Level-Set-Methoden II
Seminar Bildsegmentierung und Computervision

Martin Lemmich

Ausarbeitung zum Vortrag vom 14. November 2005

1 Wiederholung und Einführung

Die *Level-Set-Methode* bietet eine Möglichkeit, Objekte (z.B. ein Bildsegment) im Mehrdimensionalen darzustellen, und die Bewegung dieser Objekte zu beschreiben bzw. zu berechnen. Dabei wird der Rand des Objekts (*Kontur* oder *Interface*), als Nullstelle einer mehrdimensionalen Funktion aufgefasst. Die Bewegung ist meist abhängig von der Krümmung der Kontur, dem Gradienten oder einem extern gegebenen Geschwindigkeitsfeld. In all diesen Fällen gehen wir davon aus, dass die genaue Art der Abhängigkeit vorgegeben ist. Bei der Bildsegmentierung wird später die Bewegung des Interface aus den Bilddaten gewonnen, so dass sich die Kurve möglichst genau an das zu erkennende Bildobjekt annähert. Dies ist allerdings nicht Thema dieses Vortrages.
Im Folgenden werden einige grundlegende Begriffe im Zusammenhang mit der Level-Set-Methode nochmals erläutert. In Kapitel 2 wird dann ausgeführt, wie man mit Level-Set-Methoden die Bewegung eines Objektes in Richtung des Normalenvektors darstellen kann, und wie man kombinierte Bewegungen berechnen kann. Kapitel 3 behandelt anschließend die Konstruktion einer *vorzeichenbehafteten Abstandsfunktion*, mit der sich viele Berechnungen vereinfachen lassen. In Kapitel 4 wird die Idee der *Partikel-Level-Set-Methode* dargelegt, mit der man systematische Fehler der Level-Set-Methode korrigieren kann. Kapitel 5 gibt schließlich einen kurzen Ausblick auf weitere Einsatzmöglichkeiten der Level-Set-Methode.
Die Verfahren in diesem Vortrag werden im R^2 oder im R^3 dargestellt, da dies die typische Umgebung für die Bildsegmentierung ist. Die Level-Set-Methode lässt sich allerdings problemlos auch auf höhere Dimensionen übertragen.

1.1 Die Level-Set-Methode (”Niveaumengenmethode”)

Die Level-Set-Methode stellt die Kontur eines geometrischen Objektes implizit als Nullstellenmenge einer Funktion $\phi = \phi(\vec{x})$ dar. Man verlangt von dieser Funktion, dass sie innerhalb des Objektes nur negative Werte, außerhalb des Objektes nur positive Werte, und auf der Kontur des Objektes nur den Wert 0 annimmt.

Ein Spezialfall dieser Funktion ist die sog. vorzeichenbehaftete Abstandsfunktion, die außerdem die Eigenschaft besitzt, dass der Betrag des Gradienten der Funktion gleich 1 ist.

1.2 Wichtige Gleichungen und Bezeichnungen

Der Gradient ist (im Dreidimensionalen) definiert als

$$\nabla\phi(\vec{x}) = \left(\frac{\partial\phi}{\partial x}, \frac{\partial\phi}{\partial y}, \frac{\partial\phi}{\partial z}\right) \tag{1}$$

Er steht immer senkrecht zum Interface und steigt in Richtung des "steilsten Anstieges". Den Normalenvektor erhält man, indem man den Gradientenvektor auf die Länge 1 normiert:

$$\vec{N}(\vec{x}) = \frac{\nabla\phi(\vec{x})}{|\nabla\phi(\vec{x})|} \tag{2}$$

Weiterhin ist die Krümmung definiert als:

$$\kappa = \nabla \cdot \vec{N} = \frac{\partial n_1}{\partial x} + \frac{\partial n_2}{\partial y} + \frac{\partial n_3}{\partial z} \tag{3}$$

1.3 Bewegung mit der Level-Set-Methode

Indem man der Funktion jetzt neben der räumlichen Komponente auch eine zeitliche Komponente zuordnet ($\phi = \phi(\vec{x}, t)$), kann man die Bewegung des Interface beschreiben.
Seien t^n und t^{n+1} zwei Zeitpunkte mit $t^{n+1} - t^n = \Delta t$. Dann schreiben wir $\phi(\vec{x}, t^n)$, $\phi(\vec{x}, t^{n+1})$ abkürzend auch als ϕ^n bzw. ϕ^{n+1}

Die grundlegende Bewegungsgleichung ist dann die folgende Level-Set-Gleichung, wobei das Geschwindigkeitsfeld $\vec{V} = \vec{V}(t) = (u(t), v(t), w(t))$ zugrunde liegt:

$$\phi_t + \vec{V} \cdot \nabla\phi = 0 \tag{4}$$

Versucht man die partielle Ableitung von ϕ in der Zeit zu diskretisieren, so erhält man die sog. Euler'sche Zeitdiskretisierung

$$\frac{\phi^{n+1} - \phi^n}{\Delta t} + \vec{V}^n \cdot \nabla\phi^n = 0 \tag{5}$$

Löst man diese Gleichung nach ϕ^{n+1} auf, ergibt sich

$$\phi^{n+1} = \phi^n + \Delta t(-u^n\phi_x^n - v^n\phi_y^n - w^n\phi_z^n) \tag{6}$$

Aus dieser Gleichung lässt sich ablesen, wie man von einem Ausgangswert der Funktion den Funktionswert nach einer gewissen Zeitspanne ermitteln kann. Alle auftretenden Größen sind dabei bekannt, oder können aus der ursprünglichen Funktion zur Zeit n ermitteln werden.
Nehmen wir zum Beispiel als Objekt einen Kreis mit Mittelpunkt (0,0) und Radius r. Außerdem sei ϕ die vorzeichenbehaftete Abstandsfunktion. Wir betrachten die Bewegung der Punkte $\vec{x_1} = (r, 0)$ und $\vec{x_2} = \frac{1}{2}\sqrt{2}(r, r)$ unter dem Geschwindigkeitsfeld $\vec{V} = (1, 0)$, was einer Bewegung nach rechts entspricht. Zum Ausgangszeitpunkt $t^0 = 0$ gilt $\phi^0(\vec{x_1}) = \phi^0(\vec{x_2}) = 0$, d.h. die Punkte liegen auf dem Interface. Eine Zeiteinheit später ($t^1 = 1$) ergibt sich nach Gleichung (6) $\phi^1(\vec{x_1}) = -1$ und $\phi^1(\vec{x_2}) = -\frac{1}{2}\sqrt{2}$ d.h. beide Punkte liegen nun innerhalb des Interface mit neuem Abstand. Die Qualität des neuen Wertes ist dabei stark von der Feinheit der Diskretisierung abhängig. Mit anderen Worten: Ein (im Verhältnis) zu großer Zeitabstand kann zu schlechten Werten führen. Wählen wir im obigen Beispiel $r = 1/2$, dann liegt $\vec{x_1}$ bei $t = 1$ wieder auf dem Interface, obwohl wir gesehen hatten, dass $\phi^1(\vec{x_1}) = -1 \neq 0$. Dies widerspricht den oben gestellten Forderungen an die Funktion. Hätte man den Zeitabstand kleiner gewählt, so hätte man einen besseren Wert erhalten.

1.4 Upwind-Differencing

Wir untersuchen nun nur die x-Komponente der Eulerschen Zeitdiskretisierung (5). Dies führt zur Gleichung

$$\phi^{n+1} = \phi^n + \Delta t(-u^n \phi_x^n) \tag{7}$$

Nun stellt sich die Frage nach einer sinnvollen Approximation für den unbekannten Wert ϕ_x^n am Gitterpunkt x_i. Der Upwind-Ansatz (dt.: "windwärts") schlägt hierbei folgendes Schema vor:

- Ist $u_i < 0$, so wählt man $\phi_x^+ := \frac{\phi(x_{i+1}) - \phi(x_i)}{\Delta x}$,
- Ist dagegen $u_i > 0$, wählt man $\phi_x^- := \frac{\phi(x_i) - \phi(x_{i-1})}{\Delta x}$

Dabei ist Δx der jeweilige Abstand der Gitterpunkte. Die Philosophie dieses Ansatzes ist, die Informationen aus der Richtung, aus der das Interface sich bewegt, mit einzubeziehen. Genauso wird ϕ_y^n und ϕ_z^n bestimmt.

2 Bewegung in Normalenrichtung

Wir betrachten jetzt die Bewegung in Normalenrichtung, d.h. das Geschwindigkeitsfeld ist gegeben durch $\vec{V} = a\vec{N}$ (a konstant). Daraus ergibt sich mit (4) die Level-Set-Gleichung

$$\phi_t + a|\nabla\phi| = 0 \tag{8}$$

Ist hierbei $a > 0$, so dehnt sich das Objekt aus. Ist $a < 0$, so zieht es sich zusammen.

2.1 Vereinfachung durch vorzeichenbehaftete Abstandsfunktion

Ist nun ϕ eine vorzeichenbehaftete Abstandsfunktion (d.h. $|\nabla\phi| = 1$), so vereinfacht sich (8) zu $\phi_t = -a$. Die Eulersche Zeitdiskretisierung liefert dann

$$\phi^{n+1} = \phi^n - a\Delta t \tag{9}$$

Anschaulich gesprochen wird die $\phi = 0$-Höhenlinie nach einem Zeitschritt zur $\phi = -a\Delta t$-Höhenlinie, die $\phi = a\Delta t$-Höhenlinie zur $\phi = 0$-Höhenlinie. Schreibt man (9) in Gradientenversion, so ergibt sich $\nabla\phi^{n+1} = \nabla\phi^n$ (da $a\Delta t$ (räumlich) konstant ist). Das heißt, wenn ϕ^n eine vorzeichenbehaftete Abstandsfunktion ist, so ist ϕ^{n+1} wiederum eine vorzeichenbehaftete Abstandsfunktion.

2.2 Upwind-Differencing bei Bewegung in Normalenrichtung

Durch Umformen der Level-Set-Gleichung für die Bewegung in Normalenrichtung erhalten wir
(8) $\Leftrightarrow \phi_t + a\frac{(\nabla\phi)^2}{|\nabla\phi|^2} \cdot |\nabla\phi| = 0 \Leftrightarrow$

$$\phi_t + \left(\frac{a\phi_x}{|\nabla\phi|}, \frac{a\phi_y}{|\nabla\phi|}, \frac{a\phi_z}{|\nabla\phi|}\right)\nabla\phi = 0 \tag{10}$$

Wir untersuchen wiederum nur die x-Komponente des Vektors: $a\phi_x|\nabla\phi|^{-1}$ (= "Geschwindigkeit in x-Richtung", vgl. "u"im externen Geschwindigkeitsfeld). Wenn wir jetzt den Upwind-Differencing-Ansatz zur Diskretisierung verwenden wollten, müssen wir das Vorzeichen von $a\phi_x|\nabla\phi|^{-1}$ untersuchen. Nun stellt sich das Problem, dass dieses Vorzeichen von der verwendeten Approximation ϕ_x^-

oder ϕ_x^+ abhängt, d.h. möglicherweise kommt, je nachdem welche der Approximationen verwendet wird, ein unterschiedliches Vorzeichen heraus.

Falls ϕ_x^- und ϕ_x^+ das gleiche Vorzeichen haben, so können wir die Methode wie bisher anwenden. Für denn Fall $a > 0$ verwenden wir ϕ_x^-, falls beide positiv sind und ϕ_x^+, falls beide negativ sind. Problematisch ist nur der Fall, wenn ϕ_x^+ und ϕ_x^- unterschiedliche Vorzeichen haben. Eine Lösung hierfür liefert zum Beispiel die Godunov-Methode, die für den Fall $\phi_x^+ > 0$ und $\phi_x^- < 0$ $\phi_x := 0$ als Approximation vorschlägt, und für den Fall $\phi_x^+ < 0$ und $\phi_x^- > 0$ das betragsmäßige Maximum aus beiden.

2.3 Bewegung in Normalenrichtung kombiniert mit krümmungsabhängiger Bewegung und einem externen Geschwindigkeitsfeld

Kombiniert man die Bewegung in Normalenrichtung mit einer krümmungsabhängigen Bewegung, ergibt sich die Level-Set-Gleichung

$$\phi_t + a|\nabla\phi| = b\kappa|\nabla\phi| \tag{11}$$

Hier ist es möglich, den hyperbolischen Term $a|\nabla\phi|$ und den parabolischen Term $b\kappa|\nabla\phi|$ unabhängig voneinander zu diskretisieren. Dies geschieht für den ersten Ausdruck wie oben beschrieben, und für den zweiten Ausdruck z.B. über zentrale Differenzen (vgl. vorheriger Vortrag). Anschließend kann man den Term über die Euler-Diskretisierung in der Zeit entwickeln.

Fügt man nun außerdem noch ein externes Geschwindigkeitsfeld hinzu, erweitert sich (11) zu

$$\phi_t + \vec{V} \cdot \nabla\phi + a|\nabla\phi| = b\kappa|\nabla\phi| \tag{12}$$

Man kann nun den parabolischen Term $b\kappa|\nabla\phi|$ erneut unabhängig diskretisieren. Allerdings hat man nun zwei hyperbolische Terme $\vec{V} \cdot \nabla\phi$ und $a|\nabla\phi|$. Für denn Fall, dass beide Effekte den Gitterpunkt in einer Komponente in die gleiche Richtung bewegen, kann man wieder den Upwind-Ansatz verfolgen. Ist dies nicht der Fall, so muss für den entsprechenden Punkt der dominierende Effekt bestimmt werden.

3 Konstruktion vorzeichenbehafteter Abstandsfunktionen

Wir hatten gesehen, das viele Berechnungen im Umfeld der Level-Set-Methode vereinfacht werden können, wenn das Interface implizit durch eine vorzeichenbehaftete Abstandsfunktion gegeben ist. Es ist also wünschenswert, zu Beginn der Bewegung die Kontur durch eine solche Funktion zu initialisieren bzw. im Verlauf der Berechnung wieder auf diese Form zu bringen (”Reinitialisierung ”).

Im Folgenden wird ein Algorithmus dargestellt, der ein (implizit) gegebenes Interface durch eine vorzeichenbehaftete Abstandsfunktion darstellt.

3.1 Crossing Times - Schnittzeitpunkte mit dem Interface

Ein Konzept, um den Abstand eines Punktes vom Interface zu ermitteln ist die Entwicklung des Interfaces in Normalenrichtung auf den Punkt zu. Wählt man dabei den Parameter a in (8) als 1 (falls der Punkt außerhalb des Objekts liegt) oder -1 (wenn der Punkt im inneren liegt), dann kann man (durch Interpolation) den Zeitpunkt t_0 , an dem der Punkt das Interface schneidet (d.h. das Vorzeichen wechselt) genau bestimmen. Dieser Zeitpunkt ist (wg. $|a| = 1$) gleich dem Betrag des Abstandes des ursprünglichen Punktes zum Interface.

3.2 Die Fast-Marching-Methode

Die Fast-Marching-Methode ist nun ein Algorithmus zur Berechnung einer vorzeichenbehafteten Abstandsfunktion. Sie besteht aus den folgenden Schritten

1. Initialisiere ein Punkteband entlang der Kontur.
2. Berechne einen vorläufigen Wert für alle an das Band angrenzenden Punkte.
3. Füge Punkt mit niedrigstem Wert zum Band hinzu.
4. Aktualisiere vorläufige Werte der Nachbarn, springe zu 3., falls noch nicht alle Punkte im Band sind.

Wir behandeln hier den Algorithmus für den zweidimensionalen Fall.

Verwaltung der Punkte: Ein Problem dieser Methode ist die Verwaltung des vorläufigen Abstandswertes an den Gitterpunkten. Da häufig auf das Minimum des Abstandes zugegriffen wird, ist es notwendig hierfür eine effiziente Implementierung zu wählen. Eine Möglichkeit ist es, die Punkte in einem *Heap* zu verwalten. Ein Heap ist ein Binärbaum, der möglichst vollständig besetzt ist, d.h. nur in der untersten Ebende Lücken aufweisen kann, und bei dem der Elternknoten stets kleiner als die Kindknoten ist.
Die einzelnen Schritte lassen sich mit dem Heap bequem ausführen.
Schritt 2 startet mit dem leeren Heap. Für jeden neu berechneten Wert wird der neue Punkt an der ersten freien Position angehängt, und die Heap-Eigenschaft ggf. durch (eventuell mehrmaliges) Vertauschen mit dem Elternknoten wiederhergestellt.
In Schritt 3 wird die Wurzel (wegen Heap-Eigenschaft Minimum der Werte) aus dem Heap entfernt und zum Punkteband hinzugefügt. Danach wird so lange der jeweils kleinere Kindknoten an die freie Position gesetzt, bis der Baum wieder vollständig ist.
In Schritt 4 schließlich werden Punkte, zu denen noch kein vorläufiger Wert existiert in der untersten Ebene angehängt. Ist der vorläufige Wert schon berechnet, so wird er aktualisiert und durch vertauschen mit Elternknoten oder kleinerem Kindknoten die Heapeigenschaft wiederhergestellt.
Insgesamt haben alle Schritte eine Komplexität von einzeln höchstens O($log_2(n)$) wogegen bei Punkteverwaltung in einer Liste das Bestimmen des Minimums schon eine Komplexität von O(n) hätte. Dies ist ein enormer Vorteil und ermöglicht dem Algorithmus, mit einer großen Menge an Gitterpunkten umzugehen.

Schritt 1: Nun zur Initialisierung des Punktebandes: Wir gehen davon aus, dass das Interface bereits implizit (durch die Funktionswerte an den Gitterpunkten) gegeben ist. Das heißt diejenigen Gitterpunkte an denen die Funktionswerte das Vorzeichen wechseln, bilden unsere einzige Information über die Lage des Interface. Obwohl wir diese Werte direkt nutzen müssen, ist es möglich, dass diese Werte offensichtlich nicht den Abstand zum Interface beschreiben. Wenn zum Beispiel 2 Punkte auf unterschiedlichen Seiten des Interface zueinander einen kleineren Abstand haben, als die Summe der Beträge ihrer Funktionswerte, so können diese Werte offensichtlich keinen Abstand beschreiben. Die Idee ist nun, den Funktionswert benachbarter Punkte, zwischen denen das Interface liegt, so zu ändern, dass die neuen Werte in Summe den Abstand zwischen den Punkten ergeben, aber ihr Verhältnis zueinander gleich bleibt. Wir betrachten also alle $\phi_{i,j} = \phi(x_i, y_j)$, $\phi_{i+1,j}$ mit unterschiedlichen Vorzeichen und berechnen einen Kandidaten für Abstand durch Interpolation: $\phi_{i,j}^{neu} = \phi_{i,j}^{alt} \cdot \frac{\Delta x}{|\phi_{i,j}^{alt} - \phi_{i+1,j}^{alt}|}$. Analog

verfahren wir mit $\phi^{neu}_{i+1,j}$. Führt man diese Methode in allen Koordinatenrichtungen durch, kann man mehrere Kandidaten entsprechend der Dimension erhalten. Der (betragsmäßig) kleineste Kandidat wird als neuer Wert verwendet.

Schritt 2: Die Berechnung der vorläufigen Werte wird nun für Punkte außerhalb des Objekts erläutert. Für die Punkte im Inneren läuft das Verfahren analog durch "vertauschen"von außen und innen.
$\phi_{i,j}$ ist nur von denjenigen Werten an den vier Nachbarpunkten $x_{i\pm1}, y_j$ und $x_i, y_{j\pm1}$ abhängig, die schon dem Punkteband hinzugefügt wurden. Wir berechnen nun jeweils einen Wert für jeden der vier möglichen Quadranten aus x_i, y_j und $x_{i\pm1}, y_j$ und $x_i, y_{j\pm1}$. Das Minimum dieser Werte speichern wir dann als vorläufig berechneten Wert im Heap.

Ist in einem Quadranten nur ein Nachbarwert ϕ_1 bereits endgültig berechnet, so setzt man $\phi_{i,j} = \phi_1 + \Delta x$. Sind beide Nachbarwerte ϕ_1 und ϕ_2 berechnet, so ermittelt man den Abstand durch Interpolation auf der Verbindungsstrecke $\overline{\phi_1\phi_2}$ und berechnet $\phi_{i,j}$ als $\underset{\vec{h}\in\overline{\phi_1\phi_2}}{min}\left\{\phi(\vec{h}) + |(x_i, y_j) - \vec{h}|\right\}$ Dies führt zu der Gleichung

$$\phi_{i,j} = \underset{\theta_1+\theta_2=\Delta x, \theta_i\geq 0}{min}\{\theta_1\phi_1 + \theta_2\phi_2 + |(\theta_1, \theta_2)|\} \tag{13}$$

Mittels der Lagrangeschen Mutliplikatorenregel und einer kurzen Rechnung erhält man schließlich

$$(\phi_{i,j} - \phi_1)^2 + (\phi_{i,j} - \phi_2)^2 = (\Delta x)^2 \tag{14}$$

Die größere Lösung dieser quadratischen Gleichung wählt man nun als vorläufigen Wert.

4 Die Partikel-Level-Set-Methode

Ein Problem der Level-Set-Methode ist, dass im Verlauf der Bewegung ein Objekt "verfälscht"wird. In Abbildung 1 sieht man beispielsweise, wie die Rotation einer eingekerbten Kugel mittels der Level-Set-Methode berechnet wird und der Körper dabei seine charakteristische Struktur verliert. In Abbildung 2 wird ein sog. "Enright-Test"mit der Level-Set-Methode durchgeführt, bei der ein Körper erst verformt wird, und dann wieder in seinen Ausgangszustand zurücktransformiert werden soll. Hier sieht man, dass der Ausgangszustand nicht erreicht wird, und das Objekt einen Großteil seiner Substanz verliert.

Eine Möglichkeit, dem entgegenzuwirken ist die Partikel-Level-Set-Methode. Dabei werden einige Punkte in der Nähe des Interface eingestreut, und ihnen ein Einflussradius gleich dem Abstand zum Interface zugewiesen. Diese Punkte werden dann explizit (im Gegensatz zur impliziten Bewegung der Level-Set-Methode) mitbewegt. Kreuzt nun ein Punkt im Verlauf dieser Bewegung das Interface, so liegt ein Fehlverhalten der Level-Set-Methode vor. Dies wird repariert, indem der Einflussradius des Punktes zum Objekt hinzugefügt wird, falls sich der Punkt fälschlicherweise aus dem Objekt herausbewegt hat bzw. der Einflussradius abgezogen wird, falls sich der Punkt fälschlicherweise ins Objekt hineinbewegt hat. In den Abbildungen 3 und 4 sieht man, wie die Rotation und der Enright-Test mit der Partikel-Level-Set-Methode sehr gute Ergebnisse zeigen.

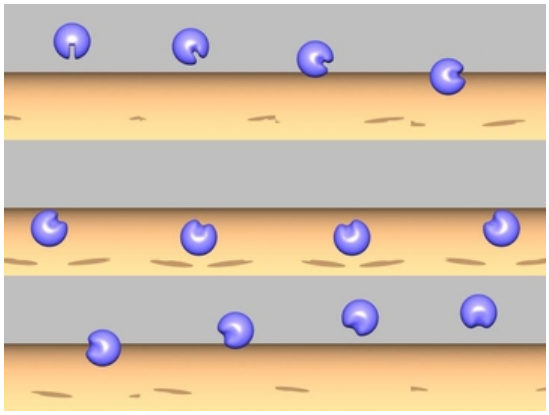

Abbildung 1: Rotation eines eingekerbten Körpers mit herkömmlicher Level-Set-Methode

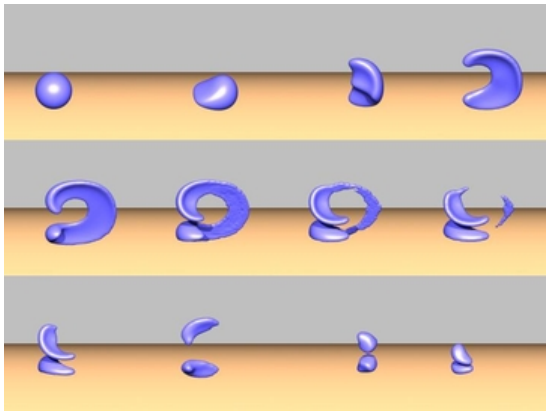

Abbildung 2: Enright-Test mit herkömmlicher Level-Set-Methode

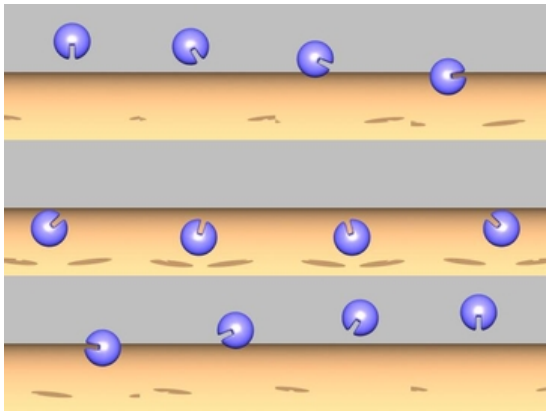

Abbildung 3: Rotation mit Partikel-Level-Set-Methode

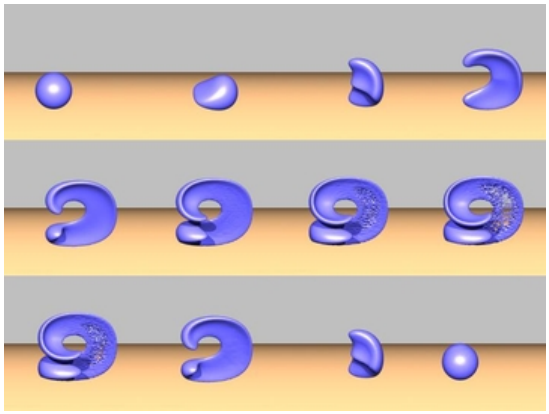

Abbildung 4: Enright-Test mit Partikel-Level-Set-Methode

5 Ausblick

5.1 Objekte mit Kodimension 2

Bisher haben wir die Level-Set-Methode verwendet, um eine Kontur zu beschreiben, deren Dimension um 1 niedriger ist, als die Umgebung, d.h. eine Kurve im R^2 oder eine Oberfläche im R^3. Man sagt auch, das Interface hat Kodimension 1. Mit der Level-Set-Methode kann man allerdings auch Konturen mit Kodimension 2 betrachten (typischerweise Kurven im R^3). Dabei wird das Interface als Schnittmenge der Nullstellen zweier Funktionen dargestellt. So definiert man beispielsweise den Tangentenvektor eines Objektes, das durch die Funktionen ϕ_1 und ϕ_2 gegeben ist als

$$\vec{T} = \frac{\nabla\phi_1 \times \nabla\phi_2}{|\nabla\phi_1 \times \nabla\phi_2|} \tag{15}$$

Davon ausgehend kann man sich Krümmung, Normalenvektor, Binormalenvektor und Torsion definieren, und die Bewegung der Kurve mittels oben beschriebener Methoden über die Funktionen ϕ_1 und ϕ_2 berechnen.

5.2 Offene Kurven und Oberflächen

Außerdem kann man mit der Level-Set-Methode auch offene Kurven und Oberflächen, deren Rand im Inneren des betrachteten Gebiets liegen, darstellen. Dabei wird der Rand als Schnitt der Nullstellen aus zwei Funktionen betrachtet. Die zweite dieser Funktionen beschreibt dann eine ”fiktive Kurve”(sog. ”ghost curve”) die zur Berechnung der ”realen”Kurve mitbewegt wird. Beispielsweise betrachtet man die Null-Isokontur von $\phi(x, y) = y$ (reale Kurve) in dem Bereich in dem $\psi(x, y) = |x| - 1$ (”ghost curve”) kleiner Null ist. Beide Kurven werden dann mit der Level-Set-Methode bewegt.

6 Quellenangabe

6.1 Literatur

Der Vortrag basiert auf ”Level Set Methods and Dynamic Implicit Surfaces”von Stanley Osher und Ronald Fedkiw, Springer 2003, vorwiegend §6-10.

6.2 Bilder

Die Bilder stammen von Ronald Fedkiws Homepage http://graphics.stanford.edu/∼fedkiw/ (letzter Zugriff am 12.11.2005). Dort findet man auch einige schöne Animationen die unter Verwendung von Level-Set-Methoden erstellt wurden.