FORSCHUNGSZENTRUM JÜLICH GmbH
KFA

Zentralinstitut für Angewandte Mathematik

Orts- und Orientierungsinterpolation zur kontrollierten Generierung von Zwischenpositionen für Animationssequenzen

Wolf-Dieter Groß

Berichte des Forschungszentrums Jülich ; 2948
ISSN 0944-2952
Zentralinstitut für Angewandte Mathematik Jül-2948

Zu beziehen durch : Forschungszentrum Jülich GmbH · Zentralbibliothek
D-52425 Jülich · Bundesrepublik Deutschland
Telefon : 0 24 61 / 61 - 61 02 · Telefax : 0 24 61 / 61 - 61 03 · Telex : 8 33 556-70 kfa d

Orts- und Orientierungsinterpolation zur kontrollierten Generierung von Zwischenpositionen für Animationssequenzen

Wolf-Dieter Groß

Zusammenfassung:

In der Computeranimation wird eine gleichmäßige Bewegungsplanung von Objekten, so z.B. einer virtuellen Kamera, benötigt. Die vorliegende Arbeit stellt das Programmpaket VIDES vor, das eine interaktive Bahnplanung für Bewegungsvorgänge ermöglicht. Die Hauptzielrichtung der Arbeit ist die Bahnplanung einer virtuellen Kamera, mit deren Hilfe die einzelnen Szenen einer Animationssequenz aufgenommen werden können.
Um eine Beschreibung von Position (Ort) und Ausrichtung (Orientierung) der virtuellen Kamera zu finden, werden die Projektionsparameter eingeführt, die zur Berechnung einer einzelnen Bildes (Projektion) herangezogen werden. Für die Ortsbeschreibung der Kamera werden verschiedenen Interpolationskurven aus der Gruppe der Splinekurven erläutert. Die in Vides verwendeten Kurven werden detailiert dargestellt. Für die Orientierungsbeschreibung werden Eulerwinkel und Quaternionen vorgestellt. da sich nur Quaternionen für eine einfache Orientierungsinterpolation eignen, wird für diese eine Methode der Orientierungsinterpolation aufgeführt.
Da in Vides auch die Interpolationsrate und damit die Bewegungsgeschwindigkeit der Kamera interaktiv frei gewählt werden kann, wird eine Zeitkurvendarstellung eingeführt, mit deren Hilfe die Interpolations- bzw. Animationsgeschwindigkeit festgelegt werden kann.
Das interaktive Modul VIDES erlaubt die direkte graphische Eingabe der Kurvenparameter im dreidimensionalen Raum mit Hilfe einer (zweidimensionalen) Maus. Die dabei verwendeten Algorithmen werden dargestellt.
Abschließend wird noch eine Überblick über das gesamte Programmodul gegeben und es wird der Entwurf einer Animationssequenz mit VIDES vorgestellt.

abstract:

In the area of computer animation it is necessary to design motion paths for objects like a virtual camera. This paper presents the programm module VIDES. VIDES allows the interactive design of motion paths. These motion paths are mainly suitable to describe the movement of a virtual camera used for generating single frames of an animation sequence. To find a usefull description of the position and orientation of the camera we introduce projection parameters. These parameters can be used to calculate a single frame (single projection) of the animation sequence.
For the interpolation of position information, several interpolation algorithms based on spline curves are presented. Using this information, we discuss the special class of spline curves used by VIDES.
For orientation interpolation we consider the methods using Euler angles and quaternions. As orientation interpolation can easily be done only using quaternions, we describe the algorithms for this task.
In VIDES you can interactivly enter the interpolation rate, resulting in different motion speed. Time curves, suitable to describe the animation speed, are described.
In VIDES position and orientation information can be interactively specified in three dimensional space. The algorithms usefull for the declaration of this information using a (two-dimensional) mouse are discussed.
The conclusion gives an overview of the entire program module. An example shows the design stage of an animation sequence using VIDES.

Inhalt

Abbildungsverzeichnis

Kapitel 1

Einleitung

1.1 Problemstellung

Durch Computersimulation können heute physikalische, technische oder chemische Vorgänge erforscht werden, deren Beobachtung an realen Systemen nicht oder nur ungenügend möglich ist. Dabei erhält man als Ergebnis eine Datenflut, die, rein zahlenmäßig dargestellt, unüberschaubar wird.

Häufig verwendet man deshalb Visualisierungsmöglichkeiten der Computergraphik, um ein anschauliches Auswerten der Daten zu ermöglichen.

Hat man z.B. das Wachstum eines Kristalles simuliert, so kann man durch eine 3D-Darstellung der Simulationsergebnisse die Form, das Aussehen oder die Größe des Kristalles leicht erkennen. Die Daten über den menschlichen Körper, die moderne bildgebende Verfahren der Diagnostik liefern, können zur Darstellung der untersuchten Organe oder Körperteile herangezogen werden.

Dabei ist die Ansicht des Kristalles, des Körpers oder anderer Abbildungen aus einer festen Position jedoch nicht ausreichend, um alle Details zu erfassen. Ansichten aus mehreren Richtungen, zu kurzen Filmsequenzen zusammengesetzt, können die Aussagekraft der Daten deutlich erhöhen.

Die Filmgenerierung verwendet existierende Verfahren zur Darstellung einzelner Ansichten. Da viele Bilder aus verschiedenen Ansichten erzeugt werden sollen, stellt sie darüber hinaus weitergehende Anforderungen. Die einzelnen Ansichten müssen spezifiziert werden, bevor die dazugehörigen Bilder berechnet werden können. Es ist aufwendig und langwierig, manuell sämtliche Parameter zu erzeugen, die die benötigten Ansichten für jeden einzelnen Zeitpunkt beschreiben.

Für den Anwender wäre es einfacher, wenn er nur einige markante Punkte für die Darstellung („Kamerapositionen“) und evtl. einige zusätzliche Parameter zu spezifizieren brauchte und der Computer danach sämtliche Zwischenpositionen automatisch erstellen könnte.

Die bisher erwähnte Methode der Animation, durch Veränderung der Kameraposition Filmsequenzen zu erzeugen, stellt nur eine Klasse der Animationssysteme dar. Eine andere Klasse, auf die hier nicht näher eingegangen werden soll, beschäftigt sich dagegen mit

Vorgängen, deren Inhalt oder Struktur sich im zeitlichen Verlauf ändert. Ein Beispiel wäre die Darstellung von Wetterdaten, wie sie von Satelliten geliefert werden. Beide Animationsklassen können auch verknüpft werden, lassen sich in der Entwicklung aber unabhängig voneinander betrachten.

1.2 Spezifizierung der Aufgabenstellung

Um die Aufgabenstellung und die Anforderungen für ein Animationssystem mit bewegter Kamera spezifizieren zu können, soll zuerst ein anschauliches Modell der Kamerabeschreibung entwickelt werden. Dazu werden drei unterschiedliche Parametergruppen beschrieben, die bei der „Aufnahme“ einer Szene benötigt werden:

1. Die **Position** oder der Ort der Kamera, von der aus die „Szene“ betrachtet wird, ist sicher der wichtigste Parameter
2. Die **Orientierung** der Kamera muß ebenfalls festgelegt werden. Darunter versteht man die Richtung, in die Kamera „blickt“, aber auch die Lage ihres Bildhorizontes.
3. Es sind noch kameraspezifische Parameter anzugeben. Dies ist einerseits der **Blickwinkel**, abhängig von der Brennweite der Kamera der oft auch unter dem Begriff **Zoomfaktor** aufgeführt wird. Andererseits sind damit abbildungsspezifische Parameter gemeint, wie Angaben über mögliche Verzerrungen im Abbildungsprozeß oder Angaben über den Abbildungsvorgang wie **Aufnahmezeit** und Blende.

Da der Anwender nur für einige Schlüsselszenen die Angaben von Ort, Orientierung und Zoomfaktor spezifizieren will und der Computer die Zwischenpositionen daraus dann automatisch generieren soll, müssen geeignete mathematische Beschreibungen für diese Angaben gefunden werden. Danach müssen Verfahren aufgezeigt werden, um aus diesen die Zwischen- oder Interpolationswerte zu berechnen.

Nun lassen sich die so erzeugten Angaben natürlich nicht nur auf eine Kamerabeschreibung übertragen. Mit diesen Modellen können vielmehr eine größere Anzahl von Problemstellungen gelöst werden:

So ist z.B. die Planung der Bewegung eines Roboterarmes eine Aufgabenstellung, die mit den beschriebenen Algorithmen durchgeführt werden kann. Als weitere Anwendungen sind z.B. die Generierung von Körperschnitten bei der 3D-Rekonstruktion von Daten eines Computertomographen, die Planung der Route eines Flugzeuges oder auch die Bewegungsplanung von Beleuchtungskörpern bei realen oder simulierten Filmaufnahmen denkbar.

1.3 Existierende Visualisierungssysteme

Betrachtet man existierende Visualisierungssysteme, so sind im kommerziellen Bereich Systeme wie das Application Visualisation System AVS verfügbar. Dieses modular aufgebaute Visualisierungssystem enthält eine Vielzahl von Modulen zur Vorverarbeitung

und graphischen Darstellung, die mit einem graphischen Benutzerinterface miteinander kombiniert werden können. Die Darstellungs-/Kameraparameter lassen sich in AVS interaktiv einstellen oder auch über eine Kommadozeile eingeben. Damit ist eine Animation möglich aber ein spezielles Bahnplanungsmodul ist bisher nicht vorhanden ([AVS]). Animationssysteme wie Explore von TDI bieten Animationsmodule, die eine Bahnplanung ermöglichen. Diese Module sind jedoch fest in die jeweiligen Animationssysteme integriert ([TDI]).

Im Bereich der frei verfügbaren Software findet man zahlreiche Programmpakete, die z.B. die 3D-Darstellung ermöglichen oder Ray-Tracing durchführen. Im Animationsbereich sind verschiedene Programmpakete verfügbar, die Animationsmöglichkeiten für zeitlich veränderliche Daten durchführen, wie z.B. VIS-5D zur Wetterdatendarstellung ([VIS5D]). Systeme wie SciAn (Scientific Visualization) bieten zwar Möglichkeiten der interaktiven Spezifikation von Kamerapositionen, aber eine Kamerabahnplanung ist in solchen Systemen ebenfalls nicht vorhanden ([SCIAN]). Verschiedene Widgetsets sind entwickelt worden, um vorhandene graphische Benutzeroberflächen wie X-Windows um Module zur direkten Ausgabe von graphischen Daten zu erweitern. Ein eigenständiges Bahnplanungsmodul ist aber nicht frei verfügbar, und vorhandene Module kommerzieller Software können nicht in eigenen Programmen verwendet werden.

1.4 Gliederung der Arbeit

In der vorliegenden Arbeit werden deshalb Algorithmen aufgezeigt, mit denen man die Bahnplanung für Animationsvorgänge automatisieren kann. Ein unter Verwendung dieser Routinen entwickeltes Programm bietet dem Anwender die Möglichkeit, interaktiv am Bildschirm Schlüsselpositionen für die Animation zu entwerfen und zu verändern. Des weiteren kann er angeben, für welche Animationsgeschwindigkeit bzw. mit welchem Abstand Zwischenpositionen generiert werden sollen.

Dem Programmierer steht das entwickelte Programmsystem zur Verfügung, um es einfach in eigene Programmpakete einzubinden. Es ist so angelegt, daß es möglichst breit verwendbar und auf verschiedenen Rechnersystemen verfügbar und portierbar ist.

Da die Simulationen heute meist auf Rechnern laufen, die mit UNIX kompatiblen Betriebssystemen und der graphischen Oberfläche X-Windows arbeiten, wurde dies als Grundvoraussetzung für den Einsatz der Programme gewählt. Zusätzlich stützen die Programme sich auf die OSF-Motif Widgetlibrary, die ebenfalls auf den meisten Workstations implementiert ist. Dies bedeutet vor allem, daß dem Anwender eine bekannte Oberfläche bei der Benutzung des Programms zur Verfügung steht.

Die verwendeten Datenstrukturen, Transformationen und die Übergabeparameter wurden in Anlehnung an die in den Graphikpaketen PHIGS und GKS-3D dargestellten Definitionen entwickelt. Diese sind so allgemein verwendbar, daß man sie unproblematisch auch auf andere Darstellungen übertragen kann.

Die vorliegende Arbeit gliedert sich in folgende Bereiche:

- Kapitel 1: Die **Einleitung** spezifiziert die Anforderungen an das System.

- Kapitel 2: Der zur Darstellung von Szenen auf graphischen Ausgabesystemen notwendige Begriff der **Projektion** wird erklärt und es werden benötigte mathematische Grundlagen aufgezeigt. Darauf aufbauend werden die verschiedenen Arten der Projektion erläutert. Nachdem die Parameter, die zur Festlegung einer Projektion dienen, in Anlehnung an PHIGS eingeführt wurden, wird aufgezeigt, wie damit eine Projektion berechnet werden kann.

- Kapitel 3: Die Zwischenpositionen werden mit Hilfe der **Ortsinterpolation** berechnet. Die hierbei verwendeten Raumkurven werden kurz dargestellt. Dazu wird der Begriff der parametrischen kubischen Kurve eingeführt und auf Anforderungen kurz eingegangen, die deren Verwendung zur Filmgenerierung stellen. Ausgehend von der allgemeinen Beschreibung wird eine spezielle Art dieser Kurven eingeführt, die vielfältige Manipulationsmöglichkeiten bietet.

- Kapitel 4: Die Ausrichtungen der Kamera werden durch die **Orientierungsinterpolation** erzeugt. Zuerst werden Methoden der Orientierungsbeschreibung erläutert und auf Ihrer Verwendbarkeit zur Interpolation untersucht. Auf die Orientierungsbeschreibung mit Hilfe von Quaternionen wird dann genauer eingegangen und ein Verfahren zur Interpolation mit Quaternionen aufgeführt.

- Kapitel 5: Eine Bewegung wird nicht nur durch ihre Positionen und Orientierungen an Zwischenstellen beschrieben, auch die **Bewegungsgeschwindigkeit** muß festlegbar und auswertbar sein. Deshalb werden in diesem Kapitel Verfahren aufgezeigt, Geschwindigkeitsverläufe zu spezifizieren und diese auf die Orts- und Orientierungsbeschreibungen zu übertragen.

- Kapitel 6: Die Spezifizierung der Schlüsselpositionen soll interaktiv für die **Orts- und Orientierungsangaben** mit den normalerweise zur Verfügung stehenden Eingabegeräten Maus und Tastatur erfolgen. Die Methode zur Eingabe dieser Angaben wird festgelegt und Hilfsmittel bei der Eingabe kurz erläutert.

- Kapitel 7: Unter Verwendung der bisher vorgestellten Verfahren wurde das **Programmpaket VIDES** entwickelt. Seine Verwendungsmöglichkeiten, Aufbau und Struktur sowie die Bedienmöglichkeiten werden vorgestellt. Ein Beispiel erläutert die Erstellung einer Animationssequenz anschaulich.

- Kapitel 8: Abschließend folgt noch eine kurze **Zusammenfassung** der Ergebnisse, und es werden eine Bewertung und ein **Ausblick** auf Weiterentwicklungsmöglichkeiten gegeben.

- Anhang: Im Anhang A findet man noch eine Bedienungsanleitung, im Anhang B Hinweise zur Einbindung von VIDES in eigene Programme und im Anhang C ein Beispielprogrammm, das die Einbindung von VIDES zeigt.

Kapitel 2

3D-Projektion

Modelle von Gegenständen oder von Objekten oder die Ergebnisse von Simulationen liegen für die Computerdarstellung normalerweise als Beschreibung in einem dreidimensionalen Koordinatensystem, häufig Welt- oder Objektkoordinatensystem genannt, vor. Da die Gegenstände Objekte in der real existierenden Umwelt beschreiben, diese aber ebenfalls drei Ausdehnungsrichtungen besitzt, ist diese Annahme naheliegend.

Die Ausgabe von graphischen Darstellungen, also von Objekten oder Modellbeschreibungen im dreidimensionalen Raum, erfolgt normalerweise auf Ausgabegeräten, die nur eine Darstellung im zweidimensionalen Raum ermöglichen, so z.B. auf Bildschirmen oder auf Druckern. Sogar bei der Stereoprojektion, bei der man zur Entstehung einer räumlichen Wirkung zwei Ansichten von unterschiedlichen Betrachtungsorten aus erzeugt, werden die Einzelbilder auf einem zweidimensionalen Ausgabemedium dargestellt. Die Abbildung von dreidimensionalen Objekten erfordert deshalb eine Projektion der dreidimensionalen Weltkoordinaten auf ein zweidimensionales Bild- oder Darstellungskoordinatensystem.

2.1 Grundlagen der Projektion und ihrer Berechnung

Allgemein versteht man unter Projektion die Abbildung von Punkten in einem Koordinatensystem der Dimension n auf ein Koordinatensystem der Dimension $m < n$. Hier soll nur die Abbildung von einem Weltkoordinatensystem der Dimension $n = 3$ auf das Bildschirmkoordinatensystem der Dimension $m = 2$ näher behandelt werden.

Es sind mehrere Schritte durchzuführen, bevor man aus einer Beschreibung verschiedener Objekte und der Angabe der Ansicht, aus der diese Objekte betrachtet werden, zu einer endgültigen Darstellung auf dem Bildschirm und damit zur Projektion kommt.

Diese Vorgänge werden am besten durch eine Projektionspipeline beschrieben, die den gesamten Abbildungsvorgang in vier Schritte unterteilt:

1. Häufig soll nicht nur ein Objekt oder Modell dargestellt werden, vielmehr sind auf einem Bild mehrere Objekte anzuzeigen. Diese Objekte werden in jeweils eigenen Koordinatensystemen beschrieben. Als erstes muß deshalb eine Überführung sämtlicher

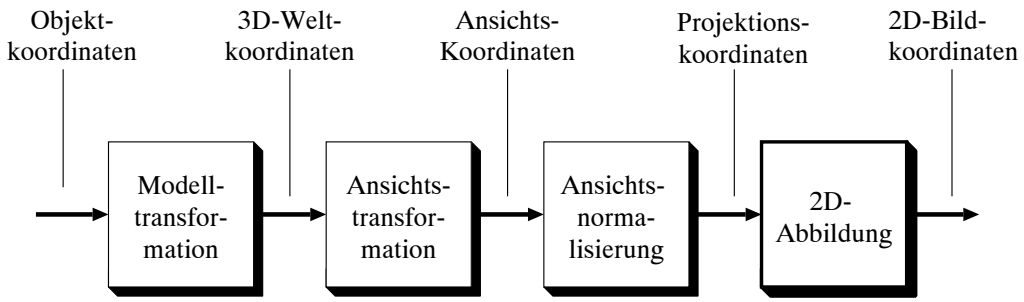


Abbildung 2.1: Projektionspipeline

Objektkoordinaten der einzelnen Modelle in ein einheitliches 3D-Weltkoordinatensystem gefunden werden. Dieser Schritt wird mit Modelltransformation bezeichnet. Da dieser Schritt jedoch abhängig von den verwendeten Objektkoordinaten ist, soll hier nicht weiter darauf eingegangen werden.

2. Die Berechnung der Projektion ist nur dann einfach lösbar, wenn die Ansicht einer vorgegebenen Grundansicht entspricht . Deshalb wird in zwei Schritten diese vorgegebene Grundansicht erzeugt . Der erste Schritt richtet die Ansicht so im Raum aus, daß Blickrichtung und Betrachtungsort mit der Grundansicht übereinstimmen. Das nennt man Ansichtstransformation; man hat danach eine Beschreibung der Objekte in einem Ansichtskoordinatensystem vorliegen.

3. Der Bereich, der dargestellt werden kann, ist meist nicht unendlich, sondern auf einen bestimmten Ausschnitt eingeengt. Im Kameramodell ergibt es wenig Sinn, Objekte darzustellen, die hinter der Kamera verborgen sind oder die sehr weit von der Kamera entfernt sind. Der zweite Schritt, die Ansichtsnormalisierung, überführt deshalb diesen beliebigen Darstellungsraum in einen fest vorgegebenen, normalisierten Bereich. Da jetzt aus diesen Koordinaten die Projektion berechnet werden kann, spricht man von Projektionskoordinaten.

4. Zuletzt wird der eigentliche Projektionsvorgang durchgeführt. Das bedeutet, daß die Projektionskoordinaten jetzt in die zweidimensionalen Gerätekoordinaten umgerechnet werden.

Um Algorithmen für diese Schritte herleiten und darstellen zu können, werden im nächsten Kapitel zuerst einige benötigte mathematische Grundlagen eingeführt. Danach wird ein kurzer Überblick über die verschiedenen Projektionsarten gegeben, bevor die Klassifizierung der Parameter erfolgt, die eine Projektion mathematisch beschreiben. Mit Hilfe dieser Parameter wird anschließend die Vorgehensweise aufgezeigt, mit der man die entsprechenden Projektionen berechnen kann.

2.2 Hilfsmittel der Projektion

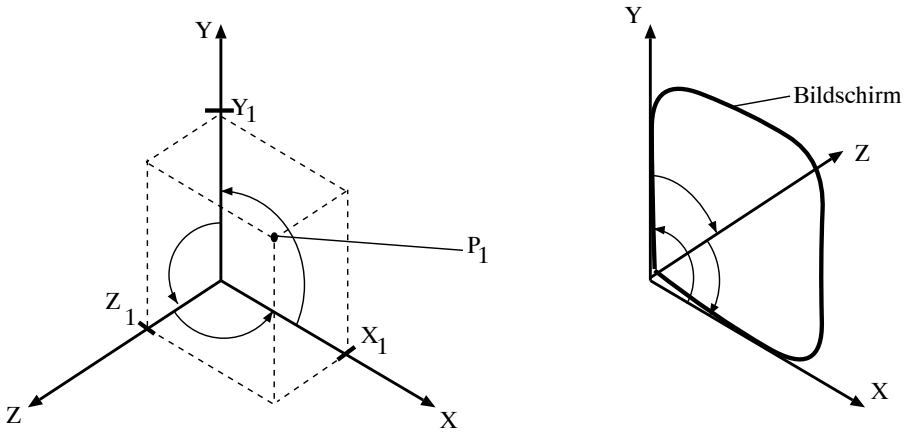
Bevor die Schritte zur Berechnung der Projektion dargestellt werden können, müssen einige Grundlagen über Koordinatensysteme und Transformationen eingeführt werden.

Nur mit diesen Voraussetzungen ist das Verständnis der Schritte zur Generierung der Projektionsmatrix, die für die Ausführung der Projektion verwendet wird, möglich.

2.2.1 Koordinatensysteme und Transformationen

Ein Punkt im dreidimensionalen Raum wird meist durch Angabe dreier skalarer Parameter in einem Koordinatensystem dargestellt. Es gibt die verschiedensten Koordinatensysteme, angefangen von geradlinigen, rechtwinkligen Koordinaten, wie dem kartesischen Koordinatensystem, über krummlinige Koordinatensysteme, bis hin zu Zylinder- oder Kugelkoordinaten. Ohne Beschränkung der Allgemeinheit soll hier nur das kartesische Koordinatensystem betrachtet werden. Andere Koordinatensysteme werden nicht eingeführt; ihre Überführung in kartesiche Koordinaten kann gegebenenfalls der Literatur, z.B. [BRO87], entnommen werden.

Das am häufigsten verwendete Koordinatensystem ist das kartesische, rechtshändige Koordinatensystem wie in Abb. 2.2 dargestellt. Drei Achsen x, y, z stehen paarweise aufeinander senkrecht. Wenn man z.B. von der positiven x-Achse auf den Ursprung des Koordinatensystems schaut, kann man durch eine Drehung um diese Achse im mathematisch positiven Sinne die y-Achse in die z-Achse überführen.



rechtshändiges Koordinatensystem

linkshändiges Koordinatensystem mit überlagertem Bildschirm

Abbildung 2.2: Koordinatensysteme

Linkshändige Koordinatensysteme finden in der Graphik ebenfalls Verwendung, da sie sich günstig auf Bildschirmkoordinaten übertragen lassen. Die intuitive Annahme, daß ein Objekt mit größerer z-Komponente weiter vom Bildschirm entfernt liegt, trifft hier direkt zu. In Abb. 2.2 ist einem linkshändigen Koordinatensystem ein Bildschirmkoordinatensystem direkt überlagert worden. Es findet in dieser Arbeit jedoch ausschließlich das rechtshändige Koordinatensystem Verwendung.

Ein Punkt P_1 in diesem Koordinatensystem wird durch ein Zahlentripel $P_1(x_1, y_1, z_1)$ dargestellt.

Modelle und Objekte werden durch Angabe charakteristischer Punkte, z.B. der Eckpunkte bei einem Würfel, beschrieben. Mit Punkten können verschiedene Transformationen durchgeführt werden. Dies sind die **Translation**, die man für die Verschiebung oder Bahnbewegung eines Punktes verwendet, ferner die **Rotation** oder Drehung, die man für die Blickrichtungsänderung benötigt, sowie die **Skalierung**, die man für die Änderung der Darstellungsgröße (Zoom) benutzt. Die **Scherung** kann man für eine Verzerrung des Bildes benutzen.

2.2.2 Homogene Koordinaten

Um diese Transformationen in mathematisch gleicher Weise darstellen und durchführen zu können, ist ein Übergang zu homogenen Koordinaten sinnvoll. Dadurch kann man einen einheitlichen Formalismus einführen. Mit homogen Koordinaten, auch Verhältniskoordinaten genannt, können alle Transformationen einschließlich der Translation, durch eine Matrixmultiplikation mit einer 4×4-Matrix ausgedrückt werden. Dies hat den Vorteil, daß mehrere Transformationen zusammengefaßt werden können, indem man ihre Transformationsmatrizen miteinander multipliziert.

Anstatt wie bei kartesischen Koordinaten einen Punkt durch ein Zahlentripel $P(x, y, z)$ darzustellen, wird bei homogenen Koordinaten eine vierte Komponente W hinzugefügt. Den Punkt P in homogenen Koordinaten erhält man aus seinen kartesischen Koordinaten, indem man als viertes Element $W = 1$ anhängt. P wird nun durch ein Quadrupel $P(x, y, z, W)$ dargestellt. Zwei Punkte $P_1(x_1, y_1, z_1, W_1)$ und $P_2(x_2, y_2, z_2, W_2)$ in homogenen Koordinaten entsprechen einander, wenn die Quadrupelkomponenten des einen Punktes ein Vielfaches der Komponenten des anderen Punktes sind, also wenn $P_1 = s * P_2$ für ein $s \neq 0$, $s \in \mathbb{R}$ ist.

Man kann einen beliebigen Punkt $P(x, y, z, W)$ also auch durch $P(\frac{x}{W}, \frac{y}{W}, \frac{z}{W}, 1)$ darstellen, indem man seine Komponenten durch W teilt (unter der Voraussetzung $W \neq 0$). Diese Darstellung nennt man homogenisiert. Punkte, deren W Komponente gleich null ist, werden als im Unendlichen liegend betrachtet. Einen Punkt in homogenen Koordinaten kann man, wenn dieser homogenisiert ist ($W = 1$), einfach durch Weglassen der vierten Komponente in kartesische Koordinaten zurückführen.

Für die Rechnungen in den folgenden Kapiteln wird eine spaltenweise Schreibweise der Punkte verwendet. Ein Punkt wird also als einspaltige Matrix wie folgt geschrieben:

$$P_1(x, y, z, W) = \begin{bmatrix} x \\ y \\ z \\ W \end{bmatrix} = [x, y, z, W]^T \tag{2.1}$$

Die Transformation eines Punktes P mit einer Transformationsmatrix M erhält man in dieser Notation, indem man das Matrixprodukt $P' = M \cdot P$ in dieser Reihenfolge bildet.

Zwei Transformationsmatrizen für eine erste Transformation M_1 und eine anschließende Transformation M_2 werden dann zu einer gemeinsamen Transformationsmatrix M zusammengefaßt, indem man $M = M_2 \cdot M_1$ bildet.

Man kann einem Punkt $P(x, y, z)$ in kartesischen Koordinaten auch durch seinen Ortsvektor $\vec{P}$ vom Ursprung zum Punkt P beschreiben. Die Strecke zwischen zwei Punkten P_1 und P_2 kann dann als Richtungsvektor $\vec{P}_s = \vec{P}_1 - \vec{P}_2$ aufgefaßt werden.

2.2.3 Translation

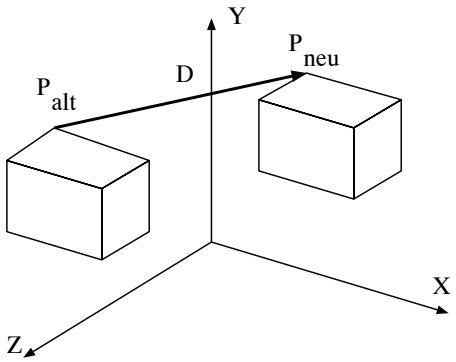


Abbildung 2.3: Translation eines Objektes

Die Translation entspricht der Verschiebung eines Punktes oder eines Objektes um einen festen, vorgegebenen Translationsvektor $D(d_x, d_y, d_z)$ (Abb. 2.3). Ihre Transformationsmatrix lautet:

$$T(d_x, d_y, d_z) = \begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

Die Translation läßt sich dann schreiben als:

$$P_{neu} = T(d_x, d_y, d_z) \cdot P_{alt} \tag{2.3}$$

2.2.4 Skalierung

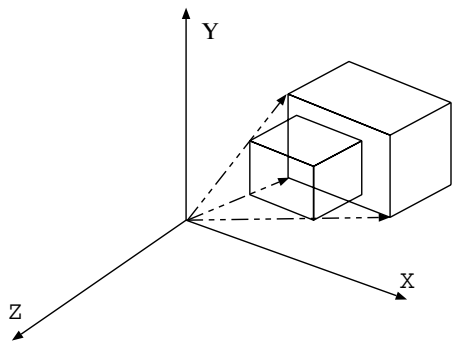


Abbildung 2.4: Skalierung um den Nullpunkt

Die Skalierung entspricht einer Streckung des Ortsvektors eines Punktes P oder eines Objektes um die Streckfaktoren s_x, s_y, s_z in Richtung der drei Koordinatenachsen (Abb. 2.4). Die Skalierung hat einen Fixpunkt, den Ursprung des Koordinatensystems.
Bei der Skalierung werden die Koordinaten von $P(x, y, z)$ mit den entsprechenden Skalierungsfaktoren multipliziert: $P' = [s_x\ x, s_y\ y, s_z\ z]^T$. Bei dieser Transformation ändert sich nicht nur die Größe, sondern auch die Position eines Objektes. Dabei wird der Nullpunkt auf sich selbst abgebildet.

Die Transformationsmatrix S der Skalierung lautet:

$$S(s_x, s_y, s_z) = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Will man die Skalierung um einen anderen Fixpunkt(F_x, F_y, F_z) als um den Ursprung ausführen, so kann man dies durch eine vorherige Translation mit $T_1(-F_x, -F_y, -F_z)$ dieses Punktes (und des zu skalierenden Objektes) in den Nullpunkt, dann durch die Ausführung der Skalierung mit Ursprung im Nullpunkt und danach durch eine Rücktransformation mit $T_2(F_x, F_y, F_z)$ erreichen: $S_F = T_2 \cdot S(s_x, s_y, s_z) \cdot T_1$.

2.2.5 Rotation

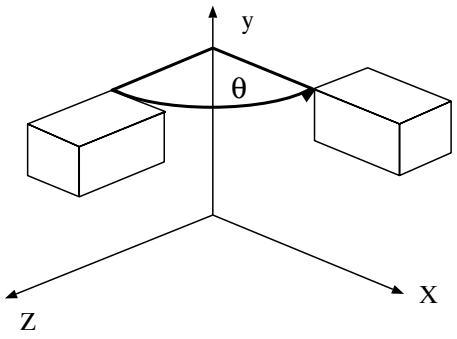


Abbildung 2.5: Rotation um die y-Achse

Die Rotation eines Punktes um eine bestimmte Rotationsachse und den Winkel θ bedeutet eine Drehung um diese Achse im mathematisch positiven Sinne.
Die Rotation um eine beliebige Achse im Raum läßt sich auf drei Rotationen um die Koordinatenachsen x, y, z zurückführen. Ein Beispiel für eine Rotation um die y-Achse zeigt Abb. 2.5.
Eine Rotation um den Winkel θ im mathematisch positiven Sinne um die x-Achse läßt sich mit folgender Rotationsmatrix erreichen:

$$R(\theta)_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\theta) & -sin(\theta) & 0 \\ 0 & sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

Ebenso ergibt sich für die Rotation um die y-Achse und die z-Achse:

$$R(\theta)_y = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(\theta) & 0 & cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R(\theta)_z = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

.

Wenn man eine Rotation um eine beliebig im Raum liegende Drehachse um den vorgegebenen Winkel θ durchführen will, sind dazu sieben Transformationen der Drehachse und des Objektes nötig:

1. Translation, so daß die Drehachse durch den Ursprung verläuft.
2. Rotation um die x-Achse, so daß die Drehachse in der xz-Ebene zu liegen kommt (Drehwinkel α).
3. Rotation um die y-Achse, so daß diese auf der z-Achse zu liegen kommt (Drehwinkel β).
4. Rotation des Objektes um die z-Achse, die jetzt der Drehachse entspricht, um den vorgebenen Rotationswinkel θ.
5. Rücktransformation der Drehachse und des Objektes durch Rotation um die y-Achse mit der Inversen der unter 3. berechneten Rotationsmatrix.
6. Rücktransformation der Drehachse und des Objektes durch Rotation um die z-Achse mit der Inversen der unter 2. berechneten Rotationsmatrix.
7. Translation der Drehachse und des Objektes um den inversen Translationsvektor aus Punkt 1.

Geht man davon aus, daß die Drehachse durch zwei Punkte $P_1(x_1, y_1, z_1)$ und $P_2(x_2, y_2, z_2)$ gegeben ist, ihr Richtungsvektor sich also zu $\vec{v} = \vec{P_1} - \vec{P_2}$ und als Einheitsvektor zu $\vec{u} = \frac{\vec{v}}{|\vec{v}|} := [x, y, z]^T$ berechnen läßt, sind die Schritte wie folgt:

zu 1. Die Translation wird so ausgeführt, daß der Punkt P_1 in den Nullpunkt verschoben wird. Es ist also eine Translation $T_1 = T(-x_1, -y_1, -z_1)$ nötig.

zu 2. Für die erste Rotation muß der Winkel α zwischen der Projektion des Drehachsenrichtungsvektors $\vec{u}'$ in die yz-Ebene und der z-Achse bestimmt werden. Die Rotation muß mit diesem berechneten Winkel ausgeführt werden. Abb. 2.6 zeigt die Lage der entsprechenden Vektoren in einem Beispiel.

Aus der Definition des Skalarproduktes zweier Vektoren $\vec{a}, \vec{b}$ läßt sich eine Formel zur Berechnung des gesuchten Winkels ableiten:

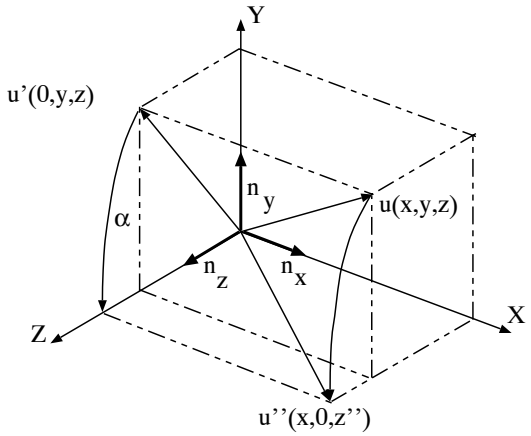


Abbildung 2.6: Beliebige Rotation: Rotation um x-Achse

$$\vec{u}' \cdot \vec{n}_z = cos(\alpha)|\vec{u}'||\vec{n}_z| \quad \text{also} \quad cos(\alpha) = \frac{\vec{u}' \cdot \vec{n}_z}{|\vec{u}'|\ |\vec{n}_z|} \tag{2.7}$$

wobei $\vec{n}_z$ der Einheitsvektor in z-Richtung $\vec{n}_z = (0,0,1)$ ist. Des weiteren ist das Skalarprodukt zweier Vektoren $\vec{v}_1 = [x_1, y_1, z_1]^T$ sowie $\vec{v}_2 = [x_2, y_2, z_2]^T$ bei Verwendung kartesischer Koordinaten definiert zu:

$$\vec{v}_1 \cdot \vec{v}_2 = x_1x_2 + y_1y_2 + z_1z_2 \tag{2.8}$$

Das Skalarprodukt der beiden Vektoren $\vec{u'} = [0, y, z]^T$ und $\vec{n_z} = [0,0,1]^T$ beträgt also in Komponentenschreibweise:

$$\vec{u}' \cdot \vec{n}_z = 0 * 0 + 0 * y + 1 * z = z \tag{2.9}$$

Der Vektor $\vec{n}_z$ ist bereits normiert, der Betrag von $\vec{u}'$ lautet:

$$|\vec{u}'| = \sqrt{y^2 + z^2} := r \tag{2.10}$$

Setzt man dies in Formel 2.7 ein, läßt der Winkel α sich schreiben zu:

$$cos(\alpha) = \frac{z}{r} \tag{2.11}$$

Über die Winkelfunktionen könnte man nun α und damit auch $sin(\alpha)$ berechnen. Man kann jedoch auch das Vektorprodukt zu Hilfe nehmen. Die (betragsweise) Definition des Vektorproduktes lautet:

$$|v_3| = |\vec{v}_1 \times \vec{v}_2| = |\vec{v}_1||\vec{v}_2||sin(\alpha)| \quad \text{also} \quad |sin(\alpha)| = \frac{|\vec{v}_1 \times \vec{v}_2|}{|\vec{v}_1||\vec{v}_2|} \tag{2.12}$$

Die Richtung des resultierenden Vektors ergibt sich durch die Rechte-Hand-Regel, $\vec{v}_1$, $\vec{v}_2$ und $\vec{v}_3$ bilden ein Rechtssystem.

In kartesischen Koordinaten läßt sich das Vektorprodukt auch schreiben als:

$$v_3 = \vec{v}_1 \times \vec{v}_2 = (y_1 z_2 - y_2 z_1, x_1 z_2 - x_1 z_2, x_1 y_2 - x_2, y_1) \tag{2.13}$$

Danach beträgt das Vektorprodukt $\vec{u}' \times \vec{n}_z = [y, 0, 0]^T$. Der Richtungsvektor des von beiden Vektoren gebildeten Vektorproduktes ist $\vec{u}_x = [1, 0, 0]^T$. Diese Richtung stimmt mit der Richtung der Drehachse überein, daher kann von $sin(\alpha)$ der Betrag weggelassen werden. Damit läßt sich $sin(\alpha)$ mit Formel 2.12 bestimmen zu:

$$sin(\alpha) = \frac{y}{r} \tag{2.14}$$

Die Rotationsmatrix lautet für die Rotation um die x-Achse nach Formel 2.5 also :

$$R_x(y, z, r) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{z}{r} & -\frac{y}{r} & 0 \\ 0 & \frac{y}{r} & \frac{z}{r} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.15}$$

zu 3. Nach der Rotation um die x-Achse um den eben berechneten Winkel liegt der Richtungsvektor der Rotationsachse $\vec{u''}$ in der xz-Ebene (Abb. 2.6). Der Betrag dieses Vektors ist 1 (da $\vec{v}$ bereits normiert war), der Vektor ist also normiert.

Betrachtet man nun ausschließlich die xz-Ebene, so kann man den Rotationswinkel β für die Rotation um die y-Achse bestimmen (Abb. 2.7).

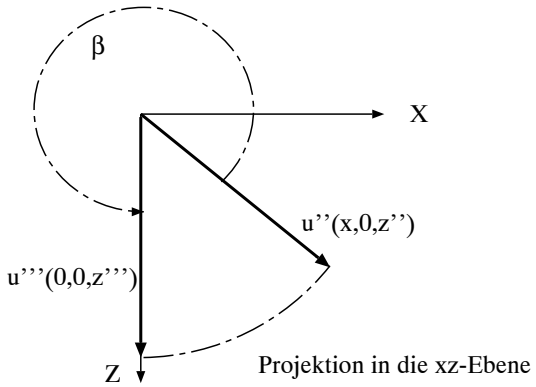


Abbildung 2.7: Beliebige Rotation: Rotation um y-Achse

Das Skalarprodukt von $\vec{u''}$ und dem Einheitsvektor in z-Richtung $\vec{n}_z$ ermöglicht es, wie oben beschrieben, $cos(\beta)$ zu bestimmen, das Vektorprodukt liefert $sin(\beta)$:

$$cos(\beta) = \frac{\vec{u''} \cdot \vec{n_z}}{|\vec{u''}| \cdot |\vec{n_z}|} = z'' \quad \text{und} \quad sin(\beta) = -\frac{|\vec{u''} \times \vec{n_z}|}{|\vec{u''}| \cdot |\vec{n_z}|} = -x \tag{2.16}$$

Da der Richtungsvektor des Vektorproduktes entgegengesetzt dem Richtungsvektor der Drehachse liegt, ist das negative Vorzeichen zu wählen. Die Rotationsmatrix lautet für die Rotation um die y-Achse nach Formel 2.6 also:

$$R_y(x, z'') = \begin{bmatrix} z'' & 0 & -x & 0 \\ 0 & 1 & 0 & 0 \\ x & 0 & z'' & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.17}$$

zu 4. Nun ist die Drehachse mit der z-Achse identisch. Man kann also die gewünschte Rotation um den vorgegebenen Winkel θ nach Formel 2.6 mit folgender Rotationsmatrix ausführen:

$$R_z = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 & 0 \\ sin(\theta) & cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.18}$$

zu 5.-7. In den Schritten 5 - 7 wird jeweils die gleiche Transformation wie in Schritt 1-3, jedoch mit den Inversen der Transformationsmatrizen T_1^{-1}, R_x^{-1} und R_y^{-1} ausgeführt.

Insgesamt kann man die Transformationsmatrix dann berechnen zu:

$$R_{gesamt} = T_1^{-1} \cdot R_y^{-1} \cdot R_x^{-1} \cdot R_z \cdot R_y \cdot R_x \cdot T_1 \tag{2.19}$$

2.2.6 Scherung

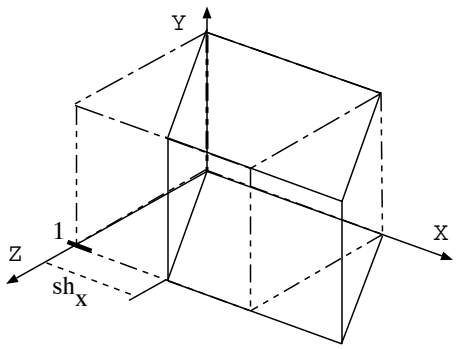


Abbildung 2.8: Scherung in der xy-Ebene

Bei der Scherung (Abb. 2.8) werden zwei Koordinaten in Abhängigkeit von der dritten Koordinate verändert. Eine Scherung erfolgt stets entlang einer Achse in der Ebene, die durch die anderen beiden Achsen aufgespannt wird: Schert man einen Punkt $P_1 = [x_1, y_1, z_1]^T$ entlang der z-Achse mit der Scherfunktion sh_x bzw. sh_y, so ergibt sich der Punkt $P_1' = [x_1 + sh_x\ z_1, y_1 + sh_y\ z_1, z_1]^T$. Die Transformationsmatrix für eine Scherung entlang der z-Achse in der xy-Ebene lautet dann:

$$SH_{xy}(sh_x, sh_y) = \begin{bmatrix} 1 & 0 & sh_x & 0 \\ 0 & 1 & sh_y & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.20}$$

Die Scherungsmatrizen entlang der anderen beiden Achsen lassen sich entsprechend bilden.

Abb.2.8 zeigt eine Scherung entlang der z-Achse, wobei zur besseren Erkennbarkeit $sh_x \neq 0$ und $sh_y = 0$ gewählt wurde. Punkte, die auf der $z = 0$-Ebene liegen (hier Rückwand des Quaders), werden auf sich selbst abgebildet.

2.3 Arten der Projektion

Bevor man mit der Berechnung einer Projektion beginnen kann, soll ein kurzer Überblick über die im Graphikbereich üblicherweise verwendeten Projektionen erfolgen. Hier wird nur die planare geometrische Projektion behandelt, da bei diesen Projektionen Bilder erzeugt werden, die dem natürlichen Empfinden des Betrachters nahekommen. Planar bedeutet, daß die Projektion auf ein ebene Fläche, die Bildschirmfläche, und nicht z.B. auf eine Kugeloberfläche erfolgt. Die planaren Projektionen zeichen sich dadurch aus, daß Geraden stets auf Geraden abgebildet werden. Dadurch ist es für Geradenstücke ausreichend, die Endpunkte abzubilden und zu verbinden.

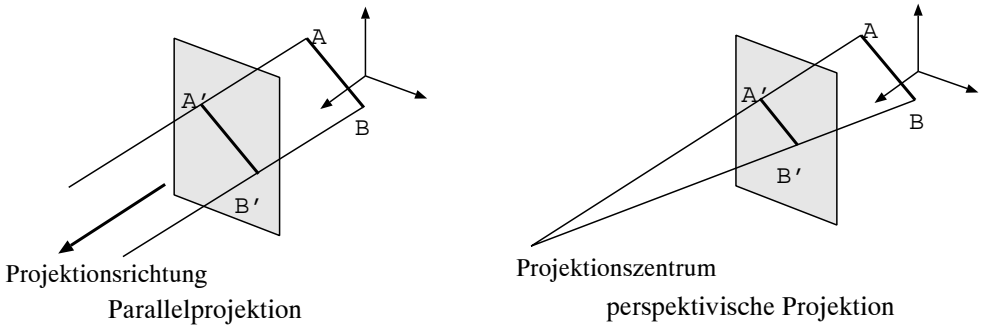


Abbildung 2.9: Projektionsarten

Anschaulich wird bei einer Projektion eine Gerade mit vorgegebener Richtung von jedem Punkt eines Objektes aus mit der Abbildungsebene geschnitten. Dieser Schnittpunkt stellt den Projektionspunkt des Objektpunktes dar. Je nach Festlegung der Richtung der Geraden kann man die Projektionen in zwei Grundklassen unterteilen:
Parallelprojektionen, bei denen die Projektionsgeraden alle die gleiche Richtung haben (also zueinander parallel sind), sowie perspektivische Projektionen, bei denen die Projektionsgeraden auf einem festen Punkt ausgerichtet sind.

2.3.1 Parallelprojektion

Bei der Parallelprojektion (Abb. 2.10) werden parallele Linien und Geraden, solange sie nicht parallel zur Projektionsrichtung liegen, stets wieder auf parallele Linien abgebildet. Stehen sie parallel zur Projektionsrichtung, so ist ihr Bild ein einzelner Punkt auf der Projektionsebene.
Linien, die parallel zur Abbildungsebene liegen, werden unabhängig von ihrer Entfernung zur Abbildungsebene gleich lang projieziert. Dies ist allerdings auch der große Nachteil der Parallelprojektion. Man kann aufgrund des Bildes nicht klar entscheiden, welche Linien wie weit vom Betrachter entfernt liegen.
Dafür bleiben Winkel und Längen auf Ebenen, die parallel zur Abbildungsebene liegen, erhalten. Parallelprojektion wird deshalb häufig auch für die Darstellung von Grund-, Auf- und Seitenrissen verwendet.

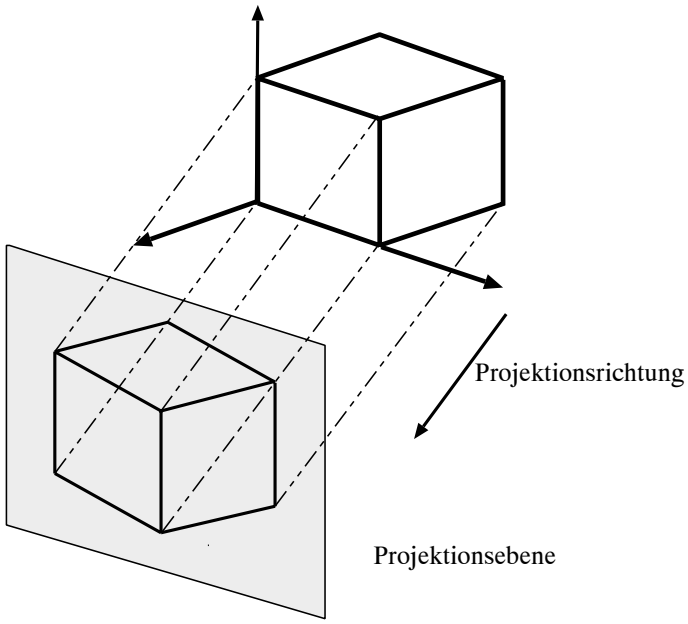


Abbildung 2.10: Parallelprojektion

Die Parallelprojektion wird in zwei unterschiedliche Klassen unterteilt:

- Bei der orthogonalen Projektion steht die Projektionsrichtung senkrecht auf der Projektionsebene, ist also parallel zur Normalen der Projektionsebene. Die häufigsten orthogonalen Projektionen sind die oben erwähnten Grund-, Auf- und Seitenrisse. In diesen Fällen liegt die Projektionsachse parallel zur Richtung einer der Hauptachsen des Koordinatensystems.

- Im anderen Fall, wenn also die Projektionsrichtung nicht senkrecht auf der Projektionsebene liegt, spricht man von schräger oder „obliquer“ Projektion. Bekannte schräge Parallelprojektionen sind die Vogelperspektive sowie die Kavaliers- und die Kabinettperspektive, bei denen eine Verkürzung einer Achse für eine Verzerrung der Projektion sorgt, um die Größenverhältnisse leichter erkennbar zu machen.

2.3.2 Perspektivische Projektion

Die perspektivische Projektion, auch Zentralprojektion genannt, zeichnet sich dadurch aus, daß sich sämtliche Projektionslinien in einem Projektionszentrum treffen (Abb.2.11). Diese Art der Projektion kommt der Sicht des menschlichen Auges und der Art und Weise, wie eine Kamera Bilder erzeugt, sehr nahe.

Bei der perspektivischen Projektion werden parallele Geraden, die nicht parallel zur Abbildungsebene verlaufen, nicht auf parallele Geraden abgebildet, sondern treffen sich in einem Punkt, genannt Flucht- oder Verschwindungspunkt.

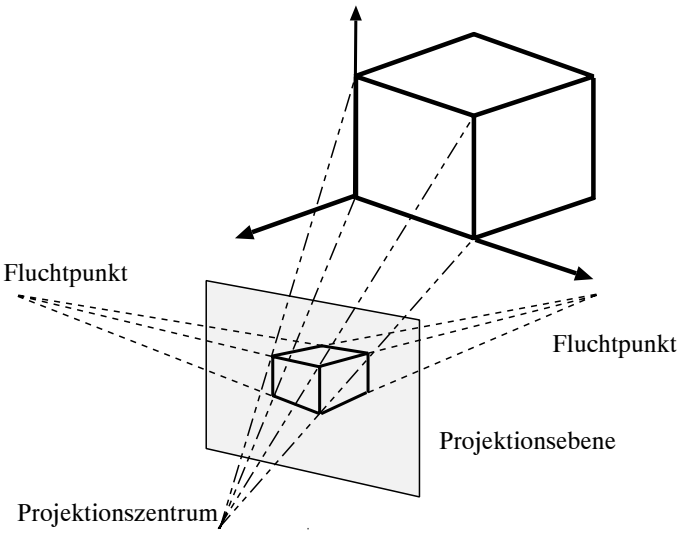


Abbildung 2.11: Perspektivische Projektion

Geradenstücke, die parallel zur Bildebene liegen, werden je nach Entfernung zur Bildebene unterschiedlich groß abgebildet. Dadurch, daß weiter entfernt liegende Streckenabschnitte auf eine kürzere Strecke abgebildet werden, entsteht ein räumlicher Eindruck. Je nachdem, ob zwei, eine oder keine der Hauptachsen des dreidimensionalen Raumes parallel zur Bildebene liegen, spricht man von einer Ein-, Zwei- oder Dreipunktperspektive.

Man kann die Parallelprojektion auch als Sonderfall der perspektivischen Projektion betrachten. Verschiebt man das Projektionszentrum ins Unendliche, geht die perspektivische Projektion in die Parallelprojektion über.

2.4 Projektionsparameter

In der Einleitung waren die Angaben, die zur Festlegung einer Kamera benötigt werden, kurz anschaulich erläutert worden. Nun muß ein Weg gefunden werden, aus diesen Angaben eine Beschreibung durch Variable zu finden, mit denen dann die gewünschte Projektion berechnet werden kann.

Die Beschreibung ist hierbei an die in PHIGS (Programmer's Hierarchical Interactive Graphics System) [GAS91] verwendete Form angelehnt, da diese im Graphikbereich als Standard definiert ist. Diese Beschreibung wird ebenfalls von GKS-3D [FOL90] verwendet. Da PHIGS in Amerika entwickelt wurde, und in der Graphikprogrammierung fast ausschließlich die englische Sprache in der Literatur verwendet wird, werden hier ebenfalls die englischen Begriffe eingeführt.

Am Anfang dieses Kapitels war der deutsche Begriff „Ansicht“ verwendet worden. Im englischen wird dieser Begriff mit „view“ übersetzt und wird hier auch dementsprechend Verwendung finden.

Zuerst wird die Beschreibung der Projektionsparameter in PHIGS aufgeführt, anschließend

daran wir kurz dargestellt, wie diese Parameter auf das Kameramodell übertragen werden können.

2.4.1 Darstellung der Projektionsparameter in PHIGS

Schon beim Kameramodell mußte zuerst ein spezieller Punkt, der Standort der Kamera, festgelegt werden. PHIGS definiert diesen Punkt mit Hilfe des View Reference Point (VRP). In diesen Punkt wird eine Ebene gelegt, die der Abbildungsebene einer Kamera entspricht. Sie wird mit View Reference Plane bezeichnet. Zur Definition der Ausrichtung dieser Ebene dient der Normalenvektor der Ebene, genannt View Plane Normal (VPN), der im Kameramodell der Blickrichtung entspricht. Diese Ebene ist häufig identisch mit der eigentlichen Projektionsebene View Plane (VP), allgemein ist die View Plane jedoch als eine zur View Reference Plane parallele Ebene im Abstand D in Normalenrichtung (VPN) gegeben.

Da das Abbildungsfenster (View Window) auf der View Plane nicht unendlich groß ist, muß neben der Normalenrichtung der View Plane noch die Ausrichtung und Ausdehnung (Größe) des Abbildungsfensters angegeben werden. Zur Festlegung der Ausrichtung dient der View Up Vector (VUP), der senkrecht auf der Unterkante des View Windows steht. Im Kameramodell entspricht dies der Normalen auf den Bildhorizont.

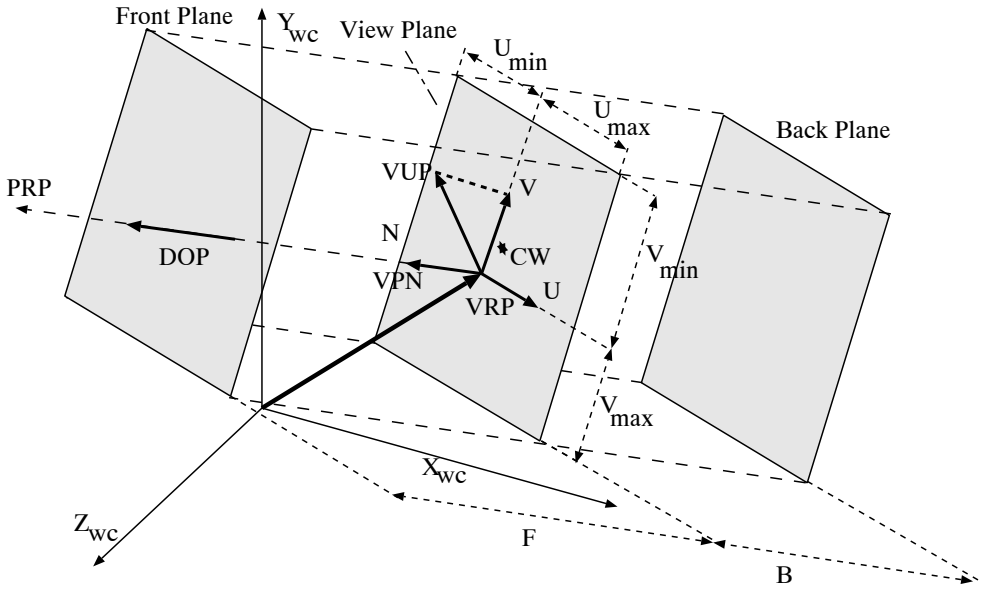


Abbildung 2.12: Festlegung einer Parallelprojektion

Um die Allgemeinheit nicht zu beschränken, muß der VUP nicht senkrecht auf dem VPN stehen, man verwendet vielmehr zur Festlegung eines neuen Koordinatensystems seine Projektion auf die View Reference Plane in Richtung des Normalenvektors (VPN). Im VRP kann man nun ein neues Koordinatensystem UVN definieren, das durch VPN als N-Vektor, die Projektion von VUP als V-Vektor und einen dritten Vektor U so festgelegt

wird, daß diese drei Vektoren in der Reihenfolge UVN ein Rechtssystem bilden. Dieses Koordinatensystem heißt View Reference Coordinate System (VRC). Die drei Vektoren VRP, VPN und VUP werden in Welt- bzw. Szenenkoordinaten angegeben. Nun kann eine seitliche Begrenzung des View Fensters in VRC-Koordinaten vorgenommen werden. Gibt man die Begrenzung als U_{min} und U_{max} bzw. V_{min} und V_{max} an, so kann man den Mittelpunkt dieses Fensters, genannt Center Window (CW) berechnen, der jedoch nicht mit dem VRP identisch zu sein braucht.

Zuletzt wird noch ein weiterer Punkt in VRC-Koordinaten angegeben, der Projection Reference Point (PRP) genannt wird. Bei der Parallelprojektion wird der Vektor vom Mittelpunkt des Abbildungsfensters CW zum PRP als Projektionsrichtung herangezogen, bei der perspektivischen Projektion gibt dieser das Projektionszentrum an.

Bei der Parallelprojektion wird der so definierte Projektionsvektor Direction of Projection (DOP) genannt. Der DOP ist bei orthogonaler Parallelprojektion parallel zum VPN, bei schiefer oder obliquer Projektion ist er dies nicht. Mit dieser Definition kann man einen unendlichen langgestreckten Quader bestimmen, der den Bildraum begrenzt . Dieser Quader wird durch zur Projektionsrichtung parallele Geraden durch die Ecken des View Window gebildet. Man kann diesen Quader noch begrenzen, indem man ihn durch zwei zur View Plane parallele Ebenen, die Front Plane im Abstand F zum VRP, und die Back Plane im (negativen) Abstand B zum VRP, beschneidet. Man hat damit einen Abbildungsquader definiert; nur Elemente, die innerhalb dieses Quaders liegen, sollen bei der Projektion berücksichtigt werden, andere Elemente durch Ausblenden (Clipping) bei der Darstellung unterdrückt werden (vgl. Abb. 2.12, View Plane identisch mit View Reference Plane).

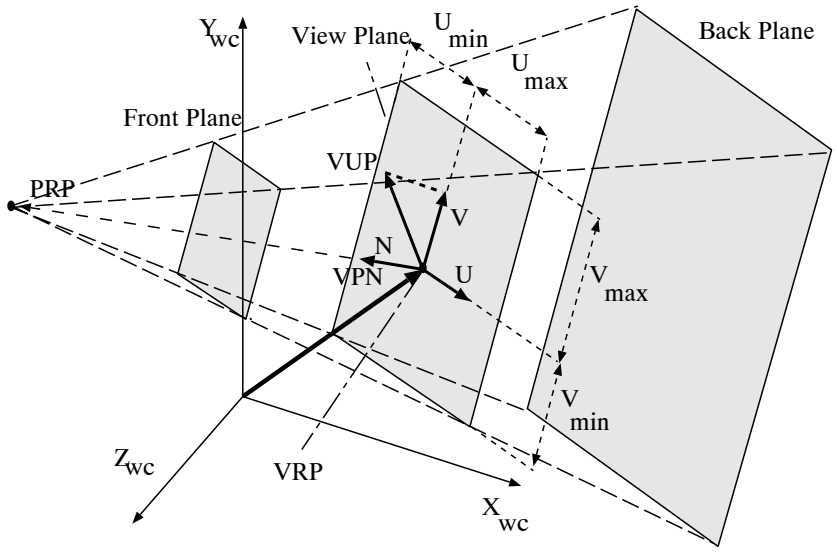


Abbildung 2.13: Festlegung einer perspektivischen Projektion

Bei der perspektivischen Projektion läßt sich eine Abbildungspyramide bilden, deren Spitze der PRP ist und deren Kanten durch die Verbindungsgeraden zwischen PRP und den Ecken des View Window gebildet wird. Betrachtet man diese Pyramide als unendlich weit ausgedehnt, so können alle Punkte, die innerhalb der Seitenflächen dieser Pyramide liegen,

abgebildet werden. Punkte außerhalb dieser Pyramide sollen nicht angezeigt werden. Genau wie bei der Parallelprojektion kann man diese Pyramide durch Front Plane und Back Plane noch begrenzen (Abb. 2.13).

2.4.2 Verwendung der Projektionsparameter im Kameramodell

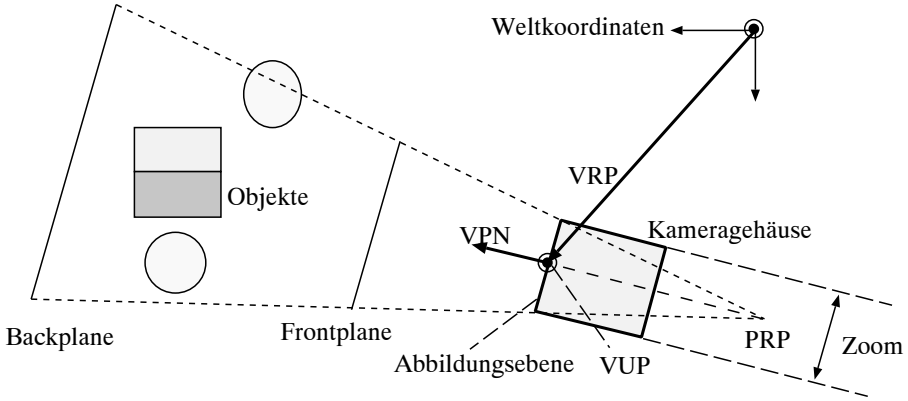


Abbildung 2.14: Übertragung der PHIGS Angaben auf das Kameramodell

Man kann diese Projektionsbeschreibung nun vollständig auf das Kameramodell übertragen (Abb. 2.14). Wie oben angedeutet, kann man den VRP als Standort der Kamera und als Mittelpunkt ihrer Projektionsebene festlegen. Durch VPN wird dann die Blickrichtung der Kamera angegeben, VUP steht senkrecht auf der Bildhorizontalen. Mit PRP wird die Abbildungseigenschaft der Kamera festgelegt, dieser sollte deshalb hinter dem VRP in der Mitte des Fensters plaziert werden, wenn die Kamera die Bilder nicht verzerren soll. Über einen skalaren Zoomfaktor kann man die Größe des View Windows, festgelegt durch U_{min}, U_{max}, V_{min} und V_{max}, an die Bildausschnittsgröße der Kamera anpassen.

Damit ist auch festgelegt, welche Parameter bei einer Bewegung der Kamera verändert werden müssen: Dies sind einerseits VRP als Ortsbeschreibung, andererseits VPN und VUP als Orientierungsbeschreibung und zuletzt der skalare Zoomfaktor. PRP, wie auch der Abstand von Front- und Backplane sind dagegen abbildungstechnische Größen der Kamera und hängen nicht von ihrer Bewegung ab.

2.5 Berechnung von Projektionen

Die Berechnung einer Projektion ist einfach herzuleiten, wenn die Projektionsparameter auf eine bestimmte Art im Raum angeordnet sind. Deshalb wird bei der Entwicklung dieser Formeln von dieser festen, vorgegebenen Anordnung der Projektionsparameter ausgegangen. Im nachfolgenden Kapitel werden dann die Transformationen aufgezeigt, die die Abbildung eines beliebigen Bildraumes in einen Bildraum, der dieser Anordnung der Projektionsparameter genügt, leisten können.

Auch für die Projektion wird hier wieder eine Transformationsmatrix entwickelt, mit deren Hilfe man die Punkte im dreidimensionalen Bildraum auf die zweidimensionale Darstellungsebene abbilden kann.

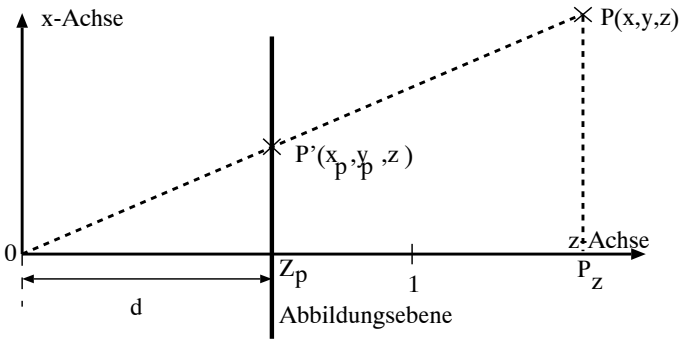


Abbildung 2.15: Zur Berechnung einer perspektivischen Projektion

Bei der perspektivischen Projektion wird davon ausgegangen, daß das Projektionszentrum im Nullpunkt liegt, die Bildebene senkrecht zur z-Achse liegt und der Abstand d vom Nullpunkt zur Bildebene zwischen 0 und 1 liegt. Die Projektion läßt sich dann durch einfache geometrische Betrachtungen herleiten.

Die Projektion des Punktes $P(x, y, z)$ auf eine Bildebene im Abstand d zum Nullpunkt läßt sich durch das Aufstellen von Dreiecksverhältnissen berechnen. Betrachtet man die $y = 0$-Ebene, so hat man den Fall in Abb. 2.15 vorliegen. Aus den Dreiecksverhältnissen der beiden ähnlichen Dreiecke $OP'Z_p$ und OPP_z in Bild 2.15 kann man x_p bestimmen:

$$\frac{x_p}{d} = \frac{x}{z} \quad , \quad x_p = \frac{d\,x}{z} \tag{2.21}$$

Ebenso läßt sich, betrachtet man die $x = 0$ Ebene, aus entsprechenden Dreiecksverhältnissen y_p bestimmen:

$$\frac{y_p}{d} = \frac{y}{z} \quad , \quad y_p = \frac{d\,y}{z} \tag{2.22}$$

Hat man einen Punkt $P = [x, y, z, 1]^T$ in homogenen Koordinaten vorliegen, so genügt eine Matrixmultiplikation mit folgender Matrix, um die Projektion zu berechnen:

$$M_{per} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \tag{2.23}$$

Als Ergebnis bekommt man einen Punkt $P' = [x, y, z, \frac{z}{d}]^T$, homogenisiert ergibt sich daraus $P' = [\frac{d\,x}{z}, \frac{d\,y}{z}, d, 1]^T$. Die ersten beiden Komponenten stellen den Bildpunkt auf der Abbildungsebene dar.

Die Parallelprojektion läßt sich für einen Spezialfall sogar noch einfacher berechnen als die perspektivische Projektion. Geht man davon aus, daß die Bildebene senkrecht zur z-Achse

liegt und durch den Ursprung geht, und die Projektionsrichtung parallel zur z-Achse ist, dann kann man die Abbildung eines Punktes $P(x, y, z)$ berechnen, indem man einfach seine z-Komponente unterdrückt. Als Matrix für einen Punkt in homogenen Koordinaten bedeutet dies:

$$M_{par} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.24}$$

2.5.1 Abbildung in den Einheitsraum

Eine Projektion, die durch die oben eingeführten Parameter von PHIGS festgelegt ist, muß in eine Darstellung überführt werden, die den Voraussetzungen der oben genannten Projektionen entspricht. Danach kann man die vorgestellten Transformationsmatrizen zur Berechnung der Projektion verwenden.

Als Vorgaben von PHIGS waren als Festlegung für das VRC-Koordinatensystem VRP, VPN und VUP eingeführt worden; Abstand und Größe der View Plane waren durch D, $U_{min/max}$ und $V_{min/max}$, Lage von Front- und Backplane durch F und B gegeben.

Man kann nun durch Translation, Skalierung, Rotation und Scherung des so definierten Abbildungssystemes erreichen, daß dieses in einen Einheitsraum abgebildet wird. Der Einheitsraum erfüllt nicht nur die Bedingungen, die für die Abbildung im vorherigen Abschnitt eingeführt wurden, durch die vordefinierte Begrenzung des sichtbaren Raumes kann man auch das Ausblenden des Objektes an der Abbildungspyramide oder am Abbildungsquader vereinfachen. Abb. 2.16 zeigt die Einheitspyramide bzw. den Einheitswürfel, die Verwendung finden.

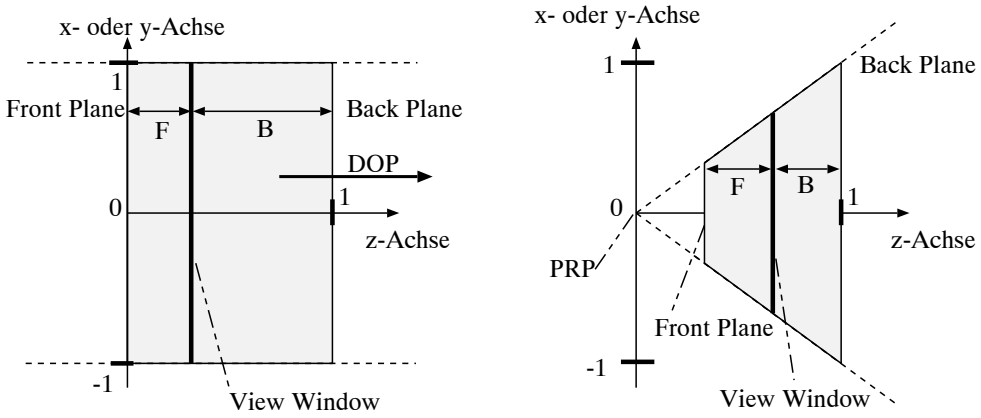


Abbildung 2.16: Einheitswürfel und Einheitspyramide

Einheitspyramide bei perspektivischer Projektion:

Für die perspektivische Projektion sind die Schritte bei der Abbildung in die Einheitspyramide folgende :

1. Translation des VRP in den Ursprung.
2. Rotation des VRC, so daß der Normalenvektor VPN parallel zur z-Achse zu liegen kommt.
3. Translation des Projektionszentrums, gegeben durch den PRP, in den Ursprung.
4. Scherung, so daß die Hauptachse der Abbildungspyramide in der z-Achse liegt.
5. Skalierung der Abbildungspyramide, so daß sie zur Einheitspyramide wird.

Sollte die View Plane nicht mit der View Reference Plane identisch sein, sondern im Abstand D von ihr entfernt liegen, so muß man erst $VRP_{neu} = VRP + D * VPN$ berechnen.

zu 1. Die Translation des VRP erfolgt mit der Translationsmatrix $T_{trans}(-VRP)$

zu 2. Die Rotation $R_x(\theta)$ um die x-Achse erfolgt mit der in Kapitel 2.2.5 angegebenen Formel. Dazu setzt man $cos(\theta) = \frac{z}{z'}$ und $sin(\theta) = \frac{y}{z'}$ mit $z' = \sqrt{y^2 + z^2}$ und x, y, z als den Koordinaten des Richtungsvektors VPN analog zur Herleitung von Formel 2.16.

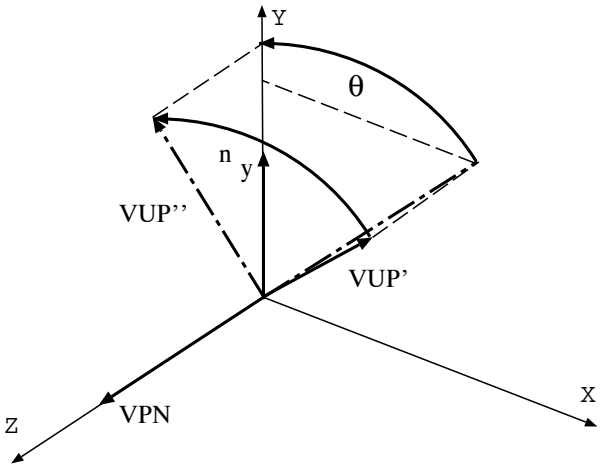


Abbildung 2.17: Rotation um die z-Achse

Für die Rotation $R_y(\theta)$ um die y-Achse mit der Formel 2.6, verwendet man $cos(\theta) = z'$ und $sin(\theta) = x$ ebenfalls analog zu dieser Herleitung. VPN liegt nun korrekt orientiert auf der z-Achse im Ursprung. VUP wurde durch diese Transformationen ebenfalls verändert und auf VUP' abgebildet. Man muß jedoch noch eine Rotation so um die z-Achse durchführen, daß VUP' auf der yz-Ebene zu liegen kommt. Abb. 2.17 zeigt die momentane Lage von VPN und VUP'.

VUP' läßt sich dabei bestimmen zu:

$$VUP'(x'_{up}, y'_{up}, z'_{up}) = R_y \cdot R_x \cdot VUP \tag{2.25}$$

$cos(\theta)$ und $sin(\theta)$ für die dritte Rotation lassen sich dann bestimmen zu:

$$cos(\theta) = \frac{y'_{up}}{l} \qquad sin(\theta) = \frac{x'_{up}}{l} \qquad l = \sqrt{x'^2_{up} + y'^2_{up}} \tag{2.26}$$

und in die Rotationsmatrix $R_z(\theta)$ aus Formel 2.6 einsetzen.

Insgesamt ist also die Rotation $R = R_z \cdot R_y \cdot R_x$ zu bilden.

Es sind nun Weltkoordinatensystem und VRC identisch, sämtliche Punkte, die im VRC angegeben sind, liegen jetzt mit den gleichen Koordinaten auch im Weltkoordinatensystem.

zu 3. Der dritte Schritt ist nun wieder eine einfache Translation mit der Transformationsmatrix $T_{per}(-PRP)$. Dadurch sind VRC und Weltkoordinatensystem bereits wieder nicht mehr identisch.

zu 4. Da die Achse von der Mitte des Projektionsfensters zum Projektionszentrum nicht unbedingt mit der z-Achse identisch ist, muß dies durch eine Scherung sichergestellt werden. Dadurch kommt das Projektionsfenster symmetrisch zur z-Achse zu liegen. Abb. 2.18 zeigt eine mögliche Lage des Projektionsfensters in der yz-Ebene.

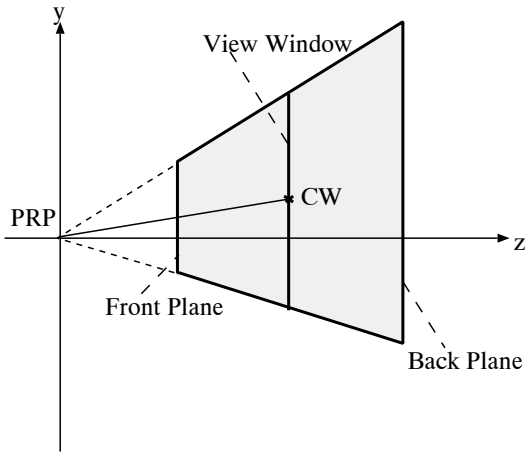


Abbildung 2.18: Bestimmung der Scherungsparameter

Die Scherung muß so vorgenommen werden, daß die z-Komponente unbeeinflußt bleibt, also eine Scherung in xy-Richtung. Für die Scherung entscheidend ist, daß die Achse *PRP-CW* auf der z-Achse zu liegen kommt. Da *PRP* hier ja im Ursprung liegt, ist die Richtung dieses Vektors mit der des Ortsvektors von *CW* in Weltkoordinaten identisch.
Hat man *CW* nicht gegeben, sondern nur U_{min} und U_{max} bzw.V_{min} und V_{max} als Begrenzungen des Abbildungsfensters, kann man *CW* in *VRC*-Koordinaten einfach berechnen zu:

$$CW_x = \frac{u_{min} + u_{max}}{2}, \qquad CW_y = \frac{v_{min} + v_{max}}{2}, \qquad CW_z = 0 \tag{2.27}$$

und in Weltkoordinaten übertragen. Mit $CW_{Welt} = CW - PRP$ lassen sich über Dreiecksverhältnisse die Scherparameter bestimmen zu:

$$sh_x = \frac{x_{CW_{Welt}}}{z_{CW_{Welt}}}, \quad sh_y = \frac{y_{CW_{Welt}}}{z_{CW_{Welt}}} \tag{2.28}$$

Die Schermatrix SH_{per} kann man Kapitel 2.2.6, Formel 2.20, entnehmen. Nach der Scherung liegt das View Window symmetrisch zur z-Achse.

zu 5. Zuletzt muß man noch die Skalierung des Abbildungsraumes in die Einheitspyramide vornehmen.
Da B und F als Abstand von Back- und Frontplane in VRC-Koordinaten gegeben waren, VRC und Weltkoordinaten aber nicht mehr identisch sind, muß man zunächst die jetzige Lage des VRP in Weltkoordinaten bestimmen.

VRP war nach den ersten beiden Schritten im Ursprung ($[0,0,0,1]^T$), danach war er durch die Translation $T(-PRP)$ und die Scherung verändert worden. VRP' läßt sich also jetzt berechnen zu:

$$VRP'(x'_{vrp}, y'_{vrp}, z'_{vrp}) = SH_{xy} \cdot T_{PRP} \cdot [0,0,0,1]^T \tag{2.29}$$

Es müssen zwei Skalierungen vorgenommen werden: Zuerst muß das View Window so skaliert werden, daß dessen Ecken auf den Hauptdiagonalen der xz-Ebene bzw. yz-Ebene zu liegen kommen. Dazu werden folgende Skalierungsfaktoren benötigt:

$$sk_x = \frac{z'_{vrp}}{\frac{u_{max}-u_{min}}{2}}, \quad sk_y = \frac{z'_{vrp}}{\frac{v_{max}-v_{min}}{2}} \tag{2.30}$$

Danach müssen alle drei Achsen so skaliert werden, daß die Back Plane durch den Punkt $z = 1$ geht. Der Skalierungsfaktor dafür lautet für alle drei Achsen:

$$sk_2 = \frac{1}{z'_{vrp} + B} \tag{2.31}$$

Insgesamt erfolgt also eine Skalierung mit der Skalierungsmatrix:

$$S_{per}\left(\frac{2\ z'_{vrp}}{(u_{max} - u_{min})\ (z'_{vrp} + B)}, \frac{2\ z'_{vrp}}{(v_{max} - v_{min})\ (z'_{vrp} + B)}, \frac{1}{z'_{vrp} + B}\right) \tag{2.32}$$

Man hat nun die Abbildungspyramide vollständig in die Einheitspyramide überführt.

Einheitswürfel bei paralleler Projektion

Um einen Abbildungsquader für eine Parallelprojektion auf den Einheitswürfel abzubilden, sind die gleichen Schritte wie bei der perspektivischen Projektion mit Ausnahme des Schrittes 3 (Transformation des PRP) nötig.

zu 1. und 2. Schritt 1 und 2 , Translation und Rotation, sind identisch mit den für die perspektivische Projektion beschriebenen Schritten.

zu 3. Der dritte Schritt entfällt völlig.

zu 4. Die Scherparameter sind bei der Parallelprojektion anders zu berechnen. Da der PRP nicht im Ursprung liegt, muß zuerst der nur für die Parallelprojektion eingeführte Projektionsrichtungsvektor DOP, gegeben durch $DOP = CW - PRP$, berechnet werden. Mit $DOP = [x_{DOP},\ y_{DOP},\ z_{DOP}]^T$ ergeben sich wieder die Scherparameter:

$$sh_x = \frac{x_{DOP}}{z_{DOP}}, \quad sh_y = \frac{y_{DOP}}{z_{DOP}} \tag{2.33}$$

Für den Spezialfall der orthogonalen Projektion sind x_{DOP} und y_{DOP} gleich null. Da die Projektionsrichtung parallel zur z-Achse liegt, sind in diesem Fall die Scherparameter null, es muß also keine Scherung vorgenommen werden.

zu 5. Da der dritte Schritt entfallen ist, liegt jetzt der VRP und daher das View Window auf der $z = 0$-Ebene. Es soll jedoch die Front Plane auf dieser Ebene zu liegen kommen. Dies wird durch eine Translation mit folgendem Translationsvektor erreicht:

$$T_{par}(-\frac{u_{max} + u_{min}}{2}, -\frac{v_{max} + v_{min}}{2}, -F) \tag{2.34}$$

Nun muß noch in allen drei Richtungen skaliert werden, damit die Ausmaße des Abbildungsquaders im Einheitswürfel zu liegen kommen:

$$S_{par}(\frac{2}{u_{max} - u_{min}}, \frac{2}{v_{max} - v_{min}}, \frac{1}{F - B}) \tag{2.35}$$

Mit diesen Schritten hat man ebenfalls die Abbildung in den Einheitsraum erzeugt.

2.5.2 Gesamter Abbildungsvorgang

Die einzelnen Schritte bei der Berechnung der Projektion lassen sich den Schritten der anfangs eingeführten Projektionspipeline zuordnen:

- Die Modelltransformation ist hier nicht behandelt worden und ist abhängig von den zur Modellbeschreibung verwendeten Koordinatensystemen.
- Die Ansichts- oder auch View Transformation ist für beide Darstellungsarten gleich, sie wird durch die Transformationsmatrix:

  $$T_{trans} = R \cdot T(-VRP) \tag{2.36}$$

  beschrieben. In PHIGS sind die dazugehörigen Daten in der View Orientation Matrix gespeichert.
- Die Ansichts- oder View Normalisation hängt von der Projektionsart ab, sie lautet:

  $$T_{trans} = S_{par} \cdot T_{par} \cdot SH_{par} \tag{2.37}$$

für Parallelprojektion und:

$$T_{trans} = S_{per} \cdot SH_{per} \cdot T_{per} \tag{2.38}$$

für die perspektivische Projektion.

In PHIGS sind die jeweils zugehörigen Parameter in der View Mapping Matrix abgespeichert.

- Der letzte Schritt umfaßt eventuell zusätzlich noch ein Ausblenden oder Clippen der Objekte am Einheitsraum (gegeben durch die Seiten des Abbildungswürfels bzw. der Abbildungsebene und die Front- und Backplane), auf das hier nicht näher eingegangen wurde, sowie die eigentliche Projektion auf das View Window. Dazu dient die in Kapitel 2.5 aufgestellte Projektionsmatrix M_{per} bzw. M_{par}.

  Wenn man den daraus resultierenden Punkt noch homogenisiert, hat man in den ersten zwei Komponenten des Abbildungspunktes als Ergebnis die Koordinaten auf der Abbildungsebene vorliegen.

Kapitel 3

Kurven zur Ortsinterpolation

Im vorangegangenen Kapitel wurde eine Beschreibung von Ort und Orientierung der virtuellen Kamera mit den Größen VRP, VPN und VUP eingeführt. Dabei repräsentiert der VRP-Vektor die Position der Kamera, die Orientierung wird durch VPN und VUP festgelegt. Will man jetzt eine Translation, also eine Ortsbewegung der Kamera, durchführen, so ist eine Kurve im Raum zu beschreiben, entlang der sich die Kamera bewegt.

Diese Kurve soll durch eine geringe Zahl von Orts- oder VRP- Angaben spezifiziert werden. Sämtliche Zwischenpositionen, die Kamerastandorten während der Bewegung entsprechen, sollen dann vom Computer automatisch generiert werden.

Aus den Stützpositionen, die vom Benutzer vorgegeben werden, soll deshalb eine Beschreibung einer Kurve entwickelt werden, entlang der sich die Kamera bewegen kann. Diese Kurven sollen einer „natürlichen“, also gleichmäßigen, ruckfreien und glatten Bewegung einer existierenden Kamera möglichst nahe kommen.

3.1 Voraussetzung der Ortsinterpolation

Es ist naheliegend, die Stützpunkte einfach durch Geradenstücke zu verbinden und die Kamerabewegung entlang dieser Geraden durchzuführen. Geraden lassen sich im Zweidimensionalen als Polynome ersten Grades $y(x) = ax + b \quad a, b \in \mathbb{R}$ darstellen. Diese Interpolation entspricht jedoch keinesfalls einer natürlichen Bewegung. An den Stützpunkten würde die Kamera plötzlich ihre Richtung ändern, dazwischen wäre die Bewegung ungewöhnlich starr und geradlinig. Deshalb wird auf kompliziertere Bewegungskurven zurückgegriffen, die durch Polynome höheren Grades beschrieben werden.

Eine Beschreibung des Kurvenverlaufes explizit durch eine Funktion $y = f(x)$ ist jedoch nicht ausreichend. Bewegungsabläufe können Kreisbahnen beschreiben, und die Kurven können ihre eigene Bahn kreuzen. All dies ist mit dem explizitem Ansatz nicht zu beschreiben (Abb. 3.1).

Deshalb geht man zu einer parametrischen Beschreibung des Kurvenverlaufes durch drei Funktionen in Abhängigkeit von einem Parameter t über:

$$x = f(t) \qquad y = g(t) \qquad z = h(t)$$

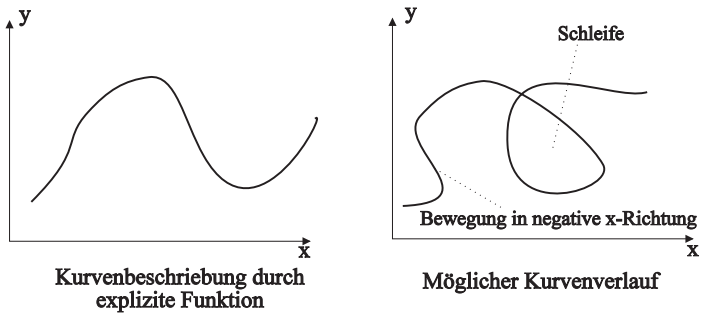


Abbildung 3.1: Explizite und parametrische Kurven im 2D

Jede dieser Kurven beschreibt in Abhängigkeit vom Parameter t den Verlauf der Kurve in einer der drei Koordinatenrichtungen.

Für die Festlegung der einzelnen Funktionen haben sich kubische Polynome als sinnvoll erwiesen, wenn die Kurven für Interpolationszwecke in der Animation verwendet werden sollen. Kubische Polynome $x(t) = at^3+bt^2+ct+d$ lassen die freie Wahl von vier Konstanten $a-d$ zu, die es Kurvensegmenten ermöglichen, durch einen Anfangs- und einen Endpunkt zu gehen und zwei weitere Freiheitsgrade zur Verfügung zu haben, die den Kurvenverlauf beeinflussen. Diese zwei Freiheitsgrade kann man nutzen, um an den Übergangsstellen zwischen den einzelnen Kurvenabschnitten Bedingungen für die Stetigkeit beim Wechsel von einem zum nächsten Kurvensegment einzuführen. Deshalb wird zuerst der Begriff der Stetigkeit bei parametrischen Kurven eingeführt und erläutert.

3.2 Mathematische und geometrische Stetigkeit

Die einzelnen Kurvensegmente sind aufgrund der Stetigkeit ihrer beschreibenden Funktionen selber stetig. Da die gesamte Kurve jedoch aus einzelnen Kurvensegmenten zusammengesetzt wird, ist die Stetigkeit der Ableitungen an den Verbindungsstellen zwischen den einzelnen Segmenten nicht automatisch gegeben.

Bei den parametrischen Kurven ist nicht mehr die Steigung der Kurve eine für den Kurvenverlauf wichtige Größe, sondern der parametrische Tangentenvektor. Dieser wird durch die erste Ableitung der die Kurve festlegenden parametrischen Funktionen bestimmt (Abb. 3.2):

$$TV = [x'(t), y'(t), z'(t)] \qquad |TV| = \sqrt{x'(t)^2 + y'(t)^2 + y'(t)^2} \tag{3.1}$$

Die Richtung des Tangentenvektors läßt sich für einen Punkt auf der Kurve aus ihrem Verlauf abschätzen, jedoch nicht dessen Länge $|TV|$.

In Abb. 3.2 ist in Punkt P2 die Richtung der beiden Tangentenvektoren TV2 und TV3 gleich, ihre Länge jedoch unterschiedlich. Dadurch folgt Q3 der Richtung des Tangentenvektors weiter als das Kurvensegment Q1.

Für die kubische, stückweise Interpolation kann man nun zwischen einer geometrischen und einer mathematischen Stetigkeit in den Verbindungsstellen unterscheiden.

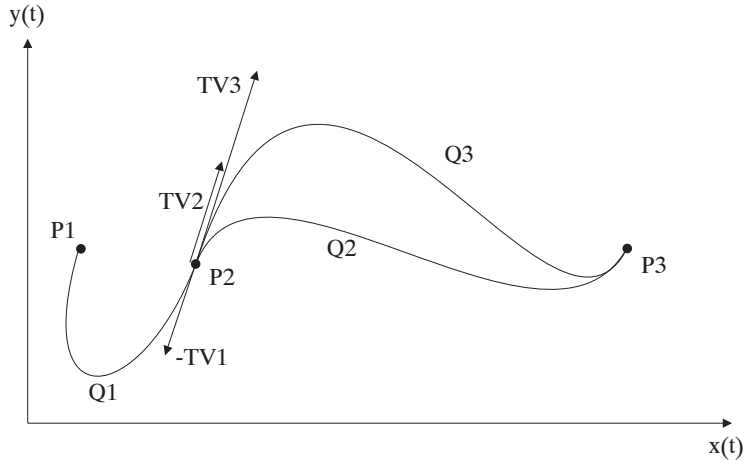


Abbildung 3.2: Parametrische Kurve mit Tangentenvektoren

Wenn zwei Kurvenstücke in einem Punkt P aneinanderstoßen, dann ist die Kurve an dieser Stelle G^0-stetig. Wenn auch die Richtung der beiden Tangentenvektoren in P übereinstimmt, dann ist die Kurve geometrisch stetig („glatt"), man sagt, sie ist G^1-stetig. Mathematisch ausgedrückt bedeutet dies, daß $TV1 = s * TV2$, $s \in \mathbb{R}$ ist. Wenn hingegen nicht nur die Richtung, sondern auch die Länge der Tangentenvektoren in P übereinstimmt, also $TV1 = TV2$, dann ist die Kurve zusätzlich mathematisch stetig, man spricht von C^1-Stetigkeit. Abb. 3.2 zeigt den Unterschied zwischen der mathematischen und der geometrischen Stetigkeit. Die Kurve Q1 Q2 ist in P2 mathematisch stetig, die Tangentenvektoren TV1 und TV2 haben die gleiche Richtung und die gleiche Länge (TV1 ist aus Sichtbarkeitsgründen mit negativem Vorzeichen gezeichnet). Hingegen ist Q1 Q3 nur geometrisch stetig; die Kurve hat zwar geometrisch keinen „Knick", jedoch sind die Längen der Tangentenvektoren TV1 und TV3 unterschiedlich.

Die geometrische Stetigkeit ist eine anschauliche Interpretation des Stetigkeitsbegriffes. Normalerweise folgt aus der mathematischen auch die geometrische Stetigkeit. Hierzu gibt es jedoch einen Ausnahmefall, wenn in P der Tangentenvektoren $TV_P = [0,\ 0,\ 0]$ ist. Hier tritt zwar mathematische Stetigkeit auf, der Kurvenverlauf kann jedoch trotzdem einen Knick machen. Dies liegt daran, daß der Tangentenvektor in P die Länge null hat, ein Nullvektor aber keine definierte Richtung aufweist.

Man kann die Stetigkeiten natürlich auch auf weitere Ableitungen ausweiten. Bei der zweiten Ableitung spricht man dann von C^2- bzw. G^2-Stetigkeit usw.

3.3 Animation und Stetigkeit

Interpretiert man den Parameter t z.B. als die Zeit, zu der sich eine Kamera am Punkte $P(t)$ befindet oder ein Gegenstand sich entlang der Kurve bewegt, so stellt der Betrag

der ersten Ableitung $|P'(t)|$ die Geschwindigkeit, der Betrag der zweiten Ableitung $|P''(t)|$ die Beschleunigung der Kamera entlang dieser Kurve dar. Für die Animation ist eine Stetigkeit auch von Geschwindigkeit und Beschleunigung erwünscht, um natürliche Bewegungsabläufe zu erzeugen, so daß man für solche Fälle auch auf G^2-Stetigkeit achten muß. In Kapitel 5 wird jedoch ein anderer Zusammenhang zwischen einer Bewegungs- und einer Zeitkurve verwendet, so daß bei der Kurvenbeschreibung nicht unbedingt die G^2-Stetigkeit gewährleistet werden muß, sondern G^2-Stetigkeit ausreicht.

3.4 Kubische Splines

Ein verbreiteter Ansatz der Interpolation mit kubischen Polynomen, die Splineinterpolation, geht davon aus, daß man ihre verbliebenen zwei Freiheitsgrade nutzt, um die C^2-Stetigkeit an den Stützpunkten zu gewährleisten.

Die Idee der Splineinterpolation liegt darin, die Punkte $x_i(t_i)$ durch eine dünne, flexible Latte (engl. Spline) zu verbinden, die durch in den Stützpunkten angreifende Kräfte fixiert wird. Dadurch erhält man glatte Interpolationskurven. Es läßt sich nachweisen, daß man aufgrund der Materialeigenschaften der Latten stückweise kubische Polynome bekommt, die an den Stützstellen C^2-stetig sind.

Für die Festlegung der Interpolationskurve (für eine der drei parametrischen Funktionen) seien $n+1$ Stützstellen t_i sowie dazugehörige Zahlenwerte x_i in einem Intervall $[a,b]$ gegeben.

Es wird nun ein Teilintervall $t \in [t_{j-1}, t_j]$ und darin die Interpolationsfunktion $S_x(t)$ für die x Variable betrachtet. Es sei $S_x(t_i) := x_i$.

Da die zweite Ableitung eines Polynomes 3. Grades eine lineare Funktion ist, gilt für das Intervall:

$$S''(t) = -S''(t_{j-1})\frac{t-t_j}{h_j} + S''(t_j)\frac{t-t_{j-1}}{h_j} \qquad \text{mit} \qquad h_j \equiv t_j - t_{j-1}$$

Durch Integration findet man:

$$S(t) = -S''(t_{j-1})\frac{(t-t_j)^3}{6h_j} + S''(t_j)\frac{(t-t_{j-1})^3}{6h_j} + c_{j1}t + c_{j0}$$

Durch die Forderung $S_x(t_i) := x_i$ lassen sich die c_{jo} und c_{j1} eliminieren.

Bildet man die zweite Ableitung dieser Gleichung und beachtet, daß $S'(t)$ und $S''(t)$ im gesamten Intervall stetig sein sollen, so erhält man das endgültige Gleichungssystem:

$$S''(t_{j-1})\frac{h_j}{6} + S''(t_j)\frac{h_j + h_{j+1}}{3} + S''(t_{j+1})\frac{h_{j+1}}{6} = \frac{x_{j+1}-x_j}{h_{j+1}} - \frac{x_j - x_{j-1}}{h_j}$$

$$j = 1, 2, \ldots, n-1$$

Zwei Bedingungen sind noch frei wählbar, z.B. kann man die Steigung am Anfang und am Ende der Gesamtkurve, also $S'(t_0)$ und $S'(t_n)$, vorgeben.

Man hat nun ein Gleichungssystem mit $n-1$ Gleichungen für die $n+1$ übrigen Unbekannten $S''(t_0) \ldots S''(t_n)$ sowie die beiden frei wählbaren Bedingungen.

Da ein solches Gleichungssystem für große n aufwendig zu lösen ist und viel Rechenzeit erfordert, ist dieses Verfahren nicht geeignet für graphische Darstellungen, bei denen es auf schnelle Neuberechnungen der Interpolationskurve ankommt. Auch wirkt sich die Veränderung eines einzigen Stützpunktes auf den gesamten Kurvenverlauf aus, dies ist bei der Verwendung in der Animation aber nicht erwünscht. Läßt man die Forderung nach C^2-Stetigkeit fallen, kann man andere Ansätze zur Kurvenbeschreibung festlegen, die mathematisch meist effizienter zu berechnen sind und im folgenden beschrieben werden.

3.5 Hermite- und Bézier-Kurven

Wie oben erwähnt, sind zur Beschreibung von kubischen Polynomen vier Koeffizienten a-d zu bestimmen. Zwei Bedingungen zur Berechnung der Koeffizienten sind der vorgegebene Anfangs- bzw. Endpunkt der Kurve. Die beiden weiteren können auf verschiedene Weise festgelegt werden. Bei den *Hermite-Kurven* wählt man die Tangentenvektoren am Anfang und Ende der Kurve, bei den *Bézier-Kurven* zwei zusätzliche Punkte, die den Endpunkt der Tangentenvektoren am Anfang bzw. am Ende der Kurve bestimmen.

Es wird wiederum für jedes Teilsegment eine unabhängige Beschreibung der Kurven gefunden, die weiteren Bedingungen werden jedoch so aufeinander abgestimmt, daß die Gesamtkurve die gewünschte G^1- oder C^1-Stetigkeit besitzt.

Die kubischen Polynome, die die Kurve in einem Abschnitt definieren, lauten:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d_x$$

$$y(t) = a_y t^3 + b_y t^2 + c_y t + d_y$$

$$z(t) = a_z t^3 + b_z t^2 + c_z t + d_z$$

Ohne Beschränkung der Allgemeinheit läßt sich t auf das Intervall $[0, 1]$ beschränken.

Man kann nun eine Vektorform der Kurvenpolynome einführen, indem man einen Zeilenvektor für die Kurve definiert:

$$Q(t) = [x(t) \qquad y(t) \qquad z(t)] =$$

$$[a_x t^3 + b_x t^2 + c_x t + d_x \qquad a_y t^3 + b_y t^2 + c_y t + d_y \qquad a_z t^3 + b_z t^2 + c_z t + d_z]$$

Setzt man

$$T = [t^3 \; t^2 \; t \; 1] \qquad \text{und} \quad C = \begin{bmatrix} a_x & a_y & a_z \\ b_x & b_y & b_z \\ c_x & c_y & c_z \\ d_x & d_y & d_z \end{bmatrix}$$

dann kann man $Q(t)$ in $Q(t) = [x(t) \; y(t) \; z(t)] = T \cdot C$ umschreiben.

Die parametrische Ableitung von $Q(t)$ ergibt sich zu:

$$Q'(t) = \begin{bmatrix} 3a_x t^2 + 2b_x t + c_x & 3a_y t^2 + 2b_y t + c_y & 3a_z t^2 + 2b_z t + c_z \end{bmatrix}$$

Da sich nun die Bedingungen für die Berechnung der Kurve unterschiedlich wählen lassen, kann man die obige Matrix C durch $C = M \cdot G$ ausdrücken, wobei M eine 4×4 Basismatrix und G ein Spaltenvektor mit den vier Elementen der geometrischen Bedingungen für die drei Koordinatenachsen x, y, z, die für eine bestimmte Kurvenart gewählt worden sind. M und G sind für ein Kurvenstück konstant, so daß durch $Q(t) = T \cdot M \cdot G$ die drei Polynome von t dargestellt werden können:

$$Q(t) = [x(t)\ y(t)\ z(t)] = [t^3\ t^2\ t\ 1] \cdot \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \\ m_{41} & m_{42} & m_{43} & m_{44} \end{bmatrix} \cdot \begin{bmatrix} G_1 \\ G_2 \\ G_3 \\ G_4 \end{bmatrix}$$

Je nach Vorgabe der Geometriebedingungen folgen unterschiedliche Interpolationskurven; für jede Wahl der Geometriebedingungen muß einmal M bestimmt werden.

3.5.1 Hermite-Kurven

Bei den Hermite-Kurven hat man als geometrische Bedingungen Anfangs- und Endpunkt P_1 und P_2 sowie die Tangentenvektoren R_1 und R_2 an Anfangs- und Endpunkt gegeben. Um die Basismatrix M_H für die Hermite-Darstellung zu finden, geht man von vier Gleichungen für die geometrischen Bedingungen aus und löst diese nach den vier Unbekannten der Polynomdarstellung von $x(t)$ auf.

Es ist:

$$G_{H_x} = \begin{bmatrix} P_{1_x} \\ P_{2_x} \\ R_{1_x} \\ R_{2_x} \end{bmatrix} \qquad G_{H_y} = \begin{bmatrix} P_{1_y} \\ P_{2_y} \\ R_{1_y} \\ R_{2_y} \end{bmatrix} \qquad G_{H_z} = \begin{bmatrix} P_{1_z} \\ P_{2_z} \\ R_{1_z} \\ R_{2_z} \end{bmatrix}$$

Ausgehend von der Definitionsgleichung:

$$x(t) = a_x t^3 + b_x t^2 + c_x t + d = T \cdot M_H \cdot G_{H_x} = [t^3\ t^2\ t\ 1] \cdot M_H \cdot G_{H_x}$$

sowie deren Ableitung erhält man für Anfangs- und Endpunkte:

$$x(0) = P_{1_x} = [0\ 0\ 0\ 1] \cdot M_H \cdot G_{H_x}$$

$$x(1) = P_{2_x} = [1\ 1\ 1\ 1] \cdot M_H \cdot G_{H_x}$$

$$x'(0) = R_{1_x} = [0\ 0\ 1\ 0] \cdot M_H \cdot G_{H_x}$$

$$x'(1) = R_{2_x} = [3\ 2\ 1\ 0] \cdot M_H \cdot G_{H_x}$$

und damit in Matrixform :

$$\begin{bmatrix} P_1 \\ P_2 \\ R_1 \\ R_2 \end{bmatrix}_x = G_{H_x} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix} \cdot M_H \cdot G_{H_x}$$

Durch Berechnung der Inversen dieser Matrix läßt sich M_H bestimmen:

$$M_H = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 0 \\ 3 & 2 & 1 & 0 \end{bmatrix}^{-1} = \begin{bmatrix} 2 & -2 & 1 & 1 \\ -3 & 3 & -2 & -1 \\ 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Damit kann die Gleichung $x(t) = T \cdot M_H \cdot G_{H_x}$ für beliebige t berechnet werden. Ebenso lassen sich $y(t) = T \cdot M_H \cdot G_{H_y}$ und $z(t) = T \cdot M_H \cdot G_{H_z}$ mit identischem M_H bestimmen. Man erhält also :

$$Q(t) = [x(t)\ y(t)\ z(t)] = T \cdot M_H \cdot G_H$$

mit G_H wie oben für G_{H_x} definiert.

Multipliziert man diese Gleichung aus, so ergibt sich:

$$Q(t) = T \cdot M_H \cdot G_H = (2t^3 - 3t^2 + 1)P_1 + (-2t^3 + 3t^2)P_2 + (t^3 - 2t^2 + t)R_1 + (t^3 - t^2)R_2$$

Wenn man nun zwei Kurvenstücke mit den Endpunkten P_1 P_2 bzw. P_2 P_3 mit G^1-Stetigkeit in P_2 aneinandersetzen will, so gilt folgende Bedingung für die Wahl der Tangentenrichtungen:

$$G_{H1} = \begin{bmatrix} P_1 \\ P_2 \\ R_1 \\ R_2 \end{bmatrix} \qquad G_{H2} = \begin{bmatrix} P_2 \\ P_3 \\ k\ R_2 \\ R_3 \end{bmatrix}$$

mit $k > 0$ für G^1-Stetigkeit und $k = 1$ für C^1-Stetigkeit.

3.5.2 Bézier-Kurven

Bei den Bézier-Kurven sind neben den Anfangs- und Endpunkten P_1 und P_4 zwei weitere Punkte P_2 und P_3 gegeben, die nicht auf der Kurve liegen. Diese kontrollieren die Tangentenvektoren am Anfang und am Ende der Kurve.

Der Geometrievektor für Bézier-Kurven lautet:

$$G_B = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix}$$

Es läßt sich leicht eine Darstellung der Tangentenvektoren R_1 und R_2 (mit einem geeigneten Skalierungsfaktor) aus den Kontrollpunkten finden:

$$R_1 = Q'(0) = 3\ (P_2 - P_1)$$

$$R_2 = Q'(1) = 3\ (P_4 - P_3)$$

Damit kann man G_B in G_H, den Geometrievektor der Hermite-Kurven, umschreiben:

$$G_H = \begin{bmatrix} P_1 \\ P_2 \\ R_1 \\ R_2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ -3 & 3 & 0 & 0 \\ 0 & 0 & -3 & 3 \end{bmatrix} \cdot \begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \end{bmatrix} = M_{HB} \cdot G_B$$

Um nun die Basismatrix M_B zu finden, geht man von der Definitionsgleichung von $Q(T)$ für Hermite-Kurven aus :

$$Q(t) = T \cdot M_H \cdot G_H = T \cdot M_H \cdot M_{HB} \cdot G_B := T \cdot M_B \cdot G_B$$

Damit erhält man für M_B:

$$M_B = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Setzt man zwei Bézier-Kurven $P_1 - P_4$ sowie $P_4 - P_7$ aneinander, so lautet die Bedingung für die Stetigkeit:

$$P_4 - P_3 = k(P_5 - P_4)$$

mit $k > 0$ für G^1-Stetigkeit und $k = 1$ für C^1-Stetigkeit.

Bei den Bézier-Kurven ist eine Erweiterung von vier auf $n \geq 4$ Stützpunkte möglich; die allgemeine Formulierung lautet:

$$Q(t) = \sum_{i=0}^{n} p_i\ B_{i,n}(t) \qquad t \in [0,1]$$

mit

$$B_{i,n}(t) = \binom{n}{i}\ t^i\ (1-t)^{n-i}$$

Mit einer größeren Anzahl von Stützpunkten hat man mehr Freiheitsgrade beim Bestimmen der Interpolation. Dadurch kann man z.B. die Stetigkeit an Stützstellen über die C^2-Stetigkeit hinaus erweitern, jedoch wird auch der Rechenaufwand entsprechend höher.

3.6 Sonderformen

Möchte man durch eine vorgegebene Anzahl von Punkten eine Kurve legen, hat jedoch keine weiteren Angaben, wie sie zur Darstellung von Bézier- und Hermite-Kurven benötigt werden (Tangentenvektoren etc.), dann gibt es spezielle Formen dieser Kurven, die die Interpolation ermöglichen.

3.6.1 Catmull-Rom Splines

Eine einfache Methode sind die sogenannten Catmull-Rom Splines. Hier wird die Richtung der Tangentenvektoren in den Interpolationspunkten einfach durch eine Parallele zu der Geraden durch die beiden Nachbarpunkte bestimmt (vgl. Abb. 3.3).

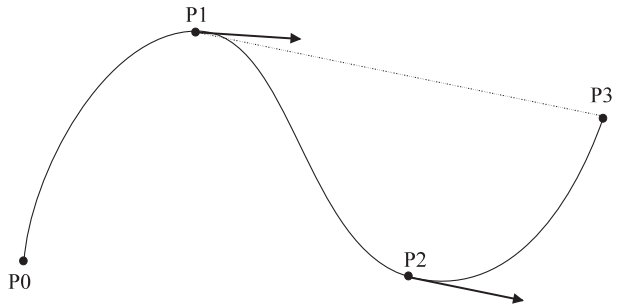


Abbildung 3.3: Catmull-Rom Spline

Hat man $n+1$ Punkte $P_0 \cdots P_n$ vorliegen, so kann man den Tangentenvektor im Punkt P_i für $i = 1, \cdots, n-1$ bestimmen:

$$R_i = \frac{1}{2}(P_{i+1} - P_{i-1})$$

Die Tangentenvektoren R_0 und R_n können frei gewählt werden.

Die mathematische Darstellung der Interpolationsgleichung mit der Basismatrix M_{CR} lautet:

$$Q_n(t) = T \cdot M_{CR} \cdot G_{CR} = T \cdot \frac{1}{2} \cdot \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \cdot \begin{bmatrix} P_{i-1} \\ P_i \\ P_{i+1} \\ P_{i+2} \end{bmatrix}$$

für die Kurve zwischen P_i und P_{i+1}.

3.6.2 Formeln von Kochanek und Bartels

Bei den Catmull-Rom Splines kann man, außer durch die Wahl der Stützpunkte, keinen Einfluß auf den Verlauf der Kurve nehmen. Dies entspricht jedoch nicht immer den Wünschen für einen Kurvenverlauf in der Animation. Teilweise sind bauchigere, rundere

Kurven erwünscht, manchmal soll der Kurvenverlauf jedoch auch nahe an der linearen Interpolationskurve liegen und es sind abrupte Richtungsänderungen durchaus erwünscht.

Doris H.U.Kochanek und Richard H. Bartels haben eine Klasse von Kurven definiert, die eine weitreichende Kontrolle des Kurvenverlauf ermöglichen. [KOC84]. Durch die Einführung dreier Parameter **Tension** (Spannung, Parameter a), **Bias** (Vorbeeinflussung, Parameter b) sowie **Continuity** (Stetigkeit, Parameter c) kann man - global oder auch lokal - Einfluß auf den Kurvenverlauf nehmen. Als Spezialfall sind in der folgenden Darstellung die Catmull-Rom Splines für $a = b = c = 0$ enthalten.

Da der Einfluß der Parameter mit Worten schwierig zu beschreiben ist, erfolgt im Anschluß eine Erläuterung der Parameter anhand von Beispielen.

Die Darstellung greift auf Bézier-Kurven zurück. Hat man $n+1$ Punkte $P_0 \cdots P_n$ vorliegen, so will man wiederum die Tangentenvektoren an den Punkten P_i für $i = 1, \cdots, n-1$ bestimmen. Man definiert die Tangentenvektoren am Beginn und am Ende eines Kurvensegmentes zwischen den Punkten P_i und P_{i+1} wie folgt:

$$\begin{aligned} R_1 = &\frac{(1-a_i)(1+c_i)(1+b_i)}{2} \quad (P_i - P_{i-1}) + \\ &\frac{(1-a_i)(1-c_i)(1-b_i)}{2} \quad (P_{i+1} - P_i) \end{aligned}$$

$$\begin{aligned} R_2 = &\frac{(1-a_{i+1})(1-c_{i+1})(1+b_{i+1})}{2} \quad (P_{i+1} - P_i) + \\ &\frac{(1-a_{i+1})(1+c_{i+1})(1-b_{i+1})}{2} \quad (P_{i+2} - P_{i+1}) \end{aligned}$$

mit $a_i\ b_i\ c_i$, den Parametern am Punkt P_i, sowie $a_{i+1}\ b_{i+1}\ c_{i+1}$, den Parametern am Punkt P_{i+1}.
Setzt man:

$$\begin{aligned} k &= \frac{(1-a_i)(1+c_i)(1+b_i)}{2} \\ l &= \frac{(1-a_i)(1-c_i)(1-b_i)}{2} \\ m &= \frac{(1-a_{i+1})(1-c_{i+1})(1+b_{i+1})}{2} \\ n &= \frac{(1-a_{i+1})(1+c_{i+1})(1-b_{i+1})}{2} \end{aligned}$$

so läßt sich auch die Basismatrix M_{KB} bestimmen zu:

$$M_{KB} = \begin{bmatrix} -k & 2+k-l-m & -2+l+m-n & n \\ 2k & -3-2k+2l+m & 3-2l-m+n & -n \\ -k & k-l & l & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Damit hat man auch eine Darstellung für $Q(t)$ mit dem G_{CR} der Catmull-Rom Splines:

$$Q(t) = T \cdot M_{KB} \cdot G_{CR}$$

3.7 Parameteränderungen bei Kochanek-Bartels Kurven

Eine Untersuchung zeigt, welchen Einfluß die Parameter Tension a, Bias b und Continuity c auf den Verlauf der Interpolationskurve haben:

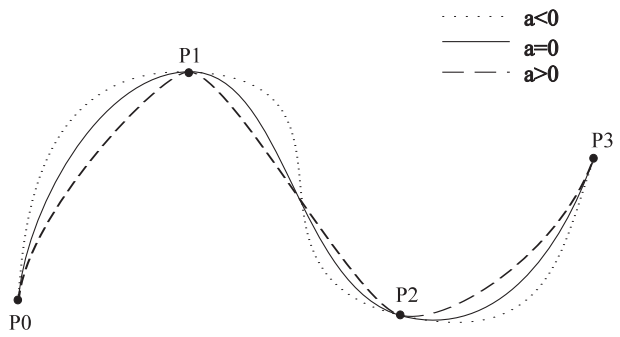


Abbildung 3.4: Variation des Tensionparameters

Mit dem **Tensionparameter** a (Abb. 3.4) kann man festlegen, wie stark sich die Kurve im Stützpunkt krümmen soll. Der Tensionparameter beeinflußt die Länge des Tangentenvektors im Stützpunkt, dadurch kann man den Verlauf der Kurve, wie in Kapitel 3.1, Abb. 3.2 dargestellt, verändern. Die Kurve bleibt immer mathematisch stetig; für $a = 1$ tritt jedoch der in Kapitel 3.1 erwähnte Sonderfall auf, daß die mathematische Stetigkeit nicht die geometrische Stetigkeit beinhaltet.

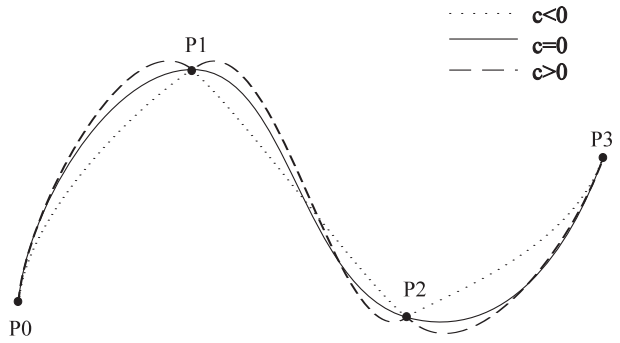


Abbildung 3.5: Variation des Continuityparameters

Man will bei der Filmgenerierung zwar normalerweise stetige Kurven erzielen, manchmal ist es jedoch auch sinnvoll, Unstetigkeit der Tangentenvektoren zuzulassen. Dies kann man mit dem Parameter c, **Continuity** (Abb. 3.5), erreichen. Er beeinflußt die Stetigkeit zwischen einlaufender und auslaufender Tangente.

$C < 0$ erzeugt in der Abb. 3.5 eine konvexe Ecke, die Tangentenvektoren laufen stark auf den jeweils vorherigen Punkt beim Eintreffen bzw. auf den nachfolgenden Punkt beim

Verlassen des Stützpunktes zu. $C > 0$ erzeugt eine konkave Ecke, die Tangentenvektoren verhalten sich genau umgekehrt. Die Kurve ist nur für $c = 0$ mathematisch und geometrisch stetig. Damit sind abrupte Richtungsänderungen möglich.

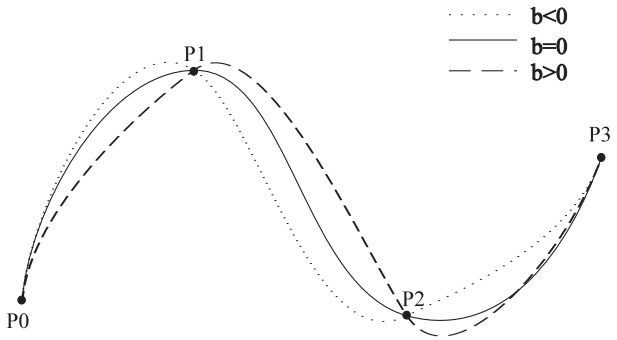


Abbildung 3.6: Variation des Biasparameters

Der **Biasparameter** b (Abb. 3.6) steuert, ob die einlaufende oder die auslaufende Tangente stärkeren Einfluß auf den Verlauf der Kurve haben soll. Er verändert damit die Richtung des Kurvenverlaufes in dem Moment, in dem die Kurve durch den Stützpunkt läuft. Im Fall $b > 0$ wird die Richtung der Kurve stärker durch die Richtung der Geraden zwischen Stützpunkt und ihrem Vorgänger bestimmt, der Kurvenverlauf schießt quasi über den Stützpunkt hinaus. Dies wird in der Animation oft als „overshot“ bezeichnet. Im Fall $b < 0$ wird die Richtung dagegen stärker durch die Richtung der Geraden zwischen Stützpunkt und ihrem Nachfolger bestimmt, der Kurvenverlauf hat also bereits schon beim Erreichen des Stützpunktes seine Richtung fast vollständig geändert.

Kapitel 4

Orientierungsdarstellung und -interpolation

Zur Definition der Lage einer Kamera ist neben ihrer Position die Ausrichtung der Kamera, ihre Orientierung, notwendig. Im Kapitel 2, 3D-Darstellung, war zur Berechnung einer Projektion neben der Position der Kamera durch den VRP noch ihre Lage und Ausrichtung anzugeben. Dies geschieht durch Angabe des View Plane Normal Vektors VPN und des View Up Vektors VUP. Mit diesen beiden Werten wurde ein neues Koordinatensystem, das VRC-Koordinatensystem, definiert.

Allgemein kann man eine Orientierungsbeschreibung als Ausrichtung des Koordinatensystems des Objektes, hier also der Kamera, in bezug auf ein anderes, meist als absolutes oder Weltkoordinatensystem bezeichnet, betrachten. Man muß also eine Beschreibung finden, mit deren Hilfe man das Weltkoordinatensystem in das Kamerakoordinatensystem überführen kann. Hierbei spricht man auch von Koordinatentransformation.

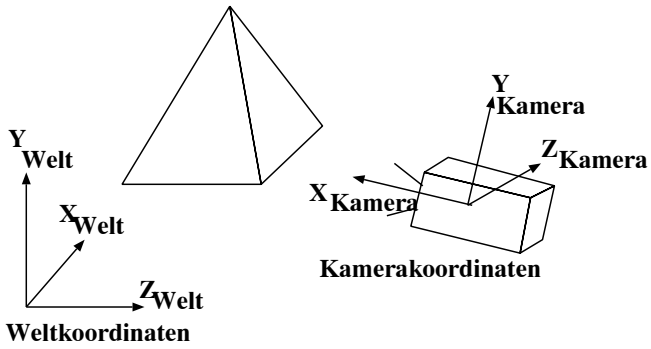


Abbildung 4.1: Koordinatensysteme und Kameramodell

Wie bei der Ortsinterpolation, bei der man auf der Bahn zwischen zwei vorgegebenen Stützpunkten Zwischenpositionen finden mußte, ist es für die Orientierungsinterpolation notwendig, Zwischenorientierungen zwischen zwei vorgegebenen Koordinatensystemen bzw. Orientierungen zu generieren.

Um die Orientierungsdarstellungen mathematisch gut handhabbar zu machen, sind Beschreibungen entwickelt worden, die für unterschiedliche Anwendungszwecke verschieden gut geeignet sind. Hier werden zwei dieser Beschreibungen, nämlich die Beschreibung mit Eulerwinkeln und mit Hilfe von Quaternionen näher erläutert. Diese beiden Beschreibungen finden in dem entwickelten Programm VIDES Verwendung. Neben diesen Beschreibungen gibt es noch weitere Darstellungen wie z.B. mit Hilfe von Rotationsmatrizen (wie in Kapitel 2 verwendet) oder mit Kardanwinkeln, die z.B. in der Robotertechnik Anwendung finden.

4.1 Kameraorientierungen in der Animation

Ausgehend von der Beschreibung der Kameraorientierung durch VPN und VUP wird zuerst kurz erläutert, welche Orientierungsmöglichkeiten bei der Kameraführung vorkommen können (Abb. 4.2):

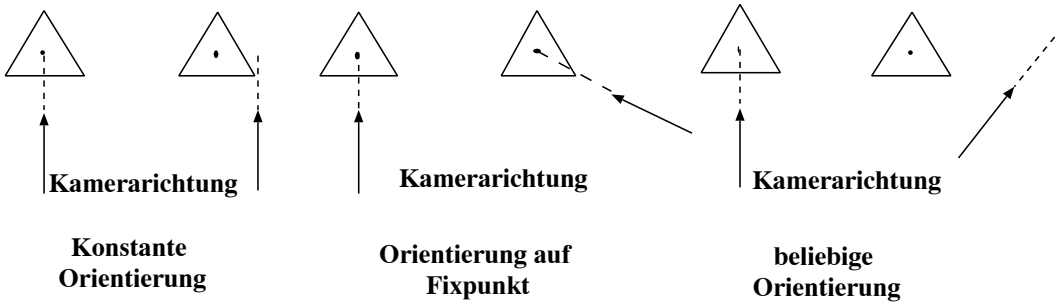


Abbildung 4.2: Orientierung der Kamera

- Wird die Kamera von einem Ort zu einem anderen bewegt (Translation des VRP), so sind zwei spezielle Orientierungen denkbar:

  Einerseits kann die Richtung der Kamera unverändert bleiben. Bei der Verschiebung wird sich also nicht die Orientierung ändern, aber der Zielpunkt, auf den die Kamera ausgerichtet ist. Das bedeutet, daß VPN und VUP bei der Bewegung nicht verändert werden.

  Andererseits kann die Kamera auf einen Fixpunkt ausgerichtet bleiben, das heißt, daß sie bei einer Bewegung weiter in Richtung des gleichen Punktes ausgerichtet wird. Dieser Zielpunkt ändert sich also nicht, während dagegen die Orientierung der Kamera verändert werden muß. Der VPN läßt sich damit als Differenzvektor zwischen VRP und einer festen Position im Raum bilden. Der VPN wird unverändert senkrecht auf diesem Vektor festgelegt, wenn keine exlizite Rotation angegeben ist.

  Für diese beiden Fälle ist keine aufwendige Orientierungsinterpolation notwendig, die Orientierung der Kamera ergibt sich aus ihrer Position.

- Da es aber auch weiterhin möglich sein soll, die Orientierung der Kamera, also des Kamerakoordinatensystems, völlig unabhängig von dessen Position zu verändern,

muß eine Möglichkeit der freien Orientierungsinterpolation gefunden werden. Häufig sind auch bei fester Position des Kamerakoordinatensystems mehrere Orientierungen möglich und sinnvoll. Die Zahl der vorgegebenen Stützpositionen muß also nicht identisch mit der Zahl der Orientierungen sein. VPN und VUP müssen beliebig festgelegt werden können, und es muß unabhängig von einer Bewegung eine Interpolation zwischen diesen Werten gefunden werden.

4.2 Koordinatensysteme

Bevor man sich auf eine Orientierungsbeschreibung festlegt, muß man zunächst die Wahl eines Koordinatensystems vornehmen, in dem diese definiert wird. In Kapitel 2 ist das in der Graphik verwendete rechtshändige Koordinatensystem vorgestellt worden, bei dem die y-Achse senkrecht nach oben zeigt. Dieses soll auch weiterhin verwendet werden.

Da bei der Definition von Eulerwinkeln meist auf die Festlegung dieser Winkel in der Luftfahrt zurückgegriffen wird, soll den Graphikkoordinaten die Festlegung des Koordinatensystems in der Luftfahrt kurz gegenüberstellt werden:

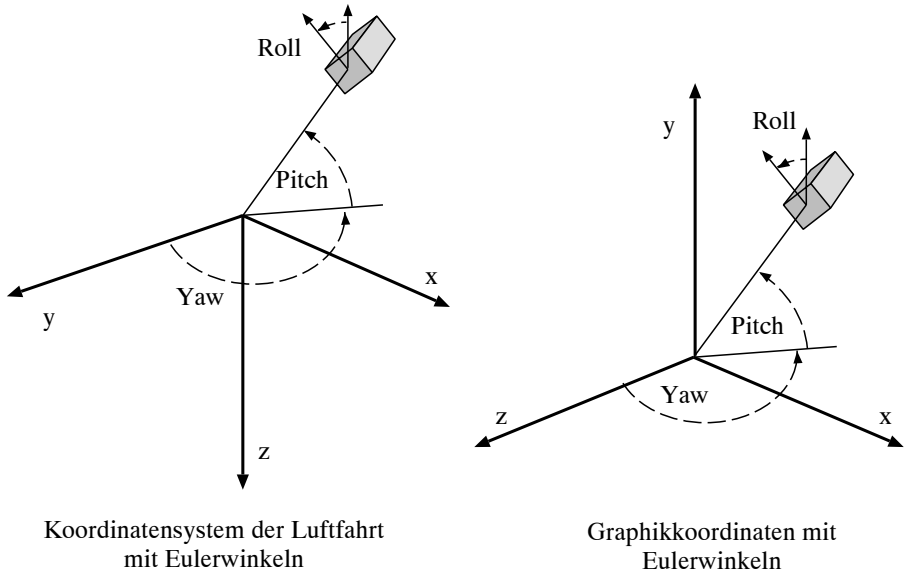


Abbildung 4.3: Eulerwinkel in verschiedenen Festlegungen

Beide Koordinatensysteme sind rechtshändige Koordinatensysteme. Beim Koordinatensystem der Luftfahrt ist die senkrechte Achse jedoch die z-Achse und nicht die y-Achse, diese ist ferner nach unten und nicht nach oben ausgerichtet. Diese beiden Koordinatensysteme sind in Abb.4.3 (mit der Festlegung der Eulerwinkel) dargestellt.

4.3 Eulerwinkel

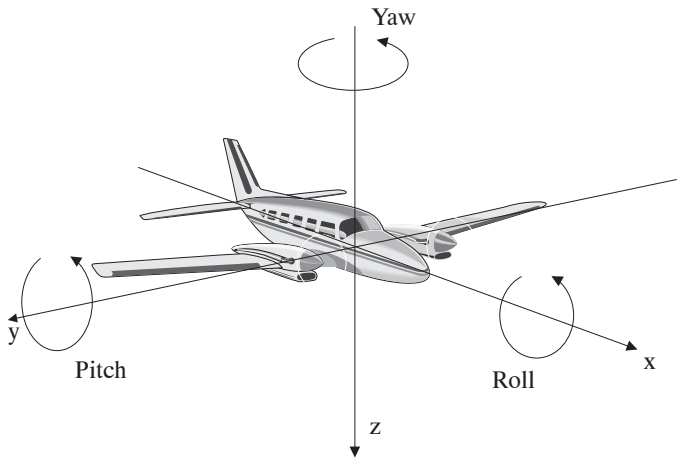


Abbildung 4.4: Eulerwinkel in der Luftfahrt

Bei der Beschreibung einer Orientierung mit Eulerwinkeln werden drei Winkel angegeben. Führt man drei Drehungen mit diesen Winkeln in der richtigen Reihenfolge aus, wird das Weltkoordinatensystem in das Kamerakoordinatensystem überführt. Diese Winkel sind in ihrer Rotationsreihenfolge mit den Drehachsen wie folgt definiert:

- YAW- oder Azimuthwinkel, dieser rotiert in der Luftfahrt um die z-Achse: Mit dem YAW-winkel wird die Richtung des Flugzeuges, also die Orientierung seiner Nase in eine (Himmels-)Richtung angegeben.
- PITCH- oder Nickwinkel, dieser rotiert um die (momentane) y-Achse: Er gibt an, ob das Flugzeug gerade, nach oben oder nach unten ausgerichtet ist, also ob es steigt oder sinkt.
- ROLL- oder Hängewinkel, der die Drehung um die (momentane) x-Achse festlegt: Er beschreibt eine Rotation um die Längsachse des Flugzeuges (und damit bei der Kamera die Lage des Bildhorizontes).

In Abb.4.3 ist bereits die Festlegung der Eulerwinkel, wie sie in der Luftfahrt verwendet werden, sowie die Übertragung dieser Winkel in das im Graphikbereich verwendete Koordinatensystem dargestellt worden.

Wie oben erwähnt ist für die Animation die Generierung von Zwischenpositionen zwischen zwei vorgegebenen Orientierungen nötig . Hat man die beiden Orientierungen durch Eulerwinkel gegeben, so liegt es auf den ersten Blick nahe, Zwischenorientierungen zu generieren, indem man einfach zwischen den jeweiligen Winkeln (YAW, PITCH und ROLL) interpoliert. Da die Kamera jedoch in diesem Fall unabhängig, aber gleichzeitig um die drei Koordinatenachsen rotiert, kann sich ihre Ausrichtung rasch und unerwartet verändern. Man stößt dabei auf eine Problematik, die in Abb. 4.5 erläutert wird. In diesem Bild wird davon ausgegangen, daß man eine Rotation zwischen zwei Orientierungen als Bewegung auf der Einheitskugel im vierdimensionalen Raum betrachten kann, da man eine

Orientierung durch Angabe von vier skalaren Werten festlegen kann. Deshalb zieht man als Modellbeispiel die Bewegung auf der Einheitskugel im dreidimensionalen Raum heran, da diese anschaulich noch klar darstellbar ist. Damit wird in Abb.?? nur eine Rotation um zwei Achsen dargestellt, der ROLL-Winkel bleibt unbeachtet. Die Orientierungsinterpolation soll entlang eines Teiles des Großkreises zwischen q_0 und q_1 erfolgen. Bei einer Rotation mit Eulerwinkeln kann sie dagegen eine taumelnde, ungleichmäßige Bewegung vollziehen. Diese Bewegung entspricht in keiner Weise der natürlichen Vorstellung von einer Orientierungsinterpolation.

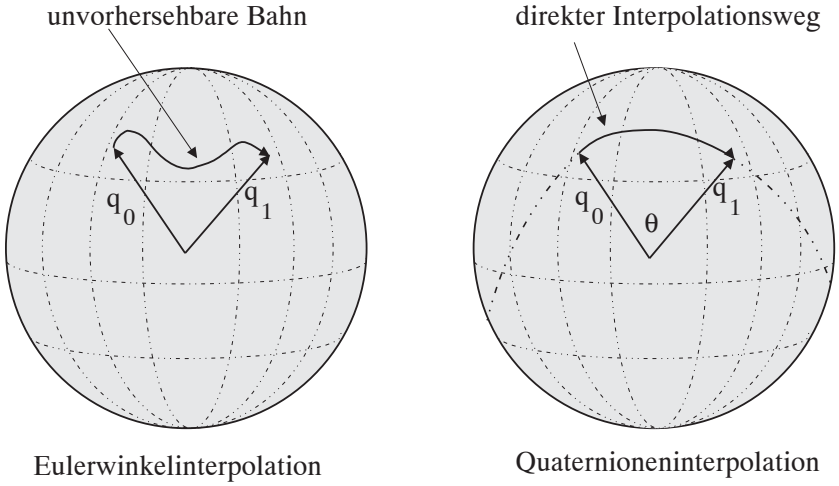


Abbildung 4.5: Orientierungsinterpolation mit Eulerwinkeln und Quaternionen

Die Darstellung einer Orientierung mit Eulerwinkeln erweist sich so für die Interpolation als ungeeignet, während sie zur anschaulichen Beschreibung, z.B. in einer Benutzerschnittstelle zur direkten Orientierungsangabe zur Berechnung einer Projektion, sinnvoll zu gebrauchen ist.

Für eine gleichmäßige Interpolation wäre es wünschenswert, die Überführung der ersten in die zweite Orientierung durch eine einzige Rotation entlang einer entsprechend zu definierenden Achse durchführen zu können. Dies ist einfach möglich, wenn man den Vektor der Drehachse $\vec{r}$ und den Drehwinkel ϕ kennt, man könnte hierzu z.B. auf eine Beschreibung durch Rotationsmatrizen zurückgreifen, wie sie im Kapitel 2 definiert worden sind. Liegt die Orientierungsbeschreibung jedoch durch Angabe von Eulerwinkeln vor, so ist die Bestimmung von $\vec{r}$ und ϕ recht aufwendig, sie führt im allgemeinen auf die Lösung eines Eigenwertproblems [ZUN93, Seite 207].

4.4 Quaternionen

Bei der Beschreibung von Orientierungen mit Hilfe von Quaternionen ist die Interpolation deutlich einfacher zu handhaben als mit Hilfe von Eulerwinkeln. Deshalb soll zuerst eine Definition des Quaternionenbegriffes und eine Einführung in den Rechenformalismus und den Umgang mit Quaternionen erfolgen. Es ist recht kompliziert, ein genaues Verständnis

aller mathematischen Grundlagen der Quaternionen zu bekommen; dies ist jedoch auch nicht notwendig, um sie in der Animation anzuwenden. Bereits Ken Shoemake, der die Verwendung von Quaternionen in der Computergraphik populär gemacht hat, schrieb in seiner Einführung: „Leser, die mit Spline-Kurven und ihrer Benutzung in der Computeranimation gut vertraut sind, sollten wenig Probleme haben", Quaternionen und die Herleitung der folgenden Methoden zu verstehen, „ obwohl sogar diese über Quaternionen ein wenig stolpern können" [SHO85, Seite 245].

4.4.1 Mathematische Grundlagen

Quaternionen wurden 1843 von Sir William Rowan Hamilton als ein Hilfsmittel entwickelt, mit deren Hilfe ein Vektor $\vec{a}$ der Dimension 3 auf mathematisch einfache Weise in einen zweiten Vektor $\vec{b}$ überführt werden kann. Hierbei wird neben der Ausrichtung des Vektors im Raum auch noch seine Länge angepaßt. Mathematisch gesehen stellen Quaternionen eine Erweiterung der komplexen Zahlen in den vierdimensionalen Raum dar. Sie können als Vektoren im vierdimensionalen Raum betrachtet werden. Sie bilden einen linearen Vektorraum der Dimension vier über dem Körper der reellen Zahlen.

Definition: Eine Quaternion $\hat{q}$ besteht aus einem skalaren Teil q_0 und einem vektoriellen Anteil $\vec{q}$:

$$\hat{q} = q_0 + q_1 i + q_2 j + q_3 k = [q_0, \vec{q}] = [S, \vec{w}]$$

mit:

$$i^2 = j^2 = k^2 = ijk = -1$$

$$ij = k = -ji \qquad jk = i = -kj \qquad ki = j = -ik$$

Hier erkennt man die Analogie zu den komplexen Zahlen mit $\mathbf{x} = x_0 + x_1 i$ mit $i^2 = -1$

Es gelten folgende Rechenregeln:

- Addition zweier Quaternionen:

  $$\hat{q}_1 + \hat{q}_2 = [S_1, \vec{w}_1] + [S_2, \vec{w}_2] = [S_1 + S_2, \vec{w}_1 + \vec{w}_2]$$

  mit dem neutralen Element $\hat{0} = [0, \vec{0}]$ und dem inversen Element $-\hat{q} = [-S, -\vec{w}]$.

- Multiplikation mit einem Skalar k:

  $$k * \hat{q} = [k\ S, k * \vec{w}]$$

- Quaternionen-Multiplikation:

  $$\hat{q}_1 \hat{q}_2 = [S_1, \vec{w}_1][S_2, \vec{w}_2] =$$

  $$[S_1 S_2 - \vec{w}_1 \cdot \vec{w}_2, S_1 \vec{w}_1 + S_2 \vec{w}_2 + \vec{w}_1 \times \vec{w}_2]$$

  Dabei entspricht $\vec{w}_1 \cdot \vec{w}_2$ dem herkömmlichen Skalarprodukt zweier Vektoren und $\vec{w}_1 \times \vec{w}_2$ dem Vektorprodukt dieser Vektoren.
  Das neutrale Element der Quaternionenmultiplikation ist $\hat{1} = [1, \vec{0}]$.

- Skalarprodukt zweier Quaternionen:

$$\hat{q}_1 \cdot \hat{q}_2 = \sqrt{(S_1 S_2)^2 + (\vec{w}_1 \cdot \vec{w}_2)^2}$$

- konjugierte Quaternion:

  Zu jeder Quaternion $\hat{q}$ läßt sich eine konjugierte Quaternion $\overline{\hat{q}}$ definieren:

$$\overline{\hat{q}} = [S, -\vec{w}]$$

- Betrag und inverses Element:

  Mit dem Betrag einer Quaternion:

$$|\hat{q}| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2}$$

  läßt sich das inverse Element bei einer Multiplikation bestimmen zu :

$$\hat{q}^{-1} = \frac{\overline{\hat{q}}}{|\hat{q}|} = [\frac{S}{|\hat{q}|}, -\frac{\vec{w}}{|\hat{q}|}]$$

- Kommutativität, Distributivität und Assoziativität:

  Während die Quaternionenaddition kommutativ ist, ist die Multiplikation zweier Quaternionen nicht kommutativ. Dies liegt daran, daß das Vektorprodukt $\vec{w}_1 \times \vec{w}_2$ nicht kommutativ ist:

$$\hat{q}_1 + \hat{q}_2 = \hat{q}_2 + \hat{q}_1$$

$$\hat{q}_1 \hat{q}_2 \neq \hat{q}_2 \hat{q}_1$$

  Die Multiplikation ist distributiv gegenüber der Addition:

$$\hat{q}_1(\hat{q}_2 + \hat{q}_3) = \hat{q}_1 \hat{q}_2 + \hat{q}_1 \hat{q}_3$$

  Ebenfalls sind beide Operationen assoziativ:

$$\hat{q}_1 + (\hat{q}_2 + \hat{q}_3) = (\hat{q}_1 + \hat{q}_2) + \hat{q}_3$$

$$\hat{q}_1(\hat{q}_2 \hat{q}_3) = (\hat{q}_1 \hat{q}_2)\hat{q}_3$$

- Einheitsquaternionen:

  Eine Untergruppe der Quaternionen bilden die Einheitsquaternionen, deren Betrag $|\hat{q}| = 1$ beträgt. Für Einheitsquaternionen gelten einige besondere Regeln:

$$\hat{q}^{-1} = \overline{\hat{q}} \quad \text{da} \quad |\hat{q}| = 1$$

$$\text{und damit} \quad \overline{\hat{q}_1 \hat{q}_2} = \overline{\hat{q}_2}\,\overline{\hat{q}_1}$$

Jede Einheitsquaternion kann in folgender Form geschrieben werden, die sich bei der Verwendung von Quaternionen im Animationsbereich als günstig erweist, da man damit eine Rotation um den Winkel θ direkt darstellen kann:

$$\hat{q} = (cos(\frac{\theta}{2}), sin(\frac{\theta}{2}) * \vec{r})$$

Dabei ist $\vec{r}$ ein Einheitsvektor, seine Länge beträgt also 1. Diese Form entspricht der Polarform von komplexen Zahlen, man kann $\hat{q}$ also schreiben als:

$$\hat{q} = e^{\vec{r}\frac{\theta}{2}} = (cos(\frac{\theta}{2}), sin(\frac{\theta}{2}) * \vec{r})$$

4.4.2 Rotation mit Quaternionen

Man kann nun die Quaternionen verwenden, um eine Orientierungsbeschreibung zu finden. Betrachtet man eine Einheitsquaternion, so hat man in der Polarform alle Angaben vorliegen, um ein Koordinatensystem in ein zweites zu überführen. Mit $\hat{q} = e^{\vec{r}\frac{\theta}{2}}$ hat man durch $\vec{r}$ die Rotationsachse gegeben, eine Rotation um den Winkel θ bezüglich dieser Achse bewirkt genau unsere Koordinatentransformation.

Will man also eine Rotation eines beliebigen Ortsvektors $\vec{v}$ (bzw. des Punktes, der durch diesen Vektor beschrieben wird) durchführen, muß man dazu erst den Vektor $\vec{v}$ in eine Quaternion $\hat{v}$ überführen, indem man $\hat{v} = [0, \vec{v}]$ setzt.

Wählt man für die Rotation nun die Quaternion $\hat{q} = e^{\vec{r}\frac{\theta}{2}} = (cos(\frac{\theta}{2}), sin(\frac{\theta}{2}) * \vec{r})$, so läßt sich eine Rotation wie folgt beschreiben:

$$\hat{v}' = \hat{q}\hat{v}\hat{q}^{-1}$$

Um die Eindeutigkeit der Rotation zu gewährleisten, muß man $|\theta| < \pi$ setzen. Andernfalls würde die Rotation nicht um den Winkel $\theta > \pi$, sondern um den Winkel $\theta' = 2\pi - \theta$ ausgeführt.

Zwei aufeinanderfolgende Rotationen lassen sich zu einer einzigen Rotation zusammenfassen, indem man das Quaternionenprodukt der beiden Rotationen bildet:

$$\hat{q}_2(\hat{q}_1\hat{v}\hat{q}_1^{-1})\hat{q}_2^{-1} = (\hat{q}_2\hat{q}_1)\hat{v}(\hat{q}_2\hat{q}_1)^{-1}$$

Eine Herleitung dieser Formeln und anschauliche Erläuterung dieser Eigenschaften ist in [EAR] zu finden.

4.4.3 Sphärische lineare Interpolation

Auch bei der Beschreibung von Orientierungen mit Hilfe von Quaternionen ist das Problem zu lösen, wie man bei der Animation zwischen zwei unterschiedlichen Orientierungen Interpolationswerte findet. Betrachtet man die Quaternionen als Punkte im vierdimensionalen Raum, so ist dies jedoch einfach. Verbindet man zwei vorgegebene Punkte durch

eine Gerade, so sind die Zwischenpunkte auf der Geraden Beschreibungen für die Zwischenorientierungen im Raum. Eine Gerade im vierdimensionalen Raum kann man sich wie im dreidimensionalem Raum definiert vorstellen. Sie beginnt an einer Quaternion (Punkt) und hat die Richtung der Differenz der beiden Quaternionen (Punkte).

Ein Problem tritt auf, wenn man die Interpolation berechnen würde, indem man die Strecke zwischen den Stützpunkten in äquidistante Stücke unterteilt. Die Rotation würde nicht mit einer konstanten Winkelgeschwindigkeit ablaufen, in der Mitte wäre die Rotationsgeschwindigkeit höher als in der Nähe der Stützpunkte [SHO85].

Ken Shoemake stellte in seinemn Artikel [SHO85] eine Interpolationsformel zwischen zwei Quaternionen $\hat{q}_1$ und $\hat{q}_2$ vor, die auf dem Großkreisbogen der Einheitskugel im vierdimensionalen Raum interpoliert. Den Großkreisbogen kann man sich genauso vorstellen wie im dreidimensionalen Fall und wie er dafür in Abb. 4.5 verwendet worden ist. Dieser Algorithmus wird Slerp (spherical linear interpolation) genannt. Dabei wird jedoch nicht wie beim obigen Ansatz ein konstantes Stück auf der Geraden bewegt, vielmehr wird eine Interpolation um einen Bruchteil t des Rotationswinkels durchgeführt:

$$slerp(\hat{q}_1, \hat{q}_2, t) = \hat{q}_{zwischen}(t) = \frac{sin((1-t)*\theta)}{sin(\theta)}\hat{q}_1 + \frac{sin(t*\theta)}{sin(\theta)}\hat{q}_2$$

mit

$$\hat{q}_1 \cdot \hat{q}_2 = cos(\theta) \qquad (Skalarprodukt,\ \hat{q}_1, \hat{q}_2\ Einheitsquaternionen)$$

Durch Unterteilen des Parameters t, der von $t = 0$ (Startorientierung) bis $t = 1$ (Zielorientierung) läuft, in gleiche Teile, kann man eine Rotation mit konstanter Winkelgeschwindigkeit erreichen.

4.4.4 Quaternioneninterpolation und Stetigkeit

Nun kann man die sphärische lineare Interpolation mit der linearen Interpolation zwischen zwei Stützpunkten im Raum vergleichen. Wie dort tritt wiederum das Problem auf, daß beim Aneinanderreihen von einzelnen Interpolationen die Stetigkeit an den Verbindungsstellen nicht gewährleistet bleibt (vgl. Kapitel 3.2 - 3.3 und Abb. 4.6).

Ken Shoemake stellte bei der Entwicklung des slerp-Algorithmus eine Lösung dar, die sich an der Konstruktion von Bézier-Kurven orientiert [SHO85, Seite 249]. Hier soll jedoch ein Ansatz erläutert werden, der eine Analogie zur Konstruktion der Catmull-Rom Splines darstellt. Dieser Ansatz wurde von John Schlag vorgestellt [SLG91]. Er orientiert sich an einer geometrischen Konstruktion, die zur Berechnung der Catmull-Rom Splines 1988 von Barry and Goldmann hergeleitet wurde.

Für eine Interpolation zwischen zwei Quaternionen $\hat{q}_1$ und $\hat{q}_2$ mit dem Vorgänger $\hat{q}_0$ und dem Nachfolger $\hat{q}_3$ läßt sich der Algorithmus in folgende drei Schritte zerlegen:

1. $\hat{q}_{10} := slerp(\hat{q}_0, \hat{q}_1, t+1)$
   $\hat{q}_{11} := slerp(\hat{q}_1, \hat{q}_2, t)$
   $\hat{q}_{12} := slerp(\hat{q}_2, \hat{q}_3, t-1)$

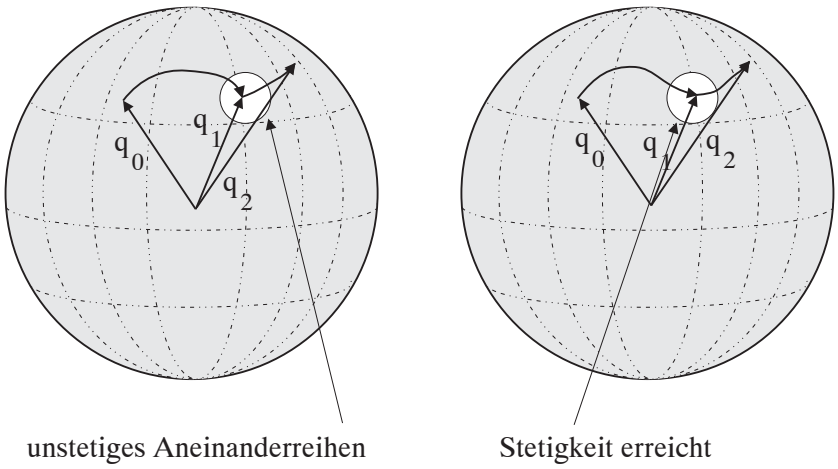


Abbildung 4.6: Stetigkeit der Orientierungsinterpolation

2. $\hat{q}_{20} := slerp(\hat{q}_{10}, \hat{q}_{11}, (t+1)/2)$
   $\hat{q}_{21} := slerp(\hat{q}_{11}, \hat{q}_{12}, t/2)$

3. $\hat{q}_{zwischen} := slerp(\hat{q}_{20}, \hat{q}_{21}, t)$

Dieser Algorithmus erzeugt Interpolationen, die an der Übergangsstelle C^1-stetig sind. Damit kann man eine gleichmäßige Orientierungsänderung über mehrere Orientierungsfestlegungen hinweg erreichen. Dies ist besonders deshalb interessant, da der slerp-Algorithmus immer auf dem kürzesten Weg die Interpolation ausführt. Wie im dreidimensionalen Fall erläutert, wird entlang dem Großkreissegment interpoliert, dessen Öffnungswinkel kleiner 180^o ist. Will man eine Bewegung andersherum, also entlang dem längeren Kreissegment durchführen, muß man diesen deshalb in mehrere Teile unterteilen. Damit die Bewegung aber dennoch auf dem gesamten Weg gleichmäßig und glatt berechnet wird, ist mit diesen Formeln ein stetiges Aneinanderreihen mehrerer Interpolationsschritte möglich.

Es läßt sich sicher noch eine Erweiterung der Quaternioneninterpolation denken, die wie die Kochanek-Bartels Splines den Interpolationsverlauf über weitere Parameter steuern läßt. Dies direkt zu übertragen ist jedoch recht aufwendig und würde den Rahmen der vorliegenden Arbeit überschreiten. Auch wird es für den Anwender schwierig sein zu verstehen, welche Auswirkungen die Parameter auf den Interpolationsverlauf haben. Während der Kurvenverlauf noch einfach zu erkennen ist, ist der Verlauf der Quaternioneninterpolation, der ja als Bewegung im vierdimensionalen Raum interpretiert werden kann, schlecht anschaulich darstellbar.

4.4.5 Umwandlung in Quaternionen

Da im Graphikbereich und in PHIGS die Orientierung der virtuellen Kamera durch die Angabe eines Blickrichtungsvektors (VPN) und eines Ausrichtungsvektors (VUP) angegeben wird, ist eine Umrechnung dieser Angaben in Quaternionen und zurück nötig. Die

Angaben beziehen sich auf das hier verwendete Koordinatensystem aus dem Graphikbereich mit senkrechter y-Achse (vgl. Abb. 4.3). Dazu wird die Darstellung von PHIGS erst in Eulerwinkel übertragen und dann in Quaternionen umgerechnet.

Hat man den VPN als dreidimensionalen Vektor in kartesischen Koordinaten vorliegen, so kann man diese in Polarkoordinaten umrechnen. Man hat damit bereits zwei der drei Eulerwinkel, nämlich PITCH und YAW, bestimmt.

Den Rollwinkel bestimmt man aus dem Winkel zwischen dem VUP und der y-Achse im kartesischen Koordinatensystem. Anschaulicher ist es sogar, für VUP in einem Programm nur die Winkelinformation abzuspeichern, da dies für den Benutzer leichter nachzuvollziehen ist. Für den VUP ist ja nicht jeder beliebige Vektor erlaubt, wenn man voraussetzt, daß VUP senkrecht auf VPN stehen soll.

Hat man die Angaben in Eulerwinkeln gegeben, kann man diesen leicht in Quaternionenangaben umrechnen: Jeder Eulerwinkel entspricht ja einer Rotation um eine Achse. Führt man diese Rotationen entsprechend nacheinander aus, also erst die Rotation um die y-Achse mit dem YAW-Winkel, dann um die x-Achse mit dem PITCH-Winkel und zuletzt um die z-Achse mit dem ROLL-Winkel, erreicht man die gewünschte Koordinatentransformation.

Die Quaternionen zu diesen Rotationen lassen sich einfach finden:

Rotation um die y-Achse:

$$\hat{q}_{YAW} = [cos(\frac{\theta}{2}), [0, sin(\frac{\theta}{2}), 0]]$$

Rotation um die (transformierte) x-Achse:

$$\hat{q}_{PITCH} = [cos(\frac{\theta}{2}), [sin(\frac{\theta}{2}), 0, 0]]$$

Rotation um die (transformierte) z-Achse:

$$\hat{q}_{ROLL} = [cos(\frac{\theta}{2}), [0, 0, sin(\frac{\theta}{2})]]$$

Da sich die einzelnen Rotationen zusammenfassen lassen, kann man die Quaternion dieser Orientierung leicht bilden:

$$\hat{q}_{Trans} = \hat{q}_{YAW}\ \hat{q}_{PITCH}\ \hat{q}_{ROLL}$$

Ebenso einfach kann man aus einer Quaternionenbeschreibung wieder den VPN-Vektor und den VUP-Vektor bestimmen.

Man geht davon aus, daß die Orientierung der Kamera in Richtung der z-Achse liegt, solange sie nicht rotiert wurde (ihre Eulerwinkel also alle gleich null wären). Da dann VPN_{Grund} durch den Vektor $[0, 0, 1]$ dargestellt wird, läßt sich der gesuchte VPN bestimmen zu:

$$VPN = \hat{q}_{Trans} VPN_{Grund} \hat{q}_{Trans}^{-1}$$

Im Graphikkoordinatensystem ist die Richtung des VUP-Vektors im unrotierten Zustand dann identisch mit dem Richtungsvektor der y-Achse ($VUP_{Grund} = [0, 1, 0]$). Damit läßt sich VUP bestimmen zu:

$$VUP = \hat{q}_{Trans} VUP_{Grund} \hat{q}_{Trans}^{-1}$$

Damit kann man die Konversion aus der Orientierungsbeschreibung einer virtuellen Kamera in PHIGS in eine Quaternionenbeschreibung und zurück durchführen.

Kapitel 5

Zeitsteuerung

Eine Animation wird nicht nur durch Definition von Orts- und Orientierungsinformationen festgelegt, zur vollständigen Definition gehören ebenfalls noch Zeit- und Geschwindigkeitsangaben.

Eine Zwischenposition ist also nur vollständig definiert, wenn neben Position und Orientierung auch der Zeitpunkt, an dem die Position erreicht sein soll, oder auch die Geschwindigkeit und Beschleunigung, mit der sich der Gegenstand, z.B. die Kamera, bewegen soll, festgelegt sind. Dabei läßt sich der Zeitpunkt aus der bisherigen Bewegungsgeschwindigkeit berechnen und umgekehrt; diese Angaben sind also miteinander verknüpft.

Um natürliche Bewegungsabläufe zu erreichen, ist es notwendig, die Bewegungsgeschwindigkeit und die Beschleunigung den physikalischen Gegebenheiten anzupassen. Ein Gegenstand soll seine Bewegungsgeschwindigkeit normalerweise nicht abrupt ändern. Die Beschleunigung kann zwar sehr groß werden, sie sollte am Übergang von einem Zwischenwert auf einen anderen aber nicht springen.

Um eine komplette Zeitsteuerung zu ermöglichen, ist es zuerst einmal notwendig, unsere Bewegungskurve in Teilstücke frei wählbarer Länge unterteilen zu können. Nur dann lassen sich Wegstrecken zurücklegen, die einer einzelnen Animationssequenz entsprechen.

5.1 Kurvenlänge einer parametrischen Kurve

Die Berechnung dieser Teilstücke erweist sich bei den hier verwendeten parametrischen Kurven allerdings als problematisch, da kein linearer Zusammenhang zwischen Parameter t und Kurvenlänge besteht. Bereits die Bestimmung der Kurvenlänge eines Bewegungsabschnittes ist nicht immer exakt möglich. Die Unterteilung in Teilabschnitte gleicher oder vorgegebener Länge läßt sich am einfachsten durch eine Kurvenlängenparametrisierung, und dann ebenfalls nur näherungsweise, lösen.

Es wird daher ein Algorithmus zur Kurvenlängenbestimmung vorgestellt, der gleichzeitig eine Kurvenlängenparametrisierung durchführen kann.

5.1.1 Allgemeine Formel

Die Kurvenlänge einer parametrischen Kurve zwischen zwei Punkten läßt sich zwar durch eine Integrationsformel ausdrücken, die geschlossene Lösung des Integrals ist aber häufig nicht zu finden.

Eine parametrische Kurve $[x(t), y(t), z(t)]$ hat zwischen t_0 und t_1 die Länge :

$$l = \int_{t_0}^{t_1} \sqrt{x'(t)^2 + y'(t)^2 + z'(t)^2}dt$$

also gilt für Interpolation mit kubischen Polynomen:

$$l = \int_{t_0}^{t_1} \sqrt{(3a_x t^2 + 2b_x t + c_x)^2 + (3a_y t^2 + 2b_y t + c_y)^2 + (3a_y t^2 + 2b_y t + c_y)^2}dt$$

Eine geschlossene Lösung dieses allgemeinen Integrals existiert nicht, deshalb müssen Näherungsverfahren zur Berechnung des Integrals verwendet werden.

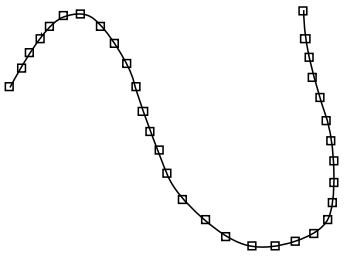
5.1.2 Iterative Verfahren

In der Literatur werden Näherungsverfahren, wie die Verwendung der Gaußschen Quadraturformeln [GUN90] oder am einfachsten die simple Annäherung durch eine ausreichend große Anzahl von Geradenstücken, vorgeschlagen.

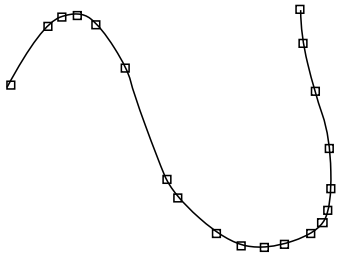
Bei den hier verwendeten kubischen Kurven ist es jedoch effizienter, sich die geometrischen Voraussetzungen dieser Kurvenbeschreibung unter Berücksichtigung des Verwendungszwecks zunutze zu machen, wie in [FIG94] eingeführt. Ein Bewegungablauf ist ausreichend exakt angenähert, wenn die Bewegungsrichtung sich zwischen zwei Teilstücken nicht merklich ändert. Es reicht daher aus, die Kurve in Teilstücke zu zerlegen, die näherungsweise kollinear, also –anschaulich betrachtet– annähernd parallel sind. Diese Teilstücke lassen sich dann durch Geradenabschnitte ersetzen, deren Länge wiederum leicht exakt bestimmbar ist. Dies entspricht einer Unterteilung in Geradenstücke durch Untersuchung des Krümmungsradius der Kurve. Auch zum Darstellen einer Kurve muß man diese in Teilabschnitte unterteilen, die durch einzelne Geradensegmente angenähert werden können. Man spricht dabei von Sampling der Kurve. Durch Untersuchung des Krümmungsradiuses kann man somit eine dem Kurvenverlauf angepaßte, adaptive Unterteilung durchführen, die in Kurvenabschnitten starker Krümmung ausreichend genau approximiert, ohne in Bereichen geradlinigen Kurvenverlaufs unnötig viele Berechnungen durchzuführen.

Den Unterschied zwischen einer Approximation mit konstanter Parametervariation und der Approximation mit der oben beschriebenen Methode zeigt Abb.5.1. Die Kurve mit konstanter Approximation muß durch eine deutlich höhere Zahl von Kurvenstücken angenähert werden, um in den Wendepunkten die gleiche Interpolationsgenauigkeit zu erreichen.

Die Untersuchung der Kollinearität von zwei Kurvenstücken durch die Punkte P_1, P_2 und P_3 ist auf verschiedene Weisen möglich ([FIG94]):



große Zahl von Stützstellen bei konstanter Unterteilung



geringere Zahl von Stützstellen bei adaptiver Unterteilung

Abbildung 5.1: Interpolation mit konstanter und adaptiver Unterteilung

- Die Fläche F des durch P_1, P_2 und P_3 aufgespannten Dreieckes ist klein.
- Der Winkel α zwischen den Teilstücken P_1P_2 und P_2P_3 beträgt annähernd 180^o.
- Der Abstand Δx des Punktes P_2 von der Geraden P_1P_3 ist klein.
- Die Kurvenlänge des Geradenstückes von P_1 nach P_3 entspricht annähernd der Summe der beiden Kurvenstücke P_1P_2 und P_2P_3.
- Die Tangenten an die Kurve in den drei Punkten P_1, P_2 und P_3 sind annähernd parallel.

Für alle Methoden ist ein passendes Toleranzkriterium zu wählen, um die gewünschte Genauigkeit zu erzielen. Dies erweist sich vor allem für das Winkelkriterium als einfach. Während das Winkelkriterium vom gewählten Maßstab völlig unabhängig ist, werden die anderen Kriterien davon beeinflußt und sind je nach vorgegebener Darstellung unterschiedlich zu wählen. So wird die Fläche F des Dreieckes z.B. größer, wenn der Abstand der Punkte P_1, P_2 und P_3 größer wird, auch wenn die Kollinearität der Kurvensegmente gleich bleibt. Das Flächenkriterium hat allerdings den Vorteil, daß es einfach zu programmieren und in der Berechnung sehr effektiv ist, da die Berechnung keinerlei Auswertung von Wurzelfunktionen enthält ([FIG94]).

5.1.3 Aliasing

Jede Wahl des Kriteriums schließt jedoch das Problem des Aliasing nicht aus. Aliasing ist ein Begriff, der aus der Signaltheorie stammt. Um eine gegebene Funktion aus einigen Abtastwerten näherungsweise wiedergewinnen zu können, muß sie mit einer Frequenz, die doppelt so groß ist wie die höchste in der Funktion enthaltene Frequenz, abgetastet werden.

Übertragen auf den graphischen Bereich bedeutet das, daß Aliasing auftritt, wenn man ein Kurvenstück durch einen Geradenabschnitt annähern würde, obwohl in diesem Abschnitt die Kurve noch einen Wendepunkt aufweist. In Abb. 5.2 ist ein kubisches Polynom durch

drei auf einer Geraden liegende Stützpunkte angenähert, obwohl die Kurve in diesem Bereich in keiner Weise gerade verläuft.
Hat man die Funktionswerte der Punkte in P_1 und P_3 gegeben und nimmt zur Untersuchung der Kollinearität den Punkt P_2 in der Mitte zwischen P_1 und P_3, so würde die oben beschriebene Methode der Untersuchung des Krümmungsradius ergeben, daß der Kurvenverlauf durch die Geradenabschnitte P_1P_2 und P_2P_3 angenähert werden könnte. Wie unschwer zu erkennen ist, entspricht dies jedoch nicht dem darzustellendem Kurvenverlauf.

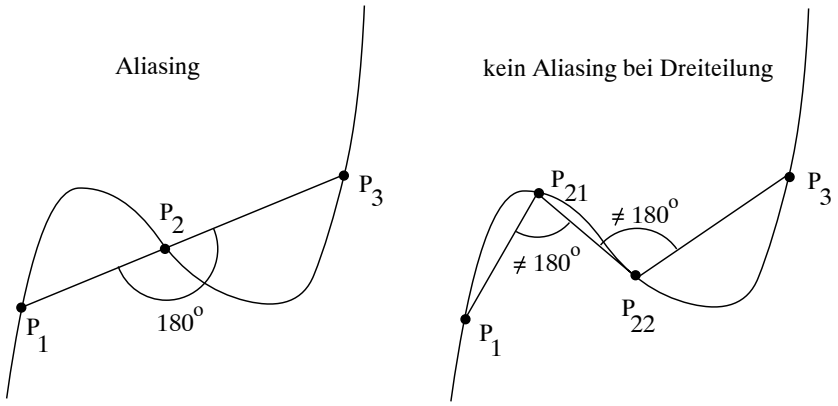


Abbildung 5.2: Aliasing

Allgemein ist das Aliasing nie ganz zu vermeiden, wenn man keine genaue Information über die Kurvendefinition hat. Ansätze, wie die zufällige Wahl des Punktes P_2 nicht genau zwischen P_1 und P_3, die Begrenzung des Abstandes zwischen P_1 und P_3 auf eine maximale Entfernung und andere Methoden können die Wahrscheinlichkeit, daß Aliasing auftritt, zwar vermindern, Aliasing jedoch nie ganz ausschließen.

Da ein kubisches Polynom, wie es hier verwendet wird, maximal einen Wendepunkt haben kann, kann man aber durch eine andere Methode das Aliasing beim Samplen der Kurve vermeiden. Dazu teilt man den Abschnitt P_1P_3 nicht in zwei gleichlange Teilstücke, sondern in drei Teilstücke durch Einfügen von zwei weiteren Punkten P_{21} und P_{22}, z.B. nach einem und nach zwei Dritteln der Wegstrecke P_1P_3. Untersucht man die drei Geradenstücke jetzt darauf, ob diese auf einer Geraden liegen, also ob an den beiden Punkten P_{21} und P_{22} der Winkel zwischen den Geradenstücken annähernd 180^o beträgt, so kann man eine sichere Aussage darüber machen, ob die Annäherung der Kurve durch ein Geradenstück zwischen P_1 und P_2 ausreichend exakt ist. Anschaulich kann man das Vermeiden des Aliasing wie folgt erklären: Eine beliebige Gerade, die mit einem kubischen Polynom geschnitten wird, kann mit diesem maximal drei Schnittpunkte gemeinsam haben. Liegen deshalb vier Punkte der Kurve annähernd auf einer Geraden, so kann dies keine Schnittgerade sein, sondern entspricht nahezu einer Tangenten an einem nur schwach gekrümmten Kurvenabschnitt. Mathematisch läßt sich das wie folgt begründen:

Will man die Schnittpunkte eines kubischen Polynoms ax^3+bx^2+cx+d mit einer Geraden $ex+f$ bestimmen, so setzt man die beiden Funktionen gleich:

$$ax^3+bx^2+cx+d=ex+f \qquad \text{also} \quad ax^3+bx^2+(c-e)x+(d-f)=0$$

Dieses Polynom dritten Grades kann aber maximal drei Nullstellen haben; damit kann das ursprüngliche Polynom nur drei Punkte mit der Geraden gemeinsam haben. Folglich können vier Punkte des kubischen Polynoms nicht auf einer Geraden liegen.

5.1.4 Kurvenlängenparametrisierung

Ein weiteres Problem ist, daß der Zusammenhang zwischen der Parametervariablen und der Kurvenlänge einer parametrischen Kurve nicht linear ist. Eine Veränderung des Kurvenparameters um ein bestimmtes Δt kann unterschiedlichen Bogenlängen auf der parametrischen Kurve entsprechen. Für eine Zeit- und Bewegungssteuerung in der Animation ist es jedoch unumgänglich, einen Abschnitt bestimmter Länge auf der Kurve zurücklegen zu können. Da dies nicht durch die entsprechende Wahl des Parameters zu gewährleisten ist, ist es sinnvoll, eine Kurvenlängenparametrisierung durchzuführen.

Man sagt, eine Kurve sei in Abhängigkeit von ihrer Länge parametrisiert, wenn eine Veränderung des Kurvenparameters um ein bestimmtes Δt genau einer bestimmten Kurvenlänge Δk entspricht. Liegt dieser Zusammenhang nicht vor, so kann man sich eine Tabelle der Parameterwerte notieren und so jederzeit den entsprechenden Parameterwert zu einer vorgegebenen Länge finden. Dieser Vorgang heißt Kurvenlängenparametrisierung.

Da dies bei kubischen Polynomen notwendig ist, kann man bei der Kurvenlängenbestimmung mit der oben beschriebenen Methode gleich eine Aufzeichnung des Kurvenverlaufes in einer Tabelle durchführen, um später jede beliebige Position auf der Kurve schnell wiederzufinden.

Insgesamt bietet sich für das adaptive Unterteilen ein rekursiver Algorithmus an, der gleichzeitig die Längenberechnung und die Kurvenlängentabellierung durchführt. Er wurde von B. Guenter und R. Parent zuerst für eine Zweiteilung in [GUN90] vorgestellt und ist hier auf die Dreiteilung des Kurvensegmentes erweitert. Interessant ist besonders, daß die Tabelle trotz rekursiven Ansatzes in der richtigen global Reihenfolge aufgebaut wird.

```
sample (P1,P3)
begin
   P21 = P1 + (1/3) * (P1-P3)
   P22 = P1 + (2/3) * (P1-P3)
   if (flat(P1,P21,P22,P3))
     begin
       parameterlist (P1,laenge)
       laenge = laenge + gerade(P1,P3)
     end
   else
     begin
       sample (P1, P21)
       sample (P21,P22)
       sample (P22,P3 )
     end
   return
end
```

Zuerst werden die beiden Zwischenpunkte P21 und P22 bestimmt. Es erfolgt eine Untersuchung des Krümmungsradius der drei Geradenabschnitte durch die Funktion `flat`, die eines der oben vorgestellten Verfahren zur Bestimmung der Kollinearität verwendet. Ist der Kurvenverlauf annähernd gerade, so werden der Parameterwert und die bisherige Kurvenlänge durch `parameterlist` in die Liste der Kurvenlängenwerte aufgenommen und es wird danach durch die Funktion `gerade` die Länge des Geradenabschnittes zwischen P1 und P3 bestimmt und zur Gesamtlänge addiert. Ist der Kurvenverlauf nicht ausreichend gut angenähert, so wird die Routine `sample` rekursiv für die drei Teilabschnitte (P1, P21), (P21, P22) und (P22, P3) aufgerufen.

5.2 Festlegung der Geschwindigkeitskurve

Zur Festlegung der Geschwindigkeit der Bewegung muß eine funktionale Beschreibung des Geschwindigkeitsverlaufes für jedes Bewegungssegment definiert werden. Dazu dienen Zeit-Geschwindigkeitskurven. Sie geben an, mit welcher Geschwindigkeit die Bewegung zu einem bestimmten Zeitpunkt durchgeführt werden soll. Konstante Geschwindigkeit würde einer konstanten Zeit-Geschwindigkeitskurve $f_{Zeit}(t)$ entsprechen, eine konstante Beschleunigung z.B. einer linear steigenden Kurve. Die bis zu einem Zeitpunkt zurückgelegte Wegstrecke läßt sich durch einfache Integration der Kurve bis zu diesem Zeitpunkt berechnen. Legt man für ein parametrisches Kurvensegment z.B. eine bestimmte Zeit-Geschwindigkeitskurve fest, so sollte das Integral unter dieser Kurve der Gesamtlänge der Bewegungskurve entsprechen. Um dies zu erreichen ist eine einfache Skalierung der Zeit-Geschwindigkeitskurve im Verhältnis k von Kurvenlänge zu Zeitkurvenintegral notwendig.

$$k = \frac{\int_{p_0}^{p_1} f_{par}(p)dp}{\int_{t_0}^{t_1} f_{Zeit}(t)dt}$$

Damit läßt sich eine Zeit-Geschwindigkeitskurve vorgegebener Länge an eine parametrische Bewegungkurve f_{par} beliebiger Länge anpassen. Daher ist es ausreichend, Zeit-Geschwindigkeitskurven mit einer festen Länge festzulegen. Definiert man für jeden Bewegungsabschnitt z.B. solch eine Kurve durch Angabe von zehn festen Geschwindigkeitswerten zu zehn festen Zeitpunkten, so kann man diese dann der jeweiligen Bewegungskurve durch Skalierung mit k anpassen.

5.2.1 Stetigkeit der Bewegung

Zur Anpassung des Bewegungsablaufes an die natürlichen Gegebenheiten gehört hier vor allem, daß sich die Bewegungsgeschwindigkeit nicht sprunghaft ändern kann. Sogar die Beschleunigung einer Bewegung sollte stetig sein. Diese physikalischen Voraussetzungen sollten bei der Definition von Zeit-Geschwindigkeitskurven berücksichtigt werden.

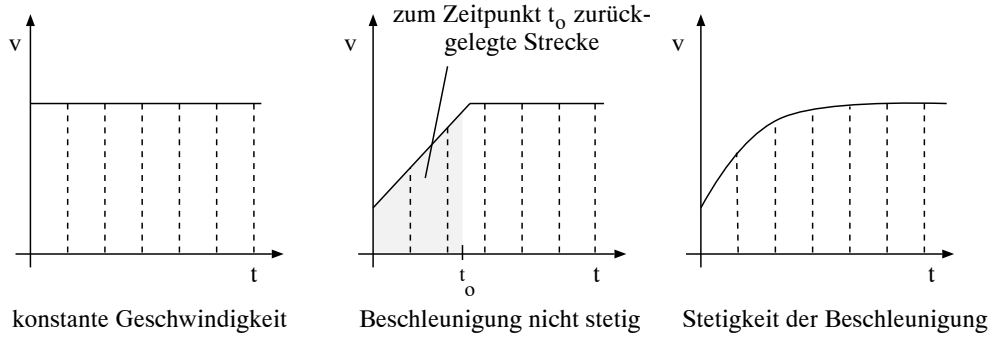


Abbildung 5.3: Kurven zur Festlegung der Geschwindigkeit

Eine stetige Bewegungsgeschwindigkeit liegt vor, wenn die Zeit-Geschwindigkeitskurve an Stellen, bei denen zwei Kurvensegmente zusammengefügt werden, keine Sprünge macht. Ist die 1. Ableitung stetig, so liegt eine stetige Beschleunigung vor, da die Ableitung einer Geschwindigkeit der Beschleunigung entspricht. Daran ändert auch die Skalierung einer Zeit-Geschwindigkeitskurve nichts, solange man dafür Sorge trägt, daß die Stetigkeit beim Zusammensetzen einzelner Kurvensegmente weiterhin erhalten bleibt.

Da man die Zeit-Geschwindigkeitskurve am einfachsten durch Angabe mehrerer Stützpunkte festlegt, also durch Angabe von einigen diskreten Werten, ist es sinnvoll, diese diskreten Werte durch eine Spline-Kurve zu verbinden. Dadurch kann man auch über mehrere Kurvensegmente hinweg die Stetigkeit der Bewegung gewährleisten. Deshalb wird im vorliegenden Programm eine Kurve benutzt, die durch zehn diskrete Werte charakterisiert und durch ein kubisches Polynom nach der Festlegung der Catmull-Rom Splines interpoliert wird. Da das Integral eines expliziten Catmull-Rom Splines (ein kubisches Polynom, keine parametrische Kurve) analytisch leicht zu berechnen ist, kann man die Integration der Wegstrecke einfach durchführen.

Hat man also die Skalierung der Zeit-Geschwindigkeitskurve vorgenommen, so kann man diese in eine der gewünschten Geschwindigkeit entsprechende Zahl von Zeitabschnitten unterteilen. Für jeden Zeitabschnitt läßt sich durch einfache Integration der Zeit-Geschwindigkeitskurve die bisher zurückgelegte Wegstrecke berechnen. Nun muß man nur noch aus der Liste der Kurvenlängenparametrisierung die zu dieser Wegstrecke gehörende Position auf der Bewegungskurve heraussuchen.

5.2.2 Berechnen der gewünschten Position

Um zu einer vorgebenen Wegstrecke den dazugehörigen Raumpunkt zu finden, sucht man aus der angelegten Liste der Kurvenlängenparametrisierung das Segment der Kurve heraus, das vor der gesuchten Wegstrecke beginnt und nach dieser endet. Da der Krümmungsradius zwischen diesen beiden Punkten vernachlässigbar ist, kann man die gesuchte Position dann durch einfache Geradeninterpolation zwischen diesen Punkten berechnen.

5.3 Interpolation von Orientierungs- und Zoomangaben

Bei der Orientierungsinterpolation ist aufgrund der Linearität des Slerp-Algorithmus keine aufwendige Umparametrisierung der Orientierungen notwendig. Man kann vielmehr den linearen Zusammenhang nutzen, um die Orientierungsinformation zu berechnen.

Will man die Orientierungsinterpolation an die oben beschriebene Zeit-Geschwindigkeitskurve ankoppeln, so genügt es, die gefundene Wegstrecke zu einem Zeitpunkt durch die gesamte Wegstrecke des Bewegungsabschnittes zu teilen. Damit hat man den Parameter bestimmt, der zur Berechnung der Orientierungsdaten benötigt wird.

Ebenso kann man diesen Parameter heranziehen, um einen Zwischenwert des Zoomfaktors, also des Abbildungsauschnittes zu berechnen. Der Zoomfaktor war in Kapitel 2 als skalarer Wert festgelegt worden, eine lineare Interpolation ist damit einfach möglich.

Kapitel 6

Interaktive 3D-Eingabe

Bisher wurden mathematische Methoden vorgestellt, mit denen man zwischen Stützpunkten fehlende Werte interpolieren kann. In diesem Kapitel werden Wege aufgezeigt, wie die Position und die Orientierungsinformation für einen Stützpunkt vom Benutzer graphisch-interaktiv eingegeben werden kann.

Die Eingabe von Geometriedaten (Positionen, Orientierungen etc.) im dreidimensionalen Raum mit Hilfe der üblichen Eingabemethoden, die man an einer Workstation oder an einem PC voraussetzen kann, erweist sich als schwierig und für den Anwender schwer verständlich. Spezielle Geräte für direkte 3D-Eingabe (z.B. Spaceball) kann man nicht als allgemein verfügbar voraussetzen. Deshalb müssen Methoden entwickelt werden, die Eingabe mit den üblicherweise vorhandenen Hilfsmitteln Tastatur und Maus zu ermöglichen. Dabei ist darauf zu achten, daß die Eingabe für den Benutzer leicht nachvollziehbar und schnell erlernbar ist, in der zweidimensionalen Projektion gut verständlich erfolgt und in möglichst wenig Schritten zu einem Ergebnis führt.

Da ein Projektionspunkt verschiedenen Punkten im Weltkoordinatensystem entspricht, ist die Zuordnung eines Bildpunktes zu einem Raumpunkt nicht eindeutig. Damit ist auch die Zuordnung einer 2D-Mausbewegung zu einer beliebigen Bewegung im 3D-Raum nicht direkt möglich.

Bei der vorliegenden Aufgabenstellung ergaben sich zwei verschiedene Arten der Eingabe:

- Einerseits muß ein Stützpunkt auf der Bewegungskurve definiert werden. Dies erfordert die Eingabe eines beliebigen Punktes im dreidimensionalen Raum. Auch einen Teil der Orientierung, den Endpunkt des VPN, also des Blickrichtungsvektors, kann man durch Eingabe eines Raumpunktes festlegen.

- Zusätzlich ist noch die Änderung des VUP-Vektors zu definieren, der durch eine Rotation senkrecht zur VPN-Achse festgelegt wird. Dazu ist eine Eingabe des Rotationswinkels bezüglich dieser Achse notwendig. Dieser Winkel entspricht dem im 5. Kapitel eingeführten Rollwinkel.

6.1 Bekannte Eingabemethoden

6.1.1 Direkte Eingabe

Am naheliegendsten ist die direkte Koordinateneingabe durch Zahlenwerte oder mit Hilfe von Schiebereglern. Dies erfordert jedoch vom Anwender ein gutes Vorstellungsvermögen; es ist vor allem schwierig, bestimmte Positionen im Raum zu treffen, deren Koordinaten nicht exakt bekannt sind. Will man einen Raumpunkt bewegen, so muß man sich erst einmal dessen neue Koordinaten berechnen, bevor man diese eingeben kann. Deshalb ist diese Art der Eingabe zwar in einigen Spezialfällen sinnvoll, jedoch langsam, nicht intuitiv und deshalb für diese Anwendung nur ergänzend geeignet.

6.1.2 Mauseingabe in den Projektionsebenen

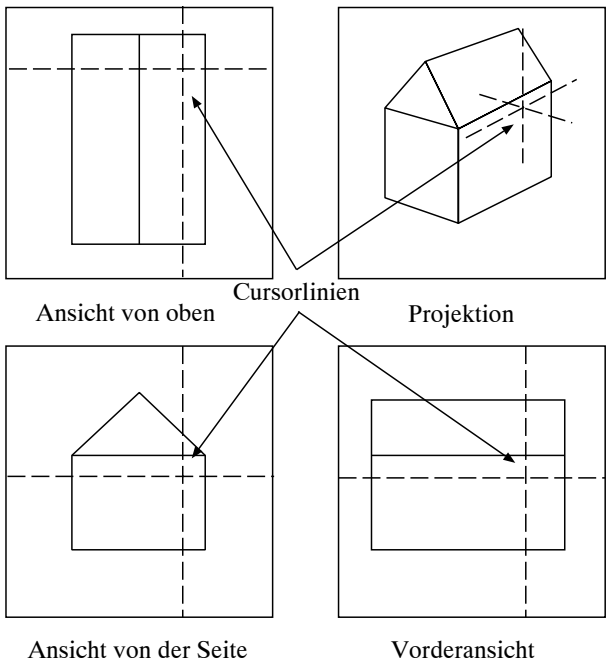


Abbildung 6.1: Maussteuerung in den Koordinatenebenen

Benutzt man zur Eingabe die Maus, so liegt es nahe, die Festlegung eines Punktes nicht direkt in der dreidimensionalen Projektion durchzuführen. Vielmehr wird die Eingabe in einzelnen Fenstern, die jeweils die Projektion einer der drei Koordinatenebenen (xy-Ebene, xz-Ebene, yz-Ebene) darstellen, vorgenommmen:

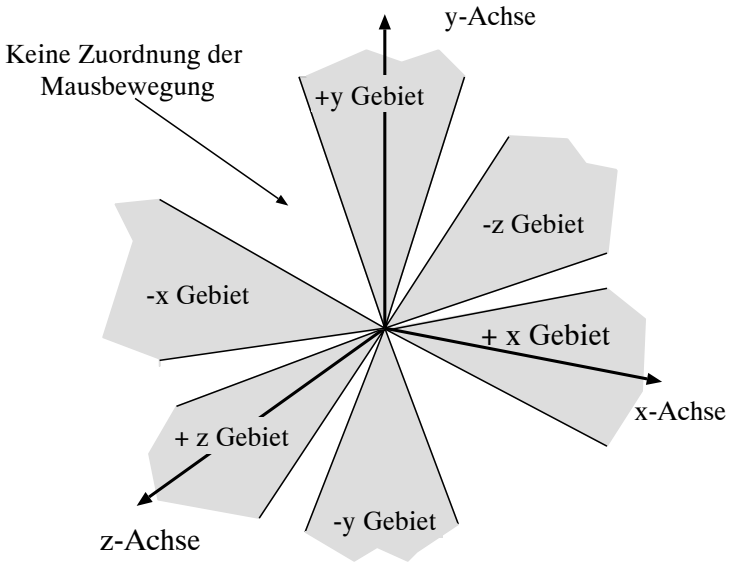
Um für den Anwender die gedankliche Übertragung der Werte in die 3. Dimension zu erleichtern, wird meist neben den drei Ebenen die Projektion dargestellt. Die Auswirkung einer Eingabe in einer Ebene wird dann direkt in der dreidimensionalen Projektion

angezeigt. Da man dabei Punkte indirekt eingibt, also nicht in der eigentlichen Projektion, sondern in der Darstellung der drei Koordinatenebenen einen Punkt definiert, ist es schwierig, einen bestimmten Raumpunkt zu finden. Um eine vollständige Definition durchzuführen, muß außerdem zwischen verschiedenen Fenstern hin- und hergesprungen werden, was den Eingabeprozeß deutlich verlangsamt.

Etwas weitergehend, aber immer noch auf eine zweidimensionale Ebene begrenzt, sind Ansätze, die die Ortseingabe von der momentanen Projektion abhängig machen. Direkt in der Projektion wird eine Punktfestlegung immer nur in der Ebene ermöglicht, die zur momentanen Projektionsebene parallel liegt. Da dies jedoch eine Rotation der Darstellung erfordert, um alle Punkte im Raum erreichen zu können, ist auch diese Methode umständlich zu handhaben.

6.1.3 Eingabe in Abhängigkeit von der Bewegungsrichtung der Maus

In der Literatur [FOL90] wird vorgeschlagen, eine Pseudo-3D-Eingabe vorzunehmen, indem man die Bewegungsrichtung einer Mausbewegung hinzuzieht, um die Mausbewegung einer Projektionsachse zuzuordnen:



Bewegungsgebiete der Mausrichtung zur Abbildung auf die Koordinatenachsen

Abbildung 6.2: Maussteuerung in Abhängigkeit von der Bewegungsrichtung

Jede der drei Achsen des Weltkoordinatensystems wird auf eine festgelegte Achse in der Projektionsebene abgebildet. Wird die Maus nun (annähernd) parallel zu einer dieser Projektionsachsen bewegt, so interpretiert man dies als Bewegung entlang der entsprechenden Achse im Raum. Die Bewegung wird dann ausschließlich als Bewegung entlang dieser Achse aufgefaßt, eine Bewegung entlang anderer Achsen ist in diesem Moment nicht möglich.

Um die Richtung der Mausbewegung auswerten zu können, muß die Maus eine größere Strecke zurücklegen. Damit die Strecke möglichst gering gehalten werden kann, muß die Maussteuerung häufig auf der Betriebssystemebene eingebunden werden. Dadurch wird die Anwendung jedoch nur schwer portabel und nicht auf jedes System übertragbar, da man nicht auf allgemeine Standards wie X-Windows zurückgreifen kann, die eine Mausbewegung nur als Pixelwert zurückliefern.

Auch für den Benutzer erweist sich diese Art der Steuerung als problematisch und gewöhnungsbedürftig. Der bewegte Punkt folgt nicht immer genau der Mausposition auf dem Bildschirm, dies kann den Benutzer irritieren. Bei bestimmten Projektionswinkeln wird die Zuordnung unmöglich, da eine Bewegungsachse nicht auf eine Projektionsachse abgebildet werden kann. Da man mit dieser Methode aber neben der Translation auch noch andere Operationen wie Rotation und Skalierung durchführen kann, ist sie für spezielle Anwendungen und nach einer längeren Lernphase zu gebrauchen, wenn man entsprechend hardwarenah programmieren kann.

6.2 Eigene Methoden

Da die bisher beschriebenen Methoden der Maussteuerung nicht optimal geeignet erschienen, wurde eine andere Maussteuerung implementiert, die sich speziell auf die Voraussetzungen des Anwendungzweckes stützt. Dabei wird zwischen der Eingabe einer Position und Orientierung für VRP und VPN und des Rollwinkels zur Festlegung von VUP unterschieden.

6.2.1 Positionieren eines Raumpunktes

Voraussetzung für die Festlegung eines Raumpunktes ist hierbei, daß es sich bei der 3D-Projektion um eine Parallelprojektion handelt, eine perspektivische Projektion ist für den vorgestellten Algorithmus nicht geeignet. Man kann den Algorithmus zwar auch auf eine Eingabe für eine perspektivische Projektion erweitern, jedoch müssen deutlich umfangreichere Berechnungen vorgenommen werden, da die Tiefeninformation ausgewertet werden muß. Damit kann die benötigte Rechenzeit so hoch sein, daß eine Realzeitberechnung der Mausbewegungen nicht mehr sichergestellt werden kann.

Als Eingabegerät für den vorgestellten Algorithmus dient eine mit drei Tasten ausgestatte Maus.

Die Bewegung der Maus wird immer auf eine Bewegung in zwei der drei Koordinatenebenen zurückgerechnet, wobei man mit Hilfe der drei Tasten jede Kombination der Koordinatenebenen auswählen kann. Man bewegt den Raumpunkt zwar stets nur in zwei Ebenen, man muß aber nicht das Darstellungsfenster wechseln oder eine Rotation der Darstellung vornehmen, um die dritte Ebene zu erreichen. Nur durch Wechsel der Maustaste ist es möglich, jeden Raumpunkt schnell und intuitiv zu erreichen. Der Punkt, der bewegt werden soll, muß nicht jedesmal neu angewählt werden, wenn die Bewegung in einer anderen Richtung erfolgen soll.

Wird ein selektierter Punkt durch eine Mausbewegung um einige Pixel bewegt, so wird diese Bewegung in eine Bewegung im Weltkoordinatensystem überführt. Die Bewegung im

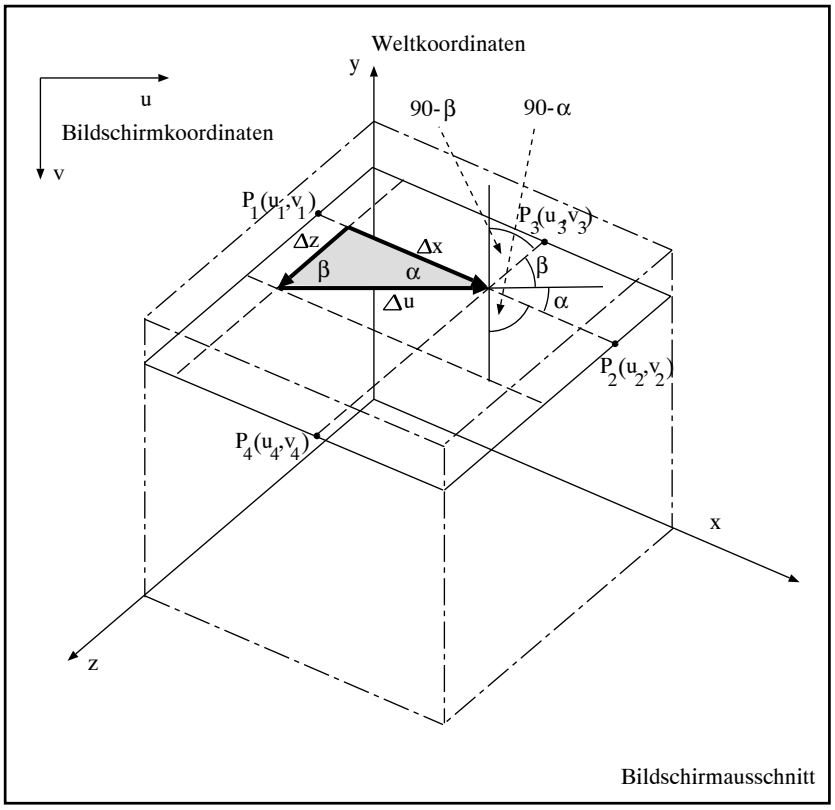


Abbildung 6.3: Maussteuerung in zwei Ebenen

Weltkoordinatensystem wird so berechnet, daß die Projektion des neuen Punktes wieder genau unter der neuen Mausposition auf dem Bildschirm zu liegen kommt. So folgt der selektierte Punkt also genau der Mausbewegung, obwohl nicht seine (zweidimensionalen) Projektionskoordinaten sondern zwei seiner Weltkoordinaten verändert worden sind.

Ein Beispiel erläutert dies für eine Bewegung in der xz-Ebene, also für konstante y-Werte (Abb. 6.3). Eine Bewegung des Punktes $P(x, y, z)$ wird aus einer Mausbewegung hergeleitet. Dazu muß berechnet werden, welche Bewegung entlang der x-Achse bzw. der z-Achse einer Mausbewegung in u- bzw. v-Richtung zugeordnet werden kann.

Zuerst müssen die Winkel α und β zwischen der projizierten x-Achse bzw. z-Achse bestimmt werden. Dazu benötigt man die Projektionskoordinaten u_1, v_1, u_2 und v_2 der Punkte $P_1(0, y, z)$ und $P_2(1, y, z)$ mit fester Lage auf der x-Achse und die uv-Koordinaten u_3, v_3, u_4 und v_4 der Punkte $P_3(x, y, 0)$ und $P_4(x, y, 1)$ fest auf der z-Achse. Damit lassen sich die Winkel α und β bzw. ihre Winkelfunktionen bestimmen zu:

$$sin(\alpha) = \frac{v_1 - v_2}{\sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2}} \qquad cos(\alpha) = \frac{u_1 - u_2}{\sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2}}$$

$$sin(\beta) = \frac{v_3 - v_4}{\sqrt{(u_3 - u_4)^2 + (v_3 - v_4)^2}} \qquad cos(\beta) = \frac{u_3 - u_4}{\sqrt{(u_3 - u_4)^2 + (v_3 - v_4)^2}}$$

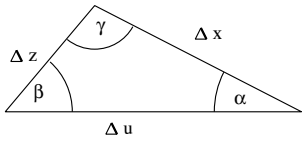


Abbildung 6.4: Dreieck der Mausbewegung

Nun kann man im Dreieck, das durch die x-Achse, die z-Achse und die u-Achse gebildet wird (Abb. 6.4) über Dreiecksgleichungen die Bewegung auf der x-Achse und z-Achse berechnen, die einer Bewegung der u-Achse entspricht. Mit der Verhältnisgleichung:

$$\frac{\Delta u}{sin(\gamma)} = \frac{\Delta x}{sin(\alpha)} = \frac{\Delta z}{sin(\beta)}$$

und dem Winkel $\gamma = 180^o - (\alpha + \beta)$ lassen sich Δx und Δz vorzeichenrichtig bestimmen zu:

$$\Delta x = \frac{\Delta u \; sin(\alpha)}{sin(\gamma)}$$

$$\Delta z = \frac{\Delta u \; sin(\beta)}{sin(\gamma)}$$

Da Δu als Mausbewegung in Pixeln angegeben wird, Δx und Δz jedoch im Weltkoordinatensystem, muß noch eine entsprechende Skalierung durchgeführt werden:

$$\Delta x^u_{Welt} = \frac{\Delta x}{\sqrt{(u_3 - u_4)^2 + (v_3 - v_4)^2}} = \frac{\Delta u \; sin(\alpha)}{sin(\gamma) \; \sqrt{(u_3 - u_4)^2 + (v_3 - v_4)^2}}$$

$$\Delta z^u_{Welt} = \frac{\Delta z}{\sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2}} = \frac{\Delta u \; sin(\beta)}{sin(\gamma) \; \sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2}}$$

Nach der gleichen Methode kann man die Bewegung der Maus in der v-Richtung in eine x_{Welt} Bewegung und eine z_{Welt} Bewegung umrechnen:

Mit $sin(90^o - \phi) = cos(\phi)$ und $cos(90^o - \phi) = sin(\phi)$ eingesetzt für $\phi = \alpha$ und $\phi = \beta$ und $\gamma = \alpha + \beta$ erhält man bereits richtig skaliert:

$$\Delta x^v_{Welt} = \frac{\Delta v \; cos(\alpha)}{sin(\gamma)\sqrt{(u_3 - u_4)^2 + (v_3 - v_4)^2}}$$

$$\Delta z^v_{Welt} = \frac{\Delta v \; cos(\beta)}{sin(\gamma)\sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2}}$$

Man kann nach der gleichen Methode auch die Gleichungen für Bewegungen in der xy-Ebene und in der yz-Ebene herleiten. Diese werden sogar noch einfacher, da bei der benutzten Parallelprojektion mit unrotiertem VUP die y-Achse stets senkrecht steht, also parallel zur v-Achse liegt.

6.2.2 Hilfen zur Interpretation der Eingabe

Zur besseren Orientierung des Anwenders beim interaktiven Eingeben von Raumpunkten wird der Arbeitsbereich, also der Raumwürfel, in dem die Position festgelegt werden kann, die ganze Zeit angezeigt.

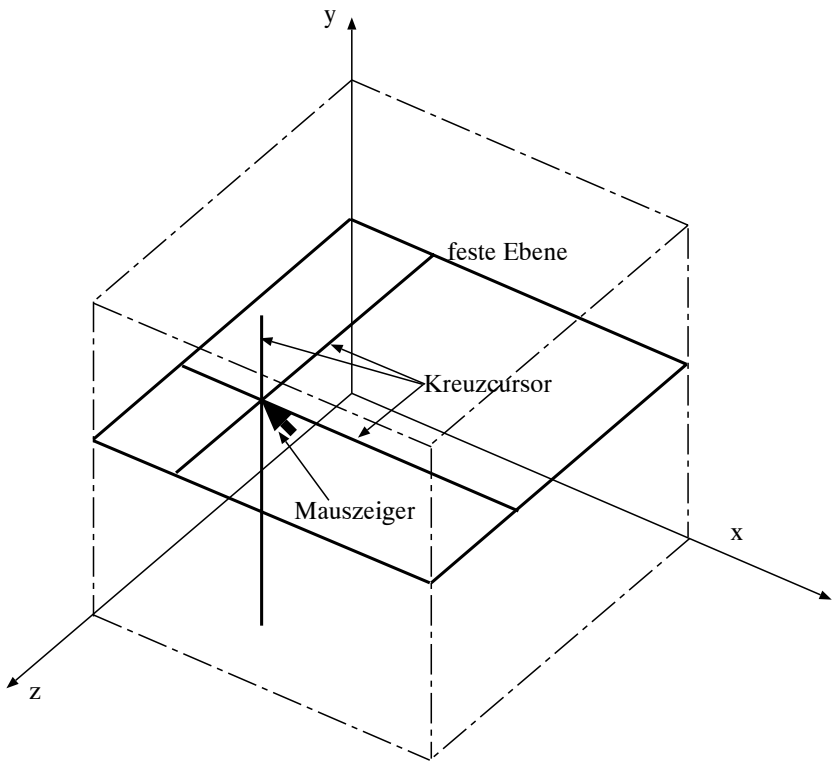


Abbildung 6.5: Hilfsmittel zur Positionssteuerung

Um dem Benutzer die Feststellung zu erleichtern, wo der angewählte Raumpunkt genau liegt, werden bei der Bewegung eines Raumpunktes zwei Hilfsmittel eingeblendet (Abb. 6.5):

- Einerseits wird ein 3D-Cursor angezeigt, der die Ausrichtung der Weltkoordinatenachsen im angewählten Punkt angibt und der bis zu den Würfelseiten ausgedehnt ist.
- Andererseits wird die jeweils feste Ebene, innerhalb der der Punkt bewegt werden kann, an ihren Umrandungen im Darstellungsraum angegeben. Die Bewegungsebene ist abhängig von der gedrückten Maustaste und wechselt mit dieser.

Dies erleichert es dem Benutzer, festzustellen, wo genau im Weltkoordinatensystem ein Punkt liegt, dem ein bestimmter Projektionspunkt zugeordnet ist. Zusätzlich ist es noch hilfreich, auch den Zahlenwert der Position anzugeben.

6.2.3 Orientierungsspezifikation

Die Orientierung wurde durch Angabe von VPN und VUP definiert. Diese beiden Werte müssen also angegeben werden, um eine komplette Orientierungsbeschreibung festzulegen.

Der VPN (Blickrichtungsvektor) wird, wie oben erwähnt, durch Angabe des Blickpunktes definiert. Aus der Position und dem Blickpunkt kann man dann den VPN berechnen. Da der Blickpunkt wie die Position ein ganz normaler Raumpunkt ist, kann man zu seiner Angabe die oben beschriebenen Methoden zur Festlegung eines Raumpunktes verwenden.

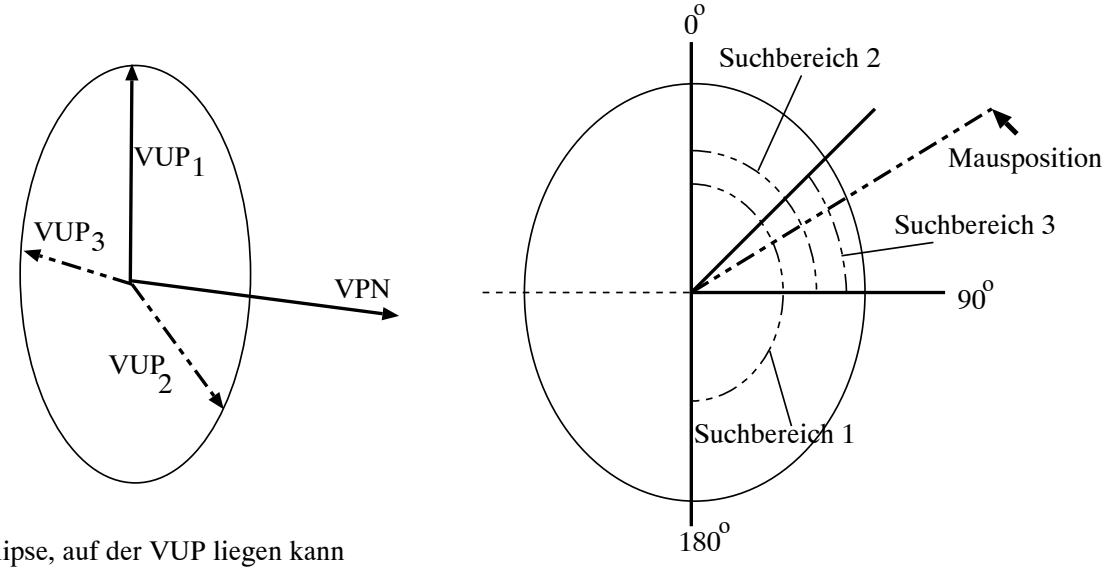


Abbildung 6.6: Hilfsmittel zur Orientierungssteuerung

Der VUP-Vektor ist nun kein frei wählbarer Vektor im Raum mehr, denn VUP und VPN sollen aufeinander senkrecht stehen. Man kann deshalb den VUP-Vektor nur noch in einer zum VPN senkrechten Ebene bewegen, indem man seinen Endpunkt um einen beliebigen Winkel um den VPN rotiert. Damit muß also lediglich der Rotationswinkel des VUP festgelegt werden. Wenn man die Länge des VUP-Vektors konstant hält, bewegt sich dessen Spitze auf einer Kreisbahn, die senkrecht zur VPN-Achse steht.

In der 3D-Projektion stellt diese Kreisbahn jedoch eine Ellipse dar (Abb.6.6), deren Hauptachsen nicht unbedingt bekannt sind.

Will man nun die Spitze des VUP-Vektors auf dieser projizierten Kreisbahn bewegen, so muß man der Mausposition einen festen Winkel auf der Kreisbahn zuordnen. Dies ist eindeutig möglich. Im Gegensatz zur Festlegung des Raumpunktes definiert man ja eine skalare Größe, den Winkel, mit Hilfe eines zweidimensionalen Eingabegerätes. Jeder Position auf der Darstellungsebene läßt sich exakt ein Winkel zuordnen, nämlich so, daß die Verbindungslinie zwischen Mausposition und Position des Raumpunktes, für den die Orientierung bestimmt werden soll, deckungsgleich mit der Projektion des VUP mit dem entsprechenden Winkel ist. Schwieriger ist jedoch die Bestimmung des gesuchten Winkels

aus der Mausposition. Da die Projektion der Rotationskreisbahn, auf der die Endpunkte des VUP liegen müssen, keinen Kreis darstellt, ist eine direkte Berechnung mit Winkelfunktionen nicht ohne weiteres möglich.

Zur Berechnung des Winkels ist deshalb ein iterativer Ansatz sinnvoll und effizient:

Abb. 6.6 zeigt die Vorgehensweise. Für die Winkel 0^o und 180^o berechnet man die Bildschirmpositionen, die den VUP-Vektoren für diesen Winkel zugeordnet sind. Untersucht man dann, auf welcher Seite der Geraden durch die beiden VUP-Positionen die Mausposition liegt, kann man bestimmen, ob der gesuchte Winkel zwischen 0^o und 180^o oder zwischen 180^o und $360^o (= 0^o)$ liegt.

Danach kann man einen rekursiven Algorithmus beginnen, der durch Intervallhalbierung den Bereich, indem der Winkel liegen kann, immer weiter einschränkt. Man unterteilt das gefundene Segment, hier also Suchbereich 1, zwischen 0^o und 180^o durch die Winkelhalbierende, hier also 90^o. Nun untersucht man, in welchem der beiden Teilsegmente $0^o - 90^o$ oder $90^o - 180^0$ die Mausposition liegt. Dazu berechnet man die Position des VUP für die Winkelhalbierende. Liegt nun die Mausposition auf der gleichen Seite der Gerade von Mittelpunkt des Kreises zum VUP der Winkelhalbierenden (hier 90^o) wie der VUP-Wert des einen Randwertes (hier 0^o), so liegt der gesuchte Winkel in diesem Segment, andernfalls innerhalb des anderen. Im Beispiel würde man also finden, daß der gesuchte Winkel zwischen 0^o und 90^o liegt. Man unterteilt das rekursiv gefundene Segment (Suchbereich 2) wieder durch die Winkelhalbierende und führt die entsprechende Untersuchung durch. Bereits nach zehn Iterationschritten hat man den Winkel mit einer Genauigkeit$< 0.5^o$ gefunden. Dies ist häufig schon genauer, als der Winkel aufgrund der Mauspositionierung spezifiziert werden kann.

6.3 Bewertung

Mit den so beschriebenen Algorithmen ist eine vollständige Eingabe aller nötigen Werte interaktiv möglich. Nachteil der entwickelten Methoden ist, daß die Genauigkeit der Eingabe von der Bildschirmauflösung abhängig ist. Ein Raumpunkt kann nicht genauer festgelegt werden, als eine Mauspositionierung möglich ist. Da eine Koordinatenebene senkrecht zur Bildebene stehen kann, ist in diesem Fall der Pixelabstand ihrer Begrenzungslinien sehr klein oder sogar null. Damit ist eine Bewegung in dieser Ebene entweder gar nicht möglich oder sehr ungenau. Um eine genauere Eingabe zu ermöglichen, muß die Darstellung rotiert werden, auch eine Veränderung der Größe des Arbeitsbereiches durch Skalierung kann die Genauigkeit bei der Eingabe erhöhen. Da man am Bildschirm sofort erkennt, wenn die Eingabe zu ungenau wird, ist diese Einschränkung dem Benutzer zuzumuten, da ansonsten die Eingabe recht intuitiv erfolgt. Ergänzend ist zur Mauseingabe noch eine direkte numerische Eingabe dieser Angaben sinnvoll. Damit kann die Ungenauigkeit der Mauseingabe umgangen werden, wenn dies die Anwendung erfordert. Außerdem kann in bestimmten Fällen, in denen die exakten Koordinaten eines Punktes bekannt sind, die numerische Eingabe schneller und einfacher sein.

Kapitel 7

Das Programm VIDES

Aufbauend auf den bisher dargestellten Routinen und Algorithmen wurde ein Programmpaket entwickelt, das die beschriebenen Methoden implementiert und verwendet. Die darin enthaltenen Routinen wurden so konzipiert, daß sie problemlos in eigene Programme eingebunden werden können.

7.1 Zielsetzung des Programmes

Das Programm VIDES ermöglicht die interaktive, graphische Festlegung von Bewegungsabläufen. Aus den Angaben zur Definiton der Bewegungsabläufe berechnet VIDES die Interpolationsdaten, die zur Erzeugung von Animationssequenzen benötigt werden. Dazu werden vom Benutzer Orts-, Orientierungs-, Geschwindigkeits- und Zoomwerte eingegeben, die zur Berechnung der Interpolationswerte herangezogen werden. Es wurde versucht, die Kurven- und Orientierungsbeschreibung so zu gestalten, daß dem Benutzer bei der Erstellung seiner Raumkurve keine wesentlichen Beschränkungen auferlegt werden.

Die berechneten Interpolationsdaten können vielseitig weiter verwendet werden, die Filmgestaltung ist nur ein Einsatzbereich des Programmes.

Bei der Entwicklung von VIDES wurde besonders auf einfache und intuitive Benutzung geachtet. Das Programm ist so gestaltet, daß sämtliche Aufgaben nur durch Mausbenutzung erledigt werden können. Die wichtigsten Editiermöglichkeiten sind direkt im Hauptfenster zu erreichen, untergeordnete Bedienelemente sind in eigenen Fenstern untergebracht. Diese Fenster können dennoch stets geöffnet bleiben, sämtliche Eingaben in einem Fenster werden in den anderen Fenstern entsprechend übernommen. Durch diese hierarchische Aufteilung der Bedienelemente ist ein effizientes Arbeiten mit VIDES möglich.

7.2 Umfeld und Aufbau des Programmes

VIDES wurde aufbauend auf den graphischen und interaktiven Möglichkeiten, die durch X-Windows und das Motif Widget Set zur Verfügung stehen, entwickelt. Dies ermöglicht einerseits weitgehende Kompatibilität über verschiedenste Hardwareplattformen hinaus,

andererseits bietet dieses Konzept dem Benutzer eine einheitliche Bedienoberfläche und einfache, meist bekannte Strukturen bei der Benutzung der Programme.

Aufgrund der Struktur von Programmen, die diese graphischen Benutzeroberflächen verwenden, ist es daher notwendig, daß auch Anwenderprogramme, die diese Routinensammlung einbinden, unter Verwendung von X-Windows entwickelt werden. Nur so ist zu gewährleisten, daß die Struktur der Ereignis- oder Event-gesteuerten Programmierung sinnvoll umgesetzt wird und das Hauptprogramm mit den vorliegenden Programmroutinen gleichzeitig und interaktiv bedient werden kann.

Als Programmiersprache der Routinensammlung dient die Sprache „C“, die bei der Programmierung unter Unix und X-Windows weit verbreitet ist, da sowohl Unix als auch X-Windows größtenteils in „C“ entwickelt wurden.

Das Programm besteht aus insgesamt zwölf Modulen, die sich grob in zwei Klassen einteilen lassen. Die Module können getrennt übersetzt werden, einzelne Module lassen sich eventuell auch anderweitig weiterverwenden. Deshalb hier eine kurze Übersicht über die Module:

- Die erste Klasse der Module enthält die in den vorangegangenen Kapiteln beschriebenen Algorithmen, die zur Projektion, Interpolation und Maussteuerung dienen. Es sind die Module `projekt.c` (Berechnung einer Projektion), `spline.c` (Kochanek-Bartels Splines), `quaternion.c` (Quaternionenrechnung) , `time.c` (Zeitkurvenparametrisierung, eigentlicher Interpolationsvorgang) und `mouse.c` (Maussteuerung im 3D-Raum).

- Die andere Gruppe der Module enthält vor allem Funktionen, die zur Generierung der X-Windows Oberfläche dienen und Prozeduren enthalten, die auf die entsprechende Eingaben des Benutzers reagieren. Die Module heißen `vides.c` (Initialisierung, Benutzerinterface), `menu.c` (Aufbau des Hauptfensters), `draw.c` (Zeichenroutinen), `color.c` (Farbauswahlfenster), `k_b_edit.c` (Kurvenparametereingabe und numerische Eingabe) und `io.c` (Laden/Speichern).

Im Anhang B werden diese Module sowie die zugrundeliegenden Datenstrukturen näher erläutert.

7.3 Bedienmöglichkeiten

In einem Bewegungsraum, der auf dem Bildschirm dargestellt wird, kann der Benutzer interaktiv die Festlegung seiner Bewegungsabläufe vornehmen. Dieser Bewegungsraum, ein Würfel im dreidimensionalen Raum, kann rotiert und skaliert werden, so daß man aus verschiedenen Perspektiven die Festlegungen betrachten kann.

Der Bildschirm von VIDES ist in verschiedene Bereiche aufgeteilt. Neben dem Fenster, das bisher getroffene Festlegungen graphisch anzeigt, sind einzelne Bedienbereiche angeordnet, die eine direkte Manipulation der Eingabedaten ermöglichen. Diesen Bedienbereichen sind einzelne Funktionsgruppen zugeordnet:

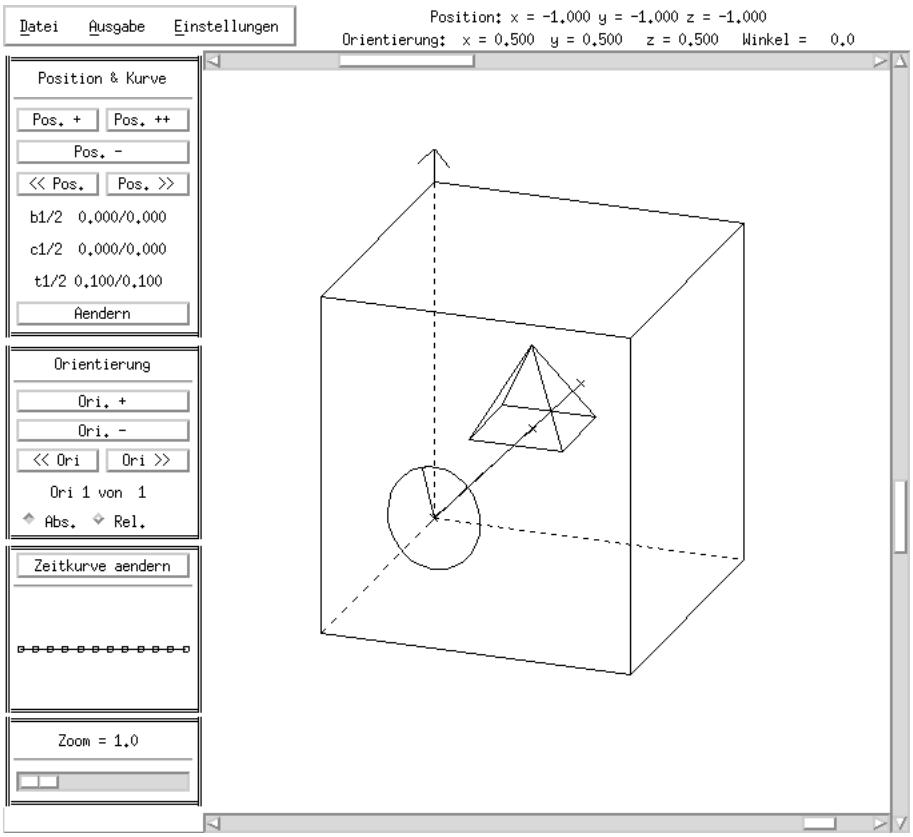


Abbildung 7.1: VIDES: Ansicht nach dem Aufruf des Programmes

- Erstellen und Verändern einer Raumkurve:
  Zu Anfang wird eine Raumkurve mit zwei Stützpunkten vorgegeben, die passend erweitert und angepaßt werden kann. Anfangs ist der Startpunkt dieser Kurve der aktuelle Punkt, auf den sich sämtliche Eingaben beziehen. Operationen wie das Löschen und Einfügen von Punkten betreffen dann diesen Punkt bzw. das Kurvensegment, das in diesem Punkt beginnt. Durch Anwahl mit der Maus kann man den aktuellen Punkt neu setzen, ebenso besteht aber auch die Möglichkeit, den aktuellen Punkt entlang der Kurve zu verschieben. Es sind Möglichkeiten vorgesehen, Stützpunkte einzufügen oder zu löschen. Wird ein Punkt hinzugefügt, plaziert VIDES diesen Punkt in der Mitte zwischen dem aktuellen Punkt und dessen Nachfolger. Wahlweise gibt es auch die Möglichkeit, mehrere Punkte gleichzeitig in äquidistanten Abständen hinzuzufügen. Beim Löschen wird der aktuelle Punkt entfernt, und sein Vorgänger wird zum neuen aktuellen Punkt.
  Mit der Maus und den in Kapitel 6 beschriebenen Verfahren lassen sich Stützpunkte im dreidimensionalen Raum positionieren und verschieben. Damit kann man die Stützpunkte für den Raumkurvenverlauf festlegen und den Anforderungen entsprechend anpassen. Bei jeder Stützpunktänderung wird die Raumkurve dynamisch mitverändert und angezeigt.

- Anpassen der Kurvenparameter:
  Des weiteren ermöglicht das Programm, den Kurvenverlauf zwischen den Punkten interaktiv zu verändern. Dazu kann man mit Schiebereglern die Parameter Tension, Continuity und Bias anpassen. Da häufig bestimmte Bewegungssituationen wiederkommen, die eine ganz bestimmte Wahl der Kurvenparameter erfordern, sind zehn Standardparameter vorgegeben, die Stützpunkten zugeordnet werden können. Auch diese Parameter können angepaßt werden, es besteht damit die Möglichkeit, den Kurvenverlauf lokal oder global zu beeinflussen.

- Festlegung der Orientierungen:
  Die Orientierungsangaben lassen sich ebenso interaktiv spezifizieren. Die gleichen Mausoperationen ermöglichen die Verschiebung des Endpunktes des VPN-Vektors, der die Blickrichtung festlegt. Den Endpunkt des VPN kann man durch Angabe eines festen Raumpunktes absolut in Weltkoordinaten festlegen, man kann ihn aber auch relativ bezogen auf die momentane Position des Stützpunktes definieren. Dies wirkt sich vor allem aus, wenn man den Stützpunkt verschiebt, bei absoluter Festlegung bleibt der Endpunkt des VPN konstant, bei relativer Festlegung wird er entsprechend mitbewegt.
  Auch der VUP-Vektor läßt sich mit der Maus und den in Kapitel 6 vorgestellten Möglichkeiten interaktiv festlegen.
  Einer einzelnen Position können mehrere Orientierungen zugeordnet werden, damit man die Möglichkeit hat, die Kamerarichtung bei konstantem Standort zu ändern. Um weitere Orientierungen einfügen und löschen zu können, sind entsprechende Bedienelemente vorgesehen. Von den Orientierungen, die zum aktuellen Punkt gehören, kann eine als aktuelle Orientierung festgelegt und dann interaktiv am Bildschirm ausgerichtet und angepaßt werden.
  Besonders bei Orientierungen erweist sich die direkte, numerische Eingabe der Orientierungswerte als nützlich. Orientierungen beziehen sich meist auf bestimmte Punkte, die durch das Objekt oder die Szene fest vorgegeben sind. Kennt man diese Punkte, so kann man durch numerische Angabe die Orientierung schneller auf diesen Punkt ausrichten. Deshalb ist zusätzlich zur graphisch-interaktiven Eingabe von Orts- und Orientierungsinformation die numerische Eingabe der Zahlenwerte möglich.

- Spezifikation der Zeitkurve:
  Jeder Orts- und Orientierungsinformation ist eine eigene Zeitkurve zugeordnet, die die Bewegungsgeschwindigkeit entlang dieser Kurve festlegt. Auch diese Kurve wird in einem graphischen Ausgabefenster angezeigt. Sie wird durch zehn Stützpunkte festgelegt; mit der Maus kann man diese Stützpunkte verändern und damit den Zeitkurvenverlauf anpassen.

- Festlegung des Zoomfaktors:
  Als Zoominformation ist jeder Orientierungsangabe ein einzelner skalarer Wert zugeordnet; zwischen den skalaren Werten zweier Orientierungsbeschreibungen wird durch das Programm entsprechend den Zeitkurven interpoliert.

- Weitere Bedienelemente:
  Das Abspeichern und Laden der erzeugten Kurveninformation ist ebenso Bestandteil des VIDES Systems wie die Anpassung der Programmoberfläche an eigene Be-

nutzerwünsche. Dies betrifft die Farbauswahl genauso wie die Elemente, die im Ausgabefenster dargestellt werden.

7.4 Datenübergabe

Die Übergabe der erzeugten Interpolationsdaten kann auf zwei verschiedene Weisen erfolgen. Die Daten können direkt im ASCII Format in eine Datei geschrieben werden, die später von anderen Programmen eingelesen werden kann. VIDES kann die Daten aber auch über eine vorgegebene Datenstruktur direkt an ein aufrufendes Programm weitergeben, wenn dies dort vorgesehen ist.

An VIDES müssen beim Aufruf mehrere Parameter übergeben werden:

- Je nach Anforderung können die Bewegungsräume unterschiedlich groß sein. VIDES hingegen hat die Ausmaße seines Bewegungsraumes fest vorgegeben. Damit die Zahl der erzeugten Stützstellen jedoch dem Anwendungszweck angepaßt ist, müssen die Grundgeschwindigkeiten vorgegeben werden, mit denen die Interpolation erfolgen soll.
- Damit sich der Benutzer im Bewegungsraum besser zurechtfindet, können Objekte in diesem Raum dargestellt werden. Da diese Objekte vom Anwendungsfall abhängig sind, können diese in einer vorgegebenen Struktur ebenfalls an VIDES übergeben werden. Diese Objekte werden nur schematisch im Drahtgittermodell dargestellt, um die hohe Interaktionsgeschwindigkeit zu erhalten.
- Zur direkten Übergabe von Interpolationsdaten kann ein Element der Datenstruktur, die von VIDES dafür vordefiniert ist, an VIDES übergeben werden.
- Zusätzlich können VIDES noch zwei Funktionen mitgeteilt werden, die aufgerufen werden, wenn neue Interpolationsdaten vorliegen bzw. wenn VIDES beendet wird. Dadurch kann VIDES direkt eine Weiterverabeitung der Interpolationsdaten einleiten.

VIDES übergibt für jede einzelne Interpolationsstelle die kompletten Daten zur Berechnung einer Projektion, also VRP, VPN, VUP und den Zoomfaktor. Die Abstände zwischen den Interpolationswerten sind an die gewünschte Geschwindigkeit angepaßt. Mit diesen Daten kann man die einzelnen Bilder berechnen und dann mit einer festen Abspielrate ausgeben. Da die Berechnung der einzelnen Bilder meist sehr viel mehr Zeit in Anspruch nimmt, als die einzelne Ansicht im Film gezeigt wird, muß man die Bilder zwischenspeichern und dann zusammen wiedergeben, z.B. auf einem Videoband. Als ganzes betrachtet ergibt sich aus diesen Bildern die Animationssequenz.

7.5 Beispiel: Entwurf einer Kamerabahn

Es werden hier jetzt exemplarisch die einzelnen Schritte gezeigt, die bei der Planung einer Filmszene, die einen Flug um eine Pyramide darstellt, durchgeführt werden. Dabei soll

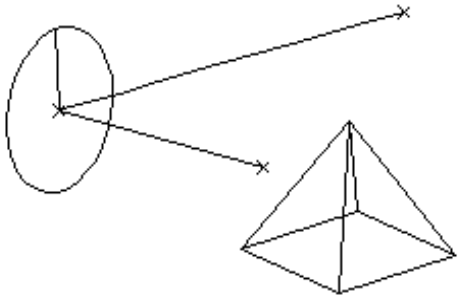


Abbildung 7.2: Beginn der Kurvenfestlegung: Ein Kurvensegment vorhanden

der Flug u. a. an einer Stelle angehalten werden, die Kamera soll entlang der Pyramide geschwenkt werden, bevor sie ihre Bewegung wieder aufnimmt.

Zu Beginn hat man eine Kurve vorliegen, die aus einem Anfangs- und einem Endpunkt besteht und deren vorgegebene Orientierung für den aktuellen Punkt (Anfangspunkt) dargestellt wird. (Abb. 7.2).

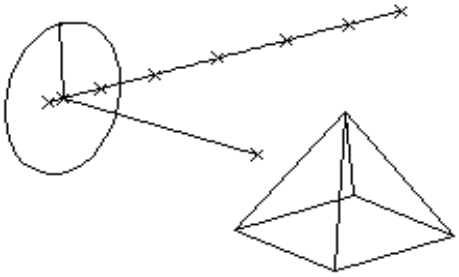


Abbildung 7.3: Einfügen von Punkten

In dieses Startsegment fügt man wie in Abb. 7.3 die benötigte Anzahl von Stützpunkten hinzu. Man kann aber auch später noch beliebig Punkte hinzufügen und löschen.

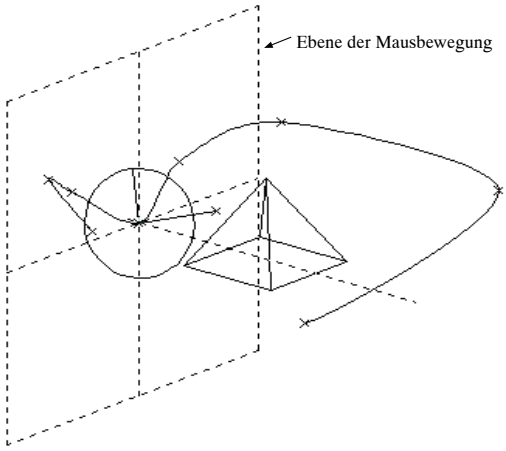


Abbildung 7.4: Positionieren der Stützstellen

Diese Punkte müssen dann mit Hilfe der Maus entsprechend plaziert werden. Dazu greift man die Punkte mit der Maus und verschiebt sie in den drei Koordinatenebenen an die gewünschten Positionen (Abb. 7.4).

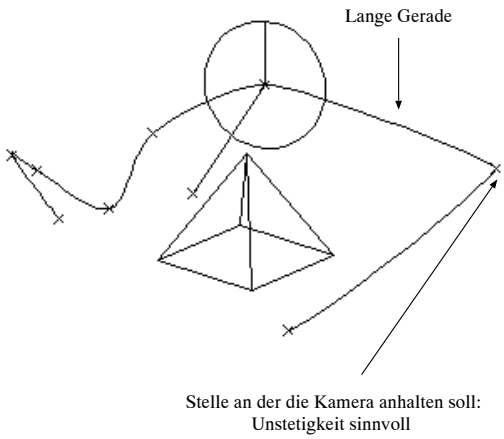


Abbildung 7.5: Anpassung der Kurvenparameter

Mit Hilfe der Kurvenparameter kann man nun den Verlauf der Kurve dem erwünschten Bewegungsverlauf anpassen. Im Beispiel in Abb. 7.5 soll der Kurvenverlauf hinter der

Pyramide möglichst geradlinig sein, an der rechten Ecke soll die Kamera stehen bleiben und einmal entlang der Pyramide schwenken. Deshalb ist hier eine Unstetigkeit der Bewegungsrichtung sinnvoll, da die Bewegung unterbrochen wird.

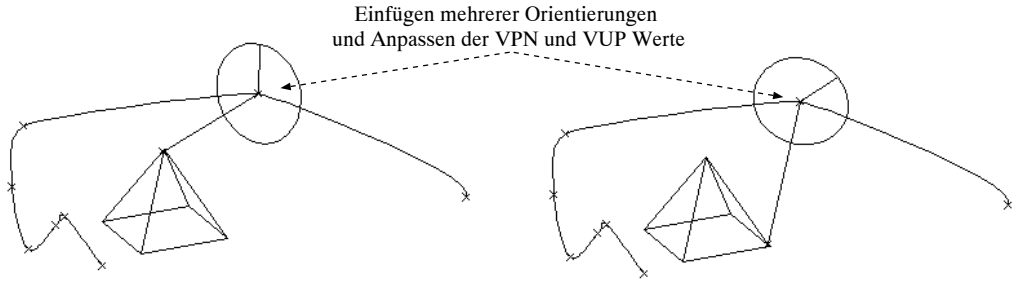


Abbildung 7.6: Anpassung der Orientierungen

Es werden jetzt sämtliche Orientierungen den gewünschten Blickrichtungen angepaßt (Abb. 7.6). Da der Blickpunkt die Spitze der Pyramide sein soll, deren Koordinaten aber bekannt sind, ist die numerische Eingabe sinnvoller. Für die Ecke, an der die Kamera geschwenkt werden soll, werden mehrere Orientierungen eingefügt. Diese werden dann auf die gewünschten Ausrichtungen positioniert.

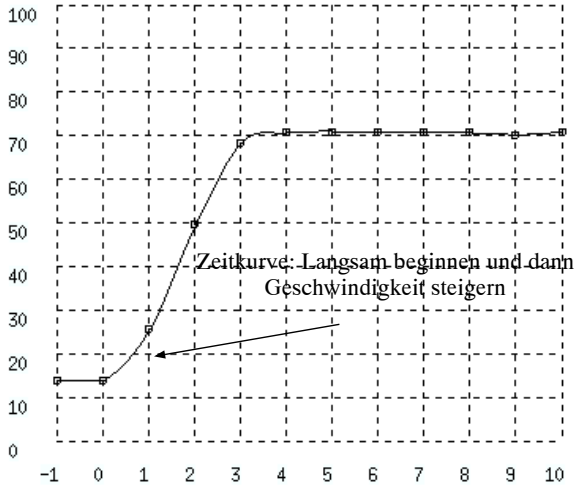


Abbildung 7.7: Anpassung der Zeitkurve

Danach kann die Zeitkurve für die einzelnen Abschnitte angepaßt werden. Da hier die

Kamera, nachdem sie geschwenkt wurde, ihre Bewegung wieder aufnimmt, sollte die Bewegungsgeschwindigkeit dort langsam wieder beginnen und dann auf den gewünschten Endwert gesteigert werden (Abb 7.7).

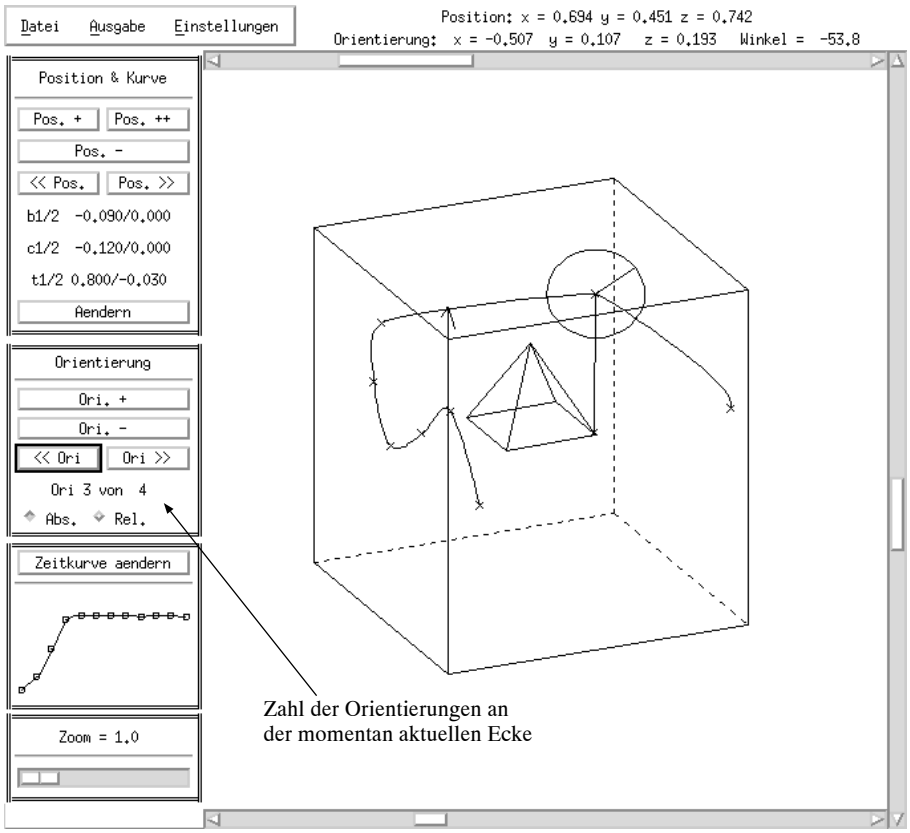


Abbildung 7.8: VIDES: Der Hauptbildschirm mit Kurvenfestlegung

Ein Überblick über die gesamte nun festgelegte Bewegungskurve ist Abb. 7.8 zu entnehmen. In den einzelnen Fenstern kann man z.B. die Anzahl der Orientierungen der aktuellen Stützstelle oder auch die Position des momentanen Stützpunktes ablesen. Der momentane Verlauf der Kurve kann durch Drehung bzw. Kippung des Darstellungraumes mit den Schiebereglern am Rand überprüft werden.

Nach der kompletten Festlegung kann man VIDES die Interpolationsdaten berechnen lassen und den Kurvenverlauf abspeichern, um ihn für spätere Korrekturen zur Verfügung zu haben.

Kapitel 8

Zusammenfassung und Ausblick

Computergraphik gewinnt in der Auswertung wissenschaftlicher Forschungsergebnisse eine immer bedeutendere Rolle. Aber auch im kommerziellen und im privaten Bereich findet die Computergraphik eine wachsende Verbreitung. Die weiter fortschreitende Leistung moderner Rechnersysteme und der breitere Einsatz von Videoaufzeichnungssystemen hat dem Gebiet der wissenschaftlich-technischen Visualisierung breitere Anwendungsfelder geöffnet.

Das in dieser Arbeit vorgestellte System zur Generierung von Kamerapositionen für Animationssequenzen trägt dem gewachsenen Bedarf an effizienter Software für diesen Bereich Rechnung.

Das Kapitel über die Projektionen hat die Grundlagen gelegt, die benötigt werden, um die vorgestellten Algorithmen verstehen zu können.

Mit den Kurvenbeschreibungen konnte eine vielseitig verwendbare Möglichkeit eingeführt werden, Kurvenplanungen durchzuführen. Da die Möglichkeiten der Kurvenbeschreibung schon recht gut erforscht sind, bietet die vorgestellte Kurvenklasse vielfältige Möglichkeiten, die Kurven jeweils an einen speziellen Anwendungsfall anzupassen. Es wurde deshalb bei der Implementierung besonders auf eine schnelle und effiziente Neuberechnung des Kurvenverlaufes geachtet, um eine interaktive Veränderung der Kurven auch im Dreidimensionalen zu ermöglichen. Eine hohe Geschwindigkeit wurde vor allem dadurch erreicht, daß die Kurven zur Bildschirmdarstellung mit einer festen Zahl von Stützstellen bereits weitgehend vorberechnet im Speicher gehalten werden und nur an die verschobenen Stützpunkte angepaßt werden müssen. Die so erzielte Rechengeschwindigkeit reicht auch auf stark belasteten Systemen aus, eine ruckfreie Veränderung der Kurven zu ermöglichen.

Die Beschreibung von Orientierungen dagegen ist, besonders in der Computeranimation, bisher noch nicht erschöpfend behandelt worden. Einerseits war die Orientierungsinterpolation in der unbewegten Graphik bisher selten erforderlich, andererseits ist es schwerer, die Orientierungsinterpolation anschaulich zu verstehen und darzustellen. Die Ansätze der Orientierungsbeschreibung mit Hilfe von Quaternionen sind in der Raumfahrt und in der Robotertechnik eingeführt worden und haben sich dort gut bewährt. Die Übertragung auf den Graphikbereich hat die Möglichkeit geschaffen, mit der sphärischen linearen Interpolation eine Grundlage für die Orientierungsinterpolation in der Animation zu legen. Der Übergang zu stetiger Orientierungsinterpolation berücksichtigt die Anforderungen im Graphikbereich und stellt damit eine für die Verwendung in dieser Arbeit

voll ausreichende Basis der Orientierungsinterpolation dar. Dennoch ist dieser Bereich noch ein interessantes Forschungsgebiet. Besonders nützlich wäre eine Erweiterung der Interpolationsbeschreibung, um eine weiterreichende, individuelle Anpassung der Interpolationsverläufe zu ermöglichen. Dazu könnte man zusätzliche Parameter einführen, die, wie bei den Kochanek-Bartels Kurven, den Interpolationsweg verändern.

Auch die gewählte Möglichkeit der Zeitsteuerung hat sich als für den Anwendungszweck voll ausreichend herausgestellt. Die individuelle Festlegung der Bewegungsgeschwindigkeit für jeden Kurvenabschnitt ermöglicht es beispielsweise, wichtige Elemente während der Animation durch langsame Bewegungsgeschwindigkeiten detailiert zeigen zu können, ohne unwichtige Details in der gleichen Ausführlichkeit in die Animation aufnehmen zu müssen.

Speziell für den hier vorgestellten Anwendungszweck wurde die Maussteuerung so konzipiert, daß sie intuitiv zu bedienen und schnell und effektiv zu erlernen ist. Da die Algorithmen auch auf andere Darstellungen direkt übertragbar sind, kann man darauf aufbauend auch Verfahren zur Skalierung, Rotation und Scherung von Gegenständen entwickeln, die hier nicht benötigt wurden.

Das unter Verwendung dieser Methoden entwickelte Programm ermöglicht damit eine gute Planung von Animationssequenzen. Mit den vorhandenen Möglichkeiten lassen sich einzelne Bewegungsabläufe effizient festlegen und jederzeit wieder ändern und neuen Anforderungen anpassen. Die Kapselung des Programmoduls nach außen ermöglicht es, das Modul aus einer Anwendung mehrfach aufzurufen und damit mehrere Kurvenplanungen gleichzeitig in einzelnen Fenstern durchzuführen.

Mit VIDES ist eine Lücke im Softwareangebot zur Herstellung von Animationssequenzen geschlossen worden. Die im Rahmen dieser Arbeit geschaffene Grundlage bietet vielfältige Erweiterungsmöglichkeiten:

Wünschenswert wäre als Erweiterung vor allem, fertige Schnittstellenmodule zu vorhandenen Visualierungssystemen wie AVS zur Verfügung zu haben. Damit könnte ein noch intensiverer Datenaustausch möglich werden und die Entwurfsphase von Animationen dadurch nochmals verkürzt werden. Da die berechneten Daten anderen Programmen direkt zur Verfügung gestellt werden können, ist es relativ einfach, entsprechende Anpassungsprogramme zu entwickeln.

Bisher wurde nur die Bewegung eines einzelnen Objektes berücksichtigt. Wenn man die Bewegungsplanung mehrerer Objekte in einem Modul durchführen könnte, hätte man damit die Möglichkeit, z.B. neben der Kamerabahnplanung auch die Bewegungsplanung von anderen Objekten (z.B. der Scheinwerfer, die die Szene ausleuchten), miteinander verknüpft durchzuführen. Dies erfordert dann Möglichkeiten der Synchronisation zwischen den einzelnen Bewegungskurven; dazu wäre eine Erweiterung der Zeitkurvendarstellung notwendig.

Das Verständnis der Interpolationsbahn bei der Orientierungsinterpolation erweist sich als deutlich schwieriger als das Verständnis der Kurven zur Ortsinterpolation. Bisher gibt es noch keine befriedigenden Methoden, die komplette Orientierunginformation für eine große Zahl von Orientierungen anschaulich und klar auf dem Bildschirm wiederzugeben. Neue Methoden der Orientierungsdarstellung müßten entwickelt und auf ihre Interpretierbarkeit untersucht werden. Eine Erweiterung des Systems VIDES in dieser Richtung würde die Verständlichkeit der selbstgetroffenen Festlegungen erleichtern und damit die Bedienung noch intuitiver gestalten.

Eine Erweiterung der Bedienmöglichkeiten von VIDES wäre ebenfalls ein sinnvoller Schritt, der den Einsatzbereich von VIDES noch erweitern könnte. Zweckmäßig wäre es in diesem Zusammenhang, Bewegungsabläufe in einzelne Teile aufzuteilen und diese als Bausteine modular zu sichern und zu laden. Mit der Zeit könnte man sich damit eine Bibliothek aus einzelnen charakteristischen Bewegungskomponenten anlegen. Um eine Animation zu gestalten, müßte man nur noch einzelne Elemente zusammenfügen und anpassen.

Insgesamt hat sich die Bahnplanung einer Kamera als weitaus komplizierter herausgestellt als es anfangs den Augenschein hatte. In eine Bewegung fließen zahlreiche Faktoren mit ein, damit die Animation eine natürlichen Eindruck macht. Alle Faktoren zu berücksichtigen, war im Rahmen der Entwicklung von VIDES nicht möglich. Durch Beschränkung auf die grundlegenden Anforderungen der zeitgesteuerten Orts- und Orientierungsinterpolation konnte dennoch ein System geschaffen werden, das die ursprünglich gestellten Erwartungen voll erfüllen kann.

Anhang A

Bedienungsanleitung für VIDES

A.1 Aufgabe des Programmoduls

Das Programm VIDES wurde entwickelt, um eine einfache, interaktive Bahnplanung einer Kamera oder anderer zu bewegender Objekte zu ermöglichen. Hauptzielrichtung ist die Generierung von Führungsgrößen zur Bewegung einer virtuellen, also gedachten, Kamera. Mit den von VIDES erzeugten Daten kann dann mit entsprechenden Programmen ein Film generiert werden, bei dem die Kameraposition und Ausrichtung den von VIDES berechneten Größen entsprechen.

Beim Entwerfen einer Raumkurve müssen nur einige markante Punkte auf der Bewegungskurve vorgegeben werden, daraus werden die Zwischenwerte von VIDES erzeugt.

Um mit VIDES arbeiten zu können, sollen daher erst die Angaben, die VIDES zur Festlegung einer Kameraposition benötigt, beschrieben werden:

- Einerseits hat die Kamera eine Position im Raum, von der aus das Bild aufgenommen werden soll. Zur Beschreibung der Position genügt ein einzelner Ortsvektor, der VRP-Vektor (View Reference Point).
- Andererseits besitzt die Kamera eine Orientierung. Dazu wird angegeben, in welche Richtung die Kamera ausgerichtet ist, bzw. auf welches Objekt die Kamera zielt. Zusätzlich muß noch festgelegt werden, in welcher Richtung die Bildhorizontale der Kamera liegt. Dies wird durch Angabe des Kamerarichtungsvektors, auch VPN (View Plane Normal) genannt und der Kameranormalen (senkrecht auf der Bildhorizontalen), auch VUP (View Up) genannt, festgelegt. Da die Kameranormale stets senkrecht auf dem Richtungsvektor stehen soll, ist es ausreichend, sie durch Angabe eines Winkels festzulegen.
- Zusätzlich kann noch eine kameraspezifische Größe festgelegt werden, die den Bildwinkel oder auch den Zoomfaktor festlegt. Darunter kann man sich die Größe des Bildausschnittes vorstellen, bei einer normalen Kamera würde dies der Brennweite des Objektives entsprechen.

Zur kompletten Beschreibung einer Bewegung gehören neben den Zwischenpositionen und Orientierungen auch Angaben über die Bewegungsgeschwindigkeit, damit man eine vollständige Festlegung hat.

Diese Angaben müssen VIDES also zur Verfügung stehen, damit es den Bewegungsablauf für eine Filmszene berechnen kann. VIDES generiert dabei nicht den Film, dies bleibt anderen Programmen überlassen, es legt lediglich die zur Berechnung der einzelnen Bilder notwendigen Daten fest.

Es soll nun zunächst das VIDES Hauptfenster mit seinen Komponenten vorgestellt werden, bevor darauf eingegangen wird, wie in VIDES die Kameradaten interaktiv festgelegt werden können. Anschließend werden die Bedienungsbereiche ausführlich erklärt. Zuletzt folgen noch Hinweise, die man beim Festlegen eines Bewegungsablaufes berücksichtigen sollte.

A.2 Benutzerschnittstellen von VIDES

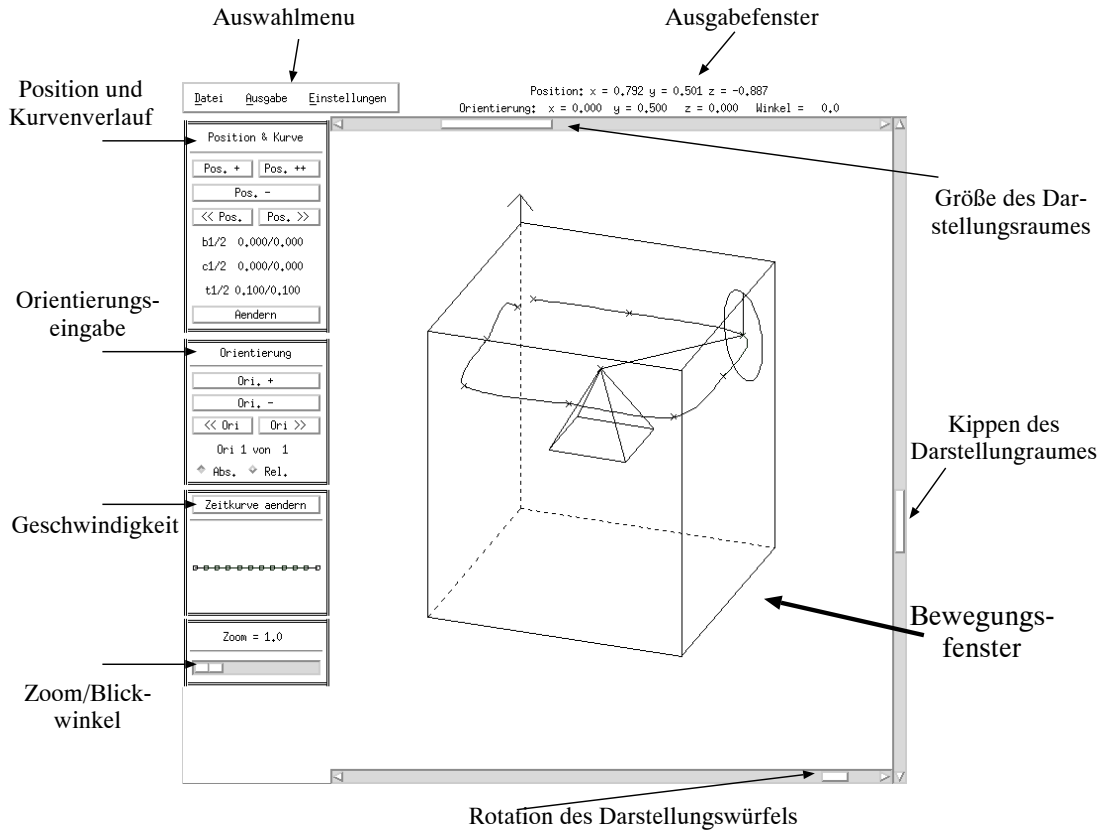


Abbildung A.1: Ansicht des VIDES Hauptfensters

Das VIDES **Hauptfenster** besteht aus mehreren Komponenten, die Abb. A.1 zu entnehmen sind.

Mittelpunkt des Hauptfensters ist das **Bewegungsfenster**, in dem die Kameraplanung vorgenommen werden kann. In diesem Fenster werden die Stützpositionen und Orientierungen mit der Raumkurve angezeigt und können darin interaktiv verändert werden.

Um das Bewegungsfenster sind verschiedene Bedienkomponenten angeordnet:

Im **Auswahlmenu** können über verschiedene Pull Down Menus Aktionen gestartet werden und Einstellungen vorgenommen werden:

- Das `Datei`-Menu enthält Befehle zum Abspeichern und Laden von Kurvenfestlegungen und zum Beenden von VIDES.
- Im `Ausgabe`-Menu startet man die Berechnung des Bewegungsablaufes.
- Im Menu `Einstellungen` kann man VIDES an persönliche Präferenzen anpassen und diese Einstellungen abspeichern und laden.

Neben den Auswahlmenu liegt das **Ausgabefenster**, in dem man laufend über aktuelle Daten von VIDES informiert wird. Normalerweise werden hier Positions- und Orientierungsangaben der aktuellen Kameraeinstellung angezeigt. Treten Bedienungsfehler auf, werden diese ebenfalls in diesem Fenster eingeblendet.

Links vom Bewegungsfenster befinden sich vier verschiedene Bedienungsbereiche, in denen ergänzende Angaben zum Bewegungsablauf eingegeben werden können:

- Das **Positions- und Kurvenfenster** dient zur erweiterten Manipulation der Kamerapositionen und des Bewegungsablaufes zwischen den Stützpunkten.
- Im **Orientierungsfenster** kann die Kameraausrichtung ergänzend beeinflußt werden.
- Das **Geschwindigkeitsfenster** stellt die Kurve der Bewegungsgeschwindigkeit dar und ermöglicht deren Veränderung.
- Im **Zoom-/Blickwinkelfenster** können Angaben zur Kameradarstellung vorgenommen werden.

Um das Bewegungsfenster sind drei **Schiebebalken** angeordnet. Sie ermöglichen die Veränderung des Darstellungsraumes. Der obere Schiebebalken ändert die Größe des Darstellungsraumes, mit dem Unteren kann man diesen rotieren, mit dem rechten Schiebebalken den Darstellungsraum kippen.

A.3 Das Bewegungsfenster

Im Bewegungsfenster wird der gesamte festgelegte Bewegungsablauf dargestellt. Die einzelnen Komponenten, die man Abb. A.2 entnehmen kann, sollen kurz erläutert werden.

Der gesamte Bewegungsablauf ist in einen Darstellungsraum eingebunden. Die Kanten des Darstellungsraumes werden angezeigt, wobei sichtbare Kanten durchgezeichnet sind und unsichtbare Kanten gestrichelt werden. Der Mittelpunkt des Darstellungsraumes ist gleichzeitig der Nullpunkt, die Ausdehnung in jede Richtungsachse ist auf ± 1 begrenzt. Der Darstellungsraum kann durch die oben erwähnten Schiebebalken vergrößert, rotiert und gekippt werden.

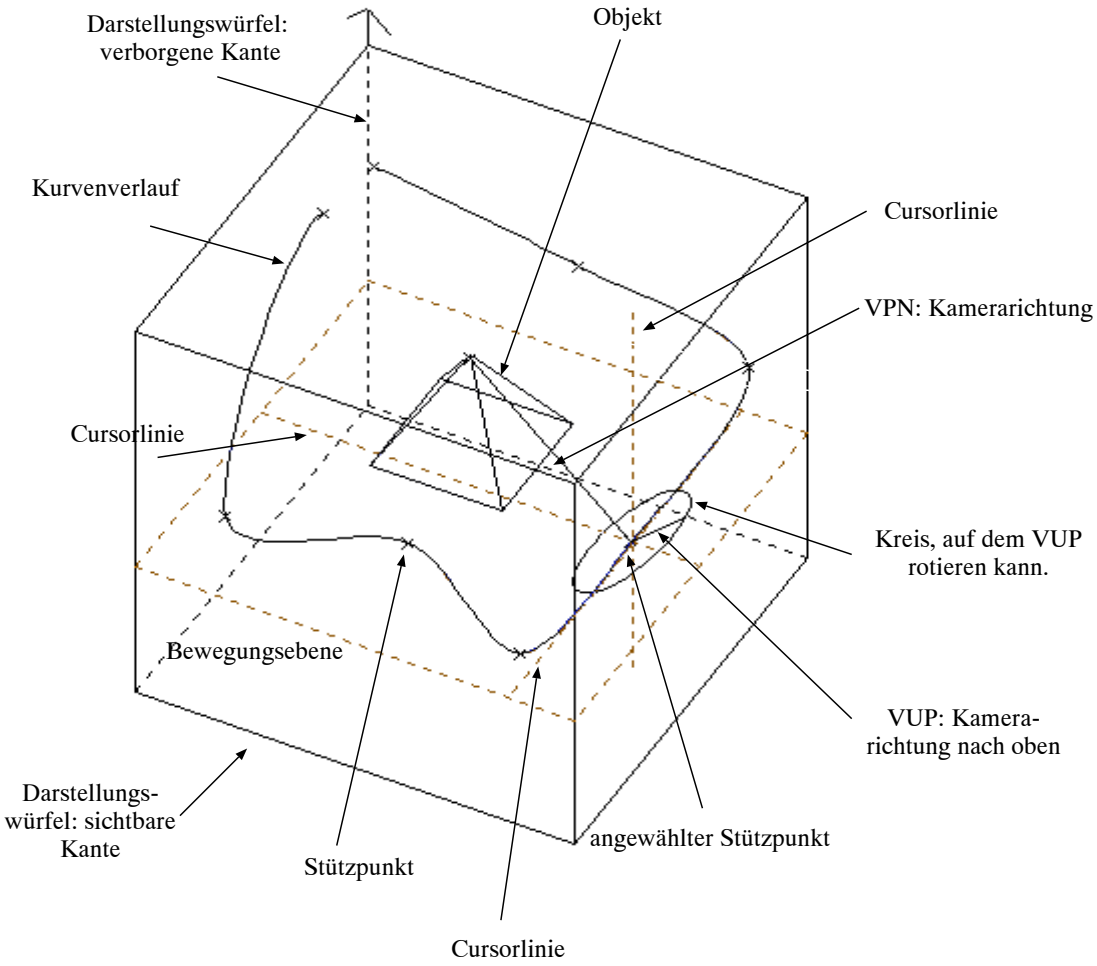


Abbildung A.2: Komponenten im Bewegungsfenster

Im Darstellungraum kann ein Modell des Objektes angegeben werden, die in der Filmsequenz dargestellt werden sollen. In Abb. A.2 ist das Objekt eine Pyramide.

Der Bahnverlauf wird durch eine Kurve dargestellt, wobei die Stützpunkte auf dieser Kurve durch ein kleines Kreuz hervorgehoben werden. Ein bestimmter Stützpunkt kann angewählt werden, der Kurvenverlauf für den Abschnitt, der an dieser Position beginnt, wird durch eine andere Farbe wiedergegeben.

Für die angewählte Position auf der Bewegungskurve wird eine Orientierung der Kamera dargestellt. Dazu wird die Kamerarichtung durch den VPN-Vektor angegeben, ebenso die Kameranormale durch den VUP-Vektor. Der VUP-Vektor kann entlang einer Kreisbahn bewegt werden; der Verlauf dieser Kreisbahn wird ebenfalls angezeigt.

A.3.1 Bewegung der Kameraposition und Orientierung

Die Kameraposition und die Orientierung können im Darstellungsfenster interaktiv festgelegt und verändert werden. Zuerst erfolgt eine kurze Beschreibung der Maussteuerung, mit der die Position und der Endpunkt des VPN-Vektors gesteuert werden:

Da man die Maus nur auf einer zweidimensionalen Ebene bewegen kann, der Darstellungsraum aber in drei Richtungen ausgedehnt ist, kann man nicht alle drei Ortsangaben gleichzeitig festlegen. Vielmehr erfolgt die Mauseingabe jeweils in einer der drei Koordinatenebenen. Drückt man die linke Maustaste, so kann man eine Bewegung der y- und z-Werte bei festgehaltenem x-Wert vornehmen, man kann also Punkte in der Ebene x=const. bewegen. Ebenso kann man durch Drücken der mittleren Taste die x- und z-Koordinaten bei festem y-Wert festlegen, die rechte Maustaste steuert Bewegungen entlang der x- und y-Koordinaten bei festem z-Wert.

Man wählt einen Punkt der Kurve an, indem man den Mauszeiger zu diesem hinbewegt und die entsprechende Maustaste drückt. Die Umrandung der festen Ebene wird daraufhin eingeblendet, ebenso werden die drei Koordinatenachsen im angewählten Punkt angezeigt. Bewegt man nun die Maus bei gedrückter Maustaste, so folgt der Bildpunkt genau der Mausbewegung, die Koordinaten des Raumpunktes werden auf der festen Ebene passend zur Bewegung verändert. Ebenso kann man die Spitze des VPN-Vektors anwählen und bewegen. Will man die Maus auf einer anderen Ebene bewegen, so wählt man den Punkt durch Drücken der entsprechenden Maustaste erneut an und führt die Bewegung durch. Abb. A.2 zeigt Bewegungsebene und Achsenkreuz für eine Bewegung in der y-Ebene an.

Zur Kontrolle werden die Koordinatenangaben des Punktes im Ausgabefenster angezeigt. Sollte eine Bewegung in einer Ansicht des Darstellungsraumes auf einer bestimmten Ebene nicht möglich sein, weil diese Ebene in der Darstellung nur sehr klein sichtbar ist, so erfolgt im Ausgabefenster eine Meldung. Durch Drehen, Kippen und Skalieren des Darstellungsraumes kann man die Ebene entsprechend vergrößern, damit man die gewünschte Bewegung durchführen kann.

Die Spitze des VUP-Vektors, also der Kameranormalen, kann man entlang eines Kreises rotieren. Dazu wählt man die Spitze des VUP-Vektors durch Drücken der linken Maustaste an. Bewegt man dann die Maus bei gedrückter Taste, so folgt der VUP-Vektor so der Mausbewegung, daß der neue VUP-Vektor auf der Verbindungslinie von der Kameraposition zur Mausposition liegt.

Mit ein wenig Übung ist die Mausbedienung leicht zu erlernen; orientiert man sich an den entsprechenden Hilfslinien, so ist die Lage im Raum gut zu erkennen.

A.4 Die Bedienfenster

Ein mit der Maus angewählter Punkt wird zum aktuellen Punkt. Das bedeutet, daß das Kurvensegment, welches in diesem Punkt beginnt, andersfarbig gezeichnet wird. Sämtliche Veränderungen von Kurvenverlaufs-, Orientierungs- oder Zeitangaben beziehen sich auf diesen aktuellen Punkt und das dazugehörige Kurvensegment.

A.4.1 Position und Kurve

Im Fenster „Position und Kurve“ können Einstellungen zum Kurvenverlauf vorgenommen und abgelesen werden:

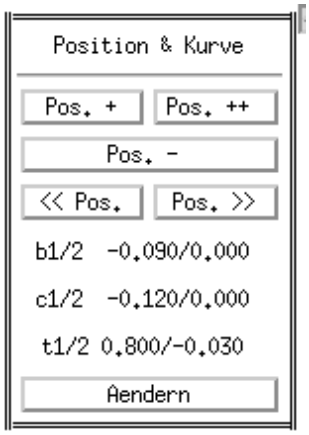


Abbildung A.3: Das Positionsfenster

`Pos.+` `Pos.++`
Mit Hilfe dieser Tasten kann man Punkte auf der Kurve hinzufügen. Drückt man `Pos.+`, so wird in die Kurve ein Punkt eingefügt, der auf der halben Strecke zwischen dem momentan angewählten Punkt und dessen Nachfolger liegt. Der neue Punkt bekommt sämtliche Eigenschaften des momentan angewählten Punktes übertragen. Das heißt, er hat sämtliche Geschwindigkeits- und Orientierungsangaben seines Vorgängers übernommen. Der hinzugefügte Punkt ist der neue aktuelle Punkt.
Auch `Pos.++` fügt Punkte hinzu. In einem kleinen Fenster kann man angeben, wieviele Punkte (1 bis max. 10) eingefügt werden sollen, diese werden dann zwischen dem momentan angewählten Punkt und seinem Nachfolger in gleichen Abständen eingefügt. Es ist jedoch nicht möglich, einen Punkt hinter dem letzten Stützpunkt einzufügen, deshalb erscheint eine Meldung im Ausgabefenster, wenn man dies versucht.

`Pos.-`
Durch Drücken dieses Knopfes wird der aktuelle Punkt gelöscht und dessen Vorgänger wird zum aktuellen Punkt. Es müssen jedoch immer zwei Punkte vorhanden sein.

`<< Pos.` `Pos.>>`
Die beiden Tasten steuern die Wahl des aktuellen Punktes der Bewegungskurve. Außer durch direkte Anwahl mit der Maus kann man mit diesen Tasten einen Stützpunkt auf der Kurve anwählen. Dabei wird durch die Taste `Pos.>>` der nachfolgende Stützpunkt des momentan aktuellen Punktes angewählt, die Taste `<< Pos.` wählt den Stützpunkt vor dem momentan aktuellen Punkt an. Ist Anfang oder Ende der Kurve erreicht, so springt der aktuelle Punkt zum jeweils anderen Ende weiter.

Zahlenwerte `b1/2, c1/2, t1/2`:
Die Zahlenwerte stellen Parameter dar, mit denen auf den Kurvenverlauf Einfluß genommen werden kann (Bias, Continuity, Tension). Die Auswirkung der Parameter ist

ausführlich in Kapitel 3.7 dieser Arbeit beschrieben. Die Kurvenparameter beeinflussen den Kurvenverlauf, wenn dieser durch den dazugehörigen Punkt führt. Es werden die Parameter des angewählten Punktes (`b1 c1 t1`) sowie des nachfolgenden Punktes (`b2 c2 t2`) angezeigt.

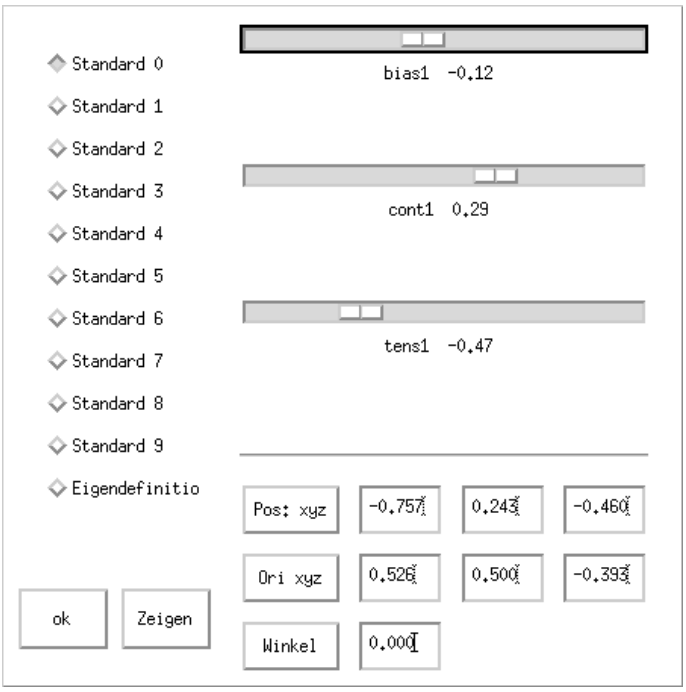


Abbildung A.4: Parameterauswahl

`Aendern`
Durch Drücken dieser Taste öffnet sich das zusätzliche Kurvenparameterfenster, das in Bild A.4 dargestellt wird.
In diesem Fenster kann man einerseits die Kurvenparameter festlegen, andererseits die Position und die Orientierungsangaben des aktuellen Punktes numerisch verändern.

Es sind zehn Standardparametersätze für die Kurvenparameter vorgegeben, zusätzlich kann für jeden Stützpunkt ein eigener Kurvenparametersatz festgelegt werden. Ist in der Knopfleiste auf der linken Seite dieses Fensters ein Knopf `Standard n` der zehn Standardwerte angewählt, so sind dem aktuellen Punkt diese Werte zugeordnet, ist `Eigendef.` angewählt, so sind Kurvenparameter ausschließlich für diesen Punkt festgelegt worden.

Die drei Schieberegler, die neben den Auswahlknöpfen angeordnet sind, zeigen die Werte der angewählten Kurvenparameterfestlegung an. Durch Verschieben eines Reglers können die Werte der aktuellen Kurvenparameter verändert werden. Ist ein Standardparameter angewählt, so beeinflußt dies nicht nur den Kurvenverlauf im aktuellen Stützpunkt sondern aller Punkte, denen dieser jeweilige Standardwert zugeordnet ist. Deshalb werden die geänderten Parameter auch nicht direkt übernommen, erst nach Drücken der Taste `ok` oder `Zeigen` werden diese Werte angezeigt und für den aktuellen Punkt gespeichert. `ok` schließt zusätzlich das Kurvenparameterfenster.

In dem Eingabefenster `Pos:xyz` (Position), `Ori:xyz` (Orientierung) sowie `Winkel` können die Zahlenwerte der Position und der Orientierung des jeweils aktuellen Punktes abgelesen und direkt eingegeben werden.

A.4.2 Orientierung

Abbildung A.5: Das Orientierungsfenster

Zu jeder Kameraposition muß mindestens eine Orientierungsangabe vorgenommen werden, es können aber bis zu zehn Orientierungen zu einer einzelnen Position festgelegt werden. Das ist deshalb sinnvoll, da die Kamera beispielsweise rotiert werden kann, während sie auf einer festen Position steht.

Mit den Knöpfen `Ori.+` und `Ori.-` kann man eine weitere Orientierung hinzuzufügen oder die aktuelle Orientierung löschen.

`Ori k von n`:
Unter diesen Knöpfen wird dargestellt, wieviele Orientierungen dem jeweiligen Punkt zugeordnet sind und welche davon gerade angezeigt wird. Dabei gibt die erste Zahl k die momentan aktuelle Orientierung, die zweite Zahl n die Anzahl der festgelegten Orientierungen an.

Die beiden Tasten `<< Ori` `Ori >>` wählen die jeweils vorherige bzw. nachfolgende Orientierung an, wenn mehr als eine Orientierung festgelegt ist.

Der VPN einer Orientierung kann fest auf einen Raumpunkt gerichtet sein oder durch Richtungsangabe relativ zum Stützpunkt festgelegt werden. Für die jeweils aktuelle Orientierung wird dies durch die beiden Knöpfe `Abs.` oder `Rel.` festgelegt. Dies hat vor allem Auswirkung, wenn der Stützpunkt der Orientierung verschoben wird. Ist die Orientierung als absolut definiert, so bleibt der Endpunkt des VPN beim Verschieben des Stützpunktes gleich, die Orientierungsrichtung ändert sich.
Ist die Orientierung hingegen relativ definiert so bleibt die Richtung der Orientierung beim Verschieben des Stützpunktes konstant, dagegen ändert sich der Punkt, auf den die Kamera gerichtet ist.

A.4.3 Zeitkurve ändern

Jedem Bewegungsabschnitt ist eine eigene Zeitkurve zugeordnet. Durch die Zeitkurve wird die Bewegungsgeschwindigkeit festgelegt, mit der sich die Kamera zwischen den einzelnen Festlegungen bewegt, egal ob es sich um eine Verschiebung der Kameraposition oder um ein Schwenken der Kamera handelt.

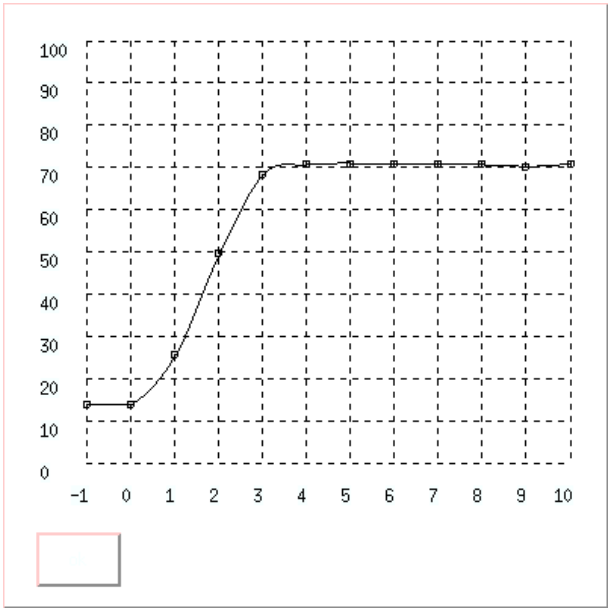


Abbildung A.6: Festlegung der Zeitkurve

Im Bedienfenster `Zeitkurve ändern` wird der Zeitkurvenabschnitt angezeigt, der dem aktuellen Punkt und der aktuellen Orientierung zugeordnet ist. Drückt man die Taste `Zeitkurve aendern`, so wird ein größeres Fenster geöffnet, in dem der Zeitkurvenabschnitt auch manipuliert werden kann (Abb. A.6). Die Geschwindigkeit kann Werte zwischen 1 und 100 annehmen. Um die Kurve in diesem Abschnitt festzulegen, kann der Kurvenwert für zehn Punkte angegeben werden, die Kurve verläuft durch diese Punkte. Wählt man einen dieser Punkte, die durch kleine Quadrate gekennzeichnet sind, mit der Maus an, so kann man den Zeitwert nach oben oder unten bewegen, die Zeitkurve ändert sich entsprechend. Zusätzlich zu den zehn Stützwerten ist noch der Geschwindigkeitswert am Ende des vorherigen Abschnittes und zu Beginn des nachfolgenden Abschnittes angezeigt, dieser kann aber nicht verändert werden.

Da jeder Position und jeder Orientierung ein eigener Zeitkurvenabschnitt zugeordnet ist, wird der Zeitkurvenabschnitt der jeweils aktuellen Position und Orientierung angezeigt. Wählt man eine neue Position und Orientierung an, so wird der zugehörige Zeitkurvenabschnitt dargestellt und kann verändert werden.

A.4.4 Zoom

Der Zoomfaktor kann durch einfaches Verändern des Schiebereglers in diesem Fenster angepaßt werden, jeder einzelnen Orientierung ist ein eigener Zoomfaktor zugeordnet.

A.5 Das Auswahlmenu

Menu Datei Menu Ausgabe Menu Einstellungen

Abbildung A.7: Die Pull Down Menus

Der Menupunkt `Datei` enthält Befehle, um eine Kurvenfestlegung zu laden und zu speichern und um VIDES zu beenden. Durch Auswahl mit der Maus oder mit der Tastenkombination `ALT-d` wird das Menu geöffnet. Wählt man dann `Lade Kurve` bzw. `Speicher Kurve` an (oder drückt `ALT-l` bzw. `Alt-s`), so bekommt man ein Dateiauswahlfenster (Abb. A.8) angezeigt. Dort kann man den Speicherpfad und den Dateinamen festlegen und danach einen Bewegungsablauf laden oder speichern. Normalerweise enden die Kurvendateien auf `.vid` für VIDES; andere Endungen kann man in dem Auswahlfenster jedoch festlegen.

Mit `quit` (oder `ALT-F4`) kann man VIDES beenden und zum aufrufenden Programm zurückkehren.

Im Menufenster `Ausgabe` kann man die Datenberechnung einleiten, mit deren Hilfe der Bewegungsablauf dargestellt werden soll. Es ist immer möglich, die berechneten Werte in eine Datei zu schreiben. Mit der Auswahl von `In Datei ausgeben` (oder `ALT-d`) werden die Werte in eine Datei geschrieben, den Dateinamen kann man vorher in einem Dateiauswahlfenster festlegen.
Wählt man hingegegen `Direkt uebergeben` (oder `ALT-u`) an, so werden die Daten direkt an das Programm, von dem VIDES aus aufgerufen wurde, übergeben. Dieses Programm kann die Daten dann weiterverarbeiten. Ob diese Ausgabe möglich ist, und wie die Daten dann weiterverarbeitet werden, hängt von der Implementierung des aufrufenden Programmes ab.

Im Menufenster `Einstellungen` kann man VIDES an eigene Voreinstellungen anpassen:

Man kann die Farben der Elemente im Darstellungsfenster verändern. Der Menupunkt `Farbe` (`ALT-f`) öffnet das Farbauswahlfenster (Abb.A.9). Nach Anwahl eines Elementes

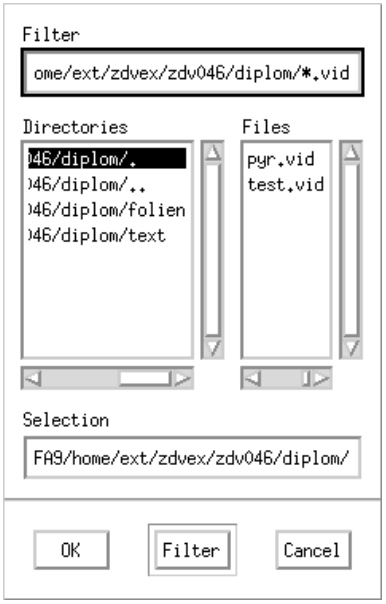


Abbildung A.8: Dateiauswahl

wie Hintergrund oder Kurvenverlauf kann man mit drei Schiebereglern den Rot-, Grün- und Blauanteil der jeweiligen Farbe verändern.

Der Menupunkt `Anzeige` (`ALT-z`) öffnet das Fenster, indem eine Auswahl der darzustellenden Elemente wie Darstellungswürfel oder VPN festgelegt werden kann. Durch Drücken der zu den einzelnen Elementen gehörenden Knöpfe wird die Darstellung des jeweiligen Elementes ein- bzw. ausgeschaltet.

Diese Einstellungen, sowie die momentanen Darstellungswinkel kann man abspeichern und wieder laden. Dazu dienen die Menupunkte `Lade Setup` (`ALT-t`) sowie `Speicher Setup` (`Alt-p`).

A.6 Hinweise zur Festlegung von Orts- und Orientierungsinformationen

A.6.1 Die voreingestellten Kurvenparameter

Die vorgegebenen Werte der Kurvenparameter Bias, Tension und Continuity sollen nun erläutert werden. Die Parameter, die bei einer Festlegung nicht erwähnt sind, können als Null angenommen werden. Für spezielle Anwendungen sind besondere Parameter sinnvoll.

Standard 0: Alle Parameterwerte gleich null; stetige, normal glatte Kurven

Standard 1: Tension = 0.3: Dadurch wird der Kurvenverlauf straffer gespannt, die Kurve verläuft zwischen den beiden Stützpunkten relativ gerade.

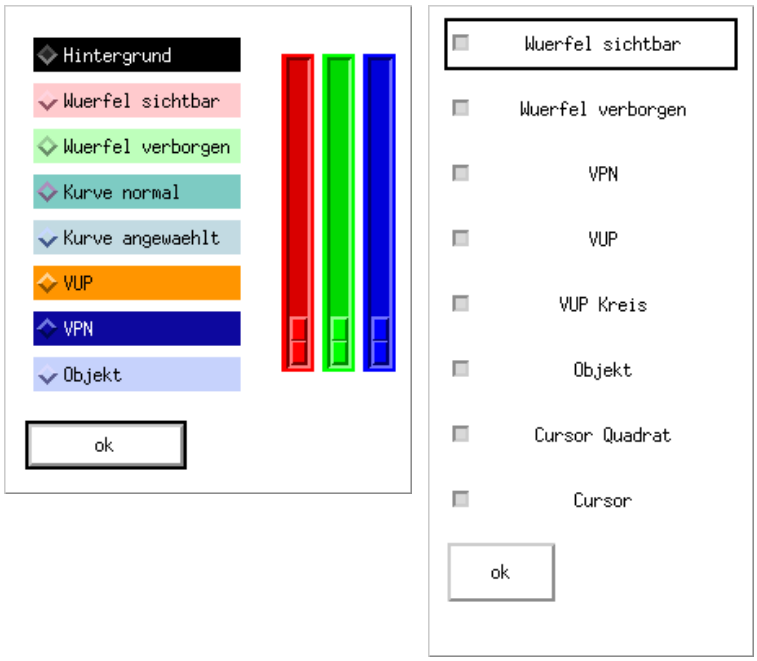


Abbildung A.9: Farbauswahl und Anzeigefestlegung

Standard 2: Tension = 0.6: Der Kurvenverlauf wird noch straffer.

Standard 3: Tension = -0.3: Die Kurve nimmt runde, bauchige Form an.

Standard 4: Tension = -0.6: Der Kurvenverlauf wird noch geschwungener.

Standard 5: Continuity = -0.5: Die Stetigkeit wird aufgehoben. Die Kurve endet im Stützpunkt in Richtung des Vorgängerpunktes, sie verläßt den Punkt in Richtung des Nachfolgers. Sinnvoll zu verwenden, wenn die räumliche Bewegung an diesem Punkt stehenbleibt und eine Rotation der Kamera stattfindet.

Standard 6: Continuity = 0.5: Die einlaufende Richtung ist auf den Nachfolger ausgerichtet, die auslaufende auf den Vorgänger.

Standard 7: Bias = 0.5: Die Bewegung schießt über den Stützpunkt hinaus und ändert erst nach dem Punkt die Richtung. In der Animation wird dies als Overshot bezeichnet und ist bei schneller Bewegung an scharfen Ecken sinnvoll.

Standard 8: Bias = -0.5: Die Bewegung ändert bereits vor dem Stützpunkt die Richtung. Sinnvoll an scharfen Kurven, die am Rande des Bewegungsraumes liegen, damit die Kamera nicht über den Bewegungsraum hinausschießt.

Standard 9: Alle Parameter null. Frei für eigene Vorgaben.

Diese Werte kann man jederzeit beliebig verändern, sie werden beim Abspeichern einer Kurve mit gespeichert.

A.6.2 Reihenfolge der Orientierungsinterpolation

Da bei der reinen Orientierungsdarstellung keine Interpolationskurve dargestellt wird, ist es wichtig zu verstehen, in welcher Reihenfolge eine Orientierungsdarstellung interpoliert wird.

Eine Interpolation erfolgt stets auf dem kürzesten Weg zwischen den beiden Beschreibungen. Der kürzeste Weg entspricht dem Kreisbogen mit dem geringsten Öffnungwinkel zwischen den einzelnen Orientierungen. Das bedeutet, daß eine Orientierungsänderung in mehrere Teile aufgeteilt werden muß, wenn sie nicht diesen Weg einnehmen soll. Soll sich eine Kamera z.B. um ihre eigene Achse drehen, so müssen mindestens zwei Zwischenorientierungen eingefügt werden. Die erste sollte nach rund einem Drittel des Interpolationskreises liegen, die zweite nach ungefähr zwei Dritteln.

Auch bei der Rotation des VUP-Vektors auf seinem Bewegungskreis wird entlang des kürzeren Kreissegmentes rotiert. Deshalb ist auch in diesem Fall darauf zu achten, größere Rotationen evtl. in mehrere Stücke zu unterteilen.

Wenn eine Bewegung zu einem neuen Stützpunkt durchgeführt wird, so wird gleichzeitig eine Orientierungsinterpolation ausgeführt. Die Orientierung erfolgt zwischen der letzten festgelegten Orientierung des Startpunktes und der ersten Orientierung des Zielpunktes. Deshalb ist auch die letzte Zeitkurve, die einem Punkt zugeordnet ist, ausschlaggebend für den Bewegunsablauf zum nachfolgenden Punkt.

Anhang B

Aufbau und Einbindung des VIDES-Systems

Aufbauend auf den vorgestellten Algorithmen wurde ein Programmpaket entwickelt, das dem Benutzer die Möglichkeit bietet, interaktiv Kamerabewegungen zu planen und festzulegen. Aus den Benutzerdaten generiert das Programm dann auf Anforderung die Interpolationswerte, die für eine Kamerabewegung entlang der Kurve benötigt werden.
Diese Routinen stehen als Programmbibliothek zur Verfügung und können in eigene Anwendungspakete eingebunden werden. Die Programmbibliothek trägt den Namen „VIDES“. VIDES läßt sich aber auch leicht eigenständig verwenden

B.1 Grundvoraussetzungen

Wie in Kapitel 7.2 erwähnt, ist VIDES aufbauend auf X-Windows/Motif entwickelt. Die Struktur der Event-gesteuerten Programmierung ist deshalb in Programmen, die VIDES verwenden, beizubehalten. Daher ist es sinnvoll, eigene Programme in der Programmiersprache „C“ zu entwickeln, da auch VIDES in dieser Programmiersprache geschrieben wurde.

Es erleichtert daher auch das Verständnis des folgenden Kapitels, wenn man grundlegende Kenntnisse in der Programmiersprache „C“ und in der Programmierung von X-Windows-/Motif besitzt. Dieses Kapitel richtet sich deshalb nicht an den Endanwender von VIDES, sondern ausschließlich an den Programmierer, der die Programmbibliothek verwendet.

B.2 Festlegungen

Die Schritte, die der Programmierer bei Benutzung von VIDES auszuführen hat, beschränken sich auf die Festlegung der Übergabeparameter und den Aufruf der Initialisierungsroutine `v_haupt` von VIDES. Optional kann der Programmierer noch eine Prozedur definieren, die von VIDES aufgerufen wird, wenn Interpolationsdaten erzeugt wurden, sowie eine Funktion, die VIDES startet, bevor es (vom Benutzer) beendet wird.

VIDES führt danach die Generierung der entsprechenden X-Windows Fenster durch, initialisiert seine Datenstrukturen und registriert die entsprechenden Callbacks und Eventhandler mit X-Windows/Motif. Danach gibt es die Kontrolle wieder an die Motif Hauptroutine zurück und wartet auf Bedienung durch den Benutzer.
Um die Definition der Übergabeparameter durchführen zu können, aber auch um die zurückgegebenen Interpolationswerte interpretieren und verstehen zu können, ist zunächst das verwendete Koordinatensystem festzulegen.

B.2.1 Festlegungen des Darstellungraumes

Es wurden zwei Definitionen getroffen, die die Größe des Raumes, in dem die Interpolation festgelegt werden kann, und das verwendete Koordinatensystem betreffen:

1. Der Abbildungsraum wurde auf einen Standardwürfel beschränkt, der für jede Koordinatenachse zwischen den Werten -1 und $+1$, also symmetrisch zum Ursprungspunkt, liegt.

2. Als Koordinatensystem wurde das rechtshändige Koordinatensystem mit nach oben gerichteter y-Achse, wie es in Kapitel 2.2.1 Abbildung 2.2 dieser Arbeit eingeführt wurde, verwendet.

Diese Definitionen sind nicht als Einschränkung zu betrachten. Jedes beliebige Koordinatensystem kann in den hier festgelegten Standardwürfel (oder Teile davon) mit den in Kapitel 2.2 beschriebenen Transformationen überführt werden.

B.2.2 Die Routine `v_haupt` und deren Übergabeparameter

Es können drei verschiedene Gruppen von Übergabeparametern definiert werden. Dies sind die Angaben zur Interpolationsgenauigkeit, die Festlegung von Objekten im Raum, die von VIDES angezeigt werden sollen, und die (optionale) Festlegung der oben bereits erwähnten Übergaberoutinen. Zusätzlich muß noch ein Element einer Datenstruktur bereitgestellt werden, in der die Rückgabe der Interpolationsparameter erfolgt.

```
void v_haupt   (move_speed,orient_speed,zoom_speed,winkel_speed,
                interpol_liste,anzahl_objekt,objekt,rueck,ende)
  float move_speed;
  float zoom_speed;
  float winkel_speed;
  float orient_speed;
  KAMERAPAR *interpol_liste;
  int anzahl_objekt;
  OBJEKT_LIST *objekt;
  void (*rueck)();
  void (*ende)();
```

1. Geschwindigkeitsparameter

   Die Parameter steuern die gewünschte Bewegungs-/Interpolationsgeschwindigkeit, damit eine der Anwendung angepaßte abgestufte Interpolation möglich ist. Bei der Interpolation sind vier Variationen sinnvoll, für die die Standardbewegungsgeschwindigkeit einzeln festgelegt werden muß:

   (a) Ortsinterpolation:
   Hier wird in der Variablen `move_speed` festgelegt, wieviele Zwischenpositionen für eine Bewegung mit der Länge (Abstand der Punkte) 1 in dem gegebenen Koordinatensystem (Standardwürfel von -1 bis 1) bei Grundgeschwindigkeit $v = 50$ (aus einem möglichen Bereich von 1-100) generiert werden sollen.

   (b) Orientierungsinterpolation:
   In der Variablen `orient_speed` ist definiert, mit welcher Geschwindigkeit die reine Orientierungsinterpolation (bei fester Position) von VPN mit oder ohne VUP erfolgen soll. Dies entspricht dem Schwenken der Kamera, wenn ihre Position unverändert bleibt.
   Bezugsgröße ist hierbei der Unterschied zwischen den Quaternionen der beiden Orientierungen, der durch die Berechnung des Skalarproduktes der beiden Quaternionen festgelegt wird (siehe Kapitel 4.4.3). Der dabei berechnete Winkel θ dient als Steuergröße, bei einem Winkel von $\theta = 90^o$ und Grundgeschwindigkeit werden `orient_speed` entsprechend viele Interpolationen durchgeführt.

   (c) Interpolation des VUP-Vektors:
   Es muß noch festgelegt werden, mit welcher Geschwindigkeit eine reine Bewegung des VUP-Vektors erfolgen soll. Diese Rotation ist durch einen Winkel in Bezug auf den VPN definiert. Die Geschwindigkeitsfestlegung erfolgt durch die Variable `winkel_speed`. Hierbei gilt wiederum, daß bei einer Bewegung mit der Winkeldifferenz von 90^o bei Grundgeschwindigkeit eine Interpolation durch `winkel_speed` Zwischenpositionen erfolgt.

   (d) Interpolation des Zoomfaktors:
   Da auch noch der Zoomfaktor alleine interpoliert werden kann, während die anderen Angaben konstant bleiben, wird in der Variablen `zoom_speed` festgelegt, mit welcher Zahl von Zwischenpositionen die Veränderung des Zoomfaktors innerhalb des möglichen Bereichs von 1 auf 100 bei Grundgeschwindigkeit durchgeführt werden soll.

   Bei der Berechnung einer zuerst aufgeführten Interpolation wird die Interpolation der nachfolgend genannten Werte, soweit nötig, ebenfalls durchgeführt. Die Interpolationsgeschwindigkeit richtet sich jedoch immer nach der in der aufgeführten Hierarchie zuerstgenannten Ebene.

2. Objekt

   Zusätzlich zu diesen Werten kann noch ein Objekt/Gegenstand an das aufrufende Programm übergeben werden, das im Eingabefenster der Interpolationskurve mit angezeigt wird. Das Objekt wird durch einige Punkte festgelegt, die durch Linien verbunden werden können.

   Um die drei Koordinaten eines Punktes festzulegen, werden diese in eine vordefinierte Struktur eingetragen:

```
typedef struct v_punkt
  { float x;
    float y;
    float z;
  } V_PUNKT;
```

Zu einem Punkt werden noch weitere Angaben benötigt. Jeder einzelne Punkt wird durch ein Element der Datenstruktur V_OBJEKT_LISTE festgelegt. Es kann eine beliebige Anzahl von Punkten in einem Array dieses Typs festgelegt und an die Routinensammlung übergeben werden.

Jedes einzelne Element von V_OBJEKT_LISTE sieht wie folgt aus:

```
typedef struct v_objekt_list
  { V_POS pos;
    unsigned short int draw;
    int number;
  } V_OBJEKT_LIST;
```

In pos werden die drei Koordinatenwerte x, y, z des Punktes gespeichert, draw legt fest, ob die Linie vom Vorgängerpunkt zu diesem Punkt gezeichnet werden soll oder nicht. Dabei gilt, wie in „C“ üblich, daß gezeichnet wird, wenn draw ungleich null („TRUE“) ist, und nicht gezeichnet wird, wenn draw gleich null („FALSE“) ist. In number wird die laufende Nummer des Punktes, angefangen bei 1, abgespeichert. Dies dient VIDES intern zur Kontrolle des Zeichenvorganges und muß angegeben werden.

Ebenso muß die Anzahl der Punkte beim Aufruf von v_haupt übergeben werden. Dazu dient die Integervariable anzahl_objekt.

3. Rückgabestruktur

   Des weiteren muß noch ein Zeiger auf ein Element der Datenstruktur V_OBJEKT_LIST mit an VIDES übergeben werden, der die Rückgabe der gewünschten Interpolationswerte ermöglicht. Der Zeiger muß auf ein gültiges Element dieser Datenstruktur zeigen.

   Die Rückgabe der Werte erfolgt dann in einer doppelt verketteten Liste der Interpolationswerte, wobei der übergebene Zeiger auf das erste Element der Liste zeigt.

   Die Datenstruktur ist wie folgt definiert:

```
typedef struct v_objekt_list
  { V_POS VRP;
    V_POS VPN;
    V_POS VUP;
    int zoom;
```

```
    interpolparameter *prev, *next;
    int number;
} V_OBJEKT_LIST;
```

VRP, VPN und VUP enthalten jeweils die Interpolationsangaben der drei Vektoren zur Festlegung der Projektion. zoom enthält einen skalaren Wert, der zur Definition des Bildausschnittes verwendet werden kann. Die Zeiger *prev und *next zeigen auf das Vorgängerelement und das nachfolgende Element der Liste der Interpolationsparameter. number zuletzt enthält zur Kontrolle eine laufende Nummer der Interpolationswerte.

Das an VIDES übergebene Element muß einen Wert < 0 in seinem number-Feld eingetragen haben. Daran erkennt VIDES, daß bisher noch keine Datenstruktur generiert wurde. Wenn nach Aufforderung durch den Benutzer von VIDES eine Interpolationsliste gebildet wurde, so wird die Liste komplett aufgebaut. Die Nummer des ersten Elementes, auf die der Zeiger weiterhin zeigt, ist nun 1. Durch Abfragen dieses Wertes kann das Hauptprogramm also feststellen, ob bereits eine Interpolation gebildet wurde.

4. Funktionen

   Da „C“ die Möglichkeit bietet, Zeiger auf Prozeduren zu übergeben und diese aus den aufgerufenen Prozeduren zu starten, verwendet VIDES diese Möglichkeit, um eine direkte Weiterverarbeitung seiner Interpolationsdaten zu initialisieren. An VIDES bzw. v_haupt können zwei Zeiger auf Prozeduren übergeben werden.

   Eine Prozedur rueck wird von VIDES aufgerufen, wenn nach Anforderung durch den Bediener eine Interpolationsreihe generiert worden ist. Diese Prozedur muß im Anwendungsprogramm wie folgt definiert werden:

```
void rueck(daten, anzahl)
     V_KAMERAPAR *daten;
     int anzahl;
  {
     /* Eigene Routinen */
  }
```

   An diese Prozedur wird ein Zeiger daten auf das erste Element der Rückgabestruktur übergeben, die im vorherigen Abschnitt erläutert wurde. Zusätzlich wird noch die Anzahl der übergebenen Elemente der Datenstruktur in anzahl übergeben.

   Die zweite Funktion, die bei Aufruf von v_haupt angegeben werden kann, wird aufgerufen, wenn der Benutzer VIDES (und nicht das Hauptprogramm) beenden will. Sie ermöglicht es dem Hauptprogramm, vor dem Ende von VIDES noch die Daten auszulesen oder auch VIDES das Terminieren insgesamt zu verbieten. Die Funktion ende muß wie folgt definiert werden:

```
int ende()
  {
    /* Eigene Routinen */
    return (1) /* VIDES das beenden erlauben */
  }
```

In der Rückgabevariablen wird mitgeteilt, ob VIDES seine Ausführung beenden darf. Wird der Wert auf 0 („FALSE“) gesetzt, so beendet VIDES die Ausführung nicht, jeder andere Wert („TRUE“) erlaubt VIDES, zu terminieren.

Wenn man an VIDES keine Prozeduren übergeben will, so ist es ausreichend, an `v_haupt` anstatt einen Pointer auf die Funktionen explizit den pointer `NULL` zu übergeben. VIDES überspringt dann die entsprechenden Prozeduraufrufe.

B.2.3 Rückgabe der Interpolationsdaten

Es gibt insgesamt drei Möglichkeiten, die Interpolationsdaten von VIDES zu übernehmen. Einerseits besteht die oben dargestellte Möglichkeit, an VIDES eine Funktion zu übergeben, die für das Hauptprogramm die Auswertung vornehmen kann.
Ebenso kann man regelmäßig oder bei Bedarf die Datenstruktur abfragen. Steht in dem `V_OBJEKT_LIST` Element, das man an VIDES übergeben hat, weiterhin der `number`-Wert -1, so wurden keine Daten generiert, steht dagegen der Wert $+1$, so sind Interpolationsdaten erzeugt worden. Da die Interpolationsdaten laufend durchnumeriert werden, enthält das letzte Element die Anzahl der insgesamt erzeugten Interpolationswerte. In der doppelt verketteten Liste entspricht das letzte Element dem Vorgängerelemente des ersten Elementes der Liste und damit dem Vorgänger des übergebenen Elementes. Es kann mit `interpol_liste->prev->number` abgefragt werden.
Zusätzlich besteht noch die Möglichkeit, VIDES die Interpolationswerte in eine Datei schreiben zu lassen. In der Datei stehen in ASCII-Text die Interpolationswerte in folgender Reihenfolge:

`VRP.x VRP.y VRP.z VPN.x VPN.y VPN.z VUP.x VUP.y VUP.z zoom`

In jeder Zeile stehen also 10 Fließkommawerte im Format `%3.4f`, die durch ein Leerzeichen getrennt werden. Jede Interpolationsgruppe steht in einer eigenen Zeile. Die Zeilen können mit den standardmäßigen „C“-Befehlen (`fscanf`) eingelesen werden.

Die interpolierten Positionen können zwischen ± 1 variieren, da dies den Grenzen des Darstellungsraumes entspricht. `VPN` und `VUP` sind auf den Betrag eins normiert, der `zoom`-Wert schwankt zwischen 1 und 100;

B.2.4 Einbinden der Library

Die Routinen von VIDES einschließlich der Initialisierungsroutine `v_haupt` sind in einer Library (Programmbibliothek) mit Namen ,,`libvides.a`‘‘ gesammelt. Will man diese Routinen verwenden, so ist lediglich diese Library zu dem entsprechenden Programmpaket zu linken sowie die dazugehörige Headerdatei einzubinden.

Die Headerdatei ,,`libvides.h`'' ist mit `#include` am Anfang des Programmes einzubinden. In ihr sind die oben aufgeführten Datenstrukturen definiert und der Prozedurkopf von `v_haupt` festgelegt.

Beim Compilieren des Gesamtprogrammes ist dann die Library `libvides.a` sowie die X-Window/Motif Libraries `libXm.a, libXt.a, libX11.a`, sowie die Standard Mathelibrary `libm.a` hinzuzulinken. Innerhalb eines Makefiles könnte ein Aufruf des Linkvorganges wie folgt aussehen, wenn das Programm `Eigen` , bestehend aus den Modulen `Eigen.o`, `eigen_eins.o` und `eigen_zwei.o` (entsprechend compiliert), gelinkt werden soll:
`cc -o Eigen Eigen.o eigen_eins.o eigen_zwei.o -lvides -lm -lXm -lXt -lX11`
Hierbei wird `libvides.a` unter `/usr/include` (oder aus anderen vom System abhängigen Pfaden) gesucht. Sollte die Library anderswo stehen, so ist der Pfad speziell anzugeben. Liegt libvides.a z.B. im aktuellen Verzeichnis, so wird sie mit `-L. -lvides` eingebunden.

B.3 Beispielprogramm

Im Anschluß ist ein Beispielprogramm aufgelistet, das die Einbindung der VIDES-Routinen anhand eines kurzen Programmes erläutert. Das Programm generiert ein Fenster, indem lediglich drei Knöpfe zur Verfügung gestellt werden. Durch Drücken des ersten Knopfes wird VIDES aufgerufen, der zweite Knopf simuliert eine Weiterverarbeitung der Daten, indem es diese ausdruckt. Der dritte Knopf beendet das Beispielprogramm. Hier sollen nun die wichtigsten Schritte dargestellt werden.

Zeile 1 Einbinden der VIDES Headerdatei `libvides.h`.

Zeile 3-9 Einbinden entsprechender X-Windows/Motif Routinen für das Hauptprogramm.

Zeile 11-14 Variablendefinition zur Übergabe.

Zeile 15-16 Definition der Funktionen, die an VIDES übergeben werden können

Zeile 20 Eigene Variablen für das Hauptprogramm (Widgets).

Zeile 24-38 `ende`: Diese Funktion wird von VIDES aufgerufen, bevor VIDES beendet wird. Als Funktionswert wird TRUE (1) zurückgegeben, damit erhält VIDES die Erlaubnis, zu terminieren.

Zeile 43-77 `read`: Diese Routine überprüft, ob von VIDES bereits Interpolationsdaten generiert wurden. Dazu wird das `number`-Feld der Datenübergabestruktur abgefragt (Zeile 59). Sind schon Daten generiert worden, so holt sich VIDES die Anzahl der Interpolationsdaten aus dem letzten Element der Liste (Zeile 61) und gibt dann in einer Schleife sämtliche Werte aus (Zeile 62-69).

Zeile 79-106 `rueck`: Prozedur, die an VIDES übergeben werden kann und von VIDES aufgerufen wird, wenn neue Interpolationsdaten erzeugt wurden. Diese werden in einer Schleife (Zeile 98-105) ausgegeben.

Zeile 109-123 In dieser Callbackprozedur wird die Initialisierungsroutine `v_haupt` von VIDES mit den Übergabeparametern aufgerufen.

Zeile 126-139 Diese Callback-Prozedur beendet das Beispielprogramm.

Zeile 150-231 Hauptroutine.

Zeile 155-193 Generierung der X-Anwendung einschließlich des toplevel-Widgets .

Zeile 196 Festlegen der Geschwindigkeitsparameter zur Übergabe an VIDES.

Zeile 199-219 Definition des Objektes: Es besteht aus zehn Punkten und stellt hier eine Pyramide dar.

Zeile 223-224 Erzeugung eines Elements der Rückgabeliste (mittels `malloc`) und setzen des `number`-Wertes auf -1.

Zeile 228-229 Starten der X-Anwendung.

B.4 Aufbau und Erzeugung der Library `libvides.a`

Das Programm besteht aus insgesamt zwölf einzelnen Modulen, die sich grob in zwei Klassen einteilen lassen:

- Module `projekt, spline,quaternion, time, mouse.`:
  Diese Module implementieren die in den vorangegangenen Kapiteln beschriebenen Algorithmen, die zur Projektion, Interpolation und Maussteuerung dienen.
- Module `vides, menu, color, k_b_edit, window_time, io` und `draw`:
  Sie enthalten vor allem Funktionen, die zur Generierung der X-Windows Oberfläche dienen und Prozeduren enthalten, die auf die entsprechenden Eingaben des Benutzers reagieren.

Alle Routinen stützen sich auf eine Datenstruktur, in der in einer doppelt verketteten Liste sämtliche Angaben der Kurve, die zur Interpolation benötigt werden, gespeichert sind. In jedem Element dieser Liste sind die Ortsangabe eines Stützpunktes mit bis zu zehn dazugehörigen Orientierungs-/Zoominformationen und die entsprechenden Zeit-/Geschwindigkeitsinformationen festgehalten. Des weiteren sind die Parameter, die den Kurvenverlauf beeinflussen (Kochanek-Bartels Parameter), festgelegt. Zusätzlich sind noch einige vorberechnete Interpolationswerte abgelegt, die der schnellen Berechnung der Interpolationskurve zur Bildschirmdarstellung dienen. Die entsprechende Struktur heißt `BEZIERPUNKT` und ist in `vides.h` vordefiniert.

Die Struktur `Projekt_Daten` speichert sämtliche zur Berechnung einer Projektion notwendigen Daten, wie sie in Kapitel 2.4.1 definiert wurden. Dies sind neben VRP, VPN, PRP und VUP noch Angaben über das Projektionsfenster durch U_{min}, U_{max}, V_{min} V_{max} sowie der Abstand der Front- und Backplane F und B und der Abstand D der View Plane von der View Reference Plane. Zusätzlich ist noch ein Pointer auf eine Projektionsmatrix enthalten, die die Abbildung mit den vorangegangenen Daten durchführt.

In der Struktur ABB_PAR sind sämtliche zur Darstellung auf dem Bildschirm benötigten Größen, wie Fenstergröße, Ausdehnung des Darstellungsraumes etc. und einige Werte für die Maussteuerung festgelegt.

Es wird versucht, möglichst viele Daten, die zur Bildschirmdarstellung benötigt werden, im Speicher zu halten und nicht jedesmal neu zu berechnen. Auch wenn dies mehr Arbeitsspeicher benötigt, konnte nur so eine Darstellungsgeschwindigkeit erreicht werden, die die verzögerungsfreie interaktive Bedienung möglich macht.

Weitere Variable wie Zählvariable, die Festlegung der Farbinformation und allgemeine Widgets von VIDES werden ebenfalls in `vides.h` definiert. Alle globalen Variablen sind zu einem struct `Global_Data` zusammengefaßt. Die Routine `v_haupt` erzeugt bei jedem Aufruf ein Element dieser Struktur. Diese Vorgehensweise hat zwei Vorteile:

- Einerseits sind die Daten so gekapselt, daß ein Hauptprogramm nicht auf sie zugreifen kann, da es nicht den Pointer auf diese Variablen kennt.
- Der größere Vorteil ist jedoch, daß dadurch `v_haupt` mehrfach aufgerufen werden kann. Da bei jedem Aufruf von VIDES bzw. `v_haupt` eine neue Instanz der Structur `Global_Data` generiert wird, kann VIDES in verschiedenen Fenstern mehrfach ausgeführt werden. Dadurch ist es z.B. möglich, den Anwender in zwei Fenstern zwei unabhängige Interpolationskurven entwickeln zu lassen.

Sämtliche Routinen von VIDES beginnen mit ,,v_'' um eine Verwechslung mit Routinen des Hauptprogrammes auszuschließen. Zur besseren Übersicht sei noch eine kurze Inhaltsangabe der einzelnen Module aufgeführt:

- `projekt.c`:
  In diesem Modul wird die Berechnung der Projektionsmatrix für eine Parallelprojektion nach dem in Kapitel 2.5 durchgeführten Algorithmus durchgeführt. Zusätzlich sind einige Prozeduren zur Rechnung mit 4×4 Matrizen enthalten.
- `spline.c`
  Hier ist die Kodierung der Algorithmen zur schrittweisen Berechnung der Interpolationskurven nach den Formeln von Kochanek-Bartels enthalten. Auf diese Routine wird in [GRO93] näher eingegangen.
- `quaternion.c`
  Dieses Modul enthält die Routinen zur Rechnung mit Quaternionen, wie Multiplikation, Addition und Normalisierung, die in Kapitel 4.4 dargestellt wurden. Ferner sind die Routinen zur Rotation eines Vektors mit Hilfe einer Quaternion sowie die Prozeduren zur sphärischen linearen Interpolation (slerp) enthalten.
- `time.c`
  In dieser Sammlung sind die Programme zur Zeitkurvenparametrisierung einer Splinekurve, wie in Kapitel 5.1.4 beschrieben, enthalten. Ferner enthält dieses Modul die Routine, die den eigentlichen Interpolationsvorgang unter Verwendung der gespeicherten Daten durchführt.

- `mouse.c`
  Im Modul `mouse.c` ist die in Kapitel 6.2 dargestellte Maussteuerung implementiert. Dabei ist sowohl die Eingabe eines Raumpunktes, wie auch die Festlegung des VUP-Winkels und die Anzeige der Hilfslinien (Koordinatenkreuz und Bewegungsebenen) enthalten.

- `vides.c`
  Die Initialisierungsroutine v_haupt ist in diesem Modul neben zahlreichen anderen Routinen zur Initialisierung, Aufbau der Datenstruktur, Festlegung der Projektionsparameter etc. enthalten. Ferner sind Routinen zur Bedienung des Programmes wie die Rotation des Darstellungswürfels, das Hinzufügen und Löschen von Punkten und Orientierungen und zur Aktualisierung des Bildschirmes enthalten.

- `menu.c`
  Hier erfolgt der Aufbau des Hauptfensters einschließlich der Pulldown Menus, sämtlicher Bedienelemente im Hauptfenster und der Zeichenfläche. Weiterhin wird mit diesen Prozeduren die Initialisierung der Farben für die Zeichnung durchgeführt.

- `color.c`
  In diesem Modul wird der Aufbau des Unterfensters zur Farbauswahl vorgenommen sowie die Steuerung und Auswertung der Benutzereingaben in diesem Fenster durchgeführt. Des weiteren wird auch das Fenster, mit dem die Darstellung der einzelnen Elemente wie Abbildungswürfel, Orientierung etc. festgelegt wird, initialisiert und gesteuert.

- `k_b_edit.c`
  Hiermit erfolgt Aufbau und Steuerung des Fensters, in dem die Kochanek-Bartels Kurvenparameter eingegeben werden können und die numerische Eingabe von Orts- und Orientierungsinformation vorgenommen werden kann.

- `io.c`
  Die Ein-/Ausgabe, wie Abspeichern und Einlesen der festgelegten Werte der Stützstellen sowie der Information zur Anpassung des VIDES-Moduls (Farbauswahl etc.) ist in diesem Modul kodiert.

Hat man die Quelldateien diese Routinen vorliegen, so ist die Library schnell zu generieren. Durch Aufruf des Makefiles `vidarch` werden die Module übersetzt und die Librarydatei `libvides.a` erzeugt. Dies geschieht durch Eingabe von `make -f vidarch`. Danach können die erzeugten *.o Objektfiles (projekt.o, splines.o usw.) gelöscht werden und die Librarydatei kann an die gewünschte Stelle, sinnvollerweise unter `/usr/include` kopiert werden.

Anhang C

Beispielprogramm

```
 1 #include "libvides.h"                /* Einbinden der Headerdatei von vides */
 2
 3 #include <Xm/Xm.h>                   /* Einbinden von X-Windows und Motif Libraries */
 4 #include <Xm/Form.h>                 /* Dienen nur dem Hauptprogramm */
 5 #include <Xm/PushB.h>
 6 #include <stdio.h>
 7 #include <math.h>
 8 #include <X11/Xutil.h>
 9 #include <X11/Shell.h>;
10
11 V_KAMERAPAR *interpol_liste;                          /* Definieren: Zeiger auf Feld zur Datenuebergabe */
12 V_OBJEKT_LIST  objekt[10];                            /* Zehn Punkte fuer Objekt in Vides */
13 int anzahl_objekt=10;                                 /* Anzahl der Punkte des Objektes in Vides */
14 float move_speed,zoom_speed,winkel_speed,orient_speed;    /* Geschwindigkeitsvariablen */
15 void rueck();                                         /* Funktionen die an Vides uebergeben werden */
16 int ende();
17
18
19
20 Widget toplevel,form,button1,button2,quit;               /* Eigene Variablen des Hauptprogrammes */
21
22
23
24 /*********************************************************************/
25 /*                                                                   */
26 /*   Funktion , die von Vides aufgerufen wird, wenn der Benutzer Vides  */
27 /*   beenden will.                                                   */
28 /*   Eingabeparameter : keine                                        */
29 /*   Ausgabeparameter : integerwert: 0 (FALSE) : vides darf nicht enden */
30 /*                                   1 (TRUE)  : vides darf enden       */
31 /*                                                                   */
32 /*********************************************************************/
33
34 int ende ()
35 {
36  int i=1;
37  printf ("vides hat um Erlaubnis gebeten, abzuschliessen\n");
38  return (i);   /* Vides erlauben, abzuschliessen */
39
40 }
41
42
43 /*********************************************************************/
44 /*                                                                   */
45 /*   Funktion , die prueft, ob Vides bereits Interpolation durchge-     */
46 /*   fuehrt hat, und dann ggf. die Daten ausgibt                     */
47 /*   Eingabeparameter : wie bei Callbackfunktion von X               */
48 /*   Ausgabeparameter : keine                                        */
```

```
 49 /*                                                                    */
 50 /**********************************************************************/
 51 void read (w,client_data, event)
 52      Widget   w;
 53      caddr_t  client_data;
 54      XEvent   *event;
 55   {
 56      int i,anzahl;            /* eigene Variablen */
 57      V_KAMERAPAR *aktuell;    /* weiterer Zeiger auf Interpolationsliste */
 58
 59      if (interpol_liste->number >=0)              /* schon Daten da ? sonst number = -1     */
 60       { aktuell = interpol_liste;                /* Zeiger auf das erste Element kopieren  */
 61          anzahl= interpol_liste->prev->number;    /* Anzahl: im letzten Element ist die lauf-*/
 62          for (i=0;i<anzahl;i++)                   /*         fende Nummer gleich der Anzahl */
 63            {
 64               printf ("Die Werte des %iten  VRP sind %f %f %f\n",aktuell->number, /*Werte ausgeben*/
 65                                                                  aktuell->VRP.x,
 66                                                                  aktuell->VRP.y,
 67                                                                  aktuell->VRP.z  );
 68               aktuell = aktuell->next;       /* Zeiger auf das naechste Element setzen */
 69            }
 70
 71        }
 72      else                                     /* leider noch keine Daten */
 73        {
 74           printf ("leider noch keine Daten \n");
 75        }
 76
 77   }
 78
 79 /**********************************************************************/
 80 /*                                                                    */
 81 /*   Funktion , die von Vides aufgerufen wird, wenn der Benutzer Inter- */
 82 /*   polationsdaten generiert hat.                                    */
 83 /*   Eingabeparameter : daten:  Zeiger auf erstes Element             */
 84 /*   Eingabeparameter : anzahl: Anzahl der Interpolationswerte        */
 85 /*   Ausgabeparameter : keine                                         */
 86 /*                                                                    */
 87 /**********************************************************************/
 88 void rueck (daten,anzahl)
 89   V_KAMERAPAR *daten;
 90   int anzahl;
 91
 92 {
 93   int i;                    /* Laufvariable */
 94   V_KAMERAPAR *aktuell;     /* Zeiger auf Ausgabeliste */
 95
 96   printf ("habe %i Punkte bekommen \n",anzahl);
 97   aktuell = daten;                                        /* Zeiger auf das erste Element kopieren  */
 98   for (i=0;i<anzahl;i++)
 99     {
100       printf ("Die Werte des %iten  VRP sind %f %f %f\n",aktuell->number,       /* Werte ausgeben */
101                                                          aktuell->VRP.x  ,
102                                                   aktuell->VRP.y  ,
103                                                          aktuell->VRP.z  );
104       aktuell = aktuell->next;                            /* Zeiger auf das naechste Element setzen */
105     }
106 }
107
108
109 /**********************************************************************/
110 /*                                                                    */
111 /*   Prozedur , die Vides aufruft                                     */
112 /*   Eingabeparameter : wie bei Callbackfunktion von X                */
113 /*   Ausgabeparameter : keine                                         */
114 /*                                                                    */
115 /**********************************************************************/
116 void edit (w,client_data, event)
```

```
117      Widget w;
118      caddr_t client_data;
119      XEvent *event;
120    {
121       v_haupt (move_speed,orient_speed,zoom_speed,winkel_speed,          /* Starten von Vides mit den */
122               interpol_liste,anzahl_objekt,objekt,rueck,ende);         /* Uebergabeparametern       */
123    }
124
125
126 /************************************************************************/
127 /*                                                                      */
128 /*   Prozedur, die Hauptprogramm beendet                                */
129 /*   Eingabeparameter : wie bei Callbackfunktion von X                  */
130 /*   Ausgabeparameter : keine                                           */
131 /*                                                                      */
132 /************************************************************************/
133 void quit_func (w,client_data, event)
134      Widget w;
135      caddr_t client_data;
136      XEvent *event;
137    {
138      exit (0);                                                          /* Hauptprogramm schliessen */
139    }
140
141
142
143 /************************************************************************/
144 /*                                                                      */
145 /*  Hauptprogramm                                                       */
146 /*                                                                      */
147 /************************************************************************/
148
149
150 void main (argc, argv)
151      int    argc;
152      char   *argv[];
153
154 {
155   Arg    wargs[10];
156   int    n = 0;
157
158 /* Zuerst X-Bildschirm aufbauen */
159
160   toplevel = XtInitialize (argv[0], "MainTest", NULL ,0, &argc, argv);    /* ToplevelWidget generieren */
161
162   XtSetArg(wargs[n], XmNheight,200);n++;                                  /* Form Widget auf toplevel  */
163   XtSetArg(wargs[n], XmNwidth, 100);n++;
164   form = XtCreateManagedWidget ("form", xmFormWidgetClass, toplevel, wargs, n);
165   n = 0 ;
166
167   XtSetArg(wargs[n], XmNtopAttachment,     XmATTACH_POSITION);    n++;    /*  Knopf definieren um */
168   XtSetArg(wargs[n], XmNtopPosition, 0);                          n++;    /*  Vides aufzurufen    */
169   XtSetArg(wargs[n], XmNbottomAttachment,     XmATTACH_POSITION); n++;
170   XtSetArg(wargs[n], XmNbottomPosition, 32);                      n++;
171   XtSetArg(wargs[n], XmNleftAttachment,    XmATTACH_FORM);        n++;
172   XtSetArg(wargs[n], XmNrightAttachment,   XmATTACH_FORM);        n++;
173   button1 = XtCreateManagedWidget("Edit Kurve",xmPushButtonWidgetClass,form,wargs,n);
174   XtAddCallback(button1, XmNactivateCallback, edit,NULL);
175
176   XtSetArg(wargs[n], XmNtopAttachment,     XmATTACH_POSITION);    n++;   /* Knopf definieren um      */
177   XtSetArg(wargs[n], XmNtopPosition, 33);                         n++;   /* Daten von vides zu holen */
178   XtSetArg(wargs[n], XmNbottomAttachment,     XmATTACH_POSITION); n++;
179   XtSetArg(wargs[n], XmNbottomPosition, 65);                      n++;
180   XtSetArg(wargs[n], XmNleftAttachment,    XmATTACH_FORM);        n++;
181   XtSetArg(wargs[n], XmNrightAttachment,   XmATTACH_FORM);        n++;
182   button2 = XtCreateManagedWidget("Hole Werte",xmPushButtonWidgetClass,form,wargs,n);
183   XtAddCallback(button2, XmNactivateCallback, read,NULL);
184
```

```
185   XtSetArg(wargs[n], XmNtopAttachment,     XmATTACH_POSITION);    n++;   /* Knopf definieren um         */
186   XtSetArg(wargs[n], XmNtopPosition, 66);                         n++;   /* um Hauptprogramm zu beenden */
187   XtSetArg(wargs[n], XmNbottomAttachment,     XmATTACH_POSITION); n++;
188   XtSetArg(wargs[n], XmNbottomPosition, 99);                      n++;
189   XtSetArg(wargs[n], XmNleftAttachment,    XmATTACH_FORM);        n++;
190   XtSetArg(wargs[n], XmNrightAttachment,   XmATTACH_FORM);        n++;
191   quit = XtCreateManagedWidget("Quit",xmPushButtonWidgetClass,form,wargs,n);
192   XtAddCallback(quit, XmNactivateCallback, quit_func,NULL);
193
194 /* Variablen zur Uebergabe an Vides festlegen */
195   /* Zuerst Grundgeschwindigkeiten festlegen  */
196   move_speed    = 100;  zoom_speed    = 100;  winkel_speed  = 100;  orient_speed  = 100;
197
198   /* Nun Objekt aus zehn Linien definieren: (Pyramide mit Quadrat als Grundflaeche */
199   anzahl_objekt = 10;
200   objekt[0].pos.x  =  0.3;  objekt[0].pos.y  =  0;   objekt[0].pos.z  =  0.3; /* 1. Punkt */
201   objekt[0].draw   =  1;    objekt[0].number =  1;
202   objekt[1].pos.x  =  0.3;  objekt[1].pos.y  =  0;   objekt[1].pos.z  = -0.3; /* 2. Punkt */
203   objekt[1].draw   =  1;    objekt[1].number =  2;
204   objekt[2].pos.x  = -0.3;  objekt[2].pos.y  =  0;   objekt[2].pos.z  = -0.3; /* 3. Punkt */
205   objekt[2].draw   =  1;    objekt[2].number =  3;
206   objekt[3].pos.x  = -0.3;  objekt[3].pos.y  =  0;   objekt[3].pos.z  =  0.3; /* 4. Punkt */
207   objekt[3].draw   =  1;    objekt[3].number =  4;
208   objekt[4].pos.x  =  0.3;  objekt[4].pos.y  =  0;   objekt[4].pos.z  =  0.3; /* 5. Punkt */
209   objekt[4].draw   =  1;    objekt[4].number =  5;
210   objekt[5].pos.x  =  0;    objekt[5].pos.y  =  0.5; objekt[5].pos.z  =  0;   /* 6. Punkt */
211   objekt[5].draw   =  1;    objekt[5].number =  6;
212   objekt[6].pos.x  =  0.3;  objekt[6].pos.y  =  0;   objekt[6].pos.z  = -0.3; /* 7. Punkt */
213   objekt[6].draw   =  1;    objekt[6].number =  7;
214   objekt[7].pos.x  = -0.3;  objekt[7].pos.y  =  0;   objekt[7].pos.z  = -0.3; /* 8. Punkt */
215   objekt[7].draw   =  0;    objekt[7].number =  8;
216   objekt[8].pos.x  =  0.0;  objekt[8].pos.y  =  0.5; objekt[8].pos.z  =  0.0; /* 9. Punkt */
217   objekt[8].draw   =  1;    objekt[8].number =  9;
218   objekt[9].pos.x  = -0.3;  objekt[9].pos.y  =  0;   objekt[9].pos.z  =  0.3; /*10. Punkt */
219   objekt[9].draw   =  1;    objekt[9].number =  10;
220
221
222   /* Nun noch eine Element fuer die Rueckgabe von Daten generieren */
223   interpol_liste = (V_KAMERAPAR *)malloc ( sizeof(V_KAMERAPAR));    /* Speicher allokieren      */
224   interpol_liste->number = -1;                                /* Nummer = -1:  noch keine */
225                                                                      /* interpolation erfolgt    */
226
227/* Jetzt noch X-Windows starten */
228   XtRealizeWidget (toplevel);
229   XtMainLoop ();
230
231 } /* of main programm */
```

Literaturverzeichnis

[AVS] C. Upson et al.: *The application visualization system: a computational environment for scientific visualization*
IEEE Computer Graphics and Applications 1990, Seite 30-42
Hersteller: AVS Inc. 300 Fifth Ave., Waltham, Ma 02154, USA

[BAR92] Alan H. Barr, Bena Currin, Steven Gabriel, John F. Hughes: *Smooth Interpolation of Orientations with Angular Velocity Constraints using Quaternions*
SIGGRAPH '92, Computer Graphics, Volume 26, Number 2, July 1992
Seite 313 ff.

[BRO87] Bronstein, Sememdjajew: *Taschenbuch der Mathematik*
Verlag Harri Deutsch, Thun und Frankfurt/Main 1987

[CHA89] Sheue-Ling Chang, Michael Shantz, Robert Rocchetti: *Rendering Cubic Curves with Integer Adaptive Forward Differencing*
SIGGRAPH '89, Computer Graphics, Volume 23, Number 3
Seite 157-166

[EAR] R.A. Earnshaw, B. Wyvil: *New Advances in Computer Graphics*
Springer-Verlag, Wien, New York, Tokyo 1989
Chapter 4: Quaternions and Motion Interpolation: A Tutorial
Seite 223-244

[E&W] R. A. Earnshaw, D.Watson: *Animation and Scientific Visualization*
Academic Press, San Diego, CA 1993
Part II Chapter 4: Development of a procedure for generating graphical computer animations.

[FEL88] Wolf-Dietrich Fellner: *Computer Grafik*
BI Wissenschaftsverlag Mannheim, Wien,Zürich 1988
Kapitel 12: Dreidimensionale Transformationen

[FIG94] L. H. de Figueiredo: *Adaptive Polygonal Approximation of parametric curves*
Bisher unveröffentlicht, eingereicht bei Computers & Graphics
(Addresse : IMPA Instituto de Matematica Pura e Aplicada Estrada Dona Castorina, 110 22460-320 Rio de Janeiro, RJ, Brasil)

[FOL90] Foley, vanDam, Feiner, Hughes: *Computer Graphics : Principles and Practice*
Addison-Wesly Publishing Company, Reading, Massachusetts 1990

[FUC87] K. Fuchs: *Flexible, sensorgesteuerte Roboterschweißsysteme*
Technisch wissenschaftlicher Bericht des Institutes für Prozeßsteuerung in der Schweißtechnik der Rheinisch-Westfälischen Technischen Hochschule Aachens 1987
Kapitel 4.2: Koordinatentransformation
Seite 100-123

[GAS91] Tom Gaskins: *PHIGS Programming Manual*
3D Programming in X
O' Reilly & Associates, Inc. Sebastopl, CA 1992

[GLA90] A. Glassner: *Graphics Gems : Planar Cubic Curves*
Academic Press, Inc., San Diego, California 1990
Seite 575 ff.

[GRO93] W.-D. Groß *Raumkurvendefinition für Bewegungsabläufe*
Studienarbeit in Elektrotechnik
Zentralinstitut für Angewandte Mathematik, Forschungszentrum Jülich 1993

[GUN90] Brian Guenter, Richard Parent: *Computing the Arc Length of Parametric Curves*
IEEE Computer Graphics & Aplications 10, 1990
Seite 72-78

[HAR87] S. Harrington: *Computer Graphics*
Mc Graw Hill, New York, New York 1987
Chapter 11 : Curves and Fractals

[HOG91] S.G. Hoggar: *Mathematics for Computer Graphics*
Academic Press, Inc., San Diego, California 1991
Cambridge University Press, Glasgow
Chapter 9: Quaternions and Rotations
Seite 377 ff.

[H&H91] T.J.L. Howard, W.T. Hewitt, R.J. Hubbold, K.M. Wyrwas: *A Practical Introduction to PHIGS and PHIGS PLUS*
Addison Wesley Publishing Company, Wokingham, Reading 1991

[JEL82] R. Jeltsch: *Numerische Mathematik für Ingenieure*
Vorlesungsskript des Institutes für Geometrie und Praktische Mathematik der RWTH Aachen, Aachen 1982
Kapitel 5 : Interpolation

[KOC84] D. Kochanek, R. Bartels: *Interpolating Splines with Local Tension, Continuity and Bias Control*
SIGGRAPH '84, Computer Graphics, Volume 18, Number 3, July 1984
Seite 33 ff.

[KRO92] D. Krömker: *Visualisierungssysteme*
Springer-Verlag, Berlin, Heidelberg 1992

[K&R83] B.W. Kernighan, D.M. Ritchie: *Programmieren in C*
Carl Hauser Verlag, München, Wien 1983

[LIN92] T. Lindgren, J. Sanchez, J.Hall: *Graphic Gems II: Curve tesselation criteria through sampling*
Academic Press, Inc., San Diego, California 1992
Seite 262-265.

[LUT88] Wolfram Luther, Martin Ohsmann: *Mathematische Grundlagen der Computer Grafik*
Vieweg Verlag Braunschweig,Wiesbaden 1988/89
Projektion in eine Bildebene

[MAA90] Maarten J.G.M. van Emmerik: *A Direct Manipulation Technique for Specifying Objekt Transformations with a 2D Input Device*
Eurographics Association: Computer Graphics Forum 9, 1990
Seite 355-361

[MAI90] Patrick-Gilles Maillot: *Graphic Gems: Using Quaternions for coding 3D Transformations*
Academic Press, Inc., San Diego, California 1990
Seite 377 ff.

[MOL93] Laura Moltedo and Serena Morigi: *ANIMA: An Interactive Tool for Scientific Data Animation*
Eurographics Association: Computer Graphics Forum 12, 1993: Seite 277-288

[MOR92] Jack Morrison: *Graphic Gems III: Quaternion Interpolation with Extra Spins*
Academic Press, Inc., San Diego, California 1992
Seite 96 ff.

[PUR] W. Purgathofer: *Graphische Datenverarbeitung*
Springer-Verlag, Wien, New York
Kapitel 3.4: Kurven und Flächen
Seite 138-155

[SCIAN] *SciAn: Scientific Visualization and Animation Package*
Frei verfügbare Software; verfügbar über anonymous FTP bei ftp.scri.fsu.edu (oder 144.174.128.34) im Verzeichnis pub/SciAn

[SCO85] Scott N. Steketee, Norman I. Badler *Parametric Keyframe Interpolation Incorporating Kinetic Adjustment and Phrasing controll*
SIGGRAPH '85, Computer Graphics, Volume 19, Number 3, July 1985 Seite 255-262

[SHO85] Ken Shoemake: *Animating Rotation with Quaternion Curves*
SIGGRAPH '85, Computer Graphics, Volume 19, Number 3, July 1985
Seite 245 ff.

[SLG91] John Schlag: *Graphic Gems II: Using Geometric Constructions to Interpolate Orientation with Quaternions*
Academic Press, Inc., San Diego, California 1991
Seite 377 ff.

[TDI] Thomson Digital Image: *Explore: Europe's leading high end animation system*
Hersteller: TDI World Headquarter, 22 Rue Hegesippe-Moreai, 75018 Paris, France

[VIS5D] Vis5D: *A System for visually exploring the output of weather models and similar data sets*
Frei verfügbare Software, verfügbar über anonymous FTP bei vis5.ssec.wisc.edu (oder 144.92.108.66) im Verzeichnis pub/vis5d

[YOU90] David A. Young: *The X window system: programming and applications with Xt*
Prentice Hall Inc. Englewood Cliffs, NJ 90

[ZUN93] Lothar Zunker : *Transputerbasierter Lichtschnittsensor als Baustein eines Multisensor-Steuerungskonzepts zur Echtzeit-Schweißprozeßführung und seine Integration in ein Roboterschweißsystem.*
Technisch wissenschaftlicher Bericht des Institutes für Prozeßsteuerung in der Schweißtechnik der Rheinisch-Westfälischen Technischen Hochschule Aachens 1993
Kapitel 11 Anhang: Theoretische Grundlagen der Koordinatentransformation

Des weiteren wurden Hinweise und Diskussionsbeiträge aus verschiedenen Beiträgen der Usergruppe comp.graphics.algorithms herangezogen.

Die vorliegende Diplomarbeit wurde am Lehrstuhl für Technische Informatik und Computerwissenschaften der Rheinisch-Westfälischen Technischen Hochschule (RWTH) Aachen erstellt.

Dem Lehrstuhlinhaber Herrn Prof. Dr. F. Hoßfeld danke ich für die Möglichkeit, die Arbeit am Zentralinstitut für Angewandte Mathematik des Forschungszentrums Jülich (KFA) anfertigen zu können.

Mein besonderer Dank gilt Herrn Bartel für die Betreuung, zahlreiche konstruktive Diskussionen und sonstige Hinweise, die zum Gelingen dieser Arbeit beigetragen haben.

An dieser Stelle möchte ich auch meinen Eltern für die Unterstützung während des Studiums und besonders bei der Erstellung dieser Arbeit danken.

Bei meiner Verlobten und Ihrer Mutter möchte ich ebenfalls für die Mühe bedanken, die sie in die Fertigstellung dieser Arbeit investiert haben.

Aachen, den 15. September 1994