

# 1 Visualization and Virtual Reality

## 1.1 Definition

Numerische Daten werden zum besseren Verständnis graphisch aufbereitet.

- Animation
- Geometrie
- Datenstrukturen
- menschl. Denken in 3D
- Realität → Computermodell → Computergrafik → Beobachter

### 1.1.1 Scientific VR

Daten aus Natur/Technik/Lebenswissensch. Numerisch, 1,2,3,4-Dimensional

### 1.1.2 Information Visualization

Unstrukturierte, nicht im Raum angeordnete Daten Visualisieren, Zugreifbar machen

### 1.1.3 Software Visualization

Struktur/Ablauf von Algorithmen

### 1.1.4 Beispiele

Strömung um das Flugzeug

- Raum  $X \subset \mathbb{R}^3$  um das Flugzeug
- $v : X \rightarrow \mathbb{R}^3$  Vektorfeld
- $p : X \rightarrow \mathbb{R}$  Druck

Verformung eines belasteten Bauteils

- Fläche  $F \subset \mathbb{R}^3$
- Vektorfeld der Verschiebung  $v : F \rightarrow \mathbb{R}^3$

## 1.2 Visualisierungspipeline

- **Filterung:** Auswahl der Daten, Fehlerkorrektur, Interpolation
- **Mapping:** Abbildung auf Geometrie, Textur, Materialattribute (Farbe, Reflexion etc)
- **Rendering:** Darstellung bzgl. Augpunkt und Lichtquellen, Shading, Raytracing, Volume-Rendering etc.

## 1.3 Virtual Reality

- Stereobilder auf Großbildschirmen (Powerwall, Shutter Glasses)
- Immersive Umgebungen (CAVE, Pos. der Augen: Tracking Gerät)
- Augmenten VR (Head Mounted Display)
- Haptische Geräte
- VR Workbench

## 2 Datenrepräsentation

Daten lassen sich meist als mathemat. Fkt.  $d \rightarrow F$  darstellen. Definitionsgebiet meist Gitter. Wertebereich Skalar, Vektor oder Tensor.

### 2.1 Tensoren

$a$  Skalar

$a_i$  Vektor

$a_{ij}$  Matrix

$a_{ijk}$  ...

**Ordnung** eines Tensor ist die Anzahl seiner Indizes

#### 2.1.1 Bsp (Kroneker, alternierender T.)

**Kroneker-Symbol:**  $\delta_{ij} = 1$ , falls  $i=j$ , 0 sonst. Oder:  $\delta_{ij} * a_j = a_i$   
**alternierender Tensor:**  $\epsilon_{ijk} = 1$ , falls  $ijk = 123, 231, 312$ ;  $-1$  falls  $ijk = 321, 213, 132$ ; 0 sonst

#### 2.1.2 Zerlegung in Symmetrischen und Antisymmetrischen Teil

$$T = S + A; S = 1/2(T_{ij} + T_{ji}) + 1/2(T_{ij} - T_{ji})$$

### 2.1.3 Kreuzprodukt

$$a \times b = (a_2b_3 - a_3b_2)e_1 + (a_3b_1 - a_1b_3)e_2 + (a_1b_2 - a_2b_1)e_3$$

oder:

$$(a \times b)_i = \epsilon_{ijk} a_j b_k (= \sum_j \sum_k \epsilon_{ijk} a_j b_k)$$

**zwei gleiche Indizes in Produkt = Summation!**

## 2.2 Gradient

**Nabla:**  $\nabla = e_i * \frac{\delta}{\delta x_i}$

Anwendung:

$$\nabla * a = (\delta a / \delta x_1, \delta a / \delta x_2, \delta a / \delta x_3)^T$$

Richtungsableitung in Richtung  $n$ :

$$\delta a / \delta n = n \circ \nabla a$$

## 2.3 Divergenz, Rotation

$$div(a) = \nabla \circ a = \delta a_1 / \delta x_1 + \delta a_2 / \delta x_2 + \dots$$

$$rot(a) = (\nabla \times a)_i = \epsilon_{ijk} \delta a_k / \delta x_j$$

## 2.4 Helmholtz-Hodge

Zerlegung des Vektorfeld in divergenz-freien Anteil und Gradient:  $v = g + \nabla f$ ;  $div(g) = 0$

Die Rotation der Ableitung ist 0, da Reihenfolge der Ableitung egal und  $\epsilon_{ijk} = -\epsilon_{ikj}$ ;

also gilt:  $rot v = rot f$  UND  $div v = div \nabla f$

Berechnung:  $v = g + \nabla f \rightarrow$  auf beiden Seiten Divergenz bilden  $\rightarrow \nabla \circ v = \nabla \circ g + \nabla \circ \nabla f$

Es gilt:  $div g = \nabla \circ g = 0!$

Nun numerische Integration von  $\nabla \circ v = \nabla^2 f$  ergibt  $\nabla f!$

## 2.5 Einfach

einfache Formate lassen sich besser optimieren, einfache Abbildbarkeit zum Rendern, einfache Umwandlung des ursprünglichen Formats in das interne Systemformat

## 2.6 Scattered Data

### Voronoi-Diagramme

- konvex,
- schneiden immer mind. 3 Kanten
- Seitenhalbierenden bilden Triangulierung
- höchstens  $2n-5$  Ecken

### Dualisierung:

Man fügt einen Punkt ein, entfernt alle Dreiecke in deren umschließenden Kreis der Punkt liegt und verbindet ihn mit allen Sichtbaren Dreiecken

## 2.7 Nachbarschaften

Dreieck, Viereck, Rechteck, Polygon

Tetraeder, Prisma, Quader

Dreieck 2. Ordnung mit je 1 zusätzlichem Wert pro Kante

Dreieck 3. Ordnung mit je 2 zusätzlichen Werten pro Kante und 1 im Schwerpunkt

## 2.8 Grid-Generierung

Gegeben: Fläche (CAD)

Gesucht: Kurvensatz

- Einfügen von Gitterpunkten im Parametergebiet, Abbilden auf Fläche
- Optimierung des Gitters oder:
- Optimierung des Kurvennetz durch conformal map

Gewünscht:

- winkelerhaltende Abbildung
- flächenerhaltend
- stetig

Aus Cauchy-Riemann Gleichungen folgt  $X_u * X_v = 0$  (Orthogonal) und  $X_u \times X_v$  entspr. Fläche  $\rightarrow$  lokale Kontrolle der Skalierung der Fläche

## 2.9 Interpolation

Coons-Patch:

- **bilineares** Coons-Patch aus 4 Randkurven ( $X(0, t), X(1, t), X(s, 0), X(s, 1)$ ):
- $P_1 X(s, t) := (1-s)X(0, t) + sX(1, t)$ ;  $P_2 X(s, t) := (1-t)X(s, 0) + tX(s, 1)$  interpolieren jeweils linear zwischen 2 Randkurven
- $QX(s, t) = P_1 X + P_2 X - P_1 P_2 X$  ( $P_1 P_2 = (1-s)(1-t)X(0, 0) + \dots$ )
- **bikubisches** Coons Patch: zusätzlich Ableitung am Rand interpolieren!
- Dazu Twist-Kompabilität notwendig ( $\delta/\delta s * X_t(s, 0) = \delta/\delta t * X_s(0, t)$ )
- $P_1 X$  und  $P_2 X$  nun über kubische **Hermite-Polynome** definiert statt linearer Interpolation ( $(1-s) \rightarrow \sum_{i=0..1} H_i(s)$  etc.

## 3 Interpolation

### 3.1 Lineare Interpolation d. baryzentrische Koordinaten

- Zelle in  $R^{n-1}$  mit n Pkt  $p_1 \dots p_n$  z.B. Dreieck
- Punkt p in der Zelle =  $\sum_{i=1}^n \alpha_i p_i$
- $\sum \alpha_i = 1, \alpha_i$  proportional zu den jeweiligen Teilflächen/volumen
- liegt p außerhalb der Fläche werden die  $\alpha_i$  negativ
- wählt man das Koordinatensystem mit Ursprung in einem der  $p_i$  und die Einheitsvektoren jeweils von dort zu den anderen  $p_i$  stimmen die ersten n-1 baryzentrischen Koordinaten mit den Kartesischen Koordinaten in dem System überein.
- Um die Attribute  $f_i$  der Eckpunkte einer Zelle linear zu interpolieren:  $f(p) = \sum_{i=1..n} \alpha_i f_i$

Isolinien sind nicht gekrümmt sondern Hyperebene/linear!

### 3.2 Multilinear

- Rechtecke: bilinear, Quader trilinear
- $f(p) = (1-v)((1-u)p_{00} + up_{10}) + v((1-u)p_{01} + up_{11})$   
(also erst linear zwischen 00 und 10 =z0, dann linear zwischen 01 und 11=z1, dann linear zwischen z0 und z1!)
- Isolinien sind gekrümmt!
- Basisfunktionen: Quader, trilinear:  
 $(1-w)((1-v)((1-u)p_{000} + up_{001}) + v((1-u)p_{010} + up_{011}))$   
 $w((1-v)((1-u)p_{100} + up_{101}) + v((1-u)p_{110} + up_{111}))$   
Ausmultiplizieren:  
 $(1-w)(1-v)(1-u)p_{000} + (1-w)(1-v)up_{001} + (1-w)v(1-u)p_{010} + vup_{011} + w(1-v)(1-u)p_{100} + w(1-v)up_{101} + wv(1-u)p_{110} + wvup_{101}$   
Ergibt also 8 Basisfunktionen und immer wenn index i des Punktes 0 ist die i-te Koordinate in der Basisfunktion mit 1- vertreten, wenn der index i des Punktes 1 ist die i-te Koordinate einfach so vertreten.  
Die 8 Basisfunktionen werden dann mit dem zugehörigem Punkt multipliziert und aufsummiert. also:  
 $\sum_{i=0..1} \sum_{j=0..1} \sum_{k=0..1} f_{ijk} B_{ijk}(u, v, w)$
- Pyramide: degenerierter Quader  
also Grundfläche (mit w\_koordinate=0 ganz normal), alles mit der w-Koordinate=1 weglassen und dafür eine neue Basisfunktion  $B_{.4} = w$
- Prisma: Grundflächen zuerst baryzentrisch, dann dazwischen linear.
- Baryzentrische Koordinaten lassen sich auf Polygone mit mehr als 4 Kanten verallgemeinern
- Allgemein: Zerlegung beliebiger Gitterelement in Tetraeder und dann linear interpolieren.  
Würfel kann man entweder in zwei Prismen, und die zwei Prismen in je 3 Tetraeder, oder direkt in 5 Tetraeder zerlegen

## 4 Bezier

Alles hier bezieht sich auf eine Parametrisierung t in [0,1]

## 4.1 de Casteljau

linear (2 Pkte): - Strecke wird linear (1, 1-t) unterteilt.

quadratisch (3 Pkt): - Zwei Strecken werden linear unterteilt, die 2 neuen Pkte verbunden, dann wieder linear.

kubisch (4 Pkte): 3 Strecken werden linear unterteilt, die 3 neuen Pkte verbunden, man hat 2 Strecken und macht weiter wie bei quadratisch. Rekursionsformel:

Der obere Index der Ursprungspunkte ist 0.

In jedem Schritt wird der obere Index erhöht, und die Anzahl der Pkte um 1 verringert, bis nur noch 1 Pkt ist. beim kubischen Fall ist das höchste j also 3:  $b_i^j = (1-t)b_i^{j-1} + tb_{i+1}^{j-1}$

$b_i^j$  entsteht also jeweils als Konvexkombination aus 2 Pkte  $b_i^{j-1} b_{i+1}^{j-1}$  gewichtet mit t bzw. (1-t)

$b_i^j$  ist ein Polynom vom Grad j.

## 4.2 Eigenschaften

- **Convex-Hull** (Kurve liegt in konvexer Hülle der Kontrollpunkte)
- **Variationsreduzierend** (Kurve gleichviel oder weniger Schnittpkte mit bel. Gerade)
- **Endpunkt-Interpolation** (Die Endpunkte exakt Interpoliert)
- **Affine Invarianz** (Bildet man die Kontrollpunkte und Kurve affin ab, entspricht die abgebildete Kurve die Bezierkurve der abgebildeten Pkte)
- **Bernstein-Basis** ( $X(t) = \sum b_i B_i^n(t)$ , mit  $B_i^n = \binom{n}{i} t^i (1-t)^{n-i}$ )
- **Symmetrie** (Umkehren der Reihenfolge der Kontrollpunkte = Umkehren der Parametrisierung ( $t = 1-t$ ))
- **Pseudo-Lokale Kontrolle** (Beim Verschieben des Pkt  $b_i$  ändert sich die Kurve am meisten bei  $t = i/n$ )
- **Graderhöhung** (indem man in alle Kontrollkanten bei  $t = (i/(n+1))$  neue Pkte einfügt und die anderen Punkte (außer start und end-pkt) entfernt).
- **Unterteilen** (indem man einfach mit deCasteljau den Punkt bestimmt, und die  $b_0^i$  (links) und  $b_0^{n-i}$  aus dem Algorithmus sind die neuen Kontrollpunkte).
- **Ableitung**: Bezier-Kurve eines niedrigeren Grades mit den Kontrollpunkten:  $b'_i = n(b_{i+1} - b_i)$   
ODER: direkt aus Casteljau:  $X'(t) = n(b_1^{n-1} - b_0^{n-1})$  - also die Steigung entspricht der letzten Strecke aus dem Casteljau-Alg vor der letzten Interpolation, die Strecke liegt also tangential an der Kurve an.

## 4.3 Interpolation

- Bezier Interpolation: Lineares Gleichungssystem:  $f_j = \sum_{i=0..n} b_i B_i^n(t_j)$  ( $j = 0..n$ ) ( $t_j =$  Stützstelle deren Wert  $f_j$  interpoliert werden soll,  $b_i =$  gesuchte Bezier-Pkte).
- Lagrange Interpolation: Lagrange-Polynome mit der Eigenschaft  $L_i(t_j) = \delta_{ij}$  Interpolation dann mit:  
 $X(t) = \sum_{i=0..n} p_i L_i(t)$ ,  $L_i(t) = \prod_{k:0..n, k \neq i} \frac{t-t_k}{t_i-t_k}$
- **Parametrisierung**: äquidistant, chordal ( $\|P_i - P_{i-1}\|$ ), zentripetal ( $\sqrt{\text{chordal}}$ ) minimiert zentripetal Beschleunigung beim Durchfahren der Kurve (Gute Qualität, geringe Überschwinger)
- Hermite Interpolation: Polynome  $H_{0/1}, \bar{H}_{0/1}$  mit  $H_i(0/1) = \delta_{ij}, \bar{H}_i(0/1) = 0, H'_i(0/1) = 0, \bar{H}_i(0/1)' = \delta_{ij}$  interpolieren jeweils zwei Punkte  $P_{0/1}$  mit Ableitungen:  $X(t) = P_0 H_0(t) + \dots$

## 5 Bezier-Flächen

### 5.1 Dreiecksflächen

reguläres Dreiecksgitter ( quadratisch: 4 Dreiecke, eins in der Mitte, kubisch: 9 Dreiecke, 6er Triangle-Fan in der Mitte )

Auswertung: de Casteljau:

Angefangen wird mit den Stützpunkten, diesmal haben sie 3 Koordinaten  $ijk$ .

Dann wird jeweils eine kubische Bezierfläche zu einer quadratischen, eine quadratische zu einem Dreieck usw., indem in jedem Dreieck anhand der 3 Parameter linear interpoliert wird und die so ermittelten Punkte (bei denen dann jeweils einen höheren oberen Index bekommen) verbunden werden.

$$b_{ijk}^r = ub_{i+1,j,k}^{r+1} + vb_{i,j+1,k}^{r+1} + wb_{i,j,k+1}^{r+1}$$

u,v,w sind die Parameter (so wie t bei der Kurve) an denen ausgewertet werden soll. Dabei muss gelten  $u+v+w=1$ , das heist es genügt zwei Parameter anzugeben, die im Intervall  $[0, 1]$  liegen sollen.

Man kann wieder Bernstein-Polynome als Basisfunktionen angeben so dass gilt:

$$f(u, v, w) = \sum_{i,j,k \geq 0; i+j+k=n} b_{ijk} B_{ijk}^n(u, v, w)$$

im Fall von n=3 ist also:

i 0 0 0 1 1 1 2 2 3

j 0 1 2 3 0 1 2 0 1 0

k 3 2 1 0 2 1 0 1 0 0

Also 10 Punkte für 9 Dreiecke.

Entlang einer Randkurve bleibt jeweils immer ein Index gleich, die anderen beiden geben nur an wie nah man am Eckpunkt ist, je gleicher die beiden index desto weiter ist man von beiden Eckpunkten weg  
Ableitung in Richtung  $d = (u_1, v_1, w_1) - (u_2, v_2, w_2)$  einer Fläche mit Grad n ist:

$$D_d f(u, v, w) = n * \sum_{i,j,k \geq 0; i+j+k=n-1} b_{ijk}^{n-1}(d) B_{ijk}^{n-1}(u, v, w)$$

## 5.2 Tensorproduktflächen

- Zunächst definiert man eine Schar von Kurven  $X_j$  mit  $X_j(s) = \sum_{i=0..m} b_{ij} B_i^m(s)$  mit dem de Casteljau für Kurven und den Stützpunkten  $b_{ij}; j = 1..m$
- Die Kurvenschar, an s ausgewertet, liefert m Stützpunkte für eine weitere Kurve:  
 $X(s, t) = \sum_{j=0..n} X_j(s) B_j^n(t)$
- Die Reihenfolge (erst in s-Richtung dann in t-Richtung oder umgekehrt ist egal)
- m=n=2 bilinear, m=n=3 biquadratisch, m=n=4 bikubisch...
- De-Casteljau: zwei obere Indizes. in jedem Schritt egal welcher obere index verringert wird, einmal wird mit t linear interpoliert zwischen zwei unterschiedlichen rechten indizes, einmal mit s zwischen zwei untersch. linken indizes:
- $b_{ij}^{k+1,l} = (1-s)b_{ij}^{kl} + sb_{i+1,j}^{kl}$
- $b_{ij}^{k,l+1} = (1-t)b_{ij}^{kl} + tb_{i,j+1}^{kl}$
- **Ableitungen, partielle** können mit dem de Casteljau wie gewohnt berechnet werden, es ist wieder die Gerade aus dem vorletztem Schritt die mit dem Grad multipliziert wird. Also nach s :  $X_s(s, t) = m(b_{10}^{m-1,n} - b_{00}^{m-1,n})$  und nach t :  $X_t = n(b_{01}^{m,n-1} - b_{00}^{m,n-1})$
- Man kann auch eigene Bezierflächen konstruieren die den Ableitungen entsprechen

### 5.2.1 Twistvektoren

Die partiellen Ableitungen in den Eckpunkten der Flächen-Segmente werden durch die Stützpunkte je einer Randkurve festgelegt.

Die Twistvektoren (Ableitung nach beiden Parametern s,t nacheinander im Eckpunkt) werden dementsprechend auch durch die inneren Kontrollpunkte festgelegt.

## 5.3 Bezier-Volumen

Das Konzept lässt sich analog auf Pentaeder-Volumen (Dreieck mit baryzent. koordinaten die in der Höhe zusätzlich per Tensorprodukt interpoliert werden) oder Tetraeder-Volumen (Verallgemeinerte Bezier-Dreiecke) oder auf Tensorprodukt-Volumen erweitern (3faches Tensorprodukt, 3 Summenzeichen)

## 6 Splines

- Bezier: Ableitungen werden an den Segmenttrennstellen gleich gesetzt
- Catmull-Rom: Ableitungen sind fest definiert:  $\frac{P_{i-1} + P_{i+1}}{2}$   
Dadurch nur C<sup>1</sup>-stetig trotz Grad 3, aber einfach zu berechnen + lokale Kontrolle

### 6.1 Dreiecke

- Lineare: Basisfkt. zu einem Pkt entspricht seinen baryzentrischen Koordinaten an dem Punkt und 0 in nicht benachbarten Dreiecken.
- Box-splines
- Bezier-dreiecke: Übergangsbedingungen erfordern Polynomgrad 5 für C<sup>1</sup> Stetigkeit
- Clough-Tocher-Interpolant: Durch zerlegen in kleinere Dreiecke kommt man mit Polynomgrad 3 aus.

## 6.2 B-Splines (Wavelets)

**B-Spline Grad k** definiert zwischen  $[x_{k-1} \dots x_{m+1}]$

- **m+1 Kontrollpkt**  $d_0 \dots d_m$  DeBoor
- **m+k+1 Knoten**  $x_0 < \dots < x_{m+k}$  Knotenvektor

(Fürs erste Segment k Punkte nötig, für jedes weitere Segment noch ein Pkt)

Uniformer B-Spline: Der Knotenvektor ist  $x_i = i$

### 6.2.1 de Boor Algorithmus

- zunächst die zwei Knoten  $x_r, x_{r+1}$  ermitteln zwischen denen t liegt
- $d_{r-k+1} \dots d_r$  (also k Kontrollpunkte) fließen in f(t) ein
- Wie beim Decasteljau wird innerhalb dieser k Pkte schrittweise verfeinert.
- Auf einer Geraden zwischen den Kontrollpunkten  $d_{i-1}, d_i$  wird also jeweils ein neuer Punkt bestimmt, und zwar im gleichem Verhältnis wie t das Intervall  $[x_i, x_{i+k-j}]$  unterteilt.
- j ist der obere index, beginnend bei 0, wird erhöht bis er k-1 ist, dann hat man nur noch einen Punkt

### 6.2.2 Basisfunktionen

$N_{i,1}(t) = 1$  falls t im Intervall  $x_i, x_{i+1}$ ; 0 sonst

$N_{i,k}(t) = (t - i)/(k - 1) * N_{i,k-1}(t) + (i + k - t)/(k - 1) N_{i+1,k-1}(t)$  (falls UNIFORM)

Dann ist  $f(t) = \sum_{i=0..m} d_i N_{i,k}(t)$

m=Anz. Kontrollpunkte,  $N_{i,k}$  ist aber nur k mal ungleich 0

### 6.2.3 Eigenschaften

- lokale Kontrolle (nur zwischen k Knoten ändert sich was)
- lokale Convex-Hull
- Basisfkt stückweise Polynome und C<sup>k</sup>-stetig

## 7 Wavelets

Es werden Räume  $V_i$  aufgespannt zu den Basisfunktionen (Wavelets) - z.B. den Basisfkt des deBoor-Alg.  $N_{i,k}$ , oder beliegbige andre 'Skalierungsfkt'.

Dazu gibt es Duale Räume  $W_j$ .

Die Räume  $V_j$  entsprechen der Darstellung in verschiedenen Auflösungen, die  $W_j$  dienen dazu die Differenzen zweier Auflösungsstufen zu speichern.

Die Umwandlung zwischen zwei Räumen  $V_i$  und  $V_j$  ist die diskrete Wavelet-Transformation.

### 7.1 Wavelet-Transformation

- Basis Phi und Psi der Räume die immer 'größer' werden, einfachste Basis ist die Haar-Basis.
- Es gilt:  $f_{V_j}(t) = \sum_k s_k^j \phi_k^j(t); f_{W_j}(t) = \sum w_k^j \psi_k^j(t)$
- Die s sind die Daten auf Detaillevel j, Die w sind die Informationen über die Differenz zum nächstem Detaillevel
- Damit verknüpft sind automatisch Folgen l, h, ...
- Die Transformation lässt sich somit schreiben als:  $s_k^j = \sum_p l_{p-2k} s_k^{j+1}$ , für w genauso nur mit Folge h.
- Rückwärts wird s\*h und w\*h kombiniert:  $s_k^{j+1} = \sum l_{k-2p} * s_p^j + h_{k-2p} * w_p^j$
- Die Folge l ist ein Tiefpassfilter, h ist ein Hochpassfilter.
- Die Folgenglieder können auch einzeln berechnet werden. z.B. bei der Haar-Transformation:  $w_i^k = s_{2i}^{k+1} - s_{2i+1}^{k+1}$  (also Unterschied zwischen Daten in der Folge eine Auflösungsstufe genauer)  $s_i^k = s_{2i}^{k+1} - 0.5w_i^k$  (also Unterschied zur Hälfte abziehen)

## 8 Filterung

**Fehlende Werte:**

- Nicht-Wert
- Interpolation

**Region of Interest**  
**Fehlerschranken**

## 8.1 Glättungsverfahren

Lowpass-Filter (Gauß, konstant über kreisförmige Umgebung) Auch gegen Aliasingeffekte vor der Abtastung einsetzen

## 8.2 Schärfen

Medianfilter die Rauschen entfernen, indem sie nur bestimmte Nachbarn zulassen

Hochpassfilter zeigen nur die Kanten

## 8.3 Transferfunktionen

durch Analyse des Histogrammes den ganzen Sichtbaren Bereich ausnutzen, alles gleichmäßig verteilen

## 9 Scattered Data Methoden

### 9.1 Shepard

gewichtete Linarkombination der Attribut-Werte:

- $F(x, y) = \frac{\sum_{i:1..n} w_i(x, y) f_i}{\sum w_i(x, y)}$
- $w_i(x, y) = \frac{1}{((x-x_i)^2 + (y-y_i)^2)^{mu_i/2 + c_i}}$

Eigenschaften der Gewichte:

- $C^0$ -Stetig
- nicht negativ
- $w_i(x_j, y_j) = \delta_{ij}$ , d.h. =1, falls  $i=j$
- $\sum w_i = 1$

$mu_i < 1 \rightarrow$  Interpolant hat Spitze,  $> 1 \rightarrow$  Interpolant hat Flachpunkt  
Flachpunkt vermeidbar durch  $c_i > 0 \rightarrow$  dann nur approximierend.

**Lokal** Mit lokaler Dämpfungsfkt. Multiplizieren:

Frank-Little:  $g(q)^{mu+} = (1 - \|q - p\|/r_i)^{mu+}$

### 9.2 Radiale Basisfunktionen

$R_i = R(\|q - p_i\|)$  Funktionen des Abstandes von den Stützpunkten

Interpolant:  $f(q) = \sum_{i:1..n} a_i R_i(q) + \sum_{j:1..m} b_j P_j(q)$

$P_j$  dabei irgendwelche Polynome

$R_i$  oft:  $R_i = ((\|q - p_i\|)^2 + r_i^2)^{mu/2}$ ,  $mu = 1$  oder  $-1$ ;  $r_i$  =mittler Abst. der Stützpunkte.

Koeffizienten  $a_i, b_j$  bestimmen aus System:

$f(p_i) = f_i$  und  $m$  physikal. Gleichgew.bedingungen (Summe der Momente=0): für  $j = 1..m : \sum_{i:1..n} a_i P_j(p_i) = 0$

Beim Least-Squares-Fitting kann  $r$  als Parameter in den Optimierungsprozess (also  $\sum_{i:1..n} (f_i - f(p_i))^2 \rightarrow min$ ) einbezogen werden!

### 9.3 Least Squares Fitting für RBF's

Nicht für jeden Stützpunkt, sondern nur für spezielle Knotenpunkte wird eine RFB konstruiert. Wie wählt man die Knoten:

- initiale(r) Knoten wählen.
- Interpolant berechnen, Fehler an restl. Stützpkt auswerten
- bei zu großen Abweichungen neue Knoten einfügen

Optimierungsprozess:  $\sum_{i:1..n} (f_i - f(p_i))^2 \rightarrow min$

Wobei statt  $b_j * P_j(q)$  einfach nur  $b$  verwendet wird.

Knotenwahl + Wahl von  $r$  = Fehler-Minimierungsprozess.

Danach Least-Squares-fitting von  $a_i, b_j * P_j$  extra Minimierungsprozess.

### 9.4 Thin-Plate-Splines

Minimierung der Biege-Energie einer sehr dünnen unendlich großen Platte die in den Interpolationspunkten aufliegt

$$\int_{R^2} (f_{xx}^2 + 2f_{xy}^2 + f_{yy}^2) dx dy \rightarrow min$$

(Um Überschwinger zu vermeiden: zusätzlich erste Ableitungen hinzuaddieren!)

Lösung: Radiale Basis Funktionen der Form:

$$R_i(x, y) = d_i^2(x, y) \log d_i(x, y)$$

( $d_i$  =Abstand des Pkt  $(x, y)$  vom Stützpunkt  $i(p_i)$ )

## 9.5 Lokale Verfahren

Einführung eines **Rechteckgitters**. Für jede Gitterzelle wird ein Spline ermittelt der die dortigen Stützpunkte Interpoliert.

Andre Möglichkeit: **Hierarchische B-Splines**: Statt 2 globalen Knotenvektoren: Quadtree-Unterteilung

- lokales Fitting mit polynomialer Fläche
- Blending der Patches zu kontinuierlicher Fläche durch Knotenentfernen(Zusammenführen)
- Fehlerbestimmung
- Quadreeunteilung der Patches mit zu großem Fehler

## 9.6 Least-Squares-Fitting Allgemein

**gesucht:**  $f(t) = \sum_{i:1..n} c_i \phi_i(t)$

Bestimmung mit System  $F(t_i) = f_i$  falls  $m = n$

Sonst **Least-Squares-Fit**:

$$\sum_{j:1..m} \|F(t_j) - f_j\|^2 \rightarrow min$$

**Fomel:**  $2A^T A c = 2A^T f$

wobei  $a_{ji} = \phi_i(t_j)$  - Also Auswertung der Basisfunktionen an den Stützpunkten

Herleitung: Über notwendige Bedingung für ein Minimum:

$d \sum_{j:1..m} \|F(t_j) - f_j\|^2 / dc_i = 0$  führt durch umformen auf das System.

**Kontinuierlich:** Statt Summe zu minimieren muss Integral minimiert werden!

## 10 Visualisierungstechniken

### 10.1 Skalardaten

#### 10.1.1 2D

**Höhenfelder** LevelOfDetail: 4-8-Textur-Hierarchien

**Isokurven**

#### 10.1.2 3D

**Cutting Planes** jedem Skalarwert wird eine Farbe zugewiesen

Die Cutting Plane kann durch das Skalarfeld hindurch bewegt werden

**Colormap**

- Skalar Datensatz wird durch Addition und Skalierung auf Werte zw.  $[0, 1]$  normiert
- Colormap = Tabelle mit fester Anzahl, oder
- Colormap = stückweise lineare Farbtabelle

RGB-Farbmodell = Würfel (schwarz und weiß liegen sich diagonal gegenüber, restl. ecken wie hls)

HLS-Farbmodell = Pyramide mit 6-eckiger Grundfläche, Spitze schwarz, Grund weiß, Ecken Farben (blau magenta rot gelb grün cyan)

**Direct Volume Rendering** Skalardaten werden opt. Eigenschaften (Farbe, Durchsichtigkeit(Transparenz), absorption, emmission) zugeordnet.

Dann **Ray-Casting**: Sehstrahlen durch das Volumen verfolgen und die Farbwerte aufintegrieren.

Beleuchtung: z.B. Phong mit Normale=Gradient (normale der isofläche)

Transparenz( $\alpha$ ) = Einflussfunktion, um bestimmte Bereiche hervorzuheben (z.B. Kanten)

$$I = \sum_{i:1..n} (\prod_{k:1..i-1} \alpha_k) (1 - \alpha_i) I_i$$

**Phong**

Beleuchtung wird durch Normale  $N$  des Flächepunktes und Vektoren  $L$  (richtung Licht) und  $V$  (Richtung Betrachter) berechnet.  $N$  zeigt dabei immer zur Seite der Fläche in Richtung Licht.

Es gibt einen **diffusen** einen **spiegelnden** und einen kleinen **ambienten** (hier 0) Term

$$I = k_{diffus} * I_{diffus} + k_{spiegel} * I_{spiegel} + k_{ambient} * I_{ambient}$$

$$I_d = I_L * (L^\circ N); I_s = I_L * (R^\circ V)^n = I_L * ((2(L^\circ N)N - L)^\circ V)^n$$

( $R$ = Reflektionsrichtung,  $n$ =je nach dem wie stark in eine Richtung es spiegelt)

**Alternative zur Beleuchtung**

Maximum Intensity (größter Skalarwert)

Average Value (Mittel der Skalarwerte) (RÖNTGENBILD)

Distance to Max

## Alternative Berechnung

Texture Planes: Beleuchtungs und Transparenz-Daten werden in Texturen gespeichert und als hintereinander-liegende Scheiben von der Grafikkarte gerendert.  
GPU-basiertes Volume-Rendering durch Shader-Programmierung

## Isoflächen Marching Cubes

Es wird von trilinear Interpolation ausgegangen.

Jede Kante wird von der **Isofläche einmal oder keinmal geschnitten**.

Die Würfel werden nach Eckpunkt-Konfigurationen (Attribut an Eckpunkt: größer Isowert oder kleiner Isowert) klassifiziert. Es gibt  $2^8 = 256$  Möglichkeiten

In einer Tabelle wird für jede Konfig eine Bitmaske gespeichert, wo bei  $bit_i = 1$  falls  $kante_i$  wird geschnitten.

Weitere Tabelle enthält Triangulierung (Liste von Dreiecken mit Kantenummern als Ecken)

256 Konfigs lassen sich durch Drehung und Inversion auf 15 zurückführen.

Dann jedoch **Inkonsistenzen**: sollen die **positiven eckpunkte 'verbunden'** sein, **oder die negativen** (Hüllt die Isofläche große Werte (wie isolinien am berg) oder kleine Werte (wie isolinien am tal) ein).

**konsistente Lösung**: Immer die negativen miteinander verbinden.

**korrekte Lösung**: Sattelpunkte der bilinear interpolierten Randflächen: ist der Isowert kleiner als der Mittelwert, werden die Positiven verbunden

Indem man zusätzliche Punkte im Zellinnerem einfügt kann man die Isofläche an die Form des trilinearen Interpolanten anpassen!

### Beschleunigung durch Contour Trees

Statt alle Zellen zu durchlaufen, wird die Isofläche jeweils von einer Saatzelle aus bestimmt

Die Knoten des Contour-Tree entsprechen den Extrempunkten (Minima, Maxima, Sattel).

Er kann im Koordinatensystem mit y-Achse als Skalarwerte aufgetragen werden. Sucht man einen Isowert zieht man eine Horizontale durch den Skalarwert auf der y-Achse. Alle Schnittpunkte im Contour-Tree entsprechen dann einer Saatzelle.

### Hierarchische Isoflächen

Octree, innerhalb jeder Octree-Zelle gewöhnlicher Marching-Cubes mit irgendeiner Auflösung.

An den Grenzen von Zellen mit unterschiedlicher Auflösung müssen die Grenzdreiecke der Zellen mit der größeren Auflösung in Triangle-Fans aufgesplittet werden, damit es keine Cracks gibt.

## 10.2 Vektorfelder

### 10.2.1 Strömungen

#### Hinzufügen von Material Time-Lines/Surfaces

Viele Teilchen werden zur gleichen Zeit eingebracht und bewegen sich dann gemeinsam weiter

- Mehrere Partikel zeitgleich starten

#### Streak Lines

Farbpartikel an einer festen Position ständig in die Strömung einbringen

mehrere Partikelbahnen

- vom selben Punkte
- jede Partikelbahn in einem anderem Zeitschritt

#### Optische Techniken Path Lines

Partikelbahnen: die Bahn einiger einzelner Partikel werden visualisiert, z.B. leuchtende Partikel, Kamera-Aufnahme mit langer Belichtungszeit.

- Integriere über die Zeit

### 10.2.2 Berechnung von Stromlinien

- Stromlinien in stationären Strömungen (hier kann die Zeit vernachlässigt werden): entsprechen Integrationskurven. Stromlinien visualisieren nur RICHTUNG des Feldes. Durch **Colormapping** kann man weitere Informationen (Betrag) mitvisualisieren.
- Path-Lines: hier ist der Kurvenparameter die Zeit, also statt  $x_i$  wird  $c(t_i)$  geschrieben. Die Schrittweite  $h_i$  ist an ein Zeitintervall  $\delta t_i$  gekoppelt

- Time-Lines: ähnlich, zusätzlich wird eine interpolierende Kurve zwischen mehreren zeitgleich gestarteten Partikeln für jeden Zeitschritt berechnet
- Streaklines: Hier werden mehrere Partikelbahnen aus verschiedenen Zeitschritten übereinandergelegt.

### Polygonzugverf. von Euler

- $x_{i+1} = x_i + h * v(x_i)$
- Genauigkeit  $O(n)$

### Modifiziertes Eulerverfahren:

- $x_{i+1} = x_i + h * v(x_i + h/2v(x_i))$
- Genauigkeit  $O(n^2)$

### Runge-Kutta

- $q1 = v(x_i)$
- $q2 = v(x_i + h/2q1)$
- $q3 = v(x_i + h/2q2)$
- $q4 = v(x_i + hq3)$
- $x_{i+1} = x_i + h/6(q1 + 2q2 + 2q3 + q4)$
- Genauigkeit  $O(n^4)$

**Adaptive Wahl der Schrittweite** Step Doubling: Vergleiche das Ergebnis mit halber Schrittweite mit dem Ergebnis. Falls unter Fehlerschranke, verdopple Schrittweite für nächsten Schritt. Falls drüber - halbiere Schrittweite in diesem Schritt und probiers nochmal

### 10.2.3 Stromlinien- Techniken

**Stream Tubes** Durchmesser einer Röhre ändern um Geschwindigkeit, Druck, Temperatur anzuzeigen

**Stream-Ribbons** um die Stromlinie wird ein Band gezeichnet, welches durch seine drehung die Rotation visualisiert.

Die Breite des Bandes kann zusätzlich die Divergenz visualisieren

**warping / displacement plots** Visualisierung der Vektordaten durch Verformung des Beobachtungsraumes, z.B. eines Balkens unter Belastung

- Skalierung: verformung sollte deutlich sichtbar sein.
- Selbstdurchdringung soll vermieden werden

**Hedgehodge** An jedem Gitterpunkt wird ein Pfeil (oder eine kurze Stromlinie) berechnet

- + Schnell zu berechnen
- - keine Interpretation, Interpolation

**local flow probe** hier werden durch versch. Ringe, scheiben, krümmung weitere Informationen wie rotation, divergenz etc visualisiert.

**Stream Polygon** Quadrat um die Stromlinie, dass sich verformt (Rotation = Drehung, Scherung etc.)

### Streamballs

- Menge von Zentren  $S_i$  der Metabälle
- Menge von Gewichten  $w_i$ , die festlegen welchen Einfluss welches Zentrum hat
- Menge von Abstandsfunktionen  $I_i$  die die Form der Metabälle bestimmen

Zur Visualisierung wird eine Isofläche der Funktion  $F_S(x) = \sum w_i I_i(x)$  bestimmt.

Nun gibt man Startpositionen einiger Zentren vor und verfolgt deren Partikelbahnen.

Die Geschwindigkeit wird durch den Abstand der Zentren veranschaulicht.

Wenn eine Fläche zum Beispiel sich verbiegt wie ein Netz in dass ein Ball fliegt, sieht man wie ein Hindernis umströmt wird.

**Illuminated Streamlines** Von den Normalen der Stromlinie wird diejenige ausgewählt, die in einer Ebene aus Lichtvektor und Stromlinientangente (vektor) liegt.

Der Reflexionsvektor liegt in einer Ebene mit der Tangente und dem Vektor  $V$  zum Betrachter.

Nun wird das Phong-Modell  $I = I_a + I_d + I_s$  angewendet.

Zur Beschleunigung werden  $I_d = k_d(L \circ N)$  und  $I_s = k_s(V \circ R)^n$  vorberechnet und in einer Textur gespeichert:  $\text{Textur}(t1,t2)=(I_a, I_d, I_s)$   
Zum Zugriff muss man nur noch  $L \circ T$  in  $t1$  ( $(t1 = L \circ T + 1)/2$ ) und  $V \circ T$  in  $t2$  umrechnen.

Dies geschieht dann durch eine einfache  $4 \times 4$  Matrix-Multiplikation auf der GPU.

## 10.2.4 Texturbasierte Techniken

**Spot Noise** Grundform:  $f(x) = \sum_i a_i h(x - x_i)$   $a_i$  Zufallswerte mit Erwartungswert 0;  $h(x) = \text{Form (Ellipse)}$

**DDA-Convolution** Vektorfeld über Bild mit weissem Rauschen  $\rightarrow$  Linie in Richtung des Vektorfeld mit DDA Rastern  $\rightarrow$  diesen Spot als Konvolutionsfilter (Mittelwertfilter) verwenden.

**LIC** Statt nur kurze Linie in Richtung des Vektorfeld zu rastern will man eine ganze Stromlinie um einen Punkt rastern und als Konvolutionsfilter benutzen.

Dazu bildet man von einem Punkt aus die Stromlinie gleichweit vorwärts und rückwärts.

Zum Beispiel mittels rasterisiertem Eulerverfahren:  $P_0 = (x + 0.5, y + 0.5)$ ,  $P_i = P_{i+1} + V(\text{abgerundet}(P_i - 1))/\text{Betrag}(V(\text{abgerundet}(P_i - 1))) * \delta s_i - 1$

Entlang der Streamline  $c(s)$  faltet man dann die Textur  $T$  mit einem Filterkern  $k$ :

$$I(x, y) = \int_{-L..L} k(s - s_0) T(c(s)) ds$$

(wird durch normierte Summe über  $T$  approximiert)

ANIMATED LIC: durch Bewegen des Filterkernes entlang der Stromlinien.

**Fast-LIC** Im Falle eines konstanten Kerns, kann das Integral für einen auf der Stromlinie verschobenen Punkt aus dem Integral für den Punkt daneben + einen Anteil für das hinzugekommen Ende und einen Anteil für das fehlende Ende.

Auf einem grobem Raster werden lange Stromlinien (runge kutta) gestartet. Für jedes Pixel wird die durch alle überlappenden Streamlines berechnete Intensität gemittelt. Wo noch keine Streamline durchgeht, wird eine neue kurze gestartet.

## 10.2.5 Stromflächen

Start mit einem Streckenzug an dessen Ecken Stromlinien gestartet werden. Zwischen 2 Stromlinien: Band spannen. Diese Fläche triangulieren

Möglichkeit vorsehen neue Stromlinien zu starten (dreieck mit spitze zur neu gestarteten stromlinien in der mitte von zwei andren) und stromlinien zu entfernen wenn die kurven sich zu nahe kommen.

Sobald das Verhältnis zwischen Vorwärts-Abstand und seitlichem Abstand nicht stimmt, wird reagiert.

## 11 Level of Detail Methoden

Vereinfachung polygonaler Oberflächenmodelle

- LOD für Manigfaltigkeiten (Flächen die lokal auf Ebene abbildbar sind)
- LOD f. beliebige Netze
- Topologie erhaltend
- Topologie vereinfachend

### 11.1 Mesh Simplification

- Vertex Decimation (entfernen einzelner Punkte)
- Edge-Collapse (verschmelzen der Endpunkte einer Kante)
- Trangle-Shrinkage (ersetzt Dreieck durch Pkt)
- Vertex Clustering

#### 11.1.1 Edge Collapse

Kollabierung einer Kante. Zwei Dreiecke werden eliminiert.

Invers: Vertex-Split

#### 11.1.2 progressive Meshes

Differenz.Vektoren beim Kanten-kolabieren werden gespeichert  $\rightarrow$  so kann das Ausgangsnetz inkrementell durch Splitten von Punkten wieder hergestellt werden.

#### 11.1.3 Fehler-Metrik

Um die Qualität des vereinfachten Netztes zu beurteilen: ZWEISEITIGE Metriken:

$$D(M1, M2) = 1/\text{anzahl\_pkte}(\sum_{v \in X1} d(v, M2) + \sum_{v \in X2} d(v, M1))$$

#### 11.1.4 Quadric Error Metrics

Inkremetell berechnbare Metrik, um die nächste zu kolabierende Kante (die mit dem geringstem Fehler) auszurechnen.

Der Fehler (Die Metrik) eines Punktes ist sein Abstand zu seiner Position im Originalem Netz.

Für jeden Punkt wird eine  $4 \times 4$  Matrix aus den Flächen der benachbarten Dreiecke gebildet.

## 11.2 Streaming Meshes

Abarbeitung entlang einer möglichst kleinen Fläche/Linie (weil ganzes Netz nicht in Hauptspeicher passt).

## 11.3 Subdivision-Surface-Wavelet

- Grobes Basis-Netz
- Unterteilen und addieren von Details
- Ausgangsnetz muss jedoch neu parametrisiert werden!! (über größere Basisgitter)

## 11.4 4-8-Meshes

- Gitter wird bei Verfeinerung/Vergrößerung um  $45^\circ$  gedreht
- lokal reguläre Gitter, die um  $45^\circ$  rotiert übereinanderliegen
- Vergrößern=halbe Anzahl Pkte (statt viertel ohne Drehung)
- Tiefpassfilter

**Entscheidung ob unterteilt werden soll:**

- Fläche die Polygon in Bildebene einnimmt
- Split-Queue und Merge Queue mit Elementen die verfeinert/vergrößert werden sollen, sortiert nach dringlichkeitsgrad

Verwendbar für Basisgitter aus 4seitigen Patches.

## 12 Volume Rendering

Ansatz:

- Transferfunktion weist Skalarwerten Farben zu
- Klassifizierung weist Skalarwerten Opazität zu

Beide Prozesse sind getrennt.

- Ergebnis: Farb/Transparenz-Werte im Volumen um Raytracing etc durchzuführen
- Mittel Raytracing ergeben sich letztendlich die Pixel-Wert

### 12.1 Clipping

Aus dem Volumen Teile rauschneiden und Schnittebenen visualisieren. Polygonale Beschreibungen/Gitter  $\rightarrow$  Clipping (Ebene) meist von Grafikhardware unterstützt.

### 12.2 Slicing / Capping

Jedem Skalarwert wird eine Farbe zu gewissen, jeder Punkt der Clipping-Plane entsprechend einfärben.

Bewegen und Drehen der Clipping Plane durch das Volumen.

Unebene Clipping Plane: Gitter mit Plane schneiden, Daten richtig interpolieren.

Hardwarebeschl. mit 3D-Textur für die Skalarwerte, 1D-Textur für die Colormap und kaskadieren der 2 Texturen.

### 12.3 Direct Volume Rendering

Raum mit Medium gefüllt, dessen opt. Eigensch. durch Skalarwerte beschr. werden.

Bsp: Licht absorbierende, Licht emittierende Partikel.

### 12.3.1 Transferfunktion

überträgt Skalarwerte (und manchmal auch deren Gradient) auf Opt. Eigenschaften (Farbe, Opazität) Möglichkeit, Iso-bereiche, Kanten etc. stärker hervorzuheben durch Transferfunktion.

### 12.3.2 Object-Order

Zellen/Voxel als Grundprimitive, werden auf Bildebene abgebildet (forward mapping)

### 12.3.3 Image-Order

Pixel als Grundprimitive, Sehstrahlen durch jedes Pixel auf der Bildebene (reverse-mapping)

## 12.4 Ray-Casting

- Image-Order
- Ausgehend von Bildebene werden Sehstrahlen durchs Volumen verfolgt. (normal Parallellprojektion (selten Perspektivische Projektion))
- 

### 12.4.1 Sampling

Nicht nur an Relektions-punkten mit Objekten (Raytracing) sondern entlang des ganzen Sehstrahls werden Einflüsse berücksichtigt.

Dazu äquidistante Proben. Abstand **mindestens eine Probe pro Zelle!**

Oder: Sehstrahl **in Segmente aufteilen** (ein Segment in genau einer Zelle).

- für Jedes Segment dann Maximum/minimum/Durchschnitt/Integral berechnen

**Voxel aufzählen** mit dem Bresenham-Algorithmus auf 3D verallgemeinert kann man die 6/18/26-verbundene Folge von Voxeln erhalten durch die der Sehstrahl geht

**vorberechnete Schablone** die Voxel durch die der Sehstrahl geht, können bei Parallellprojektion vorberechnet werden. Abtastung dann entlang der verschobenen Schablone, jedoch so dass an einer Grenze des Volumens begonnen wird

**Shear Warp** Ebenen des Volumens so gegeneinander verschieben (Shear), dass danach Abtastung senkrecht zum Gitter erfolgen kann. Entstehendes Bild muss noch entzerrt werden (Warp).

### 12.4.2 Beleuchtungsmodelle

**Reine Absorption** Skalarfeld spezifiziert die Partikeldichte, Partikel= Kugeln mit Radius r, projizierte Fläche= $\pi r^2$  Volumen-Ausschnitt der Dicke  $\delta_s$ , Grundfläche E:

- $V = E * \delta_s$
- $N = \rho * E * \delta_s$  Partikel
- Bei kleinem  $\delta_s$  :  $NA/E = \rho * A * \delta_s$  =Anteil der von Partikel abgedeckten Grundfläche
- I = Lichtintensität
- Differentialgleichung:  $dI/ds = -\rho(s) * A * I(s) = -\tau(s) * I(s)$
- Lösung:  $I(s) = I_0 * \exp(-\int_{0..s} \tau(s)ds)$

Statt  $\tau(s)$  (Absorptionskoeffizient) wird manchmal auch  $\alpha(s)$  = Opazität =  $1 - \tau(s)$  angegeben

**Reine Emmission** C=Leuchtstärke,  $\rho$ =Dichte,  $dI/ds = C(s) * \rho(s) * A = C(s) * \tau(s) =: g(s)$   
 $I(s) = I_0 + \int_{0..s} g(x)dx$

**Emmission und Absorption** Licht  $I_0$  kommt von Rückseite ( $s = 0$ ) in Volumen, geht komplett durch und gelangt hinter  $s = D$  zum Beobachter.

$dI/ds = g(s) - \tau(s) * I(s)$   
 $I(D) = I_{\text{absorptionsmodell}(D)} + I_{\text{emmissionsmodell}(D)}$  - was emittiert und danach wieder absorbiert wird.)  
 $I(D) = I_0 T(0) + \int_{0..D} g(s) * T(s)ds$   
 $T(s) = \exp(-\int_{s..D} \tau(x)dx)$  (= Transparenz des gesamtVol. von s bis D)

**Farben**  $g(s)$  in 3 Komponenten aufspalten, Transferfunktion muss jedem Skalarwert nun Opazität und Emmissionsstärke in 3 Farben zuweisen.  $I(S)$  muss nun für alle 3 Farben getrennt berechnet werden

### Transferfunktionen

- Tabellen mit 256 Einträgen (bei Grauwerten)
- Stückweise Lineare Transferfunktionen
- (Splines.)

**Sabella** Ergebnisse ähnlich Absorption+Emmission Auswertung:

- Helligkeit = Intensität nach Auswertung d. Beleuchtungsmodell
- Farbton = Maximum entlang des Sehstrahls
- Sättigung = Distanz zum Maximum (Nebel lässt entfernte Objekte verblassen)

**Levoy** Gradient geht mit ein um Eindruck von Flächen innerhalb des Volumens zu erzeugen, und geht auch in Opazität ein.

$c(x_i) \rightarrow$  weist  $x_i$  eine Farbe zu

$\alpha(x_i) \rightarrow$  weist  $x_i$  eine Opazität zu

$$c(x_i) = c * k_a + \frac{c}{m_1 + m_2 d(x_i)} * [k_d(N(x_i) \circ L) + k_s(N(x_i) \circ H)^n]$$

- $c = c_{L,lambda} =$  **Intensität** der Lichtquelle
- $k_a(,lambda)$  ambient,  $k_d$ diffus,  $k_s$ spiegelnd **Reflektionskoeff.**
- $m_1, m_2 =$  Entfernungsmodell
- $H = L + V/||L + V||$  Normierter Vektor von Lichtquelle zu Beobachter
- $N(x_i)$  Flächennormale
- $n =$  Shininess

### Zuordnung Opazität zu Skalarwert

- Isoflächen: Transferfkt. so, dass nur ein best. Isowert undurchsichtig
- Übergang zw. Regionen: Datensatz wird als Ansammlung von Regionen mit homogener Dichte angesehen Transferfkt so, dass Grenzen zwischen Regionen undurchsichtig.

### Isoflächen

- Setzt man nur Isowert auf 1, alles andere auf 0  $\rightarrow$  Aliasingartefakte!
- Daher um Isowert herum abfallen lassen, möglichst überall gleiche Schichtdicke
- Opazität fällt daher umgekehrt proportional zur Größe des Gradienten (je größer der Gradient, desto langsamer fällt Opazität mit Skalarwert)
- Es ist auch möglich mehrere Isoflächen mit einer Transferfunktion zu visualisieren.

**Gewebe Isobereiche Regionsübergänge** Um Übergänge zwischen Geweben zu visualisieren, wo der Isowert evtl. stark schwankt, kann man stattdessen auch ein ganzes Intervall hervorheben

- Die Skalar-werte unterschiedlicher Gewebe liegen nah beieinander
- Man kann die Gewebe  $f_i$  so in einer Folge ordnen, dass immer nur die Gewebe sich berühren, die in der Folge direkt hintereinander stehen.
- Jedem Gewebe-Skalarwert  $f_i$  wird eine Opazität zugewiesen, Zwischenwerte werden linear interpoliert.
- Um die Übergänge hervorzuheben, wird diese Transferfunktion mit dem Gradienten skaliert, so werden Bereiche mit großem Gradientem undurchsichtiger, der Rest wird durchsichtiger

**Ebert/Rheingans** Abweichung von traditionellen Modellen: Die aus dem Beleuchtungsmodell gewonnen Farben/Opazitäten werden im Nachhinein noch verändert:

$$\alpha(f) = (k * f)_{\text{kontrast}}^k$$

Feature-Verstärkung durch

- Skalierung der Opazität durch Gradient
- (Gradient auf Vektor in Richtung Beobachter projiziert) wird zur Skalierung der Opazität verwendet, um Regionsübergänge senkrecht zur Beobachtungsrichtung zusätzlich zu verstärken.

### 12.4.3 Integral lösen

Transparenz  $T(s)$  stückweise konst.  $T(s) = \exp(-\int_{0..s} \tau(x) dx) = \exp(-\sum_{i:1..n} \tau_i * \delta_x) = \prod_{i:1..n} (\exp(-\tau_i * \delta_x))$   
( $\exp(-\tau_i * \delta_x) := t_i$  **Transparenz des i-ten Strahlsegmentes**)

**Gesamtintegral**  $I(D) = \sum_{i:0..n} (g_i * \prod_{j:i+1..n} (t_j))$  mit  $g_1 := I_0$   
Falls man die Opazität  $\alpha$  verwendet und die Nummerierung umdreht (vom Beobachter (=0) durchs Objekt zum Licht (=n) statt vom Licht zum Beobachter) ergibt sich:  
 $E_i = C(i * \delta x) * \alpha(i * \delta x)$  und  $I(D) = \sum_{i:0..n} E_i \prod_{j:1..i} 1 - \alpha$

**Compositing** Pixelweises verknüpfen übereinanderlieg. Bilder.

F, B = Farbwerte,  $\alpha$  = Opazität  $\rightarrow$   
 $B @ F = (1 - \alpha) * B + \alpha * F = B + \alpha * (F - B)$   
Opazität der Kombination zweier Bilder, um kombinationen in beliebiger Reihenfolge zu berechnen:  
 $1 - \gamma = (1 - \alpha)(1 - \beta) \rightarrow \gamma = \alpha + \beta - \alpha * \beta$   
(Opazität= Deckkraft=1-tranzparenz. tranzparenzen multiplizieren sich wenn man sie übereinanderlegt, denn zwei übereinanderliegende sind weniger transparent.)

**assoziierte Farbwerte** schon mit Opazität vormultipliziert:  $F = \alpha * F$   
Nun kann man die kombinierte Opazität von Bild1 ( $\alpha$ ) und Bild2 ( $\beta$ ) mit  $\gamma = (1 - \beta) * \alpha + \beta$  genauso berechnen wie die kombinierte Farbe von Bild1 (F) und Bild2 (G) mit  $H = (1 - \beta) * F + G$

## 12.5 Zell-projektion

Bei nicht regulären Gittern kann man nicht mit Bresenham oder Schablonen herausfinden, welche Zellen von einem Strahl geschnitten werden.  
Daher bietet es sich an die Zellen durchzugehen, und zu schauen auf welches Pixel sie projiziert werden, statt die Pixel durchzugehen und von dort aus die zugehörigen Zellen zu suchen.

### 12.5.1 Strahlsegmente

zu jeder Zelle schaut man

- auf welche Pixel sie projiziert wird
- wo die Strahlsegmente durchgehen
- welches die Vorder, welches die Rückseite ist.

Für jeden Strahl werden die Parameter (entfernung von Bildebene) auf der Stelle wo er die Begrenzungsflächen schneidet  $t_{in}$  und  $t_{out}$  berechnet, sowie die zugehörigen Skalarwerte auf der Begrenzungsfläche (lineare Interpolation).

Innerhalb der Zelle wird ja bei Tetraedern auch Linear interpoliert  $\rightarrow$  das Strahl-Integral ist dann analytisch lösbar:

$$I_{cell} = \int_{t_{in}..t_{out}} (g(s) T_{cell(s)} ds)$$
$$T_{cell(s)} = \exp(-\int_{t_{in}..s} (\tau(x) dx))$$

### 12.5.2 Kombination der Segmente

**Sortierung der Zellen vor der Projektion** Bei ungeordnetem Gitter: Graph erzeugen

- für jede Zelle ein Knoten
- für jede Fläche zwischen 2 Zellen eine Kante (gerichtet, von Bildebene wegzeigend)
- Begonnen wird immer mit Zelle, die keine eingehende Kante hat.
- Diese Zelle, und alle ausgehenden Kanten werden dann entfernt.
- nun wieder mit **Zelle ohne eingehende Kante** weiter.

Bei **regulärem Gitter** und orthonormaler Projektion  $\rightarrow$  einfach 3fach geschachtelte Schleife

**Sortierung der Strahlsegmente nach der Projektion** Jedes Strahlsegment wird berechnet und dem jeweiligem Pixel zugeordnet.  
Dabei wird geschaut ob es zu einem schon vorhandenem Segment benachbart ist, und in dem Fall mit dem benachbartem Segment kombiniert.

### 12.5.3 Splatting

Hier werden 3-dimensionale Basisfunktionen (Kern h) mit jedem Skalarwert multipliziert und alles addiert. Danach werden die Ergebnisse auf die Bildebene ( $z=0$ ) projiziert.

$contribution_{ijk}(x, y) = f(i, j, k) * \int_{unendl} h(x - i, y - j, z) dz$   
Die Projektion einer Kugelsymmetrischen Basisfunktion auf die Bildebene kann vorberechnet werden, da sie nicht von der Position im Daten abhängt.

$$footprint_{ijk}(x, y) = \int_{unendl} h(x, y, z) dz$$

### 12.5.4 Texturbasiertes Volume Rendern

#### 2D-Texturen

- Hier werden ganz viele Ebenen mit Transparenten Texturen entsprechend der Skalarwerte gefüllt
- Die Ebenen werden von der Grafikkarte übereinandergelegt. zwischen den Skalarwerten in der Textur interpoliert die Grafikkarte bilinear.
- Allgemeine Blickrichtungen lassen sich schlecht realisieren: entweder man schaut von der Seite in die Textur-Ebenen oder man muss die Texturebenen für jede Perspektive gedreht neu berechnen, Dann findet jedoch keine korrekte Interpolation statt, da sich trilinear nicht für beliebige Perspektiven auf bilinear abbilden lässt.

#### mit 3D-Texturen

- hier wird der Skalar-datensatz einfach in eine 3D-Textur mit einem regulärem Gitter umgerechnet.
- Nun kann man interpolierte Werte von der Grafikkarte für eine beliebige Stelle innerhalb der Textur erhalten.
- So ist auch perspektivische Projektion schneller möglich

## 13 Topologie Analyse

### 13.1 Merkmale

#### 13.1.1 Kritische Punkte

- Quellen
- Senken
- Sattelpunkte

#### 13.1.2 Separatrizen

Verbinden kritische Punkte mit einer Linie, zu der die Strömung tangential ist.

#### 13.1.3 Wirbelkerne

(Vortex Cores)

### 13.2 2D Vektorfeld Topologie

Topologie = Zerlegung von B in **zusammenhängende Komponenten** der Schnitten von  $\alpha$  und  $\omega$  Grenzungen.

In jeder Komponente kommen alle Stromlinien also jeweils von der gleichen  $\alpha$  Menge und laufen zur gleichen  $\omega$  Menge.

#### 13.2.1 nach Anfang der Stromlinie

Jedem Punkt P die Startmenge seiner Stromlinie zuweisen

$$\alpha\text{-Grenzmenge}(c_p) = \{q \in B \mid \lim_{t \rightarrow \infty} c_p(t) = q\}$$

bzw. um Zykel mit zu erfassen:

Alle q in B, so dass eine Folge mit Grenzwert minus unendlich ex, in so dass die Stromlinie von P an den Folgeelementen als Parameter ausgewertet gegen q konvergiert.

#### 13.2.2 nach Ende der Stromlinie

$$\omega(c_p) = \{q \in B \mid \lim_{t \rightarrow \infty} c_p(t) = q\}$$

bzw. mit Zykeln:

Alle q, so dass es eine Folge mit Grenzwert unendlich gibt, so dass  $c_p$ (Folgeelementen) gegen q konvergiert.

#### 13.2.3 Kritische Punkte

An Nullstellen des Vektrofeldes gilt  $c_P(t) = P$

Kritische Punkte + Rand = Alpha und omega Grenzungen

**lineares Vektorfeld**  $x \rightarrow Ax + b$ ,

$$A = \begin{pmatrix} a11 & a12 \\ a21 & a22 \end{pmatrix}, b = \begin{pmatrix} b1 \\ b2 \end{pmatrix}$$

Ein kritischer Punkt:  $Ax + b = 0 \Leftrightarrow -A^{-1} * b$ ;  $A = \text{Jacobi-Matrix}$

**Stromlinie:**

$$c_P(t) = p * e^{At} = p * \sum_{k:0.. \infty} \frac{(At)^k}{k!}$$

**Jordansche Normalform** gesucht: Darstellung von  $(A*t)^k$  durch Eigenwerte, um  $e^{\cdot}$  in die Matrix reinziehen zu können.

Günstige Basis für die Eigenwerte, Basistransformation S:

$$A = S^{-1} J S$$

$$(At)^k = S^{-1} * J^k * S * t^k$$

**Drei Fälle für J:**

$$(11,0,12,0) \quad J = \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix}$$

(Beide  $> 0$  QUELLE

größer/kleiner SATTEL

Beide  $< 0$  SENKE

gleich und  $> 0$ : Sternförmige Quelle,  $< 0$ : Sternförmige Senke)

$$(11, 1, 12, 0) \quad J = \begin{pmatrix} \lambda_1 & 1 \\ 0 & \lambda_2 \end{pmatrix}$$

$\lambda > 0$ : asymmetrische Quelle,

$\lambda < 0$ : Senke)

$$\text{komplexe Eigenwerte} \quad J = \begin{pmatrix} \lambda & -\gamma \\ \gamma & \lambda \end{pmatrix} = r * e^{i\phi}$$

Wirbel/Spirale,  $r$  pos = Quelle,  $r$  neg = Senke

$\phi$  entspr. Drehsinn und Stärke der Rotation

Realteil 0  $\rightarrow$  Zentrum mit Zykel

**det A=0** Kritischer Punkt degeneriert zu einer Geraden

### 13.3 Topologie Erkennung

Verhalten um Nullstelle ähnelt linearem Feld (falls  $\det(\text{Ableitung}) \neq 0$ )

- Alle Nullstellen bestimmen (innerhalb Zellen interpolieren)
- Ableitung berechnen (wieder interpolieren)

**q Senke:** es ex. Umgebung U mit  $\lim_{t:\infty} c_P(t) = q$  f. alle  $P \in U$

(U = omega Becken)

**q Quelle:** es ex. Umgebung U mit  $\lim_{t:-\infty} c_P(t) = q$  f. alle  $P \in U$

(U = alpha Becken)

sonst: **q Sattel**

#### 13.3.1 Einfacher Topologie Algorithmus

- Finde alle Nullstellen
- Berechne Ableitung, Eigenwerte d. Nullstellen
- Falls Eigenwerte reel mit versch. Vorz. (**SATTEL**)  $\rightarrow$  Berechne Eigenvektoren
- Berechne zu jedem Eigenvektor 2 Stromlinien (vorwärts/rückwärts)

#### 13.3.2 Exakte Stromlinien im Zellinnern

**Dreieck:** 3 Pkt  $p_i$  mit 3 Vektoren  $v_i \rightarrow$  LGS:  $A * p_i + b = v_i$  lösen ergibt Vektorfeld.

Stromlinien  $c_P(t) = e_1 \phi_1(t) + e_2 \phi_2(t) + p$ ;  $\phi_{1,2}$ : Basisfkt,  $e$  Koeffizient von A abhängig

- A 2 reele Eigenwerte  $\neq 0$ ,  $p$ =kritischer Pkt,  
 $c = e_1 \exp(t * \lambda_1) + e_2 \exp(t * \lambda_2) + p$
- $\lambda_1 = 0$  :  $c = e_1 t + e_2 (\exp(t \lambda_2) - 1) + p$
- $\lambda_{1,2} = 0$  :  $c = e_1 t + e_2 t^2 + p$
- konjugiert komplexe  $\lambda$ :  
 $c = e_1 \exp(t \lambda) \cos(t \gamma) + e_2 \exp(t \lambda) \sin(t \gamma) + c$

#### 13.3.3 Rand

Außflußpunkt  $\rightarrow$  falls Vektorfeld am Rand nach außen zeigt  
Einflußpunkt  $\rightarrow$  falls Vektorfeld am Rand nach innen zeigt

**Vektorfeld parallel zum Rand: genauer untersuchen.**

Topologie (Separatrisen) ändern sich, falls man Loch in Vektorfeld schneidet (weil nicht mehr alle Stromlinien die Kurve kriegen und nun im neuem Rand enden!).

Möglichkeiten: Punkt P auf konvexe Ecke, Punkt P auf Gerade, Punkt P auf konkaver Ecke.

Jeweils:

- Einfluss vor Punkt P, Ausfluss nach Punkt P,
- Einfluss überall,
- Ausfluss Überall
- Parallel vor P, Parallel nach P
- Parallel vor P, Ausfluss nach P
- Parallel vor P, Einfluss nach P
- Parallel IN P, vorher Einfluss, nachher Ausfluss

Dazu jeweils weitere Punkte betrachten, Vektor dort nach Einfluss/Ausfluss klassifizieren.

#### 13.3.4 Zykel

$c_P$  hat Zykel falls  $\delta! = 0$  existiert mit:  $c_P(t) = c_P(\delta + t)$  f. alle t.

**Satz:**

- im innerem eines Zyklus ex. kritischer Pkt.
- kompakte Grenzmenge eines Vektorfeldes ohne Kritischen Pkt. = Zykel

#### Erkennung

- Stromlinie passiert eine Zelle zweimal.
- Alle Zellen von 1. bis 2. Durchlauf = ZELLYZKEL
- Stromlinie kann nur noch in eine Richtung aus dem Zykel raus (innen oder außen)
- Im Zellzykel an der innen oder außenseite nach **Ausgängen** (Punkte mit Ausfluss,  $v(\text{Punkt})$  zeigt aus dem Zellzykel) suchen und **rückwärts integrieren**.
- Falls beim rückwärtsintegrieren ein Umlauf geschafft wird, kann die Stromlinie dort rauslaufen.
- Falls **kein Umlauf von irgendeinem Ausgang** geschafft wird, ist es ein **Zykel**

**genaue Lage ermitteln (bisektion)** Kante im Zellzykel suchen und:

- Stromlinie starten, bis sie wieder zu der Kante kommt.
- Neuer Startpunkt in der Mitte von dem Teil der Kante, in den die Stromlinie sich verlagert hat.

**alle Zykel finden** Topologischer Graph: Kritische Punkte, Kanten und Zykel = ECKEN

Separatrisen als KANTEN

Fehlender Zykel teilt diesen Graph in 2 Hälften.

#### 13.3.5 strukturell stabil

Vektorfeld strukturell stabil (ändert Topologie unter minimalen Veränderungen nicht), falls:

- Endl. Anzahl kritische Pkte und Zykel
- Sattel durch Stromlinien verbunden
- Grenzmengen = kritische Pkte / Zykel

### 13.4 Zeitabhängige Topologie

- Vektorfeld gegeben als n Vektorattribute zwischen denen über die Zeit interpoliert wird
- Man betrachtet Veränderung der statischen Topologie über die Zeit
-

### 13.4.1 Bifurkationen

#### Lokale Bifurkationen

- Wirbel wird zu Zykel
- Sattel und Senke vereinigen sich und lösen sich auf
- kritische Punkte laufen auf Bahnen. Bis sie sich gegenseitig auslöschen, oder am Rand verschwinden

#### Globale Bifurkationen

- Betreffen die Verknüpfung der kritischen Punkte durch die Separatrizen
- 2 Separatrizen stoßen aneinander und tauschen ihre Enden über die Zeit
- 

### 13.4.2 Verfahren

- einzelne linear interpolierte Dreiecke von Zeitschritt zu Zeitschritt analysieren. (in einem Dreieck: max. 1 kritischer Pkt. → Entdeckung paarweiser Auslöschung oder Entstehung möglich)
- Man kann die Separatrizen über die Zeit alle berechnen und durch eine Fläche miteinander verbinden.

## 13.5 Tensorfeldtopologie

- Betrachtung der durch die Eigenvektoren erzeugten Vektorfelder (je nach Dimension 3 Vektorfelder)
- Eigenvektoren nicht orientierbar
- **Kritische Pkte:** hier liegen mehrfache Eigenwerte vor → zwei Eigenvektorfelder tauschen ihre Rolle

### 3 Typen von kritischen Pkt:

- Trisektorpunkt (3 Tensorlinien laufen auf Punkt zu und knicken wieder weg)
- Spaltpunkt 1 (1 Tensorlinien läuft rund um die Kurve)
- Spaltpunkt 2 (1 Tensorlinien läuft im Spitze Kante)

## 14 Feature Detektion

- Merkmale /Eigenschaften der Attribute eines Datensatz
- Teilmenge des Datensatz das die Attribute erfüllt = Feature zum Prädikat B
- Wirbelkerne (**Kurvenmerkmal**)
- Schockwellen (**meist 3-D Flächenmerkmal**)
- Extrema und Sattelpunkte in Skalarfeldern (**Punktmerkmal**)
- Separation and Attachment Lines

### 14.1 Wirbelkerne

Klassifikation nach

- Wirbelstärke (**Betrag der Rotation**  $\delta x v$ )
- Helizität (**Rotation zusätzlich skalar mit Vektorfeld multiplizieren**, da im Wirbelkern Rotation parallel zur Geschwindigkeit)
- Normierte Helizität
- Minimaler Druck innerhalb der Rotationsebene

Wirbelkern ist oft eine Kurve um die der Wirbel sich dreht. (Oder wird zu Kurve in zeitabhängigem Vektorfeld weil Wirbel sich verschiebt.)

#### 14.1.1 Banks/Singer

Verfolgung von Wirbelkernen

- Bestimme Rot. im Punkt  $p_i$
- Gehe Stück in Richtung der Rotation nach  $p'_i$
- Punkt mit minimalen Druck in der Rotationsebene von  $p'_i$  = kern

#### 14.1.2 Sujudi/Haimes

Tetraederzellen

- Ableitung berechnen
- Komplex Konjugierte Eigenwerte der Ableitung → Rotation
- **Reele Eigenrichtung bestimmen.**
- **Anteil in dieser Richtung von den Eck-Vektorfeld-Vektoren abziehen** (bleibt also nur noch der Teil übrig der senkrecht zum reellen Eigenvektor-Anteil ist)
- **Nullstellen des veränderten Feld = Kern**

### 14.1.3 Methoden basieren auf parallelität 2er Vektorfelder

$$v \parallel w = \{x | v(x) \times w(x) = 0\}$$

- banks/singer:  $\delta x v \parallel \delta p$  (rotation — ableitung des druck)
- sujudi/haimes: Vektorfeld — Beschleunigung durch Vektorfeld