

1 Computergrafik und Virtual Reality

Bernhard Eberhardt, Jens-Uwe Hahn

Virtuelle Welten und künstlich erzeugte Bilder haben mit steigender Rechenleistung und mit der Ausreifung der eingesetzten Verfahren und Techniken zunehmenden Einfluss auf unser tägliches Leben.

Die Einsatzgebiete sind dabei vielseitig und reichen von Werbe- und Kinofilmen über Computerspiele bis hin zu Virtual-Reality-(VR-) Anwendungen, die es erlauben, virtuelle Prototypen von Gebäuden, Flugzeugen, Fahrzeugen und vielen anderen Produkten in Echtzeit zu betrachten, zu bewerten und weiterzuentwickeln.

Dieses Kapitel bietet eine Einführung und einen Überblick über die Verfahren, die dabei zum Einsatz kommen. Das erste Unterkapitel befasst sich mit der Modellierung virtueller Welten und den dafür eingesetzten Datenstrukturen. Das zweite Unterkapitel behandelt die Erzeugung von Bildern solcher virtuellen Welten. Das dritte Unterkapitel gibt einen Überblick über die Techniken, die in interaktiven VR-Umgebungen zum Einsatz kommen.

1.1 Modellierung virtueller Welten

Obwohl Kameramann und Fotograf häufig Bilder realer Gegenstände zeigen, die wir aus unserer Umgebung kennen, überraschen uns die Künstler dieser Fächer immer wieder mit neuen Ein- und Aussichten in ihren Einstellungen. Aber stellen Sie sich einen Fotografen vor, der etwas fotografiert, das gar nicht da ist. Gerade in der Computergrafik und -animation besteht aber zu Beginn immer das Problem, dass selbst der Gegenstand oder das Objekt, von dem man gerne Bilder produzieren möchte, nicht realer Natur ist. Objekte, Szenerie und Darsteller existieren nur in unserer Phantasie, sie müssen erst mathematisch beschrieben und erzeugt werden. Sie bleiben aber bis zuletzt künstlich, virtuell.

Doch gerade in dieser Problematik kann auch der Reiz und Schatz von Virtual-Reality-Anwendungen oder Computeranimationen liegen, denn nur die menschliche Vorstellung begrenzt die Produktion.

Diesem Problem, d.h. der Beschreibung und schnellen Verarbeitung virtueller Modelle, werden wir uns in diesem Kapitel zuwenden. Wir wollen Datenstrukturen entwickeln, die für Virtual-Reality-Anwendungen und Computeranimationen eingesetzt werden können, und ihre jeweiligen Vor- und Nachteile besprechen.

1.1.1

Objektmodellierung, -repräsentation und ihre Datenstrukturen

Erster Schritt der „technischen“ Produktionspipeline ist es also, dreidimensionale Gegenstände, Szenen oder virtuelle Darsteller in ihrer Form und Eigenschaft zu beschreiben – das Modellieren.

Wie und wodurch beschreiben wir (dreidimensionale) Objekte? Eine erste Antwort auf diese Frage ist schnell gefunden, nämlich durch die Angabe der geometrischen Form, welche durch die Angabe der (dreidimensionalen) Punkte und Flächen gegeben ist. Diese Punkte und mathematischen Formulierungen dienen dazu, die Objekte beschreiben zu können.

Zunächst betrachten wir einfache Rendrepräsentierungen, welche eine Objektoberfläche durch geschicktes und genaues Zusammenfassen von Polygonen, die wiederum aus Punkten und Kanten beschrieben werden, als Näherung darstellen. Dann folgt die Beschreibung durch Kurven und Flächen höherer Ordnung, sogenannte Freiformflächen.

So entstehen mehrere Möglichkeiten, dreidimensionale Objekte zu beschreiben, und je nach beabsichtigter Anwendung, d.h. technischer Konstruktion, Animation oder wissenschaftlicher Simulation, besitzen diese Vor- und Nachteile. Es ist also eine sorgfältige Wahl der verwendeten Datenstruktur zu treffen.

1.1.1.1

Randrepräsentierungen

Dreidimensionale Objekte werden in ihrer Form durch ihre (sichtbare) Oberfläche beschrieben. Punkte sind durch Kanten verbunden, die zu ebenen Polygonflächen zusammengefasst werden. Eine erste Datenstruktur, wie man sie z. B. in OpenInventor, Java3D und Wavefront obj-Formaten wiederfindet, ist einfach gehalten. Sie trennt aber schon die Lage der Punkte (Geometrie) von der Verbindung zu Flächen (Topologie).

Definition: Explizite Darstellung eines Polygonnetzes (vgl. Java 3D, VRML ...)

- Facette: $P_i = (V_p^i, V_q^i, \dots, V_n^i)$, mit $V_j^i \in \mathbb{R}^d$
- Netz: $P = (P_0, P_1, \dots, P_{m-1})$

Dieses Vorgehen wird an folgendem Beispiel deutlich: Wir wollen einen einfachen (Einheits-)Quader darstellen:

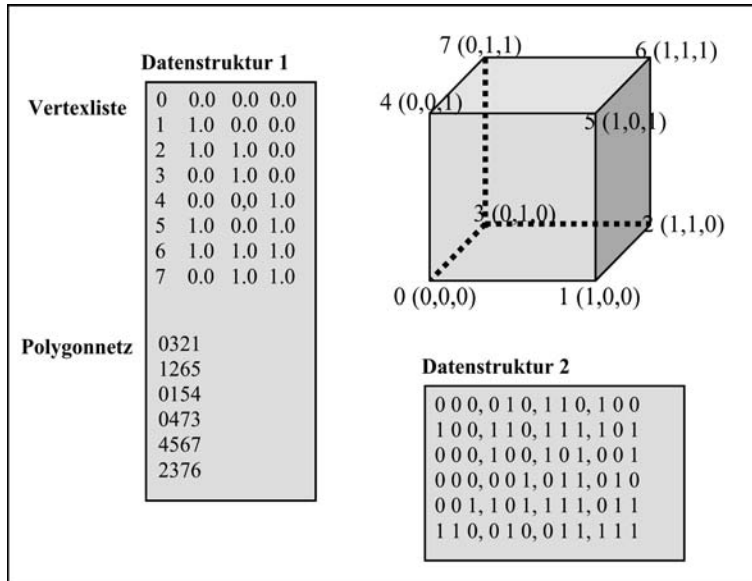


Abb. 1.1:
Beispiele für die Darstellung eines Polygonnetzes

Sie sehen in Abb. 1.1 bereits zwei mögliche Datenstrukturen für einen einfachen dreidimensionalen Einheitswürfel. Datenstruktur 1 besteht aus einer Liste der im Modell vorkommenden Punktkoordinaten und einer davon getrennten Liste der das Modell definierenden Polygonflächen.

Datenstruktur 2 hingegen definiert die Polygonflächen direkt durch die Angabe der Koordinaten der Punkte unter der Vereinbarung, dass jeweils vier (oder drei) aufeinander folgende Punkte eine geschlossene Fläche bilden.

Datenstruktur 1 besitzt einige Vorteile: Zuerst ist sie sehr einfach und intuitiv. Durch die Trennung von Topologie und Geometrie sind bei einer Lageänderung eines oder mehrerer Punkte nur die neuen Koordinaten der Punkte neu zu setzen bzw. zu übertragen. Der innere Zusammenhang, d.h. die Gitterstruktur der Kanten oder besser die Topologie des Körpers, ändert sich nur dann, wenn der Körper auseinanderbrechen würde.

Der Nachteil dieser sog. knotenbasierten Datenstrukturen wird sofort deutlich, wenn man Anwendungen programmieren möchte, bei denen sich die Form und Geometrie der Körper ändert. Will man die Geometrie ändern, so muss der entsprechende Eintrag in der Vertex-liste geändert werden bzw. muss in allen Flächen, welche diesen Punkt enthalten, entsprechend die Koordinate abgeändert werden. Aufwendige Suchfunktionen sind dafür notwendig. Noch schwieriger wird es, wenn die Topologie des Körpers verändert wird, wie z.B. bei einer booleschen Differenz oder Summe in einem CAD-Modellierer.

Diese knotenbasierten Datenstrukturen sind deswegen häufig dort im Einsatz, wo es schlicht um eine Visualisierung und kleine Datenformate bei der Übertragung geht.

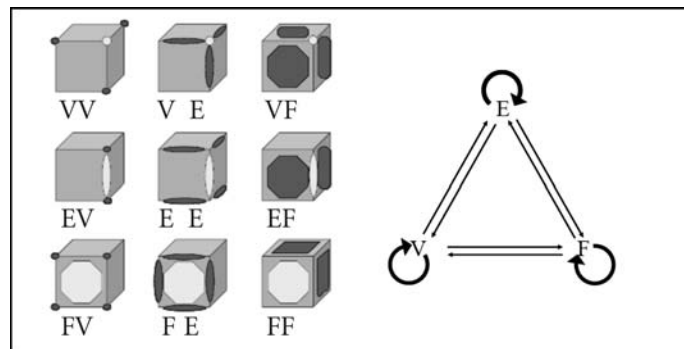
All die Schwierigkeiten kreisen um ein und dasselbe Problem. Wie können Nachbarschaftsinformationen, die wir für eine Verarbeitung des Modells benötigen, schnell bereitgestellt werden? Aber überlegen wir zunächst, was das für Informationen sind und wie viele es gibt?

Alles, was wir brauchen, um Oberflächenmodelle von Körpern darzustellen, sind Punkte (Vertices) V, Kanten (Edges) E und Flächen (Faces) F. Will man nun Informationen zusammenstellen, so ergeben sich genau neun verschiedene Fragen:

- | | | |
|-----------|----------------------------|----|
| 1. Vertex | alle benachbarten Vertices | VV |
| 2. Vertex | alle benachbarten Edges | VE |
| 3. Vertex | alle benachbarten Faces | VF |
| 4. Edge | alle benachbarten Vertices | EV |
| 5. Edge | alle benachbarten Edges | EE |
| 6. Edge | alle benachbarten Faces | EF |
| 7. Face | alle benachbarten Vertices | FV |
| 8. Face | alle benachbarten Edges | FE |
| 9. Face | alle benachbarten Faces | FF |

bzw.:

Abb. 1.2:
Nachbarschaftsbeziehungen in
Oberflächenmodellen



Schnell wird klar, dass nicht alle 9 Informationen in einer Datenstruktur abgelegt werden können, wenn diese Datenstruktur noch mit einem erträglichen Aufwand verwaltet und gespeichert werden soll.

Betrachten wir noch einmal die beiden knotenbasierten Datenstrukturen. Die Information, welche Fläche welche Vertices besitzt, ist explizit abgelegt. Will man jedoch alle zu einem Vertex benachbarten Flächen wissen, so muss die komplette Flächenliste nach diesem Vertex durchsucht werden. Diese Abfrage ist also von der Größe des abgespeicherten Modells abhängig und kann unter Umständen lange dauern.

Es stellt sich also die Frage, welche dieser 9 Nachbarschaftsinformationen bestmöglich ausreichen, um einen möglichst kleinen Aufwand bei der Suche nach den anderen Informationen zu haben.

Die Vielzahl der Datenstrukturen unterscheidet sich also in der Zahl der explizit abgelegten (Nachbarschafts-)Informationen. Datenstrukturen können daher bezüglich Speicheraufwand und Komplexität der Algorithmen zur Nachbarschaftssuche klassifiziert werden. Wir gehen hier auf die wichtigsten Datenstrukturen ein. Eine genauere Untersuchung ist in [NB94] zu finden.

1.1.1.2

Kantenorientierte Datenstrukturen

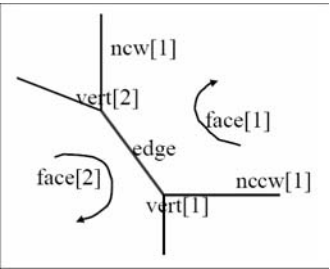
Bei einer Suche nach einem Ausweg beobachten wir, dass die wesentliche Information über die Form von Körpern in den Kanten des Polygonnetzes steckt. Eine Fläche ist z. B. genau durch einen geschlossenen Kantenzug gegeben. Es liegt also nahe, dreidimensionale Objekte durch eine Datenstruktur zu definieren, die als wesentliche Einträge Kanten nutzt.

Im Jahre 1975 stellte Bruce G. Baumgart [Bau75] eine solche kantenorientierte Datenstruktur vor. Folgende Überlegungen führen zu dieser Winged-Edge-Datenstruktur:

Winged-Edge-Datenstruktur

- Jede Kante besitzt genau 2 Nachbarflächen.
- In der Orientierung der Fläche gibt es für jede Kante genau eine vorhergehende und nachfolgende Kante.
- Deswegen der Name dieser Datenstruktur: An jeder Kante hängen zwei „Flügel“ ncw (next clockwise) und nccw (next counterclockwise).
- Sind die Flächen konsistent orientiert, kommt jede Kante einmal + und – vor.

Abb. 1.3:
Winged-Edge-Datenstruktur



Für unser Würfel-Testmodell ergeben sich die in Abb. 1.4 dargestellten Einträge in die abzuspeichernde Datenstruktur.

Die Vorteile dieser Datenstruktur werden sofort klar, wenn man sich z. B. die Aufgabe stellt, eine beliebige Fläche aus dem obigen Datensatz zu zeichnen. Wir wollen versuchen, die fünfte Fläche (Index 4 der Facelist) zu zeichnen:

Beginnen werden wir wie angegeben mit Kante e11, die wir positiv, d. h. von Vstart (Vertex 7) zu Vend (Vertex mit Index 4) durchlaufen. Der Eintrag ncw, für die Kante 11, ist mit e4 angegeben (ncw beim positiven Durchlauf), welche Vstart 0 und Vend 4 besitzt. Da wir nun aber mit der ersten Kante bei Vertex 4 angekommen sind, müssen wir, um e4 zu zeichnen, diese Kante in negativer Richtung, d. h. von Vend nach Vstart, durchlaufen. Also folgt als nächste zu zeichnende Kante nccw e3 usw.

Nach Durchlaufen der vierten Kante stellen wir fest, dass wir wiederum als folgende Kante e11 zeichnen müssten, die jedoch schon zu Beginn verwendet wurde. Wir sind also einen geschlossenen Kantenzug abgelaufen und haben somit die Fläche f4 durch „lokales Umrunden“ gezeichnet.

Damit haben wir gleichzeitig die Nachbarschaftsinformation FE (die zu einer Fläche inzidenten Kanten) bereitgestellt. Die Abfrage dauert also so lange, wie man braucht, um einmal die Fläche zu umrunden, ist damit nur von der lokalen Eigenschaft abhängig und hängt somit nicht

Abb. 1.4:
Beispiel für
Winged-Edge-Datenstruktur

Vertexliste	
0	0,0 0,0 0,0
1	1,0 0,0 0,0
2	1,0 1,0 0,0
3	0,0 1,0 0,0
4	0,0 0,0 1,0
5	1,0 0,0 1,0
6	1,0 1,0 1,0
7	0,0 1,0 1,0

Edgelliste				
e	vstart	vend	ncw	nccw
0	0	1	e1	e4
1	1	2	e2	e5
2	2	3	e3	e6
3	3	0	e0	e7
4	0	4	e8	e3
5	1	5	e9	e0
6	2	6	e10	e1
7	3	7	e11	e2
8	4	5	e5	e11
9	5	6	e6	e8
10	6	7	e7	e9
11	7	4	e4	e10

Facelliste	
0	e0 +
1	e8 +
2	e5 +
3	e6 +
4	e11 +
5	e8 -

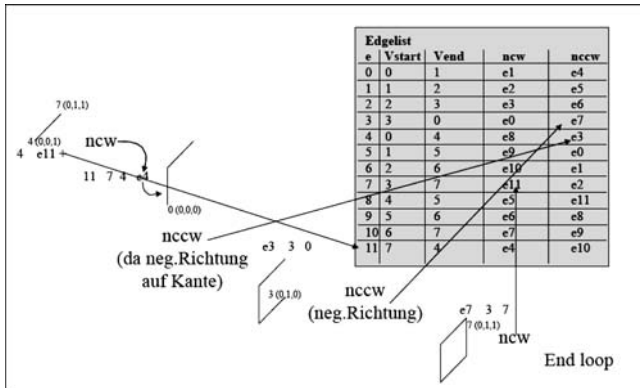


Abb. 1.5:
Bestimmung der benachbarten Kanten
einer Fläche durch lokales Umrunden

von der Größe des gesamten Polygonnetzes ab. Die Antwort dieser Nachbarschaftsabfrage ist also in konstanter Zeit zu erwarten!

Jedoch sind die anderen Nachbarschaftsabfragen wie z.B. FF oder EF leider noch immer nur mit Durchsuchen der gesamten Flächen und Kantenliste zu lösen. Um diese letzten Nachbarschaftsbeziehungen auch noch in konstanter Zeit abfragen zu können, betrachten wir die folgend beschriebene Full-Winged-Edge-Datenstruktur.

Die Winged-Edge-Datenstruktur kann auf interessante Weise erweitert werden: Jede Kante besitzt genau zwei Nachbarflächen. Anders herum wird eine Kante bezüglich einer Fläche nur in einer Richtung durchlaufen. Daher konnten wir in der Winged-Edge-Datenstruktur + oder - (ein Bit) nutzen, um genau die Fläche anzugeben. Legt man diese Information nun noch in die Kantenliste, so kann auf das Bit in der Flächenliste verzichtet werden: Wir erhalten die sog. Full-Winged-Edge-Datenstruktur, bei der die Kanten zusätzlich Information über die zugehörige Fläche bei positivem und negativem Durchlauf enthalten.

Bei der Full-Winged-Edge-Datenstruktur können alle Nachbarschaftsinformationen durch lokales Umrunden, d.h. in konstanter Zeit $O(1)$, berechnet werden. Die Abfragen E^* sind jedoch wesentlich langsamer als F^* .

Full-Winged-Edge-Datenstruktur

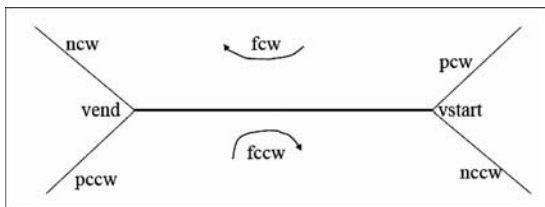
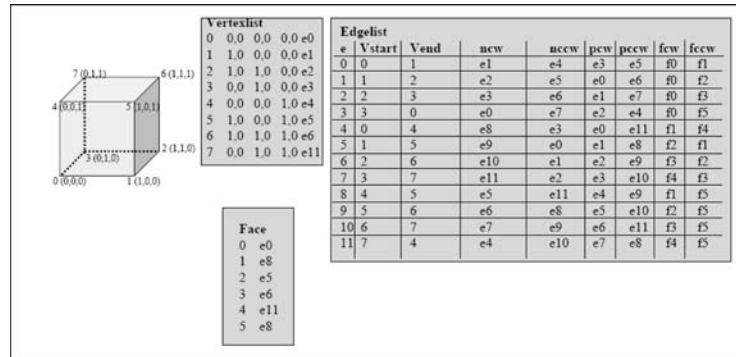


Abb. 1.6:
Full-Winged-Edge-Datenstruktur

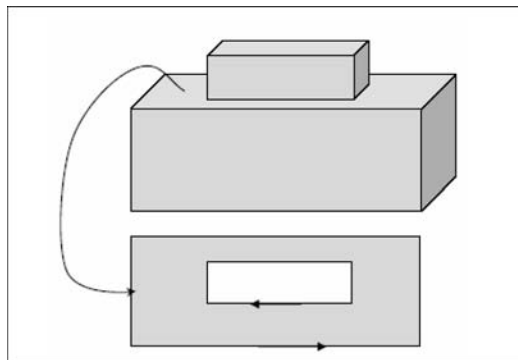
Abb. 1.7:
Beispiel für
Full-Winged-Edge-Datenstruktur



Trotz der großen Vorteile der Full-Winged-Edge-Datenstruktur existiert noch ein gravierendes Problem: Die bisher beschriebenen Edge-Datenstrukturen gehen davon aus, dass eine Fläche durch nur *eine* Randkuve (d. h. eine Folge von Kanten) definiert ist.

Es gibt aber viele Körper, die Flächen mit Löchern haben, wie im folgenden Beispiel. Die obere Fläche, mit aufgesetztem kleinerem Quader, besitzt zwei Randkurven:

Abb. 1.8:
Fläche mit Loch



Eine mögliche Lösung ist, eine zusätzliche Kante einzufügen. Diese Kante hat nun die gleiche Fläche in den Einträgen fcw und fccw.

Eine zweite Möglichkeit ist in obigem Beispiel dargestellt. Die obige Fläche hat *zwei* Randkurven, jedoch mit unterschiedlichen Umlaufrichtungen (zwei Einträge in der Flächenliste).

Einen dritten Ausweg bietet die Einführung von sogenannten Halbkanten [Mä88]. Eine Kante kann prinzipiell in zwei Richtungen durchlaufen werden. Eine Halbkante ist also definiert durch zwei Punkte und eine Durchlaufrichtung. Daher braucht man 2 Halbkanten

für eine volle Kante. Eine Darstellung von Halbkantendatenstrukturen findet man in [ES97, Mä88].

1.1.2

Kurven

Sind die Oberflächen der darzustellenden Objekte nicht eben und eher krummliniger, organischer Natur, so braucht man eine hohe Anzahl von Geradenstücken oder ebenen Flächen, um diese gut zu approximieren. Ihre weitere Bearbeitung wird dadurch langsam und aufwendig. Jedoch ist dies nicht der alleinige Nachteil: Wo kommen die vielen Punkte der Annäherungskurven und -ebenen zu liegen, d. h., wie ermitteln wir deren Koordinaten?

Nehmen wir das einfache Beispiel einer exakten Kreislinie in der Ebene. Annäherungen wären ein Quadrat, ein Sechs-, Acht- oder n -Eck. Einige Koordinaten der Linienpunkte können mit Zirkel und Lineal konstruiert werden. Sollen jedoch die Punkte von einem 1000-Eck angegeben werden, brauchen wir eine „Formel“ zur Berechnung. Diese Formel kann leicht mit den „Kreisfunktionen“ Sinus und Kosinus ausgedrückt werden.

Geraden- und Ebenenstücke reichen also nur eingeschränkt aus, um dreidimensionale Objekte zu gestalten. Besser wäre es, wenn ein paar wenige Punkte (sog. Kontrollpunkte) genügen würden, um den Verlauf der Fläche mit seinen vielen Punkten zu steuern. Zwei Herangehensweisen finden sich in der Computergrafik: Zum einen sind dies parametrisierte Kurven und Flächen und zum anderen eine Beschreibung der Objektoberfläche durch Angabe von Punkteigenschaften (implizite Flächen) der Oberfläche.

1.1.2.1

Monome und parametrisierte Kurven

Wir wollen uns zuerst auf den einfacheren Kurvenfall beschränken. Was ist nun eine sog. parametrisierte Kurve?

Definition: Sei $I=[a,b] \subset \mathbb{R}$ ein Intervall. Eine Abbildung $g:I \rightarrow \mathbb{R}^3$ heißt parametrisierte Kurve:

Parametrisierte Kurve

$$g(t) = \begin{pmatrix} x(t) \\ y(t) \\ z(t) \end{pmatrix} \in \mathbb{R}^3, \text{ für alle } t \in [a,b]$$

Eine parametrisierte Kurve g heißt *n -mal stetig differenzierbar*, falls jede Koordinatenfunktion $x(t)$, $y(t)$ und $z(t)$ n -mal stetig differenzierbar

ist, und man schreibt $g \in C^n(I, \mathbb{R}^3) \lim_{x \rightarrow \infty}$. Eine parametrisierte Kurve heißt *regulär* (in t_0), falls

$$g'(t_0) = \begin{pmatrix} x'(t_0) \\ y'(t_0) \\ z'(t_0) \end{pmatrix} \neq 0$$

Der Gradient $g'(t) \in \mathbb{R}^3$ ist die Tangente in t an g und gibt die Momentangeschwindigkeit (in Betrag und Länge) beim Durchlaufen der Kurve an. Die *(Bogen-)Länge einer Kurve* kann durch

$$S(u) = \int_a^u \|g'(s)\| ds$$

berechnet werden.

Zwei reguläre Kurven, $g_1: [a, b] \rightarrow \mathbb{R}^3$ und $g_2: [s, t] \rightarrow \mathbb{R}^3$, heißen *äquivalent*, wenn es eine bijektive differenzierbare Abbildung $\varphi: [a, b] \rightarrow [s, t]$ mit $\varphi'(u) > 0$ gibt, so dass

$$g_1(u) = g_2(\varphi(u))$$

gilt.

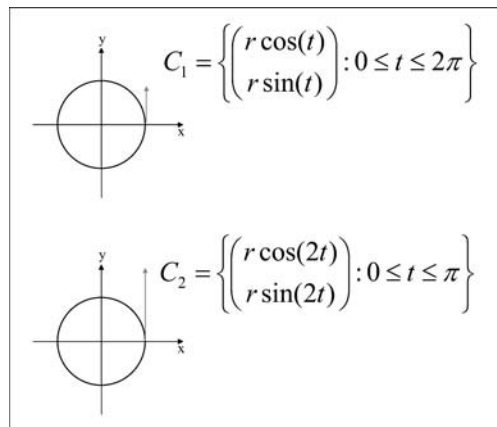
Wir sagen in diesem Fall auch, dass g_2 durch φ *reparametrisiert* wurde, und nennen φ einen richtungserhaltenden Parameterwechsel oder eine *Reparametrisierung* von g_1 .

Parametrische Stetigkeit

Zwei Kurven können aneinandergefügt werden. Zwei n -mal stetig differenzierbare reguläre Kurven $g_1: [a, b] \rightarrow \mathbb{R}^3$ und $g_2: [s, t] \rightarrow \mathbb{R}^3$ schließen an der Stelle b, s C^n -stetig aneinander (*parametrisch stetig*), genau dann, wenn

$$g_1^{(k)}(b) = g_2^{(k)}(s) \quad \text{für alle } 0 \leq k \leq n.$$

Abb. 1.9:
Zwei äquivalente Kurven



Für Animationen und sonstige Anwendungen der Computergrafik ist diese Definition des parametrisch stetigen Übergangs zu strikt. Es zählt vielmehr der visuell-glatte Übergang. Daher definieren wir (*Geometrisch stetiger Anschluss* oder *G^n -stetiger Übergang*): Zwei n -mal stetig differenzierbare reguläre Kurven $g_1: [a, b] \rightarrow \mathbb{R}^3$ und $g_2: [s, t] \rightarrow \mathbb{R}^3$ schließen an der Stelle b, s G^n -stetig aneinander, falls es eine zu g_1 äquivalente Kurve $r: [a, b_1] \rightarrow \mathbb{R}^3$ gibt, so dass r_0 und g_2 an der Stelle b_1, s C^n -stetig aneinanderschließen.

Also existiert eine bijektive differenzierbare Abbildung $\varphi: [a, b] \rightarrow [a_1, b_1]$ mit $\varphi'(.) \geq 0$, so dass $r = g_1 \circ \varphi$.

Damit ergibt sich mit der Kettenregel für die Ableitungen:

$$\begin{aligned} g_2(s) &= r(b_1) = g_1(\varphi(b)) \\ g_2'(s) &= g_1'(\varphi(b))\varphi'(b) \\ g_2''(s) &= g_1''(\varphi(b))\varphi'(b)^2 + g_1'(\varphi(b))\varphi''(b) \end{aligned}$$

Also heißt G^0 - und G^1 -Stetigkeit:

- G^0 -Stetigkeit entspricht der C^0 -Stetigkeit.
- G^1 -Stetigkeit: Der Übergang ist stetig und beide Tangentenvektoren haben am Übergang die gleiche Richtung.

Der dreidimensionale Raum \mathbb{R}^3 steht hier stellvertretend für allgemein d -dimensionale Räume \mathbb{R}^d . Es sind dann natürlich entsprechend mehr oder weniger Koordinatenfunktionen zu definieren.

Man sieht, dass zur Angabe einer parametrisierten Kurve neben dem Intervall die Angabe von Koordinatenfunktionen genügt, wie in dem in Abb. 1.10 dargestellten Beispiel einer ebenen parametrisierten Kurve.

Im Prinzip darf man für die Koordinatenfunktionen jede mögliche Funktion wählen, jedoch besitzen vor allem die Polynomfunktionen besonders schöne (aber auch schlechte) Eigenschaften. Insbesondere, und das ist in der Informatik besonders wichtig, sind sie sehr schnell

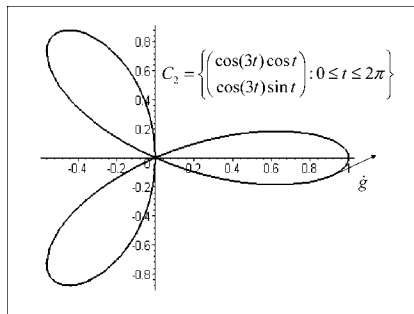


Abb. 1.10:
Beispiel einer ebenen parametrisierten Kurve

auszuwerten. Dies ist ein Grund, dass man sich in der Computergrafik vor allem diesen Funktionsarten verschrieben hat.

Definition: Die Menge

$$P^n([a,b]) = \left\{ \sum_{i=0}^n \alpha_i t^i, \text{ für } \alpha_i \in \mathbb{R}^3 \text{ und } t \in [a,b] \right\}$$

aller Polynome vom Grad n ist ein endlich dimensionaler Unterraum von $C([a,b], \mathbb{R}^3)$, dem Raum der stetigen, parametrisierten Kurven. Die Vektoren α_i heißen Taylorkoeffizienten des Polynoms.

*Monobasis
Taylorbasis*

Der Unterraum $P^n[a,b]$ hat Dimension $n+1$ und die Menge der Monome $\{1, t, t^2, \dots, t^n\}$ bildet eine Basis, die sog. Monom- oder Taylorbasis.

Wie man leicht sehen kann, ist es egal, ob man drei polynomielle Koordinatenfunktionen mit reellen Koeffizienten oder vektorielle Koeffizienten wählt: Ein Beispiel:

$$p(t) = \begin{pmatrix} 1+t+1.5t^2-3t^3 \\ 1.5-t+3t^2 \\ 3+6t-t^2+6t^3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1.5 \\ 3 \end{pmatrix} + \begin{pmatrix} 1 \\ -1 \\ 6 \end{pmatrix} t + \begin{pmatrix} 1.5 \\ 3 \\ -1 \end{pmatrix} t^2 + \begin{pmatrix} -3 \\ 0 \\ 6 \end{pmatrix} t^3$$

Hornerschema

Polynomfunktionen sind besonders schnell auszurechnen, d. h., ein Funktionswert an einer vorgegebenen Intervallstelle t_0 kann durch das sog. Hornerschema ausgewertet werden. Das vollständige Hornerschema erlaubt sogar die Berechnung der Ableitungen:

Ist ein Polynom in Monomdarstellung

$$p(t) = \alpha_n t^n + \alpha_{n-1} t^{n-1} + \dots + \alpha_2 t^2 + \alpha_1 t + \alpha_0$$

gegeben, so errechnet sich leicht der Funktionswert $p(t_0)$ an der Stelle t_0 nach folgendem Schema (Hornerschema): Von oben nach unten wird mit t_0 multipliziert – von links nach rechts. Der Funktionswert steht rechts in der Tabelle.

	α_n	α_{n-1}	α_{n-2}	...	α_0
+	0	$\alpha_n t_0$	$\alpha_{n-1} t_0$...	α_0
$\cdot t_0$	$\alpha_n t_0$	$(\alpha_{n-1} + \alpha_n t_0) t_0$	$\alpha_{n-2} t_0$...	α_0
					$p(t_0)$

Das vollständige Hornerschema sei an einem Beispiel demonstriert: Gesucht sei der Funktionswert des folgenden Polynoms an der Stelle $t_0 = 2$

$$p(t) = t^4 + 2t^3 - 3t^2 - 7$$

Dann ergibt sich:

$$\begin{array}{r|rrrrr}
 & 1 & 2 & -3 & 0 & -7 \\
 + & 0 & 2 & 8 & 10 & 20 \\
 \hline
 \cdot 2 & 1 & 4 & 5 & 10 & 13 = p(2) \\
 + & 0 & 2 & 12 & 34 & \\
 \hline
 \cdot 2 & 1 & 6 & 17 & 44 & \cdot 1! = p'(2) \\
 + & 0 & 2 & 16 & & \\
 \hline
 \cdot 2 & 1 & 8 & 33 & \cdot 2! = 33 \cdot 2 \cdot 1 = p''(2) \\
 + & 0 & 2 & & & \\
 \hline
 \cdot 2 & 1 & 10 & \cdot 3! = 10 \cdot 3 \cdot 2 \cdot 1 = p'''(2) \\
 + & 0 & & & & \\
 \hline
 \cdot 2 & 1 & \cdot 4! = 1 \cdot 24 = p^{(4)}(2)
 \end{array}$$

Die Auswertung eines Polynoms mit dem Hornerschema ist effektiver als die Auswertung nach Funktionsvorschrift.

Wie ist aber unser Programm der Objektmodellierung zu realisieren? Gegeben sei eine komplexe Form und diese soll als parametrisierte Kurve dargestellt werden, am besten noch als Polynomfunktion. Dadurch könnte eine komplexe, kontinuierliche Form durch ein paar wenige Punkte, d.h. die Koeffizienten α_p vollkommen definiert werden. Schnell stellt sich die Frage, ob das überhaupt möglich ist und, falls ja, wie die Koeffizienten α_i zu berechnen sind?

1.1.2.2

Interpolationsaufgabe, Monominterpolation

Wir kennen jedoch ein positives Resultat, welches uns auf unserem Weg bestärkt. Zunächst noch einmal die Aufgabe:

Gegeben: Eine Reihe von (Mess-)Punkten (u_i, P_i) , $i=0, \dots, n$, mit $u_i \in \mathbb{R}$ und $P_i \in \mathbb{R}^3$.

Gesucht: (Interpolationsaufgabe IA) Eine Funktion f , welche der Bedingung genügt, dass

$$f(u_i) = P_i \quad \text{für alle } i=0, \dots, n$$

die Kurve f also zu den vorgegebenen Parameterwerten u_i durch die vorgegebenen Punkte P_i verläuft. u_i heißen Stützstellen, der Vektor (u_0, \dots, u_n) Stützstellenvektor und P_i heißt Stützpunkt. Die Paare (u_i, P_i) heißen Knoten der IA.

Wie bereits erwähnt, gibt es zu diesem zentralen Problem eine positive Antwort. Diese Tatsache ist nicht nur für die Modellierung von Objekten von Bedeutung, sondern das nachfolgende Theorem hat vielfältige Anwendungen in der Numerik und damit in der Computergrafik.

Theorem: Es seien $n+1$ paarweise verschiedene Knoten (u_i, P_i) , $i=0, \dots, n$ mit $u_i \in \mathbb{R}$ und $P_i \in \mathbb{R}^d$ gegeben. Dann existiert genau ein Polynom p vom Grad $\leq n$, mit $n+1$ Koeffizienten $c_i \in \mathbb{R}^d$, so dass

$$p(t) = \sum_{i=0}^n c_i t^i \quad \text{und} \quad p(u_i) = P_i \quad \text{für alle } i = 0, 1, \dots, n$$

Beweis: Nach Voraussetzung gilt:

$$\begin{aligned} c_0 + c_1 u_0 + c_2 u_0^2 + \dots + c_n u_0^n &= P_0 \\ c_0 + c_1 u_1 + c_2 u_1^2 + \dots + c_n u_1^n &= P_1 \\ &\vdots \\ c_0 + c_1 u_n + c_2 u_n^2 + \dots + c_n u_n^n &= P_n \end{aligned}$$

In Vektorschreibweise ergibt sich dann:

$$\begin{pmatrix} 1 & u_0 & u_0^2 & \dots & u_0^n \\ 1 & u_1 & u_1^2 & \dots & u_1^n \\ \vdots & & & & \vdots \\ 1 & u_n & u_n^2 & \dots & u_n^n \end{pmatrix} \begin{pmatrix} c_0 \\ c_1 \\ \vdots \\ c_n \end{pmatrix} = \begin{pmatrix} P_0 \\ P_1 \\ \vdots \\ P_n \end{pmatrix}$$

$$M \cdot \vec{C} = \vec{P}$$

Die Matrix M hat die bekannte Form der sog. Van der Monde'schen Matrix. Die Zeilen sind jeweils fortschreitende Potenzen von u_i . Sind die Stützstellen u_i paarweise verschieden, ist die Matrix M invertierbar und das Gleichungssystem ist lösbar. Die unbekannten Koeffizienten c_i sind dadurch eindeutig gegeben.

Diese Methode (Interpolation mit Monomen) liefert die Existenz einer Lösung der Interpolationsaufgabe und deren Eindeutigkeit. Das Problem ist aber, die unbekannten Koeffizienten c_i zu finden. Fügt man einen neuen Knoten ein (hat man einen zusätzlichen Interpolationspunkt), so muss die volle Matrix invertiert werden. Eine Lösung des Problems ist die Newton-Interpolation [Br00].

Schwieriger als das eben beschriebene Problem ist jedoch folgendes Phänomen: Stellen wir uns eine Kurve vor, welche wir durch Monominterpolation gewonnen haben:

$$p(t) = \sum_{i=0}^n c_i t^i \quad \text{und} \quad p(u_i) = P_i \quad \text{für alle } i = 0, 1, \dots, n$$

Wobei also die Knoten (u, P_i) , $i=0, \dots, n$, vorgegeben und die Koeffizienten c_i durch Lösen des Gleichungssystems gewonnen wurden. Wird nun auf diese Kurve eine affine Transformation (z. B. eine Rotation) angewandt, so stellt sich die Frage, ob die Kurve wieder durch die nun ebenfalls rotierten Punkte P_i geht. Leider ist dem nicht so!

Untersuchen wir diese Eigenschaft einmal genauer: Es sei Φ eine Abbildung (z. B. Translation, Rotation, Streckung) in \mathbb{R}^d . Φ heißt eine affine Abbildung, wenn für alle Skalare $\lambda_0, \lambda_1, \dots, \lambda_n \in \mathbb{R}$ mit $\sum \lambda_i = 1$ und alle Punkte P_0, \dots, P_n folgende Gleichung gilt:

$$\Phi\left(\sum_i \lambda_i P_i\right) = \sum_i \lambda_i \Phi(P_i)$$

Nun sind aber $c_0, c_1, \dots, c_n \in \mathbb{R}^d$ und $p(u)$ eine Polynomkurve mit $p(u) = \sum_{0 \leq i \leq n} c_i f_i(u)$. Diese Darstellung ist genau dann affin invariant, wenn

$$\sum_{i=0}^n f_i(u) = 1 \quad \forall u \in [a, b], \text{ denn dann gilt}$$

$$\Phi\left(\sum_{i=0}^n f_i(u) P_i\right) = \sum_{i=0}^n f_i(u) \Phi(P_i) \quad \forall u \in [a, b]$$

Anschaulich erhält man den gleichen Kurvenverlauf, unabhängig davon, ob man zuerst die Stützpunkte affin verschiebt und dann die Kurve berechnet, oder ob man zuerst die Kurve berechnet und dann die einzelnen Kurvenpunkte affin transformiert.

Für Monome gilt jedoch im Allgemeinen $\sum u^i \neq 1$ für $u \in [a, b]$, daher ist die Polynomdarstellung mit Monomen (Taylorbasis) *nicht* affin invariant.

Betrachten wir dazu ein Beispiel:

Wir nehmen an, dass wir den Weg eines berühmten Bergsteigers auf den Gipfel des Mount Everest modellieren wollen. Er befindet sich zu gewissen Zeiten u_i in den Basislagern P_i , die über GPS bestimmt wurden. Sein Weg wird durch Bestimmung der Taylorkoeffizienten computergrafisch modelliert. Leider dreht sich die Erde in 24 Stunden einmal um sich selbst und damit wird in einem globalen, geozentrischen Koordinatensystem die Kurve unseres Bergsteigers auf den Gipfel rotiert. Die Kurve muss immer dieselbe sein und es genügt nun nicht, einfach die Taylorkoeffizienten zu rotieren, um die Kurve zeichnen zu können! Um die Kurve durch die Basislager zu zwingen, müssen wir, Bild für Bild, zur Bestimmung der Taylorkoeffizienten c_i ein Gleichungssystem lösen.

Außerdem ist es nicht möglich, die Koeffizienten c_i irgendwie mit dem Verlauf der Kurve in Beziehung zu setzen. Sie haben keine geometrische Bedeutung für den Kurvenverlauf.

Dies alles sind schwerwiegende Probleme bei der Interpolation mit Monomen. Die einzige Lösung aus diesem Dilemma ist, eine neue Menge von Basisvektoren für den Raum aller Polynomkurven zu finden, die die gewünschten Eigenschaften besitzt:

- Einfache Berechnung der Koeffizienten c_i
- Geometrische Deutung der Koeffizienten c_i
- Affine Invarianz der Kurve
- Einfache Berechnung der Kurve $p(t)$

1.1.2.3

Interpolationsaufgabe, Lagrangekurven

Die ersten drei Eigenschaften sind leicht durch folgenden Ansatz zu erreichen:

$$L_i^n(u) = \prod_{j=0, j \neq i}^n \frac{u - u_j}{u_i - u_j} = \frac{(u - u_0)(u - u_1) \cdots (u - u_{i-1})(u - u_{i+1}) \cdots (u - u_n)}{(u_i - u_0)(u_i - u_1) \cdots (u_i - u_{i-1})(u_i - u_{i+1}) \cdots (u_i - u_n)}$$

Es seien $n+1$ Knoten (u_i, P_i) gegeben, mit $i=0, \dots, n$ und $u_0 < u_1 < \dots < u_n$. Weiter sei $I=[a, b]$ ein Intervall. Für jedes $i=0, \dots, n$ definieren wir das folgende Polynom vom Grad n (i -tes Lagrange-Polynom vom Grad n):

Die Interpolationsaufgabe ist durch

$$p(u) = \sum_{i=0}^n L_i^n(u) P_i, \quad u \in [a, b]$$

eindeutig gelöst. Außerdem ist $\sum_{0 \leq i \leq n} L_i^n(u) = 1$, d. h., die Kurve ist affin invariant.

Diese Tatsache ist leicht einzusehen. Zentral ist die Eigenschaft der Lagrangepolynome, dass

$$L_i^n(u_k) = \delta_{ik} = \begin{cases} 1 & i = k \\ 0 & \text{sonst} \end{cases}$$

Damit scheinen wir alle Punkte erreicht zu haben. Leider ist die Kurve nicht einfach zu ermitteln und muss nach der Definition gerechnet werden. Kommt ein neuer Knotenpunkt hinzu, so ändert sich komplett die Form der Lagrangepolynome.

Beispiel: Gegeben seien drei Knoten (u_i, P_i) $i=0,1,2$ mit $(0, (1|0))$, $(1, (-1|0))$ und $(3, (3|3))$. Die Lagrange Polynome sind dann

$$L_0^2(u) = \frac{(u-1)(u-3)}{(0-1)(0-3)} = \frac{u^2 - 4u + 3}{3}$$

$$L_1^2(u) = \frac{u(u-3)}{-2}$$

$$L_2^2(u) = \frac{u(u-1)}{6}$$

und die Interpolationskurve ist durch

$$p(u) = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \frac{u^2 - 4u + 3}{3} + \begin{pmatrix} -1 \\ 0 \end{pmatrix} \frac{u(u-3)}{-2} + \begin{pmatrix} 3 \\ 3 \end{pmatrix} \frac{u(u-1)}{6}$$

gegeben. Der Kurvenverlauf ist in Abb. 1.11 dargestellt.

Versucht man mit diesen Kurven Objekte zu modellieren, so stellt man leicht fest, dass diese Kurven ihre Form stark ändern, wenn ein (Interpolations-)Punkt verschoben wird. Leider ändert sich die Form der Kurve auch nicht immer in Richtung der Verschiebung des Punktes. Zudem kann am Anfang und Ende der Kurve die Steigung nicht vorgegeben werden, was ein „glattes“ Aneinanderfügen von Kurvenstücken sehr schwer macht.

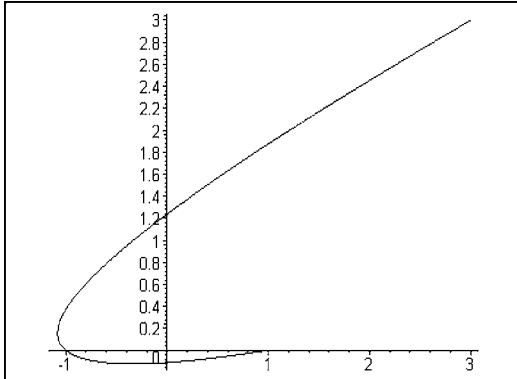


Abb. 1.11:
Beispiel einer Lagrangekurve

1.1.2.4

Interpolationsaufgabe, Hermitekurven

Da Kurvensegmente in der Monom- und Lagrangebasis in der Regel nur mit C^0 -Stetigkeit aneinandergefügt werden können, sind diese Basen nicht für stückweise C^1 -stetige Interpolation geeignet.

Für diesen Fall bieten sich z. B. *kubische Hermite-Splines* an:

Sind die Parameterwerte $t_0, t_1, \dots, t_n \in \mathbb{R}$, die dazugehörigen Punkte $p_0, p_1, \dots, p_n \in \mathbb{R}^d$ und in den Knotenpunkten die Ableitungen $m_0, m_1, \dots, m_n \in \mathbb{R}^d$ gegeben, so erfüllen folgende Segmente $q_i, i=0, \dots, n-1$ die Interpolationsaufgabe:

$$q_i(u) = p_i H_0^3(s_i) + \Delta_i m_i H_1^3(s_i) + \Delta_i m_{i+1} H_2^3(s_i) + p_{i+1} H_3^3(s_i), u \in [t_i, t_{i+1}]$$

wobei

$$s_i = \frac{(u - t_i)}{(t_{i+1} - t_i)} \quad \Delta_i = t_{i+1} - t_i$$

Die Polynome H_0^3, H_1^3, H_2^3 und H_3^3 heißen Hermitepolynome vom Grad 3. Den Parameter u nennt man *globalen* Parameter des Splines und Parameter s_i heißt *lokaler* Parameter für das Intervall $[t_i, t_{i+1}]$. Die vier kubischen Hermitepolynome über dem Intervall $[0,1]$ sind in Abb 1.12 definiert.

Abbildung 1.13 zeigt einige Beispiele von kubischen Hermitekurven.

Dennoch gibt es bei dieser Art von Interpolationskurven ein Problem: Oft hat man die Tangenten m_i nicht alle explizit gegeben.

Abb. 1.12:
Die vier kubischen Hermitepolynome

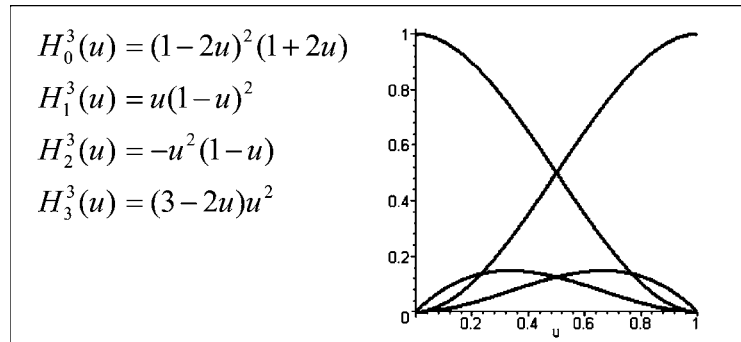
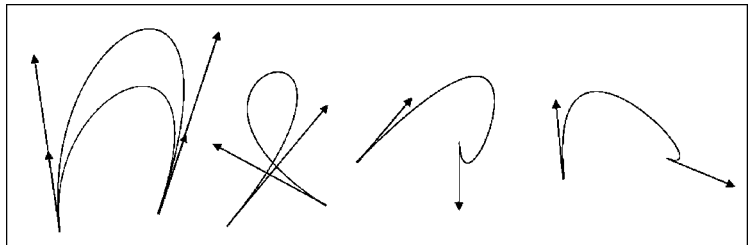


Abb. 1.13:
Beispiele kubischer Hermitekurven



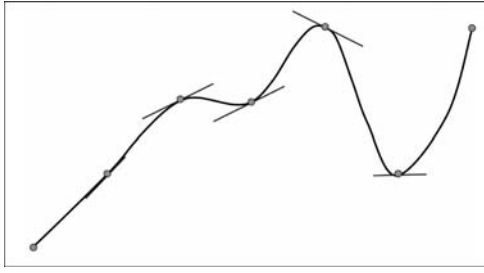


Abb. 1.14:
Vorgabe der Tangenten an allen
Stützstellen

Trick: Berechne die fehlenden Tangenten aus Stützstellen. Die Tangentenrichtung m_i im Punkt P_i wird parallel zur Sehne durch P_{i-1} und P_{i+1} gewählt. Der entstehende Interpolant wird auch als *Catmull-Rom-Spline* bezeichnet (FMILL-Schätzung).

Mit Hermite-Interpolationskurven ist einfach und intuitiv zu modellieren. Jedoch ist die explizite Angabe der Steigungstangenten der Kurve in der Praxis oft ein Problem.

1.1.2.5

Bézierkurven

Wieder sind wir auf der Suche nach einem Satz von Polynomfunktionen als Basis des $P'([s,t])$. Dafür betrachten wir das Intervall $[s,t]$ und berechnen:

$$\begin{aligned}(t-s)^n &= (t-u+u-s)^n \\ &= \sum_{i=0}^n \binom{n}{i} (t-u)^i (u-s)^{n-i}\end{aligned}$$

Wir können abkürzend schreiben:

$${}_s^t B_i^n(u) = \frac{1}{(t-s)^n} \binom{n}{i} (t-u)^i (u-s)^{n-i}, \quad i=0, \dots, n.$$

Dies sind $n+1$ Polynome vom Grad n über dem Intervall $[s,t]$ und aus der Definition ergibt sich direkt, dass

$$1 = \sum_{i=0}^n {}_s^t B_i^n(u), \quad \text{für alle } u \in [s,t].$$

Diese $n+1$ Polynome heißen Bernsteinpolynome vom Grad n über dem Intervall $[s,t]$. Häufig wählt man das Einheitsintervall $[0,1]$ und schreibt einfacher B_i^n . Es kann gezeigt werden, dass diese $n+1$ Poly-

Bernsteinpolynome

nome linear unabhängig sind und daher eine Basis des Polynomraums $P^n([s,t])$ bilden.

Viele weitere Eigenschaften der Bernsteinpolynome sind einfach aus der Definition nachzurechnen:

$$\sum_{i=0}^n {}^t B_i^n(u) = 1, \quad \text{für } u \in [s,t] \quad \text{Partition der Eins}$$

$${}^t B_i^n(u) \geq 0, \quad \text{für } u \in [s,t] \quad \text{Positivität}$$

$${}^t B_i^n(u) = \frac{u-s}{t-s} {}^t B_{i-1}^{n-1}(u) + \frac{t-u}{t-s} {}^t B_i^{n-1}(u) \quad \text{Rekursion}$$

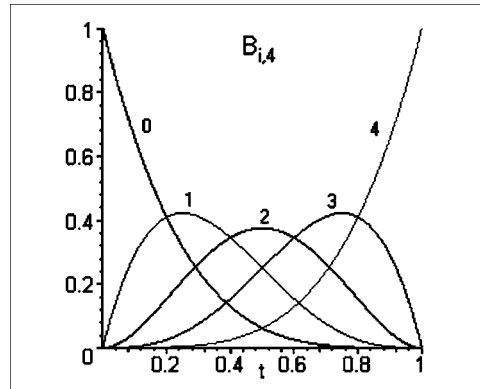
$${}^t B_i^n(u) = {}^t B_{n-i}^n(t-(u-s)) \quad \text{Symmetrie}$$

$${}^t B_i^n(s) = 0 = {}^t B_i^n(t), \quad {}^t B_0^n(s) = 1 = {}^t B_n^n(t), \quad {}^t B_0^n(t) = 0 = {}^t B_n^n(s)$$

$$\max_s {}^t B_i^n(u) = {}^t B_{n-i}^n(i/n)$$

Für den Fall $n=4$, $s=0$ und $t=1$ ergeben sich die folgenden Bernsteinpolynome $B_0^4 = (1-u)^4$, $B_1^4 = 4(1-u)^3u$, $B_2^4 = 6(1-u)^2u^2$ sowie $B_3^4 = 4(1-u)u^3$ und $B_4^4 = u^4$:

Abb 1.15:
Bernsteinpolynome vom Grad 4 über
dem Intervall $[0,1]$



Diese Basisfunktionen sind nun Ausgangspunkt für folgende Definition einer Bézierkurve: Sei mit $I=[s,t]$ ein Intervall und $b_0, b_1, \dots, b_n \in \mathbb{R}^d$ Punkte gegeben, dann heißt

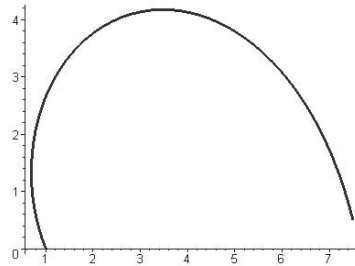
$$p(u) = \sum_{i=0}^n b_i \cdot {}^t B_i^n(u), \quad u \in [s,t]$$

Bézierkurve vom Grad n . Die Punkte b_0, b_1, \dots, b_n heißen Bézierpunkte und definieren zusammen das Bézierpolygon in \mathbb{R}^d .

Beispiel: $b_0=(1|0)$, $b_1=(0|2)$, $b_2=(1|5.5)$, $b_3=(6|5.5)$, $b_4=(7.5|0.5)$ seien Bézierpunkte in der Ebene. Wir wählen das Einheitsintervall als Parametervorrat. Die Bézierkurve ist dann durch

$$p(u)=(1|0) (1-u)^4 + (0|2) 4 (1-u)^3 u + (1|5.5) 6 (1-u)^2 u^2 + (6|5.5) 4(1-u)u^3 + (7.5|0.5) u^4$$

gegeben. Das Schaubild sieht folgendermaßen aus:



Aus den oben genannten Eigenschaften der Bernsteinpolynome können sofort elementare Eigenschaften der Bézierkurven abgeleitet werden.

Die erste Eigenschaft (Partition der Eins) zusammen mit der Positivität der Bernsteinpolynome zeigt direkt die affine Invarianz der Bézierkurven. Außerdem schließen wir daraus auch, dass die Bézierkurve innerhalb der konvexen Hülle der Bézierpunkte verläuft, und zwar als *Konvexkombination der Bézierpunkte*.

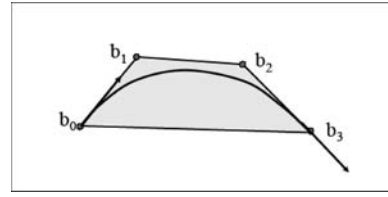
Die Bézierkurve verläuft durch Anfangs- und Endpunkt b_0 und b_n . Dies folgt aus den vorletzten Eigenschaften der Bernsteinpolynome, die die Werte 0 und 1 genau an den Grenzen des Parameterintervalls annehmen.

Betrachtet man die Ableitung einer Bézierkurve am Anfang und Ende, so ist leicht zu zeigen, dass die respektiven Ableitungen durch

$$p'(s) = \frac{n}{t-s}(b_1 - b_0) \text{ und } p'(t) = \frac{n}{t-s}(b_n - b_{n-1})$$

gegeben sind. Dies bedeutet aber, dass die Bézierkurve innerhalb der konvexen Hülle der Bézierpunkte verläuft und tangential an das Bézierpolygon ist.

Abb. 1.16:
Bézierkurve mit Bézierpunkten
und -polygon



Allgemeiner gilt für die i -te Ableitung der Bézierkurve:

$$p^{(i)}(s) = \frac{1}{(t-s)^i} \frac{n!}{(n-i)!} \sum_{j=0}^i \binom{i}{j} (-1)^{i-j} b_j$$

$$p^{(i)}(t) = \frac{1}{(t-s)^i} \frac{n!}{(n-i)!} \sum_{j=0}^i \binom{i}{j} (-1)^j b_{n-j}$$

Konvention: Im Folgenden betrachten wir $s=0$, $t=1$, d.h. $I=[0,1]$ und schreiben einfach $B_i^n(u)$ statt ${}_0B_i^n(u)$. Im Übrigen kann jederzeit durch die folgende Abbildungen

$$\tilde{u} = \frac{u-s}{t-s}, \quad u \in [s,t] \quad \text{bzw.} \quad u = s(1-\tilde{u}) + t\tilde{u}, \quad \tilde{u} \in [0,1]$$

reparametrisiert werden.

1.1.2.6

Der De-Casteljau-Algorithmus

Bei der Monomdarstellung von Polynomfunktionen haben wir das Hornerschema zur Berechnung von Funktionswerten der Kurve kennengelernt. Auch im Falle von Bézierkurven existiert nun ein Berechnungsschema, das die schnelle Berechnung von Kurvenwerten erlaubt:

Das *Schema nach de Casteljau* oder der *De-Casteljau-Algorithmus*: Seien $I=[0,1]$ und $b_0, b_1, \dots, b_n \in \mathbb{R}^d$ gegeben und

$$p(u) = \sum_{i=0}^n b_i B_i^n(u)$$

eine Bézierkurve, dann liefert für jedes $u \in I=[0,1]$ das rekursiv definierte Schema

$$b_i^0(u) = b_i \quad i = 0, \dots, n$$

$$b_i^r(u) = u \cdot b_{i+1}^{r-1}(u) + (1-u) \cdot b_i^{r-1}(u) \quad r = 1, \dots, n, \quad i = 0, \dots, n-r$$

den Funktionswert $p(u) = b_0^n(u)$.

Wir betrachten folgendes Beispiel: Es seien $I=[0,1]$ und die Bézierpunkte $b_0=(1|0)$, $b_1=(0|2)$, $b_2=(1|5.5)$, $b_3=(6|5.5)$ und $b_4=(7.5|0.5)$ gegeben. Wir suchen den Funktionswert $p(u)$ für $u=0.5$:

De-Casteljau-Schema

$$\begin{array}{l}
 b_0^0(0.5) = b_0 = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \quad b_0^1(0.5) = \begin{pmatrix} 0.5 \\ 1 \end{pmatrix} \quad b_0^2(0.5) = \begin{pmatrix} 0.5 \\ 2.125 \end{pmatrix} \quad \begin{pmatrix} 1.25 \\ 3 \end{pmatrix} \quad \begin{pmatrix} 2.40625 \\ 3.34375 \end{pmatrix} \\
 b_1^0(0.5) = b_1 = \begin{pmatrix} 0 \\ 2 \end{pmatrix} \quad b_1^1(0.5) = \begin{pmatrix} 0.5 \\ 3.25 \end{pmatrix} \quad b_1^2(0.5) = \begin{pmatrix} 2 \\ 3.875 \end{pmatrix} \quad \begin{pmatrix} 3.5625 \\ 3.6875 \end{pmatrix} \\
 b_2^0(0.5) = b_2 = \begin{pmatrix} 1 \\ 5.5 \end{pmatrix} \quad b_2^1(0.5) = \begin{pmatrix} 3.5 \\ 5.5 \end{pmatrix} \quad b_2^2(0.5) = \begin{pmatrix} 5.125 \\ 3.5 \end{pmatrix} \\
 b_3^0(0.5) = b_3 = \begin{pmatrix} 6 \\ 5.5 \end{pmatrix} \quad b_3^1(0.5) = \begin{pmatrix} 6.75 \\ 2.5 \end{pmatrix} \\
 b_4^0(0.5) = b_4 = \begin{pmatrix} 7.5 \\ 0.5 \end{pmatrix}
 \end{array}$$

Dieses Schema wird von links nach rechts aufgebaut. Jeweils zwei untereinander stehende Punkte werden mit Koeffizienten s und $(1-s)$ multipliziert und addiert, um einen neuen Wert der nächsten Spalte zu ergeben. Der gesuchte Funktionswert steht ganz rechts am Ende dieses Dreiecksschemas. Also ist

$$p\left(\frac{1}{2}\right) = \begin{pmatrix} 2.41 \\ 3.34 \end{pmatrix}$$

Das Schema wird größer, je mehr Bézierpunkte für die Kurve verwendet werden.

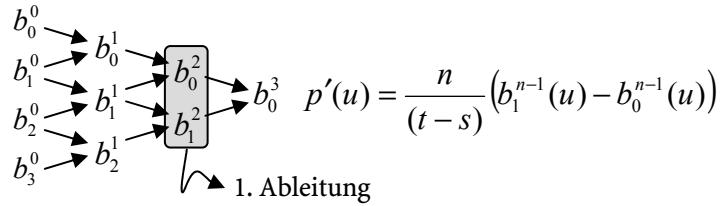
1.1.2.7

Unterteilung und Graderhöhung von Bézierkurven

Diesem Schema ist noch mehr zu entnehmen! Dazu betrachten wir die Ableitungen der Bézierkurve einmal näher. Die i -te Ableitung $p^{(i)}(u)$ ergibt sich aus dem De-Casteljau-Schema in der $(n-i)$ -ten Stufe.

$$p^{(i)}(u) = \frac{1}{(t-s)^i} \frac{n!}{(n-i)!} \sum_{k=0}^i \binom{i}{k} (-1)^{i-k} b_k^{n-i}(u)$$

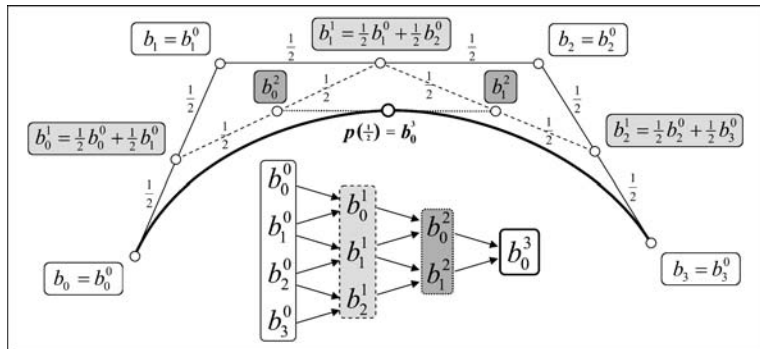
Für die erste Ableitung ist die vorletzte Stufe des Schemas entscheidend:



Das De-Casteljau-Schema dient also nicht nur zur Berechnung von Funktionswerten der Kurve, sondern kann auch für die Berechnung der Ableitungen verwendet werden.

Eine Besonderheit des De-Casteljau-Algorithmus ist seine unmittelbar geometrische Veranschaulichung. Zur einfacheren Konstruktion wollen wir einmal 4 Bézierpunkte in der Ebene betrachten. Die einzelnen Konstruktionsebenen sind in Abb. 1.17 farblich markiert.

Abb. 1.17:
Geometrische Konstruktion nach de
Casteljau



Die geometrische Konstruktion in der Ebene ist exemplarisch für gegebene Bézierpunkte.

Die Strecke b_0^2 und b_1^2 liegt, wie wir uns überlegt haben, tangential zur Bézierkurve. Der Funktionswert $p(1/2)$ ist markiert.

Der De-Casteljau-Algorithmus erlaubt es, eine gegebene Bézierkurve leicht in zwei Bézier-Teilsegmente zu zerlegen. Die neuen Bézierpunkte der Teilsegmente ergeben sich aus dem oberen und unteren Rand des Schemas. Dies kann wieder in der Konstruktion dargestellt werden:

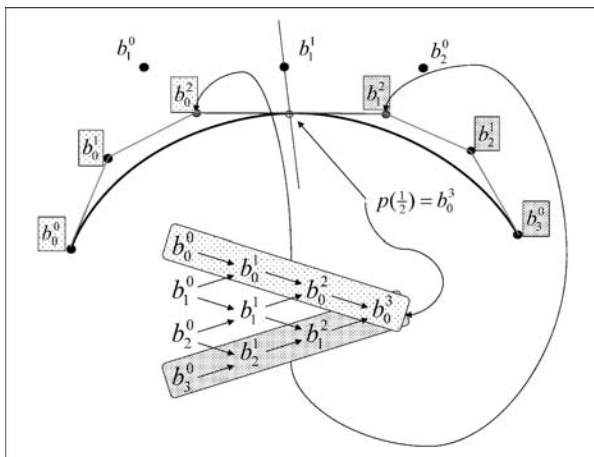


Abb. 1.18:
Zerlegung einer Bézierkurve
in Teilstimente

Ist also $q \in [s, t]$ ein beliebiger Punkt des Intervalls, so kann die Bézierkurve in q geteilt werden, und die neuen Bézierpunkte auf den Teilstegmenten $[s, q]$ und $[q, t]$ ergeben sich aus dem De-Casteljau-Algorithmus wie folgt:

$$p(u) = \begin{cases} \sum_{i=0}^n {}^q B_i^n(u) b_0^i(q) & s \leq u \leq q \\ \sum_{i=0}^n {}^t B_i^n(u) b_1^{n-i}(q) & q \leq u \leq t \end{cases}$$

Man beobachtet nun, dass bei mehrmaliger, fortgesetzter Unterteilung der Bézierkurve die Folge der Bézierpolygone gegen die eigentliche Kurve konvergiert. Bei fortgesetzter Halbierung des Parameterintervalls ist diese Konvergenz exponentiell und liefert ein praktisches Verfahren zur Darstellung einer Bézierkurve durch eine stückweise lineare Approximation.

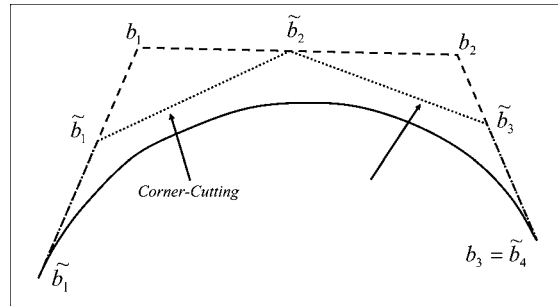
Werden zur genaueren Approximation einer geometrischen Figur durch eine Bézierkurve mehr Bézierpunkte benötigt als vorhanden, so kann der Grad der Kurve erhöht werden. Bei Erhöhung um einen Grad wird ein weiterer Kontrollpunkt in das Kontrollpolygon eingefügt, ohne dass sich die geometrische Form der Kurve ändert.

Graderhöhung einer
Bézierkurve

Ist eine Bézierkurve p vom Grad n mit den Bézierpunkten $b_0, b_1, \dots, b_n \in \mathbb{R}^d$ gegeben, so ist p als Bézierkurve vom Grad $n+1$ darstellbar, so dass gilt:

$$\sum_{i=0}^n b_i {}^t B_i^n(u) = p(u) = \sum_{i=0}^{n+1} \tilde{b}_i {}^t B_i^{n+1}(u), \quad u \in [s, t]$$

Abb. 1.19:
Graderhöhung einer
Bézierkurve



Die neuen Bézierpunkte sind gegeben durch:

$$\begin{aligned}\tilde{b}_0 &= b_0 \\ \tilde{b}_j &= \frac{j}{n+1} b_{j-1} + \left(1 - \frac{j}{n+1}\right) b_j, \quad j = 1, \dots, n \\ \tilde{b}_{n+1} &= b_n\end{aligned}$$

Dieses Vorgehen wird uns an anderer Stelle noch einmal begegnen und zeigen, wie Bézierkurven mit Subdivisionalgorithmen erzeugt werden können. An dieser Stelle sei einfach die Idee in einer Figur dargestellt.

Eigenschaften von Bézierkurven

Bézierkurven vereinigen einige sehr gute Eigenschaften, die zur (interaktiven) Modellierung gebraucht werden. Die Kurven sind schnell zu berechnen, sie sind affin invariant und haben zahlreiche geometrische Eigenschaften: Das Bézierpolygon vermittelt den ungefähren Verlauf der Kurve. Neben der erwähnten Tatsache, dass die Kurve einfach auszurechnen ist, hat man die Ableitungen ebenso unter Kontrolle; die Kurve verläuft am Anfang und Ende tangential zum Bézierpolygon der Kurve und die Ableitung kann aus dem De-Casteljau-Algorithmus bestimmt werden.

Außerdem können Bézierkurven unterteilt und weiter verfeinert werden. Verändert man interaktiv die Lage der Bézierpunkte, so folgt die Kurve der Richtung der Lageänderung. Dies ermöglicht erst ein intuitives und interaktives Modellieren.

Doch sind neben all diesen schönen Eigenschaften auch ein paar Punkte, die wir noch verbessern sollten: Sind $n+1$ Bézierpunkte gegeben, so sind $n+1$ Bernsteinpolynome zur Gewichtung notwendig. Diese besitzen den Grad n und bilden eine Basis des Vektorraums P^n aller Polynomkurven vom Grad n in \mathbb{R}^d . Der Grad einer Bézierkurve ist also direkt mit der Anzahl der Bézierpunkte gekoppelt. Je mehr Bézierpunkte oder „Gestaltungspunkte“ man für eine Kurve betrachtet, desto größer werden der Grad und damit die Welligkeit der Kurve.

Ein weiterer Punkt: Betrachtet man einmal die Bernsteinpolynome, so erkennt man, dass diese zwar positiv sind und sich zu 1 addieren, jedoch sind sie auf dem gesamten Intervall positiv und verschwinden nie. Dies bedeutet dann aber wiederum, dass der Einfluss eines Bézierpunktes, der mit einem Bernsteinpolynom gewichtet wird, sich über das gesamte Intervall erstreckt. Der Bézierpunkt beeinflusst global die Form der Kurve. Dies ist ein Nachteil, will man doch lokal die Form der Kurve gestalten und nicht durch lokale Änderung eines Kurven(kontroll)-punktes eine globale Formänderung der Kurve in Kauf nehmen müssen.

1.1.2.8

Splines, Béziersplines

Um den Grad des Interpolationspolynoms nicht allzu groß werden zu lassen, unterteilen wir das Problem (Interpolationsaufgabe, Approximation von Kurven) in Teilstücke von Kurvensegmenten mit niederem Grad (in der Praxis meist vom Grad 3) und fügen diese Kurvensegmente G^n stetig aneinander. Wir definieren:

Ein Spline ist eine stetige Abbildung q von einer Menge von Intervallen in den Vektorraum \mathbb{R}^d .

Die Intervalle $[t_i, t_{i+1}]$, $i=0, \dots, l-1$ werden durch einen *Stützstellenvektor* $T=(t_0, t_1, \dots, t_l)$ mit $t_0 \leq t_1 \leq \dots \leq t_l$ definiert.

Jedes Intervall $[t_i, t_{i+1}]$ wird dabei auf ein Polynomsegment, das *Spline-Segment*, abgebildet.

So können wir nun mit Bézierkurven einen solchen Spline gestalten. Interessant sind die Stellen t_i , an welchen die Teilkurven aneinanderstoßen:

Gegeben seien Intervalle $[t_i, t_{i+1}]$, $i=0, \dots, l-1$, durch einen *Stützstellenvektor* $T=(t_0, t_1, \dots, t_l)$ mit $t_0 \leq t_1 \leq \dots \leq t_l$ und Bézierpunkte $b_{ij} \in \mathbb{R}^d$ für $i=0, \dots, l$ und $j=0, \dots, n$. Dann heißt die durch die Segmente

$$q_i(u) = \sum_{j=0}^n B_j^n(u) b_{i,j}, \quad u \in [t_i, t_{i+1}]$$

definierte Kurve $q = \sum q_i$ *Bézierspline vom Grad n* .

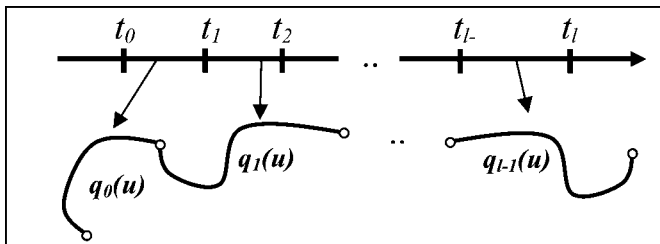


Abb. 1.20:

Aneinanderfügen von Kurvensegmenten

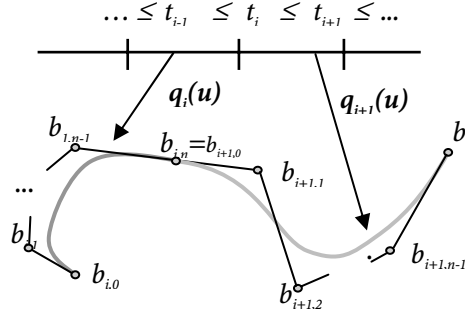
Zu Beginn des Kapitels hatten wir Reparametrisierung und Anschlüsse von Kurven beschrieben und dabei parametrisch und geometrisch richtige Anschlüsse unterschieden. Zur Erinnerung: Wir definierten die Ableitungsvektoren folgendermaßen:

$$T(u) = p'(u), u \in [s, t] \quad \text{Tangentenvektor an die Kurve in } u$$

$$K(u) = p''(u), u \in [s, t] \quad \text{Krümmungsvektor an die Kurve in } u$$

In Anfangs- und Endpunkt einer Kurve sind die rechts- und linksseitigen Ableitungen zu betrachten.

Sind nun zwei Spline-Segmente $q_i: [t_{i-1}, t_i] \rightarrow \mathbb{R}^d$ und $q_{i+1}: [t_i, t_{i+1}] \rightarrow \mathbb{R}^d$ mit den Bézierpunkten $b_{i,0}, \dots, b_{i,n}$ und $b_{i+1,0}, \dots, b_{i+1,n}$ gegeben, so stoßen sie in t_i aneinander.



Stetigkeit an den Anschlussstellen

Zu untersuchen ist, welchen Stetigkeitsgrad, parametrisch stetig (C^n) oder geometrisch stetig (G^n), dieser Anschluss besitzt.

Die G^1 -Stetigkeit an der Verbindung von q_i und q_{i+1} ist gleichbedeutend mit $b_{i,n} = b_{i+1,0}$ und der *Kollinearität* der Punkte $b_{i,n-1}, b_{i,n}, b_{i+1,0}, b_{i+1,1}$. Wie wir wissen, verläuft nämlich die Bézierkurve q_i in $b_{i,n}$ tangential zum Vektor $b_{i,n-1} - b_{i,n}$ und die Bézierkurve q_{i+1} in $b_{i+1,0}$ tangential zu $b_{i+1,0} - b_{i+1,1}$.

Sind also die vier Bézierpunkte kollinear, so zeigen die Ableitungen am Ende und Anfang der Bézier-Spline-Segmente in dieselbe Richtung und dann sind q_i und q_{i+1} G^1 -stetig in t_i .

C^1 -Stetigkeit fordert die Eigenschaft, dass sowohl Richtung als auch Länge der beiden Tangenten (im Ende und Anfang der Spline-Segmente) gleich ist. Also sind in diesem Falle nicht nur die Kollinearität der vier Bézierpunkte zu testen, sondern auch noch die Längenverhältnisse. Es muss gelten:

$$\frac{n}{t_i - t_{i-1}}(b_{i,n} - b_{i,n-1}) = \vec{q}_i'(t_i) = \vec{q}_{i+1}'(t_i) = \frac{n}{t_{i+1} - t_i}(b_{i+1,1} - b_{i+1,0})$$

G^2 - und C^2 -Stetigkeit sind auch noch von Interesse. Für G^2 -Stetigkeit ist Folgendes zu überprüfen: Der Übergang ist stetig, beide Tangentenvektoren haben am Übergang die gleiche Richtung und die Krümmungsvektoren haben auch die gleiche Richtung. Für unsere Bézier-Spline-Segmente bedeutet dies, dass die Punkte $b_{i,n-2}$, $b_{i,n-1}$, $b_{i,n}$ und $b_{i+1,0}$, $b_{i+1,1}$, $b_{i+1,2}$ *komplanar* sein müssen und natürlich $b_{i,n} = b_{i+1,0}$ gelten muss.

1.1.2.9

B-Splines

Béziersplines sind Kurven, die schon viele gute Eigenschaften realisieren. Jedoch ist noch immer eine gewisse Bindung des Grades der gesamten Kurve an die Zahl der Bézierpunkte festzustellen. Die wirkliche Unabhängigkeit von Grad und Anzahl der Kontrollpunkte ist nur durch das Erreichen eines lokalen Trägers der Polynomfunktionen zu schaffen. Dann muss auch nicht durch Einfügen eines neuen Kontrollpunktes der Grad der Kurve erhöht werden. Im Falle von Bézier-splines wäre dies nur durch eine größere Umordnung zu schaffen.

Alle „guten“ Eigenschaften (affine Invarianz, konvexe Hülleneigenschaft, De-Casteljau-Berechnung) sollten jedoch für unsere neue Kurvenart in ähnlicher Weise gelten. Wir müssen neue Basisfunktionen finden, mit denen wir wiederum gegebene Punkte gewichten, die den Kurvenverlauf steuern.

Erinnern wir uns an die Eigenschaften der Bernsteinpolynome, so war die Rekursionsgleichung besonders wichtig. Mit ihrer Hilfe ergab sich der De-Casteljau-Algorithmus, mit seiner Hilfe konnte die konvexe Hülleneigenschaft bewiesen werden und und und ...

Also sollte diese Rekursionsgleichung eine zentrale Rolle für die neuen Basispolynomfunktionen spielen. Warum also nicht die Rekursionsgleichung als Definitionsbasis hernehmen?

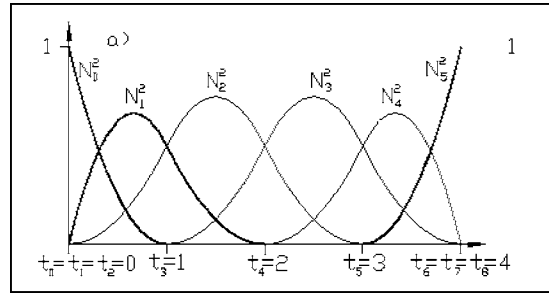
Es seien Zahlen $n \leq m \in \mathbb{N}$ und eine schwach monotone Folge $T = (t_0 = \dots = t_n, t_{n+1}, \dots, t_m, t_{m+1} = \dots = t_{m+n+1})$ von Knoten $t_i \leq t_{i+n+1}$, $0 \leq i \leq m$, gegeben (schwach monoton = monoton, aber es dürfen auch gleiche Elemente vorkommen).

Die *normalisierten B-Spline-Basisfunktionen* N_i^n vom Grad n über T sind durch

$$N_i^0(u) = \begin{cases} 1 & \text{falls } t_i \leq u \leq t_{i+1} \\ 0 & \text{sonst} \end{cases}$$

$$N_i^r(u) = \frac{u - t_i}{t_{i+r} - t_i} N_i^{r-1}(u) + \frac{t_{i+1+r} - u}{t_{i+1+r} - t_{i+1}} N_{i+1}^{r-1}(u), 1 \leq r \leq n$$

Abb.1.21:
B-Spline-Basisfunktionen



rekursiv definiert. Vorkehrung ist für $t_{i+r} = t_i$ und $t_{i+r+1} = t_{i+1}$ zu treffen (schwach monotone Folge). In diesem Fall definieren wir den Summanden gleich 0.

Zu beachten ist, dass die B-Spline-Basisfunktionen abhängig vom gewählten Trägervektor T sind. Sind die Abstände $t_{i+n+1} - t_i$, für alle $0 \leq i \leq m$ stets gleich, so sprechen wir von gleichförmigen (uniform), andernfalls von nicht-gleichförmigen (non-uniform) B-Spline-Basisfunktionen.

Wie für Bernsteinpolynome, so können viele Eigenschaften der B-Spline-Basisfunktionen abgeleitet werden. Sie können sämtlich durch vollständige Induktion bewiesen werden.

- So bestehen die B-Spline-Basisfunktionen $N_i^n(u)$ stückweise aus Polynomen vom Grad n über T .
- Die Funktionen N_i^n besitzen einen lokalen Träger.
- Es gilt $N_i^n(u) \geq 0$ für alle $u \in [t_i, t_{i+n+1}]$ und
- $N_i^n(u) = 0$ für alle $u \notin [t_i, t_{i+n+1}]$.
- Wie die Bernsteinpolynome summieren sich die B-Splines für jeden Parameterwert auf zu eins:

$$\sum_{i=0}^n N_i^n(u) = 1, \quad u \in [t_0, t_{n+m+1}]$$

Wichtig für die Glattheit der Kurve ist die Aussage, dass falls t_j , $j \in \{0, \dots, m+n+1\}$, ein einfacher Knoten ist ($t_{j-1} \neq t_j \neq t_{j+1}$), so ist $N_i^n(t_j)$ mindestens C^{n-1} -stetig.

Wir sehen, dass die Eigenschaften der Bernsteinpolynome erweitert wurden. Man erkennt sogar, dass die B-Splines $N_i^n(u)$ die Bernsteinpolynome ${}_s B_i^n(u)$ über einem Intervall $[s, t]$ als Spezialfall enthalten:

Es sei $s < t \in \mathbb{R}$ und $T = (s, \dots, s, t, \dots, t)$ ein Knotenvektor mit $(n+1)$ -fachen Randknoten s und t . Dann stimmen für $s \leq u \leq t$ die B-Splines N_i^n auf T mit den Bernsteinpolynomen ${}_s B_i^n$ auf $[s, t]$ überein.

Jetzt ist es an der Zeit, Kurven aus diesen neuen Basisfunktionen zu definieren: Wie üblich wählen wir uns zuerst einen Trägervektor T und dazu Zahlen $n \leq m \in \mathbb{N}$ und eine schwach monotone Folge

$$T = (t_0 = \dots = t_n, t_{n+1}, \dots, t_m, t_{m+1} = \dots = t_{m+n+1})$$

mit $t_i \leq t_{i+n+1}$, $0 \leq i \leq m$.

Die B-Spline-Basispolynome sind damit eindeutig definiert und wir können die Punkte $d_0, \dots, d_m \in \mathbb{R}^d$ mit den Basisfunktionen gewichten.

$$q(u) = \sum_{i=0}^m d_i N_i^n(u), \quad u \in [t_0, t_{n+m+1}]$$

Die entstehende Kurve heißt B-Spline-Kurve oder einfach B-Spline über T vom Grad n . Häufig beschränkt man sich jedoch auf kubische B-Splines. Die Punkte d_0, \dots, d_m heißen Kontroll- oder De-Boor-Punkte von q . Sie bilden das Kontroll- oder De-Boor-Polygon.

Die Eigenschaften der B-Spline-Basisfunktionen zeigen, dass die Kurve affin invariant (Partition der 1) ist, innerhalb der konvexen Hülle der De-Boor-Punkte verläuft (Konvex-Kombination) und durch Anfangs- und Endpunkt geht. Wie sieht es nun mit der gesuchten lokalen Einflusseigenschaft eines De-Boor-Punktes aus? Aus Definition der Basisfunktionen $N_i^n(u)$ ist klar, dass $N_i^n(u) \geq 0$ für alle $u \in [t_i, t_{i+n+1}]$ und $N_i^n(u) = 0$ für alle $u \notin [t_i, t_{i+n+1}]$ ist. Also ist das Produkt des De-Boor-Punktes d_i mit $N_i^n(u)$ nur auf dem Parameterstück $[t_i, t_{i+n+1}]$ ungleich null und verschwindet auf dem ganzen Rest des Intervalls $[t_0, t_{n+m+1}]$. Die Form der Kurve wird über dem Intervall $[t_i, t_{i+n+1}]$ also nur von den De-Boor-Punkten d_{i-n}, \dots, d_i beeinflusst und die Basisfunktionen $N_i^n(u)$ schalten bei Segmentübergängen t_i neue De-Boor-Punkte an oder aus.

Außerdem folgt die Kurve der Verschiebungsrichtung des De-Boor-Punktes wie bei Bézierkurven. Aber Bézierkurven sind auch ein Spezi-

Kontrollpunkte (De-Boor-Punkte)
Kontrollpolygon

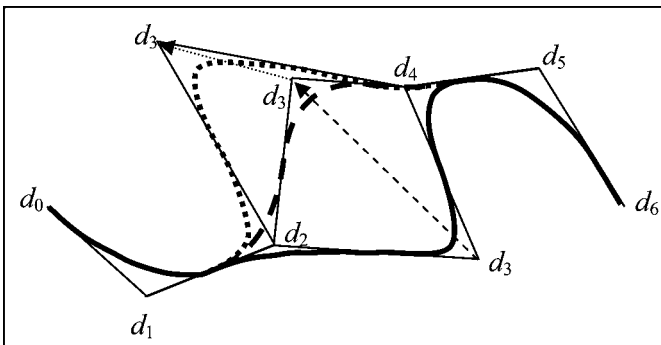


Abb. 1.22:
Manipulation eines B-Splines durch
Verschiebung der Kontrollpunkte

alfall, da der Trägervektor $T=(s,s,\dots,s,t,t,\dots,t)$ speziell gewählt wurde (n -mal s und t)

1.1.3

De-Boor-Algorithmus für B-Splines

Auch die Berechnung des Kurvenverlaufs kann wie bei Bézierkurven durch ein rekursives Schema durchgeführt werden: Gegeben sei ein *B-Spline* vom Grad n über

$$T = (t_0 = \dots = t_n, t_{n+1}, \dots, t_m, t_{m+1} = \dots = t_{m+n+1})$$

mit $t_i \leq t_{i+n+1}$, $0 \leq i \leq m$:

$$q(u) = \sum_{i=0}^m d_i N_i^n(u), \quad u \in [t_l, t_{l+1}]$$

dann liefert das rekursiv definierte Schema (*De-Boor-Algorithmus*) den Funktionswert $q(u) = d_{l-n}^n(u)$.

$$d_i^0(u) = d_i, \quad i = l-n, \dots, l$$

$$d_i^r(u) = \left(1 - \frac{u - t_{i+r}}{t_{i+n+1} - t_{i+r}}\right) \cdot d_i^{r-1}(u) + \frac{u - t_{i+r}}{t_{i+n+1} - t_{i+r}} \cdot d_{i+1}^{r-1}(u)$$

$$i = l-n, \dots, l-r \quad \text{und} \quad 0 \leq r \leq n$$

Noch einmal sieht man hier den Zusammenhang zwischen allgemeineren B-Splines und Bézierkurven: Ist der Trägervektor $T = (s, \dots, s, t, \dots, t)$, dann geht der De-Boor-Algorithmus in den De-Casteljau-Algorithmus über. Insbesondere stimmt dann die B-Spline-Darstellung einer B-Spline-Kurve q über T mit der Bézierdarstellung von q über $[s, t]$ überein.

Wir wollen einmal diesen Algorithmus an einem Beispiel testen. Bekannt ist, dass der B-Spline durch Anfangs- und Endpunkt verläuft. Kommt dies auch beim De-Boor-Algorithmus heraus? Dazu betrachten wir beispielhaft: $n=2$, $T=(0,0,0,1,2,3,3,3)$ und $d_0=(0|0)$, $d_1=(1|0)$, $d_2=(1|1)$, $d_3=(0|1)$. Wir wollen den Anfangspunkt berechnen, somit ist $u=0$ und damit $u \in [0,1]=[t_2, t_3]$, also $l=2$. Der De-Boor-Algorithmus liefert:

$$\begin{array}{ccccc} d_0^0 = (0|0) & & d_1^0 = (1|0) & & d_2^0 = (1|1) \\ & \swarrow \quad \searrow & & \swarrow \quad \searrow & \\ d_1^1 = (0|0) & & d_1^1 = (1|0) & & \\ & \swarrow \quad \searrow & & \swarrow \quad \searrow & \\ d_2^2 = (0|0) & & & & \end{array}$$

Auch diesmal kann der De-Boor-Algorithmus als geometrische Konstruktion interpretiert werden, wie wir an einem weiteren Beispiel, das in Abb. 1.24 dargestellt ist, sehen können: Es sei wieder $n=2$, $T=(0,0,0,1,2,2,2)$ und $d_0=(0|0)$, $d_1=(1|0)$, $d_2=(1|1)$, $d_3=(0|1)$.

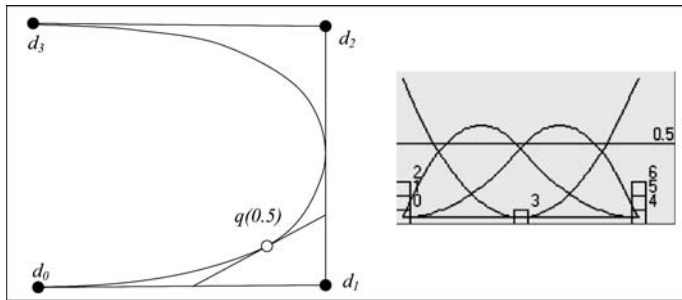


Abb. 1.23:
Geometrische Konstruktion
des De-Boor-Algorithmus

Viele Eigenschaften lassen sich für B-Splines beweisen. Hier sind die wichtigsten zusammengefasst:

Für $t_i \leq u \leq t_{i+1}$ liegt $q(u)$ in der konvexen Hülle der $n+1$ Kontrollpunkte d_{i-n}, \dots, d_i (konvexe Hüllen-Eigenschaft, convex hull property):

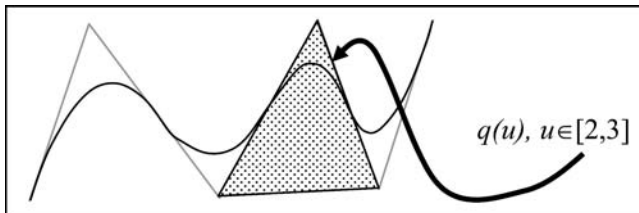


Abb. 1.24:
Konvexe Hüllen-Eigenschaft

Fallen n Kontrollpunkte $d_{i-n+1}=d_{i-n}=\dots=d_i=d$ zusammen, so gilt $q(t_{i+1})=d$, d. h., die Kurve verläuft durch diesen Kontrollpunkt.

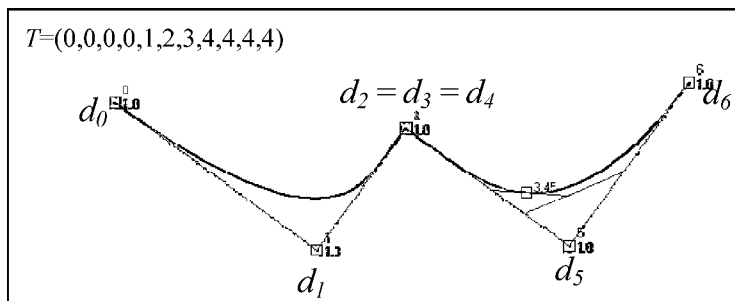
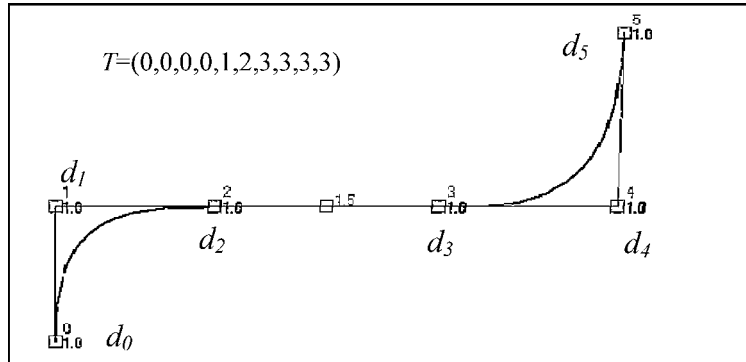


Abb. 1.25:
 n -facher Kontrollpunkt

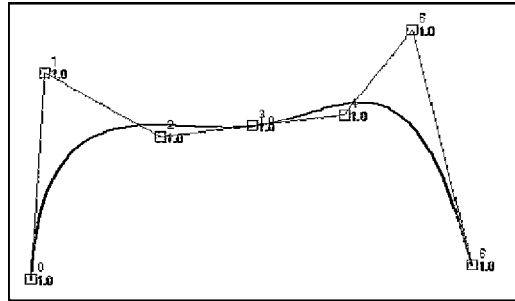
Liegen $n+1$ Kontrollpunkte $d_{l-n} = \dots = d_l$ auf einer Geraden L , so gilt $q(u) \in L$ für $t_l \leq u \leq t_{l+1}$.

Abb. 1.26:
 $n+1$ kollineare Kontrollpunkte



Fallen n Knoten $t_{l+1} = \dots = t_{l+n} =: t$ zusammen, so ist $q(t) = d_l$ ein Kontrollpunkt, und die Kurve schmiegt sich dort tangential an das Kontrollpolygon. Insbesondere verläuft die Kurve bei $(n+1)$ -fachen Anfangs- und Endknoten durch die Endpunkte des Polygons und ist dort tangential zum Kontrollpolygon.

Abb. 1.27:
 n -facher Knoten



Einfügen weiterer Kontrollpunkte

Wie bei Bézierkurven will man durch interaktives Einfügen weiterer Kontrollpunkte die Geometrie der Kurve „verfeinern“. Der B-Spline lässt sich flexibler gestalten. Analog wollen wir einen neuen De-Boor-Punkt d_r^* einfügen, allerdings ohne gleichzeitig den Grad des B-Splines zu erhöhen!

Gegeben sei ein B-Spline vom Grad n über

$$T = (t_0 = \dots = t_n, t_{n+1}, \dots, t_m, t_{m+1} = \dots = t_{m+n+1}).$$

$$q(u) = \sum_{i=0}^m d_i N_i^n(u)$$

Es sei weiter $d_r^* \in \mathbb{R}^d$ und etwa $t_r \leq t^* \leq t_{r+1}$. Dann besitzt $q(u)$ die Darstellung vom Grade n über $T = (t_0 = \dots = t_n, t_{n+1}, \dots, t_r, t^*, t_{r+1}, \dots, t_m, t_{m+1} = \dots = t_{m+n+1})$

$$q(u) = \sum_{i=0}^{m+1} d_i^* N_i^n(u)$$

Die neuen De-Boor-Punkte d_i^* sind:

$$d_i^* = (1 - a_i) d_{i-1} + a_i d_i \quad \text{mit}$$

$$a_i = \begin{cases} 1 & \text{für } i \leq r - n \\ \frac{t - t_i}{t_{i+n} - t_i} & \text{für } r - n + 1 \leq i \leq r \\ 0 & \text{für } r + 1 \leq i \end{cases}$$

bleiben

nur n De-Boor-Punkte

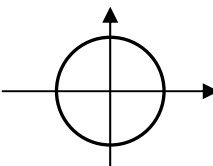
geschiftet

1.1.3.1

Rationale Kurven

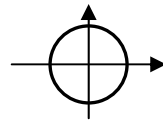
Mit den B-Splines besitzen wir ein mächtiges Werkzeug zur Gestaltung von Kurven. Sie bieten die Möglichkeit, interaktiv zu modellieren und sind schnell auszurechnen. Außerdem sind diese Kurven stabil unter affinen Transformationen. Jedoch tritt wieder ein Problem auf: Es gibt leider Kurven, die wir nicht als B-Spline beschreiben können.

Eine der elementarsten Kurven ist ein Kreis:

$$K(u) = \begin{pmatrix} r \cos(2\pi u) \\ r \sin(2\pi u) \end{pmatrix}, \quad u \in [0, 1]$$


Jedoch sieht man nun sofort, dass die Koordinatenfunktionen aus Sinus- und Cosinusfunktion bestehen, die wiederum nicht als Polynom endlichen Grades dargestellt werden können. Sie sind Exponentialreihen. Also gibt es keine Chance, Kreise als Monom-, Lagrange-, Hermite-, Bézier- oder B-Spline-Kurve darzustellen. Es wäre immer nur eine gute Näherung, aber nicht exakt ein Kreis.

Aber es gibt auch hier einen Ausweg! Kreise erhält man auch als Projektionen eines Kegelmantels (Kegelschnitt). Die parametrisierte Kurve eines Kreises hat daher auch die Form



$$K(u) = r \cdot \left(\frac{1-u^2}{1+u^2} \mid \frac{2u}{1+u^2} \right)$$

Nun aber sind die Koordinaten durch (ganz)rationale Funktionen gegeben. Aus dieser Not wird eine Tugend und wir definieren:

Rationale Bézierkurve

Gegeben seien Bézierpunkte $b_i \in \mathbb{R}^d$, $i=0, \dots, n$, und Gewichte $\omega_i \geq 0$, $i=0, \dots, n$, mit $\omega_0 = \omega_n = 1$. Dann heißt

$$r(u) = \frac{\sum_{i=0}^n \omega_i b_i \binom{n}{i} u^i (1-u)^{n-i}}{\sum_{i=0}^n \omega_i \binom{n}{i} u^i (1-u)^{n-i}}, \quad u \in [0, 1]$$

rationale Bézierkurve vom Grad n .

Nun kann man nicht nur rationale Bézierkurven betrachten, sondern auch dasselbe für die allgemeineren B-Splines durchführen.

Rationale B-Spline-Kurve

Sei $T = (t_0 = \dots = t_n, t_{n+1}, \dots, t_m, t_{m+1} = \dots = t_{m+n+1})$ ein Trägervektor und sind $m+1$ Kontrollpunkte $d_i \in \mathbb{R}^d$ und Gewichte $\omega_i \geq 0$, $i=0, \dots, m$, gegeben, so heißt

$$q(u) = \frac{\sum_{i=0}^m d_i \omega_i N_i^n(u)}{\sum_{j=0}^m \omega_j N_j^n(u)} = \frac{\sum_{i=0}^m d_i \omega_i N_i^n(u)}{\sum_{j=0}^m \omega_j N_j^n(u)}$$

NURBS

rationale B-Spline-Kurve. Sind die Intervallgrößen in T unterschiedlich, heißt q nicht-gleichförmiger rationaler (non-uniform-rational) B-Spline (NURBS).

In beiden Kurvenarten sind jeweils Bézier- oder B-Spline-Kurven in Zähler und Nenner zu finden. Zur Berechnung der Kurve könnte man nun naiv Zähler und Nenner mit dem De-Casteljau- oder De-Boor-Algorithmus auswerten und den Quotient als Kurvenwert berechnen. Jedoch geht es besser, wenn man sich daran erinnert, dass wir Projektionen vor uns haben (Kreis als Kegelschnitt)!

Gegeben seien Bézierpunkte $b_i = (x_i, y_i, z_i) \in \mathbb{R}^3$, $i=0, \dots, n$, und Gewichte $\omega_i \geq 0$, $i=0, \dots, n$, mit $\omega_0 = \omega_n = 1$.

Dann ist $b_i^h = (\omega_i x_i, \omega_i y_i, \omega_i z_i, \omega_i) \in \mathbb{R}^4$, und

$$F(u) = \sum_{i=0}^n b_i^h \binom{n}{i} u^i (1-u)^{n-i}, \quad u \in [0, 1]$$

definiert eine Bézierkurve im \mathbb{R}^4 .

Diese Bézierkurve wird nun auf die Hyperebene $\mathbb{R}^3 = (*, *, *, 1)$ projiziert, d. h. durch die letzte Koordinate dividiert. Man erhält die *rationale Bézierkurve* $r(u)$. Gleiches gilt für die B-Spline-Kurven und (N)URBS.

Für die Berechnung wendet man also den De-Casteljau- oder De-Boor-Algorithmus auf b_i^h an. Wir erhalten den Funktionswert der vierdimensionalen Kurve $F(u) = (x|y|z|w)$. Den gesuchten Funktionswert der rationalen Kurven erhält man nun, indem man die Koordinaten x, y und z durch die vierte berechnete Koordinate w dividiert und somit auf die dreidimensionale Hyperebene (mit $w=1$) projiziert.

Sei zum Beispiel $T = (0,0,0, 1,1,1)$ gegeben mit $b_0 = (1|0)$, $b_1 = (1|1)$ und $b_2 = (0|1)$ mit Gewichten $\omega_0=1=\omega_1$ und $\omega_2=2$. Dann ist die rationale Bézierkurve des Viertelkreises gegeben durch:

$$r(u) = \frac{\sum_{i=0}^2 \omega_i b_i B_i^2(u)}{\sum_{i=0}^2 \omega_i B_i^2(u)}, \quad u \in [0,1]$$

$$= \frac{\omega_0 \binom{1}{0} (1-u)^2 + \omega_1 \binom{1}{1} 2u(1-u) + \omega_2 \binom{0}{1} u^2}{\omega_0 (1-u)^2 + \omega_1 2u(1-u) + \omega_2 u^2}$$

Wie nun leicht nachzurechnen ist, gilt

$$r(u) = \begin{pmatrix} x(u) \\ y(u) \end{pmatrix} = \begin{pmatrix} \frac{1-u^2}{1+u^2} & \frac{2u}{1+u^2} \end{pmatrix}^T, \quad \text{mit } x^2(u) + y^2(u) = 1$$

Also beschreibt $r(u)$ einen Kreisbogen, der in $(1|0)$ beginnt und in $(0|1)$ endet. Zur Berechnung (und Darstellung) der Kreisbogenpunkte wird jedoch nicht die obige Formel verwendet, sondern es wird der De-Casteljau-Algorithmus auf die Punkte $b_0^h = (1|0|1)$, $b_1^h = (1|1|1)$ und $b_2^h = (0|1|2)$ angewendet. Es entsteht eine dreidimensionale Bézierkurve, welche dann auf die Ebene $x_3=1$ projiziert wird.

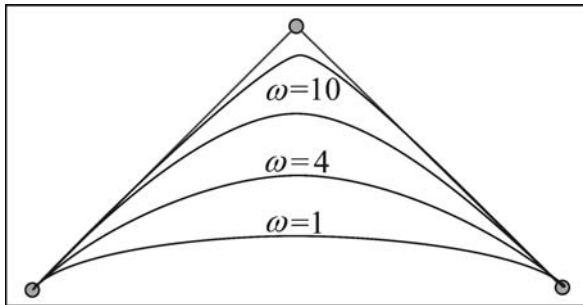


Abb. 1.28:
Einfluss der Gewichte auf den
Kurvenverlauf

Wird das Gewicht ω_k vergrößert, so bewegt sich die Kurve zum k -ten Kontrollpunkt.

1.1.4 Flächen

Im vorhergehenden Kapitel hatten wir Kurven immer nach dem gleichen Prinzip gebildet: Wir hatten gegebene Kontrollpunkte mit verschiedenen (polynomiellen) Basisfunktionen, die über einem Intervall $[s,t]$ definiert waren, gewichtet. Was, wenn nun die Kontrollpunkte selbst wieder durch Kurven definiert werden? Man erhält die sogenannten Tensorproduktflächen. Zunächst betrachten wir zur Begriffsklärung allgemeine Flächenfunktionen.

Sei $\Omega \subseteq \mathbb{R}^2$ ein Gebiet in der Ebene – der Parameterbereich. Meist verwendet man Rechtecks- oder Dreiecksgebiete als Parameterbereich. Eine Fläche ist eine Abbildung $q: \Omega \rightarrow \mathbb{R}^d$.

Im Falle $d=3$ und $\Omega=[0,1] \times [0,1]$ ist eine Fläche also durch drei Koordinatenfunktionen x , y und z in jeweils zwei Parametern u und v gegeben:

$$q: (u, v) \rightarrow \begin{pmatrix} x(u, v) \\ y(u, v) \\ z(u, v) \end{pmatrix}, \quad (u, v) \in [0, 1]^2$$

Stetigkeit von Flächen

Wieder sind wie im Kurvenfall d Koordinatenfunktionen zu wählen, die Stetigkeitseigenschaften besitzen. So sagen wir, die Fläche ist C^n -stetig oder n -mal differenzierbar, falls die Abbildung q mindestens n -mal stetig differenzierbar ist, d.h., q ist n -mal stetig partiell differenzierbar.

Reguläre Flächen

Eine Fläche heißt regulär in (u_0, v_0) , falls q einmal stetig differenzierbar ist und die Vektoren

$$q_u(u_0, v_0) = \frac{\partial q}{\partial u}(u_0, v_0) = \begin{pmatrix} \frac{\partial x}{\partial u}(u_0, v_0) \\ \frac{\partial y}{\partial u}(u_0, v_0) \\ \frac{\partial z}{\partial u}(u_0, v_0) \end{pmatrix} \quad q_v(u_0, v_0) = \frac{\partial q}{\partial v}(u_0, v_0)$$

Tangentialvektoren

linear unabhängig sind (und damit nicht verschwinden). q_u und q_v heißen Tangentialvektoren.

Ist q regulär, so spannen q_u und q_v eine Tangentialebene auf. Die Normale dieser Tangentialebene ist definiert durch das Kreuzprodukt (oder Vektorprodukt) der Tangentialvektoren

Normale

$$n(u_0, v_0) = \frac{q_u(u_0, v_0) \times q_v(u_0, v_0)}{\|q_u(u_0, v_0) \times q_v(u_0, v_0)\|}$$

und heißt Normalenvektor in (u_0, v_0) .

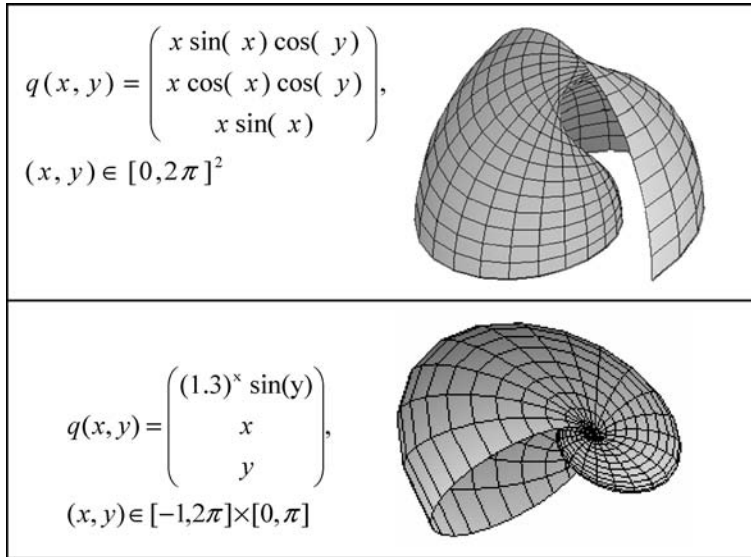


Abb. 1.29:
Beispiele für parametrisierte Flächen

1.1.4.1

Tensorproduktflächen

Tensorproduktflächen gewinnt man leicht aus den uns bekannten Kurvendarstellungen. Betrachten wir zum Beispiel eine Kurve q , gegeben durch

$$q(u) = \sum_{i=0}^n C_i F_i(u)$$

mit Kontrollpunkten C_i und Basisfunktionen F_i .

Nun bewegen wir diese Kurve mittels eines weiteren unabhängigen Parameters v durch den Raum, wobei wir auch Deformationen der Kurve zulassen werden. Deformation der Kurve bedeutet dabei Veränderung der Kontrollpunkte C_p , d. h., wir beschreiben diese Verände-

rung wieder durch eine Kurve mit dem neuen Parameter v und möglicherweise anderer Basisfunktionen G_j :

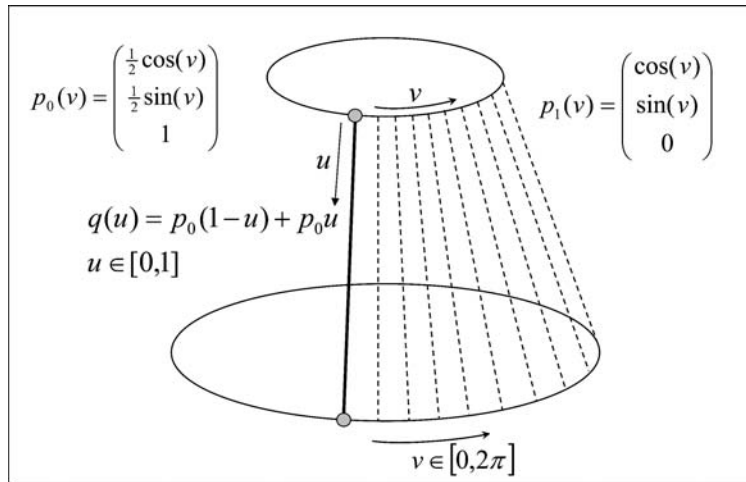
$$C_i(v) = \sum_{j=0}^m A_{ij} G_j(v).$$

Insgesamt entsteht so die Tensorproduktfläche:

$$q(u, v) = \sum_{i=0}^n \sum_{j=0}^m A_{ij} G_j(v) F_i(u), \quad (u, v) \in [a, b] \times [c, d]$$

Ein einfaches Beispiel einer Tensorproduktfläche ist ein Geradenstück, bei dem die beiden Endpunkte mittels einer Kurve verändert werden. So entsteht zum Beispiel eine Kegelstumpf-Mantelfläche:

Abb. 1.30:
Beispiel einer Tensorproduktfläche



Eine Tensorproduktfläche mit rechteckigem Parametergebiet kann auch in Matrixform angegeben werden:

Sei $(u, v) \in [a, b] \times [c, d]$

$$\begin{aligned} q(u, v) &= \sum_{i=0}^n \sum_{j=0}^m A_{ij} G_j(v) F_i(u) = \\ &= (F_0(u) \quad \cdots \quad F_n(u)) \begin{pmatrix} A_{00} & \cdots & A_{0m} \\ \vdots & & \vdots \\ A_{n0} & \cdots & A_{nm} \end{pmatrix} \begin{pmatrix} G_0(v) \\ \vdots \\ G_m(v) \end{pmatrix} \end{aligned}$$

Man erkennt deutlich die regelmäßige Gitterstruktur der gegebenen Kontrollpunkte A_{ij} , die für eine so gebildete Tensorproduktfläche

notwendig gegeben sein müssen. Regelmäßig soll heißen, in der gleichen Anzahl der Kontrollpunkte in u - und v -Richtung und nicht in der Raumlage der Punkte. Sind die Kontrollpunkte unregelmäßig verteilt (scattered), müssen andere Techniken eingesetzt werden, wie sie bei der Scattered-Data-Interpolation entwickelt wurden [LF99].

1.1.4.2

Interpolation

Wie im Kurvenfall stellt sich in der Praxis oft die Aufgabe, eine Fläche (Kurve) durch eine gegebene Anzahl von Kontrollpunkten zu legen. Es seien also $n \times m$ Kontrollpunkte P_{ij} , $i=0, \dots, n$, und $j=0, \dots, m$, gegeben zu Parameterwerten u_i und v_j . Gesucht sei nun eine (polynomielle) Fläche $X(u, v)$, die diese Punkte interpoliert. Es ergibt sich aus der Voraussetzung $X(u_i, v_j) = P_{ij}$ für $i=0, \dots, m$, $j=0, \dots, n$ ein Gleichungssystem $P = FAG$ aus Matrizen:

$$P = \begin{pmatrix} P_{00} & \cdots & P_{0m} \\ \vdots & & \vdots \\ P_{n0} & \cdots & P_{nm} \end{pmatrix} \quad A = \begin{pmatrix} A_{00} & \cdots & A_{0m} \\ \vdots & & \vdots \\ A_{n0} & \cdots & A_{nm} \end{pmatrix}$$

$$F = \begin{pmatrix} F_0(u_0) & \cdots & F_n(u_0) \\ \vdots & & \vdots \\ F_0(u_n) & \cdots & F_n(u_n) \end{pmatrix} \quad G = \begin{pmatrix} G_0(v_0) & \cdots & G_m(v_0) \\ \vdots & & \vdots \\ G_m(v_0) & \cdots & G_m(v_m) \end{pmatrix}.$$

Dieses Gleichungssystem ist nach der unbekannten Matrix A aufzulösen. Sind wiederum wie im Kurvenfall die Parameterwerte u_i und v_j paarweise verschieden, und wählt man als Basisfunktionen F_i und G_j Monome, dann sind F und G invertierbare van der Monde'sche Matrizen. Das Gleichungssystem kann dann zu $A = F^{-1}PG^{-1}$ aufgelöst werden und liefert die Existenz und Eindeutigkeit einer Interpolationsfläche.

Werden für F und G Lagrangepolynome verwendet, d. h.

$$F_i^n(u_k) = \delta_{ik} = \begin{cases} 1 & \text{falls } i = k \\ 0 & \text{sonst} \end{cases}$$

$$G_j^m(v_l) = \delta_{jl},$$

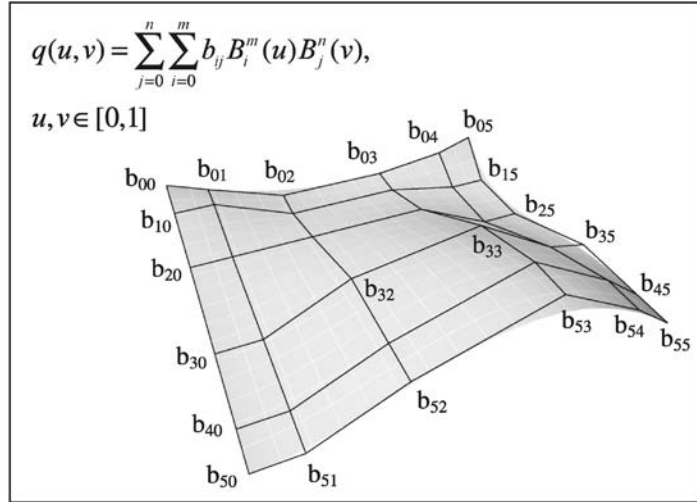
so sind F und G Einheitsmatrizen und wir erhalten, wie im Fall der Kurven, eine besonders einfache Lösung der Interpolationsaufgabe mit $A=P$.

1.1.4.3

Bézier-Tensorproduktflächen

Verwendet man als Basisfunktionen F und G die bekannten Bernsteinpolynome über $[0,1]$ oder $[s,t]$ vom Grad n bzw. m , so erhalten wir mit gegebenen Bézierpunkten $b_{ij} \in \mathbb{R}^d$ eine Bézier-Tensorproduktfläche oder ein Bézierpflaster:

Abb. 1.31:
Bézier-Tensorproduktfläche



Kontrollnetz

Die Bézierpunkte bilden das Kontrollnetz des Bézierpflasters.

Die Parameterlinien liefern Bézierkurven, wie man leicht durch geeignetes Klammern erkennt:

$$q(u_0, v) = \sum_{j=0}^n \left(\sum_{i=0}^m b_{ij} B_i^m(u_0) \right) B_j^n(v) = \sum_{j=0}^n b_j(u_0) B_j^n(v)$$

Die Bézier-Tensorproduktfläche liegt für die Parameterwerte $(u, v) \in [0, 1]$ ebenfalls in der konvexen Hülle des Kontrollnetzes, wie man aus den Eigenschaften

$$\underbrace{\sum_{i=0}^m \left(\underbrace{\sum_{j=0}^n B_j^n(v)}_{=1} \right) B_i^m(u)}_{=1} = 1 \quad \text{und}$$

$$\sum_{i=0}^m \sum_{j=0}^n B_i^m(u) B_j^n(v) \geq 0$$

erkennen kann. Die erste der obigen Eigenschaften sichert auch die affine Invarianz dieser Flächen.

Eine weitere Eigenschaft ist direkt aus den Bernsteinpolynomen abzuleiten, ähnlich wie im Kurvenfall: Die Bézierpunkte der Randkurven sind die Randpunkte des Bézier-Tensorproduktflächen-Kontrollnetzes.

Die Ableitungen in den Randpunkten lassen sich wie bei den Kurven aus den Bézierpunkten der Randkurven berechnen:

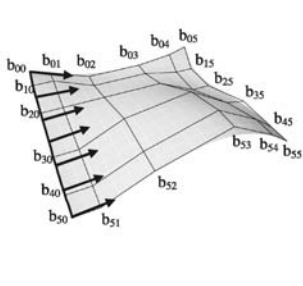
$$\left| \begin{array}{l} \frac{\partial}{\partial u} q(u, v) \big|_{u=0} = m \sum_{j=0}^n B_j'(v) (b_{1j} - b_{00}) \\ \frac{\partial}{\partial u} q(u, v) \big|_{u=1} = m \sum_{j=0}^n B_j'(v) (b_{mj} - b_{-1}) \\ \frac{\partial}{\partial v} q(u, v) \big|_{v=0} = n \sum_{i=0}^m B_i'(u) (b_{i1} - b_i) \\ \frac{\partial}{\partial v} q(u, v) \big|_{v=1} = n \sum_{i=0}^m B_i'(u) (b_{im} - b_{i-1}) \end{array} \right|$$


Abb. 1.32:
Ableitungen in den Randpunkten von
Bézier-Tensorproduktfläche

Der bereits von den Bézierkurven bekannte De-Casteljau-Algorithmus kann auf Tensorproduktflächen erweitert werden:

$$q(u_0, v_0) = \sum_{j=0}^n \underbrace{\left(\sum_{i=0}^m b_{ij} B_i^m(u_0) \right)}_{\substack{\text{für festes } i, u_0 \text{ aus den} \\ b_{ij}^u(u_0) = b_{ij}^n(u_0) \\ \text{mit de Casteljau} \\ b_j(u_0) = b_{ij}^n(u_0) \text{ berechnen}}} B_j^n(v) = \sum_{j=0}^n \underbrace{b_j(u_0) B_j^n(v_0)}_{\substack{\text{mit de Casteljau Kurve an} \\ \text{der Stelle } v=v_0 \text{ auswerten}}}$$

Die vorletzten Elemente des De-Casteljau-Schemas liefern wieder die Richtungen der partiellen Ableitungen q_u bzw. q_v der Bézierfläche.

1.1.5

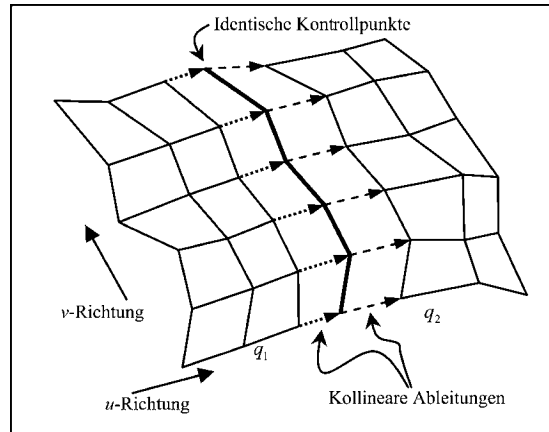
Anschlüsse von Bézier-Tensorproduktflächen

Wie wir wissen, definieren die Randpunkte des Kontrollnetzes den Verlauf der Randkurve der Tensorproduktfläche. Durch sie können wir auch zwei Tensorproduktflächen q_1 und q_2 , die wir aneinanderfügen wollen, kontrollieren.

$$q_1(u, v) = \sum_{i=0}^m \sum_{j=0}^n b_{ij}^{r_1} B_i^m(u) B_j^n(v)$$

$$q_2(u, v) = \sum_{i=0}^m \sum_{j=0}^n c_{ij}^{r_2} B_i^m(u) B_j^n(v)$$

Abb. 1.33:
Aneinanderfügen zweier
Bézier-Tensorproduktflächen



Sollen sie stetig aneinandergefügt werden, so müssen die jeweiligen Randpunkte der Kontrollnetze an der Anschlussstelle identisch sein. Dann besitzen die beiden Bézierflächen dieselbe Randkurve. Ein wenig schwieriger wird es, die beiden Flächenstücke an der Übergangsstelle C^1 oder G^1 stetig zu gestalten.

Wir wissen jedoch, wie sich die Randableitungen, d.h. die partiellen Ableitungen an den Rändern, berechnen.

Um zwei Pflaster entlang der Randkurve in v -Richtung (C^1 -stetig) aneinanderschließen zu können, müssen erstens die beiden Pflaster dieselbe Randkurve in v -Richtung besitzen, d.h., für die Bézierpunkte gilt

$$b_{mj} = c_{0j}, \quad j=0, \dots, n,$$

und zweitens müssen entlang der Randkurve, d.h. zu jedem festen v , die Ableitungen in u -Richtung in Betrag und Richtung übereinstimmen. Für C^1 -Stetigkeit muss also gelten (Anschluss in v -Richtung), dass die Ableitungen in u -Richtung entlang der gemeinsamen v -Kurve gleich sind:

$$b_{mj} - b_{m-1,j} = c_{1j} - c_{0j}$$

Fordert man beim Aneinanderfügen nur paarweise kollineare Tangentenvektoren – also gleiche Tangentenrichtung –, was der G^1 -Stetigkeit entlang der gesamten Nahtlinie entspricht, so müssen die Polygonsegmente des Kontrollnetzes zwar kollinear sein, aber in keinem bestimmten Längenverhältnis stehen.

Allgemeiner kann festgestellt werden, will man einen C^k -stetigen Übergang erzwingen, müssen die Kontrollpunkte des Netzes so gewählt werden, dass alle Kurven (in u - und v -Richtung) C^k -stetig sind.

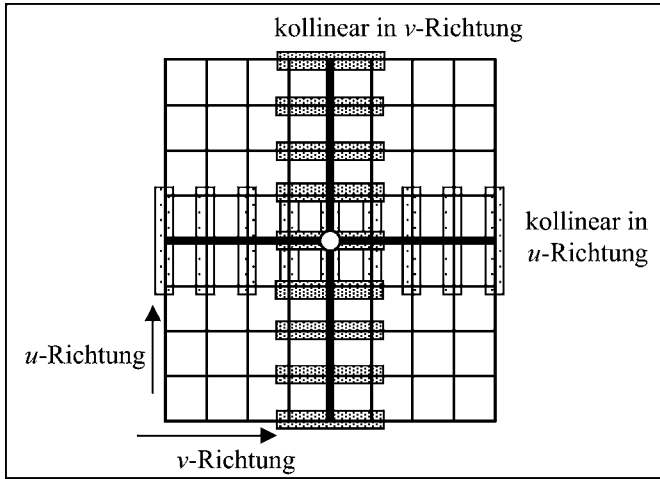


Abb. 1.34:
Anschlussbedingungen beim
Aneinanderfügen mehrerer
Bézierpflaster

Beim Aneinanderfügen von drei oder mehr Bézierpflaster sind so eine ganze Reihe von Bedingungen an die Kontrollpunkte zu erfüllen. Vor allem am gemeinsamen Punkt sind Vorkehrungen zu treffen.

Wie man aus Abb. 1.34 leicht erkennen kann, sind schon im G^1 -Fall die vier nächsten Kontrollpunkte zum gemeinsamen Punkt in einer Ebene zu wählen.

Will man vier Bézierpflaster so aneinanderfügen, dass die Übergänge C^k -(G^k -)stetig sind, so sind am besten die Randkurven (schwarz) glatt vorzugeben und die inneren (freien) Kontrollpunkte entsprechend der geforderten Stetigkeit zu setzen.

1.1.5.1

Rationale Bézier-Tensorproduktflächen

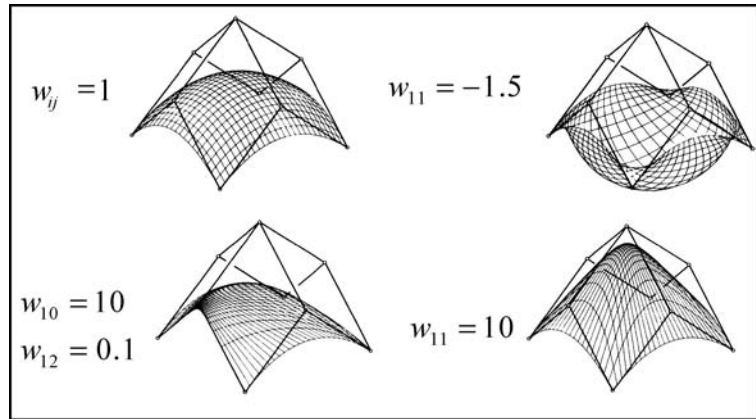
Sind ein Kontrollnetz $b_{ij} \in \mathbb{R}^d$ und Gewichte $w_{ij} \in \mathbb{R}$ gegeben ($i=0, \dots, m$ und $j=0, \dots, n$), kann eine rationale Bézierfläche gebildet werden:

$$R(u, v) = \frac{\sum_{i=0}^m \sum_{j=0}^n \tilde{b}_{ij} B_i^m(u) B_j^n(v)}{\sum_{i=0}^m \sum_{j=0}^n w_{ij} B_i^m(u) B_j^n(v)},$$

wobei

$$\tilde{b}_{ij} = \begin{cases} w_{ij} b_{ij}, & \text{falls } w_{ij} \neq 0 \\ b_{ij} & \text{sonst.} \end{cases}$$

Abb. 1.35:
Beispiele rationaler
Bézier-Tensorproduktflächen



Um diese rationale Bézierfläche durch die Eckpunkte zu zwingen, fordert man $w_{m0}=w_{0n}=1$.

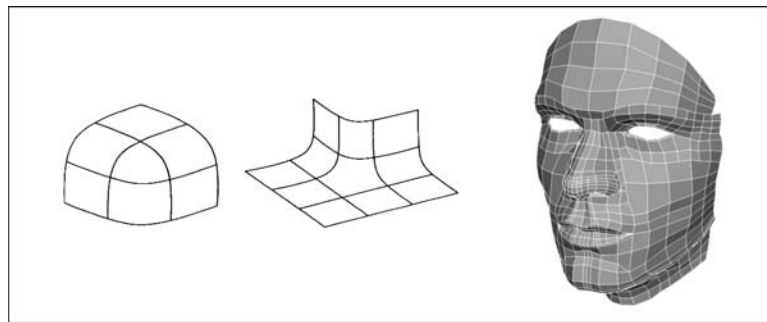
Die rationale Bézierfläche ist wieder als Projektion einer Fläche in \mathbb{R}^4 in den dreidimensionalen Raum zu betrachten. Daher übertragen sich die Eigenschaften wie konvexe Hülle, affine Invarianz, Stetigkeit u. A. auch auf diesen Flächentyp.

1.1.5.2

Tensorproduktflächen über Dreiecken

Wie schon erwähnt werden nicht nur rechteckige Parametergebiete zur Flächenbildung verwendet. In der Tat gibt es Modelle, die durch Rechteckspflaster nicht zu realisieren sind:

Abb. 1.36:
Modelle, die nicht nur aus
Viereckspflastern bestehen



Vor allem zur Modellierung von Augen-, Nasen- und Mundbereichen sind Mischungen von Dreiecks- und Vierecksflächen notwendig.

Um diese Situationen lösen zu können, sind zwei Strategien möglich: Zum einen können „entartete“ Vierecksnetze verwendet werden, bei denen zwei Kontrollpunkte zusammenfallen.

Zum anderen können, um wieder auf Viereckspflaster zu kommen, zusätzliche Punkte eingefügt werden:

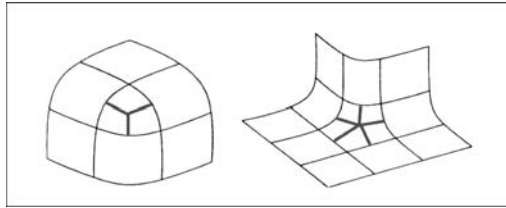


Abb. 1.37:
Einfügen zusätzlicher Punkte,
um Viereckspflaster zu erreichen

Die dritte Möglichkeit besteht in der Verallgemeinerung von Tensorproduktflächen auf Dreiecksgebieten.

Sind nun drei Punkte R , S und T der Ebene gegeben, welche nicht kollinear sind, so kann bekanntlich jeder weitere Punkt U der Ebene in baryzentrischen Koordinaten der Punkte R , S und T ausgedrückt werden:

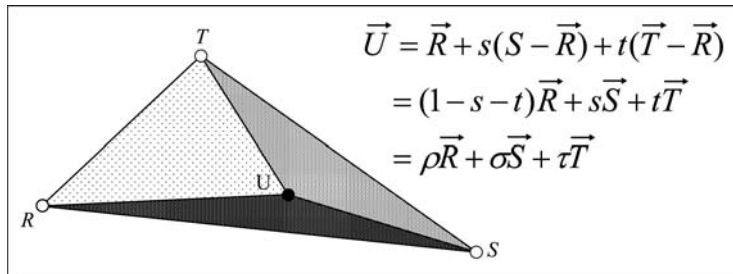


Abb. 1.38:
Baryzentrische Koordinaten

Die Koeffizienten $\rho(U)$, $\sigma(U)$ und $\tau(U)$ heißen baryzentrische Koordinaten des Punktes U bezüglich des Dreiecks RST .

Dann definieren die Funktionen

$${}_{RST}B_{i,j,k}^n(U) := \binom{n}{i,j,k} \rho(U)^i \sigma(U)^j \tau(U)^k, \quad i + j + k = n$$

die sog. verallgemeinerten Bernsteinpolynome bezüglich des Referenzdreiecks RST . Dabei ist

Verallgemeinerte Bernsteinpolynome

$$\binom{n}{i,j,k} = \frac{n!}{i!j!k!}$$

der bekannte verallgemeinerte Binomialkoeffizient. Wie die Bernsteinpolynome eine Basis für den Raum der reellwertigen univariaten Polynome (Polynome in einer Variablen) bilden, so bilden die oben definierten verallgemeinerten Bernsteinpolynome eine Basis für den Raum aller reellwertigen bivariaten Polynome (Polynome in zwei Variablen) vom Grad n .

Für jedes bivariate Polynom $q(u,v)$, $q: \mathbb{R}^2 \rightarrow \mathbb{R}^3$ existiert also eine Bézierdarstellung

$$q(u,v) = q(U) = \sum_{i+j+k=n} B_{ijk}^n(U) b_{ijk}$$

bezüglich des Referenzdreiecks RST und den Bézierpunkten $b_{ijk} \in \mathbb{R}^d$. Die Punkte b_{ijk} bilden das Kontroll- oder Béziernetz. Setzen wir für eine so gegebene Bézierfläche

$$\begin{aligned} b_{i,j,k}^0(U) &:= b_{ijk}, \quad \text{mit } i+j+k=n \\ \text{und } b_{i,j,k}^r(U) &:= \rho(U) b_{(i+1),j,k}^{r-1}(U) + \\ &\quad + \sigma(U) b_{i,(j+1),k}^{r-1}(U) + \\ &\quad + \tau(U) b_{i,j,(k+1)}^{r-1}(U), \quad \text{mit } i+j+k+r=n, \end{aligned}$$

dann ist der Funktionswert $q(U)$ mit dem letzten Glied $b_{0,0,0}^n(U)$ des rekursiven Schemas berechnet (de Casteljau für Dreiecksflächen).

Wieder sind die bekannten Resultate für diese Dreiecksflächen ableitbar:

- Liegt U im Dreieck RST , so liegt $q(U)$ innerhalb der abgeschlossenen konvexen Hülle des Kontrollpolygons.
- Fläche und Kontrollpolygon sind affin invariant.
- Die Fläche verläuft durch die drei Eckpunkte des Bézierpolygons, d.h. $q(R) = b_{n,0,0}$, $q(S) = b_{0,n,0}$ und $q(T) = b_{0,0,n}$.
- Die Fläche verläuft in den Eckpunkten $q(R)$, $q(S)$ und $q(T)$ tangential an das Kontrollnetz.
- Die Einschränkung von q auf die Gerade (S,T) (und analog für die Geraden (R,S) und (T,R)) ist eine Bézierkurve mit den Bézierpunkten $b_{0,n-k,k}$, d.h., für $U \in (S,T)$ gilt:

$$q(U) = \sum_{k=0}^n {}^T S B_k^n(U) b_{0,n-k,k}.$$

Für den Anschluss von Dreiecksflächen sei auf die Literatur verwiesen [ES97] bzw. [HL89].



<http://www.springer.com/978-3-540-36629-4>

Kompendium Medieninformatik

Medienpraxis

Schmitz, R. (Hrsg.)

2007, XII, 314 S., Hardcover

ISBN: 978-3-540-36629-4