

Visualisierung

Vorlesung im Wintersemester 2016/17

**4) Skalardaten und Volumenvisualisierung
(Teil 2)**

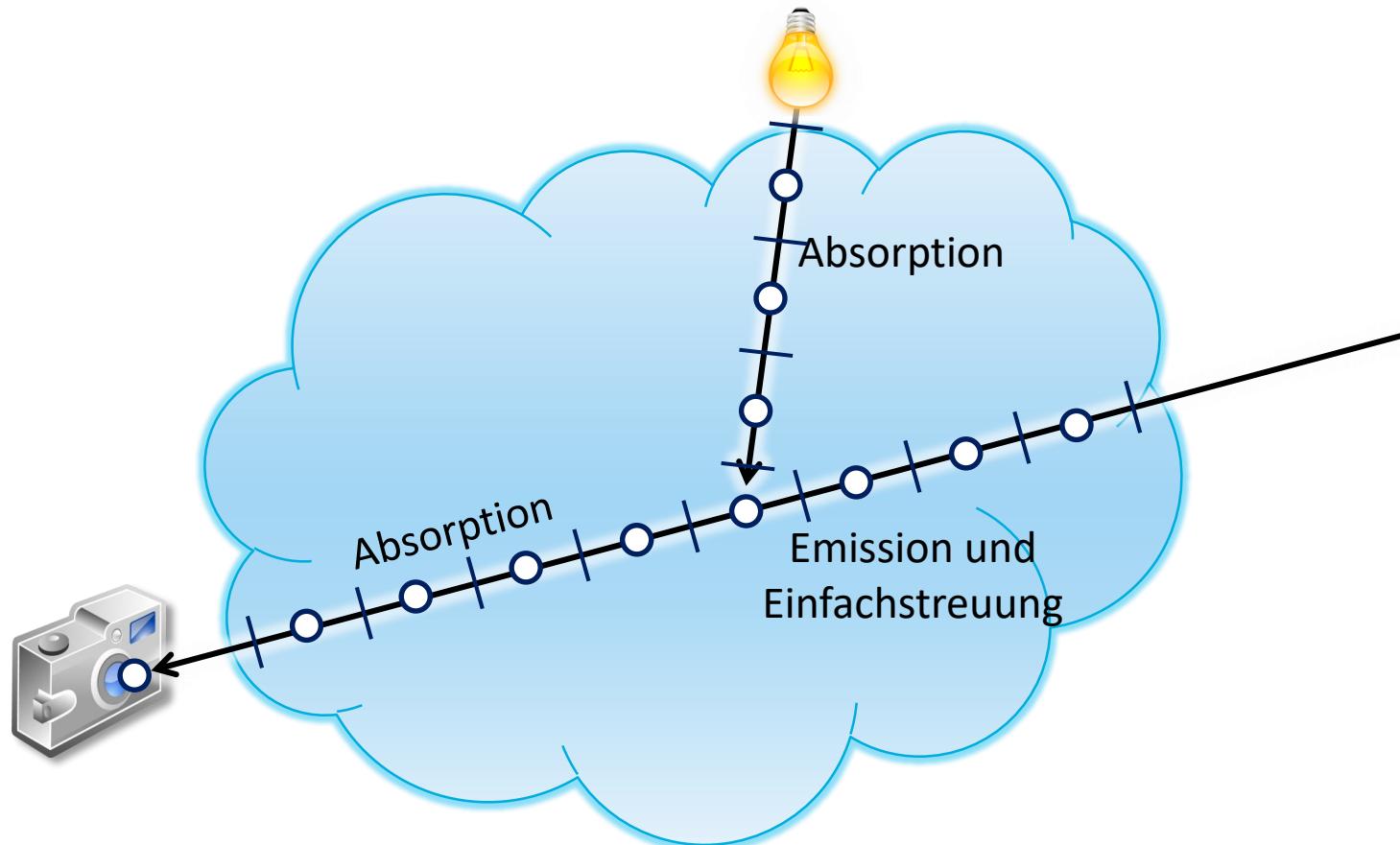
Prof. Dr.-Ing. Carsten Dachsbacher
Lehrstuhl für Computergrafik
Karlsruher Institut für Technologie



Volumen Rendering Gleichung

Diskretisierung

- wir betrachten das Volumen an n Stellen entlang eines Sichtstrahls
- Vorstellung: n Intervalle mit konstanter Absorption und Emission
- wie berechnet man damit $L(s)$?



Diskretisierung der Volumen Rendering Gleichung

- betrachte Emission und Absorption im Intervall $[s_{k-1}; s_k]$
(s_k ist näher an der Kamera als s_{k-1} wie vorhin auf der Folie)

$$L(s_k) = L(s_{k-1})e^{-\tau(s_{k-1}, s_k)} + \int_{s_{k-1}}^{s_k} L_e(s)e^{-\tau(s, s_k)} ds$$

- Annahme: konstante Absorption, keine Absorption des emittierten Lichts

► Transmittance $\theta_k = e^{-\tau(s_{k-1}, s_k)} \approx e^{-\sigma_t(s_k)\Delta s}$

► Emission $b_k = \int_{s_{k-1}}^{s_k} L_e(s)e^{-\tau(s, s_k)} ds \approx L_e(s_k)\Delta s$

$$\Rightarrow L(s_n) = L(s_{n-1})\theta_n + b_n$$

$$= (L(s_{n-2})\theta_{n-1} + b_{n-1})\theta_n + b_n = \dots = \sum_{k=0}^n \left(b_k \prod_{j=k+1}^n \theta_j \right)$$

- mit Opazität $\alpha_n = 1 - \theta_n$

$$L(s_n) = L(s_{n-1}) \cdot (1 - \alpha_n) + b_n$$

Compositing

- ▶ bedeutet hier: iterative Berechnung des diskretisierten Volumenintegrals
- ▶ wir können das Volumen traversieren von
 - ▶ vorne nach hinten (i.d.R. bei Raycasting)
 - ▶ hinten nach vorne (i.d.R. bei texturbasiertem Volumenrendering)

Back-to-Front Compositing

- ▶ ergibt sich direkt aus dem diskretisierten Integral:

$$L(s_n) = L(s_{n-1}) \cdot (1 - \alpha_n) + b_n$$

- ▶ Emission $L(s_k)$ und Opazität α_k aus der Transferfunktion
- ▶ $L(s_k)$ beinhaltet ggf. Beleuchtungsberechnung
- ▶ Achtung: b_n und α_n sind abhängig vom Slice-Abstand Δs

Front-to-Back Compositing

- wir müssen die Reihenfolge der Summation umdrehen
- bisher:

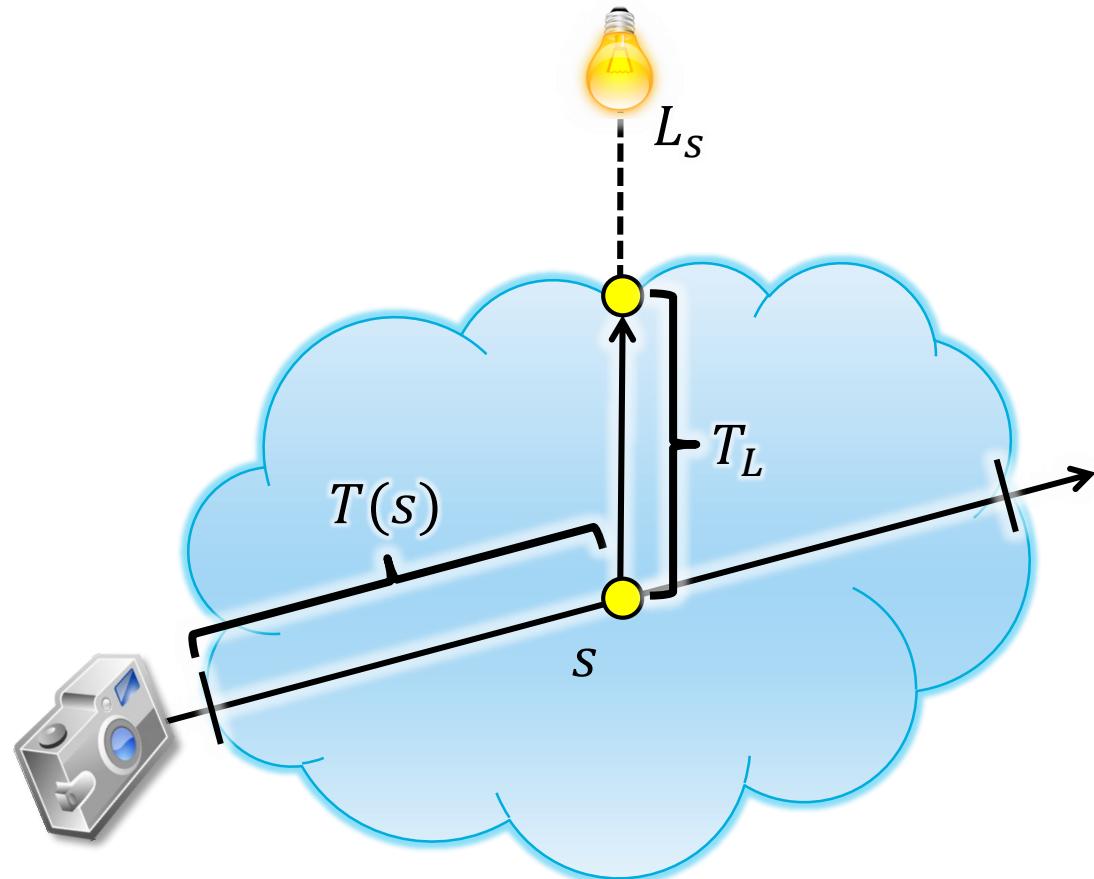
$$\begin{aligned} L(s_n) &= \sum_{k=0}^n \left(b_k \prod_{j=k+1}^n \theta_j \right) \\ &= b_0 \prod_{j=1}^n \theta_j + b_1 \prod_{j=2}^n \theta_j + \dots + b_{n-1} \prod_{j=n}^n \theta_j + b_n \end{aligned}$$

- Reihenfolge in der wir das Volumen traversieren: $b_n, b_{n-1}, \dots, b_1, b_0$
- wir benötigen als Gewichte die akkumulierten Transparenzwerte:
 $1, \theta_n, \theta_n \theta_{n-1}, \theta_n \theta_{n-1} \theta_{n-2}, \dots$
bzw. mit Opazitätswerten $(1 - 0), (1 - \alpha_n), (1 - \alpha_n)(1 - \alpha_{n-1}), \dots$
- können beim Raymarching leicht mitgeführt werden

Emissions-Absorptions-Modell

Beleuchtung und Schatten

- ▶ Beleuchtung kann Verschattung berücksichtigen
 - ▶ berechne dazu die Transmittance T_L entlang des Schattenstrahls ebenfalls mit Raymarching



Beleuchtung und Schatten

- ▶ aus dem Rendering bekannte Techniken hielten ebenfalls Einzug in die Volumenvisualisierung, z.B. (Approximationen von) Ambient Occlusion um die Wahrnehmung räumlicher Strukturen zu unterstützen
- ▶ Beispiele aus aktueller Forschung am Ende des Abschnitts!



(a) 3.7 FPS



(b) 1.6 FPS



(c) 48.3 FPS



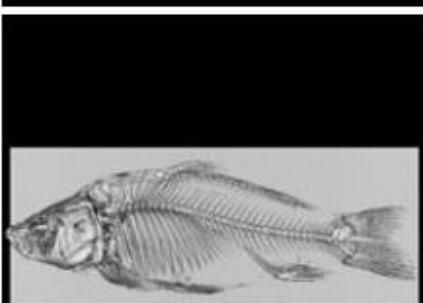
(d) 8.8 FPS



(e) 13.3 FPS



(f) 16.0 FPS

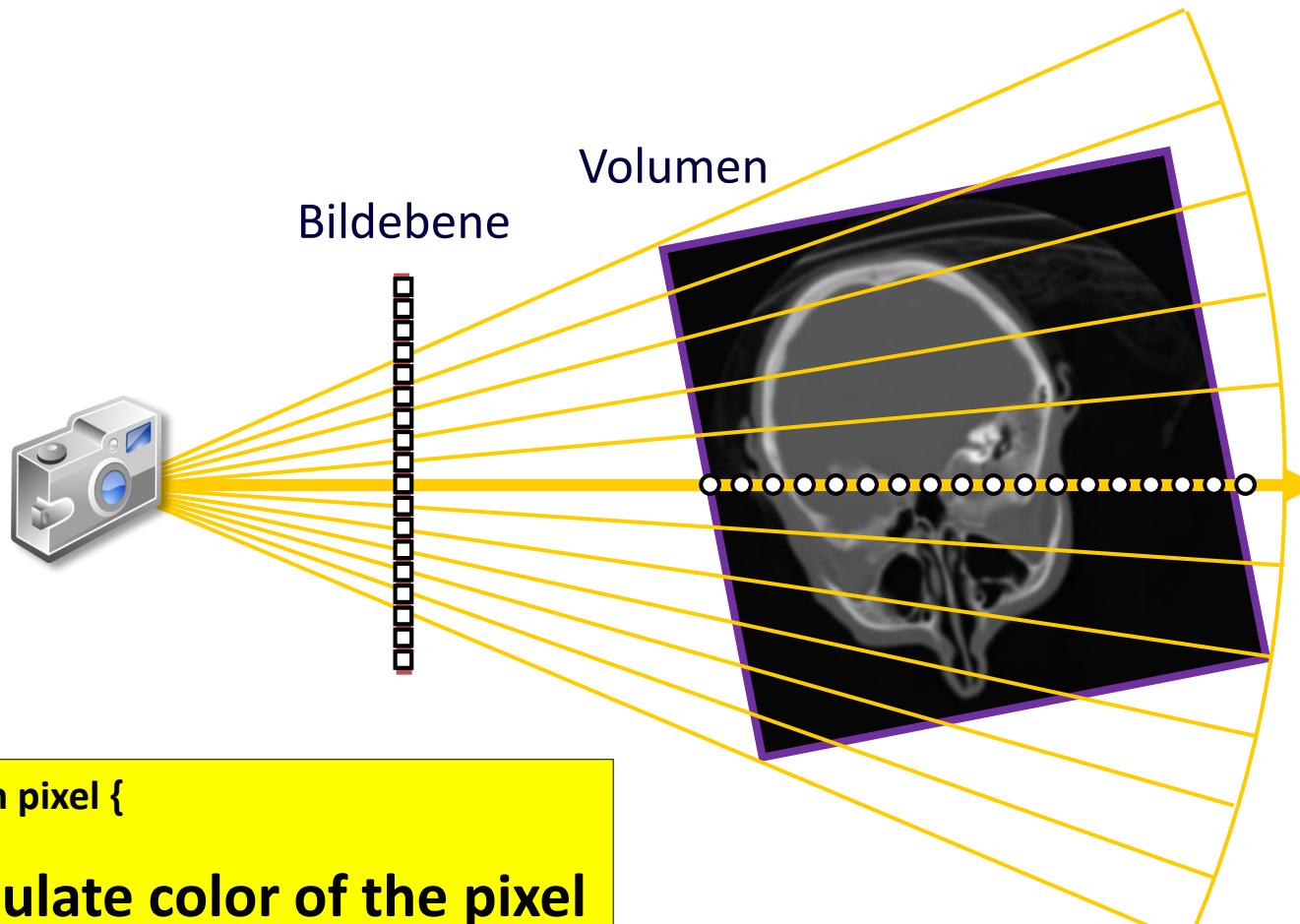


(g) 59.0 FPS



(h) 29.1 FPS

Wann genau wird die Transferfunktion eigentlich ausgewertet?



Transferfunktionen

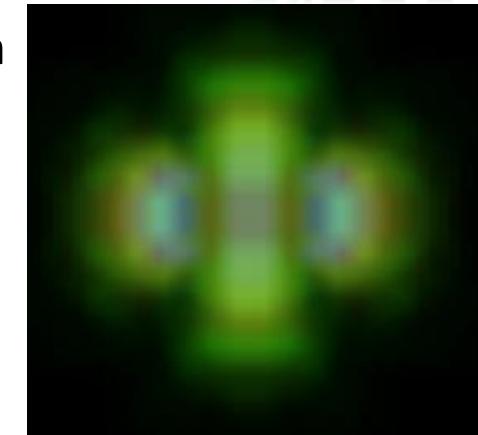
Pre- und Post-Shading / Prä- und Post-Klassifikation

- ▶ Pre-Shading: zuerst Klassifizieren, dann Interpolieren
(auch: präinterpolative Transferfunktion)
 - ▶ weise den Skalar-/Funktionswerten die Emission/Absorption zu
 - ▶ interpoliere dann zwischen den Emissions/Absorptionswerten
- ▶ Post-Shading: zuerst Interpolieren, dann Klassifizieren
(auch: postinterpolative Transferfunktion)
 - ▶ interpoliere die Skalarwerte
 - ▶ wende TF auf die interpolierten Werten an,
um Emission/Absorption zu bestimmen
- ▶ Interpolation erfolgt in beiden Fällen meist trilinear
 - ▶ manchmal mit höherer Ordnung (trikubisch), allerdings teuer

Transferfunktionen

Pre- und Post-Shading/Klassifikation

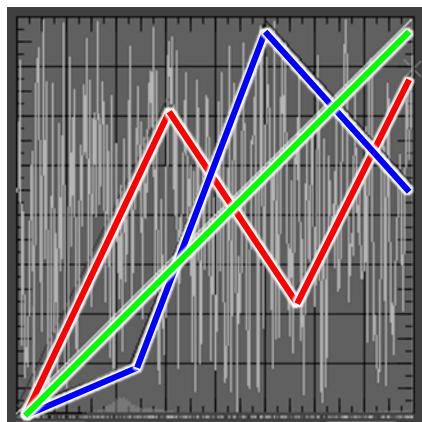
Präklassifikation



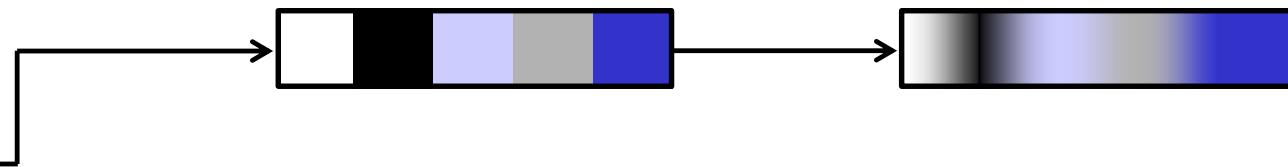
Transferfunktionen

Klassifikation

Interpolation



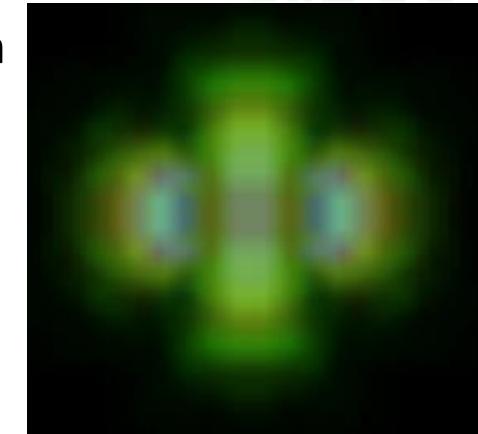
Voxel



Transferfunktionen

Pre- und Post-Shading/Klassifikation

Präklassifikation



Transferfunktionen

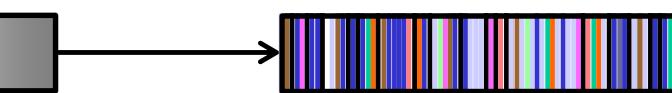
Klassifikation

Interpolation

Voxel

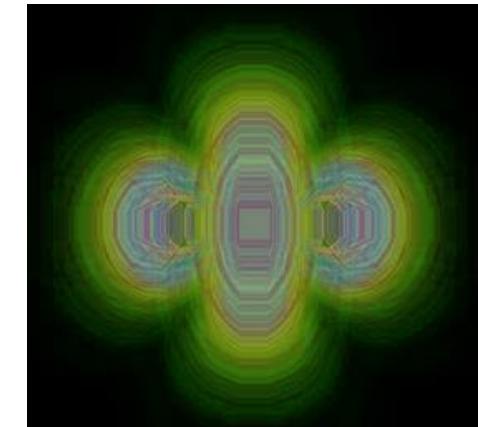
Interpolation

Klassifikation



Wichtig: Transferfunktionen können hochfrequente Details erzeugen!

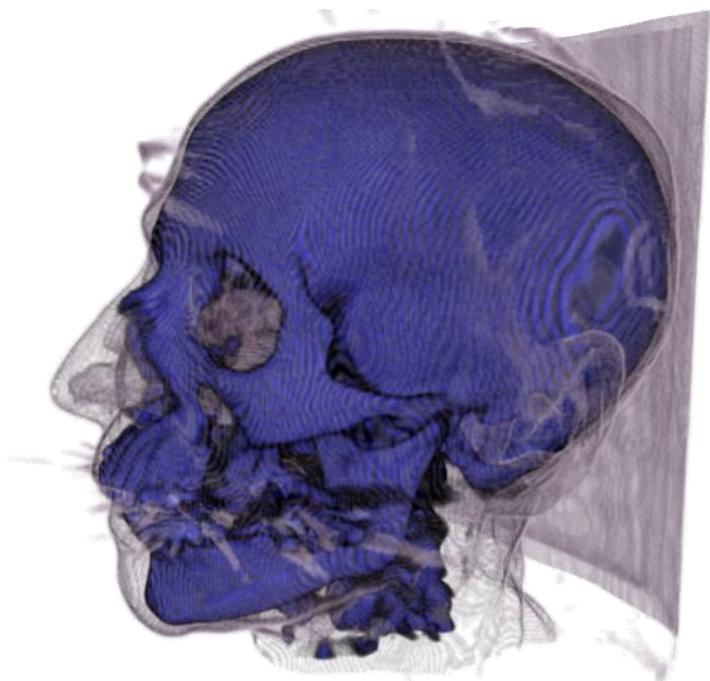
Postklassifikation



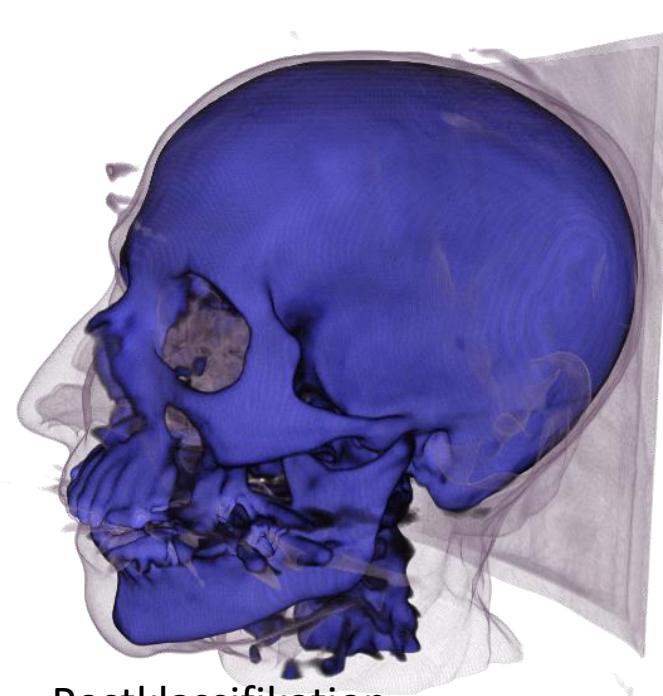
Transferfunktionen

Prä- vs. Postklassifikation: Vergleich der Darstellungsqualität

- ▶ gleiche Transferfunktion, gleiche Auflösung und Schrittweite
- ▶ Extraktion einer Iso-Fläche = hohe Frequenzanteile



Präklassifikation

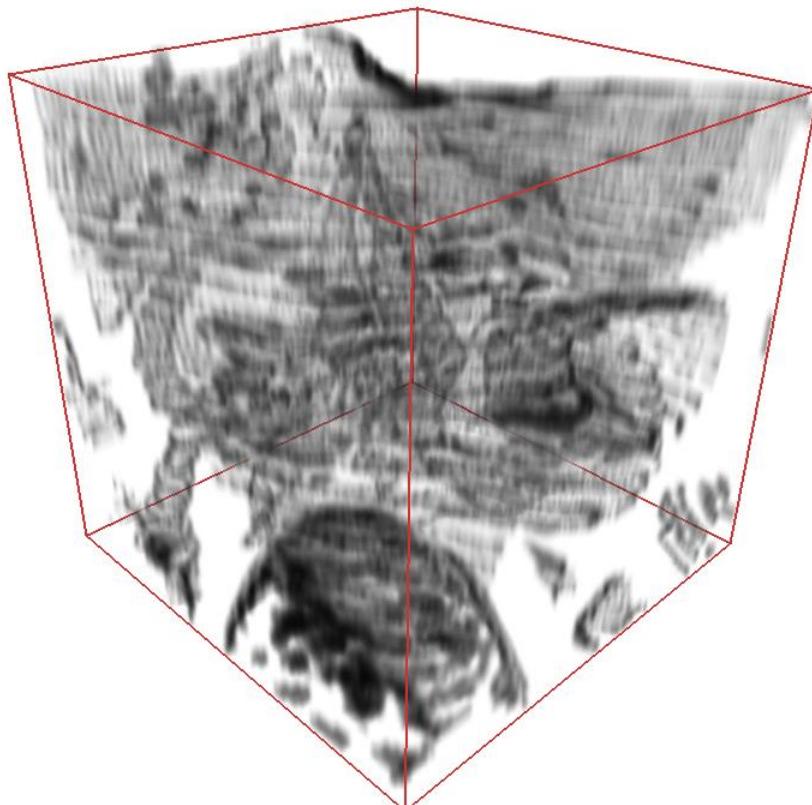


Postklassifikation

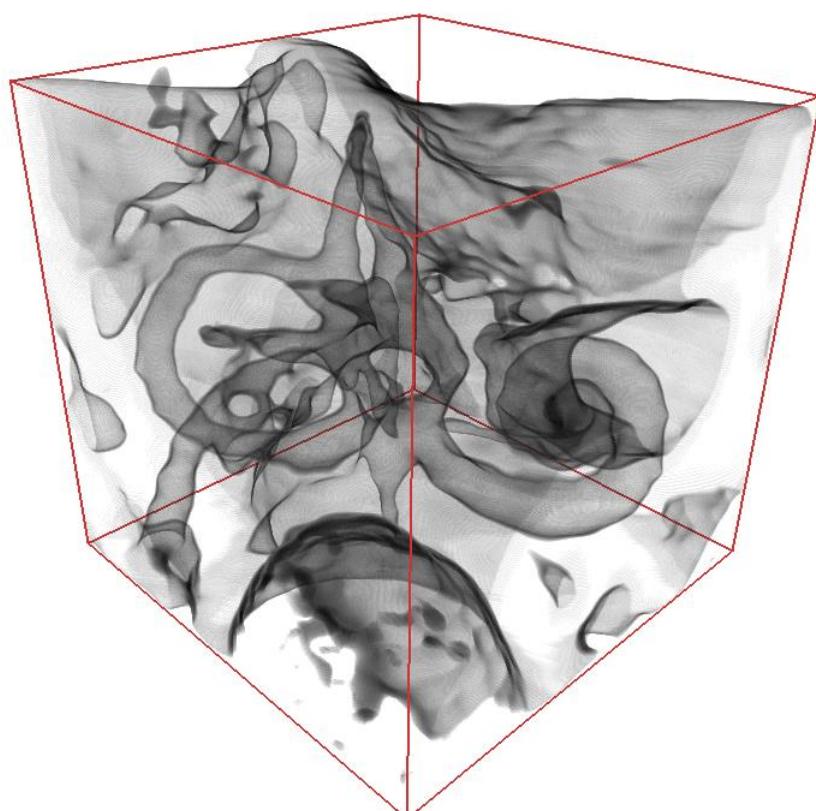
Transferfunktionen

Prä- vs. Postklassifikation: Vergleich der Darstellungsqualität

- ▶ gleiche Transferfunktion, gleiche Auflösung und Schrittweite
- ▶ Extraktion einer Iso-Fläche = hohe Frequenzanteile

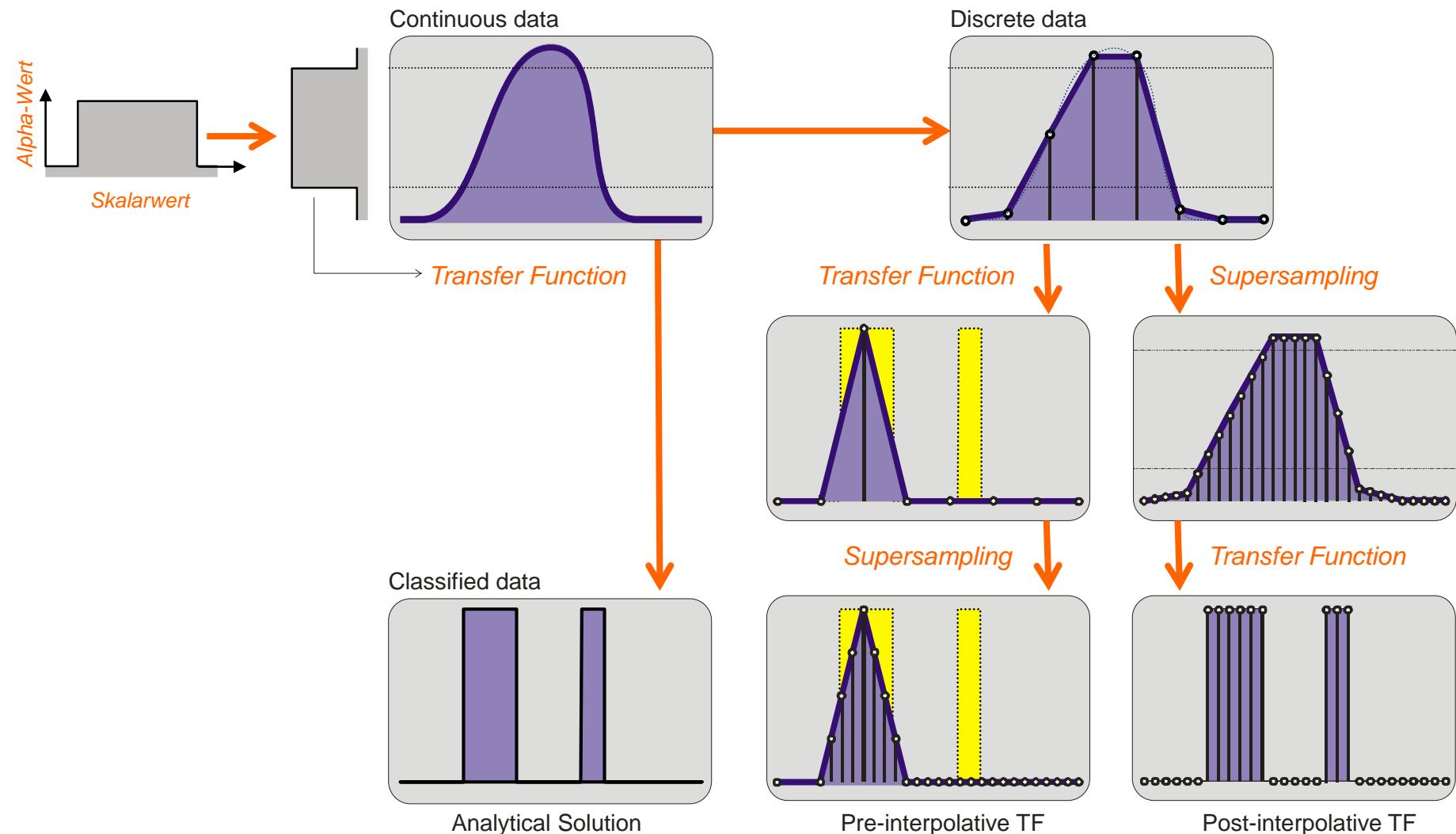


Präklassifikation



Postklassifikation

Prä- vs. Post-Klassifikation: Analyse



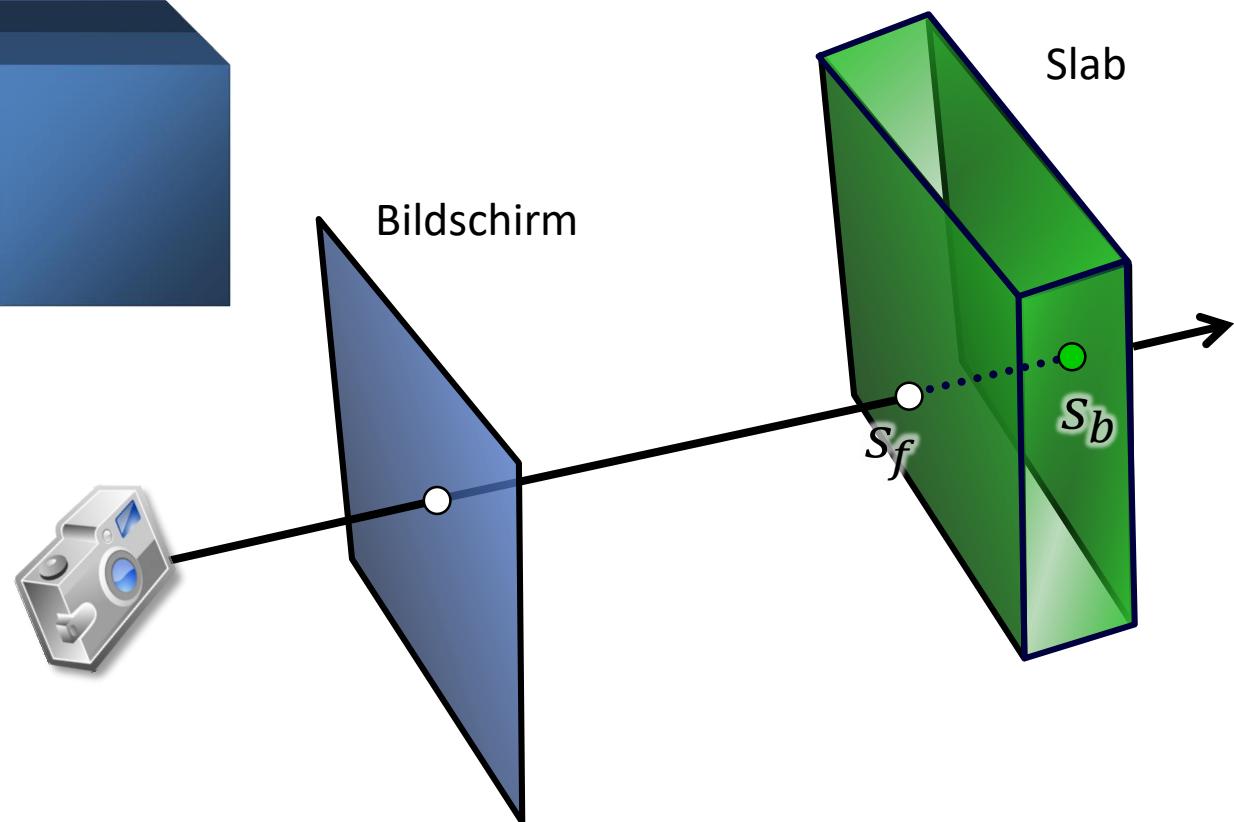
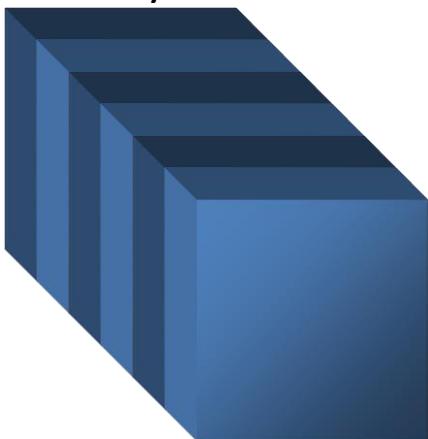
Klassifikation mit Präintegration

- ▶ Postklassifikation: Klassifizieren an einzelnen abgetasteten Stellen
- ▶ aber: kann und muss man berücksichtigen, wie sich die Werte dazwischen verhalten und was die TF daraus macht?

Slice-by-slice

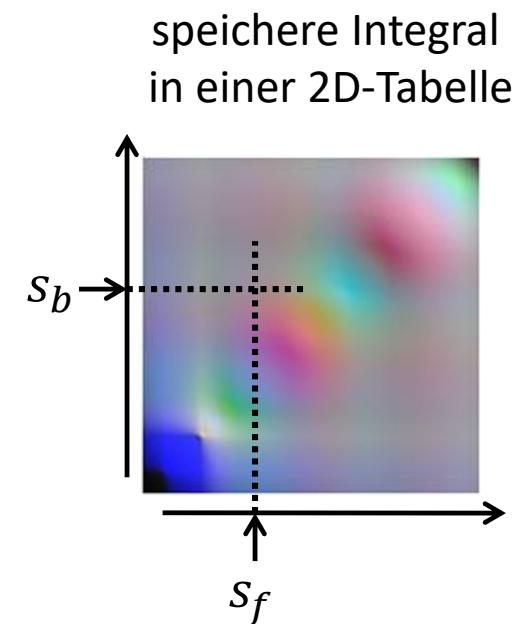
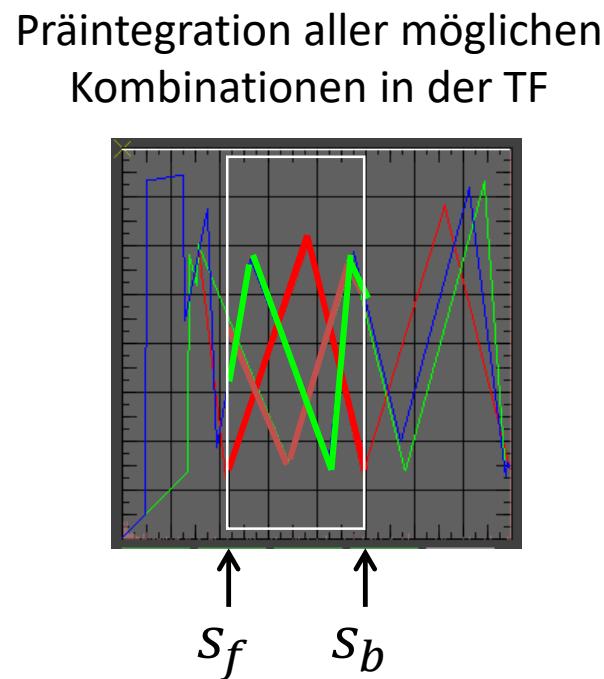
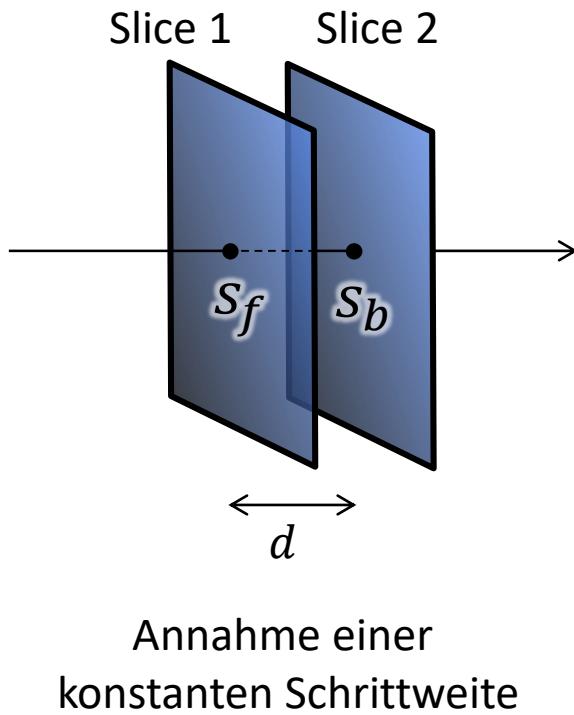


Slab-by-slab



Klassifikation mit Präintegration

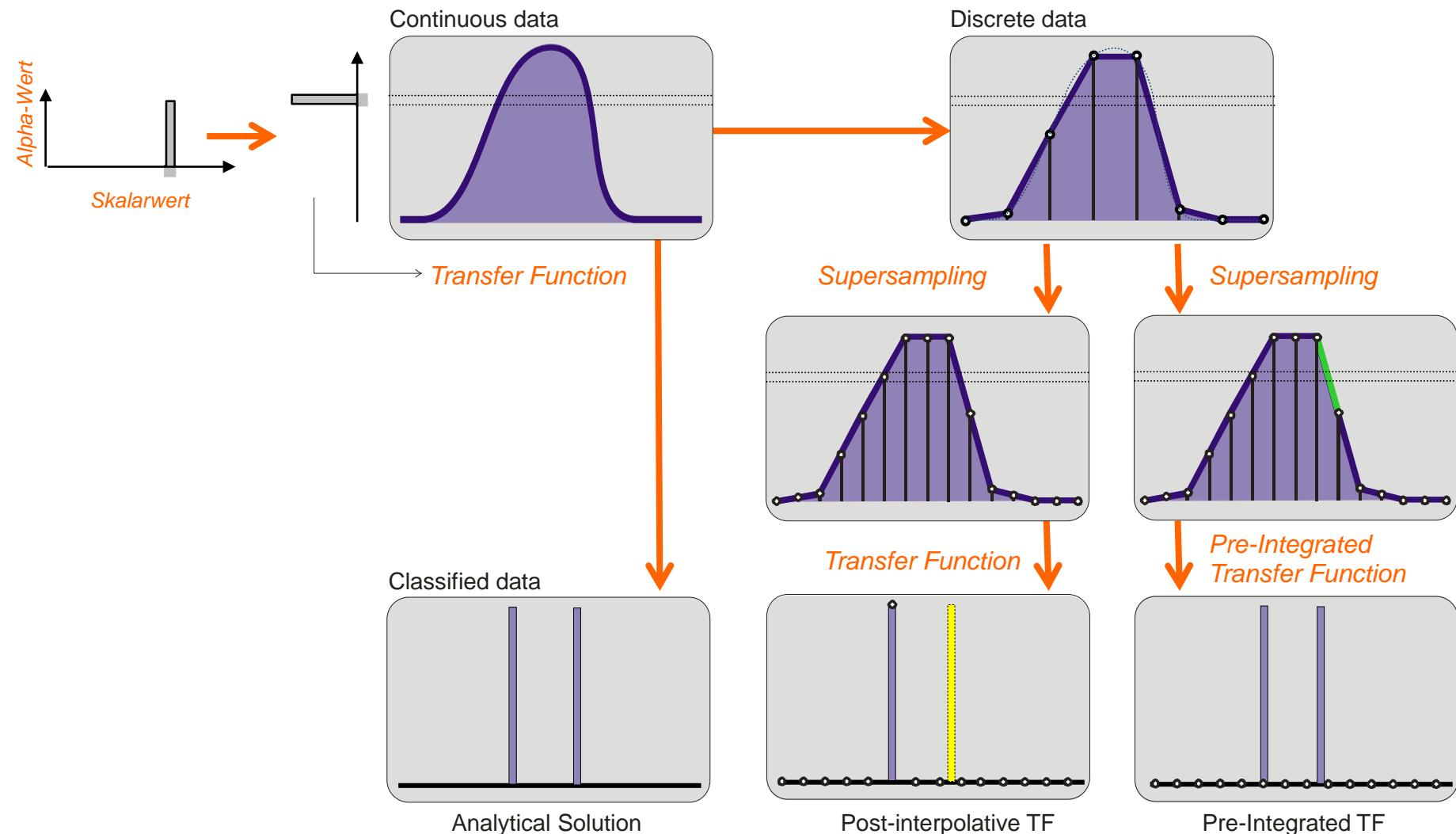
- ▶ Präintegration = wir berechnen den Beitrag eines Intervalls vor, mit...
 - ▶ mit Skalarwerten s_f und s_b an den Enden, und Länge d
 - ▶ speichern gesamte Emission und Absorption eines solchen Strahlsegments in einer Tabelle (Details nächste Folie)



Klassifikation mit Präintegration

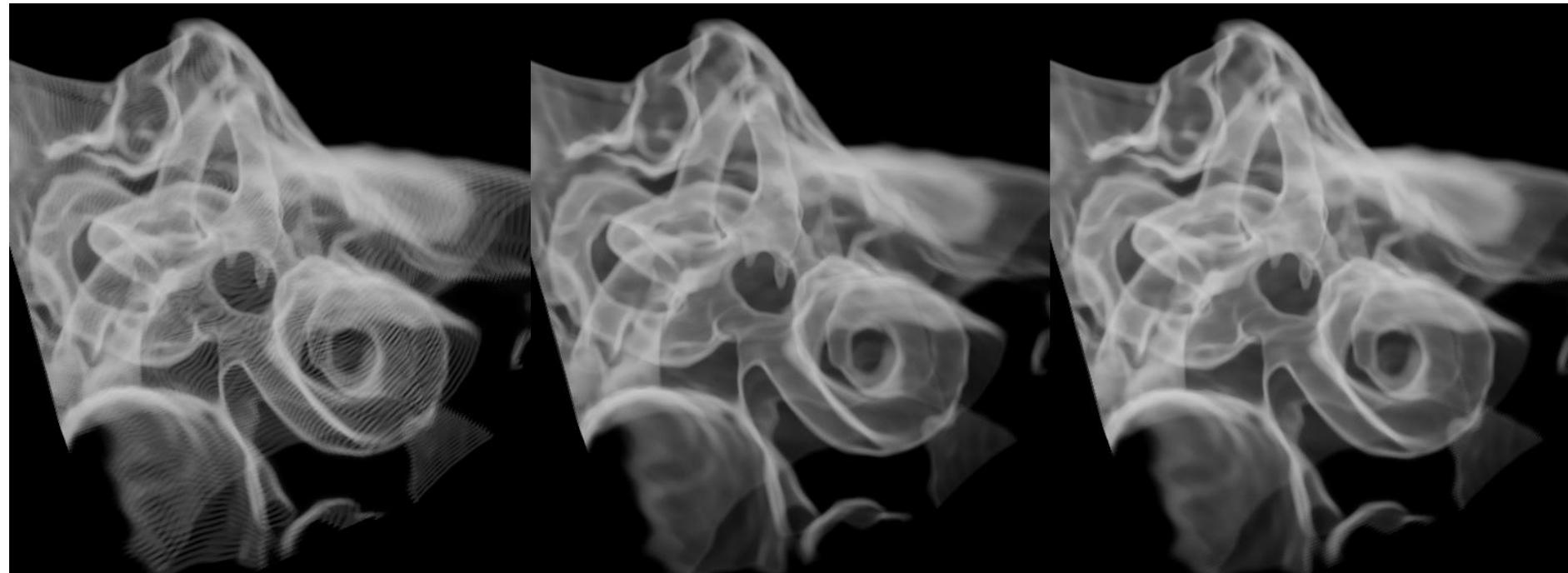
- ... dabei trifft man Annahmen:
 - ▶ lineare Interpolation der Skalarwerte zwischen s_f und s_b
 - ▶ konstante Dicke d der Slabs (sonst keine 2D-Tabelle)
 - ▶ zunächst nur 1D-Transferfunktionen
 - ▶ nur eine Approximation, aber i.d.R. sehr gute Resultate
- Vorberechnung aller möglichen Beiträge eines Slabs
 - ▶ Skalarwerte innerhalb des Slabs
$$s(t) = s_f + t(s_b - s_f), \text{ mit } t \in [0; 1]$$
 - ▶ wende Transferfunktion an und berechne mit hoher Genauigkeit
$$\theta = e^{-\int_0^d \sigma_a(t) dt}$$
$$b = \int_0^d L(t) e^{-\int_t^d \sigma_a(s) ds} dt$$
 - ▶ speichere vorintegrierte Emission b (RGB) und Alpha-Wert ($= 1 - \theta$)

Postklassifikation vs. Präintegration



Klassifikation mit Präintegration

Beispiele



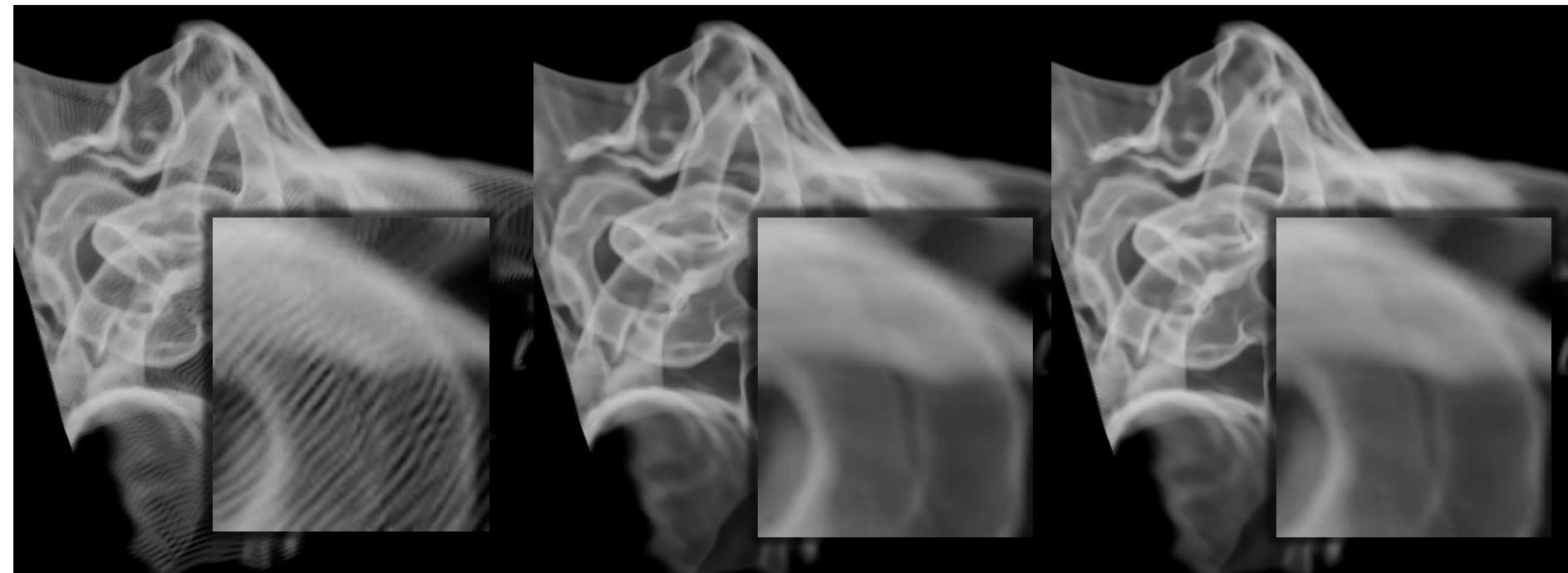
128 Slices
(Postklassifikation)

284 Slices
(Postklassifikation)

128 Slabs
(Präintegration)

Klassifikation mit Präintegration

Beispiele

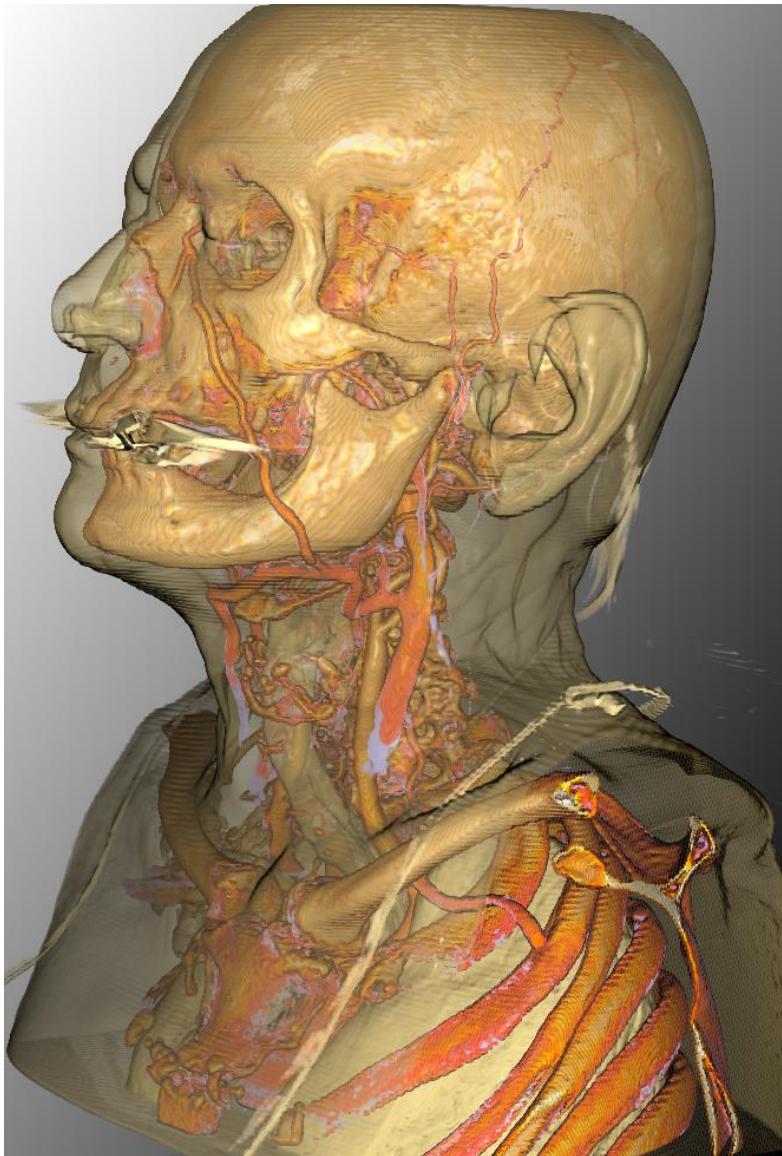
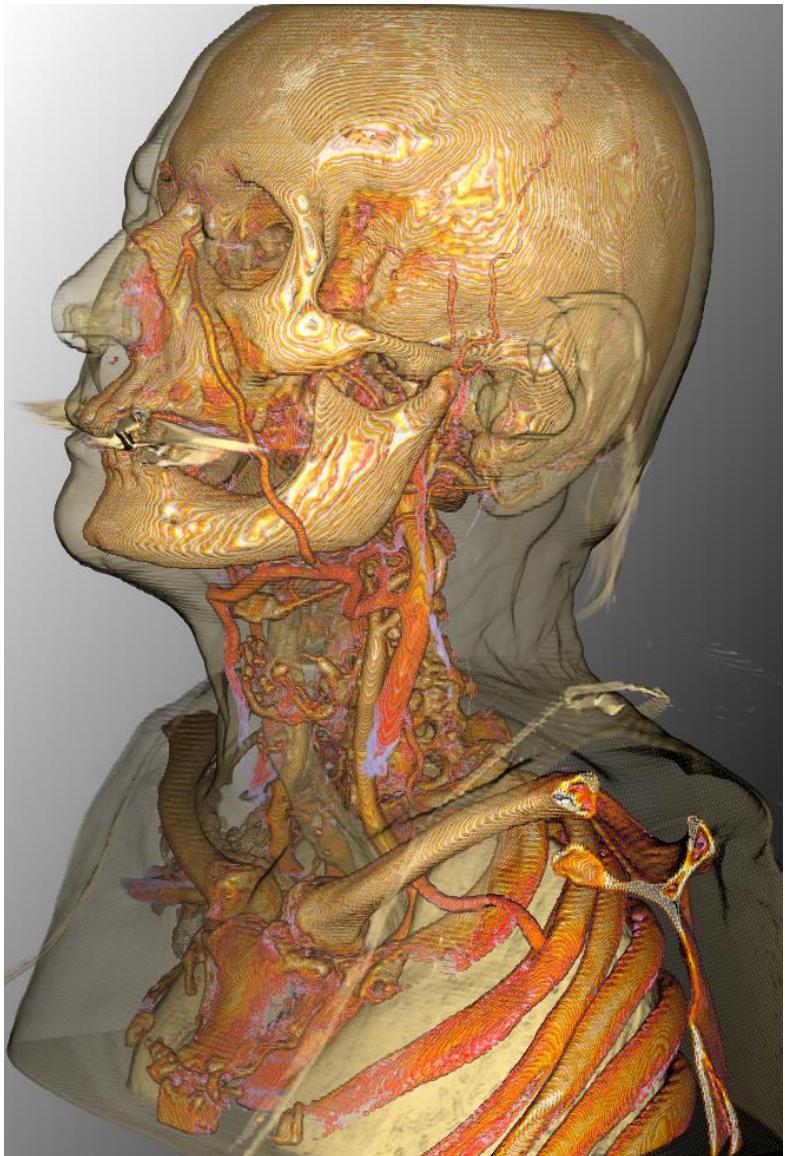


128 Slices
(Postklassifikation)

284 Slices
(Postklassifikation)

128 Slabs
(Präintegration)

Klassifikation mit Präintegration



Präklassifikation

- ▶ wenn möglich vermeiden (nur auf alter Grafik-Hardware verwendet)
- ▶ notfalls für Visualisierung einfacher segmentierter Volumendaten

Postklassifikation

- ▶ bei „glatten“ Transferfunktionen
- ▶ gute Qualität und hohe Performance

Präintegration

- ▶ bei Transferfunktionen mit hohen Frequenzen
- ▶ beste Qualität
- ▶ Änderung der Transferfunktion erfordert Neuberechnung der Tabelle
- ▶ Erweiterung für Multidimensionale-Transferfunktionen möglich

▶ Pre-Integrated Volume Rendering for Multi-Dimensional Transfer Functions
<http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.219.6056>

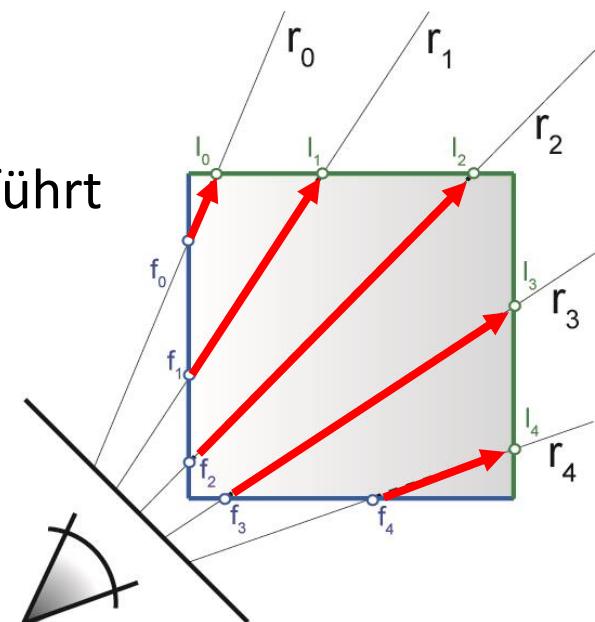
DVR mit Raycasting

Implementation, Optimierung etc.

- im Folgenden kurz einige nennenswerte technische Details bei Raycasting bzw. Raymarching Verfahren

Option 1: Rendering mit einem Fragment Programm

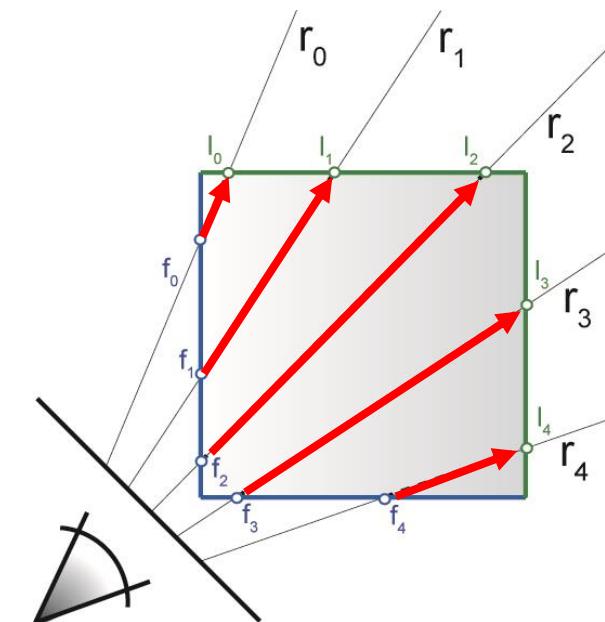
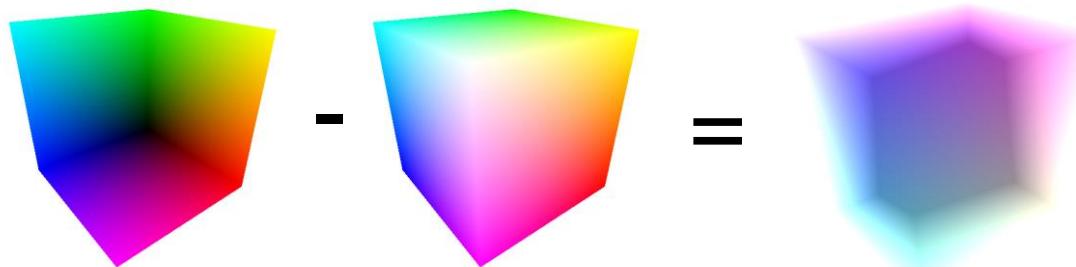
- keine Beiträge außerhalb des Volumens → berechne Ein- und Austrittspunkte der Sichtstrahlen in das/aus dem Volumen
- Vorgehen:
 - zeichne ein bildschirmfüllendes Rechteck
 - jeder Pixel entspricht einem Sichtstrahl
 - pro Pixel wird ein Fragment Programm ausgeführt
 - berechne Sichtstrahlen und Schnitte
 - „marschiere“ in kleinen Schritten entlang des Strahls



DVR mit Raycasting

"Image-Based" Ray Setup/Termination

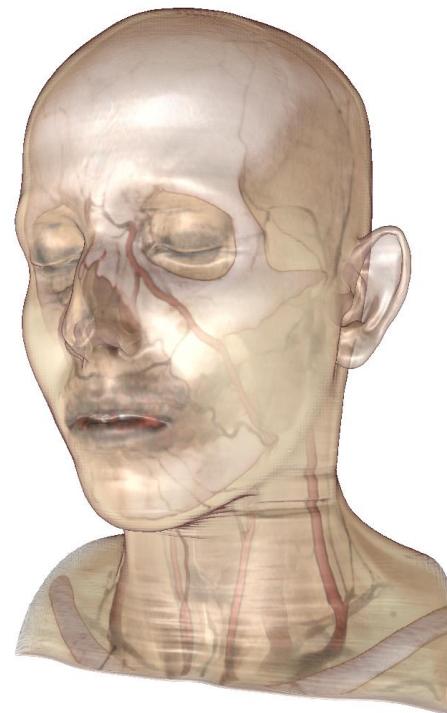
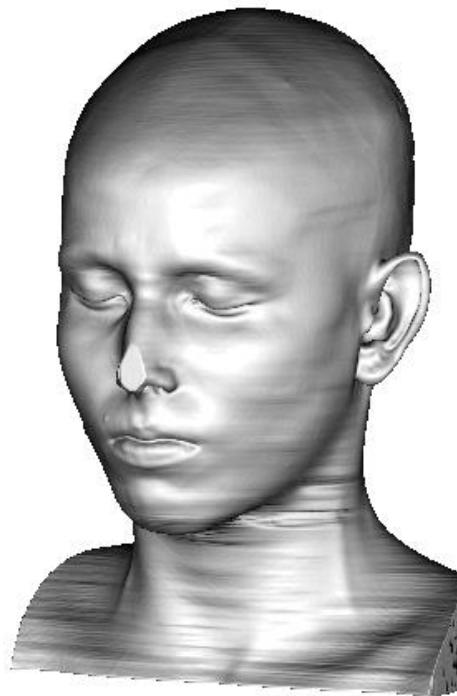
- ▶ zeichne Vorder- und Rückseiten der Bounding Box [Krüger und Westermann, 2003]
 - ▶ Position auf der Oberfläche der Bounding Box wird als Farbe kodiert und in einem Rendertarget (= Textur) gespeichert
 - ▶ für jeden Pixel:
 - ▶ Eintrittspunkt = Position der sichtbaren Vorderseite
 - ▶ Richtungsvektor des Strahls: "Rückseite – Vorderseite"
- ▶ Vorteil: einfach, unabhängig von der Projektion (orthogonal, perspektivisch, ...)



Raycasting: Optimierungen

Early Ray Termination

- ▶ wenn wir Isoflächen extrahieren (z.B. über $|f(x, y, z) - c| < \epsilon$) kann das Raycasting beendet werden, sobald eine Isofläche getroffen wird
- ▶ oder wenn die Opazität (akkumuliert entlang des Strahls bei Front-to-Back) eine festgelegte Schwelle überschreitet



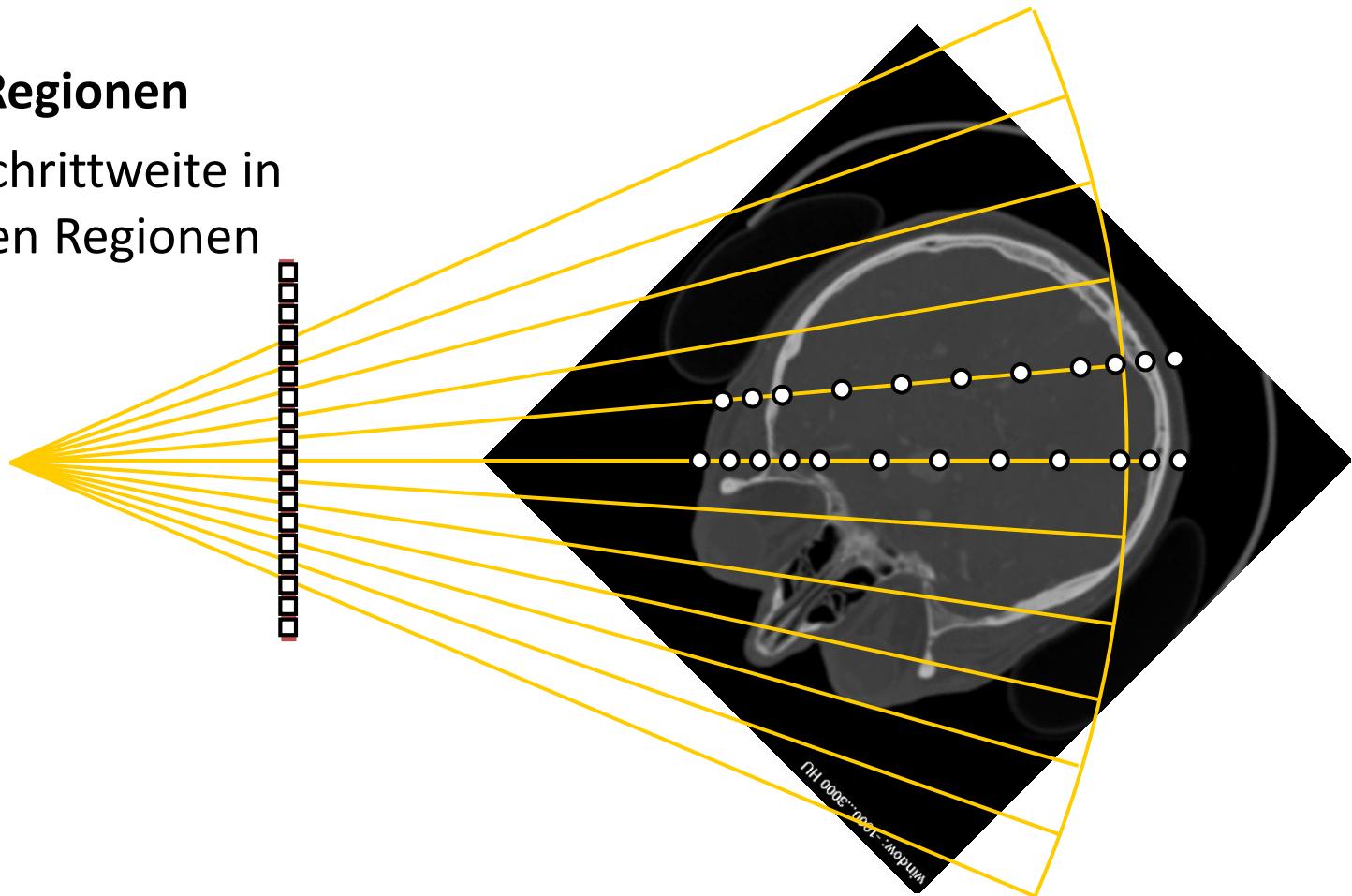
Raycasting: Optimierungen

Empty Space Skipping / Space Leaping

- Ziel: überspringe transparente Bereiche und beginne Raycasting nahe am ersten nicht-transparenten Voxel
 - diese hängen natürlich von der Transferfunktion ab

Homogene Regionen

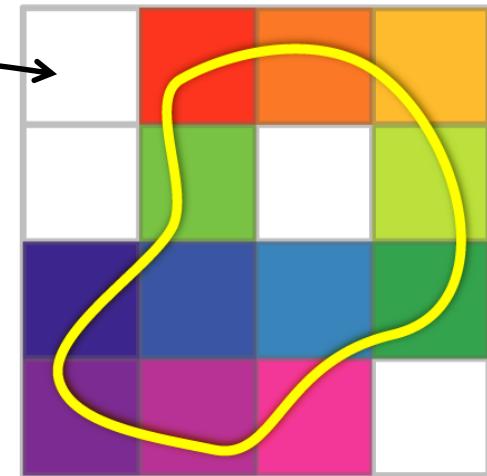
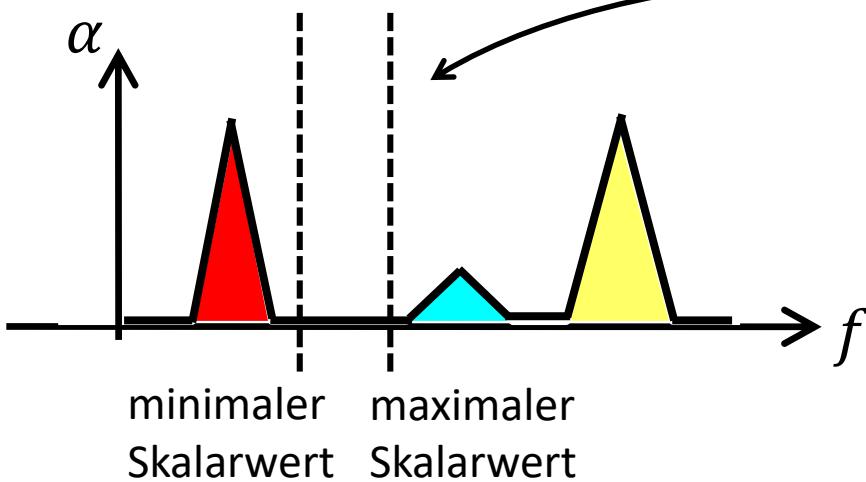
- größere Schrittweite in homogenen Regionen



Ray-Casting: Optimierungen

Image-Order Empty Space Skipping / Space Leaping

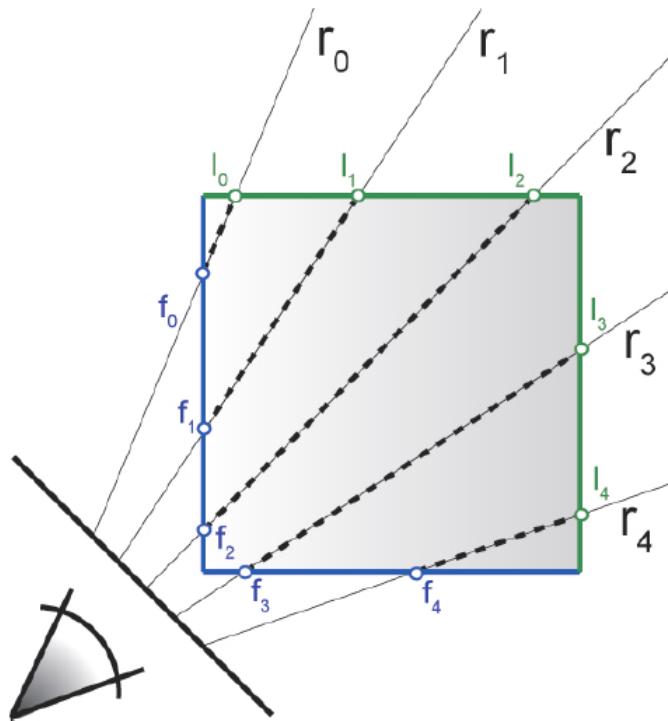
- Überspringen transparenter Bereiche, Bildraumansatz (= pro Sichtstrahl)
 - verwende Octree od. reguläres Gitter zur Strukturierung des Volumens
 - speichere minimalen und maximalen Skalarwert für jede Zelle (genannt „Volume Bricks“)
 - anhand dieser Werte kann man schnell bestimmen, ob
 - eine Isofläche durch die Zelle verläuft
 - die Zelle transparent ist (benötigt TF)
 - Raycasting nur in aktiven Bricks



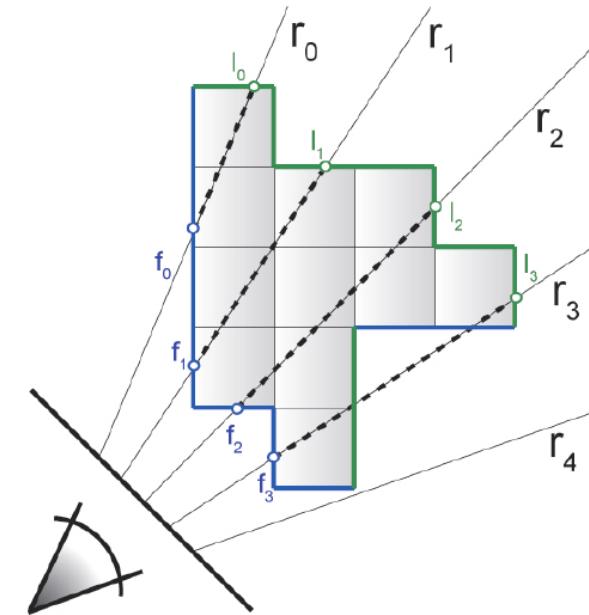
Ray-Casting: Optimierungen

Object-Order Empty Space Skipping

- ▶ verwende die Information der Zellen, um die Ein-/Austrittspunkte in die relevanten (also nicht-transparenten) Bereiche besser zu bestimmen
- ▶ Anwendung vor dem Raycasting (daher ein Objektraum-Ansatz) durch Image-Based Ray Setup/Termination



rasterisiere Bounding Box

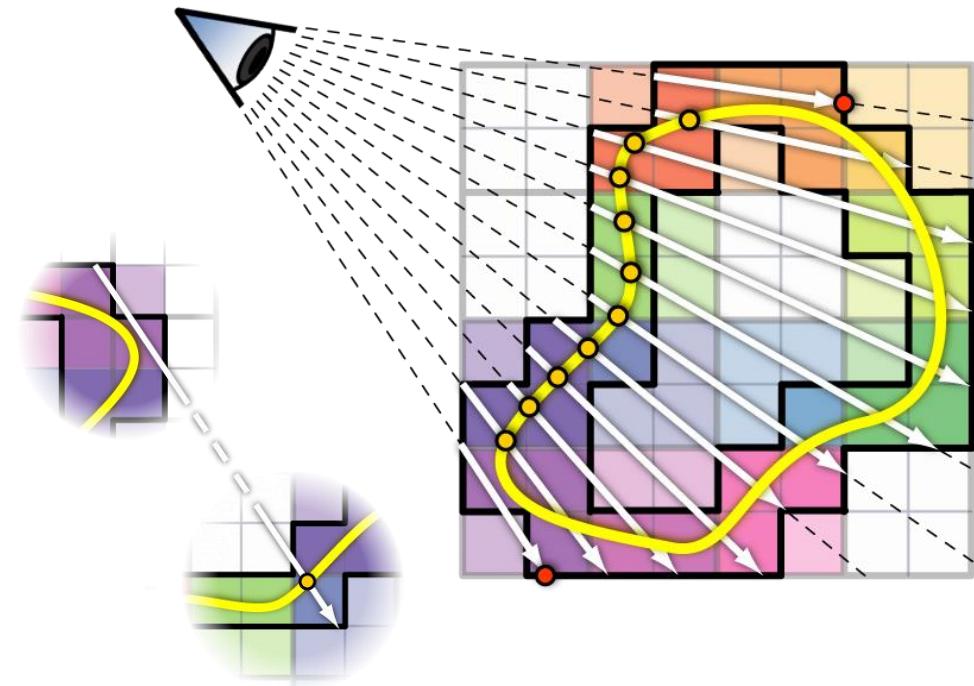


rasterisiere enger anliegende
Bounding Geometrie

Ray-Casting: Optimierungen

Object-Order Empty Space Skipping

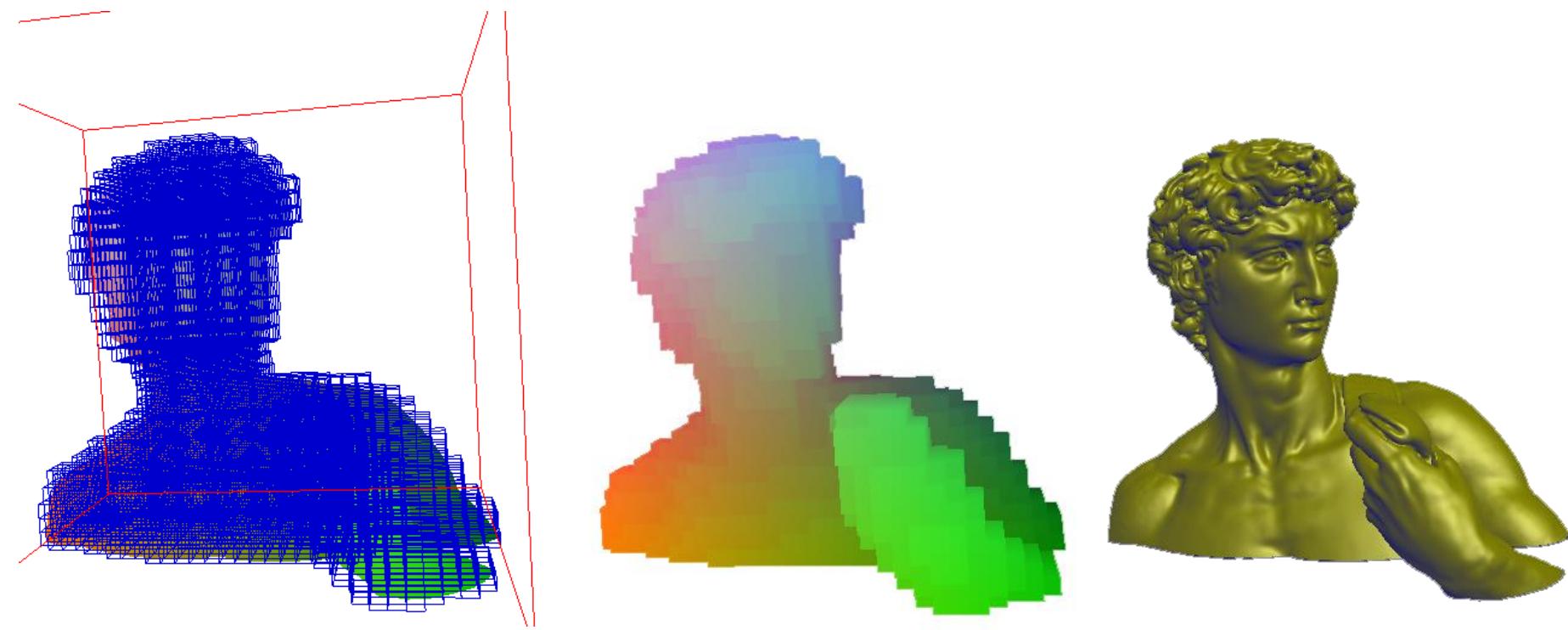
- ▶ verwende die Information der Zellen, um die Ein-/Austrittspunkte in die relevanten (also nicht-transparenten) Bereiche besser zu bestimmen
- ▶ rasterisiere Vorder- und Rückseiten der aktiven Bricks
 - ▶ Vorderseiten mit `glDepthFunc(GL_LESS)`;
 - ▶ Rückseiten mit `glDepthFunc(GL_GREATER)`;
- ▶ nicht der gesamte freie Raum wird übersprungen (Silhouette)
- ▶ Überspringen transparenter Bereiche im Innern ist schwierig



Ray-Casting: Optimierungen

Object-Order Empty Space Skipping

► Beispiel

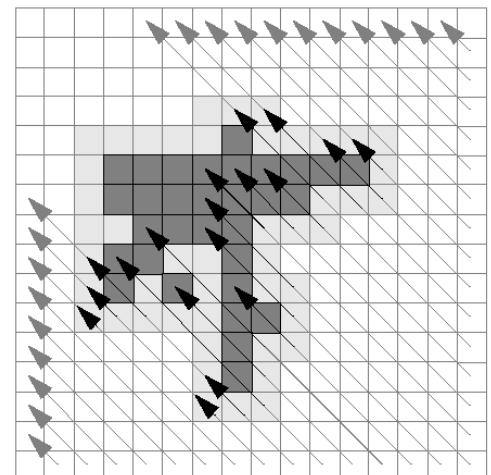
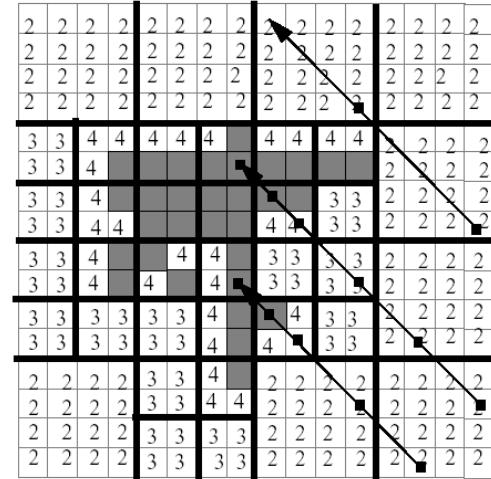


Raycasting: Optimierungen

Homogene Regionen (heuristische Techniken)

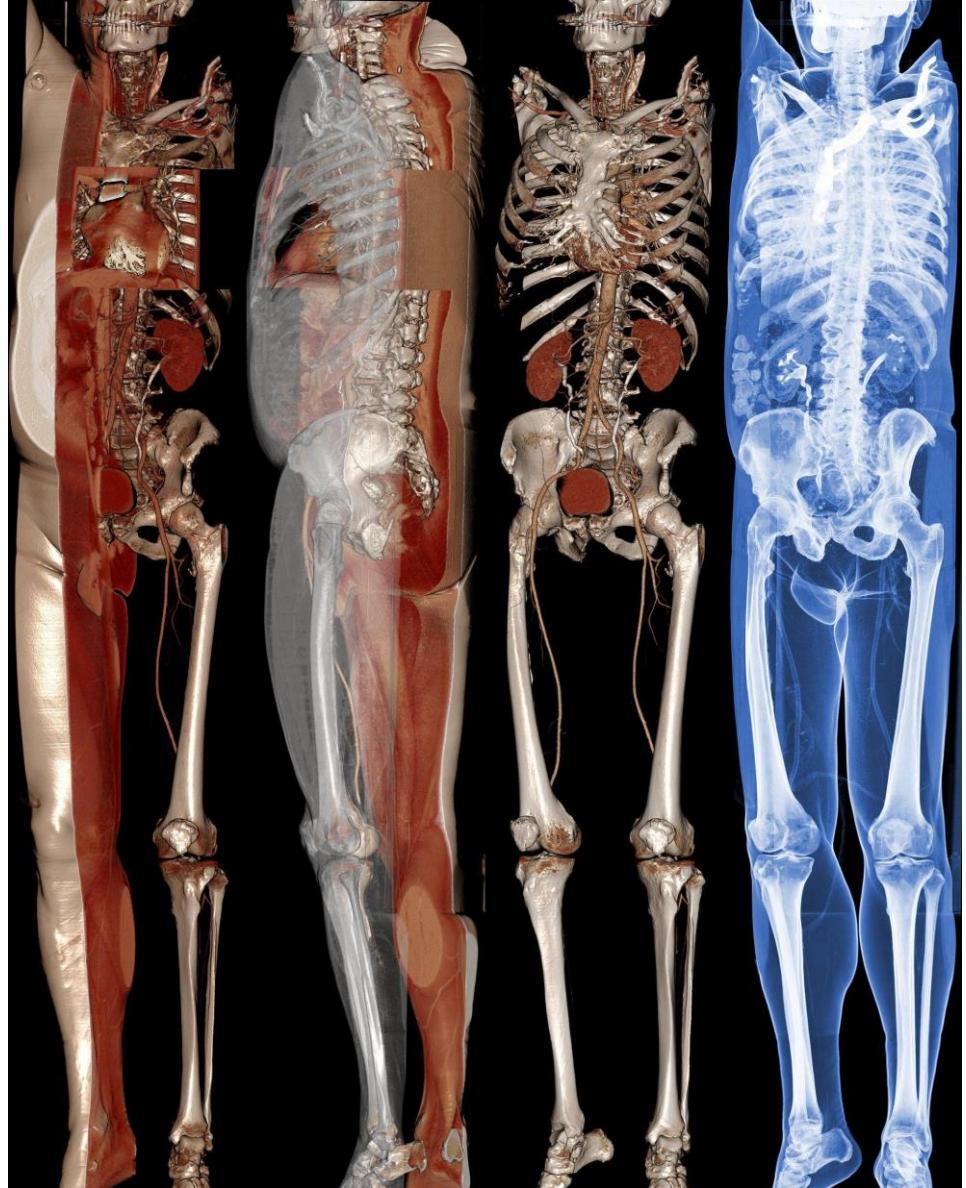
- ▶ verwende Octree oder grobes Gitter
 - ▶ speichere Mittelwert und Varianz der Skalarwerte in einer Zelle
 - ▶ leite daraus Transparenz und Homogenität ab
 - ▶ passe Schrittweite an (Bild rechts: Zahlen zeigen, wie groß ein uniformer Bereich ist)

- ▶ Speichere/bestimme pro Voxel
 - ▶ „Traversierungs-Geschwindigkeit“ (kleiner in der Nähe von interessanten Strukturen), oder
 - ▶ Entfernung zu interessanten Strukturen



„Große“ Volumendaten

- ▶ Long-leg study
512×512×3172
@16bit ~ 1.7GB



„Große“ Volumendaten

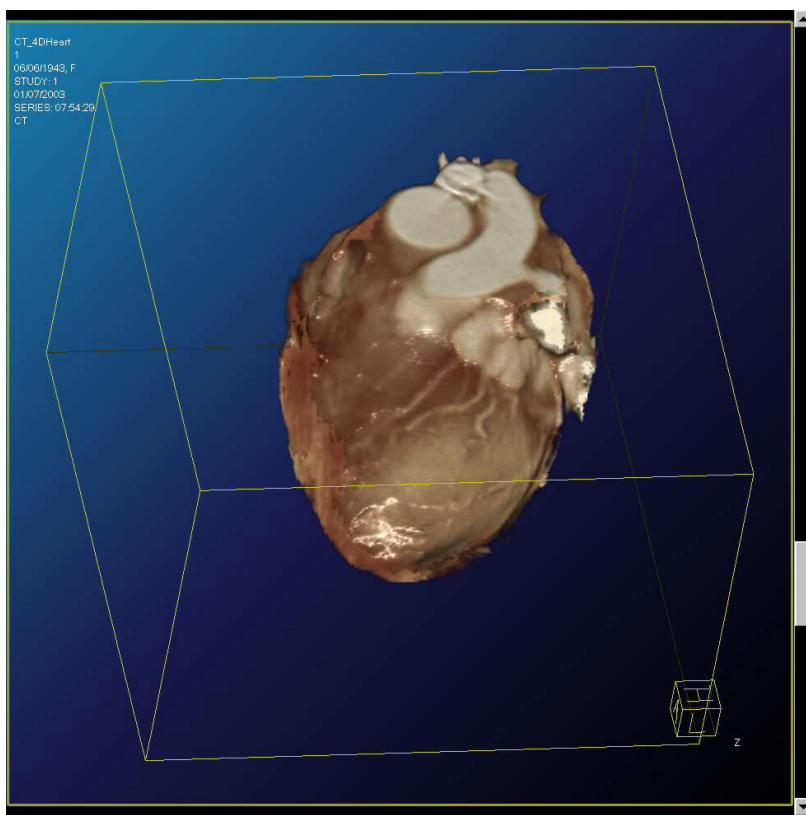
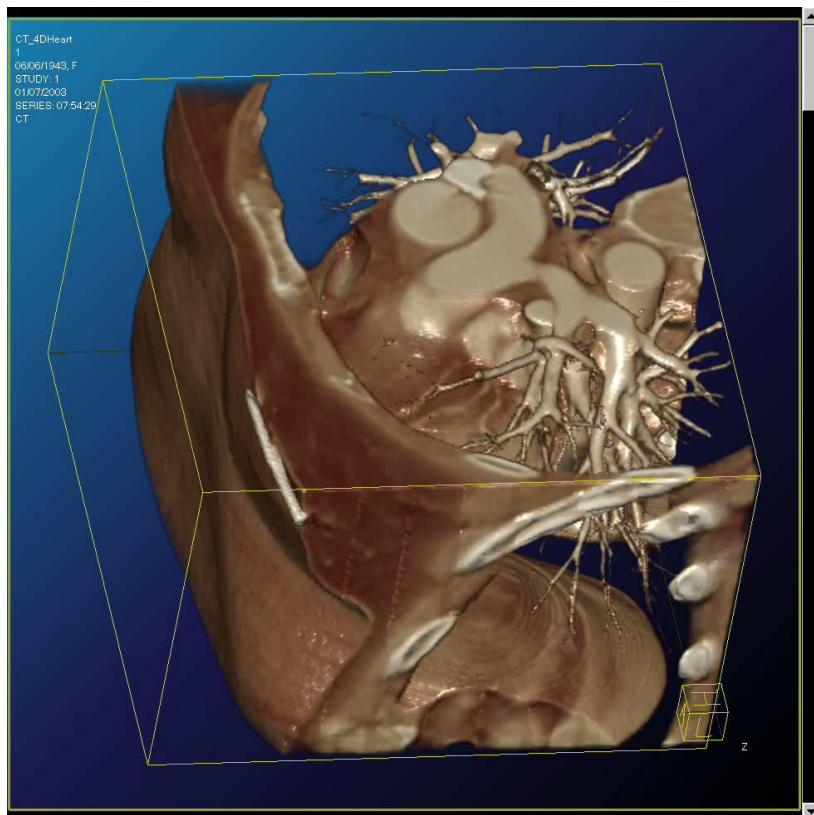
- ▶ Visible Male Cryosection RGB Data:
2048×1216×877@24bit ~ 14GB



THE VISIBLE HUMAN PROJECT®

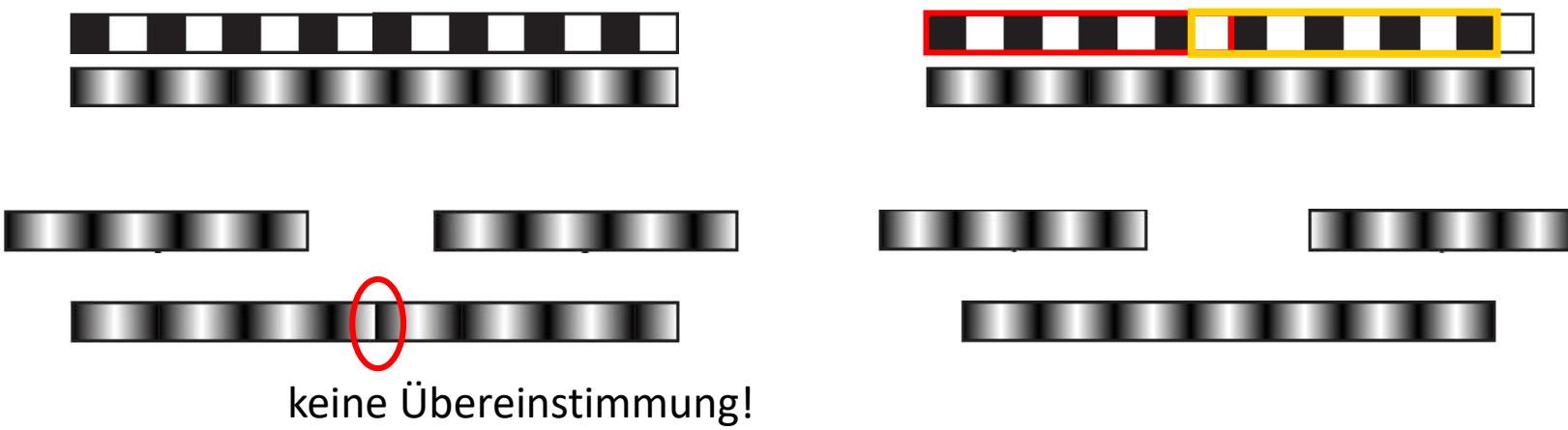
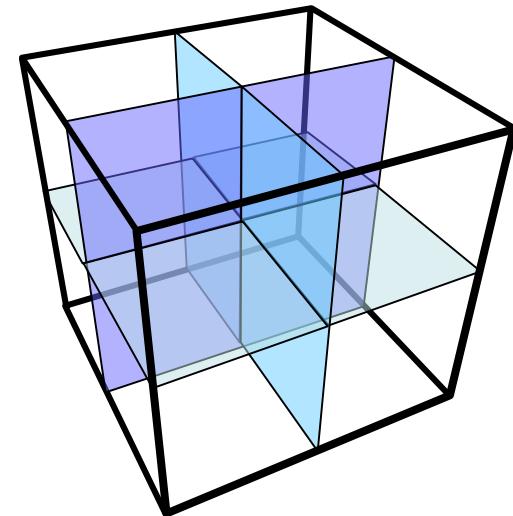
„Große“ Volumendaten

- zeitabhängige Daten:
4D Kardiodaten mit $512 \times 512 \times 240$ @16 bit, 20 Frames $\sim 2.5\text{GB}$



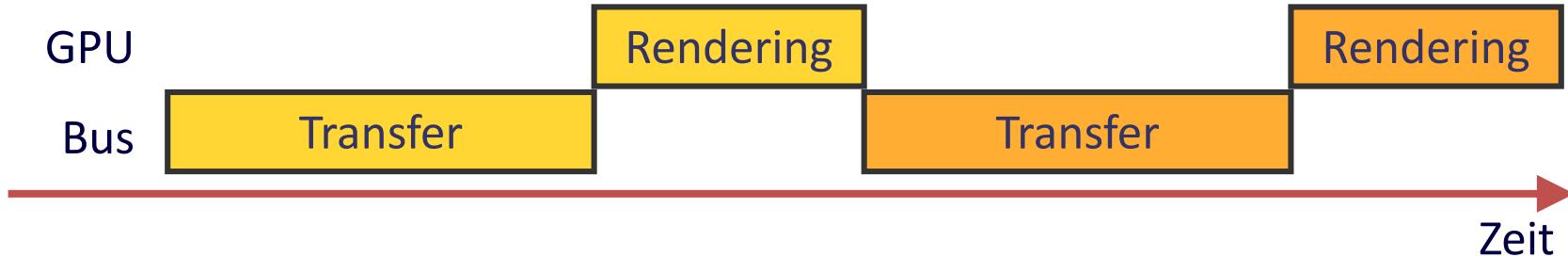
Bricking

- ▶ was tun, wenn die Daten nicht in den Speicher (der GPU) passen?
 - ▶ teile die Daten in sog. **Chunks** oder **Bricks** auf, um sie nacheinander abzuarbeiten (reguläre Gitter, Octrees, ...)
- ▶ bei (tri)linearer Interpolation muss eine Voxel-Schicht dupliziert werden, um korrekte Interpolation zu gewährleisten
 - ▶ bei Interpolation höherer Ordnung entsprechend mehr



Bricking

- naive Lösung:

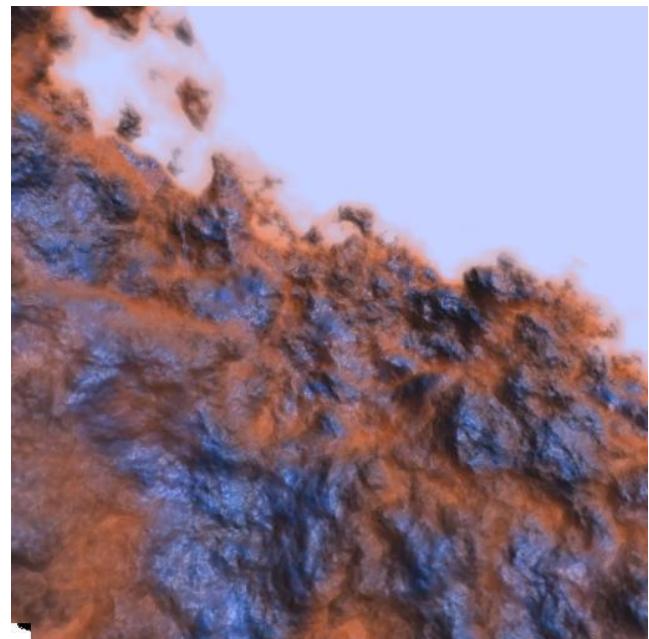
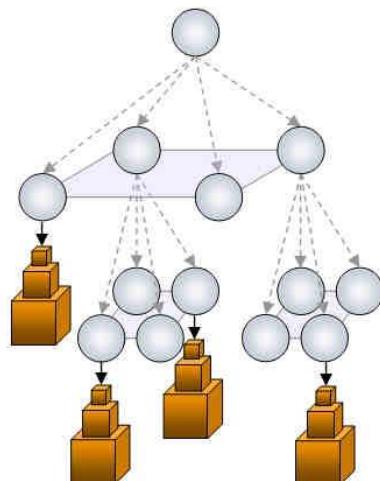
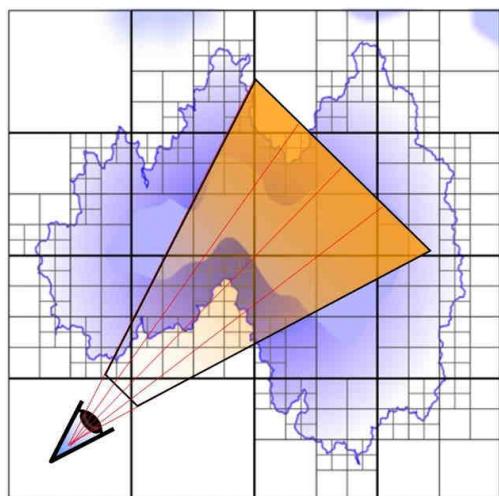


- effizienteres Rendering durch:

- ▶ Herausfinden, welche Bricks ein Strahl der Reihe nach schneidet oder Compositing der Beiträge der Bricks im Rendertarget
 - ▶ mehr als 1 Brick muss in den Videospeicher passen:
nur so können Daten asynchron zum Rendering transferiert werden!
 - ▶ effektives Load Balancing ist nicht trivial
-
- ▶ mögliche Optimierung: niedrig aufgelöste Versionen von Bricks vorberechnen und nutzen, wenn ausreichend („Mip-Mapping“)
 - ▶ Achtung: Anwendung der TF auf Mip-Map \neq Mip-Map des Resultats!

Gigavoxels

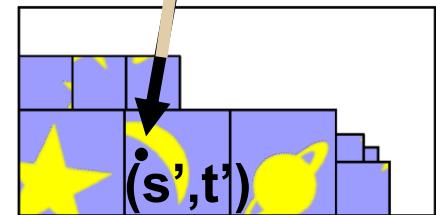
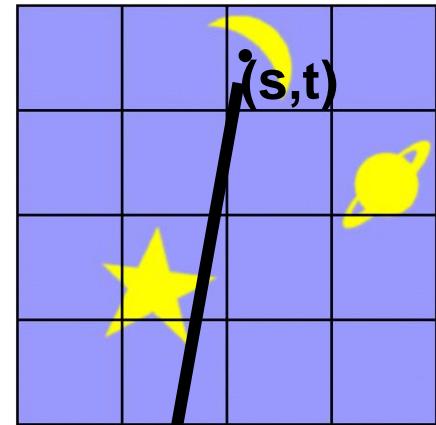
- Gigavoxels ist ein Verfahren für effizientes Bricking und Out-of-Core Rendering von RGBA Daten (keine Transferfunktion o.ä.)
 - basierend auf einem Octree und einer Mip-Map Pyramide für Bricks
 - <http://maverick.inria.fr/Membres/Cyril.Crassin/index.html>



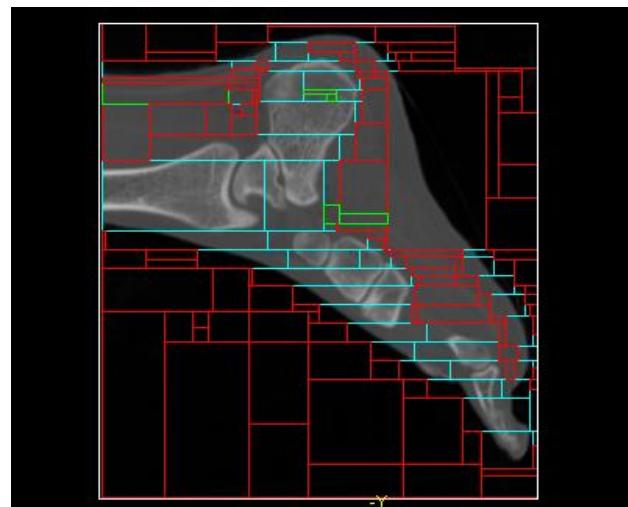
Große Volumendaten

Weitere Ansätze

- ▶ Sparse-Volume Data (Kraus et al., Adaptive Texture Maps, Graphics Hardware Workshop'02)
 - ▶ grobes Gitter dessen Zellen auf eine Art 3D-Texturatlas verweisen

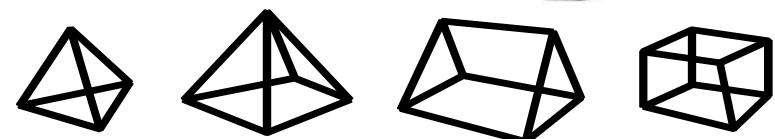
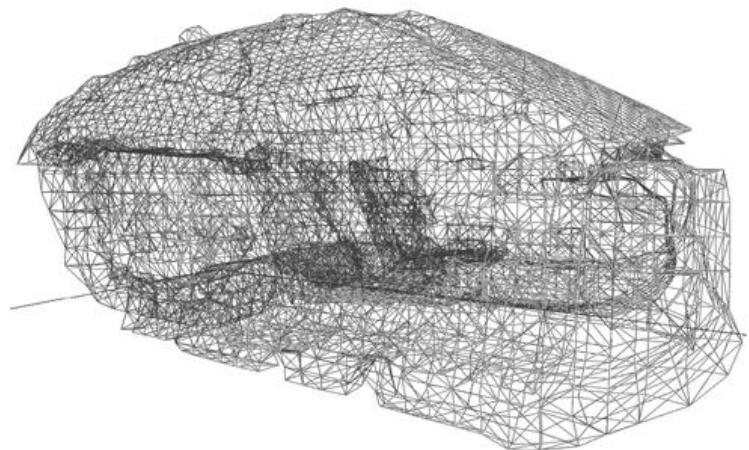


- ▶ Zerlegung des Volumens mit einem kD-Baum in homogene/heterogene Bereiche
 (Li-Yi Wei et al. – Texture Partitioning and Packing for Accelerating Texture-based Volume Rendering, GI 2003)



Motivation

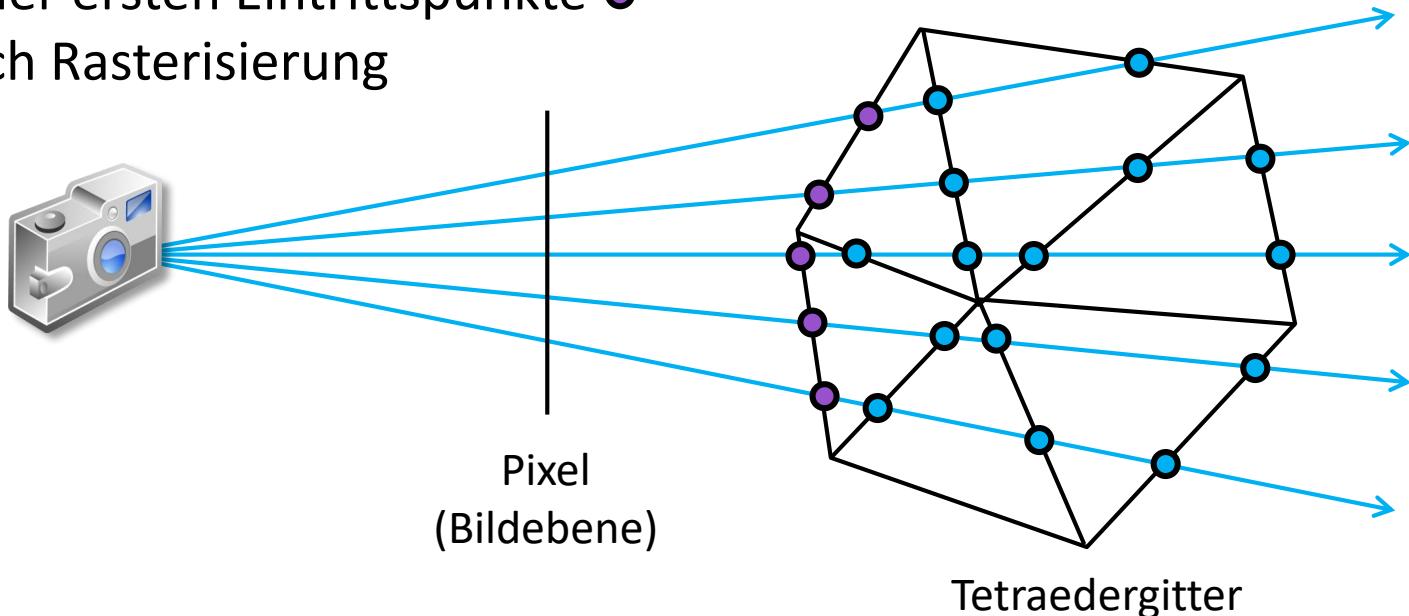
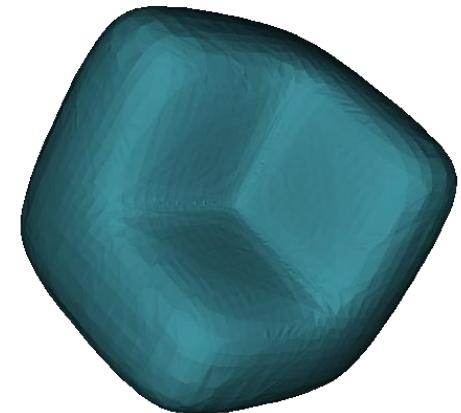
- ▶ nicht immer sind die Daten auf uniformen Gittern gegeben...
- ▶ universell sind Tetraedergitter
 - ▶ Simplizes: lineare Interpolation in Zellen
 - ▶ der Hauptunterschied zu uniformen Gittern ist die Traversierung
„Raytracing Irregular Volume Data“, M. P. Garrity, VolVis, p. 35-40, 1990
(behandelt Gitter aus allg. konvexen Zellen)



Tetraedergitter: Raycasting

Grundidee

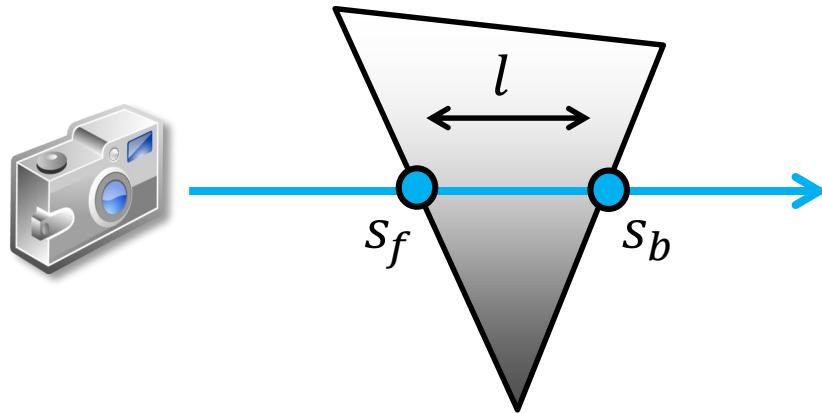
- ▶ Traversierung von vorne nach hinten
- ▶ berechne Ein- und Austrittspunkt der Zellen
- ▶ berechne Emission/Absorption der Strahlsegmente
- ▶ Compositing der Segmente wie bisher
- ▶ Austrittspunkt = Eintrittspunkt der nächsten Zelle
 - ▶ benötigt Datenstruktur mit Nachbarschaften
- ▶ Finden der ersten Eintrittspunkte ●
 - z.B. durch Rasterisierung



Tetraedergitter: Raycasting

Beitrag eines Strahlsegments: Präintegration

- ▶ Skalarwerte s_f und s_b werden durch lineare Interpolation der Werte an den Ein- und Austrittspunkten berechnet
- ▶ zusätzlicher Parameter: Länge l des Segments
- ▶ berechne eine 3D-Präintegrationstabelle mit Parametern s_f, s_b, l
- ▶ die baryzentrische (= lineare) Interpolation innerhalb eines Tetraeder ist „kompatibel“ mit dem angenommenen linearen Verhalten entlang des Strahls (im Gegensatz zu trilinearer Interpolation bei regulären Gittern!)

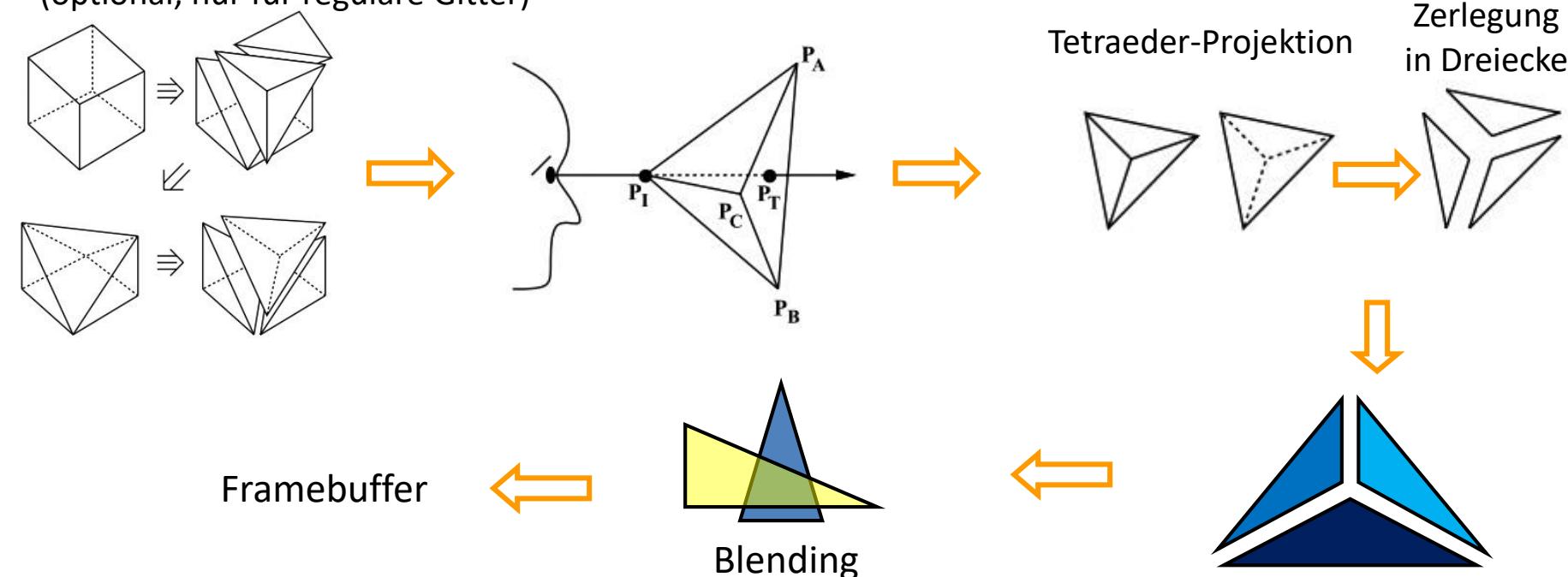


Tetraedergitter: Cell Projection

Motivation: Alternative zu Raycasting (eine ältere Technik)

- ▶ betrachtet man die Projektion der Kanten eines **Tetraeders** auf die Bildebene, so ergeben sich max. 4 Dreiecke
- ▶ Idee: berechne Emission und Opazität an deren Vertices und zeichne Dreiecke mit Interpolation, Compositing mit Blending

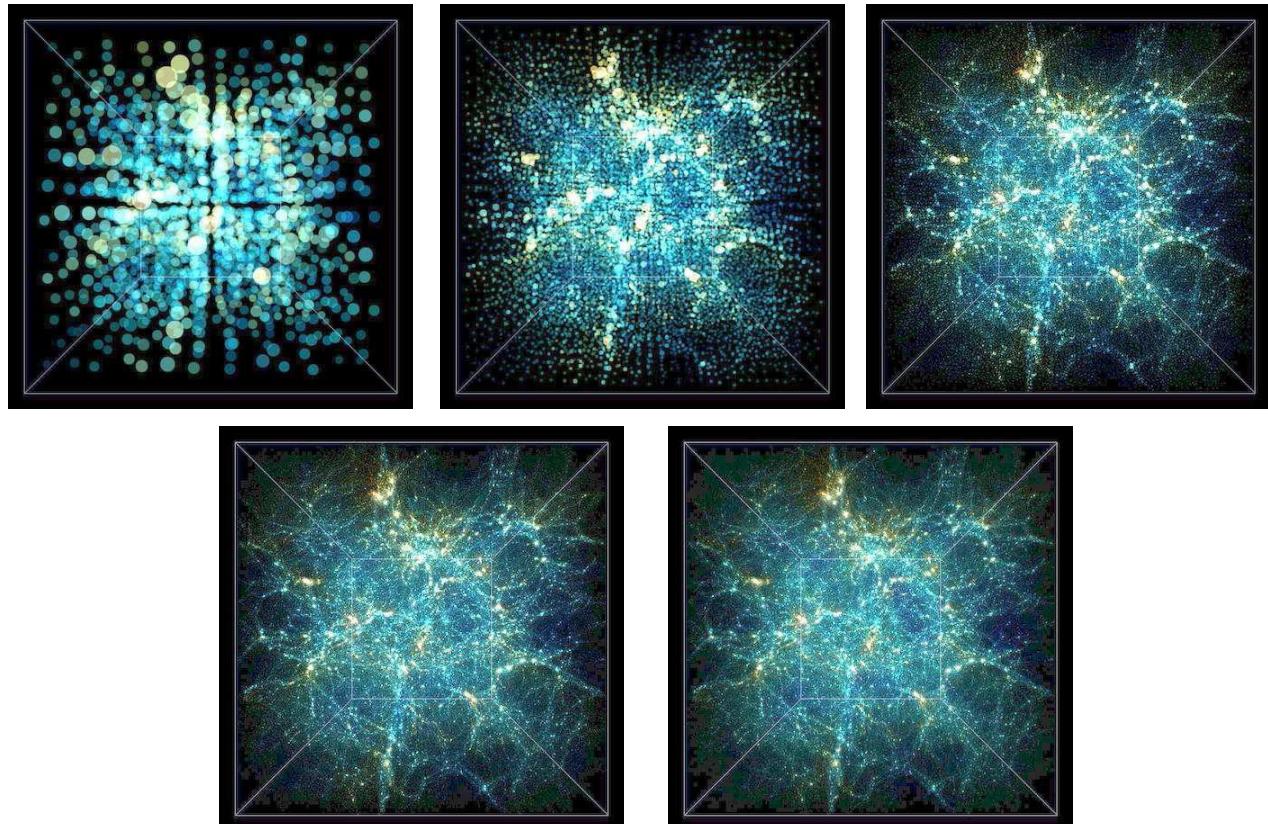
Zerlegung eines Würfels in Tetraeder
(optional, nur für reguläre Gitter)



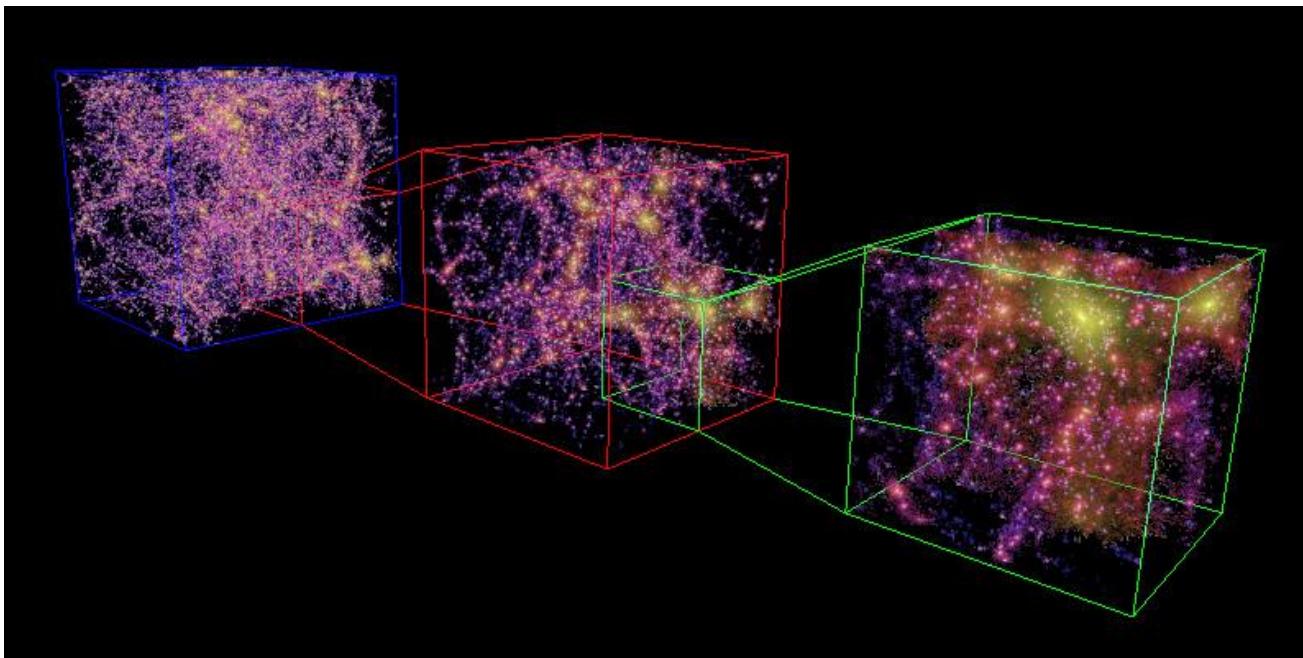
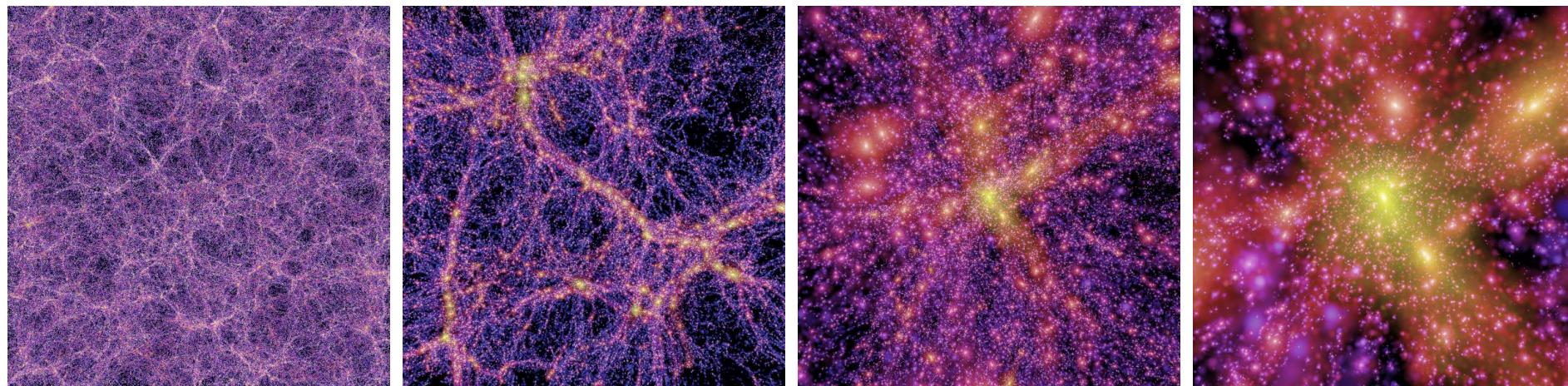
Splatting [Westover 1990]

Volumenrendering für gitterfreie Daten und (auch) unstrukturierte Gitter

- projiziere jedes Sample auf die Bildebene und berechne Beitrag
 - ▶ Sample = Abtastpunkt bei gitterfreien Daten (Hauptanwendung)
 - ▶ Object-Order Methode
- Bsp. durch genügend viele Splats („Farbklekse“) entsteht ein Bild des Volumens

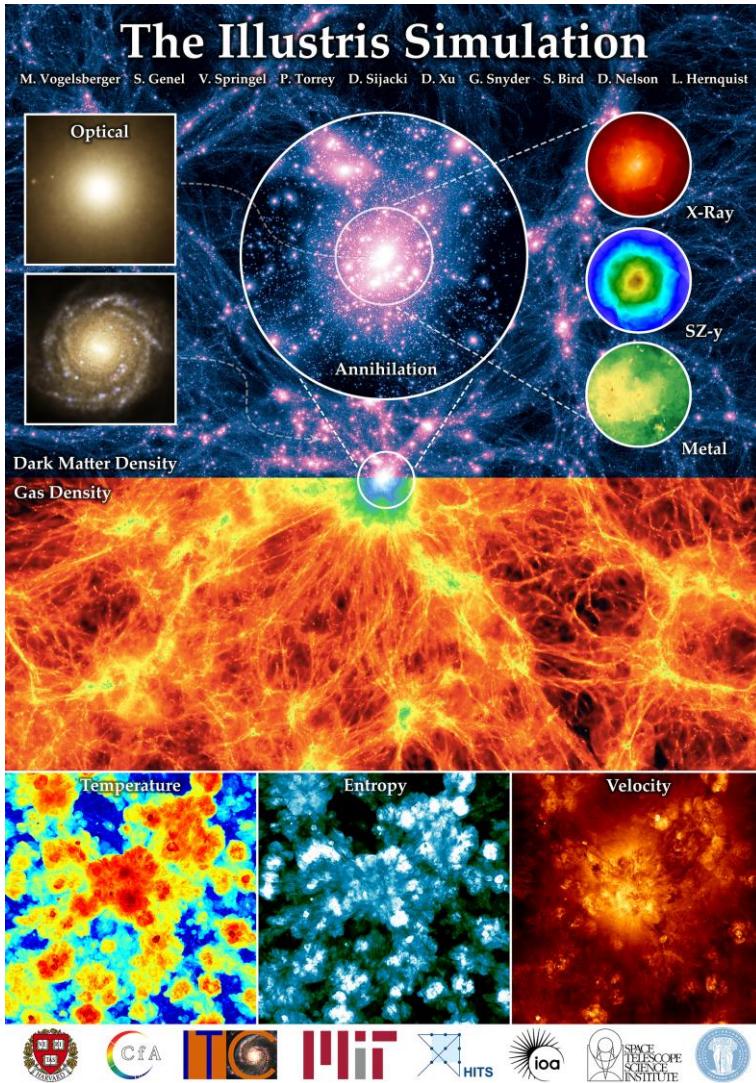


Millenium Run (10 Milliarden Partikel)

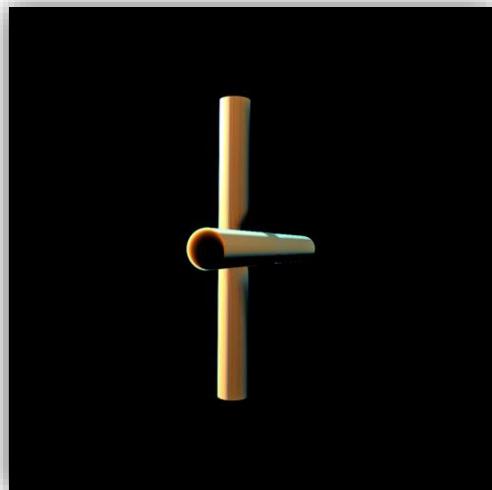


The Illustris Simulation

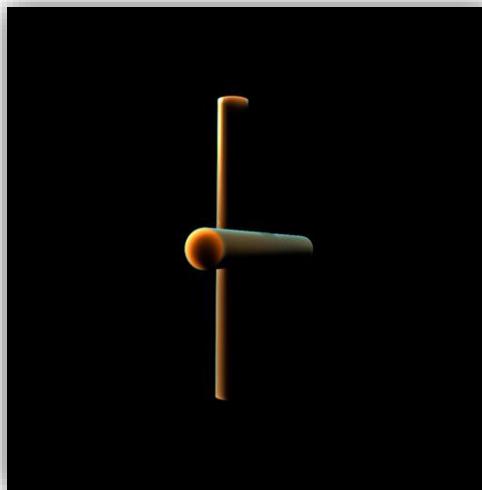
- ... the total data volume is ~ 265 TB, including ~ 800 full volume snapshots and $\sim 30,000$ subbox snapshots (<https://arxiv.org/abs/1504.00362>)



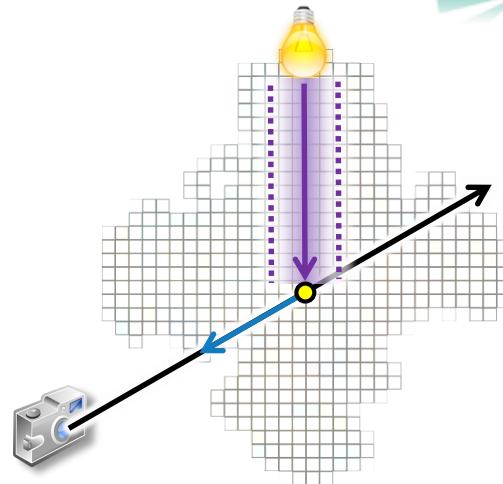
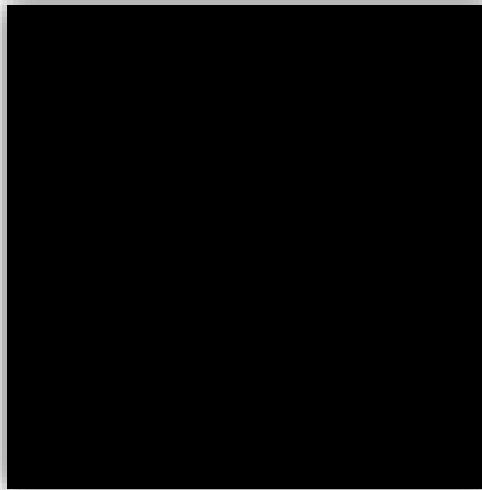
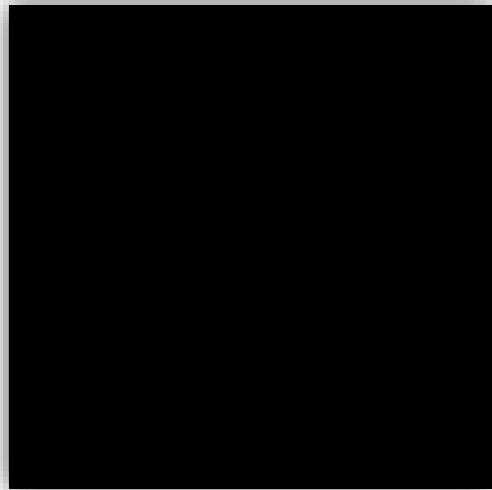
Forschung: Weiche Schatten



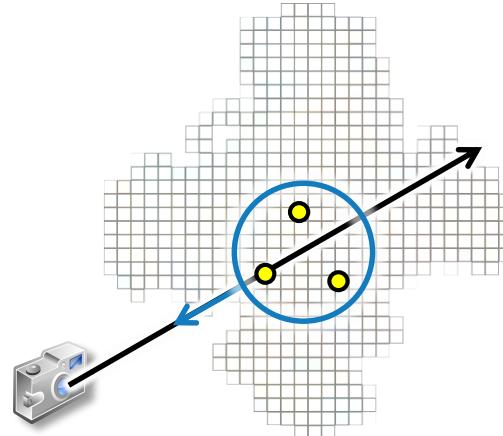
vorher



nachher



Low-Pass Filtered Volumetric Shadows
Ament, Sadlo, Dachsbacher, Weiskopf
IEEE Trans. on Visualization and Computer Graphics (Proc. IEEE VIS), 2014

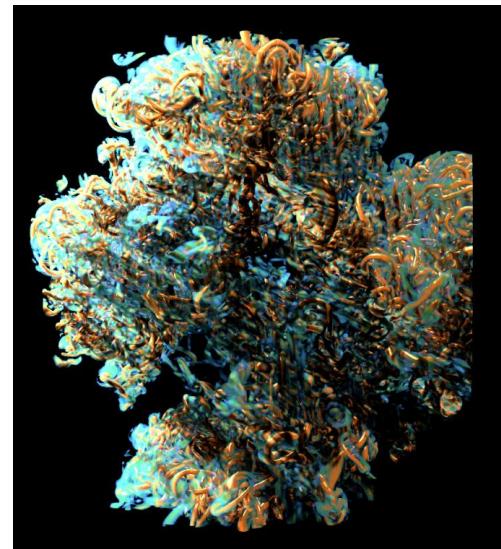
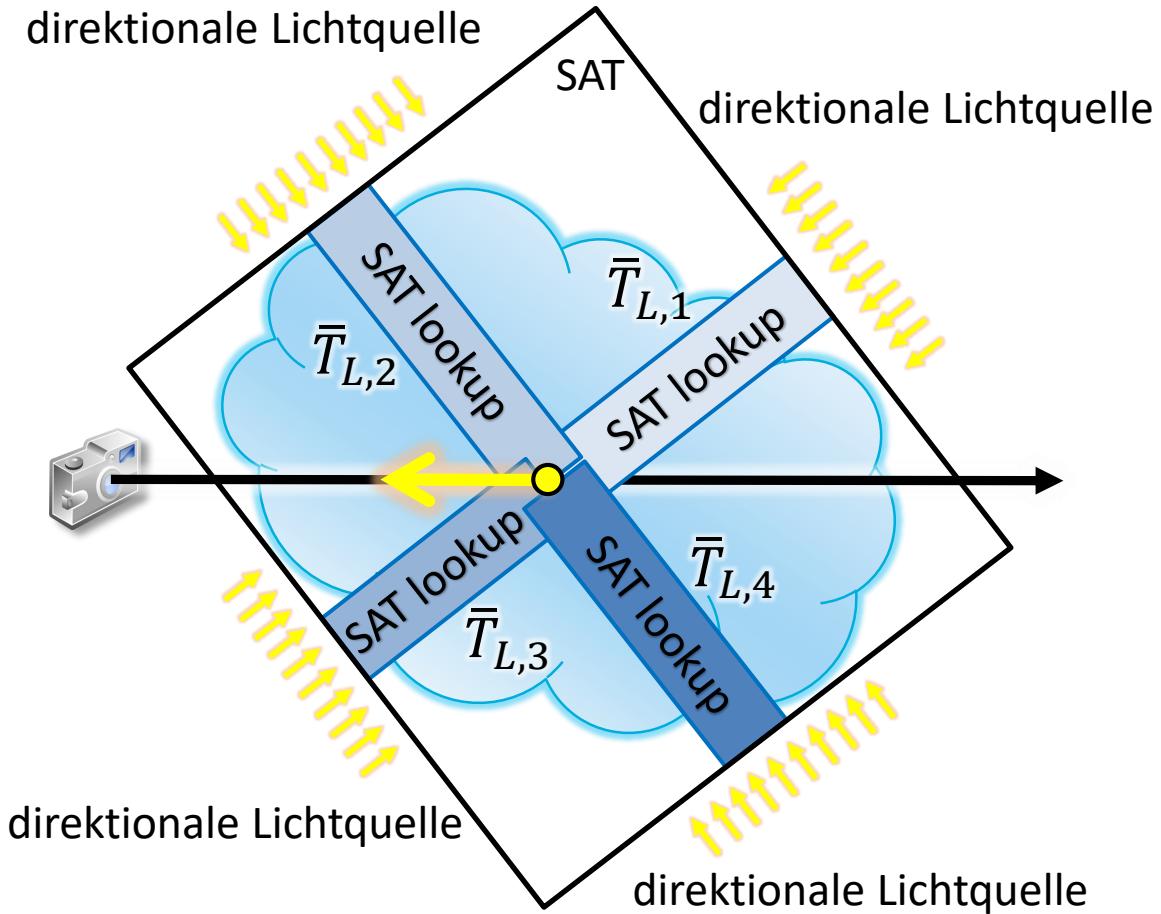


Anisotropic Ambient Volume Shading
Ament, Dachsbacher
IEEE Trans. on Visualization and Computer Graphics (Proc. of IEEE VIS), 2015

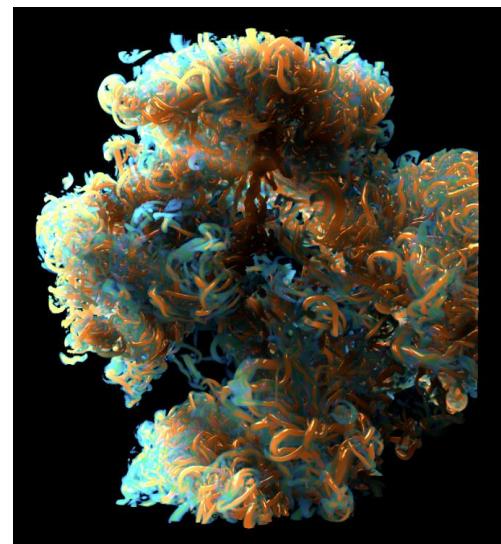
Forschung: Weiche Schatten

Low-pass filtered volumetric shadows

- effizientes Tiefpassfiltern der Transmittance mit 3D Summed Area Tables (SAT)

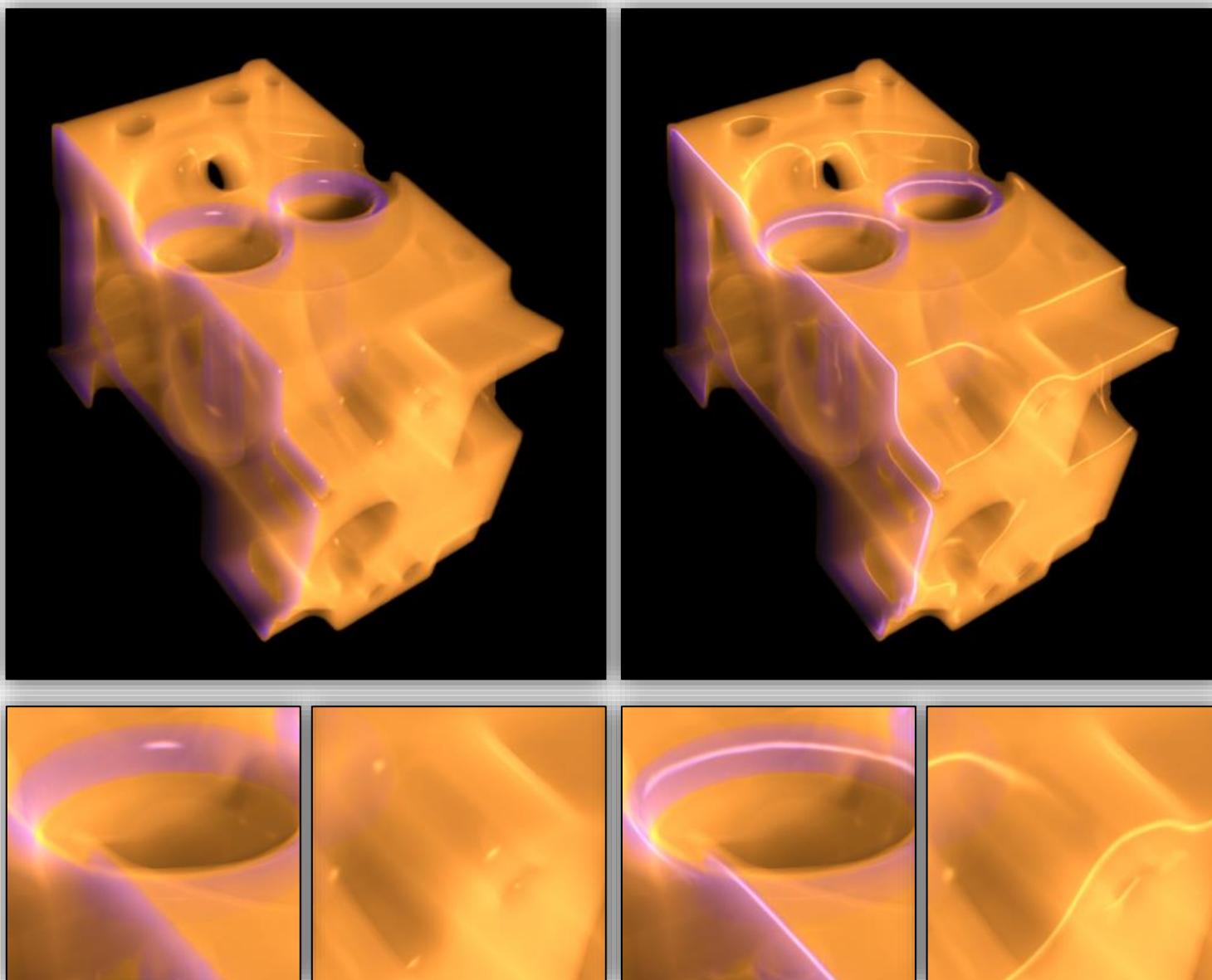


ohne Tiefpassfilter



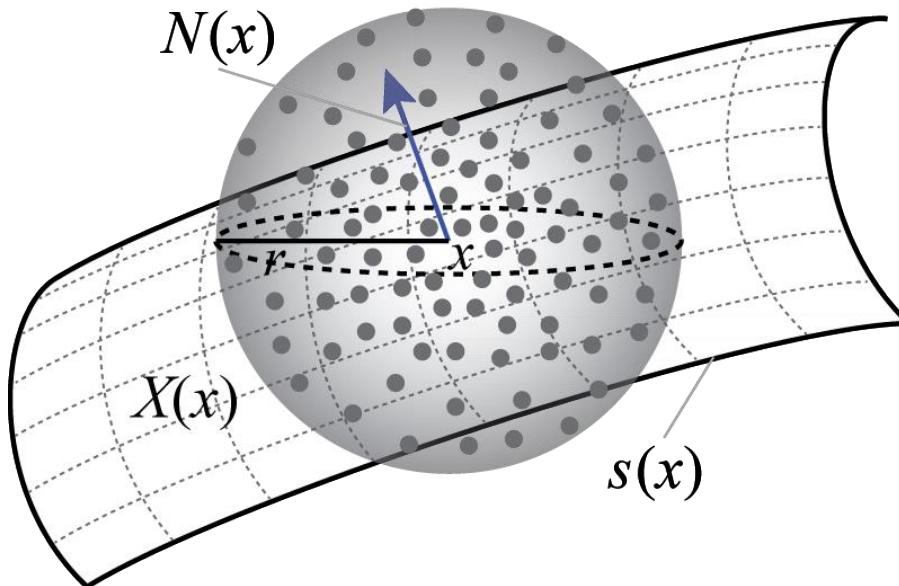
mit Tiefpassfilter

Forschung: Anisotrope Oberflächenreflexion



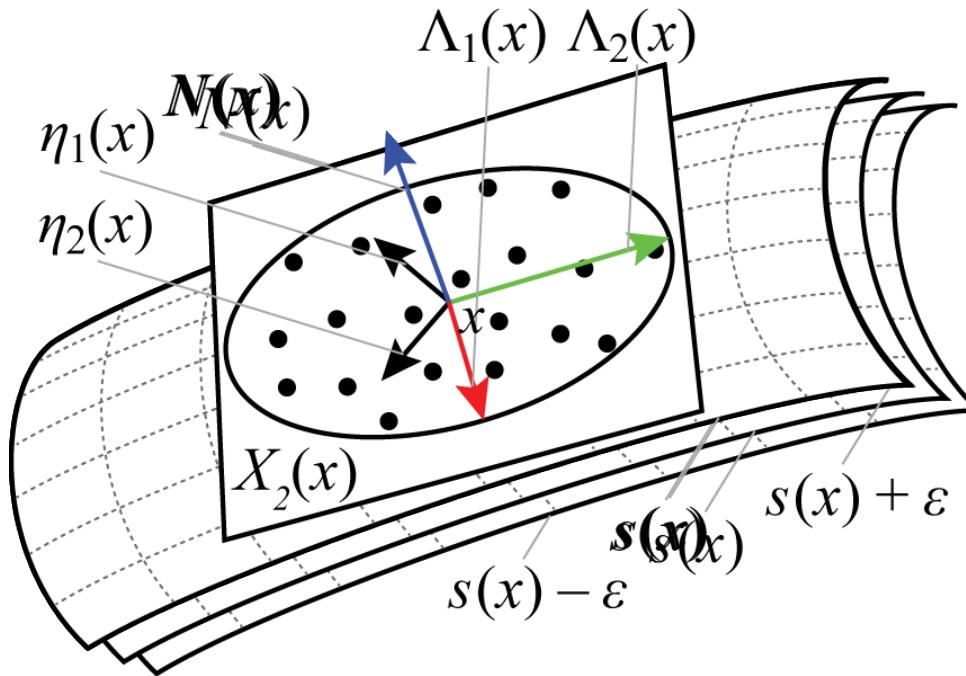
Ambience Exploration

- ▶ Punkt x auf der (gedachten) Isofläche mit Skalarwert $s(x)$
- ▶ Normale $N(x)$ der Fläche: Gradient des Skalarfelds
- ▶ betrachte zufällige Punkte $X(x)$ in der r -Umgebung von x



Ambience Exploration

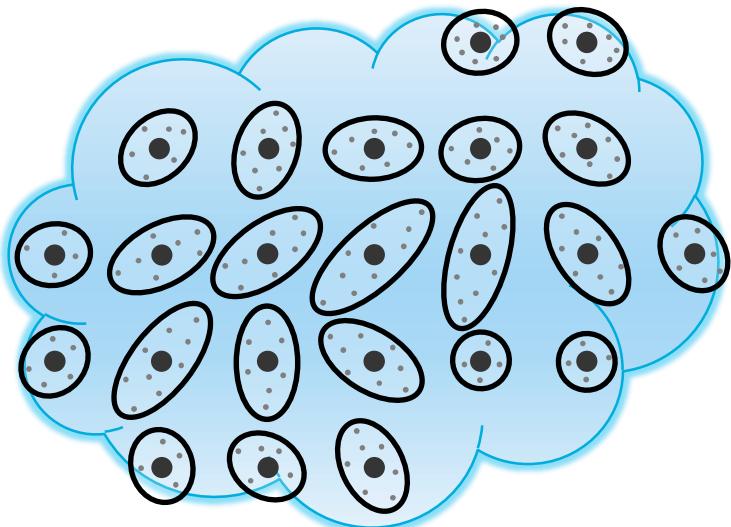
- betrachte zufällige Punkte $X(x)$ in der r -Umgebung von x
- ... und hiervon die Stellen für die gilt: $\|s(x_i) - s(x)\| < \epsilon$
- bestimme Eigenvektoren und -werte aus der Kovarianzmatrix



- Anisotropie nach $\frac{\bar{\Lambda}_1}{\|\bar{\Lambda}_1\|}$ und $\frac{\bar{\Lambda}_2}{\|\bar{\Lambda}_2\|}$ und dem Verhältnis der Eigenwerte

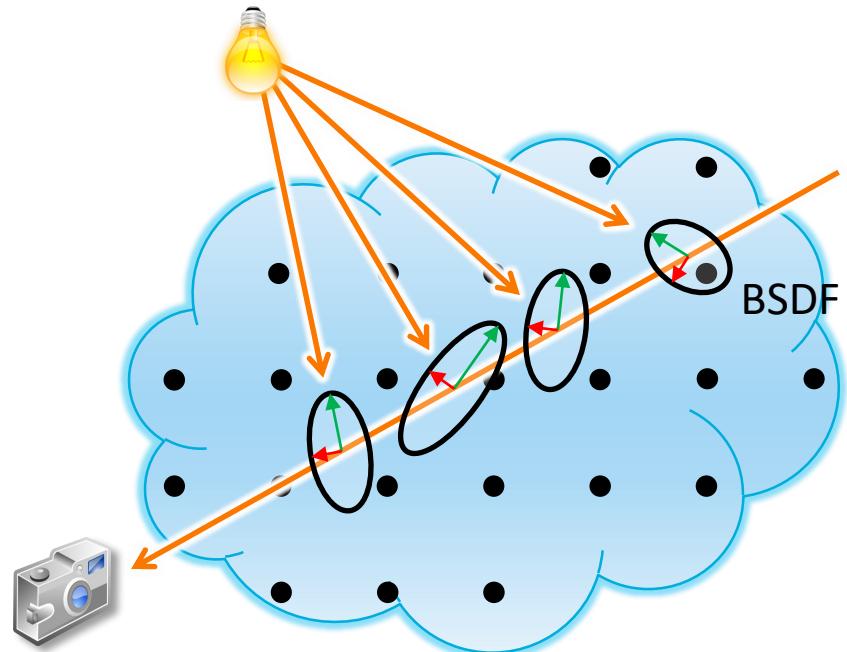
Volume Rendering Algorithmus (CUDA)

Ambience Exploration und
Bestimmung der Anisotropie



→ Gitter mit symmetrischen 2×2
Kovarianzmatrizen (3 Werte pro Voxel)

Direktes Volumen Rendering mit
Transferfunktion und anisotropen BSDFs



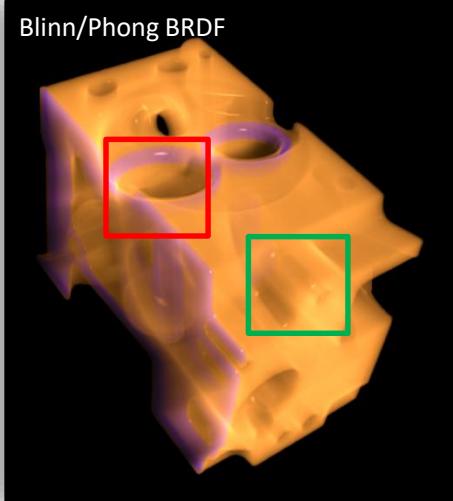
Forschung: Anisotrope Oberflächenreflexion

Ergebnisse

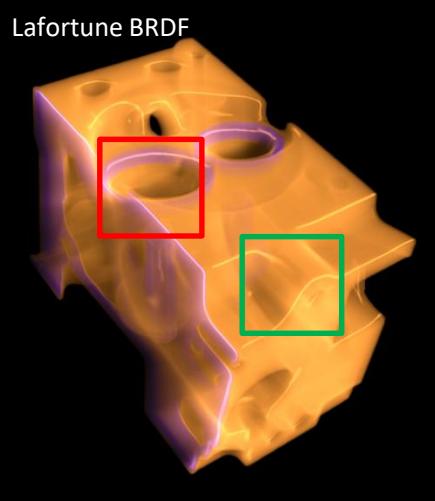
GPU: NVIDIA Titan

Engine: $256 \times 256 \times 110$

isotrop



anisotrop



97 fps

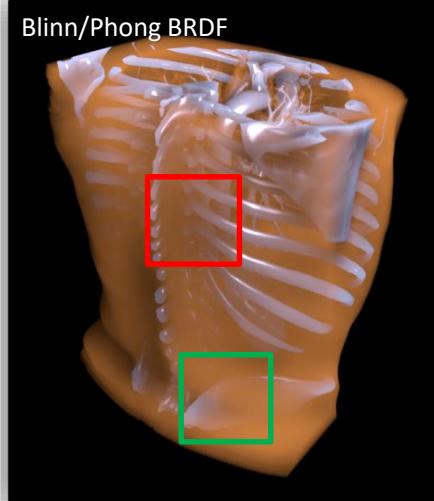
69 fps

Anisotr.: 627 ms / 21 MB

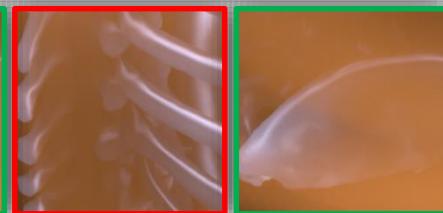
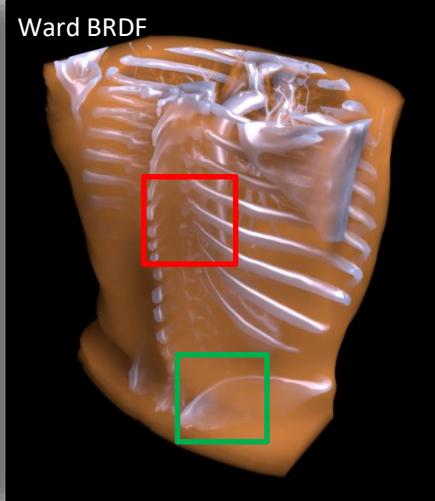
GPU: NVIDIA Titan

Mecanix: $256 \times 256 \times 302$

isotrop



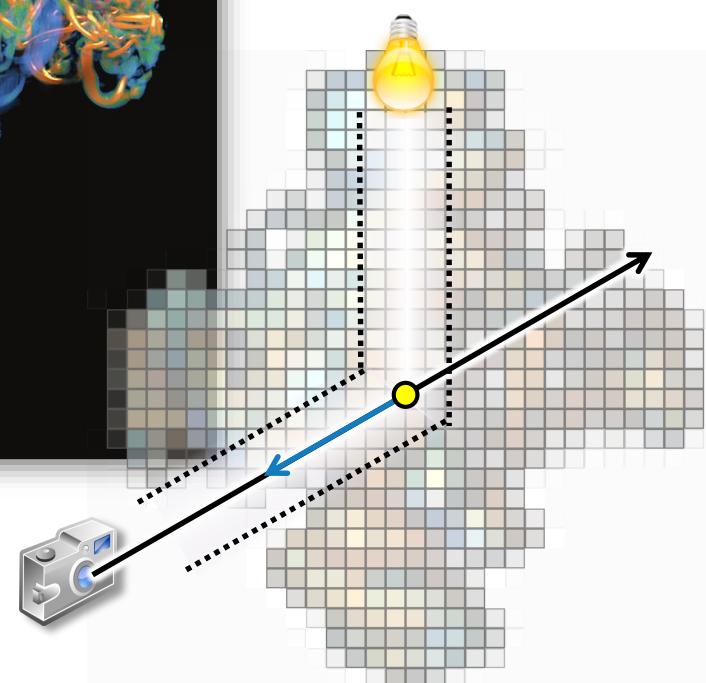
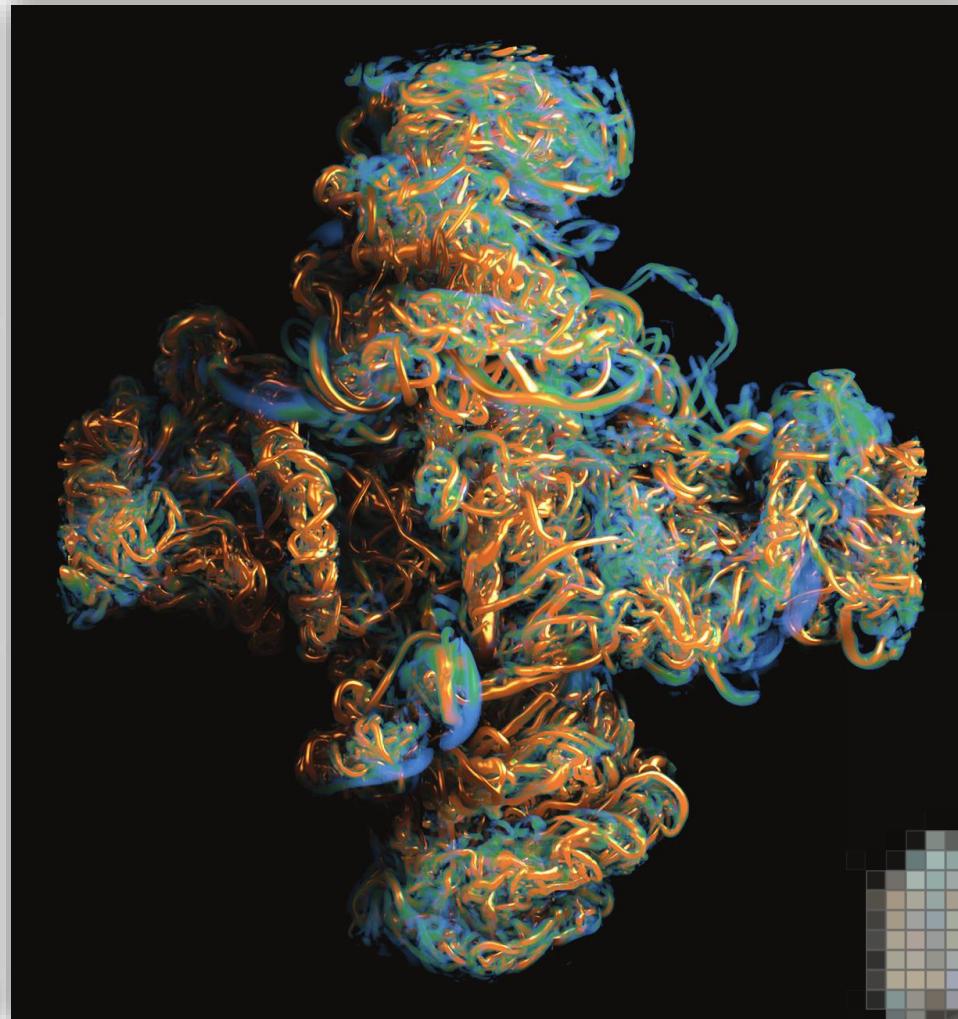
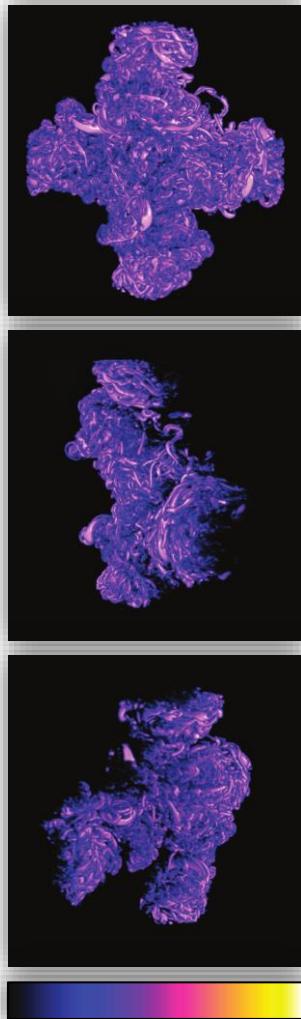
anisotrop



168 fps

59 fps
Anisotr.: 2149ms / 57 MB

Volumenvisualisierung

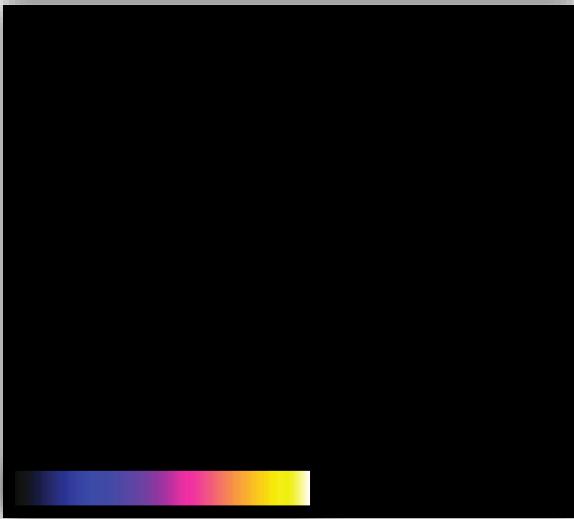
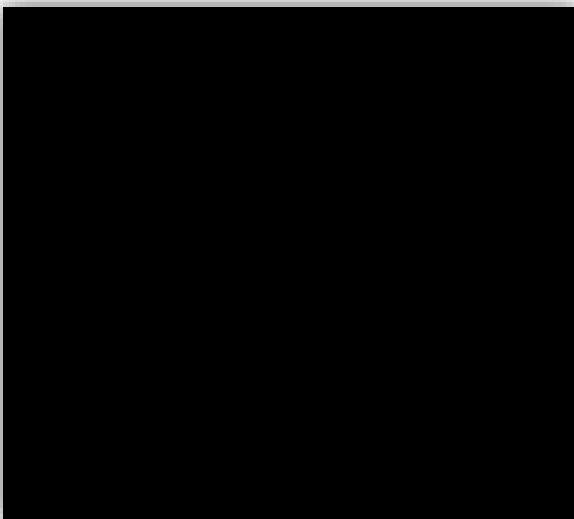


Extinction-Optimized Volume Illumination

Ament, Zirr, Dachsbacher

IEEE Trans. on Visualization and Computer Graphics, 2016

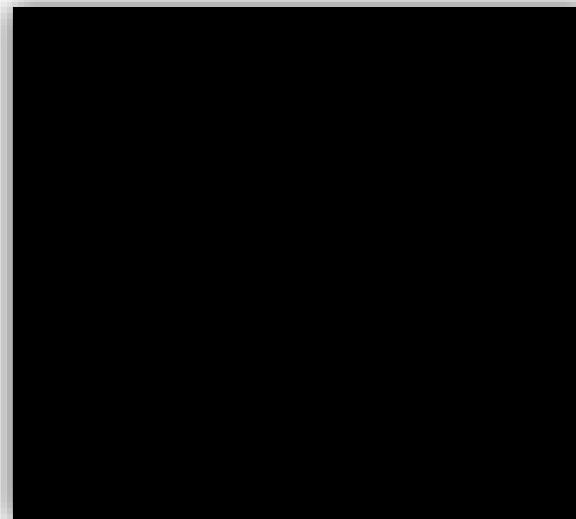
Forschung: Opacity Optimization



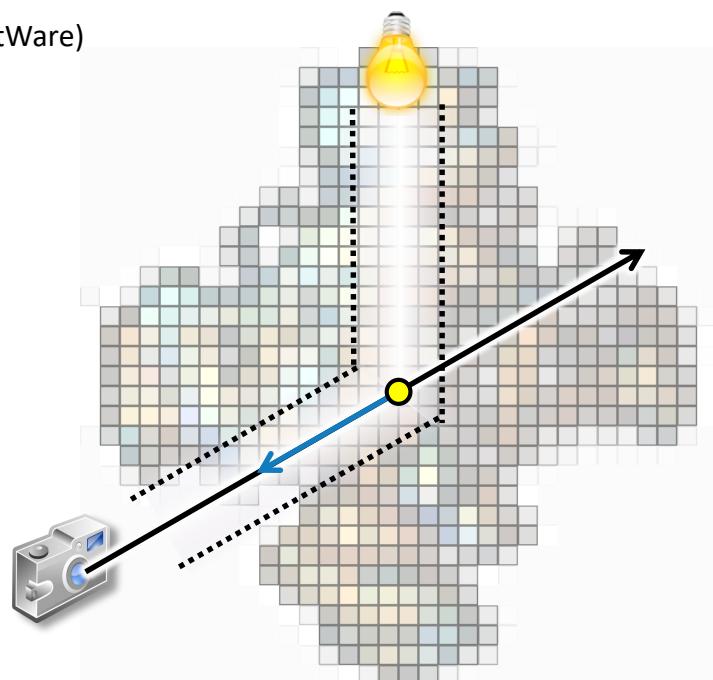
Extinction-Optimized Volume Illumination

Ament, Zirr, Dachsbacher

IEEE Trans. on Visualization and Computer Graphics, 2016

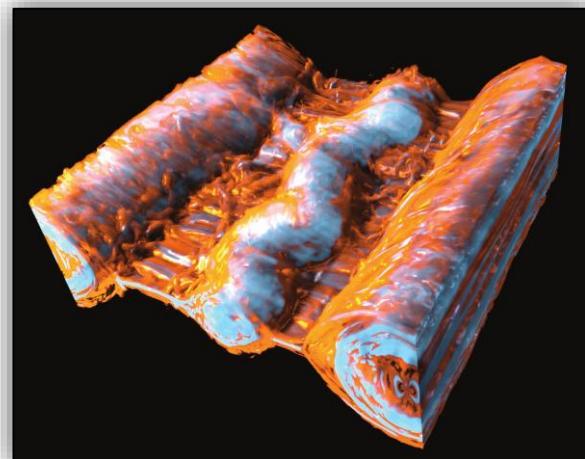
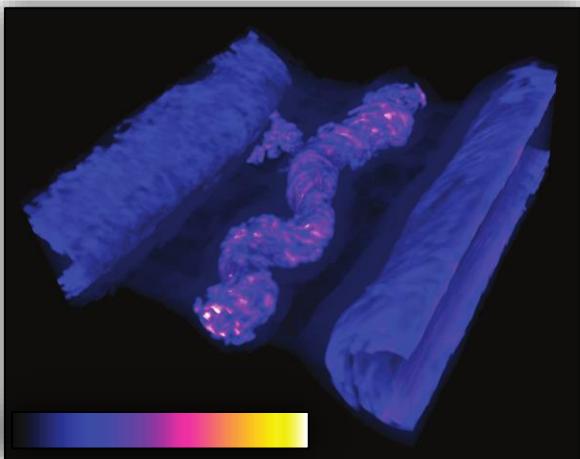
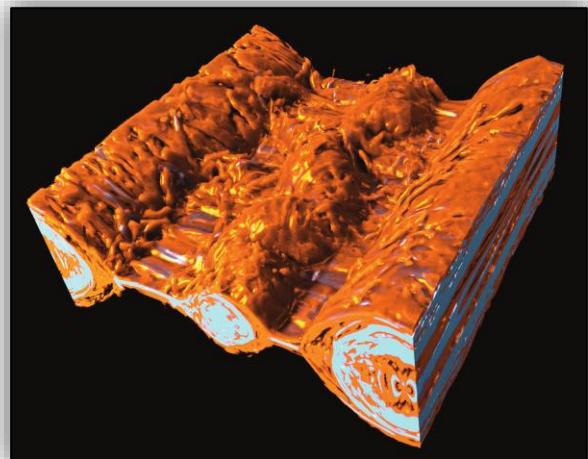


Simulation:
Bill Daughton (LANL), Berk Geveci (KitWare)



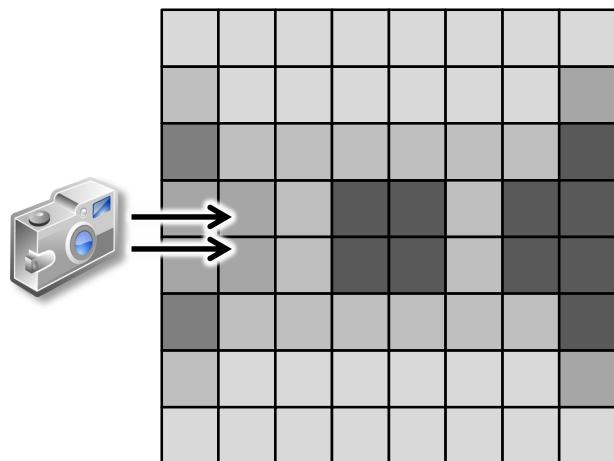
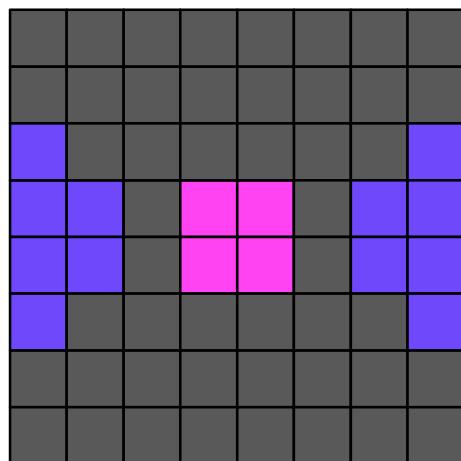
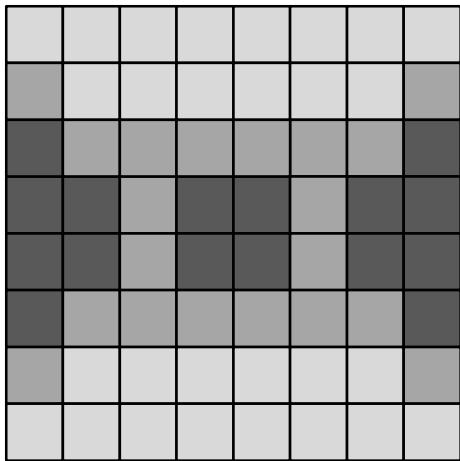
Forschung: Opacity Optimization

Optimierung von Sichtbarkeit und Beleuchtung



Forschung: Opacity Optimization

Optimierung von Sichtbarkeit und Beleuchtung



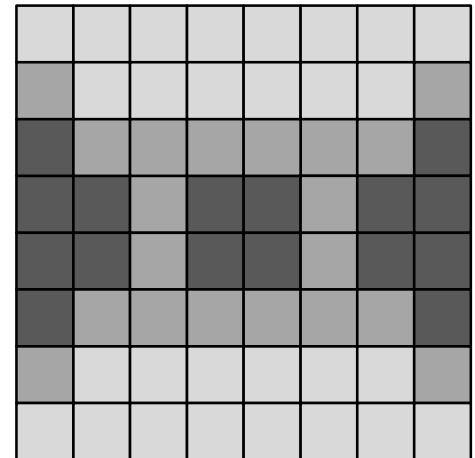
Forschung: Opacity Optimization

Optimierung von Sichtbarkeit und Beleuchtung [Günther et al. 2013]

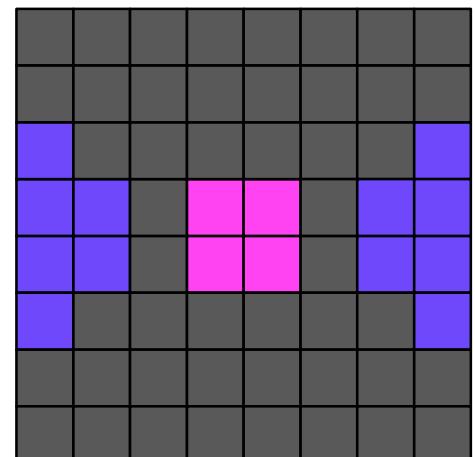
- Wichtigkeit g_i von Zellen modifiziert die Opazität α_i (n = Anz. Zellen!)

$$E(\bar{\alpha}) = p \sum_{i=1}^n (\bar{\alpha}_i - \alpha_i)^2 \quad \bar{\alpha} = (\bar{\alpha}_1, \dots, \bar{\alpha}_n)$$

Opazität α_i



Importance g_i



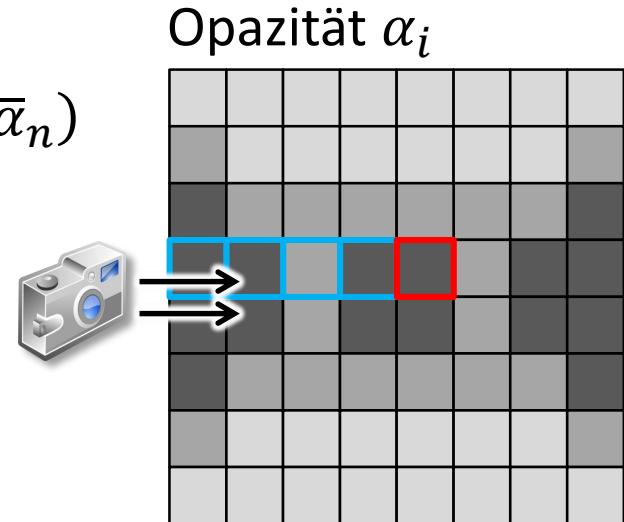
Forschung: Opacity Optimization

Optimierung von Sichtbarkeit und Beleuchtung [Günther et al. 2013]

- Wichtigkeit g_i von Zellen modifiziert die Opazität α_i (n = Anz. Zellen!)

$$E(\bar{\alpha}) = p \sum_{i=1}^n (\bar{\alpha}_i - \alpha_i)^2 \quad \bar{\alpha} = (\bar{\alpha}_1, \dots, \bar{\alpha}_n)$$

$$+ q \sum_{i=1}^n \sum_{j=1}^n (\bar{\alpha}_i (1 - g_i)^\lambda h_{ij} g_j)^2$$



Forschung: Opacity Optimization

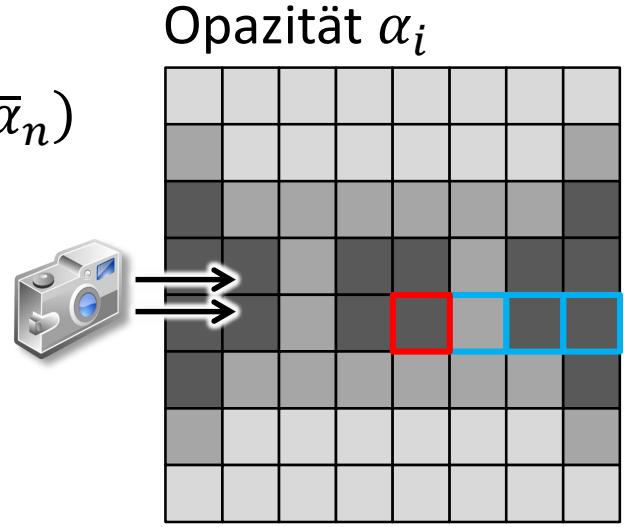
Optimierung von Sichtbarkeit und Beleuchtung [Günther et al. 2013]

- Wichtigkeit g_i von Zellen modifiziert die Opazität α_i (n = Anz. Zellen!)

$$E(\bar{\alpha}) = p \sum_{i=1}^n (\bar{\alpha}_i - \alpha_i)^2 \quad \bar{\alpha} = (\bar{\alpha}_1, \dots, \bar{\alpha}_n)$$

$$+ q \sum_{i=1}^n \sum_{j=1}^n (\bar{\alpha}_i (1 - g_i)^\lambda h_{ij} g_j)^2$$

$$+ r \sum_{i=1}^n \sum_{j=1}^n (\bar{\alpha}_i (1 - g_i)^\lambda h_{ji} g_j)^2$$



Forschung: Opacity Optimization

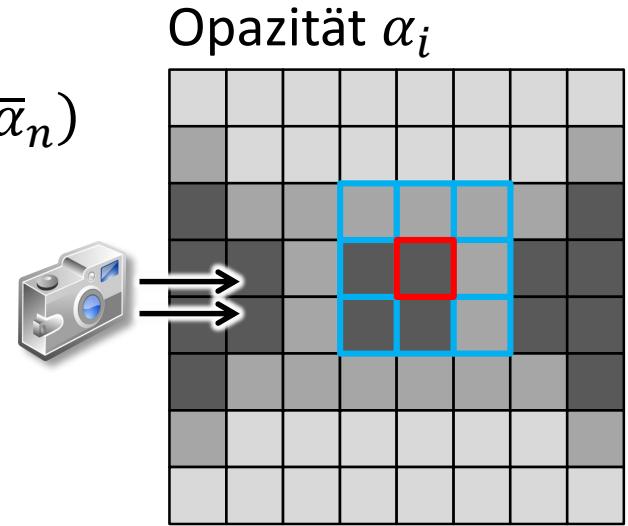
Optimierung von Sichtbarkeit und Beleuchtung [Günther et al. 2013]

- Wichtigkeit g_i von Zellen modifiziert die Opazität α_i (n = Anz. Zellen!)

$$E(\bar{\alpha}) = p \sum_{i=1}^n (\bar{\alpha}_i - \alpha_i)^2 \quad \bar{\alpha} = (\bar{\alpha}_1, \dots, \bar{\alpha}_n)$$

$$+ q \sum_{i=1}^n \sum_{j=1}^n (\bar{\alpha}_i (1 - g_i)^\lambda h_{ij} g_j)^2$$
$$+ r \sum_{i=1}^n \sum_{j=1}^n (\bar{\alpha}_i (1 - g_i)^\lambda h_{ji} g_j)^2$$

$$+ s \sum_{i=1}^n \sum_{j=1}^n \alpha_{ij} (\bar{\alpha}_i - \bar{\alpha}_j)^2$$



- optimiere: $\arg \min E(\bar{\alpha})$
 $\bar{\alpha} \in \mathbb{R}_+^m$

Forschung: Opacity Optimization

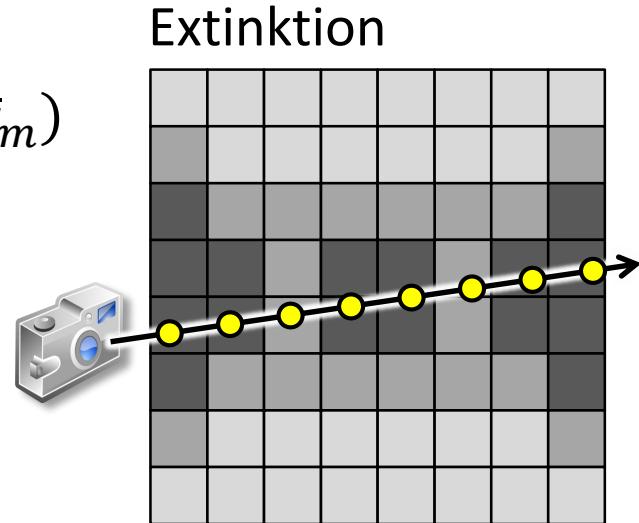
Optimierung von Sichtbarkeit und Beleuchtung

- betrachte zunächst m Stellen entlang eines einzelnen Strahl

$$E(\bar{\sigma}) = p \sum_{i=1}^m (\bar{\sigma}_i - \sigma_i)^2 \quad \bar{\sigma} = (\bar{\sigma}_1, \dots, \bar{\sigma}_m)$$

$$+ q \sum_{i=1}^m \sum_{j=1}^m (\bar{\sigma}_i (1 - g_i)^\lambda h_{ij} g_j)^2$$

$$+ r \sum_{i=1}^m \sum_{j=1}^m (\bar{\sigma}_i (1 - g_i)^\lambda h_{ji} g_j)^2$$



- optimiere: $\arg \min E(\bar{\sigma})$
 $\bar{\sigma} \in \mathbb{R}_+^m$

Extinction-Optimized Volume Illumination

Ament, Zirr, Dachsbaucher

IEEE Trans. on Visualization and Computer Graphics, 2016

Forschung: Opacity Optimization

Optimierung von Sichtbarkeit und Beleuchtung

- Fehler $E(\bar{\sigma}) = \sum_{i=1}^m E_i(\bar{\sigma}_i)$
 → betrachte i -te Stelle entlang des Strahls

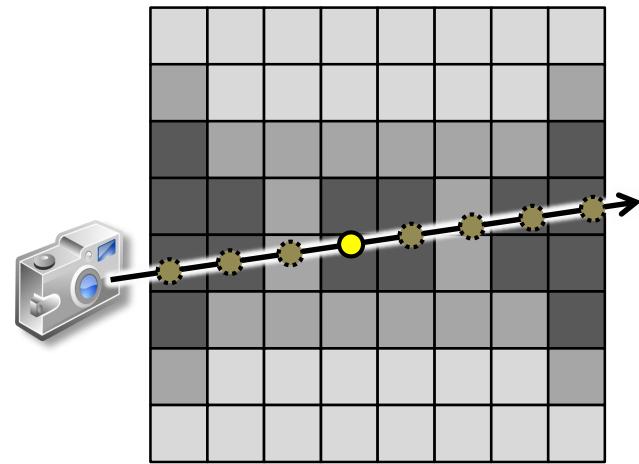
$$E_i(\bar{\sigma}_i) = p(\bar{\sigma}_i - \sigma_i)^2$$

$$+ q \sum_{j=1}^m (\bar{\sigma}_i (1 - g_i)^\lambda h_{ij} g_j)^2$$

$$+ r \sum_{j=1}^m (\bar{\sigma}_i (1 - g_i)^\lambda h_{ji} g_j)^2$$

- optimiere: $\arg \min_{\bar{\sigma}_i \in \mathbb{R}_+} E_i(\bar{\sigma}_i), i \in \{1, \dots, m\}$ und somit $E(\bar{\sigma})$

Extinktion



Forschung: Opacity Optimization

Optimierung von Sichtbarkeit und Beleuchtung

- Fehler $E(\bar{\sigma}) = \sum_{i=1}^m E_i(\bar{\sigma}_i)$
 → betrachte i -te Stelle entlang des Strahls

$$E_i(\bar{\sigma}_i) = p(\bar{\sigma}_i - \sigma_i)^2$$

$$+ q \sum_{j=1}^m (\bar{\sigma}_i (1 - g_i)^\lambda h_{ij} g_j)^2 \quad h_{ij} = \begin{cases} 1 & \text{wenn } j > i \\ 0 & \text{sonst} \end{cases}$$

$$+ r \sum_{j=1}^m (\bar{\sigma}_i (1 - g_i)^\lambda h_{ji} g_j)^2 \quad h_{ji} = \begin{cases} 1 & \text{wenn } j < i \\ 0 & \text{sonst} \end{cases}$$

- optimiere: $\arg \min_{\bar{\sigma}_i \in \mathbb{R}_+} E_i(\bar{\sigma}_i), i \in \{1, \dots, m\}$ und somit $E(\bar{\sigma})$

Forschung: Opacity Optimization

Optimierung von Sichtbarkeit und Beleuchtung

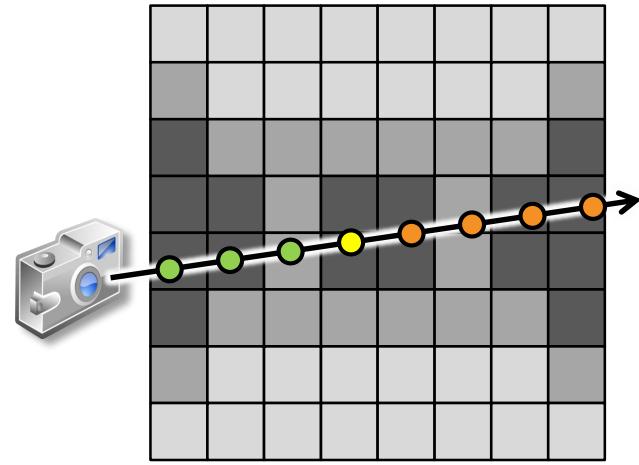
- Fehler $E(\bar{\sigma}) = \sum_{i=1}^m E_i(\bar{\sigma}_i)$
→ betrachte i -te Stelle entlang des Strahls

$$E_i(\bar{\sigma}_i) = p(\bar{\sigma}_i - \sigma_i)^2$$

$$+ q \bar{\sigma}_i^2 (1 - g_i)^{2\lambda} \sum_{j=i+1}^m g_j^2$$

$$+ r \bar{\sigma}_i^2 (1 - g_i)^{2\lambda} \sum_{j=1}^{i-1} g_j^2$$

Extinktion



- optimiere: $\arg \min_{\bar{\sigma}_i \in \mathbb{R}_+} E_i(\bar{\sigma}_i), i \in \{1, \dots, m\}$ und somit $E(\bar{\sigma})$

Extinction-Optimized Volume Illumination

Ament, Zirr, Dachsbaucher

IEEE Trans. on Visualization and Computer Graphics, 2016

Forschung: Opacity Optimization

Optimierung von Sichtbarkeit und Beleuchtung

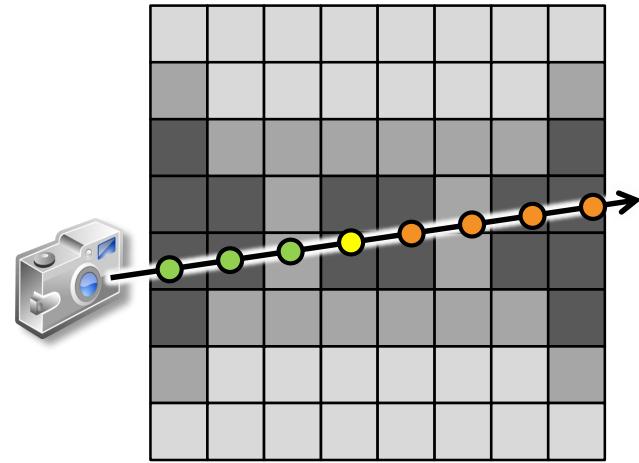
- Fehler $E(\bar{\sigma}) = \sum_{i=1}^m E_i(\bar{\sigma}_i)$
→ betrachte i -te Stelle entlang des Strahls

$$E_i(\bar{\sigma}_i) = p(\bar{\sigma}_i - \sigma_i)^2$$

$$+ q \bar{\sigma}_i^2 (1 - g_i)^{2\lambda} \sum_{j=i+1}^m g_j^2$$

$$+ r \bar{\sigma}_i^2 (1 - g_i)^{2\lambda} \sum_{j=1}^{i-1} g_j^2$$

Extinktion



- optimiere: $\arg \min_{\bar{\sigma}_i \in \mathbb{R}_+} E_i(\bar{\sigma}_i), i \in \{1, \dots, m\}$ und somit $E(\bar{\sigma})$

Extinction-Optimized Volume Illumination

Ament, Zirr, Dachsbaucher

IEEE Trans. on Visualization and Computer Graphics, 2016

Volumenvisualisierung

Optimierung von Sichtbarkeit und Beleuchtung

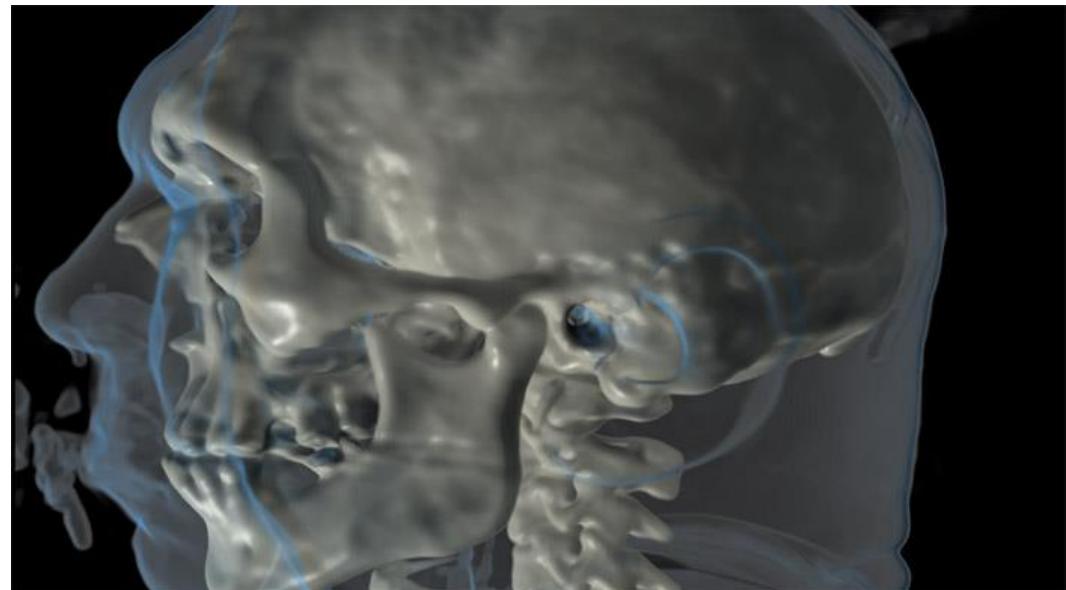
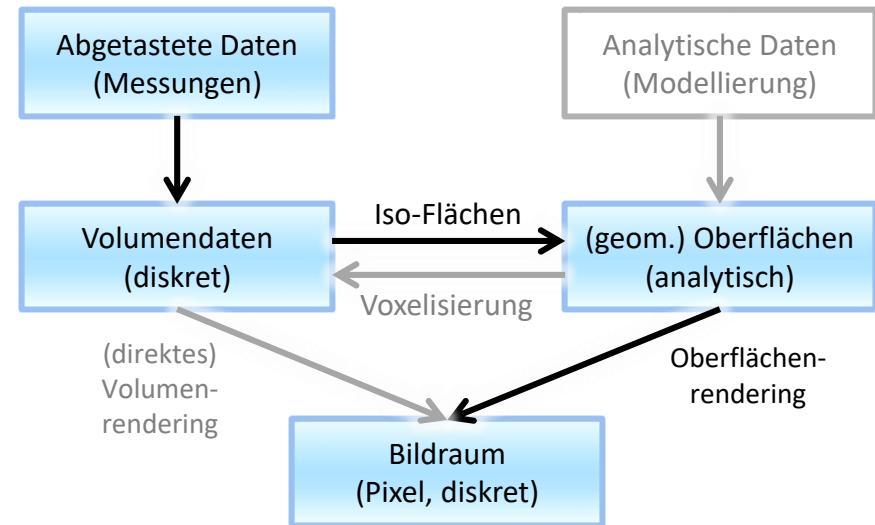
$$E_i(\bar{\sigma}_i) = \left(p + (1 - g_i)^{2\lambda} + \left(r \sum_{j=1}^{i-1} g_j^2 + q \sum_{j=i+1}^m g_j^2 \right) \right) \bar{\sigma}_i^2 - 2p\sigma_i \bar{\sigma}_i + p\sigma_i^2$$

- ▶ räumliche Kohärenz durch Tiefpaßfilterung von g_j
 - ▶ ähnliche Auswirkung wie $s \sum_{i=1}^n \sum_{j=1}^n \alpha_{ij} (\bar{\sigma}_i - \bar{\sigma}_j)^2$
 - ▶ erhält Diskontinuitäten in σ_i
 - ▶ keine ungewollte Glättung in Bereichen mit konstantem g_j

Oberflächen- vs. Volumenrendering

Indirekte Volumenvisualisierung

- ▶ Beispiel
 - ▶ CT-Messung
 - ▶ Iso-Stack-Conversion
 - ▶ Berechnung von Isoflächen (Marching Cubes)
 - ▶ Oberflächenrendering (OpenGL)

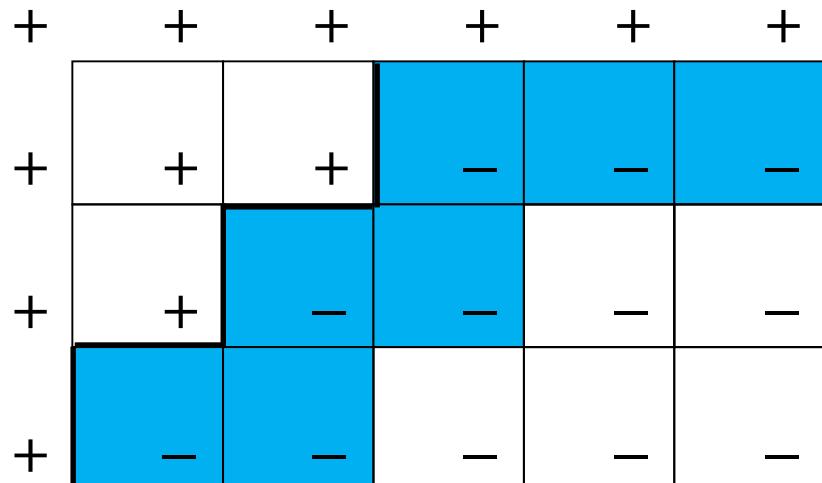


Literatur

- ▶ ***The Visualization Handbook:***
 - ▶ Kap. 2 (Accelerated Isosurface Extraction Techniques)
- ▶ ***The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics***
(2nd edition):
 - ▶ Kap. 6.2 (Fundamental Algorithms – Scalar Algorithms)
 - ▶ Kap. 9.1 (Advanced Algorithms – Scalar Algorithms)

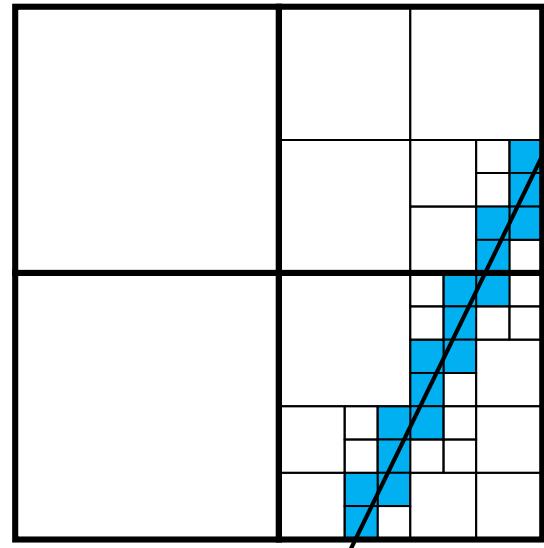
Cuberille Meshes

- Cuberille Ansatz: zeichne Voxel entlang der Isofläche als opake Würfel
[Herman 1979]
- Binarisierung des Volumens: ist der Skalarwert eines Voxels kleiner oder größer als der Isowert?
- bestimme Flächen an der Grenzschicht
 - alle Flächen zw. unterschiedlich klassifizierten Voxel deren Normale zum Betrachter hin und aus der Zelle hinaus zeigt
 - zeichne schattierte Polygone
- Voxel-Sicht auf das Volumen: keine Interpolation in den Zellen



Cuberille Meshes

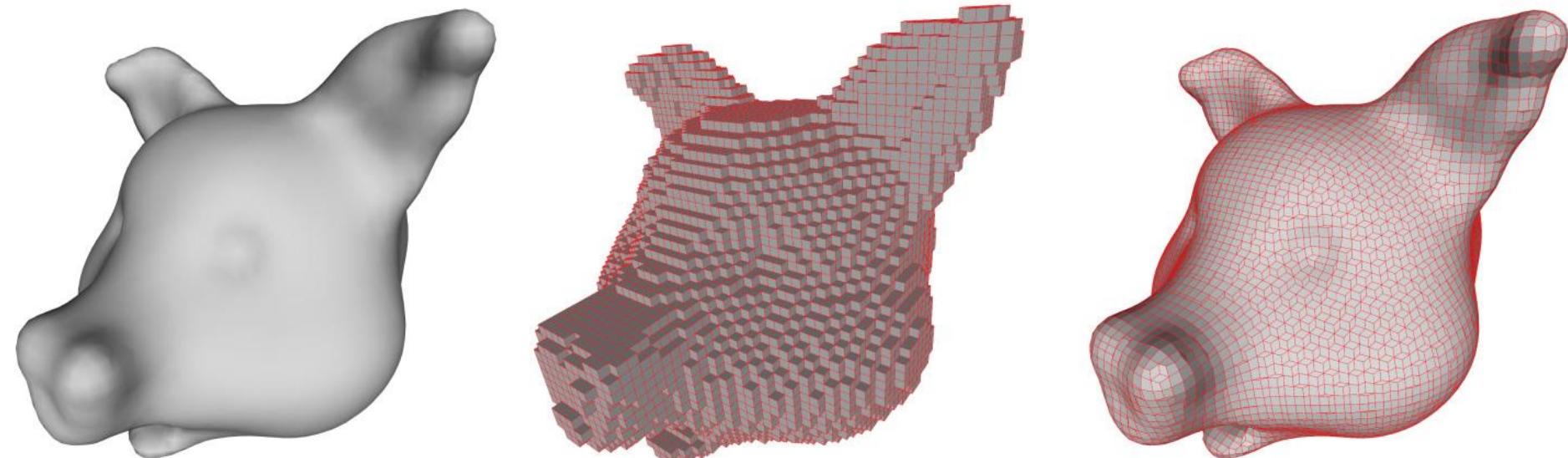
- ▶ Cuberille Ansatz ergibt blockige Oberflächen
- ▶ Verbesserung durch adaptive Unterteilung
 - ▶ Unterteile eine markierte Zelle in 8 kleinere Würfel
 - ▶ bestimme Werte der neuen Zellen durch trilineare Interpolation
 - ▶ wiederhole bis Würfel klein genug (z.B. Pixelgröße)



- ▶ oder: Verbesserung durch Projektion der Vertizes
 - ▶ verschiebe Vertizes (entlang des Gradienten) bis sie auf der Isofläche liegen („hacky“)

Cuberille Meshes

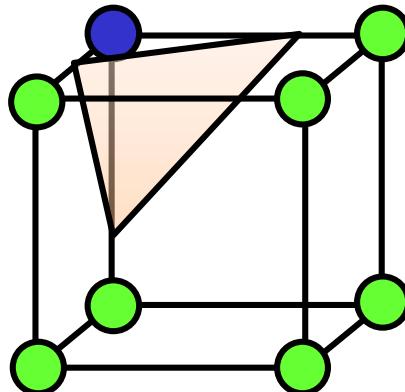
- Beispiel: Isofläche (exakt), Cuberilles, Cuberilles mit Projektion



Bilder: <http://www2.imm.dtu.dk/~jab/gallery/polygonization.html>

Marching Cubes

- ▶ Approximation der Isofläche mit dem Marching-Cubes (MC) Algorithmus
[Lorensen, Cline 1987]
 - ▶ 3D Pendant zum Marching Squares Algorithmus
 - ▶ arbeitet (ohne Unterteilung) direkt mit den Originaldaten
 - ▶ erzeugt direkt ein Dreiecksnetz
 - ▶ Vertex-Positionen durch lineare Interpolation entlang der Zellkanten
 - ▶ Gradienten als Normalenvektoren der Isofläche
 - ▶ Verwendung einer Lookup-Tabelle zur effizienten Berechnung
- ▶ DAS Standardverfahren zur geometriebasierten Isoflächen-Extraktion

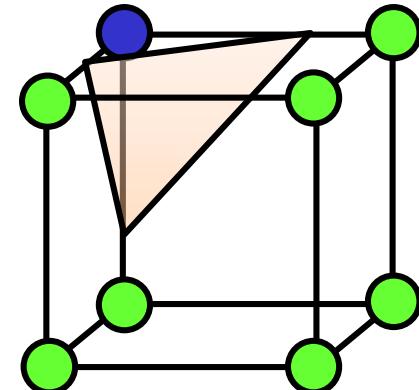


Marching Cubes

Algorithmus

- eine Zelle besteht aus 8 Ecken, mit je einem Skalarwert

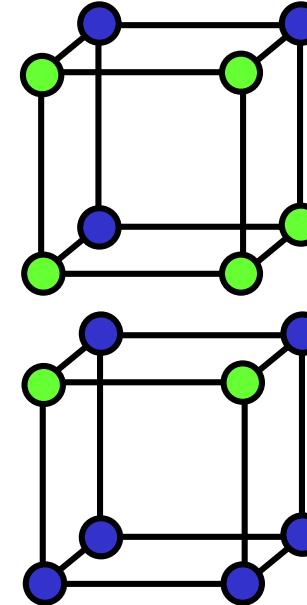
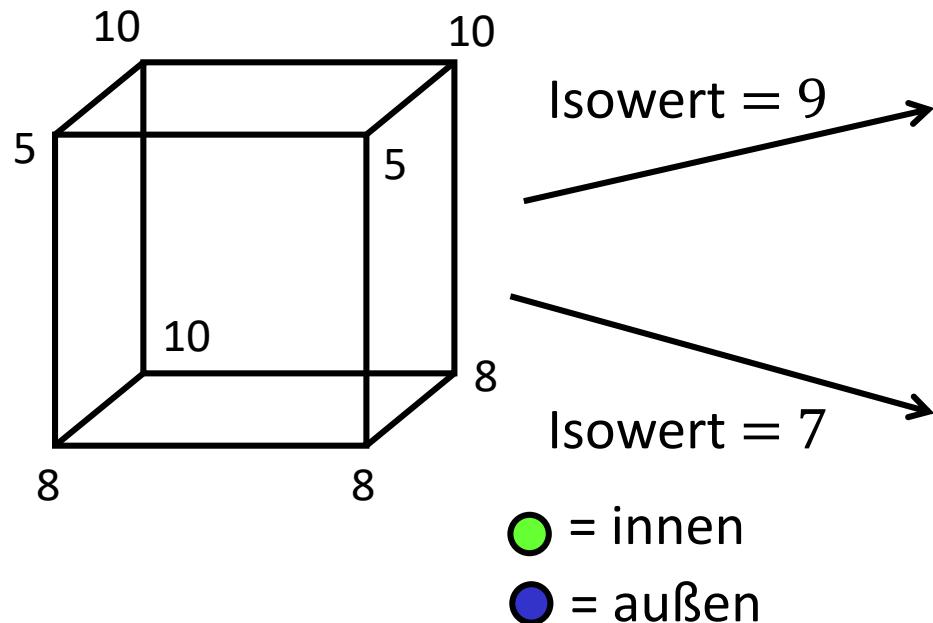
1. betrachte jede Zelle für sich und
klassifiziere jeden Vertex als „innen“ oder „außen“
2. berechne daraus einen Index für die Lookup-Tabelle
3. erhalte Kantenliste aus Tabelle[Index]
4. Interpoliere die Vertizes der Kanten
5. berechne Gradienten
6. behandle uneindeutige Fälle
7. weiter mit nächste Zelle



Marching Cubes

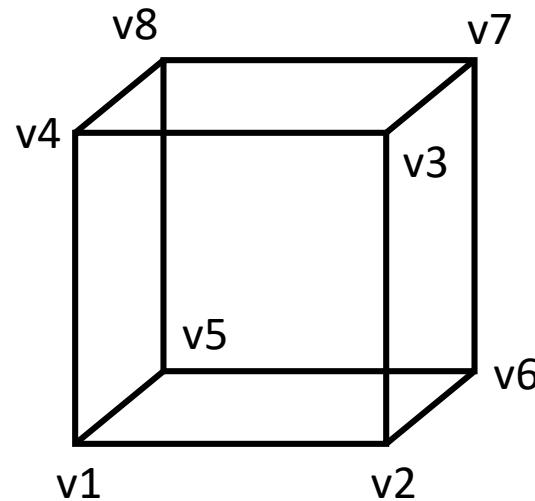
Schritt 1: klassifiziere jeden Vertex als „innen“ oder „außen“

- außen: Skalarwert > Isowert
- innen: Skalarwert \leq Isowert



Marching Cubes

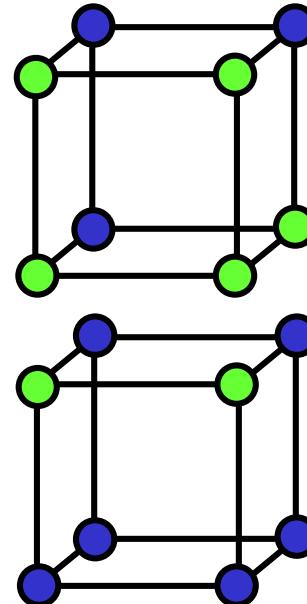
Schritt 2: berechne daraus einen Index für die Lookup-Tabelle



- innen = 1
- außen = 0

Index:

v1	v2	v3	v4	v5	v6	v7	v8
----	----	----	----	----	----	----	----



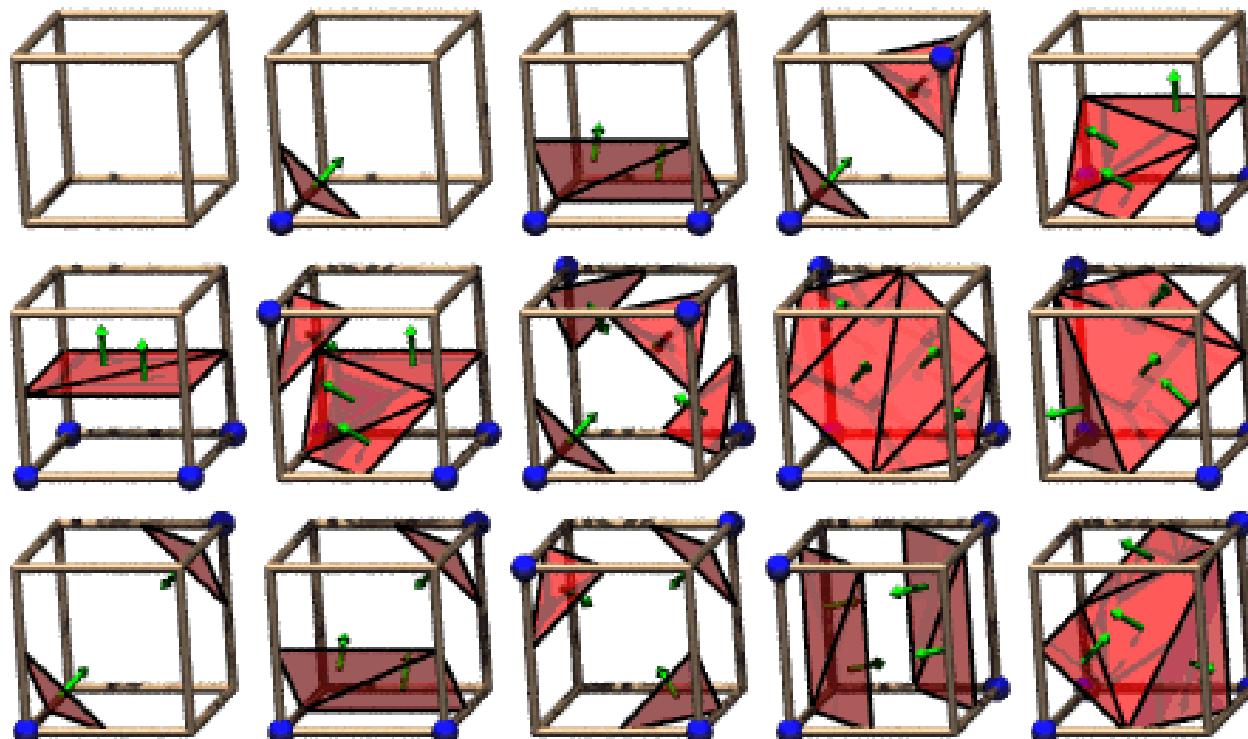
11110100

00110000

Marching Cubes

Schritt 3: verwende Index und erhalte Kantenliste aus einer Tabelle

- es entstehen zwischen 0 und 5 Dreiecke pro Zelle
- alle 256 Fälle können durch Symmetrien aus den 15 Basisfällen erhalten werden:

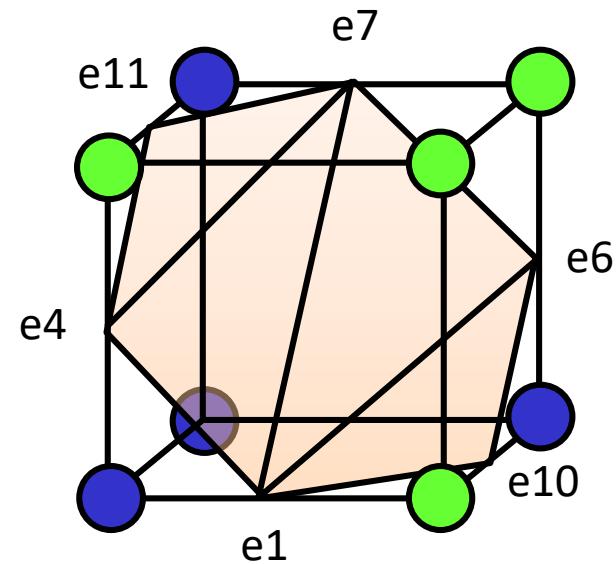


Marching Cubes

Schritt 3: verwende Index und erhalte Kantenliste aus einer Tabelle

► Beispiel:

- Index = 10110001
- Lookup-Tabelle liefert:
erzeuge 4 Dreiecke mit den Vertizes
auf den folgenden Kanten
 - Dreieck 1 = e4, e7, e11
 - Dreieck 2 = e1, e7, e4
 - Dreieck 3 = e1, e6, e7
 - Dreieck 4 = e1, e10, e6



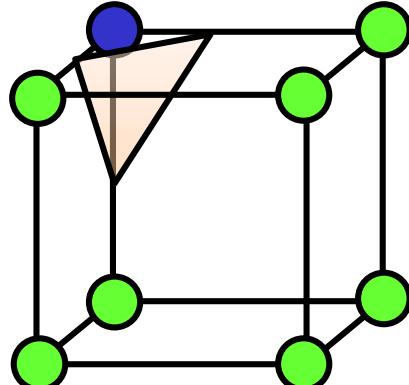
Marching Cubes

Schritt 4: Bestimmen der Vertexpositionen

- ▶ Interpoliere die Vertizes der Kanten durch lineare Interpolation der Skalarwerte an den Zellenecken

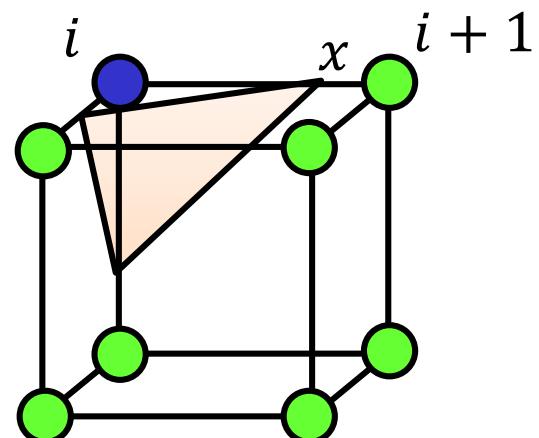
$$x = i + \left(\frac{\text{Isowert} - v[i]}{v[i+1] - v[i]} \right)$$

Isowert = 5



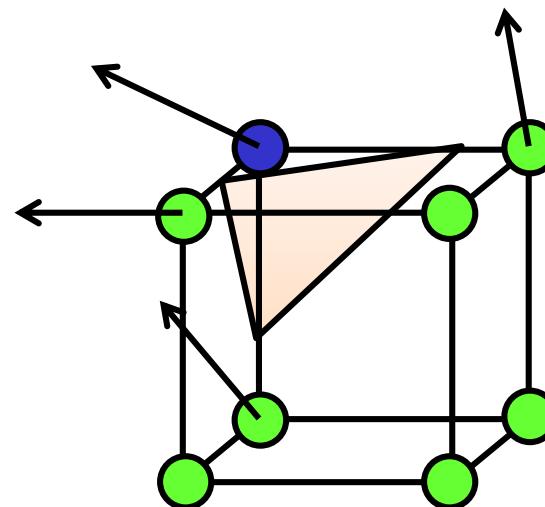
$$\begin{aligned} \text{green circle} &= 10 \\ \text{blue circle} &= 0 \end{aligned}$$

Isowert = 8



Schritt 5: Normalenberechnung

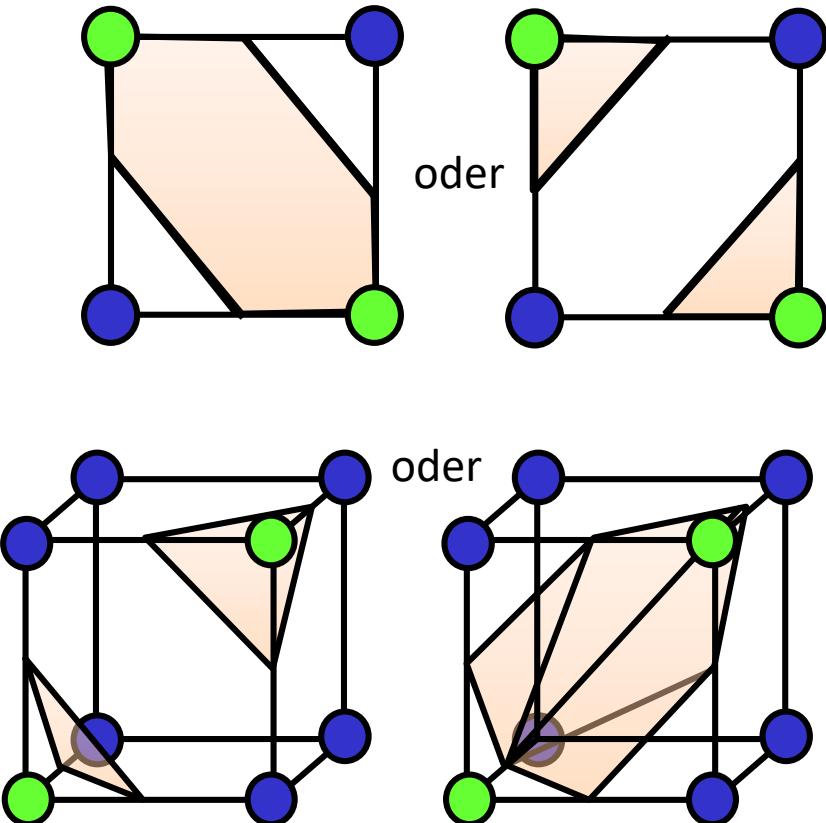
- ▶ berechne zuerst die Normale/Gradient an den Zellenecken (z.B. über zentrale Differenzen)
- ▶ verwende lineare Interpolation zur Berechnung der Vertex-Normalen der Isofläche
- ▶ NICHT die Normalen aus der extrahierten Iso-Fläche bestimmen



Marching Cubes

Schritt 6: uneindeutige Fälle

- ▶ Fälle: 3, 6, 7, 10, 12, 13
- ▶ immer dann, wenn
 - ▶ benachbarte Vertizes: unterschiedliche Klassifikation
 - ▶ diagonale Vertizes: gleiche Klassifikation
- ▶ Löcher in der Isofläche entstehen, wenn diese Fälle nicht richtig behandelt werden
- ▶ Lösung: Asymptotic Decider¹ (siehe Marching Squares)



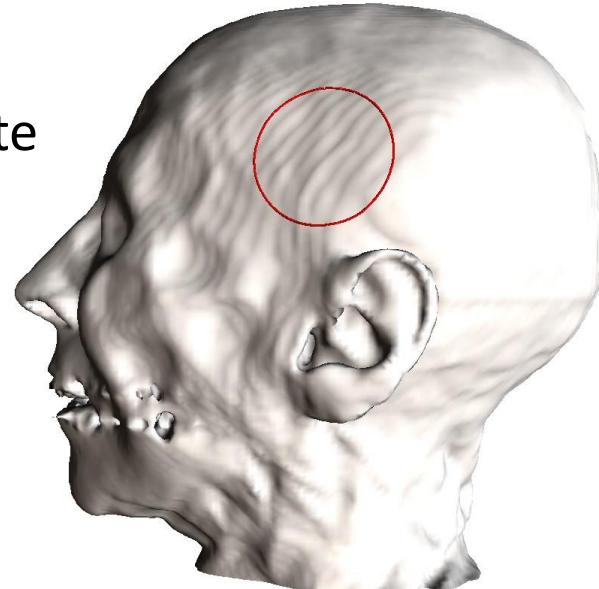
¹The asymptotic decider: resolving the ambiguity in marching cubes
Gregory M. Nielson, Bernd Hamann, VIS'91

Marching Cubes

Zusammenfassung

- ▶ 256 Fälle, „nur“ 15 bei Berücksichtigung von Symmetrien
- ▶ es entstehen bis zu 5 Dreiecke pro Zelle
- ▶ wenn mehrere Isoflächen dargestellt werden sollen
 - ▶ mehrfache Ausführung von MC
 - ▶ Semitransparenz erfordert Sortierung
- ▶ Achtung: MC ist im Prinzip eine Abtastung der Isofläche
 - ▶ Vorsicht mit Strukturen in der Größe der Gitterauflösung!
- ▶ es gibt noch ein paar Details, z.B. eine verbesserte Lookup-Tabelle ohne Mehrdeutigkeiten (aber einige Fälle sind nicht mehr symmetrisch)

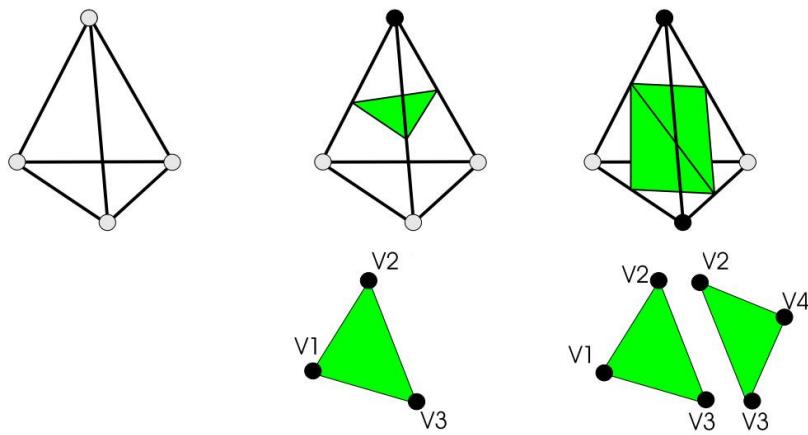
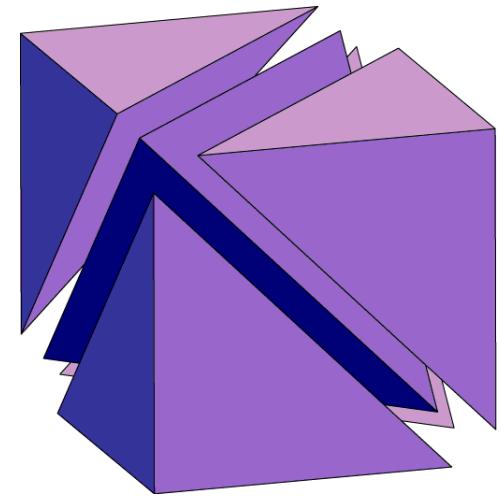
[Claudio Montani, Riccardo Scateni, Roberto Scopinio,
„A Modified Look-up Table for implicit disambiguation
of Marching Cubes,“ The Visual Computer, Vol. 10, No. 6,
1994, 353-335]



Marching Tetrahedra

- ▶ hauptsächlich verwendet für unstrukturierte Gitter
 - ▶ jeder Zelltyp kann in Tetraeder aufgeteilt werden

- ▶ verarbeite jeden Tetraeder (analog zu MC)
 - ▶ es gibt aber nur 2 interessante Fälle:
 - ▶ 1× „-“ und 3× „+“ (oder umgekehrt):
es wird ein Dreieck erzeugt
 - ▶ 2× „-“ und 2× „+“: Schnitt mit der Isofläche ergibt ein Viereck:
unterteile es in 2 Dreiecke entlang der kürzeren Diagonale



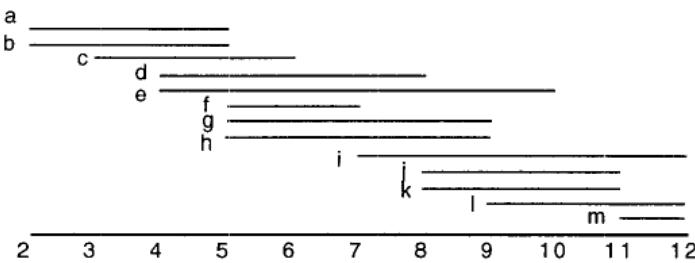
Eigenschaften

- ▶ deutlich weniger Fälle (3 statt 15, mit Symmetrien insgesamt 16)
 - ▶ ebenfalls lineare Interpolation in einer Zelle
 - ▶ keine Probleme mit der Konsistenz zw. benachbarten Zellen
 - ▶ wenn strukturierte Gitter in Tetraeder zerlegt werden entstehen deutlich mehr Dreiecke als bei Marching Cubes
-
- ▶ Verbesserungen (gilt ebenso für MC)
 - ▶ hierarchische oder blickpunktabhängige Oberflächenrekonstruktion
 - ▶ nachträgliche Vereinfachung der Netze (Mesh Decimation)
 - ▶ moderne Varianten: GPUs, Geometry Shader, Transform Feedback

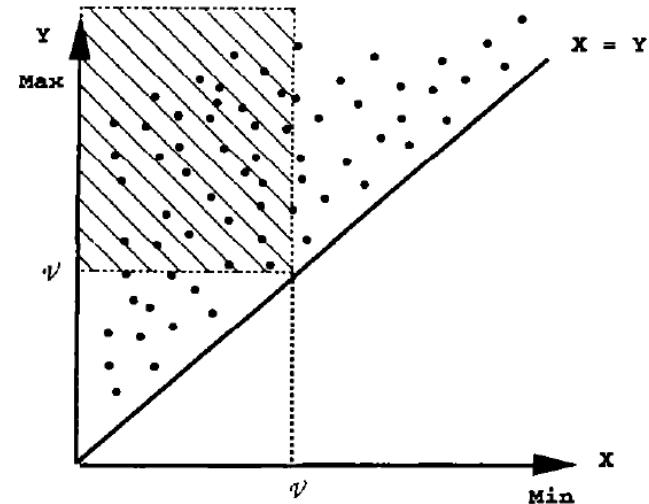
Beschleunigung der Isoflächenextraktion

Range Queries

- Ziel: schnelles Auffinden der Zellen, die die Isofläche schneiden
- wir verwenden eine Datenstruktur die auf den Skalarwerten basiert (keine räumliche Datenstruktur)
- Idee: betrachte Minimum und Maximum der Skalarwerte für jede Zelle (und organisiere die Intervalle dann in einer speziellen Datenstruktur)



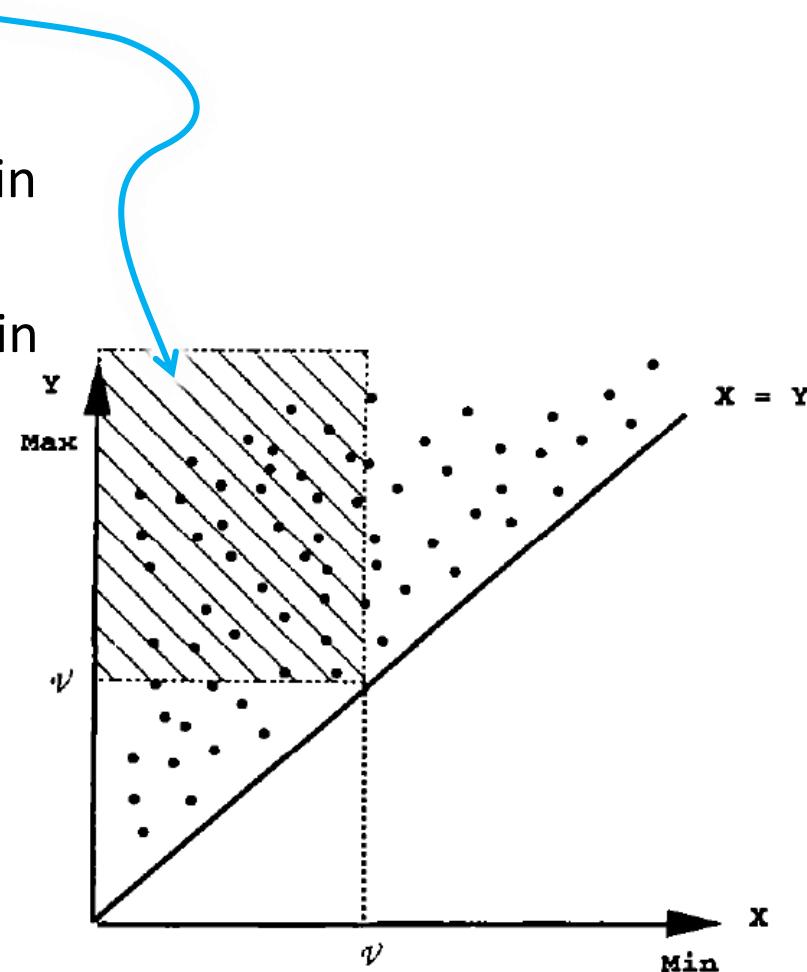
Min/Max-Intervalle für
13 Zellen (a bis m)



Span Space:
jedes Intervall entspricht einem Punkt

Range Queries für Isoflächenextraktion

- ▶ jeder Punkt im Span Space entspricht einer Zelle (bzw. ihrem minimalen und maximalen Skalarwert)
- ▶ die – für einen Isowert v – relevanten Zellen befinden sich in einem rechteckigen Bereich des Span Space
 - ▶ Intervalle der Zellen: $[a_i; b_i]$
 - ▶ das Minimum a_i muss kleiner als v sein
→ rechte Grenze
 - ▶ das Maximum b_i muss größer als v sein
→ untere Grenze
- ▶ wir findet man diese Zellen effizient?



Range Queries für Isoflächenextraktion

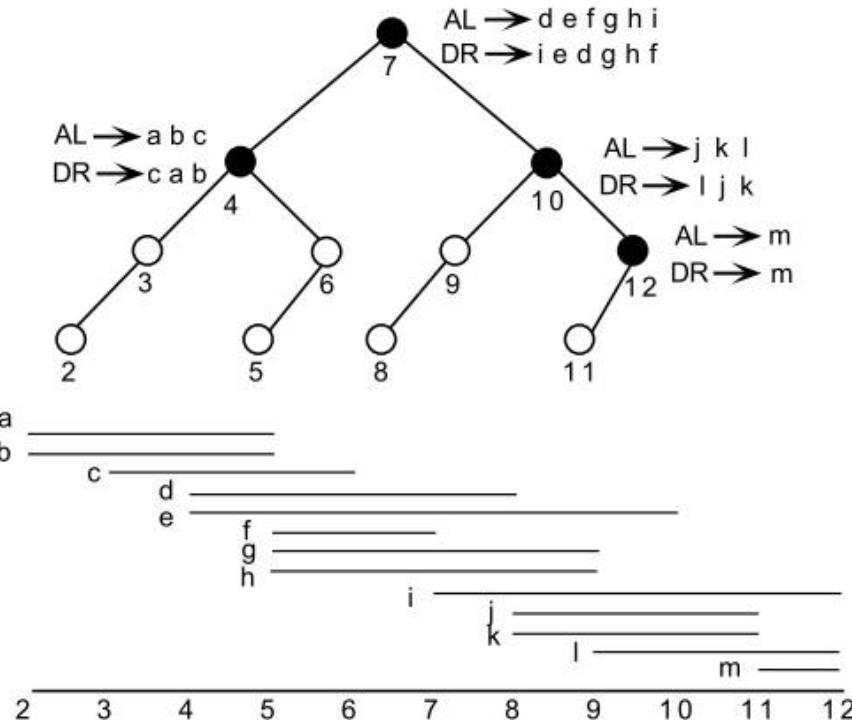
- „Optimal Isosurface Extraction from Irregular Volume Data“

[Cignoni et al. 1996] basierend auf

- Interval Trees [Edelsbrunner 1980]

- balancierter Baum mit h Extrema
(Minima und Maxima) → Höhe $\log h$

- jeder **Knoten** speichert
 - diskriminanten Skalarwert δ : gewählt so, dass er die Menge der Extrema in zwei gleichgroße Gruppen unterteilt

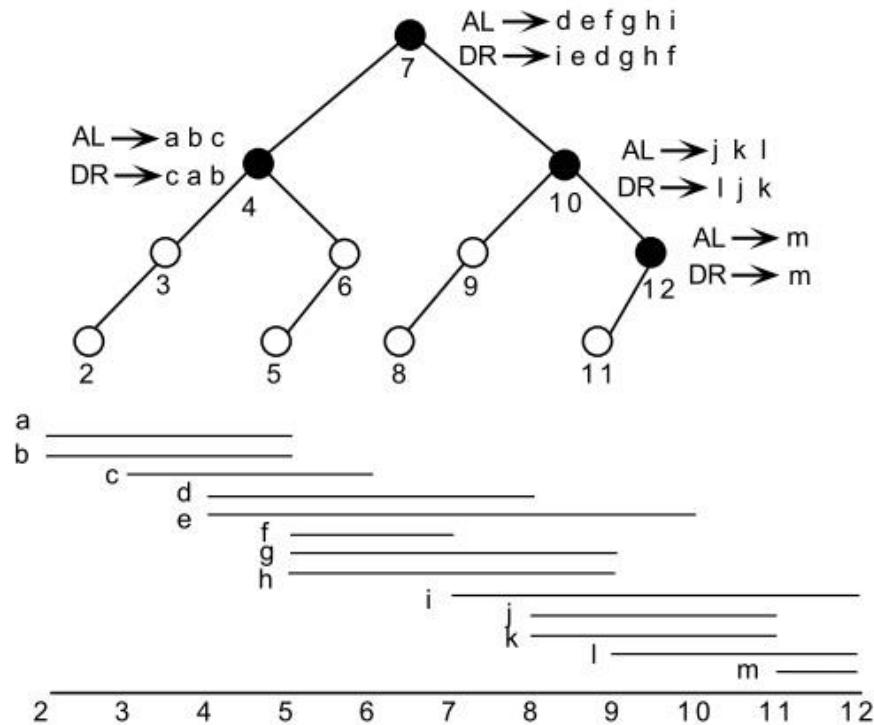


Liste aller Extrema (Minima und Maxima)
anhand derer der Binärbaum konstruiert wird:
 $\{2,3,4,5,6,7,8,9,10,11,12\}$

Range Queries für Isoflächenextraktion

Interval Trees [Edelsbrunner 1980] [Cignoni et al. 1996]

- ▶ jeder **Knoten** speichert
 - ▶ diskriminanten Skalarwert δ
 - ▶ Liste AL (ascending left): alle Intervalle $a_i \leq \delta \leq b_i$, sortiert nach aufsteigendem Minimum
 - ▶ Liste DR (descending right): Intervalle $a_i \leq \delta \leq b_i$ sortiert nach absteigendem Maximum
- ▶ Intervalle $b_i < \delta$ werden im linken (bzw. $a_i > \delta$ im rechten) Teilbaum gespeichert \leftarrow diese Intervalle sind nicht in AL/DR

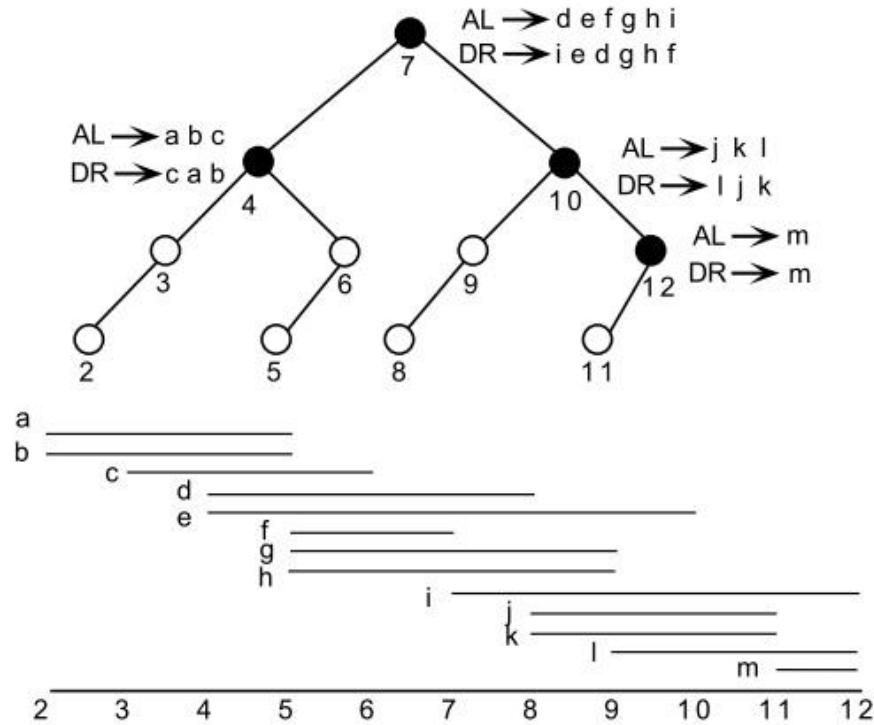


Liste aller Extrema (Minima und Maxima) anhand derer der Binärbaum konstruiert wird:
 $\{2,3,4,5,6,7,8,9,10,11,12\}$

Range Queries für Isoflächenextraktion

Interval Trees [Edelsbrunner 1980] [Cignoni et al. 1996]

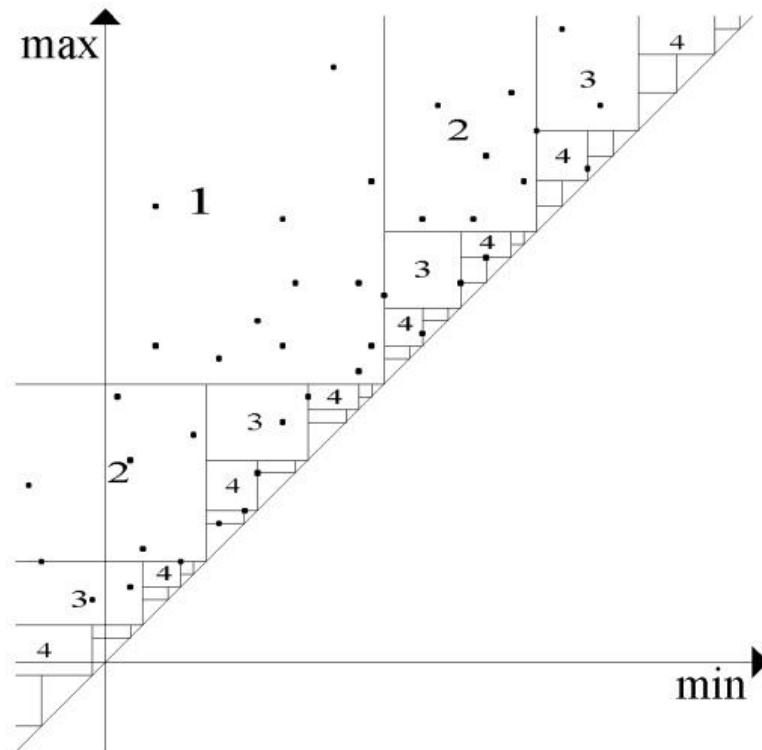
- ▶ Laufzeit $O(k + \log h)$, Traversierung des Baums: $\log h$ (=Höhe)
- ▶ rekursive Traversierung der Knoten:
 - ▶ wenn Isowert $v < \delta$:
gebe vorderste Elemente von AL zurück bis $a_i > v$,
traversiere linken Teilbaum
 - ▶ wenn $v > \delta$:
gebe vorderste Elemente von DR zurück bis $b_i < v$,
traversiere rechten Teilbaum
 - ▶ wenn $v = \delta$:
gebe AL (oder DL) zurück
 - ▶ wenn k Intervalle als Ausgabe zurückgegeben werden, dann werden genau $(k + 1)$ bei der Suche (in AL bzw. DL) betrachtet,
d.h. Aufwand ist ausgabesensitiv



Range Queries für Isoflächenextraktion

Interval Trees [Edelsbrunner 1980] [Cignoni et al. 1996]

- ▶ Laufzeit $O(k + \log h)$, Traversierung des Baums: $\log h$ (=Höhe)
- ▶ rekursive Traversierung der Knoten:
 - ▶ wenn Isowert $v < \delta$:
gebe vorderste Elemente
von AL zurück bis $a_i > v$,
traversiere linken Teilbaum
 - ▶ wenn $v > \delta$:
gebe vorderste Elemente
von DR zurück bis $b_i < v$,
traversiere rechten Teilbaum
 - ▶ wenn $v = \delta$:
gebe AL (oder DL) zurück
 - ▶ wenn k Intervalle als Ausgabe
zurückgegeben werden, dann werden
genau $(k + 1)$ bei der Suche
(in AL bzw. DL) betrachtet,
d.h. Aufwand ist ausgabesensitiv



Darstellung eines Intervall Trees
im Span Space (Zahl = Ebene im Baum)