Constructing Signed Distance Functions

Christoph Meier

7. Dezember 2010

1 Motivation

Im bisherigen Verlauf des Seminars wurde deutlich, dass sich viele Probleme einfacher handhaben lassen, wenn ϕ eine Signed Distance Function (SDF), also eine vorzeichenbehaftete Abstandsfunktion ist. Die Beziehung $|\nabla\phi| = 1$ lässt sich ausnutzen und vereinfacht die zugrundeliegenden Differentialgleichungen.
Leider bleibt diese nützliche Eigenschaft nicht erhalten, wenn man die bisher besprochenen Verfahren anwendet, um die PDE zu lösen. Ziel dieses Kapitels ist es, Techniken zu entwickeln, sodass die SDF-Eigenschaft erhalten bleibt.

2 Reinitialisierung

Eine naheliegende Idee ist es, ϕ nach jedem Zeitschritt zu *reinitialisieren*, also wieder zu einer Signed Distance Function zu machen. Das liefert folgenden Basisalgorithmus:

1. Initialisiere ϕ als SDF
2. Führe einen Zeitschritt durch
3. Reinitialisiere ϕ
4. Gehe zu 2.

Im einfachsten Fall liegt ϕ zu Beginn natürlich bereits als SDF vor. Ist dies nicht der Fall, muss eine Initialisierung vorgenommen werden. Man möchte also, dass $\phi = \pm d$, wobei d der Abstand des Punktes zur $\phi = 0$-Isokontur ist und das Vorzeichen beschreibt, ob x in Ω^+ bzw. Ω^- liegt. Bei der Initialisierung können natürlich die selben Methoden wie bei der Reinitialisierung verwendet werden. Da nur die $\phi = 0$-Isokontur von Bedeutung ist, genügt es, ϕ in Schritt 2 nur an Punkten in einer Umgebung dieser Isokontur zu berechnen und die restlichen Punkte erst in Schritt 3 zu aktualisieren. Dies liefert größere Freiheiten für das numerische Verfahren, da es nur für Punkte nahe des Interfaces "gutartig" sein muss. Man bezeichnet dieses Vorgehen als *local level set method*.
Die einfachste Methode ϕ zu einer SDF zu machen besteht darin, die $\phi = 0$-Isokontur zu diskretisieren und für alle Gitterpunkte den Abstand explizit zu berechnen. Das kann jedoch sehr lange dauern, außerdem ist es oft schwer, das Interface zu lokalisieren und zu diskretisieren. Nachfolgend werden daher andere Techniken zur Reinitialisierung vorgestellt.

2.1 Crossing Times

Wie oben erwähnt kann es unter Umständen sehr schwer sein, das Interface zu lokalisieren und zu diskretisieren. Mit der Methode der *Crossing Times* kann man dieses Problem umgehen: Man betrachte dazu einen Punkt x in Ω^+. Um den Abstand zum Interface zu bekommen, wird das Interface mittels der Gleichung $\phi_t + a\,|\nabla\phi| = 0$ aus dem letzten

Vortrag mit $a = 1$ in Normalenrichtung nach außen bewegt. Speichert man die Werte von ϕ bei x nach jedem Zeitschritt, so kann man mittels Interpolation feststellen, zu welchem Zeitpunkt t_0 das Interface x passiert hat. t_0 wird daher *Crossing Time* genannt. Da die Bewegung in Normalenrichtung mit konstanter Geschwindigkeit $a = 1$ geschieht, ist der Wert von t_0 identisch zum Wert des Abstands d von x zum Interface. Für Punkte aus Ω^- funktioniert die Überlegung analog mit $a = -1$. Damit kann man für jeden Gitterpunkt den Abstand zum Interface bestimmen und ϕ zu einer SDF updaten.

2.2 Die Reinitialisierungsgleichung

Damit ϕ eine SDF ist, muss $|\nabla\phi| = 1$ gelten. Die Idee bei der Reinitialisierungsgleichung ist, zur linken Seite dieser Gleichung den Term ϕ_t zu addieren und die resultierende Gleichung

$$\phi_t + |\nabla\phi| = 1 \tag{1}$$

bis zu einem *steady-state*, also einem zeitlich konstanten Zustand zu lösen (denn dann gilt $\phi_t = 0$ und (1) vereinfacht sich zu $|\nabla\phi| = 1$). Gleichung (1) transportiert Information in Normalenrichtung nach außen. Daher fließt Information von kleineren Werten von ϕ zu größeren. Dabei wird aber die $\phi = 0$-Isokontor durch die negativen Werte von ϕ beeinflusst und bleibt nicht an der gleichen Stelle. Deshalb kann diese Gleichung nur in Ω^+ angewendet werden. Je nachdem welche Ordnung das angewendete Verfahren haben soll, ist es nötig, einen Streifen von Werten um das Interface zu initialisieren, die als Anfangswerte benutzt werden. Entsprechend muss für Punkte aus Ω^- die Gleichung

$$\phi_t - |\nabla\phi| = -1 \tag{2}$$

bis zu einem steady-state gelöst werden, wobei auch hier eventuell zuerst ein Streifen um $\phi = 0$ initialisiert werden muss.

(1) und (2) zusammengefasst ergeben die *Reinitialisierungsgleichung*:

$$\phi_t + S(\phi_0)(|\nabla\phi| - 1) = 0 \tag{3}$$

wobei ϕ_0 die Anfangswerte von ϕ beinhaltet und $S(\phi_0)$ eine Signumsfunktion ist mit Werten 1 in Ω^+, -1 in Ω^- und 0 auf dem Interface. Mit dieser Gleichung lassen sich alle Gitterpunkte aktualisieren, denn die Signumsfunktion entscheidet, ob Gl. (1) oder (2) anzuwenden ist. Außerdem bleibt die $\phi_t = 0$-Isokontur erhalten, denn für Punkte auf dem Interface ist $S(\phi_0) = 0$ und damit wird Gleichung (3) zu $\phi_t = 0$, also zeitlich konstant.
Solange ϕ relativ glatt ist, funktioniert diese Methode gut. Ist jedoch ϕ nicht glatt oder auf einer Seite des Interfaces viel steiler als auf der anderen, können Fehler entstehen. Numerische Tests zeigen, dass eine Glättung der Signumsfunktion zu besseren Ergebnissen führt. So ist

$$S(\phi_0) = \frac{\phi_0}{\sqrt{\phi_0^2 + (\Delta x)^2}} \tag{4}$$

eine gute Wahl,

$$S(\phi) = \frac{\phi}{\sqrt{\phi^2 + |\nabla\phi|^2 (\Delta x)^2}} \tag{5}$$

sogar noch besser.
In Gleichung (4) ist es wichtig, $S(\phi)$ regelmäßig neu zu berechnen, damit der $|\nabla\phi|$-Term den gewünschten Effekt hat, während Gleichung (3) nur einmal mittels den Anfangswerten ϕ_0 berechnet wird.
Idealerweise bleibt das Interface bei der Reinitialisierung erhalten, numerische Fehler können aber zur Folge haben, dass das Interface sich bis zu einem gewissen Grad ändert. Je nach zugrunde liegendem Problem kann in Gleichung (3) ein Term hinzugefügt werden, der eine Flächen-oder Volumenerhaltung repräsentiert (z.B. bei einer Flüssigkeitssimulation). Ändert sich das Interface bei der Reinitialisierung nicht, bleibt das Volumen erhalten. Eine detaillierte Ausführung dazu ist im Buch ab Seite 67 zu finden.

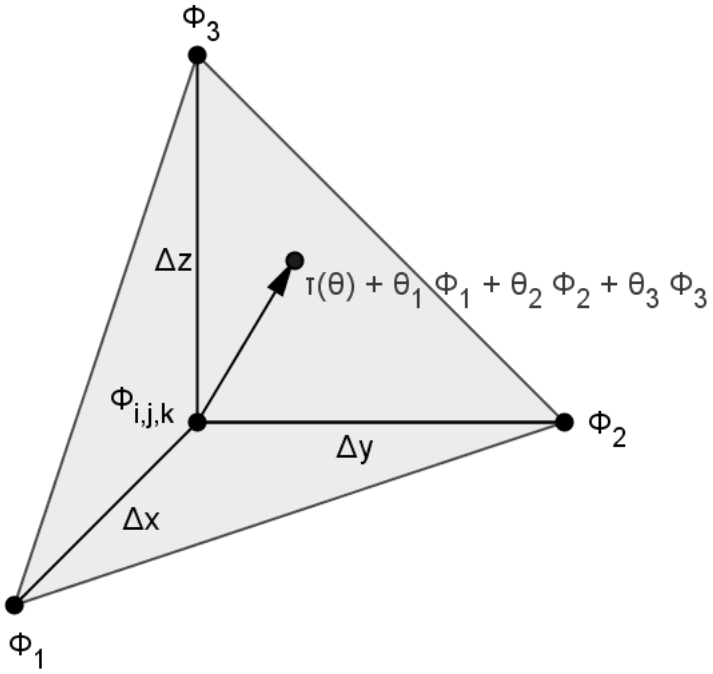
2.3 Die Fast Marching Method

Bei der *Crossing-Time*-Methode bewegt sich das Interface in Normalenrichtung nach außen und überschreitet die Gitterpunkte nach einer Zeit äquivalent zu deren Abstand zum Interface. Der Abstand jedes Punktes wird also berechnet, wenn das Interface den Punkt passiert. Bei der Fast Marching Methode geht man ähnlich vor, indem man vom Interface beginnend alle Gitterpunkte abläuft und ϕ berechnet. Die hier vorgestellte Methode beschränkt sich auf Punkte in Ω^+, die sich leicht für Ω^- anpassen lässt.
Angenommen, alle dem Interface benachbarten Gitterpunkte sind mit den korrekten Abstandswerten initialisiert (dazu später mehr). Ausgehend von diesem Streifen möchte man jeden Gitterpunkt außerhalb ablaufen und den Abstand basierend auf den bereits berechneten Werten bestimmen.
Im Detail: Beginne mit einem *Startbereich* von Werten von Punkten, die dem Interface am nächsten sind. Markiere diese Punkte als *akzeptiert*. Markiere alle Nachbarn der Punkte aus *akzeptiert* (die selbst nicht schon *akzeptiert* sind) als *nahe*, den Rest als *fern*. Dann beginnt der Algorithmus:

1. Sei x der Punkt in *nahe* mit dem kleinsten Abstandswert
2. Markiere diesen Punkt als *akzeptiert*
3. Setze alle Nachbarn von x, die in *fern* liegen, auf *nahe*
4. Aktualisiere die Abstände alle Nachbarn von x aus *nahe*, nur mit Hilfe von Werten aus *akzeptiert*
5. Gehe zu 1

Der aufwendigste Teil dieses Algorithmus ist die Suche nach dem Punkt mit dem minimalen Abstandswert. Hierzu verwendet man einen Binärbaum zur Speicherung der Abstandswerte aus *nahe*, indem jeder Knoten einen kleineren Abstandswert hat als die beiden unter ihm. Daher ist der Punkt mit dem kleinsten Wert immer ganz oben im Baum zu finden. Neue Punkte werden ganz unten hinzugefügt und danach der Baum neu geordnet. Nimmt man jetzt den obersten Punkt heraus, rücken die Punkte darunter entsprechend nach. Die aktualisierten Werte aus Schritt 4 des Algorithmus werden eingefügt und der Baum eventuell nochmal geordnet. Insgesamt hat dieser Algorithmus eine sehr schnelle Laufzeit von $\mathcal{O}(N\ log\ N)$, wobei N die Gesamtzahl der Gitterpunkte ist.
Da man immer nur den Punkt mit dem kleinsten Abstand hinzufügt, liefert dieser Algorithmus ein gute Approximation, weil dieser Abstand ja nicht von den anderen (noch nicht aktualisierten) Punkten abhängt, die weiter weg liegen.

Zur Aktualisierung der Abstandswerte:



Sei $\phi_{i,j,k}$ der Funktionswert, der aktualisiert werden soll. Dieser Wert soll so klein wie möglich sein. Dazu sucht man in jedem Quadranten eine charakteristische Richtung $\vec{\theta} = (\theta_1, \theta_2, \theta_3)$ mit $\theta_s > 0$ $(s = 1, 2, 3)$ und $\sum_s \theta_s = 1$, die den minimalen Wert liefert. Dann vergleicht man die Werte aus allen Quadranten und wählt den kleinsten davon als Wert für $\phi_{i,j,k}$. In jedem Quadranten wird die Gleichung

$$\phi_{i,j,k} = \tau(\vec{\theta}) + \theta_1\phi_1 + \theta_2\phi_2 + \theta_3\phi_3 \tag{6}$$

über alle Richtungen $\vec{\theta}$ minimiert, wobei

$$\tau(\vec{\theta}) = \sqrt{(\theta_1\Delta x)^2 + (\theta_2\Delta y)^2 + (\theta_3\Delta z)^2} \tag{7}$$

die Distanz zum Startpunkt $\sum \theta_s\phi_s$ ist. Das ergibt acht Quadranten mit Startpunkten $\phi_1 = \phi_{i\pm1,j,k}$, $\phi_2 = \phi_{i,j\pm1,k}$ und $\phi_3 = \phi_{i,j,k\pm1}$. Liegt einer dieser Punkte nicht in *akzeptiert*, wird er ignoriert, indem man $\theta_s = 0$ in dieser Richtung setzt. Die Minimierung von (6) kann man nach ein wenig Rechnung zurückführen auf das Lösen der quadratischen Gleichung

$$\left(\frac{\phi_{i,j,k} - \phi_1}{\Delta x}\right)^2 + \left(\frac{\phi_{i,j,k} - \phi_2}{\Delta y}\right)^2 + \left(\frac{\phi_{i,j,k} - \phi_3}{\Delta z}\right)^2 = 1 \tag{8}$$

welche eine Approximation erster Ordnung von $|\nabla\phi|^2 = 1$, also dem Quadrat von $|\nabla\phi| = 1$ ist. Gleichung (8) kann eventuell mehr als eine Lösung besitzen. Für die genaue Handhabung sei auf das Buch verwiesen.

Zur Initialisierung des Startbereichs:
Die einfachste Methode ist, ϕ unabhängig in jede Koordinatenrichtung zu betrachten und mittels linearer Interpolation bei einem Wechsel des Vorzeichens einen Wert festzulegen

und dann über alle Koordinatenrichtungen zu minimieren.
Diese Initialisierung ist genau wie die Aktualisierung der Werte ein Verfahren erster Ordnung. Höhere Ordnungen können erreicht werden, indem in Gleichung (8) der Gradient besser approximiert wird und die Startwerte mit besserer Interpolation initialisiert werden.

3 Zusammenfassung

Es wurden einige Techniken aufgezeigt, wie man ϕ zu einer Signed Distance Function aktualisieren kann. Die Fast Marching Method ist ein sehr schneller Algorithmus, aber nur erster Ordnung. Die Reinitialisierungsgleichung kann verwendet werden, um mit entsprechendem Löser höhere Ordnung zu bekommen. Leider kann das sehr zeitaufwändig sein oder nur schlecht funktionieren, vor allem wenn ϕ zu Beginn sehr weit entfernt von einer SDF ist. Daher ist die Fast Marching Methode häufig die bessere Wahl. Höhere Ordnungen können auch hier erreicht werden, wenn man die Diskretisierung des Gradienten entsprechend verbessert und die Anfangswerte besser als linear interpoliert.