

Entwicklung eines Kameraeditors Studienarbeit

vorgelegt von
Katharina Hupf



UNIVERSITÄT
KOBLENZ · LANDAU

Institut für Computervisualistik
Arbeitsgruppe Computergraphik

Prüfer: Prof.Dr.Stefan Müller

Oktober 2003

Vorwort

Die *Computergraphik* als Bestandteil der Informatik hat sich seit ihrer Entstehung in den frühen 50er Jahren in drei Stufen entwickelt. In der ersten Entwicklungsstufe ging es um das Darstellen einzelner realistisch-aussehender Bilder auf dem Bildschirm. In der zweiten Stufe ging es um das Entwickeln und Darstellen ganzer Animationen aus mehreren Einzelbildern. Die letzte Stufe und somit Stand der Entwicklung ist die Virtuelle Realität. Sie beschäftigt sich mit der Interaktion eines Benutzers mit der Animation. Die Computeranimation ist somit ein wichtiger und grundlegender Bestandteil der Computergraphik.

Die *Animation* im Allgemeinen ist „ein Verfahren unbelebten Objekten [...] Bewegung zu verleihen“ und stammt aus der Trickfilmtechnik. Dabei nutzt sie die Funktionsweise der menschlichen Wahrnehmung aus. Durch schnell aufeinanderfolgende Einzelbilder entsteht der Eindruck einer kontinuierlichen Bewegung.

Das gleiche Prinzip nutzt die *Computeranimation*. Durch den Benutzer werden die wichtigsten Bilder der Animation, die sogenannten *Keyframes* definiert. Sie sind Momentaufnahmen der Szene. Der Rechner übernimmt dann das „*inbetweening*“. Er berechnet die nötigen Zwischenbilder mittels geeigneter Algorithmen. Dabei muss die Anzahl der Bilder pro Zeit, *Framerate* genannt, geeignet gewählt werden. Ist die Framerate zu gering, nimmt der Betrachter eine Folge einzelne Bilder wahr. Ist sie zu hoch, verwischen die Bilder vor dem Auge des Betrachters.

Die Kameraanimation ist elementarer Bestandteil der Computeranimation. Denn sie bestimmt, was von der Szene alles zu sehen ist. Sie ist das Auge des Betrachters und somit Bezug zu der virtuellen Welt. So muss also für jede Computeranimation auch immer die Kamera mit animiert werden. Dabei verbindet die Kameraanimation zwei wichtige Bestandteile der Computeranimation. Ein Objekt hier die Kamera soll entlang eines Pfades mit einer bestimmten Geschwindigkeit und einer bestimmten Ausrichtung bewegt werden. Dabei sollen Teile der Szene und zwar nur die relevanten - auf dem Bildschirm abgebildet werden.

Inhaltsverzeichnis

1	Einführung	4
2	Anforderungserhebung	5
2.1	Anforderungsdefinition	6
2.2	Datenstrukturen	8
3	Funktionalitäten des Programms	11
3.1	Grundlagen für die implementierten Funktionalitäten	12
3.2	Interpolation der Position	14
3.3	Definition des Geschwindigkeitsverhaltens	19
3.3.1	Interpolation der Geschwindigkeit	20
3.3.2	Parametrisierung über die Bogenlänge	23
3.4	Interpolation der Blickrichtung	26
4	Der entwickelte Kameraeditor	29
5	Zusammenfassung	32
6	Ausblick	34
7	Anhang	37
7.1	Alternative Geschwindigkeitsberechnung	37
7.2	Alternative Bogenlängenparametrisierungen	39
7.3	Alternative Interpolation der Blickrichtung	42
8	Quellenangaben	44

Abbildungsverzeichnis

1.1	Kameradefinition in OpenGL	4
2.1	Definition der Klasse Keyframe	9
2.2	Die KeyframeListe	9
2.3	Definition der Klasse KeyframeList	10
3.1	Interpolierende Kurve	14
3.2	Die Hermite-Interpolation	15
3.3	Eine zusammengesetzte Hermite-Kurve	16
3.4	Unerwünschtes Ergebnis	17
3.5	Erwünschtes Ergebnis	18
3.6	Geschwindigkeits-Zeit-Funktion beim Ease-in	20
3.7	Geschwindigkeits-Zeit-Funktion beim Ease-Out	21
3.8	Ease-In und Ease-Out entlang einer Raumkurve	22
3.9	Approximation der Bogenlänge durch Uniform Sampling	24
3.10	Winkel zwischen den Vektoren	26
3.11	Lineare und Sphärische Interpolation	27
4.1	Das GUI des Kameraeditors	29
4.2	Das 3D-Ausgabefenster	30
4.3	Das Keyframefenster	31
4.4	Das Geschwindigkeitsfenster	31
7.1	Die Sinus-Funktion	37
7.2	Die Weg-Zeit-Funktion	37
7.3	Abstand des Midpoints zu der annähernden Geraden	39
7.4	Test bei der Bestimmung der Midpoint Distanz	40
7.5	Approximation der Bogenlänge mittels der Midpoint Distanz	40
7.6	Die Varianz der Approximation der Kurve	40
7.7	Approximation mittels Bestimmung der Varianz	41
7.8	Sphärische Interpolation mittels vier Vektoren	42

Kapitel 1

Einführung

Ein wichtiger Bestandteil der Computeranimation ist die Kameraanimation. Denn jede Navigation in der Szene ist an eine Kamerabewegung gebunden. Zu diesem Zweck muss die Kamera mit einer bestimmten Geschwindigkeit entlang eines gewünschten Pfades bewegt werden. Sie muss unter Umständen während dieser Bewegung gedreht werden, so dass sie immer in die gewünschte Richtung - auf das „*center of interest*“ - zeigt. Ziel dieser Arbeit ist es,

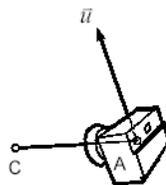


Abbildung 1.1: Kameradefinition in OpenGL

dem Leser einen Einblick in die Thematik der Kameraanimation zu geben und die damit verbundenen Problemstellungen aufzuzeigen. Dies soll an Hand des Kameraeditors geschehen, der im Rahmen dieser Arbeit entstanden ist. Dieser Text soll nun den Verlauf des Projektes und seine Ergebnisse dokumentieren. Die Überlegungen, die zu den entsprechenden Design-Entscheidungen geführt haben, sollen hier zunächst vorgestellt werden (Kapitel 2). Die Algorithmen, die dem Programm zugrunde liegen, sollen erläutert (Kapitel 3) und kritisch reflektiert werden (Kapitel 5).

Kapitel 2

Anforderungserhebung

In diesem Kapitel sollen zunächst einmal die grundlegenden Rahmenbedingungen des Projekts geklärt werden. Die Anforderungen, die vor dem Entwurf des Systems formuliert wurden, sollen hier vorgestellt werden. Sie bilden die Grundlage des Programms und sind für das weitere Verständnis unumgänglich.

Innerhalb des Verlaufes des Projekts erfolgte zunächst eine Anforderungsanalyse. Dabei wurden die Anforderungen an das System erhoben und definiert. Die Ergebnisse dieser Anforderungsanalyse werden in Abschnitt 2.1 vorgestellt. Die Anforderungsdefinition legt fest, was das Programm letztendlich alles können muss.

Des Weiteren werden in Abschnitt 2.2 die benötigten Datenstrukturen vorgestellt. Sie repräsentieren die Daten, auf denen das Programm arbeitet.

2.1 Anforderungsdefinition

Im Rahmen einer Anforderungsanalyse wurde geklärt, was der zu programmierende Kameraeditor alles leisten soll. Auf der Basis der so erschlossenen Anforderungen und unter Einbezug der Datenstrukturen wurde dann das „wie“ geklärt (siehe Kapitel 3). Die Anforderungsdefinition und die Festlegung der Datenstrukturen bilden somit die Grundlage für die zu implementierende Funktionalität.

Der Kameraeditor soll aus dem ihm übergebenen Keyframes eine Kameraanimation erstellen. Dazu übernimmt das Programm das inbetweening. Es berechnet die Zwischenbilder für die Animation. Die somit zur Erstellung der Kamerafahrt relevanten Daten müssen vom Benutzer spezifiziert werden. Für diese Interaktion wurde ein *Graphical User Interface (GUI)* implementiert. Über diese Benutzerschnittstelle erfolgt die Steuerung des Programms.

Das GUI setzt sich aus drei Bestandteilen zusammen: Einem Fenster zur Darstellung der Szene, einem Fenster zur Eingabe und Veränderung der Keyframewerte, und einem Fenster zur Festlegung der Geschwindigkeiten für die einzelnen Keyframes.

Für die einzelnen Komponenten des Systems können die Anforderungen dann wie folgt definiert werden:

Das 3D-Ausgabefenster

- In diesem Fenster wird die Szene gezeichnet (hier nur ein Test-Setting).
- Es gibt einen Navigationsmodus zur freien Navigation in der 3D-Szene.
- Es gibt einen Eingabemodus. Durch eine bestimmte Tastenkombination (CTRL+N) können Keyframes eingefügt werden. Der Keyframe wird immer hinter den aktuellen Keyframe angefügt. Die Attributwerte entsprechen der Kamerakonfiguration zum Zeitpunkt der Eingabe.
- Die Keyframes können in der Szene gezeichnet und auch wieder ausgeblendet werden.
- Die erstellte Kameraanimation wird in diesem Fenster ausgegeben.

Das Keyframefenster

- Dieses Fenster zeigt immer das Datenblatt des aktuellen Keyframes an.
- Es können Keyframes hinzugefügt werden. Der Keyframe wird dann immer hinter den aktuellen Keyframe angefügt.

- Die Eingabe folgender Werte ist in diesem Fenster möglich:
 - die Position der Kamera
 - das „center of interest“ oder auch Blickpunkt genannt
 - die Geschwindigkeit
 - die Länge der Tangente (siehe Kapitel 3.2)
- Jeder dieser Werte muss nachträglich verändert werden können.
- Der Benutzer kann zwischen den Datenblättern der Keyframes blättern.
- Keyframes können auch wieder entfernt werden.

Das Geschwindigkeitsfenster

- In diesem Fenster kann der Benutzer die Geschwindigkeit für die einzelnen Keyframes festlegen.
- Für einen Keyframe kann die Geschwindigkeit spezifiziert und auch nachträglich wieder verändert werden.

Allgemeine Anforderungen

- Die bereits spezifizierten Werte können in einer Datei abgespeichert werden.
- Ebenso können Keyframewerte aus einer Datei ausgelesen werden.

2.2 Datenstrukturen

Im Interesse dieser Arbeit steht die Entwicklung eines Programms, das es dem Benutzer erlaubt, interaktiv eine Kameraanimation zu spezifizieren. Dazu muss der Benutzer Daten zur Steuerung der Animation eingeben. Welche Daten hierbei von dem Benutzer gefordert werden, hängt direkt von der Anwendung ab.

Eine Kamera soll entlang eines Pfades bewegt werden. Zwischen den Keyframes soll also ihre Position interpoliert werden.

Sie soll mit einer bestimmten Geschwindigkeit diesem Kamerapfad folgen. Dies erfordert die Spezifikation von Geschwindigkeitswerten für jeden Keyframe.

Da die Kamera festlegt, welche Teile der Szene abgebildet werden, muss auch die Richtung, in die sie zeigen soll, spezifiziert werden. Dies entspricht der Definition eines „center of interest“, hier auch Blickpunkt genannt, auf den die Kamera blicken soll.

Die Spezifikation der Tangente dient der Steuerung der Form des Kamerapfades und wird unter Abschnitt 3.2 erläutert.

Die Key-Nummer dient der Übersicht. Sie gibt die Position des Keyframes innerhalb der Folge von Keyframes an. Sie ist durch den Benutzer nicht veränderbar, sondern wird einzig und allein durch die Position des Keyframes in der Liste definiert.

Daraus ergibt sich, dass für jeden Keyframe folgende Daten spezifiziert werden müssen:

- Key-Nummer
- Position
- Länge der Tangente
- Blickpunkt (Center of interest)
- Geschwindigkeit

Da sowohl die Position als auch der Blickpunkt ein Punkt im dreidimensionalen Raum sind, umfasst ihre Definition jeweils eine x-, y- und z-Koordinate. Das lässt dann auf die Klassendefinition für einen Keyframe aus Abb.2.1 schließen. Für die Erstellung der Kameraanimation ist es erforderlich, dass mehrere, zumindest jedoch zwei Keyframes spezifiziert worden sind. Es wird daher eine Datenstruktur zur Ablage der Keyframes benötigt.

Die Kamera soll die Keyframes in der Reihenfolge passieren, in der sie von

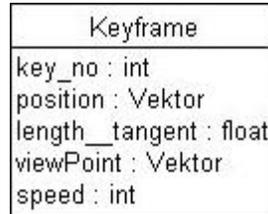


Abbildung 2.1: Definition der Klasse Keyframe

dem Benutzer spezifiziert worden sind. Dabei ist hier nicht die zeitliche Abfolge der Bearbeitung, sondern die Reihenfolge der Sortierung durch den Benutzer gemeint. Die hier verwendete Datenstruktur ist eine Liste. Sie hat gegenüber einem Array den Vorteil, dass sie bzgl. ihrer Elemente flexibler zu handhaben ist. So ist mit Hilfe einer Liste z.B. das Einfügen oder Löschen eines Keyframes mit weniger Aufwand umzusetzen.

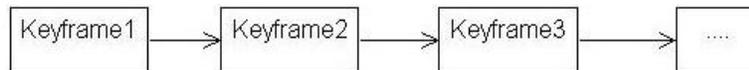


Abbildung 2.2: Die KeyframeListe

Die Keyframes werden in einer Liste abgelegt. Zur Realisierung der in Abschnitt 2.1 genannten Anforderungen werden folgende Operationen auf der Liste zur Verfügung gestellt.

Es muss zunächst mittels des Konstruktors *KeyframeList()* eine leere Liste angelegt werden. Über die Methode *newKeyframe()* können Keyframes in die Liste eingefügt werden. Der Keyframe wird dabei immer hinter den aktuellen Keyframe, das sogenannte *currentItem*, in die Liste eingefügt. Das Löschen eines Keyframes erfolgt über die Methode *deleteKeyframe()*. Über die Methoden *first()*, *last()*, *next()* und *end()* ist eine Navigation innerhalb der Liste möglich. Es kann somit zum ersten, vorherigen, nächsten und letzten Element in der Liste gewechselt werden.

Dies führt zu der Klassendefinition für die Liste aus Abb.2.3.

Die KeyframeList und die in ihr enthaltenen Keyframes sind die grundlegenden Datenstrukturen des Programms. Auf ihnen arbeiten die Algorithmen zur Erstellung der Kameraanimation. Ihre Struktur lässt sich direkt aus der Anwendung ableiten. Ihre Werte werden vom Benutzer selbst spezifiziert und

KeyframeList
currentItem
KeyframeList() newKeyframe() deleteKeyframe() first() last() next() end()

Abbildung 2.3: Definition der Klasse KeyframeList

bestimmen Form und zeitlichen Verlauf der Kamerafahrt.

Kapitel 3

Funktionalitäten des Programms

Neben den allgemeinen Anforderungen an das Programm wie z.B. das Öffnen und Speichern von Dateien soll das Programm in erster Linie eine Kamerafahrt aus den Keyframes, die der Benutzer eingegeben hat, erstellen.

Die Kamerafahrt wird durch mehrere Funktionalitäten realisiert. Die Algorithmen, die diesen Funktionalitäten zugrunde liegen, sollen in den folgenden Abschnitten näher erläutert werden.

Das Problem der Kameraanimation lässt sich in folgende Funktionalitäten aufspalten:

- **Interpolation der Position**

Die Kamera soll entlang eines Pfades bewegt werden. Dies entspricht einer Interpolation der Kameraposition.

- **Definition des Geschwindigkeitsverhaltens**

Um die Geschwindigkeit der Kamera kontrollieren zu können, muss festgelegt werden, wann sich die Kamera wo auf der Kurve befindet. Dazu muss zwischen den Geschwindigkeiten an den Keyframes interpoliert werden. In Abhängigkeit von der Geschwindigkeit kann mittels Integration der zurückgelegte Weg berechnet werden. Dem zurückgelegten Weg muss dann eine Position auf der Kurve zugeordnet werden. Dies geschieht mittels einer Parametrisierung über die Bogenlänge

- **Interpolation der Blickrichtung**

Für jeden Punkt auf der Kurve muss der Blickpunkt, das sogenannte „center of interest“, berechnet werden. Dies geschieht mit Hilfe der Interpolation der Blickrichtung.

3.1 Grundlagen für die implementierten Funktionalitäten

Um die Verständlichkeit und Lesbarkeit des Textes zu erhöhen, soll an dieser Stelle zunächst eine Einführung in die mathematischen Grundlagen gegeben werden. Des Weiteren soll die hier verwendete Terminologie erläutert werden. Dies ist erforderlich, da sich die Bewegung der Kamera durch zwei verschiedene Parameter beschreiben lässt. Zum einen ist sie als Bewegung im Raum zu betrachten. Zum anderen ist sie aber auch als Veränderung der Position über die Zeit zu sehen. Aus diesem Grund müssen zur Lösung des Problems der räumliche so wie der zeitliche Aspekt der Kameraanimation diskutiert werden. Beide Sachverhalte müssen klar gegeneinander abgegrenzt werden. Der Pfad, dem die Kamera während der Animation folgt, ist im mathematischen Sinne eine Kurve im dreidimensionalen Raum. Es ist somit ein elementarer Aspekt dieser Arbeit, sich mit Kurven im R^3 zu beschäftigen. Grundsätzlich kann man Kurven auf verschiedene Weisen darstellen. Die hier gewählte Form ist die *Parameterform*: Eine dreidimensionale parametrische Kurve der Form $C(u)=(x(u),y(u),z(u))$. Die x-, y- bzw. z-Koordinate eines jeden Punktes auf der Kurve können in Abhängigkeit von u berechnet werden. Der Parameter u ist aus einem Intervall [a,b], meist (wie hier auch) aus dem Intervall [0,1].

Wie oben erwähnt, ist der Kamerapfad eine Kurve im R^3 , die mittels eines Parameters u ausgedrückt wird. Andererseits will man in diesem Zusammenhang aber die Kamera mit der Zeit entlang eines Weges bewegen. Folglich lässt sich die Kameraposition durch zwei verschiedene Parameter ausdrücken. Aus diesem Grund wird im folgenden Text zwischen der *Raumkurve* und der *Weg-Zeit-Funktion* unterschieden.

Die *Raumkurve* spezifiziert den Pfad im R^3 , dem das Objekt folgen soll. Sie wird durch Interpolation der Position der Keyframes ermittelt. Sie legt Form und Verlauf des Kamerapfades fest.

Die *Weg-Zeit-Funktion* hingegen setzt die Zeit mit dem zurückgelegten Weg in Relation. Sie ordnet jedem Zeitpunkt t die bis dahin zurückgelegte Wegstrecke s zu.

Um das Problem der Kamerabewegung entlang eines gewünschten Pfades zu lösen, muss man also eine Funktion finden, die einem Zeitwert t eine Position auf der Raumkurve zuordnet.

Die zurückgelegte Wegstrecke entlang der Kurve wird *Bogenlänge*, kurz s, bezeichnet. Sie entspricht der Länge des Weges auf der Kurve.

Auch der zurückgelegte Weg lässt sich sowohl in Abhängigkeit vom Kurvenparameter u als auch der Zeit t bestimmen.

Soll der zurückgelegte Weg s in Abhängigkeit von der Zeit t berechnet werden, so wird dies mit $s(t)$ notiert. Dies entspricht der oben erläuterten Weg-Zeit-Funktion. Sie ordnet jedem Zeitwert t den in der Zeit zurückgelegten Weg s zu. Sie beschreibt somit das Geschwindigkeitsverhalten der Kamera.

Soll die Bogenlänge allerdings in Abhängigkeit der parametrischen Variablen u ausgedrückt werden, so wird dies mit $s(u)$ notiert.

Für die Kameraanimation ist es erforderlich, jedem Zeitwert t eine Position auf der Raumkurve zuzuordnen. Dies ist durch eine Weg-Zeit-Funktion der Form $s(t)$ nicht möglich, da der Kamerapfad durch eine Raumkurve gegeben ist, die ausschließlich über die Variable u ausgedrückt wird. Daher muss der Zusammenhang zwischen dem Parameterwert u und dem zurückgelegten Weg s geklärt werden.

Sei die zu einem bestimmten Zeitpunkt t zurückgelegte Wegstrecke s bekannt. Um den entsprechenden Punkt auf der Kurve berechnen zu können, wird der entsprechende Kurvenparameter u benötigt. Es muss also eine Funktion hergeleitet werden, die jeder Bogenlänge s einen Parameterwert u zuordnet. Die Herleitung einer solchen Abbildung heißt *Parametrisierung über die Bogenlänge*.

3.2 Interpolation der Position

Grundlegender Bestandteil der Kameraanimation ist es also, den Pfad festzulegen, dem die Kamera folgen soll. Da dieser Pfad einer Kurve im dreidimensionalen Raum entspricht, muss man zunächst eine geeignete Kurvenform finden, die den Ansprüchen der Anwendung gerecht wird.

Die mathematische Repräsentation der Kurve sollte möglichst einfach zu handhaben sein, so dass sie mit möglichst geringem Aufwand zu berechnen ist. Dennoch sollte der Kurvenverlauf den Anforderungen der Anwendung genügen.

Durch den Benutzer ist eine Folge von n Punkten P_0, \dots, P_{n-1} gegeben. Die Kamera soll während ihrer Kamerafahrt auch alle diese Punkte in der vorgegebenen Reihenfolge passieren soll. Daher wird hier eine *interpolierende* Kurve gesucht, denn eine approximierende Kurve würde die Punkte nur möglichst genau annähern.

Für n Keyframes kann man ein interpolierendes Polynom $(n-1)$ -ten Grades

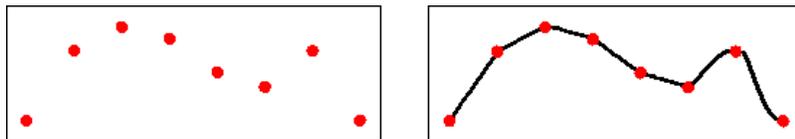


Abbildung 3.1: Interpolierende Kurve

konstruieren. Das würde bedeuten, dass schon bei relativ wenigen Punkten Polynome hohen Grades verwendet würden. Da dies aber auch eine hohe Komplexität mit sich zieht, setzt man die Kurve aus mehreren Kurvenstücken geringeren Polynomgrades zusammen. Statt einem Polynom $(n-1)$ -ten Grades verwendet man eine aus mehreren Kurvenstücken zusammengesetzte Kurve geringeren Grades. Eine solche zusammengesetzte Kurve nennt man *Spline*. Grundlegende Bedingung ist es, dass die Kamerafahrt als eine *kontinuierliche* Bewegung wahrgenommen wird. Dies erzwingt von der Konstruktion der Kurve, dass die einzelnen Kurvensegmente „weich“ ineinander übergehen. Es darf also keine Sprünge in den Werten zwischen den einzelnen Kurvensegmenten geben.

Bei dem zu erstellenden Kameraeditor soll es sich um ein interaktiv steuerbares Programm handeln. Bis zu der Festlegung des endgültigen Kamerapfades kann es unter Umständen mehrfach vorkommen, dass ein einzelner Punkt verändert werden soll. Aus diesem Grunde ist es wünschenswert, dass in diesem Fall nicht die komplette Kurve neu berechnet werden muss, sondern nur der zu verändernde Teil. Dies nennt man *lokale Kontrolle*. Der einzelne Punkt

beeinflusst den Kurvenverlauf einzig und allein in einer bestimmten Region. Das hier verwendete Verfahren, das all diese Kriterien erfüllt, ist die *Hermite-Interpolation*.

Die Hermite-Interpolation generiert ein kubisches Polynom zwischen zwei Punkten P_0 und P_1 . Ein kubisches Polynom hat die Form

$$f(u) = a \cdot u^3 + b \cdot u^2 + c \cdot u + d \quad (3.1)$$

Da ein kubisches Polynom vier Koeffizienten besitzt, werden zur Lösung des linearen Gleichungssystems mit den vier Unbekannten a, b, c und d weitere Daten benötigt. Die Hermite-Interpolation fordert daher zusätzlich die Angabe der Start- und der Endtangente T_0 und T_1 des Kurvenstückes.

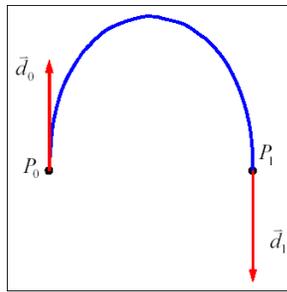


Abbildung 3.2: Die Hermite-Interpolation

Für das Kurvenstück soll Folgendes gelten. Die Kurve soll im Punkt P_0 beginnen und im Punkt P_1 enden. Daraus folgt, dass $C(0) = P_0$ und $C(1) = P_1$. Die Tangentenvektoren sind durch die erste Ableitung der Kurve gegeben. Somit soll ebenfalls gelten $C'(0) = T_0$ und $C'(1) = T_1$.

Dies führt zu folgendem linearen Gleichungssystem:

$$a \cdot 0^3 + b \cdot 0^2 + c \cdot 0 + d = P_0$$

$$a \cdot 1^3 + b \cdot 1^2 + c \cdot 1 + d = P_1$$

$$3a \cdot 0^2 + 2b \cdot 0^1 + c = T_0$$

$$3a \cdot 1^2 + 2b \cdot 1^1 + c = T_1$$

Somit ergeben sich folgende Lösungen:

$$a = 2 \cdot P_0 - 2 \cdot P_1 + T_0 + T_1$$

$$b = -3 \cdot P_0 + 3 \cdot P_1 - 2 \cdot T_0 - T_1$$

$$c = T_0$$

$$d = T_1$$

Ein Einsetzen der Lösungen in die Gleichung 3.1 ergibt:

$$C(u) = (2 \cdot P_0 - 2 \cdot P_1 + T_0 + T_1) \cdot u^3 + (-3 \cdot P_0 + 3 \cdot P_1 - 2 \cdot T_0 - T_1) \cdot u^2 + T_0 \cdot u + T_1$$

Mittels Umformungen erhält man dann folgende Gleichung zur Darstellung der Kurve:

$$C(u) = f_0(u) \cdot P_0 + f_1(u) \cdot P_1 + f_2(u) \cdot T_0 + f_3(u) \cdot T_1$$

mit

$$f_0(u) = (2 \cdot u^3 - 3 \cdot u^2 + 1)$$

$$f_1(u) = (-2 \cdot u^3 + 3 \cdot u^2)$$

$$f_2(u) = (u^3 - 2 \cdot u^2 + u)$$

$$f_3(u) = (u^3 - u^2)$$

Damit wird aber noch nicht der vollständige Kamerapfad beschrieben, sondern erst zwei Punkte miteinander verbunden. Die vollständige Kurve muss bei n Keyframes aus $(n-1)$ Kurvenstücken zusammengesetzt werden. Um einen kontinuierlichen Kurvenverlauf zu gewährleisten, wird folgendes für die Wahl der Tangenten vorausgesetzt: Die Endtangente des einen Kurvensegments wird als Starttangente des darauffolgenden Segments verwendet.

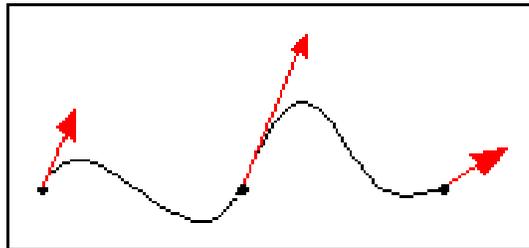


Abbildung 3.3: Eine zusammengesetzte Hermite-Kurve

Somit lässt sich jedes Kurvensegment i , mit $0 \leq i < (n-1)$, wie folgt definieren:

$$C_i(u) = f_0(u) \cdot P_i + f_1(u) \cdot P_{i+1} + f_2(u) \cdot T_i + f_3(u) \cdot T_{i+1}$$

Die Richtungen der Tangente wurden hier so gewählt, dass sie den Blickrichtungen im Keyframe entsprechen. Ist also B_i der Blickpunkt im Keyframe P_i , so lässt sich die Tangente wie folgt festlegen:

$$T_i = P_i \vec{B}_i = B_i - P_i$$

Ein grundlegendes Prinzip der Hermite-Interpolation ist die Angabe der Start- und der Endtangente eines jeden Kurvensegments. Durch die Tangenten ist der Kurvenverlauf bzgl. seiner Richtung und seiner Krümmung steuerbar.

Die Länge der Tangente bestimmt die Krümmung der Kurve. Die Tangentenrichtung bestimmt die Richtung der Kurve im Keyframe. Diese Eigenschaft war auch ausschlaggebend für die Wahl der Hermite-Interpolation als Verfahren zur Bestimmung der Kurve. Der entscheidende Vorteil, der sich daraus ergibt, ist die Möglichkeit, den Verlauf der Kurve zu kontrollieren.

Bei einer Kamerafahrt wird die Kamera durch eine Szene bewegt. Hierbei können unerwünschte Effekte entstehen. So kann es z.B. vorkommen, dass die Kamera das Terrain verlässt oder Objekte der Szene durchfliegt.

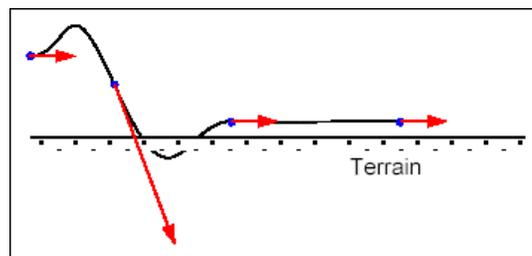


Abbildung 3.4: Unerwünschtes Ergebnis

Während die Richtung der Tangente auch die Richtung der Kurve im Keyframe definiert, so legt die Länge der Tangente die Krümmung der Kurve fest. Je länger also die Tangente ist, um so größer ist die Ausbuchtung. Aus diesem Grund ist bei dem hier entwickelten Kameraeditor die Länge der Tangente ein Parameter, den der Benutzer interaktiv steuern kann. Dies erlaubt es dem Benutzer, den Kamerapfad an die individuellen Begebenheiten der Szene anzupassen.

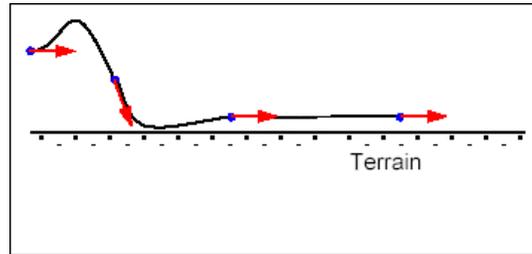


Abbildung 3.5: Erwünschtes Ergebnis

Mittels Hermite wird also die Position der Keyframes interpoliert. Auf diese Art und Weise wird eine Kurve im dreidimensionalen Raum zur Darstellung des Kamerapfades berechnet. Der Kamerapfad kann nun mittels einer solchen Kurve spezifiziert werden.

3.3 Definition des Geschwindigkeitsverhaltens

Da die Kamera nicht nur mit konstanter Geschwindigkeit bewegt werden soll, ist es erforderlich, das Geschwindigkeitsverhalten der Kamera genauer zu betrachten. Um die Geschwindigkeit kontrollieren zu können, muss herausgefunden werden, wann sich die Kamera wo auf der Kurve befindet. Dazu muss jedem Zeitpunkt t eine Position auf der Kurve zugeordnet werden. Durch den Benutzer sind Geschwindigkeiten, mit denen die Kamera die Keyframes passieren soll, definiert worden. Um das Geschwindigkeitsverhalten der Kamera entlang der Kurve beschreiben zu können, muss zunächst zwischen diesen Geschwindigkeitswerten interpoliert werden. So erhält man eine Geschwindigkeits-Zeit-Funktion $v(t)$. Sie definiert für jeden Zeitpunkt t die Momentangeschwindigkeit der Kamera. Von dieser Funktion $v(t)$ kann die Weg-Zeit-Funktion $s(t)$ abgeleitet werden.

Dies ist der erste Schritt, um die Kamerabewegung kontrollieren zu können. Der zeitliche Aspekt der Kamerabewegung kann so formuliert werden. Jedem Zeitpunkt kann eine zurückgelegte Strecke s zugeordnet werden. Um die Position, also das „wo“ auf der Kurve definieren zu können, muss der zugehörige Kurvenpunkt zu dieser Bogenlänge s gefunden werden. Die Punkte auf der Kurve sind durch den Parameter u und nicht durch die Bogenlänge s definiert. Daher muss eine Funktion gefunden werden, die für jede Bogenlänge s einen Parameterwert u liefert. Mit Hilfe dieses Parameters u kann dann der entsprechende Punkt auf der Kurve berechnet werden.

3.3.1 Interpolation der Geschwindigkeit

Durch den Benutzer sind für jeden der n Keyframes Geschwindigkeitswerte v_0, \dots, v_{n-1} festgelegt worden. Mit dieser Geschwindigkeit soll die Kamera während der Animation den Keyframe passieren. Um eine kontinuierliche Bewegung zu erhalten, muss zwischen diesen Werten interpoliert werden.

Dabei kann man zwei Fälle voneinander unterscheiden. Die Kamera soll zwischen zwei Keyframes beschleunigt werden oder aber die Kamera soll abgebremst werden; auch *Ease-in* bzw. *Ease-out* genannt.

Das *Ease-in* beschreibt die beschleunigte Kamerabewegung zwischen zwei Keyframes. Um „weichere“ Übergänge an den Keyframes zu erhalten, wird die Kamera nur innerhalb eines Zeitintervalls $[t_1, t_2]$ beschleunigt. Außerhalb dieses Intervalls wird die Kamera mit konstanter Geschwindigkeit bewegt.

Nachdem die Kamera mit der Geschwindigkeit v_i den Keyframe passiert hat, wird sie eine Zeit lang mit dieser konstanten Geschwindigkeit weiterbewegt. Anschließend wird sie beschleunigt bis sie die gewünschte Geschwindigkeit v_{i+1} erreicht hat. Bis zum Passieren des Keyframes bewegt sie sich mit konstanter Geschwindigkeit weiter.

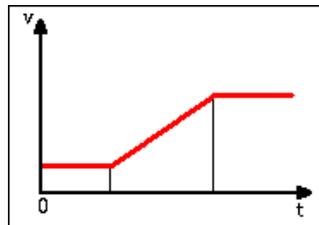


Abbildung 3.6: Geschwindigkeits-Zeit-Funktion beim *Ease-in*

Das *Ease-out* beschreibt das Abbremsen der Kamerabewegung. Nachdem die Kamera mit der Geschwindigkeit v_i den Keyframe passiert hat, wird zeitverzögert mit dem Vorgang des Abbremsens begonnen. Hat die Kamera die vorgegebene Geschwindigkeit v_{i+1} erreicht, wird sie mit konstanter Geschwindigkeit bis zum Erreichen des Keyframes fortbewegt. Der Vorgang ist analog zum *Ease-in*, nur dass die Kamera abgebremst statt beschleunigt wird. Aus diesem Zusammenhang ergibt sich, dass die Kamera zu jedem Zeitpunkt eine Momentangeschwindigkeit $v(t)$ hat, die nur von der Zeit abhängig ist.

Sei v_i die Geschwindigkeit am i -ten Keyframe und v_{i+1} die Geschwindigkeit am darauffolgenden Keyframe, so lässt sich die Geschwindigkeit $v(t)$ für das

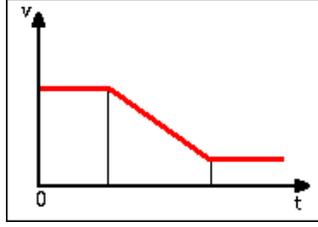


Abbildung 3.7: Geschwindigkeits-Zeit-Funktion beim Ease-Out

Ease-in wie folgt definieren:

$$v(t) = \begin{cases} v_i & \text{für } t < t_1 \\ v_i + \frac{t}{t_1} \cdot (v_{i+1} - v_i) & \text{für } t_1 \leq t \leq t_2 \\ v_{i+1} & \text{für } t > t_2 \end{cases}$$

Auf Grund der physikalischen Zusammenhänge kann der in der Zeit zurückgelegte Weg abgeleitet werden. Ist die Geschwindigkeit als Funktion der Zeit gegeben, so kann durch Integration der Weg als Funktion der Zeit berechnet werden:

$$s(t) = \int_0^t v(t) dt$$

Dies ergibt dann als Weg-Zeit-Funktion beim Ease-in:

$$s(t) = \begin{cases} v_i \cdot t & \text{für } t < t_1 \\ v_i \cdot t + \frac{v_{i+1} - v_i}{2 \cdot t_1} \cdot (t^2 - t_1^2) & \text{für } t_1 \leq t \leq t_2 \\ v_i \cdot t_2 + \frac{v_{i+1} - v_i}{2 \cdot t_1} \cdot (t_2^2 - t_1^2) + v_{i+1} \cdot (t - t_2) & \text{für } t > t_2 \end{cases}$$

Analoges Vorgehen ergibt folgende Gleichungen für die Geschwindigkeit und den zurückgelegten Weg beim Ease-out:

$$v(t) = \begin{cases} v_i & \text{für } t < t_1 \\ v_i \frac{t-t_1}{t_2-t_1} \cdot (v_{i+1} - v_i) & \text{für } t_1 \leq t \leq t_2 \\ v_{i+1} & \text{für } t > t_2 \end{cases}$$

$$s(t) = \begin{cases} v_i \cdot t & \text{für } t < t_1 \\ v_i \cdot t + \frac{v_{i+1} - v_i}{2 \cdot (t_2 - t_1)} \cdot (t^2 - t_1^2) & \text{für } t_1 \leq t \leq t_2 \\ v_i \cdot t_2 + \frac{v_{i+1} - v_i}{2} \cdot (t_2 + t_1) + v_{i+1} \cdot (t - t_2) & \text{für } t > t_2 \end{cases}$$

Folgende Sonderfälle sind zu betrachten. Ist die Startgeschwindigkeit beim Ease-in gleich 0, so würde die Kamera in dem Zeitintervall $[0, t_1]$ nicht bewegt

werden. Aus diesem Grund wird in dem Fall $v_i = 0$ die Zeitgrenze $t_1 = 0$ gesetzt. Gleiches gilt für das Ease-out. Ist dort die Endgeschwindigkeit $v_{i+1} = 0$, so wird der Schwellwert $t_2 = 0$ gesetzt.

Ist die Geschwindigkeit für jeden der n Keyframes gegeben, so kann eine Funktion erstellt werden, die die Geschwindigkeit der Kamera zwischen zwei Keyframes interpoliert. Damit ist für jeden Zeitpunkt t eine Momentangeschwindigkeit $v(t)$ definiert. Mittels Integration kann von dieser Funktion eine Weg-Zeit-Funktion $s(t)$ abgeleitet werden. Sie ordnet jedem Zeitwert t die Wegstrecke s zu, die bis dahin zurückgelegt wurde.

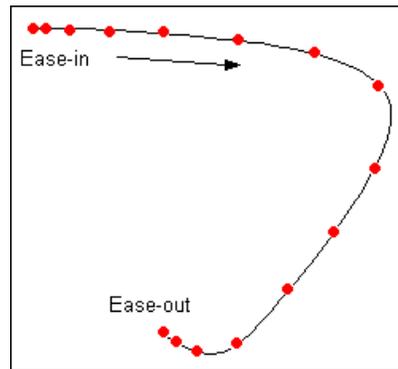


Abbildung 3.8: Ease-In und Ease-Out entlang einer Raumkurve

Mit Hilfe dieser Weg-Zeit-Funktion $s(t)$ kann jedem Zeitpunkt t der zurückgelegte Weg s zugeordnet werden. Um nun das Geschwindigkeitsverhalten der Kamera kontrollieren zu können, muss eine Lösung gefunden werden, die dem zurückgelegten Weg s entlang der Kurve einen Punkt auf der Kurve zuordnet. So wäre es dann möglich, für jeden Zeitpunkt t die Position der Kamera auf der Kurve über die Bogenlänge s zu bestimmen.

3.3.2 Parametrisierung über die Bogenlänge

Um die Bewegung der Kamera entlang der Kurve zu kontrollieren, muss bestimmt werden, wann sich die Kamera wo auf der Kurve befindet. Die Weg-Zeit-Funktion $s(t)$ berechnet für jeden Zeitpunkt t den Weg, den die Kamera bis dahin zurückgelegt hat. Um dem „wann“ das „wo“ auf der Kurve zuordnen zu können, muss für jede Bogenlänge s der entsprechende Punkt auf der Kurve bestimmt werden. Da die Kurve mittels des Parameters u repräsentiert wird, muss der Zusammenhang zwischen dem Parameter u und der Bogenlänge s somit formuliert werden.

Es gibt unterschiedliche Ansätze, die Kurve über die Bogenlänge neu zu parametrisieren.

Eine Möglichkeit ist es, die Bogenlänge auf *analytischem* Wege zu berechnen. Die Länge einer Kurve zwischen zwei Punkten $C_1(u)$ und $C_2(u)$ kann dann gelöst werden, indem man das Integral zur Bestimmung der Bogenlänge näher betrachtet:

$$s = \int_{u_1}^{u_2} |dC/du| du$$

Dieses Integral ist allerdings äußerst komplex ist. Es lässt sich nur mit sehr viel Aufwand - für manche Kurvenformen sogar überhaupt nicht - berechnen. Eine Möglichkeit zur Lösung sind da *numerische* Verfahren. Die Lösung des Integrals wird mit Hilfe numerischer Methoden abgeschätzt.

Bei dem hier vorgestellten Kameraeditor wurde ein dritter Ansatz gewählt. Die Kurve wird durch Geraden angenähert. Vorteile dieser Methode sind der einfache Algorithmus, der diesem Konzept zugrunde liegt, und der geringe Rechenaufwand. Da die Ergebnisse zufriedenstellend waren, wurde von der Implementierung einer anderen „genaueren“ Methode, die immer auch im Trade-Off mit mehr Aufwand zu sehen ist, abgesehen. Die Abweichungen sind je nach Feingliedrigkeit der Unterteilung vertretbar. Hierbei gilt der Grundsatz, je feiner die Unterteilung, desto geringer die Abweichung, aber auch umso größer der Rechenaufwand.

Bei dem hier gewählten Verfahren wird wie folgt eine Wertetabelle konstruiert:

- Der Parameter u wird von 0 bis 1 in n äquidistante Intervalle unterteilt.
- Für jeden Wert für u wird der zugehörige Punkt auf der Kurve $C(u)$ bestimmt.
- Die Geraden zwischen diesen Punkten werden berechnet.

- Die Bogenlänge jedes Punktes wird berechnet. Dabei berechnet sich die Bogenlänge eines Punktes als die Summe der Geraden, die die Kurve annähern, indem sie alle seine Vorgänger miteinander verbinden.
- In einer Tabelle werden dann jeweils der Wert für u und der Wert für die Bogenlänge s gespeichert.

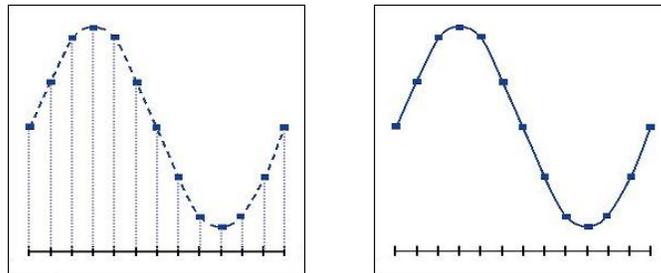


Abbildung 3.9: Approximation der Bogenlänge durch Uniform Sampling

Parameterwert	Bogenlänge
0.0	s_0
0.1	s_1
0.2	s_2
0.3	s_3
0.4	s_4
0.5	s_5
0.6	s_6
0.7	s_7
0.8	s_8
0.9	s_9
1.0	s_{10}

Tabelle 3.1: Parameter-Bogenlängen-Wertetabelle

Nach dieser Vorberechnung steht nun eine Liste zur Verfügung, die jeweils den Parameterwert u als auch die Bogenlänge s verzeichnet.

In der Anwendung der Kameraanimation ist also die Bogenlänge s bekannt und der zugehörige Parameterwert u wird gesucht. Also wird in dieser Tabelle nach dem entsprechenden Eintrag für s gesucht, dessen Wert für die Bogenlänge die geringste Differenz zu dem gesuchten Wert hat. Der entsprechende Eintrag für den Parameterwert u wird dann zurückgegeben und steht

somit für die weitere Berechnung zur Verfügung.

Mit Hilfe von diesem Verfahren kann der Bogenlänge s ein Parameterwert u zugeordnet werden. In Hinsicht auf die Kameraanimation bedeutet dies, dass die Kamera beschleunigt oder abgebremst entlang der Kurve bewegt werden kann.

3.4 Interpolation der Blickrichtung

Bei der Kameraanimation soll die Kamera während der Animation mit einer bestimmten Geschwindigkeit entlang einer Kurve bewegt werden. Der Benutzer hat aber auch die Möglichkeit, für jeden Keyframe ein „center of interest“ festzulegen. Er kann also einen Punkt bestimmen, auf den die Kamera zu einem bestimmten Zeitpunkt zeigen soll. Das bedeutet, dass zwischen diesen Blickpunkten ebenfalls interpoliert werden muss. Also muss für jeden vorher berechneten Parameterwert u der Punkt bestimmt werden, auf den die Kamera zeigt.

Dazu müssen zunächst für jeden der n Keyframes n Blickpunkte B_0, \dots, B_{n-1} definiert werden sein.

Mit Hilfe dieser n Blickpunkte können dann n Vektoren D_0, \dots, D_{n-1} für die Blickrichtung berechnet werden:

$$D_i = \vec{P_i B_i} = B_i - P_i$$

Interpoliert werden soll nun der Winkel θ_i zwischen den Vektoren D_i und D_{i+1} .

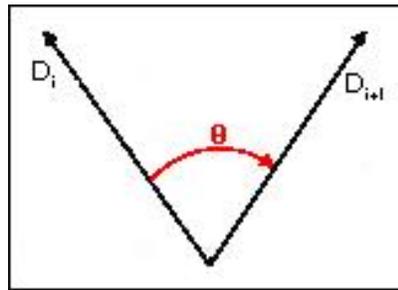


Abbildung 3.10: Winkel zwischen den Vektoren

Also muss zunächst der Winkel θ_i berechnet werden. Dazu müssen die Vektoren D_i und D_{i+1} normiert werden. D.h. sie müssen jeweils auf die Länge 1 gebracht werden.

$$\vec{D}_i^0 = \frac{1}{|\vec{D}_i|} \cdot \vec{D}_i$$

mit

$$|\vec{D}_i| = \sqrt{x_i^2 + y_i^2 + z_i^2}$$

Danach kann mit Hilfe des Skalarprodukts der normierten Vektoren der Winkel θ_i bestimmt werden. Dies entspricht dem Kosinus des Winkels θ_i , da die beiden Vektoren D_i und D_{i+1} vorher normiert wurden und somit die Länge 1 besitzen.

$$\begin{aligned}\vec{D}_i^0 \circ \vec{D}_{i+1}^0 &= |\vec{D}_i| \cdot |\vec{D}_{i+1}| \cdot \cos \theta_i = \cos \theta_i \\ \Rightarrow \theta_i &= \arccos(\vec{D}_i^0 \circ \vec{D}_{i+1}^0)\end{aligned}$$

Jetzt muss eine Funktion gefunden werden, die korrekt zwischen den beiden Richtungsvektoren D_i und D_{i+1} interpoliert.

Die lineare Interpolation zwischen diesen beiden Vektoren ist zwar einfach zu berechnen, würde aber ungleichmäßige Bewegungen produzieren. Aus diesem Grund muss ein geeigneteres Interpolationsverfahren gefunden werden.

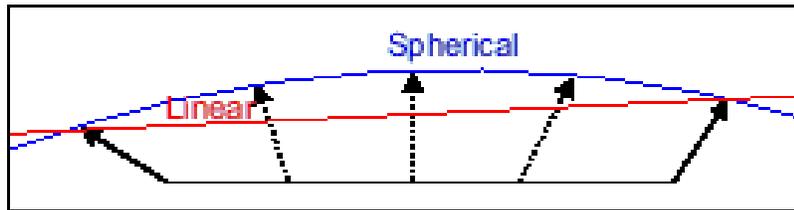


Abbildung 3.11: Lineare und Sphärische Interpolation

Ein solches Verfahren ist die sogenannte *sphärische Interpolation*. Dabei werden alle Einheitsvektoren im dreidimensionalen Raum betrachtet. Sie haben alle die gleiche Länge (also 1), zeigen aber in alle möglichen Richtungen. Würde man ihren Fußpunkt in den Ursprung verschieben, so erhielte man eine Einheitskugel. Das hier angewandte Interpolation entspräche dann einer Bewegung auf dieser Kugeloberfläche. Aus diesem Zusammenhang kann dann die folgende Gleichung zur Interpolation der beiden Vektoren D_i und D_{i+1} hergeleitet werden:

$$D_i(u) = \frac{\sin((1-u) \cdot \theta_i)}{\sin \theta_i} \cdot D_i + \frac{\sin(u \cdot \theta_i)}{\sin \theta_i} \cdot D_{i+1}$$

Wie die obige Gleichung zeigt, wird bei dieser Interpolation durch $\sin \theta_i$ geteilt. Dies impliziert den Sonderfall, dass $\sin \theta_i = 0$ ist. Dies ist der Fall für $\theta_i = 0^0$ oder $\theta_i = 180^0$.

Für den Fall $\theta_i = 0^0$ soll gelten, dass sich die Blickrichtung in zwei aufeinanderfolgenden Keyframes nicht ändern soll. Dieser Fall ist zu überprüfen und es muss für jeden Wert von u $D_i(u) = D_i$ gelten.

Der Fall $\theta_i = 180^0$ ist auszuschließen. Man müsste davon ausgehen, dass die Kamera in zwei aufeinander folgenden Keyframes in entgegengesetzte Richtungen zeigen soll. Auch die Frage, ob dies im oder gegen den Uhrzeigersinn geschehen soll, kann auf diese Weise nicht geklärt werden.

Der interpolierte Richtungsvektor $D_i(u)$ für die Blickrichtung ist nun berechnet worden. Der Fußpunkt dieses Vektors wird jetzt an den bereits berechneten Kurvenpunkt verschoben. Und in Abhängigkeit davon wird dann der Blickpunkt neu berechnet.

Kapitel 4

Der entwickelte Kameraeditor

In diesem Kapitel soll der entwickelte Kameraeditor vorgestellt werden. Die Funktionalität des Kameraeditors beruht auf den theoretischen Grundlagen, die im vorherigen Kapitel behandelt wurden. An dieser Stelle soll nicht die Funktionsweise der Algorithmen, sondern das Ergebnis dieser Arbeit im Vordergrund stehen.

Um dem Benutzer eine Interaktion mit dem Programm zu ermöglichen, wurde ein GUI implementiert. Das GUI ist wesentlicher Bestandteil des Systems, denn es stellt die Verbindung zwischen dem Benutzer und der Anwendung dar. Es ermöglicht somit dem Benutzer eine Interaktion mit dem System. In diesem Fall dient das GUI der Steuerung der Anwendung und der Festlegung der Keyframes, die für die Erstellung der Kamerafahrt benötigt werden.

Das GUI des Kameraeditors setzt sich aus einem Fenster zur Darstellung der Szene, einem Fenster zur Darstellung der Keyframes und einem Fenster zur Darstellung der Geschwindigkeitswerte zusammen.

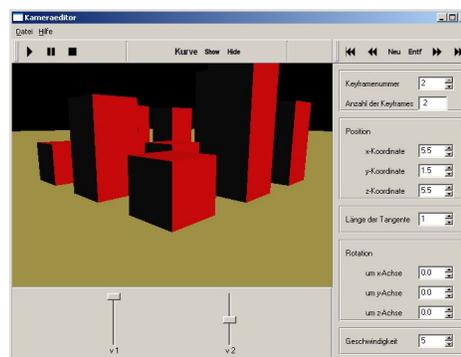


Abbildung 4.1: Das GUI des Kameraeditors

Das 3D-Ausgabefenster

Die Hauptaufgaben des 3D-Ausgabefensters sind das Darstellen der Szene und die Ausgabe der Kameraanimation.

In diesem Fenster wird die Szene gezeichnet. Da im Rahmen dieser Arbeit die Implementierung der Funktionalität im Vordergrund stand, besteht die Szene hier nur aus einem Test-Setting.

Der Benutzer kann frei in dieser Szene navigieren. Die Steuerung der Navigation erfolgt über die Mausbewegung und deren Tasten.

Durch die Tastenkombination CTRL+N kann ein neuer Keyframe spezifiziert werden. Dieser Keyframe wird immer hinter den aktuellen Keyframe in der Liste eingefügt. Die Werte des Keyframes entsprechen dann der aktuellen Kamerakonfiguration.

Zur Unterstützung der Bedienbarkeit des Programms kann sich der Benutzer die bereits spezifizierten Keyframes in der Szene anzeigen lassen. Dies erfolgt über eine Visualisierung ihrer Position. Nach Bedarf können sie auch wieder verborgen werden.

Letztendlich kann die Animation auch in diesem Fenster ausgegeben werden. Die Steuerung dieser Funktionalität erfolgt über den Toolbar oberhalb des Ausgabefensters.

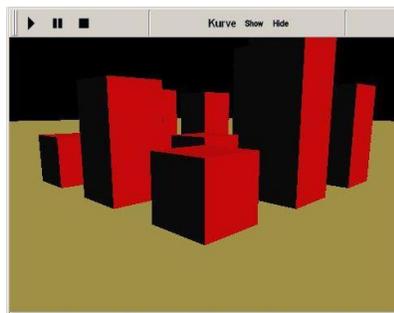


Abbildung 4.2: Das 3D-Ausgabefenster

Das Keyframefenster

Das Keyframefenster zeigt immer nur das Datenblatt des aktuellen Keyframes an.

Über das Keyframefenster kann der Benutzer jedes Attribut des aktuellen Keyframes verändern.

Da das Keyframefenster immer nur das Datenblatt des aktuellen Keyframes repräsentiert, bietet der Toolbar oberhalb des Fensters die Möglichkeit, zwischen den Datenblättern der einzelnen Keyframes zu blättern. Er realisiert

somit die Operationen auf der KeyframeListe.

Der Benutzer kann zum ersten, vorherigen, nächsten und letzten Keyframe in der KeyframeListe wechseln. Zusätzlich stehen die Möglichkeiten zur Verfügung, einen leeren Keyframe, d.h. einen Keyframe dessen Werte mit 0 initialisiert sind, einzufügen und den aktuellen Keyframe aus der Liste zu löschen.



Abbildung 4.3: Das Keyframefenster

Das Geschwindigkeitsfenster

Dieses Fenster dient der Steuerung der Geschwindigkeitswerte. Es bietet dem Benutzer die Möglichkeit, die Geschwindigkeiten für die einzelnen Keyframes über Slider zu spezifizieren und auch nachträglich wieder zu verändern. Dadurch erhält der Benutzer auch einen Überblick über das Geschwindigkeitsverhalten der Kamera entlang der gesamten Kurve.



Abbildung 4.4: Das Geschwindigkeitsfenster

Kapitel 5

Zusammenfassung

Im Rahmen dieses Projekts ist ein interaktives Programm zur Erstellung einer Kamerafahrt entstanden.

Es ist für den Benutzer über ein GUI steuerbar. Die Keyframes zur Erstellung der Kamerafahrt können über das GUI festgelegt und jederzeit verändert werden. In einem Ausgabefenster, das Teil des GUI ist, wird die erstellte Animation dann abgespielt. Die so spezifizierten Daten können in einer Datei abgespeichert werden und umgekehrt auch aus einer Datei geladen werden. Dem Erstellen der Kameraanimation liegen verschiedene Funktionalitäten zugrunde.

Zum einen soll die Kamera entlang eines Pfades bewegt werden, der durch die Keyframes spezifiziert wurde. Dies entspricht einer Interpolation der Position, die an Hand der Hermite-Interpolation erfolgt ist. Dies gibt dem Benutzer direkte Kontrolle über den Verlauf des Kamerapfades, da der Verlauf der Kurve über die Länge der Tangenten steuerbar ist.

Zum anderen soll das Geschwindigkeitsverhalten der Kamera entlang des Pfades kontrolliert werden. Dazu müssen die Geschwindigkeiten, die durch den Benutzer spezifiziert wurden, zwischen den einzelnen Keyframes interpoliert werden. Dies ermöglicht die Definition einer Geschwindigkeits-Zeit-Funktion $v(t)$. Mittels Integration über die Zeit kann so eine Weg-Zeit-Funktion $s(t)$ abgeleitet werden. Sie ordnet jedem Zeitpunkt t den bis dahin zurückgelegten Weg s zu. Mit Hilfe einer Parametrisierung über die Bogenlänge kann dann der Punkt auf der Kurve gefunden werden, an dem sich die Kamera zu diesem Zeitpunkt befinden soll.

Des Weiteren soll die Kamera während ihrer Fahrt in unterschiedliche Richtungen zeigen. Für jeden Keyframe kann der Benutzer einen Blickpunkt, ein sogenanntes „center of interest“, festlegen. Um eine „weiche“ Animation zu erhalten, muss auch zwischen diesen Blickpunkten interpoliert werden. Dies erfolgt über eine Interpolation des Winkels zwischen den Vektoren, die sich

aus den Blickpunkten und der Kameraposition berechnen lassen.
Mit Hilfe dieser Funktionalitäten kann die Kamera dann entlang einer Kurve bewegt werden und während dieser Bewegung auch so gedreht werden, dass sie jederzeit in die gewünschte Richtung zeigt. Ihre Bewegung kann beschleunigt oder abgebremst werden.

Kapitel 6

Ausblick

Der hier entstandene Kameraeditor ist innerhalb eines festen Zeitrahmens entwickelt worden. Innerhalb dieses fest definierten Zeitraums war es nicht möglich, verschiedene Ansätze zu implementieren und zu evaluieren. Die hier verwendeten Ansätze sind in vielerlei Hinsicht erweiterbar. Hier sollen einige Verfahren vorgestellt werden, mit deren Hilfe der bestehende Kameraeditor unter Umständen verbessert werden könnte. Die genauen Funktionsweisen dieser Verfahren werden im Anhang erläutert.

Eine Implementierung anderer Algorithmen zur Umsetzung der gegebenen Funktionalität, wäre aufgrund der zugrunde liegenden Datenstrukturen mit relativ geringem Mehraufwand durchführbar.

Der Kameraeditor kann bzgl. folgender Funktionalitäten optimiert werden:

- Interpolation der Position
- Interpolation der Geschwindigkeit
- Parametrisierung über die Bogenlänge
- Interpolation der Blickrichtung

Interpolation der Position

Die Interpolation der Position könnte auch durch andere Interpolationsverfahren realisiert werden. Die meisten dieser Verfahren arbeiten ebenfalls mit der Erstellung einer Kurve, die sich aus polynominalen Kurvensegmenten zusammensetzt. Sie wären somit ohne große Erweiterungen im Programm umsetzbar.

Andere Verfahren wie z.B. die B-Splines arbeiten mit Kontrollpunkten, die sich aus den Kurvenpunkten berechnen lassen, um so ebenfalls eine Kurve zu konstruieren. Dies würde eine Erweiterung um eine Datenstruktur zu Speicherung dieser Kontrollpunkte mit sich ziehen. Diese Verfahren eignen sich

allerdings nicht so gut für diese Anwendung, da ihre Struktur zur Konstruktion der Kurve nicht flexibel genug ist. Jede Veränderung in den Kurvenpunkten würde die Kontrollpunkte beeinflussen, und somit eine Neuberechnung dieser mit sich bringen würde.

Ein Nachteil der meisten anderen Verfahren ist es auch, dass im Gegensatz zu der hier verwendeten Hermite-Interpolation der Kurvenverlauf nicht steuerbar ist. Er berechnet sich ausschließlich aus den Kurvenpunkten, bei manchen Verfahren werden die Start- und Endtangente der kompletten Kurve in die Berechnung miteinbezogen. Dadurch ist der Kurvenverlauf aber nur minimal steuerbar.

Interpolation der Geschwindigkeit

Bei dem hier entwickelten Kameraeditoren beruht die Definition der Geschwindigkeit auf einer linearen Interpolation. Zwischen den Keyframes wird die Geschwindigkeit linear interpoliert, wenn auch zeitverzögert nur in einem bestimmten Intervall $[t_1, t_2]$. Um weichere Übergänge zu erhalten, wäre es möglich die Geschwindigkeit nicht linear zu interpolieren. Ein Ansatz dazu wäre es, mit Hilfe der Sinus-Kurve eine Weg-Zeit-Funktion $s(t)$ zu definieren. Diese Möglichkeit soll als Alternative zu der hier gewählten Methode im Anhang genauer erläutert werden.

Parametrisierung über die Bogenlänge

Nachdem die Weg-Zeit-Funktion $s(t)$ aufgestellt wurde, musste die Kurve über die Bogenlänge neu parametrisiert werden. Dabei wurde hier die Kurve durch gleichförmiges Unterteilen des Parameterraumes angenähert. Hierbei entstehen jedoch große Abweichungen von der ursprünglichen Kurve. Um die Abweichungen zu minimieren und den Rechenaufwand dennoch möglichst gering zu halten, wurden adaptive Verfahren entwickelt. Dabei wird die Kurve an den Stellen, an denen die Abweichung zu gross würde, feiner unterteilt und der Fehler so minimiert. Die Kriterien, wann eine solche feinere Unterteilung vorgenommen werden müsste, sind unterschiedlich. Zwei dieser Verfahren sollen im Anhang vorgestellt werden. Mittels adaptiver Verfahren könnte der Fehler bei der Parametrisierung durch die Bogenlänge zumindest minimiert werden. Eine analytische oder numerische Methode wäre ebenso anwendbar. Dabei bliebe abzuwägen, ob die Ergebnisse den Rechenaufwand rechtfertigen.

Interpolation der Blickrichtung

Ein unerwünschter Effekt, der bei dem hier gewählten Verfahren zur Interpolation der Blickrichtung auftreten kann, ist der Sprung in den Werten an den Übergängen zwischen den Keyframes.

Ein solches Beispiel wäre eine Interpolation von drei Vektoren, die jeweils

in Richtung der positiven x, y- und z-Achse zeigen. Bei dem hier angewandten Verfahren würde die Interpolation zwischen den ersten beiden Vektoren Werte ergeben, die in der xy-Ebene liegen würden. Die Werte zwischen den anderen beiden Vektoren würde Werte in der yz-Ebene berechnen. Dadurch würde eine nicht kontinuierliche Animation entstehen. Ein Verfahren, das diesen Nebeneffekt umgeht, ist eine Variante des hier verwendeten Interpolationsverfahrens. Dabei wird unter Berücksichtigung des vorherigen und nachfolgenden Keyframes interpoliert. Die Funktionsweise dieser Methode wird im Anhang erläutert.

Fazit:

Das Programm ist bzgl. seiner Funktionalität in vielerlei Hinsicht erweiterbar. Innerhalb des hier gesteckten Zeitrahmens war es jedoch nicht möglich, verschiedene Methoden für eine Funktionalität zu implementieren und gegeneinander abzuwägen. Der hier entwickelte Kameraeditor ist somit als eine Art „Alpha-Version“ zu sehen. Die vorher definierten Ziele werden durch ihn erfüllt und wurden auch in dem vorher festgelegten Zeitrahmen realisiert.

Um das Programm zu optimieren, müsste noch Zeit und Arbeit darin investiert werden, die implementierte Funktionalität hinsichtlich der Qualität ihrer Ergebnisse zu verbessern. Um dies zu erreichen, müsste man wahrscheinlich alternative Algorithmen implementieren und ihre Ergebnisse dann evaluieren.

Ein weiterer Gesichtspunkt, der bei diesem Programm noch nicht berücksichtigt wurde, ist die Performance. Das Programm wurde „quick and dirty“ programmiert. Auch im Bereich der Effizienz sind noch Verbesserungen möglich.

Grundsätzlich erfüllt der Kameraeditor allerdings, die an ihn gestellten Anforderungen. Er erstellt aus den gegebenen Keyframes in Echtzeit eine Kameraanimation, bei der die Kamera, beschleunigt oder abgebremst, entlang einer Kurve bewegt wird und während dieser Kamerafahrt immer in die gewünschte Blickrichtung zeigt.

Kapitel 7

Anhang

7.1 Alternative Geschwindigkeitsberechnung

Alternativ zu dem hier verwendeten Verfahren der linearen Interpolation kann die Geschwindigkeit auch mittels der Sinus-Funktion interpoliert werden. Diese Verfahren soll hier für das Ease-in vorgestellt werden. Dabei entspricht die Weg-Zeit-Funktion $s(t)$ der Sinus-Funktion in dem Intervall von $-\pi/2$ bis $+\pi/2$.

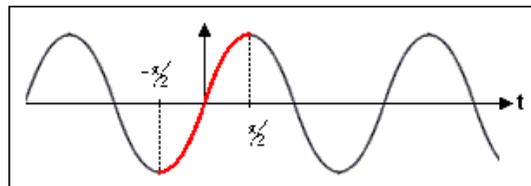


Abbildung 7.1: Die Sinus-Funktion

Der Wertebereich der Sinus-Funktion von -1 bis $+1$ wird dann anschließend auf einen Wertebereich von 0 bis 1 abgebildet.

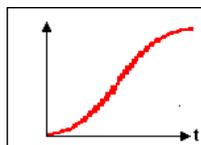


Abbildung 7.2: Die Weg-Zeit-Funktion

Dies kann dann durch folgende Gleichung ausgedrückt werden:

$$s(u) = \frac{\sin(u \cdot \pi - \frac{\pi}{2}) + 1}{2}$$

Die Vorgehensweise beim Ease-out ist analog, nur dass dabei der Verlauf der Sinus-Kurve im Intervall von $\pi/2$ bis $3\pi/2$ betrachtet wird.

7.2 Alternative Bogenlängenparametrisierungen

In diesem Abschnitt sollen zwei adaptive Verfahren zur Parametrisierung einer Kurve über ihre Bogenlänge vorgestellt werden. Adaptive Verfahren beurteilen die Abweichung von der ursprünglichen Kurve und unterteilen die Kurve an den Stellen weiter, an denen die Abweichung zu gross würde. Auf diese Weise wird der Fehler, der bei dieser Annäherung gemacht wird, reduziert, und die Unterteilungen so gering wie möglich gehalten. Die adaptiven Verfahren unterscheiden sich in ihren Kriterien zur Abschätzung der Abweichung.

Hier sollen zwei unterschiedliche Verfahren vorgestellt werden:

- mittels der Midpoint Distanz
- mittels der Varianz

Midpoint Distanz

Bei diesem Verfahren wird der sogenannte „Midpoint“ zwischen zwei Punkten $C(u_1)$ und $C(u_2)$ berechnet, indem man ihre Parameterwerte mittelt. In Abhängigkeit von dem so berechneten Parameterwert $u_m = (u_2 - u_1)/2$ wird der zugehörige Punkt auf der Kurve bestimmt. Die Distanz D des Midpoints zu der Geraden zwischen Punkt $C(u_1)$ und $C(u_2)$ wird nun als Unterteilungskriterium betrachtet. Für $D > \varepsilon$ wird zwischen den Punkten $C(u_1)$ und $C(u_m)$ und $C(u_m)$ und $C(u_2)$ fortgefahren.

Bei diesem Verfahren kann der Fall auftreten, dass die Gerade, die die beiden Endpunkte des Segments verbindet, den Kurvenverlauf kreuzt.

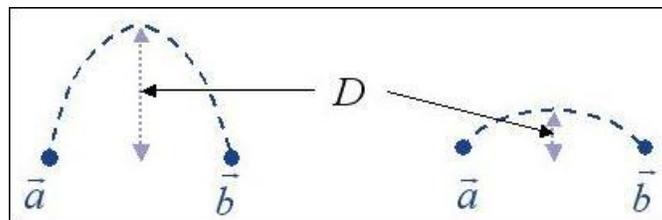


Abbildung 7.3: Abstand des Midpoints zu der annähernden Geraden

Der Sonderfall, dass der Midpoint dann innerhalb der Toleranzgrenze liegt, wird hierbei nicht erkannt. Das Kurvensegment wird irrtümlicherweise nicht

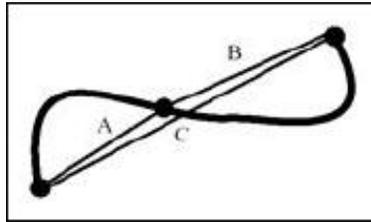


Abbildung 7.4: Test bei der Bestimmung der Midpoint Distanz

weiter unterteilt.

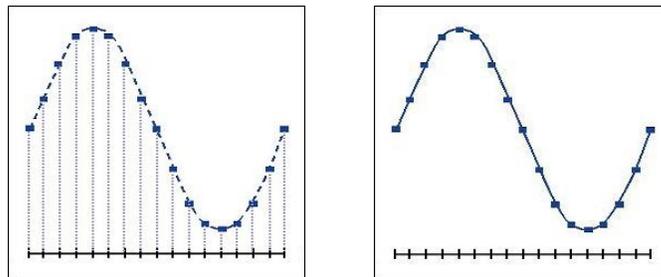


Abbildung 7.5: Approximation der Bogenlänge mittels der Midpoint Distanz

Varianz

Bei diesem Verfahren wird die Varianz zwischen der Kurve und der Geraden, die die Endpunkte des zu unterteilenden Segments verbindet, berechnet. Überschreitet die Varianz einen Schwellwert, so wird unterteilt, ansonsten wird mit dem nächsten Segment fortgefahren.

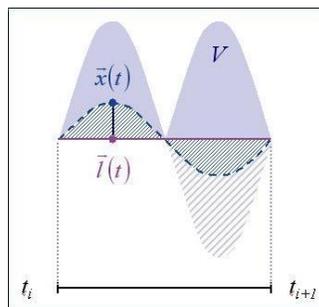


Abbildung 7.6: Die Varianz der Approximation der Kurve

Sei das zu unterteilende Segment gegeben durch u_1 und u_2 und $\vec{l}_i(u)$ sei die Gerade, die die Endpunkte des Segments verbindet. So ist die Varianz definiert durch:

$$V = \int_{u_1}^{u_2} |\vec{x}(u) - \vec{l}_i(u)|^2 du$$

Ein Nachteil dieses Verfahrens ist die Komplexität des Integrals, dessen Lösung je nach gewählter Kurvenform mit einem relativ hohen Rechenaufwand verbunden ist.

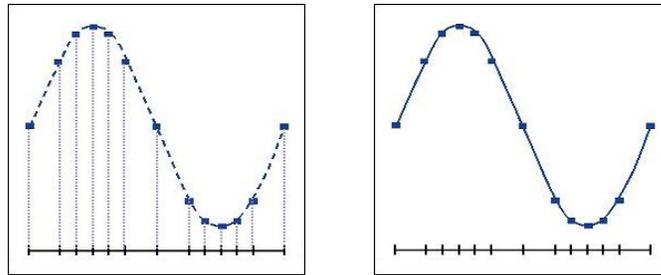


Abbildung 7.7: Approximation mittels Bestimmung der Varianz

7.3 Alternative Interpolation der Blickrichtung

Die hier verwendete Interpolation der Blickrichtung interpoliert ausschließlich zwischen zwei Vektoren D_i und D_{i+1} .

Um aber weichere Übergänge zu erhalten, ist es möglich, den vorherigen Vektor D_{i-1} und den nachfolgenden Vektor D_{i+2} mit in die Berechnung einzubeziehen. Im Segment i interpoliert diese Funktion dann zwischen den Vektoren D_i und D_{i+1} , bezieht aber auch die Vektoren D_{i-1} und D_{i+2} gewichtet mit in die Interpolation ein.

Um dies zu erreichen, werden für jedes Segment zwei zusätzliche Richtungsvektoren X_i und Y_i berechnet. Diese können z.B. wie folgt berechnet werden:

$$X_i = D_i + 1/6 \cdot (D_{i+1} - D_{i-1})$$

$$Y_i = D_{i+1} - 1/6 \cdot (D_{i+2} - D_i)$$

Die Interpolationsfunktion sollte nun zu Beginn eines jeden Segments X_i

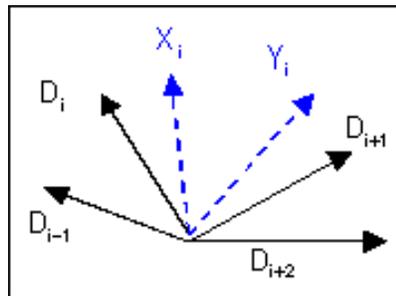


Abbildung 7.8: Sphärische Interpolation mittels vier Vektoren

mehr Gewicht zuweisen. Am Ende eines jeden Segments sollte Y_i mehr ins Gewicht fallen. Dies kann durch folgende Funktion erreicht werden:

$$D_i(u) = (1 - u)^3 \cdot D_i + 3u \cdot (1 - u)^2 \cdot X_i + 3u^2 \cdot (1 - u) \cdot Y_i + u^3 \cdot D_{i+1}$$

Dies entspricht im Prinzip einer Bezier-gewichteten Summe, denn die hier verwendeten Polynome stimmen mit den bei Bezier-Kurven verwendeten Bernstein-Polynomen überein.

Bei Verwendung dieser Interpolationsart bliebe noch zu überdenken, wie bei dem Start- und Endsegment vorgegangen werden soll. Eine Möglichkeit wäre es, den Benutzer jeweils einen Start- und Endvektor D_{-1} und D_n spezifizieren lassen, damit dem Programm ausreichend Informationen zur Berechnung

der Vektoren X_i und Y_i für alle $0 \leq i < n$ zur Verfügung stehen.
In jedem Fall stellt der Einbezug weiterer Vektoren eine Verbesserung der Interpolation dar. Da die Segmente an den Keyframes glatt ineinander übergehen würden.

Kapitel 8

Quellenangaben

- Eberly, David H.
3D Game Engine Design. A Practical Approach to Real- Time Computer Graphics,
Morgan Kaufmann Publishers,
2000
- Parent, Rick
Computer Animation Algorithms and Techniques,
Morgan Kaufmann Publishers,
2002
- Salomon, David
Computer Graphics Geometric Modeling,
Springer,
1999
- Shirley, Peter
Fundamentals of Computer Graphics,
AK Peters,
2002
- Gieen, Prof. Dr.-Ing. Heinrich: Texte zur Computergraphik
- http://isgwww.cs.uni-magdeburg.de/masuch/computerspiele/PSVortraege/Animation_in_Computerspielen.pdf
- <http://www.uni-koblenz.de/cg/veranst/ss03/material.html>
- <http://www.gg.caltech.edu/cs274/Lectures/lecture01.pdf>
- http://cgvr.korea.ac.kr/Courses/2001/CSE714/TP/chapter7_2.ppt

- <http://www.cg.tuwien.ac.at/studentwork/CESCG-2002/GSchroecker/node13.html>