Visualisierung zeitabhängiger Vektorfelder durch Flächenstrukturen

Tobias G. Pfeiffer

Berlin, 31. Dezember 2008

Inhaltsverzeichnis

1 Einführung

Das Ziel dieses Dokuments ist eine Einführung in zwei verschiedene aber verwandte Methoden, zeitabhängige Vektorfelder im dreidimensionalen Raum mit Hilfe von Oberflächen zu visualisieren. Ein zeitabhängiges Vektorfeld ist eine Abbildung $v : \Omega \times [T_0, T_1] \to \mathbb{R}^3$, wobei $\Omega \subset \mathbb{R}^3$ ein räumlicher Definitionsbereich und $[T_0, T_1]$ ein Zeitintervall ist. Es wird also jedem Punkt im Raum zu jeder Zeit eine gewisse „Richtung“ zugeordnet.

Natürlich kann man zeitabhängige Vektorfelder rein analytisch definieren, in der Realität treten sie aber insbesondere im Bereich der Analyse und Simulation von Flüssen und Strömungen auf, wie beispielsweise als Luftströme in einem Windkanal bei der Untersuchung von Formen im Fahrzeugbau. Man unterscheidet dabei zwischen Vektorfeldern mit Daten, die durch bestimmte Messungen gewonnen wurden (z. B. Windgeschwindigkeiten an gewissen Orten), und solchen, die das Ergebnis einer Simulation / Berechnung sind. Solche Berechnungen, hier ist insbesondere der Bereich der *Computational Fluid Dynamics (CFD)* zu nennen, haben oft zum Ziel, teure und aufwändige reale Versuche zu ersetzen.

Die Visualisierung von (nicht nur zeitabhängigen) Vektorfeldern ist seit längerem Gegenstand der Forschung. Sie ist notwendig, um das Verhalten von Strömungen

besser verstehen zu können, und Stellen zu finden, an denen charakteristisches Verhalten auftritt, z. B. Verwirbelungen. Die Ergebnisse können dann, beispielsweise in der Konstruktion, dazu dienen, Formen so zu optimieren, dass unerwünschtes Verhalten verschwindet.

Visualisierung von Vektorfeldern ist insbesondere deswegen kompliziert, da der Zielraum dreidimensional ist, und man keinen regulären Funktionsgraphen im $\mathbb{R}^4$ aufzeichnen oder die Funktionswerte als Farbwerte darstellen kann. Bei der Visualisierung zeitabhängiger Vektorfelder mit Hilfe von statischen Bildern (im Gegensatz zu Animationen) kommt dazu, dass die Zeit als ausgezeichnete Dimension besonders behandelt werden muss, um aussagekräftige Bilder zu erhalten.

Ein möglicher Weg, mit diesen Problemen umzugehen, ist die Verwendung von *Bahnlinien* (engl. *pathlines* oder *integral curves*), d. h. die Berechnung der Bahn eines (masselosen) Partikels, den man an einer bestimmten Stelle im Vektorfeld „aussetzt", siehe z. B. Abb. 1. Formal ist die Bahnlinie zu einem Punkt $(x_0, t_0) \in \Omega \times [T_0, T_1]$ eine Kurve $\gamma : [t_0, T_1] \to \mathbb{R}^3$ mit $\gamma(t_0) = x_0$ und $\gamma'(t) = v(\gamma(t), t)$ (d. h. sie ist zu jedem Zeitpunkt tangential zum Vektorfeld).

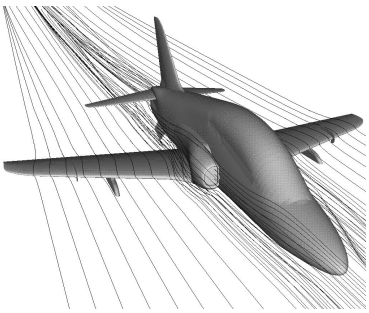


Abbildung 1: *Bahnlinien in einem Vektorfeld [Fin]*

Ein anderer Begriff aus der Strömungsmechanik, der zur Visualisierung oft verwendet wird, ist der der *Streichlinie* (engl. *streakline*); dies ist eine Kurve, die die Bewegung mehrerer an der gleichen Stelle, aber zu unterschiedlichen Zeitpunkten ins Vektorfeld ausgesetzten Punkten beschreibt. Anschaulich kann man dies beschreiben als die Linie, die man zum Zeitpunkt t_1 sieht, wenn man zum Zeitpunkt $t_0 \leq t_1$ an der Stelle x_0 beginnt, mit einer Schnur verbundene Bojen in einen Fluss zu setzen.

Diese beiden Linientypen fallen für zeitunabhängige Vektorfelder zusammen und sind dann identisch mit den bekannten *Stromlinien* (engl. *streamlines*). Die Idee der beiden im Folgenden vorgestellten Verfahren ist es, nicht nur die Bahn- oder Streichlinien einzelner isolierter Punkte zu verwenden, sondern die Punkte einer gegebenen Ausgangskurve in Ω zu betrachten und die von den entsprechenden Bahn- bzw. Streichlinien aufgespannte Fläche zu visualisieren.

2 Stream Surfaces

Mathematisch betrachtet ist eine Stream Surface (auf den Neologismus der „Bahnfläche“ sei hier verzichtet) eine parametrisierte Fläche

$$F : [0, 1] \times [T_0, T_1] \to \mathbb{R}^3, \qquad (s, t) \mapsto F(s, t),$$

wobei $c_0 : s \mapsto F(s, T_0)$ die Ausgangskurve in Ω zum Zeitpunkt T_0 ist. F beschreibt dann die Bewegung aller Punkte von c_0 im Vektorfeld, d. h. F ist die Vereinigung aller Bahnlinien $\{\gamma_s \mid \gamma_s(T_0) = c_0(s), s \in [0, 1]\}$; also ist $F(s, t) = \gamma_s(t)$.[1] Umgekehrt kann man F jedoch auch als die Vereinigung aller *Zeitlinien* (engl. *timelines*) $c_t : [0, 1] \to \mathbb{R}^3$, die die Position zum Zeitpunkt t der im Vektorfeld bewegten Kurve c_0 beschreiben, betrachten. In Abb. 2 ist eine solche Stream Surface dargestellt und skizziert, wie man sie als Vereinigung von Bahnlinien bzw. Zeitlinien verstehen kann.

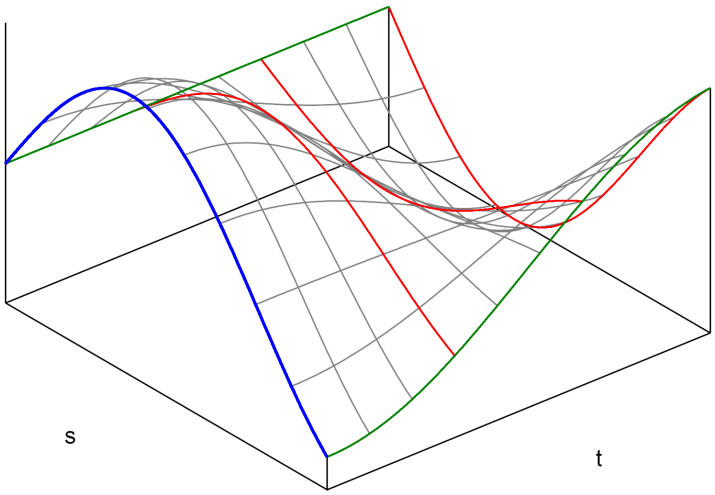


Abbildung 2: *Ausgangskurve (blau), Bahn- (grün) und Zeitlinien (rot)*

Zur Visualisierung solcher Flächen muss an vielen Stellen Einschränkungen bei der Arbeit mit dem Computer beachten:

- Das Vektorfeld, auf dem man arbeitet, ist im Allgemeinen nur auf gewissen Knotenpunkten definiert, dazwischen muss interpoliert werden. Eine überzeugend aussehende Visualisierung darf nicht darüber hinwegtäuschen, dass sie nicht näher an der Wirklichkeit sein kann, als die Gitterweite der Vektorfelddaten ermöglicht.

- Bahnlinien können in der Regel nicht exakt berechnet werden, die Funktions-

[1] Im Folgenden sei γ_{x_0,t_0} die Bahnlinie zum Punkt (x_0, t_0) und γ_s, im Kontext einer konkreten Ausgangskurve c_0, die Bahnlinie zum Punkt $(c_0(s), T_0)$.

werte von γ_s müssen immer über ein numerisches Verfahren bestimmt werden und liegen dann nur als Werte an gewissen Stützstellen t_j vor. Außerdem können nur endlich viele Bahnlinien berechnet werden, d. h. es kann auch immer nur eine Approximation der Stream Surface erstellt werden.

- Zum hardwareunterstützten Rendern einer Stream Surface (z. B. mit OpenGL) muss die Fläche durch ein polygonales Gitter approximiert werden, wobei die Schwierigkeit auftritt, dass in Bereichen, in denen die Fläche stark gekrümmt ist oder anderes „besonderes“ Verhalten zeigt, ein möglichst feines Gitter gewünscht ist, während in weniger interessanten Gebieten zur Reduzierung von Speicher und Renderzeit ein gröberes Gitter verwendet werden sollte.

2.1 Frühere Verfahren

Ein ganz natürliches und intuitives Verfahren, eine Stream Surface zu erzeugen, gibt der folgende Algorithmus:

- Zu fest gewählten Punkten $s_i \in [0, 1]$ Bahnlinien γ_{s_i} berechnen.
- Geeignete $t_j \in [T_0, T_1]$ wählen und die Punktmenge $\{x_{i,j} \mid x_{i,j} = \gamma_{s_i}(t_j)\}$ unter Verwendung der Nachbarschaftsinformationen der Bahnlinien triangulieren.

Eine Möglichkeit der adaptiven Verfeinerung ist dabei, an Stellen, an denen sich benachbarte Bahnlinien zu den Punkten s_i, s_{i+1} weit voneinander entfernen, einen weiteren Punkt $\frac{s_i+s_{i+1}}{2}$ einzufügen, dazu die Bahnlinie zu berechnen und in die Triangulierung mit einzubeziehen. Diese Herangehensweise bringt jedoch einige Probleme mit sich. Um an einer Stelle t_0, die sich fern von T_0 befindet, eine Verfeinerung zwischen zwei Bahnlinien durchzuführen, muss die entsprechende neue Bahnlinie komplett von T_0 an durchgerechnet werden, obwohl für Zeiten vor t_0 diese Rechenarbeit nicht notwendig gewesen wäre. Außerdem erhöht diese Einführung von neuen globalen Bahnlinien auch in den „uninteressanten“ Gebieten die Anzahl der Polygone, was sich negativ auf die Renderzeit auswirkt. Des Weiteren ist nicht sofort ersichtlich, wie man mit Unstetigkeiten der Stream Surface (die z. B. bei umströmten Objekten häufig auftreten) umgehen soll.

Um mit diesen Problemen umzugehen und die Geschwindigkeit des Verfahrens zu erhöhen, beschreibt Hultquist in [Hul92] die Berechnung einer (polygonalen) Stream Surface mittels einer *Advancing Front*. Die Idee dabei ist, das Stück zwischen zwei benachbarten Anfangsstücken von Bahnlinien, ein sog. *Band* (engl. *ribbon*) mit einer existierenden Triangulierung zu betrachten, die beiden Bahnlinien jeweils einen Zeitschritt weiter zu berechnen, eine der beiden möglichen Kanten, die sich nun für ein weiteres Dreieck ergeben, hinzuzunehmen (siehe Abb. 3), ggf. die Front der Kanten durch rekursive Anwendung auf benachbarte Bänder wieder stetig zu machen, und beim nächsten Dreieck des ursprünglichen Bandes fortzufahren.

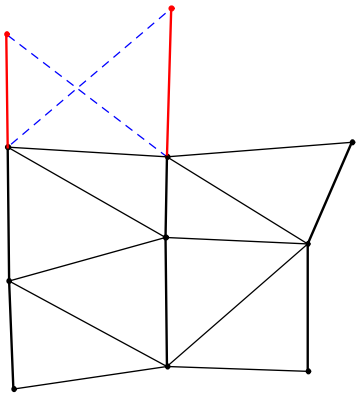


Abbildung 3: *Advancing-Front-Verfahren nach Hultquist: Existierende Triangulierung zweier Bänder (schwarz), zwei Bahnlinien werden einen Zeitschritt weiter berechnet (rot), eine Diagonale (neue Frontkante) wird ausgewählt (blau).*

Adaptive Gitterweite wird realisiert, indem der Abstand der beiden neu berechneten Punkte auf den beiden benachbarten Bahnlinien betrachtet wird und ggf. ein Punkt in der Mitte eingefügt wird, der dann als eigene Bahnlinie fortgesetzt wird. Analog können zwei benachbarte Bänder verschmolzen werden, indem die mittlere Bahnlinie beendet und die Triangulierung durch einen geeigneten Abschluss in einem gemeinsamen Band fortgesetzt wird, vgl. Abb. 4. Unstetigkeiten werden behandelt, indem ein Band aufgelöst wird, wenn die benachbarten Bahnlinien in sehr verschiedene Richtungen gehen, und die Bereiche links und rechts von diesem Band separat voneinander weiter behandelt werden.

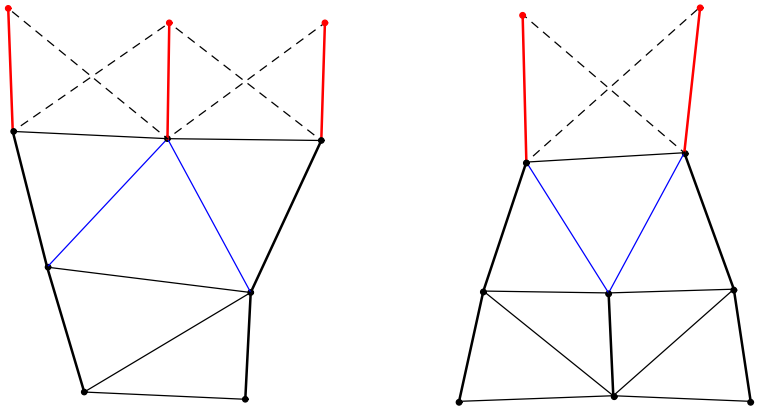


Abbildung 4: *Adaptive Gitterweite beim Advancing-Front-Verfahren: Einfügen einer neuen Bahnlinie bzw. Entfernen einer überflüssigen*

Ein Problem bei diesem Verfahren ist, dass durch die Kombination von Iteration und Rekursion in Hultquists Algorithmus (vgl. Pseudocode in [Hul92]) weitere /

andere Verfeinerungsstrategien (die nicht nur auf dem Abstand zwischen zwei benachbarten Bahnlinien basieren, sondern z. B. auch Krümmung in Betracht ziehen wollen, für deren Berechnung man mehr als zwei Punkte benötigt) schwierig zu implementieren und damit auch fehleranfällig sind.

Ein anderes, tiefliegendes Problem ist die Schwierigkeit, zur Berechnung der Bahnlinien Verfahren mit adaptiver Schrittweitensteuerung zu verwenden, so dass eine Triangulierung zwar in Richtung der Zeitlinien je nach Beschaffenheit des Vektorfeldes verfeinert werden kann, nicht aber in Richtung der Bahnlinien, wodurch die Adaptivität wieder eingeschränkt wird.

2.2 Verfahren von Garth, Krishnan, Tricoche, Bobach und Joy

In [GKT+08] wird als grundlegende Ursache vieler Probleme anderer Verfahren die fehlende Trennung zwischen der Berechnung einer guten Approximation einer Stream Surface einerseits und einer geeigneten polygonalen Repräsentation dieser Approximation andererseits genannt. Die Autoren geben ein Verfahren an, mit dem diese Trennung hergestellt wird und viele der genannten Probleme behandelt werden können.

Die grundlegende Idee ist, zu gewählten Zeitpunkten $t_j \in [T_0, T_1]$ iterativ die Zeitlinien der Stream Surface zu berechnen, wobei deren Genauigkeit bzw. Auflösung in „interessanten“ Bereichen erhöht wird, vgl. Abb. 5. Anschließend wird aus diesen Zeitlinien und den partiellen Bahnlinien, die bei der Berechnung entstehen, ein Gitter generiert, das dann zum Rendern verwendet wird.

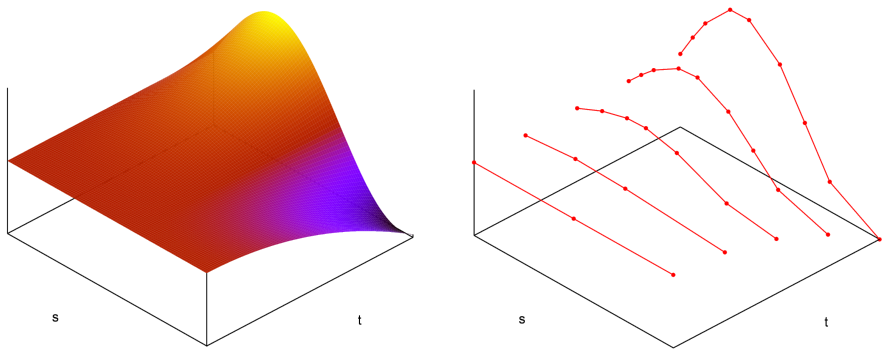


Abbildung 5: *Verfeinerung der stückweise linearen Zeitlinien in Bereichen hoher Krümmung*

2.2.1 Verfeinerung von Kurven

Zunächst wird ein allgemeines Verfahren vorgestellt, um zu einer stückweise glatten Kurve mit höchstens endlich vielen Unstetigkeiten durch Verfeinerung der Auswahl der Stützstellen s_i eine stückweise lineare Interpolierende Lf zu erhalten, die der

ursprünglichen Kurve „ähnlich genug“ ist. Dazu seien zwei Indikatoren Q_{discont} und Q_{refine} gegeben, die zu einem Tripel von benachbarten Punkten (s_{i-1}, f_{i-1}), (s_i, f_i), (s_{i+1}, f_{i+1}) einer stückweise glatten Kurve f angeben, ob die Kurve dort (wahrscheinlich) unstetig ist oder dort zwecks Erhöhung der Genauigkeit verfeinert werden sollte. Nun kann man eine gegebene stückweise lineare Kurve, definiert durch eine Menge von Stützstellen (s_i) mit entsprechenden Funktionswerten (f_i), wie folgt verfeinern:

- Wenn für ein Tripel benachbarter Punkte (s_{i-1}, f_{i-1}), (s_i, f_i), (s_{i+1}, f_{i+1}) der Indikator Q_{discont} wahr ist, trenne die Kurve zwischen s_i und s_{i+1} auf und behandle beide Teile rekursiv.

- Wenn für ein Tripel benachbarter Punkte (s_{i-1}, f_{i-1}), (s_i, f_i), (s_{i+1}, f_{i+1}) der Indikator Q_{refine} wahr ist, füge $\frac{s_i+s_{i+1}}{2}$ zur Menge der Stützstellen hinzu, berechne $f(\frac{s_i+s_{i+1}}{2})$ und füge dies der Menge der Funktionswerte hinzu. (Man beachte, dass dies nur geht, wenn man f überhaupt an beliebigen Stellen auswerten kann.)

- Der Algorithmus endet, wenn Q_{refine} und Q_{discont} für alle benachbarten Punkttripel jedes zusammenhängenden Kurvenstücks falsch sind.

So erhält man eine stückweise lineare Kurve, die – in Bezug auf die gegebenen Indikatoren – eine „gute“ Approximation der echten Kurve f ist. Dabei ist die Wahl der Indikatoren offenbar essentiell; wenn diese schlecht gewählt sind, muss der Algorithmus nicht einmal notwendigerweise terminieren. Eine gute Wahl für Eigenschaften, die in Q_{refine} einfließen könnten, um Konvergenz gegen f zu gewährleisten, sind Abstand zwischen benachbarten f_i, zweite Differenzen (diskretes Analogen zur zweiten Ableitung), Winkel zwischen den drei gegebenen Punkten o. ä. Durch die einfache Austauschbarkeit dieses Verfeinerungsindikators hat man hier bereits eine viel größere Flexibilität als im Algorithmus von Hultquist.

2.2.2 Iterative Berechnung der Zeitlinien

Es sollen nun zu gegebenen Zeitpunkten (t_j) Approximationen der Zeitlinien der vorgegebenen Ausgangskurve berechnet werden. Sei dazu $c_j : \Omega \to \mathbb{R}^k$ eine stückweise glatte Kurve, deren stückweise lineare Interpolierende Lc_j durch die Stützstellen $s_{j,i}$ und die Funktionswerte $c_{j,i}$ gegeben ist (dabei ist c_0 die Ausgangskurve mit einer vergleichsweise groben Wahl von Stützstellen). Dann definiere c_{j+1} durch $c_{j+1}(s) := \gamma_{Lc_j(s),t_j}(t_{j+1})$, d. h. der Punkt $Lc_j(s)$ auf der linearen Interpolierenden von c_j wird auf der entsprechenden Bahnlinie einen weiteren Zeitschritt (von t_j zu t_{j+1}) durch das Vektorfeld transportiert. Beachte, dass dann, obwohl nur eine stückweise lineare Funktion transportiert wird, c_{j+1} selbst i. A. nicht stückweise linear ist!

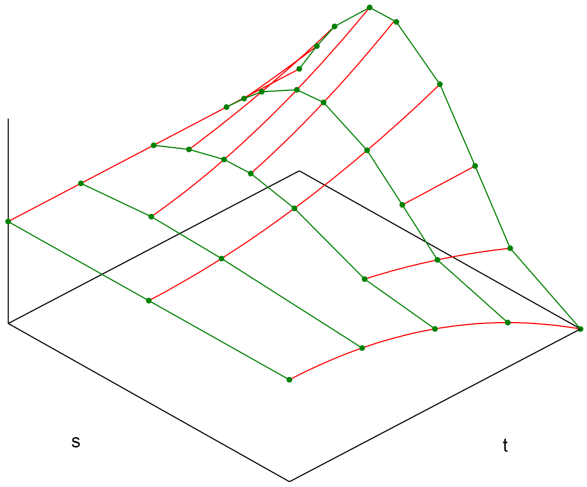


Abbildung 6: *Stützpunkte der Zeitlinien (grün) erhält man durch Fortführen der Stützpunkte der vorherigen Zeitlinien und ggf. zusätzlicher Punkte entlang von Bahnlinien (rot).*

Für jede Auswertung von c_{j+1} an einer Stelle s muss ein Schritt mit einem numerischen DGL-Löser berechnet werden; um die Kurve abspeichern zu können, wird deswegen wieder eine stückweise lineare Interpolierende verwendet. Wähle dazu zunächst alle Stützstellen, die auch schon für c_j verwendet wurden ($s_{j+1,i} := s_{j,i}$), berechne die $c_{j+1,i}$ durch Transportieren entlang von Bahnlinien und füge dann, falls notwendig, mit dem in 2.2.1 beschriebenen Algorithmus Stützstellen hinzu. Dies bedeutet, dass wenn für drei benachbarte Stützstellen $s_{j+1,i-1}$, $s_{j+1,i}$, $s_{j+1,i+1}$ der Indikator Q_{refine} wahr ist, eine Stützstelle bei $\frac{1}{2}(s_{j+1,i} + s_{j+1,i+1})$ eingefügt und der entsprechende Funktionswert durch Transportieren von $Lc_j(\frac{1}{2}(s_{j+1,i} + s_{j+1,i+1}))$ entlang einer Bahnlinie berechnet wird. Bemerke, dass nun die Stützstellen von c_{j+1} eine Obermenge der Stützstellen von c_j sind.

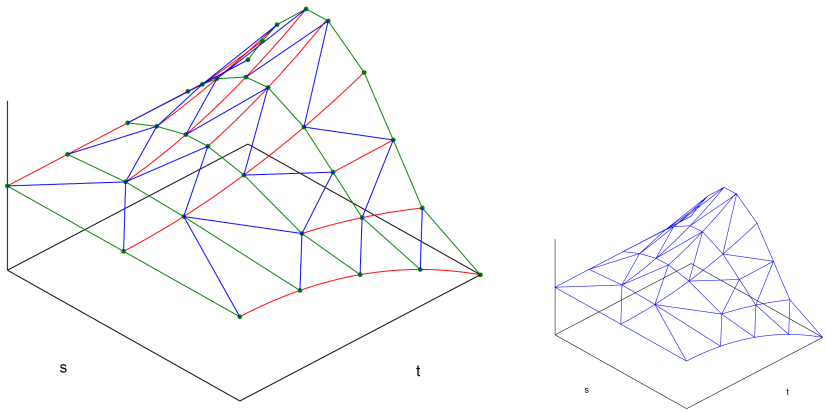
Für jede Stützstelle $s_{j+1,i} \in [0, 1]$ ist nun ein Schritt mit einem numerischen Verfahren durchgeführt worden, der als Nebenprodukt einen Teil einer Bahnlinie liefert, vgl. Abb. 6. Diese Bahnlinien werden ebenfalls abgespeichert, jedoch nicht als diskrete Menge von Punkten, sondern als Folge von Polynomen, wobei bei Runge-Kutta-Verfahren diese Polynome direkt aus den Zwischenschritten konstruiert werden können und das in [GKT+08] verwendete Dopri5-Integrationsverfahren bereits Polynome ausgibt.

Eine Vergröberung der Zeitlinien, d. h. eine Reduzierung der Anzahl der Stützstellen, findet bei Garth *et al.*, im Gegensatz zum Verfahren von Hultquist, nicht statt, was damit begründet wird, dass die Komplexität „interessanter“ Vektorfelder häufig nicht wieder abnimmt, wenn einmal eine starke Deformation stattgefunden hat, und es schwierig wäre, geeignete Grenzwerte zu finden, ab denen eine Kurve wieder vergröbert werden kann, ohne evtl. Details zu verlieren.

2.2.3 Konstruktion der Triangulierung

Nach Abschluss der vorherigen Phase hat man zu jedem der vorher ausgewählten Zeitpunkte $(t_j)_j$ Stützpunkte $(s_{j,i}, c_{j,i})_i$ berechnet, die stückweise lineare Approximationen der Kurven c_j (die wiederum Approximationen der Zeitlinien von c_0 sind) repräsentieren, sowie stückweise polynomielle Bahnlinien, die bei einem gewissen t_j beginnen und dann bis T_1 verlaufen. Diese stückweise linearen Zeit- und stückweise polynomiellen Bahnlinien (siehe Abb. 6) bilden das Gerüst der Triangulierung. Um diese konstruieren zu können, müssen jedoch zunächst noch Punkte auf den Bahnlinien bestimmt werden, die in die Triangulierung einbezogen werden sollen. Dabei werden die Indikatoren aus 2.2.1 verwendet, d. h. auf jeder Bahnlinie γ wird zunächst ein grobes Gitter aus Stützstellen ausgewählt (nämlich alle t_j, die auf der Kurve liegen) und dann mehr Stützstellen hinzugefügt, wo dies notwendig ist; die entsprechenden Funktionswerte kann man durch Auswertung der Polynome erhalten.

Nun werden, beginnend bei T_0, nacheinander die Bänder zwischen je zwei Bahnlinien trianguliert, indem eine der beiden Diagonalen zwischen den aktuellen und den jeweils nächsten Punkten auf den Bahnlinien ausgewählt wird. Kreuzt man dabei eine Zeitlinie, muss geprüft werden, ob an der aktuellen Stelle t_j eine neue Bahnlinie in der Mitte dieses Bandes beginnt; ist dies der Fall, so werden beide nacheinander rekursiv mit der selben Methode behandelt. So erhält man schließlich eine Triangulierung, die direkt mit einer Textur versehen und gerendert werden kann.



(a) Stückweise lineare Zeitlinien (grün), stückweise polynomielle Bahnlinien (rot), eingefügte Dreieckskanten (blau), hier ohne weitere eingefügte Stützstellen auf den Bahnlinien.

(b) Finale Triangulierung

Abbildung 7: *Triangulierung des erhaltenen Gitters aus Zeit- und Bahnlinien*

3 Streak Surfaces

Streak Surfaces, also von Streichlinien aufgespannte Flächen, haben eine ähnliche Funktion wie Stream Surfaces, nämlich die Visualisierung eines (zeitabhängigen) Vektorfeldes durch eine Fläche, die die Bewegung von Teilchen von einer Ausgangskurve ausgehend beschreibt; sie erreichen dies jedoch durch eine andere Methode und haben damit auch andere Eigenschaften. In [vFWTS08] verwenden von Funck *et al.* Streak Surfaces, um die Bewegung von Rauch in einem Vektorfeld in Echtzeit zu simulieren. Diese sind dafür konzeptionell die richtige Wahl, denn während Stream Surfaces zwar die Bahn einer Menge von Teilchen visualisiert, die zu einem festen Zeitpunkt von einem festen Ort in den Fluss ausgesetzt wurden, treten sie in der Realität zu keinem Zeitpunkt auf, denn ein Teilchen befindet sich zu einem bestimmten Zeitpunkt immer an einem Ort (jedenfalls in der klassischen Physik) und bildet daher keine Linie. Streichlinien hingegen zeigen tatsächlich eine Momentaufnahme der Position einer ganzen Menge von Teilchen, die alle zu einem Zeitpunkt in der Vergangenheit an einem festen Ort waren, deswegen sind Streak Surfaces als Vereinigung von Streichlinien für die Betrachtung von Rauchbewegung geeignet.

Zur Generierung von Streak Surfaces wird eine Menge von Partikeln verwendet, die in jedem Zeitschritt entlang des Vektorfeldes bewegt und dann mit einer geeigneten Triangulierung versehen werden. Damit diese Fläche nicht irgendwann vollständig verschwindet, werden ihr außerdem in jedem Zeitschritt von der Ausgangskurve aus neue Partikel hinzugefügt. Diese Berechnung *aller* Partikelbewegungen und das anschließende *Remeshing* verhinderten es bisher, Streak Surfaces in Echtzeitanwendungen einzusetzen.

3.1 Echtzeitberechnung von Streak Surfaces

Von Funck *et al.* umgehen das Problem des Remeshing, indem sie ein Dreiecksnetz mit fester Konnektivität verwenden und durch Wahl geeigneter Transparenzwerte den Eindruck von Rauch mit verschiedener Dichte an verschiedenen Stellen simulieren.

Zunächst wird ein trianguliertes Gitter aus $(m + 1) \times (n + 1)$ Vertices $x_{i,j}$ mit Zylindertopologie erzeugt, d. h. es gibt (von einem stehenden Zylinder ausgehend) $n + 1$ Zeilen und m Spalten von Vertices (denn $x_{0,j} = x_{m,j}$). Die Ausgangskurve sei als Polygon mit $n + 1$ Vertices gegeben. Zu Beginn werden alle Vertices auf der Ausgangskurve positioniert und dann wird in jedem Zeitschritt t_k eine weitere Spalte (mit $n + 1$ Vertices) in die Berechnung der Laufbahn miteinbezogen, d. h. ihre Bewegung innerhalb des Vektorfeldes wird durch ein numerisches Verfahren berechnet und die Position dementsprechend angepasst. Zum Zeitpunkt t_m befinden

sich dann alle Spalten im Fluss und wurden einmal bewegt, ab diesem Punkt wird dann in jedem Zeitschritt zyklisch die letzte Reihe wieder auf die Ausgangskurve zurückgesetzt und von dort aus im Fluss bewegt. Der folgende Pseudocode beschreibt das Verfahren:

```
# Initialisierung: platziere Vertices auf Polygon
for i = 0 to n:
  for j = 0 to m-1:
    x(i, j) = s(i, T0)
t = T0
for k = 0 to NUM_TIMESTEPS:
  # falls alle Spalten im Fluss, letzte Spalte auf Ausgangskurve zurueck:
  if k >= m-1:
    for i = 0 to n:
      x(i, k mod (m-1)) = s(i, t)
  # iteriere ueber Spalten im Fluss:
  for j = 0 to min(k, m-1):
    for i = 0 to n:
      # bewege Vertex im Fluss einen Zeitschritt weiter:
      x(i, j) = move(x(i, j), t, Delta_t)
  t = t + Delta_t   # naechster Zeitschritt
```

Hierbei ergibt sich der Vorteil, dass während der Laufzeit kein neuer Speicher alloziert werden muss, da der Speicherplatz für die Vertices konstant ist und vor Ablauf des Algorithmus reserviert werden kann. Außerdem ist das Verfahren sehr gut parallelisierbar, da die Bewegung der einzelnen Vertices voneinander unabhängig sind. Ein Problem, das aus der festen Konnektivität resultiert, ist, dass im Fluss divergierende, benachbarte Vertices sehr große Dreiecke aufspannen, die zunächst unnatürlich wirken und dann in bestimmten Bereichen keine aussagekräftige Streak Surface mehr erzeugen würden. Das Ziel dieses Verfahrens ist jedoch in erster Linie die Simulation von Rauch und an solchen Stellen, bei denen benachbarte Partikel sehr stark divergieren, hat Rauch in der Regel eine geringe Dichte, ist also weniger sichtbar; durch einen höheren Transparenzwert für solche Dreiecke kann das Problem der Darstellung also teilweise behoben werden.

3.2 Bestimmung der Transparenzwerte

Von Funck *et al.* bestimmen nach der Bewegung aller Teilchen für jedes der $2 \cdot m \cdot (n+1)$ Dreiecke einen Lichtundurchlässigkeitsgrad α (d. h. der Transparenzgrad ist $1 - \alpha$), der zu großen Teilen durch die Rauchdichte α_{density} innerhalb eines flachen Prismas über dem Dreieck, abhängig von Flächeninhalt des Dreiecks und Blickwinkel auf das Dreieck, bestimmt wird. In gewissen Situationen – insbesondere

beim Umfließen von Hindernissen im Fluss oder bei den Dreiecken, die die aktuell erste und letzte Vertexreihe des Zylinders verbinden – kann es jedoch vorkommen, dass Flächeninhalt und Blickwinkel nicht sehr stark variieren, das Dreieck jedoch trotzdem stark verzerrt ist und daher einen hohen Transparenzwert erhalten sollte, vgl. Abb. 8.

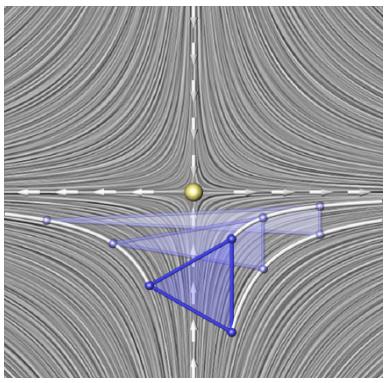


Abbildung 8: *Ein im Fluss transportiertes und dabei stark verzerrtes Dreieck einer Streak Surface mit annähernd konstantem Flächeninhalt. Aus [vFWTS08].*

Es werden nun diverse Parameter verwendet, die ein Dreieck in bestimmten Situationen weniger sichtbar machen; im Einzelnen sind dies:

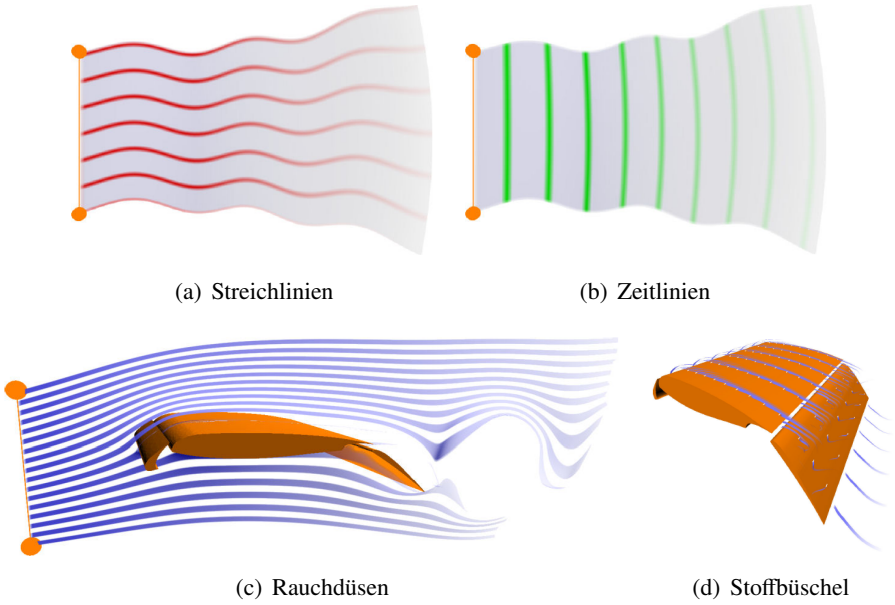
- α_{shape}, das vom Verhältnis zwischen kürzester Seite und Umkreisradius des Dreiecks abhängt und damit die „Qualität“ des Dreiecks im Sinne der Verzerrungsfreiheit angibt; für ein gleichseitiges Dreieck ist $\alpha_{\text{shape}} = 1$.
- $\alpha_{\text{curvature}}$, das ein gewisses diskretes Krümmungsmaß enthält und damit Dreiecke in Gebieten hoher Krümmung (in denen die grobe Auflösung beim Rendern Artefakte erzeugen würde) weniger sichtbar macht.
- α_{fade}, das von der Zeitdauer, die ein Dreieck sich schon im Fluss befindet, abhängt und damit das Verblassen von Rauch im Laufe der Zeit simulieren soll.

α_{density} und $\alpha_{\text{curvature}}$ müssen dabei ggf. in das Intervall $[0, 1]$ eingepasst werden. Der Gesamtwert α für den Grad der Sichtbarkeit ergibt sich dann durch Multiplikation der vier genannten Parameter.

3.3 Erweiterungen

Das vorgestellte Verfahren kann nicht nur zur Berechnung von Streak Surfaces verwendet werden, sondern durch kleine Modifikationen auch für andere Anwendungen dienen. Beispielsweise kann man, statt eines Polygons nur einen einzigen

Punkt als Ausgangsstruktur verwenden und erhält dann Streichlinien. In einer normalen Streak Surface kann man diese Streichlinien nur durch Änderung der Textur hervorheben, indem man die Vertices einer *Zeile* des Zylinders besonders einfärbt (Abb. 9(a)). Zeitlinien hingegen kann man erhalten, wenn man die Vertices einer *Spalte* des Zylinders besonders einfärbt (Abb. 9(b)). Analog kann man auch eine *Time Surface* erhalten, wenn man die Zylindertopologie auflöst, das gesamte Gitter zu einem Zeitpunkt in den Fluss gibt und transportieren lässt.



(a) Streichlinien (b) Zeitlinien

(c) Rauchdüsen (d) Stoffbüschel

Abbildung 9: *Diverse Erweiterungen des Verfahrens aus [vFWTS08]*

Um eine Reihe von Rauchdüsen zu simulieren, wie sie in Windkanälen verwendet werden, kann man jede zweite Zeile von Dreiecken auf vollständig transparent setzen (Abb. 9(c)). In Windkanälen werden außerdem häufig kleine Stücke aus Stoff an der Oberfläche eines Körpers befestigt, um Stellen zu finden, an denen sich der Luftstrom von der Oberfläche entfernt. Solche Experimente können simuliert werden, indem an den Stellen, an denen man Stoff anbringen würde, kurze Ausgangspolygone platziert werden, deren Streak Surfaces dann gerendert werden (Abb. 9(d)).

4 Vergleich

Die vorgestellten Verfahren haben, obwohl sie beide die Visualisierung zeitabhängiger Vektorfelder durch Flächenstrukturen zum Ziel haben, sehr unterschiedliche Ansprüche, Prioritäten und damit auch andere Anwendungsgebiete.

Das Verfahren aus [vFWTS08] hat im Wesentlichen den Anspruch, Rauchvisualisierung in Echtzeit zu liefern und erreicht dieses nach Angabe der Autoren auch

noch bei mehreren Zehntausend Vertices in der Streak Surface. Allerdings geben sie nicht an, mit welchem numerischen Integrationsverfahren sie ihren Algorithmus ausgestattet haben und rechnen nur auf strukturierten Gittern, bei denen die Interpolation des Vektorfeldes schnell durchführbar ist. Die Qualität der generierten Fläche wird in visueller Hinsicht durch die Einführung von teilweise heuristisch gewählten Parametern, die den Transparenzgrad der Dreiecke steuern, gewährleistet; es ist ansonsten jedoch nicht vorgesehen, in „interessanten“ Gebieten genauer zu rechnen oder die Anzahl der Vertices lokal zu erhöhen, um die Qualität auch in numerischer Hinsicht zu erhalten. Man kann jedoch davon ausgehen, dass in Anwendungen, die interaktive Geschwindigkeiten erfordern, oft auf dieses Maß an numerischer Qualität verzichtet werden kann, und es dort mehr auf den visuellen Eindruck ankommt.

Das Verfahren aus [GKT+08] hat zum Ziel, genaue Stream Surfaces zu berechnen, die nach vom Anwender festgelegten Kriterien lokal verfeinert werden können. Die Geschwindigkeit ist hier nicht der wesentliche Aspekt bei der Berechnung der triangulierten Fläche, so werden auch keine absoluten Angaben zur Rechenzeit gemacht, die Autoren geben jedoch die Interpolation des Vektorfeldes (im Falle eines unstrukturierten Gitters) als dominierenden Faktor an. Die Qualität der generierten Fläche wird durch adaptive Verfeinerung sichergestellt, allerdings haben die erzeugten Bilder weniger ästhetischen als mehr analytischen Wert, wenn es darauf ankommt, Verhalten an bestimmten Stellen des Vektorfeldes genau zu untersuchen.

Literatur

[Fin] Image Gallery of Finflo Ltd. http://www.finflo.fi/Hawk/pullup.html. Stand: 30.11.2008).

[GKT+08] Christoph Garth, Hari Krishnan, Xavier Tricoche, T. Bobach, and Kenneth I. Joy. Generation of accurate integral surfaces in time-dependent vector fields. *Proceedings of IEEE Visualization '08*, October 2008.

[Hul92] J. P. M. Hultquist. Constructing stream surfaces in steady 3d vector fields. In *VIS '92: Proceedings of the 3rd conference on Visualization '92*, pages 171–178, Los Alamitos, CA, USA, 1992. IEEE Computer Society Press.

[vFWTS08] Wolfram von Funck, Tino Weinkauf, Holger Theisel, and Hans-Peter Seidel. Smoke surfaces: An interactive flow visualization technique inspired by real-world flow experiments. *IEEE Transactions on Visualization and Computer Graphics (Proceedings Visualization 2008)*, 14(6), November - December 2008. accepted for publication.