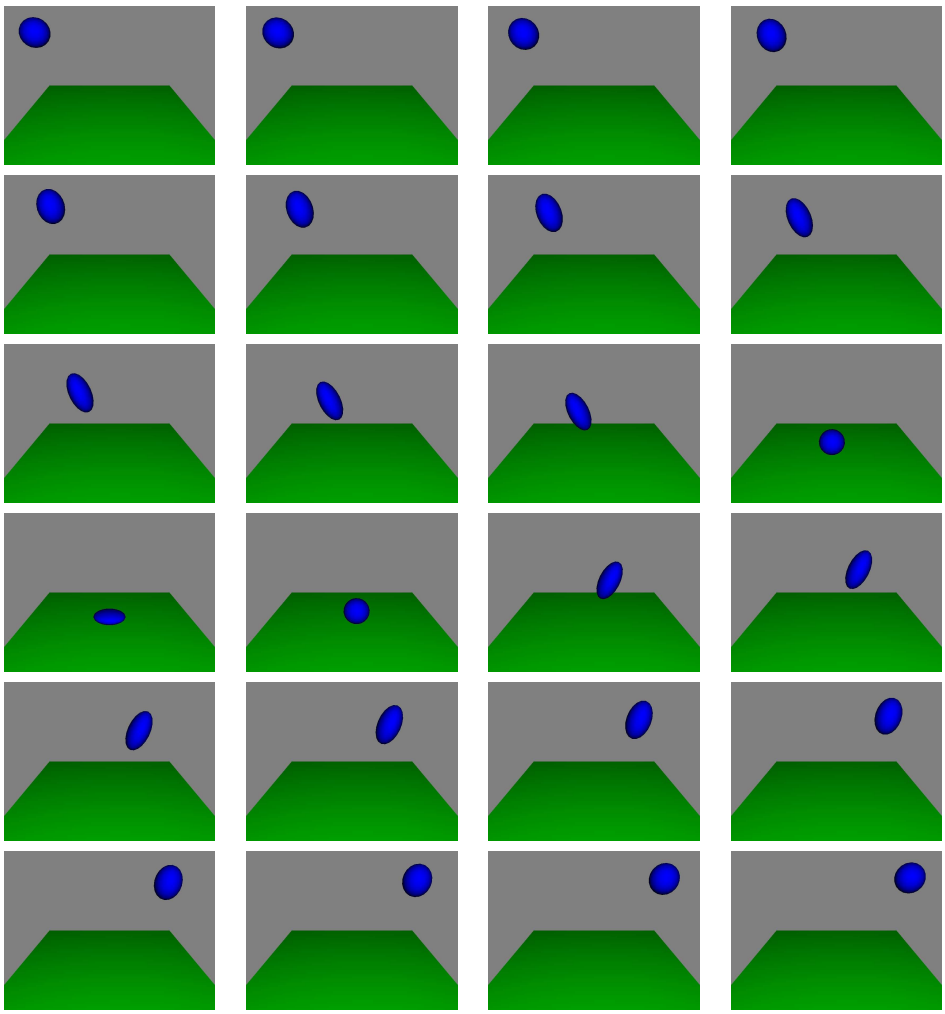

Abbildung 29: Einzelbilder der Beispielanimation SIMPLEANIMATION1.