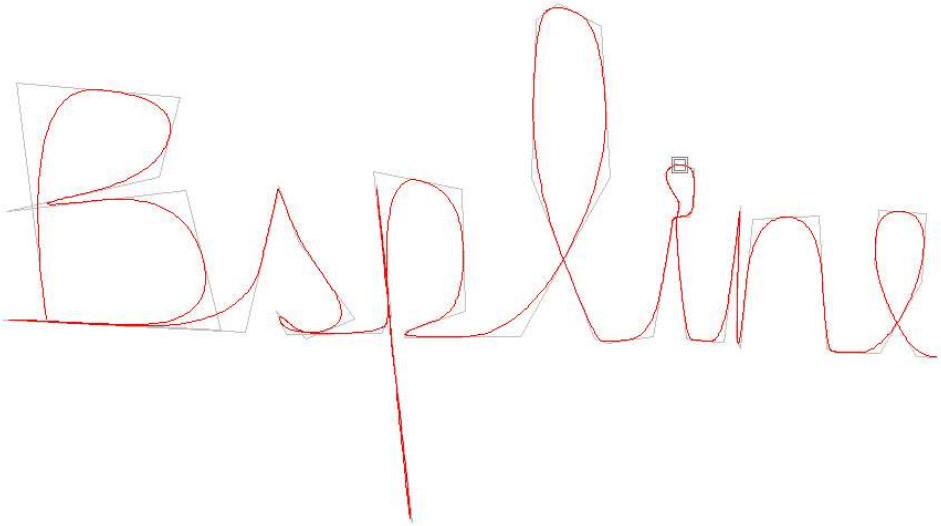

Uniform Non Rational B-Splines

Modellierung von Kurven im 2D Raum

von Stefan Beck (11.Info)

Inhalt

1. Allgemeines

1.1 Wieso Kurven ?

Kurven-Darstellung im Computer-Bereich ist heute aus fast keiner Disziplin der Computergrafik wegzudenken. In der produzierenden, der entwickelnden als auch in den forschenden Bereichen der Wirtschaft ist die Möglichkeit, Kurven mit speziellen Eigenschaften darzustellen hoch gefragt. Mich persönlich hat die Thematik von Kurven insofern gereizt da auch in der Entwicklung von Spielen eine Vielzahl von Techniken auf dem Prinzip von Kurven beruht. Weg-Punkte Systeme, Zufällig generierte Pfade, als auch Oberflächen Darstellung sind nur eine kleine Auswahl der fast unerschöpflichen Möglichkeiten die uns durch Kurven geboten werden.

1.2 Arten von Kurven zur Darstellung im Computer

Es gibt verschiedene Ansätze Kurven durch den Computer darzustellen. Es sei dabei Erwähnt dass man sich für die gegebene Problemstellung die „richtige“ Kurve aussuchen soll. Die unterschiedlichen Arten von Kurven haben jeweils vor und Nachteile, wobei es auch Kurven gibt die von der Gesamtheit der Vorteile überwiegen. Allgemein kann man sagen um so allgemeiner die Kurve formuliert ist, also wenn sie keinen Spezialfall darstellt, kann Sie auf ein größeres Aufgaben-Spektrum angewandt werden.

	Hermite	***Bézier***	***Uniform B-Spline***	***Uniformly shaped ß-Spline***	***Non uniform B-Spline***	***Catmull-Rom***	***Kochanek-Bartels***
Konvexe Hülle durch KP definiert	Nicht vorhanden	Ja	Ja	Ja	Ja	Nein	Nein
Interpolation mancher KP	Ja	Ja	Nein	Nein	Nein	Ja	Ja
Interpolation aller KP	Ja	Nein	Nein	Nein	Nein	Ja	Ja
Schwierigkeit von „sub division“	Einfach	Sehr einfach	Durchschnitt	Durchschnitt	Hoch	Durchschnitt	Durchschnitt
Stetigkeit der Representation	C^0 G^0	C^0 G^0	C^2 G^2	C^0 G^2	C^2 G^2	C^1 G^1	C^1 G^1
Einfach zu erreichende Stetigkeit	C^1 G^1	C^1 G^1	C^2 G^2	C^1 G^2	C^2 G^2	C^1 G^1	C^1 G^1
Anzahl Kontroll-Parameter	4	4	4	6	5	5	7

Abbildung 1.0 Auszug aus dem Buch : Computer Graphics Principles and Practices

1.3 Einsatzgebiete von Kurven

Kurven werden in den verschiedensten Bereichen benötigt. Eine vollständige Liste würde den Rahmen dieser Dokumentation sprengen. Einige Beispiele wären :

Auto Industrie	:	Karosserie Gestaltung.
Maschinenbau	:	Exakte Kurvenfahrt von CNC Fräßen.
Architektur	:	Modellierung gebogener Körper.
Spiele Industrie	:	Weg–Systeme, 3D Modellierung,
Medizin	:	Anatomische Körper Darstellung.
Mathematik	:	Approximation von Kurven

1.4 Unterschiedliche B-Splines.

Es gibt verschiedene Arten von B-Splines. Im folgenden werfen wir einen kurzen Blick auf die verschiedenen Arten. Hauptsächlich geht es in diesem Projekt aber um „uniform non rational B-Splines“. Das englische Wort „Spline“ hat seinen Ursprung bei den länglichen, flexiblen Metall Streifen (stripes), die von Flugzeug Erbauern zur Gestaltung der Oberfläche benutzt wurden. Über des „B“ im Wort „B-Splines“ scheint sich die Literatur teilweise unschlüssig zu sein. Am gebräuchlichsten ist „Basis“ darunter zu verstehen. Aber auch der nach de Boor benannte rekursive Algorithmus wird gerne herangezogen.

Uniform non rational B-Splines :

Diese Kurven sind die einfachsten Vertreter ihrer Gattung. Durch das anführende Wort „uniform“ wird ein gleichmäßiger Knoten Abstand innerhalb des Knoten Vektors festgelegt. Das „non rational“ dient zur Unterscheidung von rationalen Kurven.

Uniform rational B-Splines:

Hier liegt eine rationale Basis-Funkion vor, der Knoten Abstand ist aber nicht veränderbar.

Non uniform non rational B-Splines

Diese Art von B-Spline Kurven haben den Vorteil das die Knoten nicht den selben Abstand haben müssen. Somit erhält man die Möglichkeit den einzelnen Kontrollpunkten eine Gewichtung zu verleihen. Je höher die Gewichtung ist, um so stärker approximiert sich die Kurve dem gewichteten Kontrollpunkt.

Non uniform rational B-Splines (Nurbs)

Nurbs stellen die komplexeste Form von B-Splines dar. Sie besitzen Knoten mit verschiedenen Abständen sowie eine rationale Basis-Funktion.

1.5 Vorteile von B-Splines gegenüber anderen Kurven.

B-Splines haben gegenüber ander Kurven (z.B. Bézier) einen entschiedenen Vorteil. Kontrollpunkte haben bei B-Splines nur „lokale“ Auswirkungen auf den Kurven-Verlauf. Und genau diese Tatsache ist in der Praxis gewünscht. Stellen sie sich vor Sie modellieren an einer Auto-Karosse die Rückspiegel und haben Auswirkungen bis zum Heck (global), was natürlich ungünstig ist. Da ist eine Veränderung die sich maximal auf den nahen umliegenden Bereich des Rückspiegels bezieht (lokal) wesentlich wünschenswerter. Und genau diese Möglichkeit, der lokale Veränderung, erhalten Sie mit B-Splines.

Ein weiterer wesentlicher Vorteil von B-Splines ist ihre eingebaute Stetigkeit. Das heißt dass innerhalb des Kurven-Verlaufs keine Sprünge, also Strecken die nicht glatt sind, auftreten. Mehr zur Stetigkeit von B-Splines im Kapitel „Die Mathematik“.
Fügt man zwei B-Splines aneinander, kann natürlich eine Unstetigkeit auftreten. Anfügung von Kurven ist eine Thematik für sich. Jedoch sei hier soviel gesagt: Werden zweit B-Splines zusammengefügt, und die ersten n Kontrollpunkte beider Kurven haben die gleichen Koordinaten ist der Übergang stetig. Wobei n aus der Anzahl gleicher „führender“ Knoten resultiert.

Und noch einen Vorteil den B-Splines haben. Sie können rein durch Polynome beschrieben werden. Das ermöglicht seitens des Computers eine sehr effiziente Verarbeitung. Dieser Vorteil bezieht sich nur gegenüber Kurven die auf mathematische Modelle aufbauen die die Kurve nicht durch Polynome beschreibt. Alleine aus Gründen der Geschwindigkeit bauen heute in der Praxis fast alle Kurven auf seine Beschreibung durch Polynome auf.

2. Die Mathematik

2.1 Der Grundsätzliche Aufbau von B-Splines.

Ein B-Spline (Basis-Spline) repräsentiert grafisch eine Kurve. Diese Kurve wird durch so genannte Kontrollpunkte gesteuert. Die folgende Grafik zeigt dreimal die selbe B-Spline Kurve mit jeweils unterschiedlichem Grad. Jede Kurve wird durch vier Kontrollpunkte beschrieben. Wie leicht zu erkennen ist wird die Kurve mit Abnehmendem Grad „ungenauer“. In der Praxis wird aus Performance Gründen meistens Grad 3 gewählt. Wenn wir nun die einzelnen Knoten-Vektoren betrachten stellen wir fest dass mit steigendem Grad sich die Anzahl der Knoten erhöht, der letzte Wert des Knoten-Vektors jedoch sinkt. Die Kurve wird durch einen rekursiven Algorithmus aufgebaut der, die Kurve vom ersten bis zum letzten Kontrollpunkt durchläuft.
Das Intervall dessen sich der rekursive Algorithmus bedient resultiert aus dem Konten-Vektor. Es geht immer von Null bis zum Wert des letzten Knotens des Knoten-Vektors. Für den Fall mit Grad 2 würde das Intervall von 0 bis 2 gehen.
Wenn das Intervall nur von Null bis Eins geht, wie es im dritten Fall vorliegt, haben wir aus dem B-Spline eine Bézier Kurve gemacht. Das verdeutlicht dass Bézier eine Verallgemeinerung von B-Spline Kurven ist. Und hier wird auch gleich der Unterschied ersichtlich, Bézier bewegt sich nur in einem Intervall von 0 bis 1, B-Spline errechnet jedes mal wenn ein neuer Kontrollpunkt hinzukommt , ein Kontrollpunkt gelöscht wird, oder der

Grad verändert wird, das Intervall (bzw. den Knoten-Vektor) neu.
Betrachten wir uns jetzt noch die Kontrollpunkte können wir erkennen dass nur im dritten Fall (das B-Spline ist eine Bézier Kurve), die Kurve genau im letzten Kontrollpunkt endet. In den beiden ersten Fällen, wo ein reines B-Spline vorliegt, endet die Kurve schon kurz vor dem letzten Kontrollpunkt. Es kann bei B-Splines zwar vorkommen dass die Kurve im letzten Kontrollpunkt endet, im Gegensatz zu Bézier Kuren muss das aber nicht eintreten.

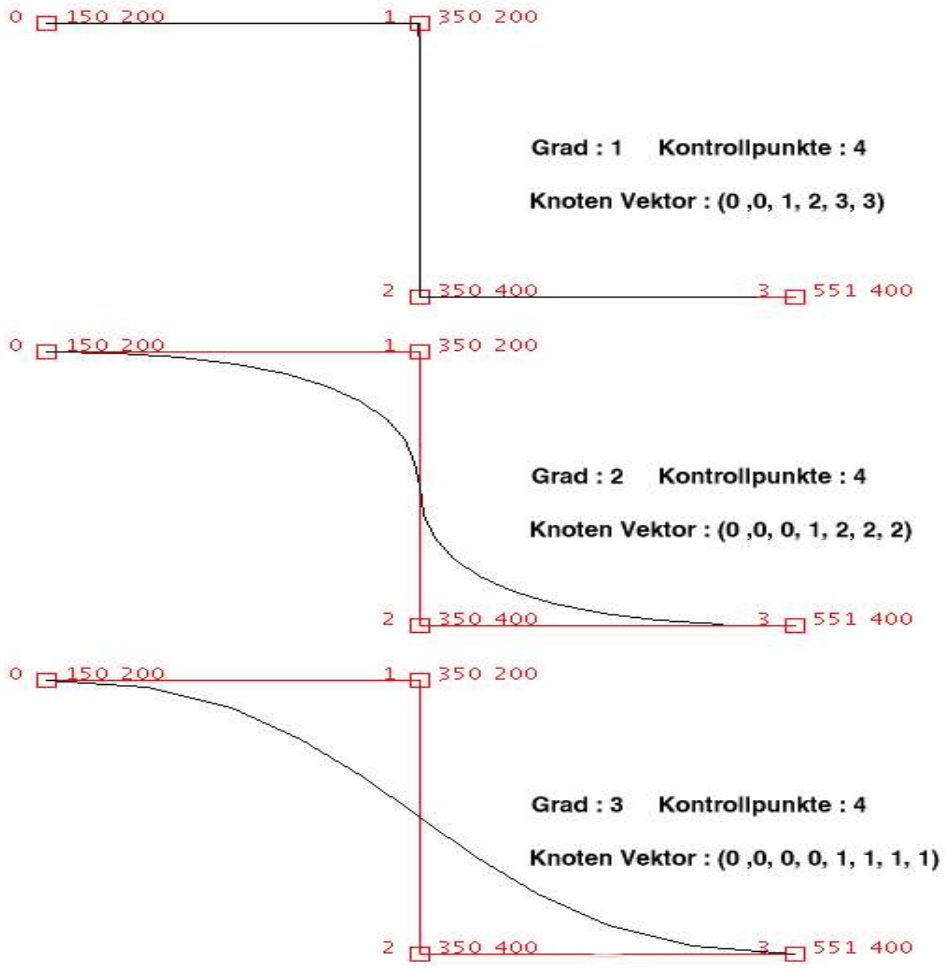

Abbildung 2.0 Drei mal die selbe Kurve mit unterschiedlichem Grad

Tragen wir abschließend noch die Parameter zusammen die ein B-Spline benötigt, und definieren sie mit entsprechenden Namen. Diese Namen werden auch im weiteren Dokument verwendet.

n	=	*Grad der Kurve*
m	=	*Anzahl der Kontrollpunkte*
KV	=	*Knoten-Vektor (dessen Länge = m+n+1)*
u	=	*Lauf-Variable (von 0 bis Wert des letzten Knotens)*

2.2 Der Knoten-Vektor

Der Knoten-Vektor bestimmt das Intervall in dem die Kurve definiert ist.
Bei der Erzeugung des Knoten-Vektors ist zu beachten dass **m immer größer als n** ist.
Er ist durch folgende Formel definiert:

$$Knoten\,Vektor = KV = [t_0 = \ldots = t_n, t_{(n+1)}, \ldots, t_m, t_{(m+1)} = \ldots = t_{(m+n)}]$$

Schauen wir uns die Formel etwas genauer an. Wenn wir das letzte und das erste Element des Knotens anschauen stellen wir fest das der Knoten-Vektor auf einem Intervall von t_0 bis $t_{(m+n)}$ definiert ist, dass heißt das unser Knoten-Vektor insgesamt m+n+1 Elemente besitzt.

Weiter können wir erkennen dass am Anfang und am Ende des Knoten-Vektors eine gewisse „Multiplizität" der Knoten herrscht. Diese Vielfachheit der Knoten bewirkt, dass die Kurve auch wirklich den ersten und letzten Kontrollpunkt interpoliert, also durchläuft. (Die dazwischenliegenden Kontrollpunkte werden , abgesehen von Grad 0 und Grad 1, nur approximiert (angenähert)) Hätte man diese multiplen Knoten nicht so würde die Kurve grafisch erst kurz nah dem ersten Kontrollpunkt beginnen, und kurz vor dem letzten enden. Betrachten wir nun einige Beispiele zu dem Knoten-Vektor.

Bsp 1: *(0, 0, 0, 1, 2, 3, 3, 3)* *n= 2, m=5, Anzahl Knoten = 8*

Bsp 2: *(0, 1, 2, 3, 4, 5, 6)* *n=0, m=6, Anzahl Knoten = 7*
In diesem Beispiel passiert die Lauf-Variable ausschließlich die Koordinaten der Kontrollpunkte ()

Bsp 3: *(0, 0, 0, 0, 1, 1, 1, 1)* *n=3, m=4, Anzahl Knoten =8*
Bei diesem letzten Beispiel handelt es sich um eine Bézier Kurve
Da der Intervall nur von 0 bis 1 geht.

Immer wenn ein Kontrollpunkt hinzugefügt, bzw. entfernt wird, so ist der Knoten-Vektor neu zu berechnen, damit sich ein neues Intervall ergibt.

2.3 Rekursion mit de Boor

Um eine Kurve entsprechend darzustellen geht man rekursiv vor. Zum näheren Verständnis erst ein kleines Beispiel. Folgende Abbildung zeigt eine Kurve die durch drei Kontrollpunkte gegeben ist. Der Grad der Kurve ist 2, demnach muss der Knoten-Vektor gleich KV=(0, 0, 0, 1, 1, 1) sein, da wir drei Kontrollpunkte haben. In diesem konkreten Beispiel hat u den Wert 0.5 und wird durch das Viereck auf halber Strecke der Kurve gekennzeichnet.

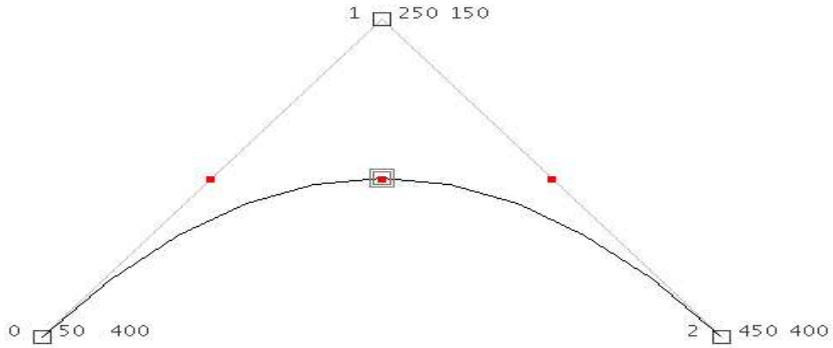

Abbildung 2.1 Kurve mit drei Kontrollpunkten vom Grad 2

Der Algorithmus von de Boor macht nichts anderes als die Teil-Strecken, (in unserem Fall die von 0 bis 1, und die von 1 bis 2) entsprechend der Lauf-Variable p hintereinander zu verbinden, und von der neu ermittelten Strecke den entsprechenden p Wert liefern.

Bevor wir uns dem Algorithmus zuwenden, noch ein letztes Beispiel für den obigen Fall. Nehmen wir an wir suchen den Punkt an dem u=0 ist. (Dieser Punkt muss zweifelsohne dem ersten Kontrollpunkt entsprechen). Nehmen wir wieder auf beiden Teil-Strecken (0 bis 1 und 1 bis 2) den nullten Punkt, (wir haben ja u=0 gewählt) so erhalten wir genau die Strecke von 0 bis 1. Von dieser erhaltenen Strecke nehmen wir als Ergebnis erneut den nullten Punkt und haben genau die Koordinaten des nullten Kontrollpunktes.
Die folgende Formel führt das zuvor in Worten beschriebene aus. Allerdings ist das obige Beispiel nur für den Grad 2. Für entsprechend höhere Grade wird die Rekursion entsprechend oft ausgeführt. Es sei jedoch erwähnt dass in der Praxis meistens Grad drei verwendet wird.

Die normalisierte B-Spline Basis Funktion sieht wie folgt aus:

$$N_i^k(u) = \frac{t - t_i}{t_{i+k} - t_i} * N_i^{k-1}(u) + \frac{t_{i+k} - t}{t_{i+k+1} - t_{i+1}} * N_{i+1}^{k-1}(u) \quad \text{mit} \quad k = 1, \ldots, n$$

Das rekursive Abbruch Kriterium ergibt sich aus:

$$N_i^0(u) := \left\{ \begin{array}{c} 1 : t_i \le u < t_{i+1} \\ 0 : rekursion\ ausführen \end{array} \right\} \quad \text{mit} \quad i = 0, \ldots, n+m-1$$

Das heißt in Worten, sobald u außerhalb des Intervalls liegt bricht die rekursive Formel ab. Grafisch gesehen baut de Boor eine pyramiden ähnliche Struktur auf.

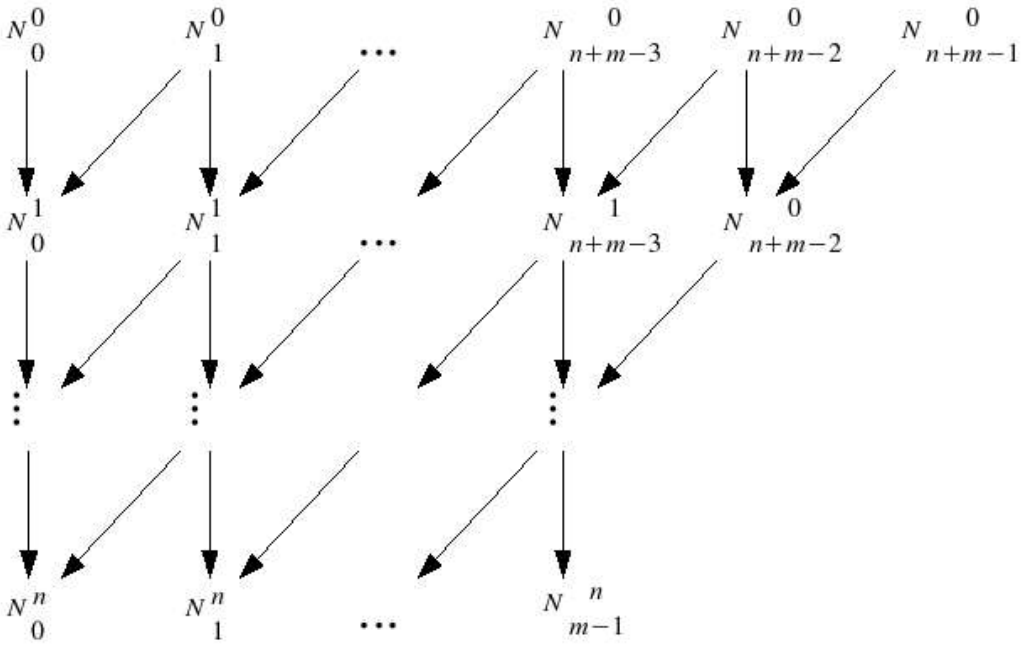

Abbildung 2.2 Schema nach dem de Boor die Kurve approximiert.

3. Das Programm

3.1 Die Grundlegende Funktionalität

Ein Programm um Kurven zu realisieren wurde in zwei verschiedenen Varianten entwickelt. Eins in Java, welches umfangreicher ist und eins in C++.
Funktionen die beide Programme aufweisen sind :

- Hinzufügen von Kontrollpunkten mit der linken Maus-Taste
- Anzeige des Knoten Vektors
- Anzeige der Koordinaten der Kontrollpunkte

3.2 Der Algorithmus in Java

Der folgende in Java realisierte Algorithmus stellt die oben erwähnte de Boor Formel dar. Sie liefert eine Vektor 3D zurück, wobei jedoch nur die x und y Komponennte benutzt wird, da wir uns im 2D Raum befinden. Wenn r == 0 ist, sprich die Rekursion hat die Koordinaten eines Kontrollpunktes erreicht so bricht die Rekursion ab und liefert die Koordinaten dieses Kontrollpunktes.
Im anderen Fall wird die rekursive Formel angewandt. Zur Programm technischen Vereinfachung habe ich doppelte Formel Teile durch die Variable „pre“ ersetzt.

```
private Vector3D deBoor(Graphics g, int k, int i, double u)
{
	// soll die Rekursion abgebrochen werden ?
	if( k == 0 ){ return ( (Vector3D) cP.get(i) ); }
	// ansonsten Rekursion weiterführen
	else
	{
		// pre calculation aus Übersichtlichkeits Gründen
		double pre = (double)(u-nV[i+k])/(nV[i+n+1]-nV[i+k]);
		return ( (Vector3D)
			(
				deBoor(g, k-1, i, u).mul(1-pre)
			).add(
				deBoor(g, k-1, i+1, u).mul(pre)
			)
		);
	}
}
```

Der nächste Quellcode Ausschnitt zeigt die Berechnung des Knoten-Vektors.
Dieser Algorithmus wird auf die Klassen interne Variable nV angewandt.
Hinter nV verbirgt sich ein statisches Feld ausreichender Größe in den die einzelnen Knoten gespeichert werden. Beim Zugriff auf dieses Feld werden jeweils nur die e ersten n+m+1 Elemente berücksichtigt, also die Anzahl der Knoten.

```
// n=degree, m=nr of controll points, nV=nodeVector
private void createNodeVector()
{
	int knoten = 0;

	// alle Knoten werden auf Null gesetzt
	for(int j=0; j<(n+m+1); j++){ nV[j] = 0; }

	for(int i=0; i<(n+m+1); i++)		// n+m+1 = Anzahl Knoten
	{
		// sobald i grösser als der Grad ist ...
		if(i>n)
		{
			// wenn i grösser als Grad und kleiner gleich
			// Knotenanzahl wird der Knotenwert erhöht
			if(i<=m){ nV[i] = ++knoten; }

			// ansonsten bleibt er gleich
			else	{ nV[i] = knoten; }
		}
		// Knoten bekommt den Wert = 0
		else{ nV[i] = knoten; }
	}
}
```

3.3 B-Splines mit Java

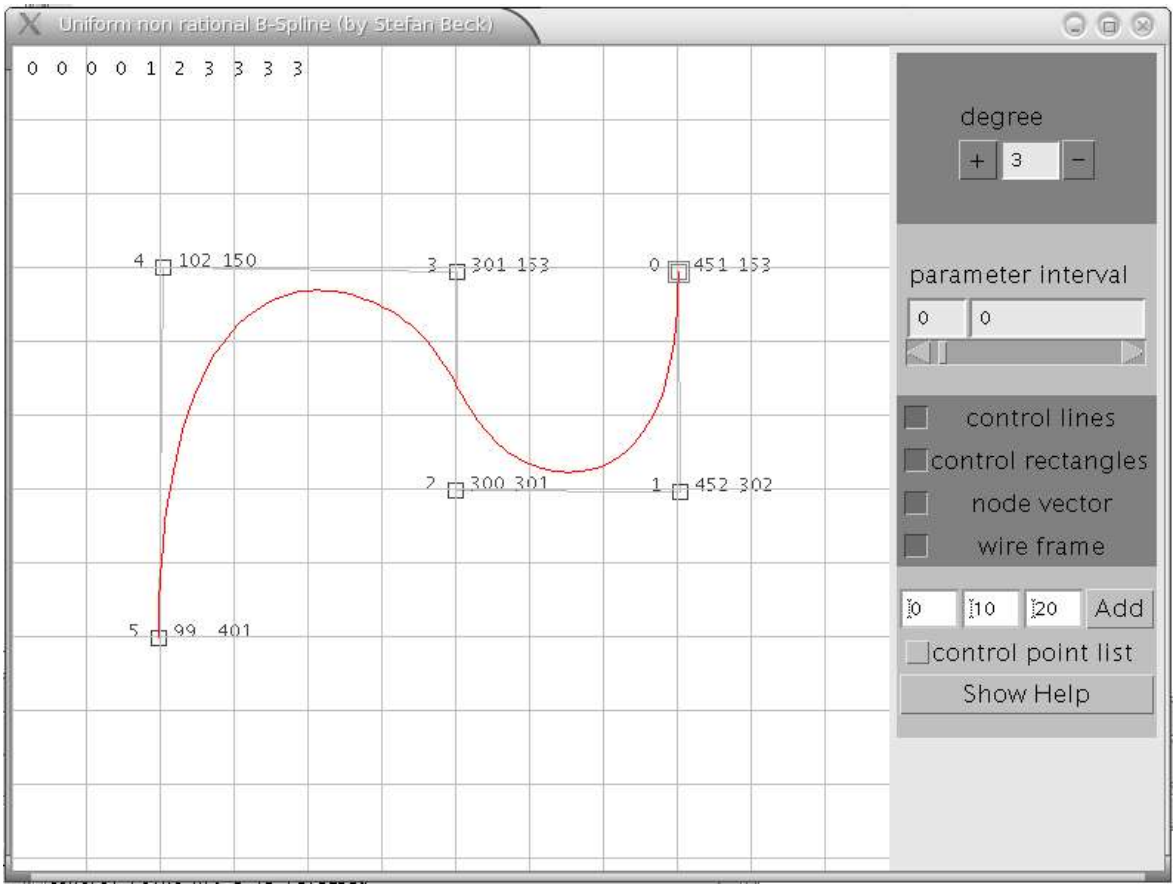

Abbildung 3.1 Das Java-Applet Programm

Das wesentlich umfangreicherer Programm unter Java bietet folgende zusätzliche Funktionen:

- Veränderung des Kurven Grades von 0 bis n

- Aktivierung / Deaktivierung der Anzeige von Kontroll-Linien, Kontrollpunkten Knoten-Vektor und Kontrollpunkt Koordinaten. Sowie ein Gitter zur einfacheren Anordnung. Es ist allerdings zu empfehlen, jedes überflüssige Kontroll-Element auszuschalten, um ein flüssigeres Bild zu erhalten.

- Ein fügen von Kontrollpunkten an beliebiger Stelle (per Maske) oder per linke Maus-Taste nach dem letzten Kontrollpunkt. Fügt man einen neuen Kontrollpunkt per Maske hinzu ist zu beachten, dass das erste Feld den Index an dem der Knoten einzufügen ist repräsentiert. Im zweiten Feld wird der X und im dritten der Y Wert angegeben.

- Löschen von beliebigen Kontrollpunkten erfolgt durch einen rechts Klick auf den entsprechenden Kontrollpunkt.

- Durchlauf des gesamten Parameter Intervall per Schiebe-Regler.
Im linken Feld wird der Startwert (dieser ist immer Null) angezeigt. Im rechten Feld ist der Wert des letzten Knotens zu sehen.

- Einblenden eine kurzen Beschreibung durch einen Hilfe Knopf.

Das Programm verwendet folgende wesentliche Klassen:

Vector3D	:	Für alle Vektor-Rechnungen
Bspline	:	Bspline Algorithmen und Hilfsfunktionen (z.B. Konrollines)
GraphicArea	:	Grafische Ausgabe
ControlArea	:	Steuer Elemente
MyFrame	:	Container für GraphicArea und ControlArea

Eine weitere Klasse namens MyFrame beinhaltet Instanzen der Klassen ControlArea und GraphicArea, wobei in ControlArea die gesamten Steuer-Elemente untergebracht sind und in GraphicArea die komplette Zeichen Arbeit vorgenommen wird.
GraphicArea wiederum beinhaltet eine Instanz von BSpline.
Die „Haupt Klasse“ namens BsplineApplicattion dient nur dazu um eine Instanz von MyFrame zu erzeugen.

3.4 B-Splines mit C++

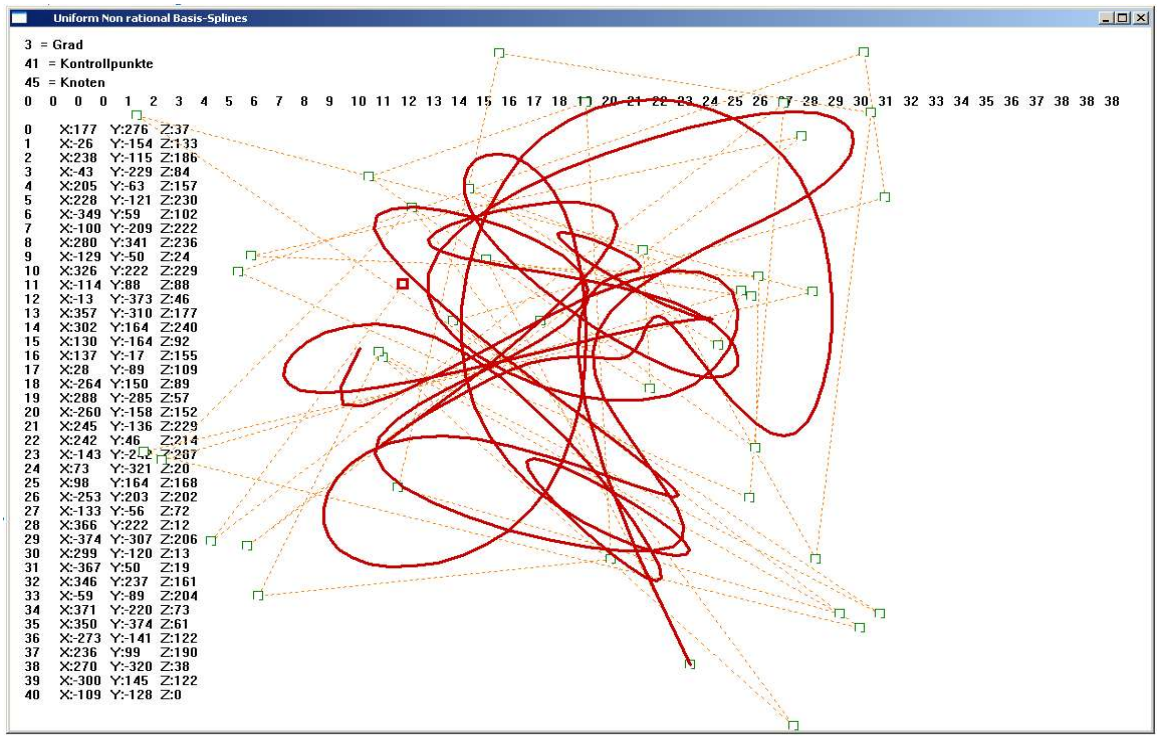

Abbildung 3.2 Das C++ Programm

Das wesentlich einfachere und auch erste Programm welches ich anfänglich programmiert hatte diente mir als ein guter Einstieg in die Materie. Zuerst wurde eine Bezier Kurve und das „picking“ (bewegen von Kontrollpunkten durch den Maus Zeiger) realisiert. Anschließend wurde der Knoten-Vektor und der de Boor Algorithmus implementiert. Dieses Programm ist mehr als Spielerei anzusehen.

Folgende Funktionen bietet die C++ Realisierung der B-Splines:

- Löschen von Kontrollpunkten durch einen rechts Klick.

- Hinzufügen von Kontrollpunkten durch einen links Klick

- Der Knoten-Vektor und eine Liste mit den Kontrollpunkten ist permanent sichtbar.

- Der Grad ist fest auf drei eingestellt und kann nicht geändert werden.

4. Die Werkzeuge

4.1 Programme zur Erstellung

Zur Visualisierung der B-Splines wurden zwei unabhängige Programme von mir erstellt. Das erste Programm das mir eine Einarbeitung und einen Einstieg über Bézier Kurven ermöglichte wurde unter Windows mit der Visual C++ IDE realisiert. Hierbei wurde massiv auf die Windows API Funktionen zurückgegriffen.
Das zweite Programm was eine größere Funktionalität bietet, wurde unter Mandrake Linux mit Java entwickelt. Die hierfür verwendete IDE ist Netbeans. Gründe einer Entwicklung unter Java ist die Unabhängigkeit von einem Betriebssystem, die einfache Veröffentlichung im Internet sowie persönliche Interesse an einer leistungsstarken C++ Alternative.
Auf die Angabe von kleinen Tools wie Packer, Browser, und sonstigen Werkzeugen wurde aus Platzgründen verzichtet.

Unter Windows:

Microsoft Windows XP	Verwendetes Betriebssystem
Microsoft Visual Studio 6.0	Verwendete IDE
C++ (Ansi) mit MS API Calls	Verwendete Hochsprache

Unter Linux:

Mandrake Linux 9.2	Verwendetes Betriebssystem
Netbeans 3.5.1	Verwendete IDE
Java 2.0 incl. Swing & AWT	Verwendete Hochsprache
Open Office 1.1.0 (Draw, Writer)	Zur Erstellung dieser Dokumentation
Gimp 1.2.5	Zur Erstellung von Grafiken

4.2 Quellen Verzeichnis

Print-Medien :

- Computer Graphics Principles and Practices | Addison Wesley
- Einführung in die Computergrafik | Vieweg

Web Publikationen :

- „Geometrische Modellierung“ von G. Greiner überarbeitet S. Bergler (PDF)