Volume Visualization Mapping

Content

Computer Graphics and Visualization

- Data Preparation
  - Reconstruction
  - Tetrahedral meshes
  - Filtering
- Indirect Volume Visualization
  - Slicing
  - Contouring
- Direct Volume Visualization
  - Compositing
  - Volume Rendering Integral
  - Transfer Functions & Pre-Integration
  - Rendering Algorithms
  - Continuous Histograms & Scatter Plots
  - Multi-Dimensional Transfer Functions

Indirect Volume Visualization

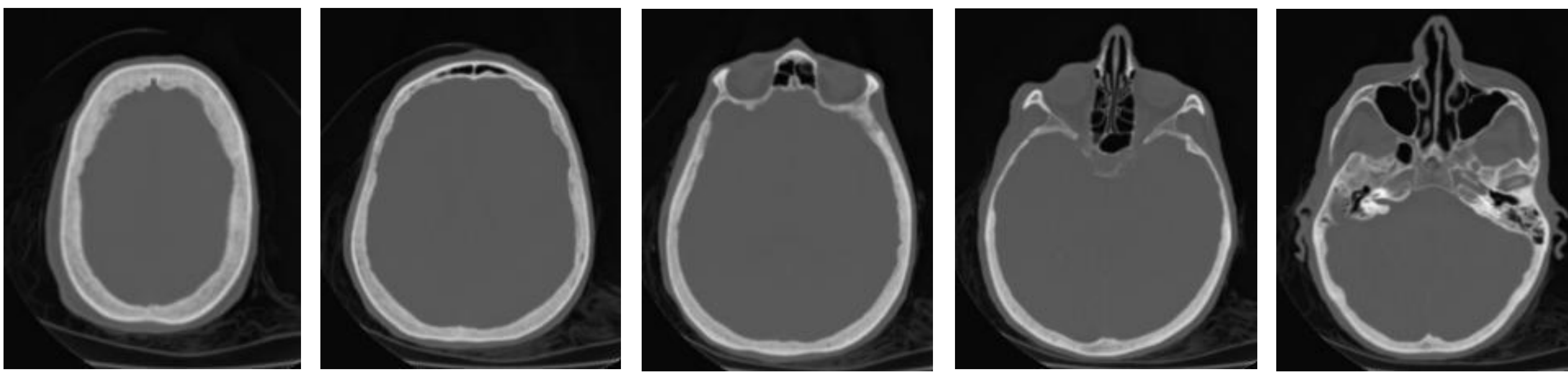
SLICING

Sliced Image Ackquisition

- Voxel datasets in TIFF or DICOM format are organized in image stacks of slices orthogonal to z
- In memory one linearizes the three indices i,j,k of the x,y,z direction to single index I:

$$I = i + j \cdot n_x + k \cdot n_x \cdot n_y$$

- The slice distance in physical space is typically different from the pixel distance inside a slice
- physicians often work directly on 2D visualization of the slices



Slice 20 30 40 50 60

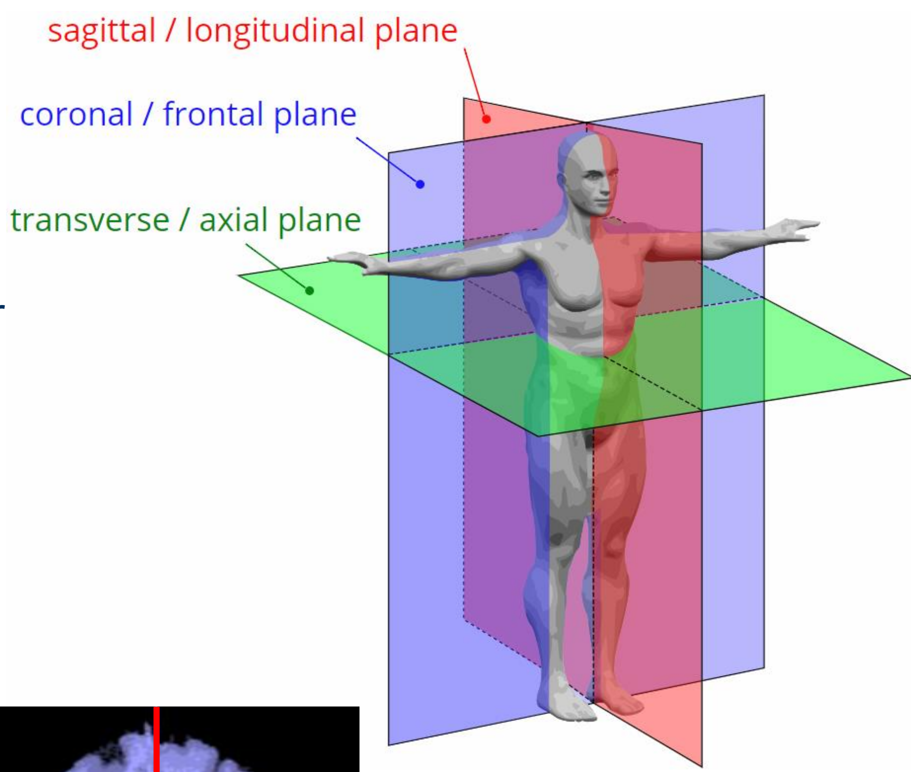
CT data set

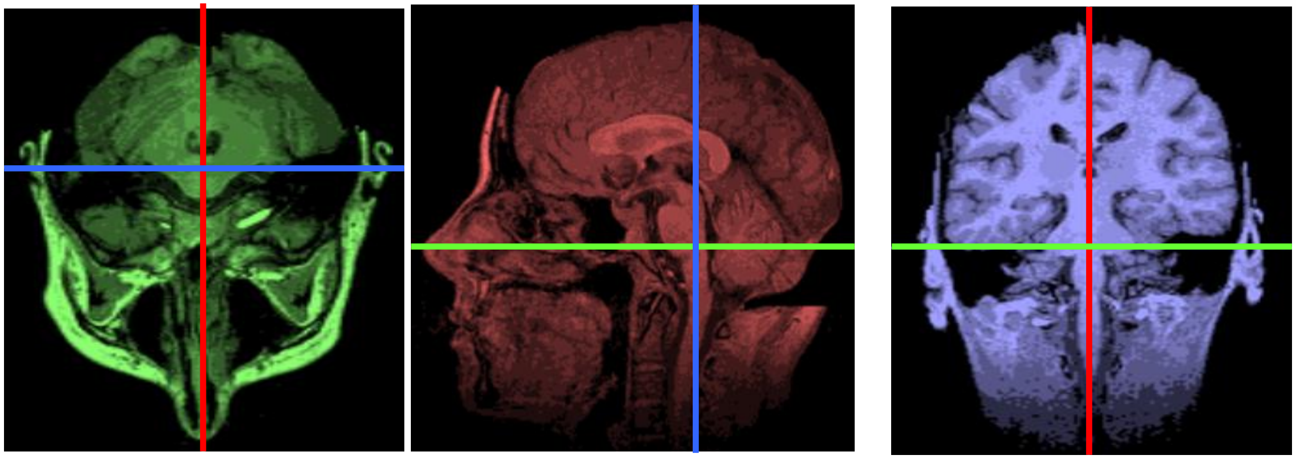
Orthogonal Slicing

- for slicing along planes orthogonal to the main axes x,y,z the voxel values S_{ijk} are permuted
- The pixels $X_{\hat{\imath}\hat{\jmath}}$ of slice $i = i_0$ with $\hat{\imath} = 0 \ldots n_y - 1$ and $\hat{\jmath} = 0 \ldots n_z - 1$ are for example computed from:

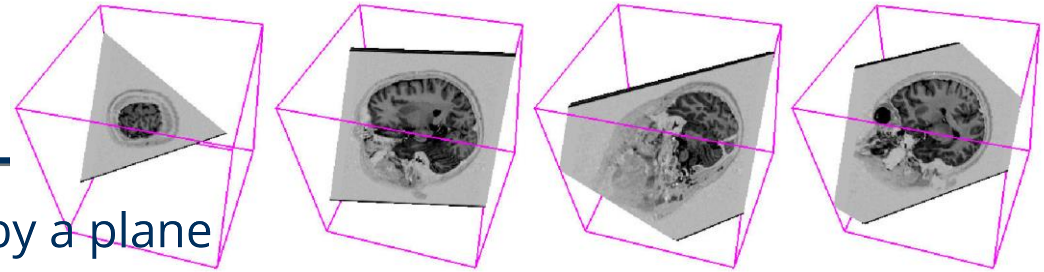
$$X_{\hat{\imath}\hat{\jmath}} = X\left[\hat{I} = \hat{\imath} + n_y \cdot \hat{\jmath}\right]$$
$$= S\left[I = i_0 + \hat{\imath} \cdot n_x + \hat{\jmath} \cdot n_x \cdot n_y\right] = S_{i_0\hat{\imath}\hat{\jmath}}$$

- Often three orthogonal slices around reference point are shown together

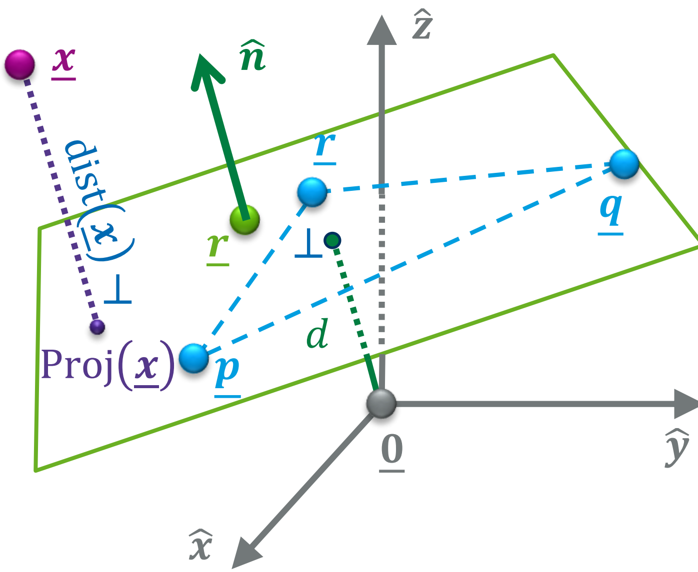




Oblique Slicing



- An oblique Slicing is defined by a plane
- A plane can be defined by three points $\underline{\boldsymbol{p}}$, $\underline{\boldsymbol{q}}$, $\underline{\boldsymbol{r}}$ or by a plane normal $\hat{\boldsymbol{n}}$ and the signed orthogonal distance d of the plane from the origin $\underline{\mathbf{0}}$ of the coordinate system ($d > 0$ if $\underline{\mathbf{0}}$ is on opposite side of plane as $\hat{\boldsymbol{n}}$)



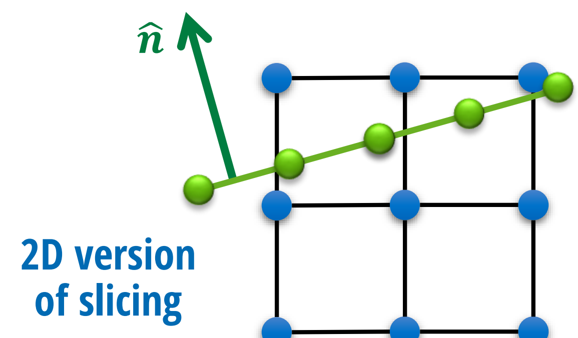
- For a given point $\underline{\boldsymbol{x}}$ we can compute its signed distance from the plane according to

$$\mathrm{dist}(\underline{\boldsymbol{x}}) = \langle \hat{\boldsymbol{n}}, \underline{\boldsymbol{x}} \rangle - d$$

- $\mathrm{dist}(\underline{\boldsymbol{x}})$ is 0 if $\underline{\boldsymbol{x}}$ is on plane, <0 / >0 if it is on opposite / on same side as $\hat{\boldsymbol{n}}$
- We can project orthogonally onto the plane:

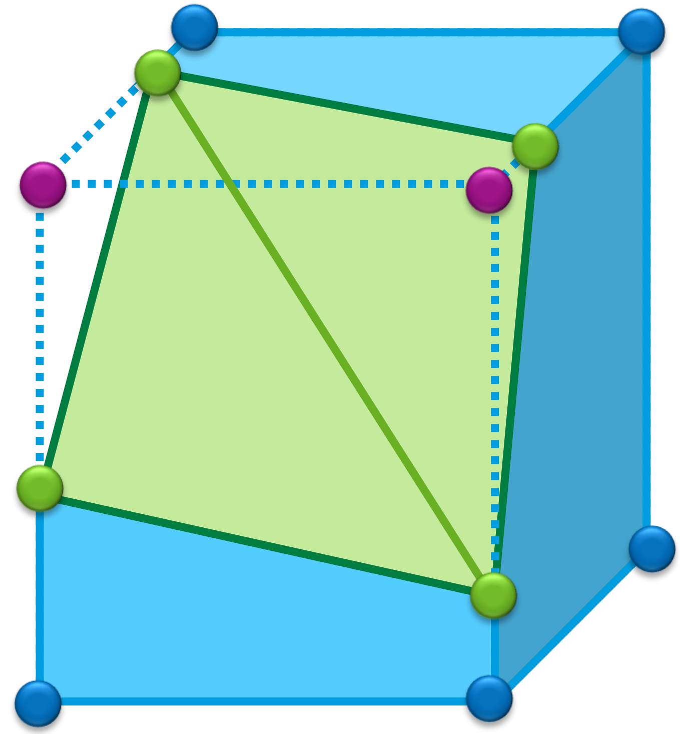
$$\mathrm{Proj}(\underline{\boldsymbol{x}}) = \underline{\boldsymbol{x}} - \mathrm{dist}(\underline{\boldsymbol{x}}) \cdot \hat{\boldsymbol{n}}$$

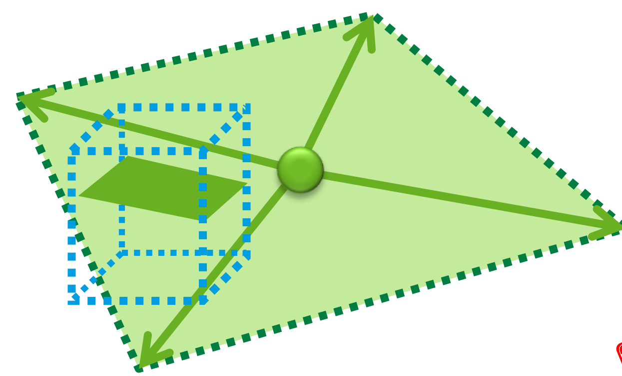
- To slice a voxel grid, interpolation (trilinear or cubic) is needed



Rendering Oblique Slices

- Rendering of oblique slices through regular voxel grids can be implemented with 3D texture mapping, along two approaches
- CPU approach:
  - compute intersection polygon of plane with volume box:
    - use distance function to classify box corners in inside and outside
    - construct edge point on each edge connecting differently classified corners
    - arrange edge points along face adjacencies
  - render resulting polygon as triangle fan with texture coordinates and 3D texturing
- GPU approach: tessellate infinite plane and use the clipping functionality of the GPU, with 6 clipping planes set to the sides of the volume box



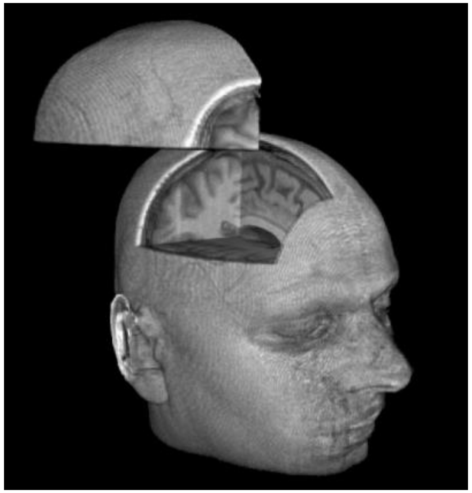


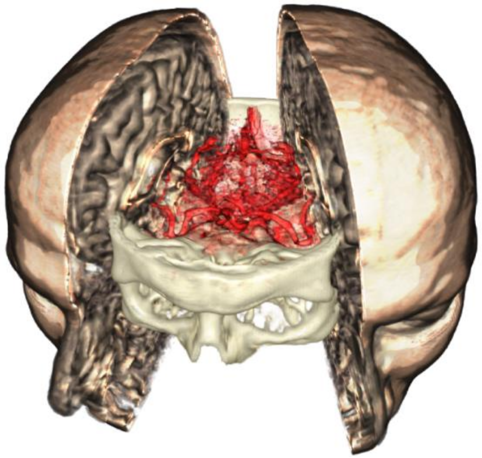
details TOP SECRET

Cutting

Planes can be used for cutting to

- cut away parts of the volume
- to split the volume into several parts and transform the parts individually
- to switch rendering styles, e.g. iso-surface on one side and direct volume rendering on the other side





Indirect Volume Visualization

CONTOURING

Contouring – Motivation

- In volume contouring we want to extract surfaces that separate different materials
- We can define different entities:
  - iso-surfaces from an iso-value S_0: $\forall(x,y,z): S(x,y,z) = S_0$
  - iso-bands from two iso-values S_0 and S_1: $\forall(x,y,z): S_0 \leq S(x,y,z) \leq S_1$
  - volume segments on labeled data composed of all grid faces where one adjacent voxel belongs to the segment and the other not

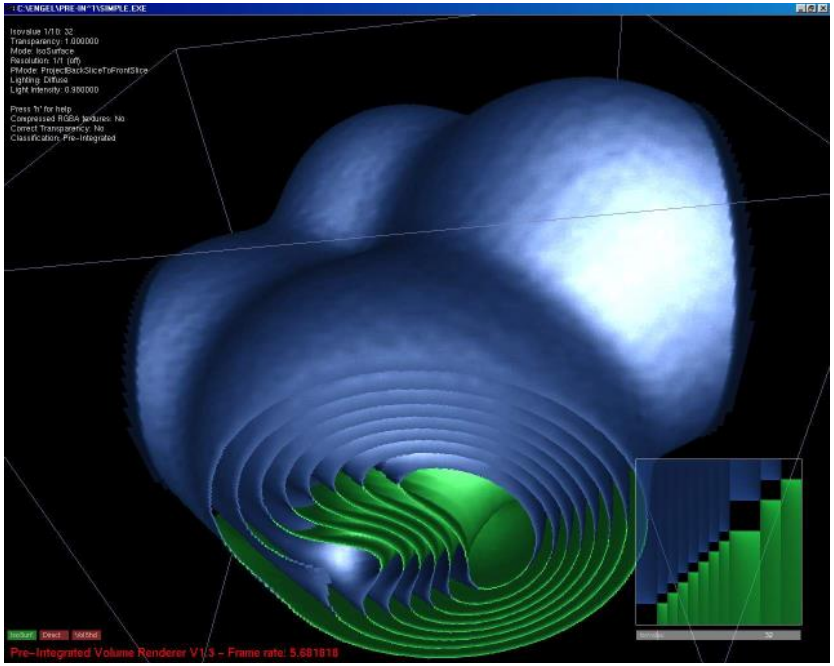
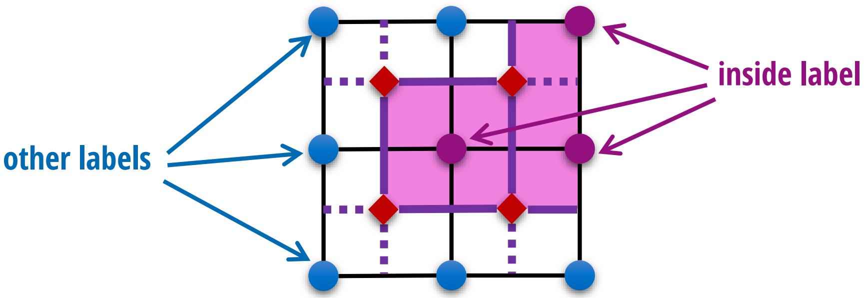
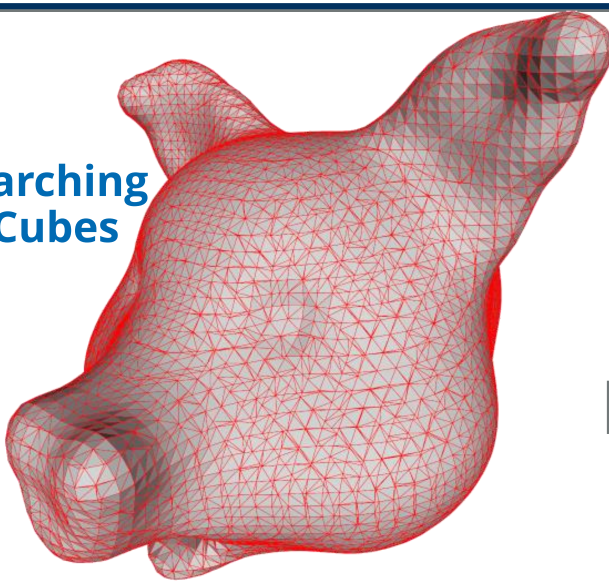


Image: multiple iso-surfaces

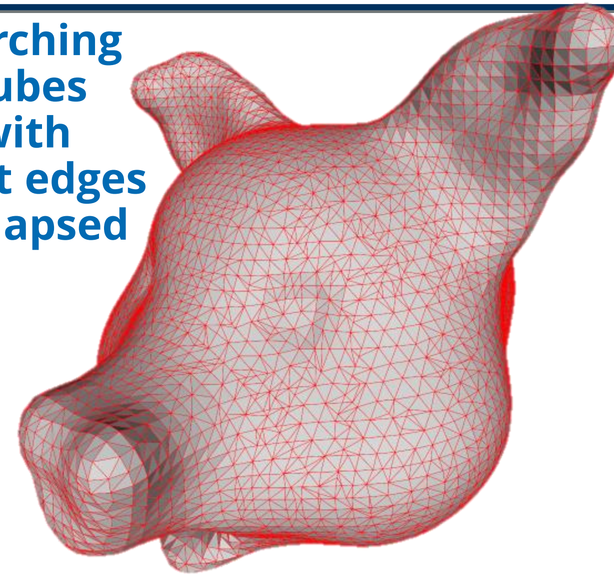


Contouring – Method Comparison

Marching Cubes

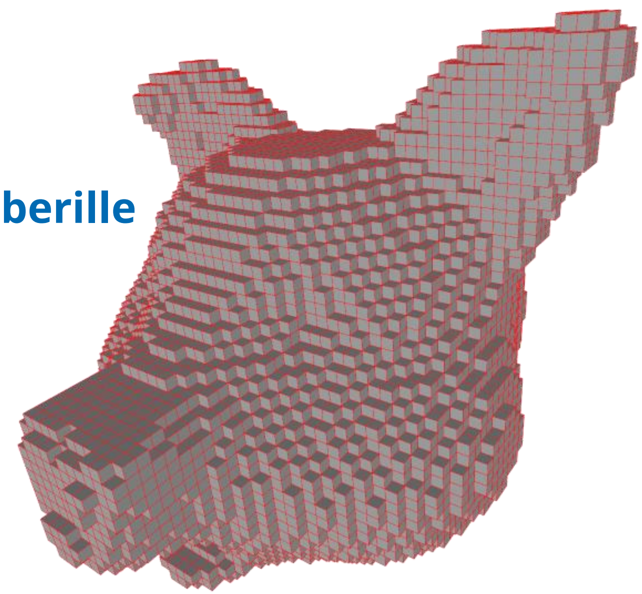


Marching Cubes with short edges collapsed

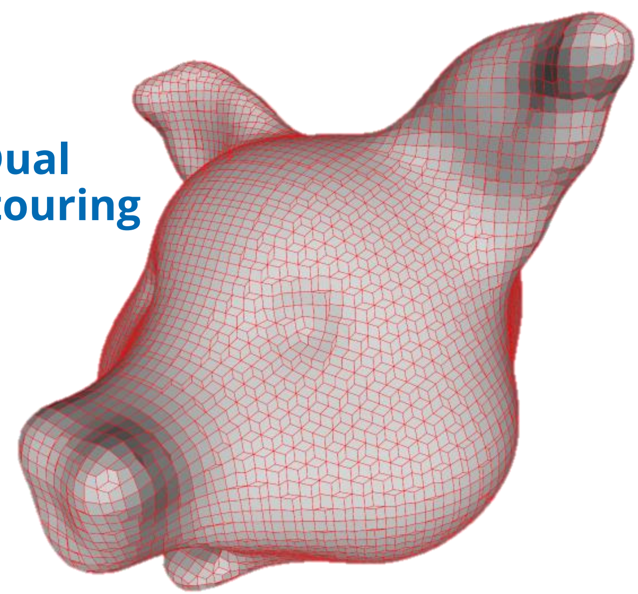


Primal Methods

Cuberille



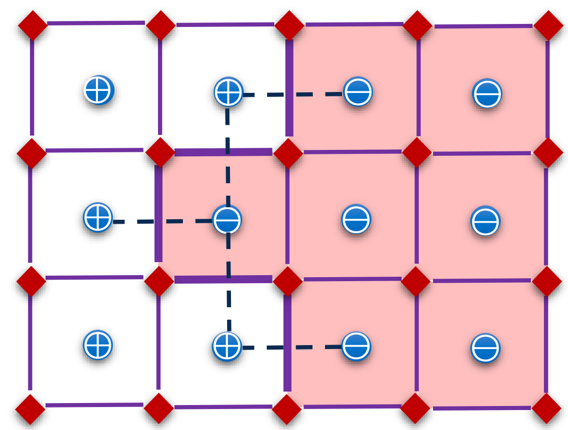
Dual Contouring



Dual Methods

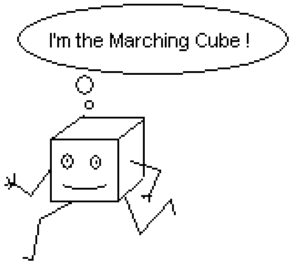
Contouring – Method Overview

- Cuberille
  - Classify all voxels in inside ⊖ / outside ⊕
  - fill dual cell of interior voxels
  - For all edges connecting interior with exterior, add dual face to the Cuberille-surface
- Dual Contouring (see paper)
  - move dual vertices onto iso-surface
  - Cuberville surface is a pure quadrilateral mesh
- Marching Cubes
- Marching Tetrahedra



Contouring – Marching Cubes

- William E. Lorensen, Harvey E. Cline, *Marching Cubes: A high resolution 3d surface construction algorithm*, Siggraph'87, (pdf) … **20346** Zitationen[10.06.24]
  - Proposed algorithm defines regular grid over domain and marches cubes through all grid cells
  - Outputs 0 … 4 triangles per cube
  - Fast implementation by using lookup tables

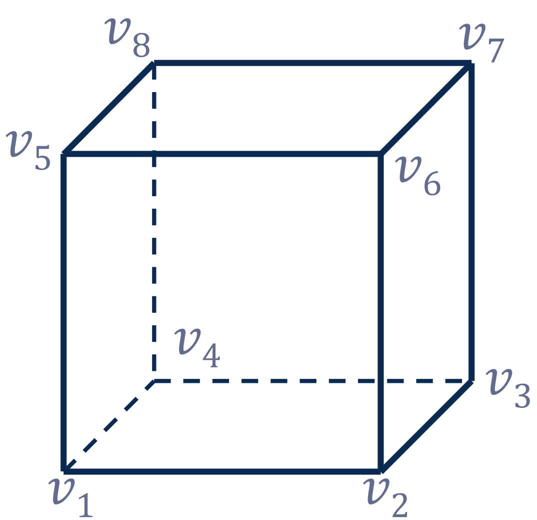


- **Algorithm:**
- iterate all voxel cells …

1. classify 8 knots in inside / outside & create 8-bit index
2. Lookup cut edges and compute edge points with normals of interpolated voxel gradients
3. Lookup triangulation

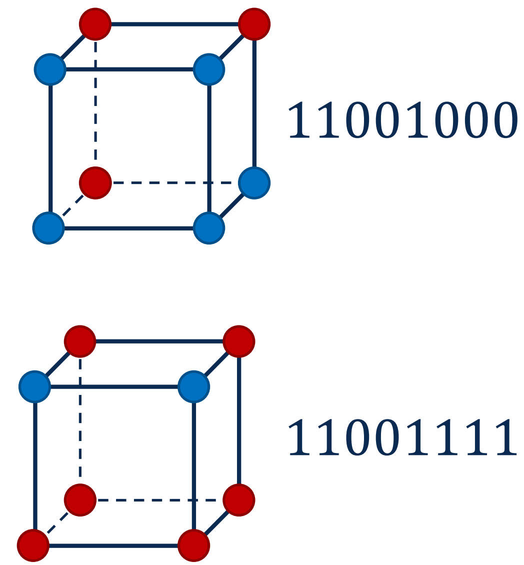
Contouring – MC – Index Computations

- Define numbering v_1 to v_8 of the voxels in a cell
- One bit of classification per voxel
- 8-bit index from concatenation of the bits gives a total of 256 cases



...outside: 0

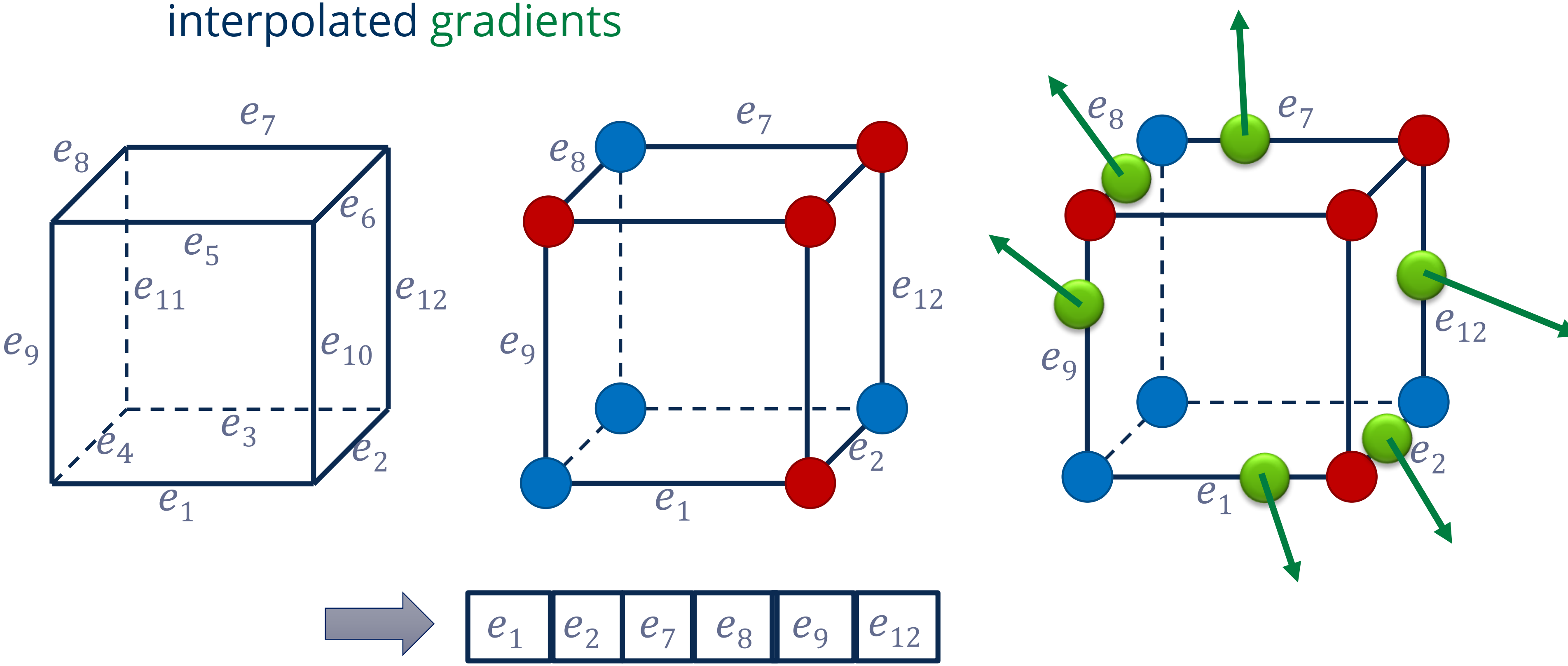
...inside: 1



	v_8	v_7	v_6	v_5	v_4	v_3	v_2	v_1
8-bit index:	128	64	32	16	8	4	2	1

Contouring – MC – Lookup Edges

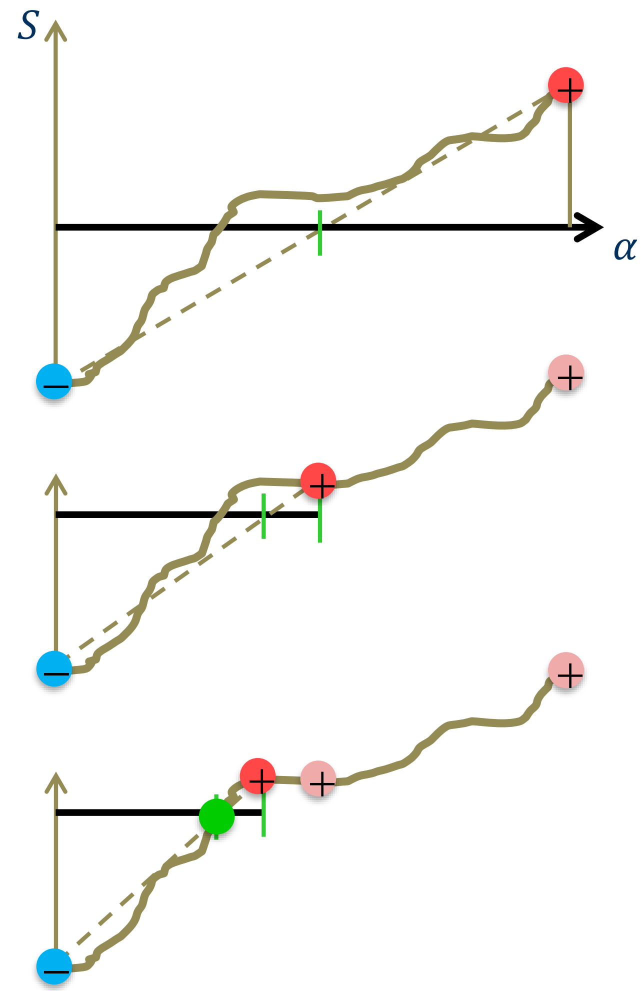
- Define numbering e_1 to e_{12} of the cell edges
- For each case, store a list of edges that intersect iso-surface
- Compute locations of edge points by assuming linear interpolation along edge or a bisection technique, and interpolated gradients



Contouring – MC – Edge Point

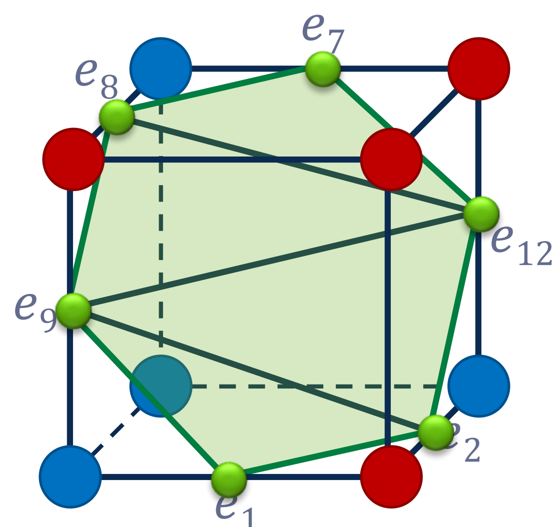
Computer Graphics and Visualization

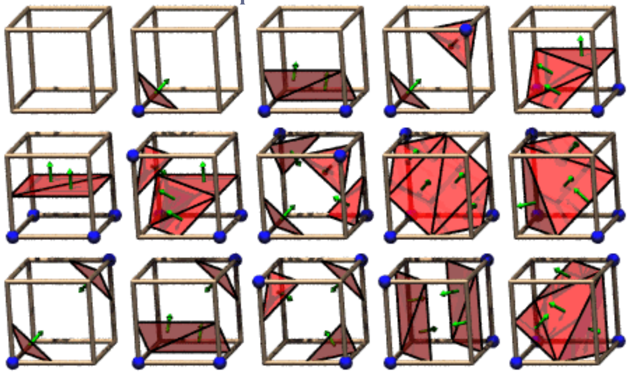
- If an edge connects the inside with the outside, there must be an iso-surface crossing on the edge.
- If you assume a linear interpolation along the edge (correct for trilinear interpolation), you can estimate the position of the iso-surface crossing.
- If the linear approximation is not accurate enough, the edge can be divided and iterated at the estimated iso-surface crossing until the desired accuracy is reached.

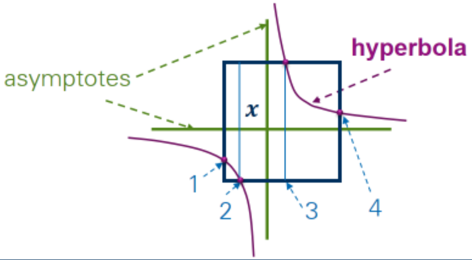


Contouring – MC – Lookup Triangles

- Table-Index = 01110010 = 114
- Entry:
  - 6 edges: $e_1, e_2, e_7, e_8, e_9, e_{12}$
  - 4 triangles: $(e_2, e_1, e_9), (e_2, e_9, e_{12}),$ $(e_{12}, e_9, e_8), (e_{12}, e_8, e_7)$
- lookup table stores cut-edge- and triangle-lists for all 256 cases without exploiting symmetries (otherwise only 15 cases)
- Fixed resolution of ambiguities (to account for trilinear interpolation asymptotic decider per face necessary: per face connect in x-, y- or z-sort order)

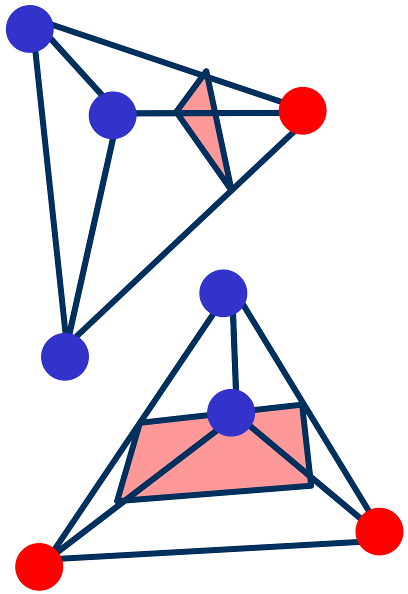



The 15 Cube Combinations

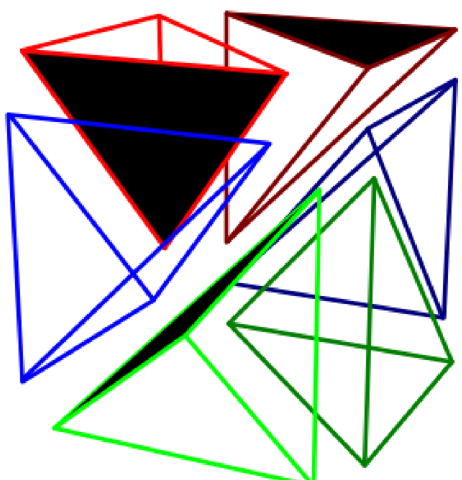


Contouring – Marching Tetrahedra

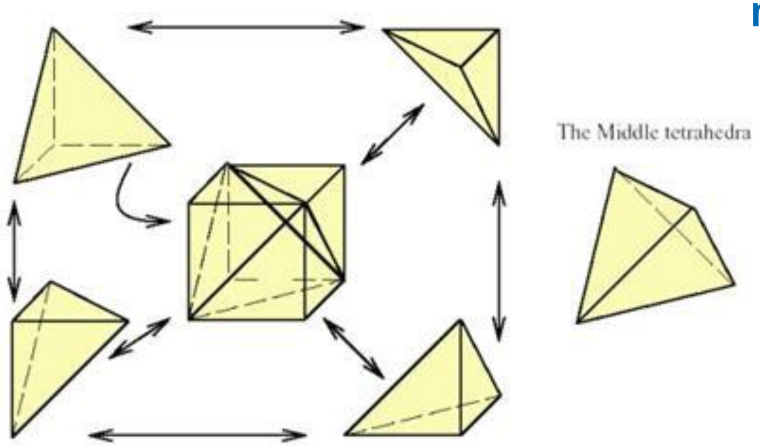
- For tetrahedral meshes, only 2 cases exist such that no lookup table is necessary
- One can convert any voxel grid into a tetrahedral mesh **but**
  - one can split each cube in **5 or 6** tetrahedral
  - tetrahedralizations of adjacent cells need to be compatible on incident faces



The two cases of marching tetrahedra



cube decomposition into 6 tetrahedra



cube decomposition into 5 tetrahedra

Direct Volume Rendering

COMPOSITING

Compositing

Computer Graphics and Visualization

- In direct volume visualization we want to show at each pixel a combination of all values S_t along a ray from the eye point through the pixel
- For this we need to sample the locations along the ray
- The techniques to aggregate the samples' scalar values into a final pixel color are called compositing techniques
- Compositing needs to heavily compress the sampled data

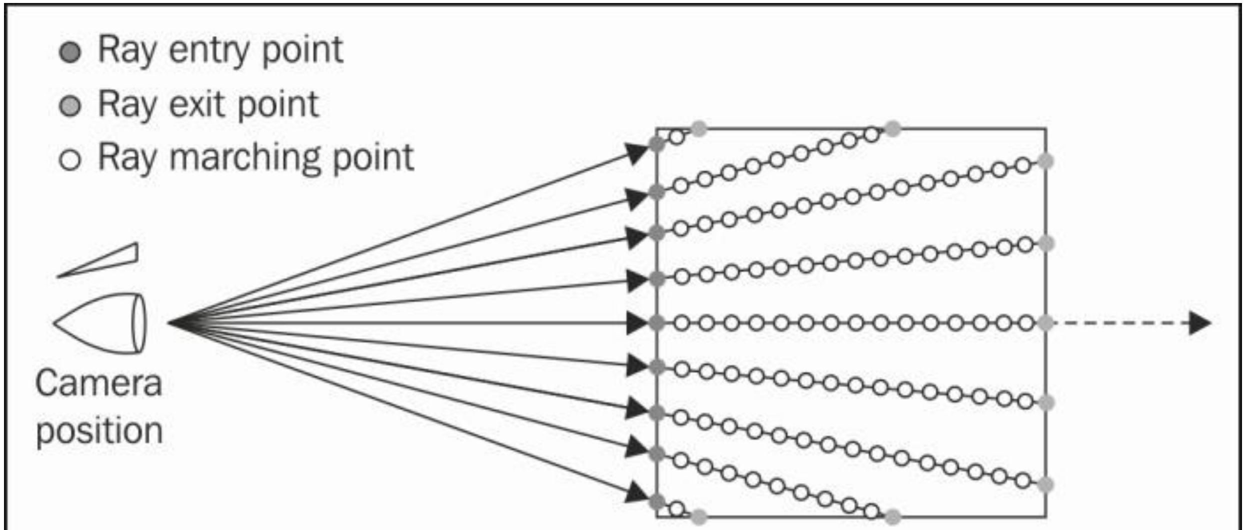
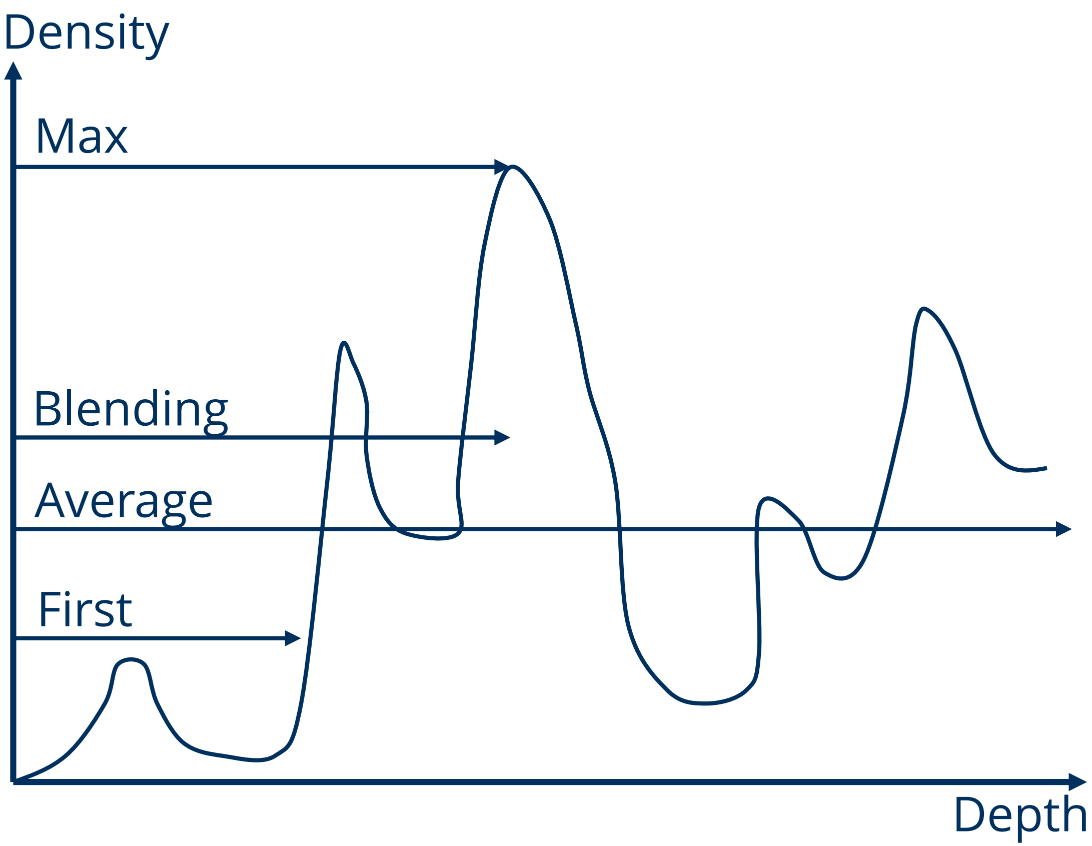
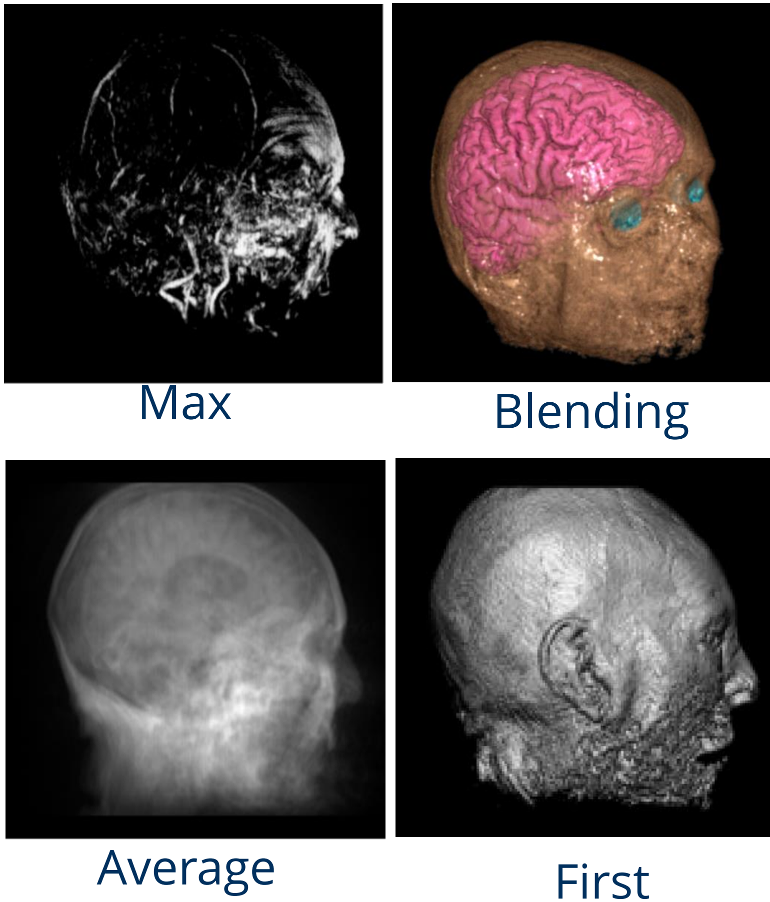


image source: https://www.safaribooksonline.com/library/view/opengl-development-cookbook/9781849695046/ch07s03.html

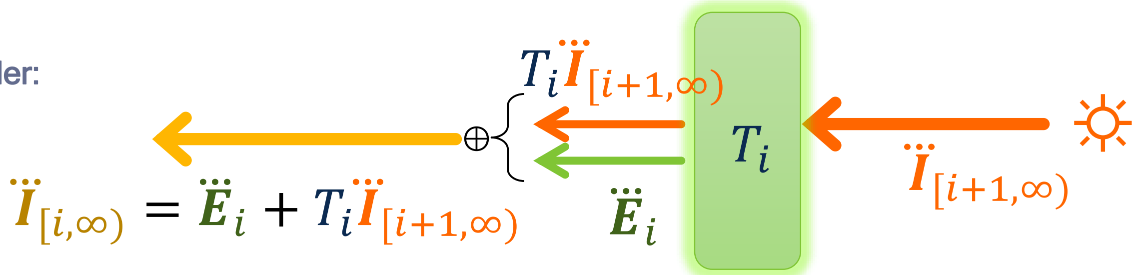
Compositing Strategies





Compositing – Blending

„back to front“-order:



Absorption

- Each layer has a transparency value $T_i \in [0,1]$ that tells us the percentage of light that passes the layer
- Opacity $O_i \in [0,1]$ is percentage of light absorpted in layer: $O_i = 1 - T_i$

Emission

- Emission $\ddot{\boldsymbol{E}}_i$ is the amount of light emitted by the layer as color value (RGB)
- Often emission is set proportional to opacity and chromaticity: $\ddot{\boldsymbol{E}}_i = O_i \cdot \ddot{\boldsymbol{c}}_i$

Blending or Over-Operator

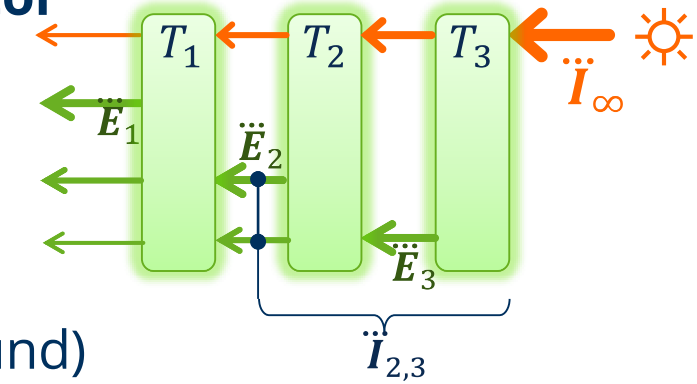
- Order „back ($z = \infty$) to front“: $\ddot{\boldsymbol{I}}_{i,\infty} = (1 - T_i)\ddot{\boldsymbol{c}}_i + T_i \ddot{\boldsymbol{I}}_{i+1,\infty}$
- Order „front ($z = 0$) to back“: $\ddot{\boldsymbol{I}}_{0,i} = \ddot{\boldsymbol{I}}_{0,i-1} + T_{0,i-1} \ddot{\boldsymbol{E}}_i$,
  accumulate transparency: $T_{0,i} = T_i \cdot T_{0,i-1}$

Direct Volume Rendering

THE VOLUME RENDERING INTEGRAL

Iterated Blending

symbols used for discrete blending-operator



- $T_i \in [0,1]$ … transparency of layer i
- $O_i \in [0,1]$ … opacity ($O_i = 1 - T_i$)
- $\dddot{\boldsymbol{E}}_i = O_i \cdot \dddot{\boldsymbol{c}}_i$… emission (RGB) of layer i
- $i \in \{1, n\} \cup \{\infty\}$ … layer index (∞ … background)
- $\dddot{\boldsymbol{I}}_{i,j}$ … intensity in front of layer i, accumulated over layers i…j
- $T_{i,j}$ … transparency through layers i…j

blending-operator

- „back to front“: ←

$$\dddot{\boldsymbol{I}}_{n+1,\infty} = \dddot{\boldsymbol{I}}_\infty \rightarrow \forall i = n \dots 1: \dddot{\boldsymbol{I}}_{i,\infty} = (1 - T_i)\dddot{\boldsymbol{c}}_i + T_i \dddot{\boldsymbol{I}}_{i+1,\infty}$$

- „front to back“: →

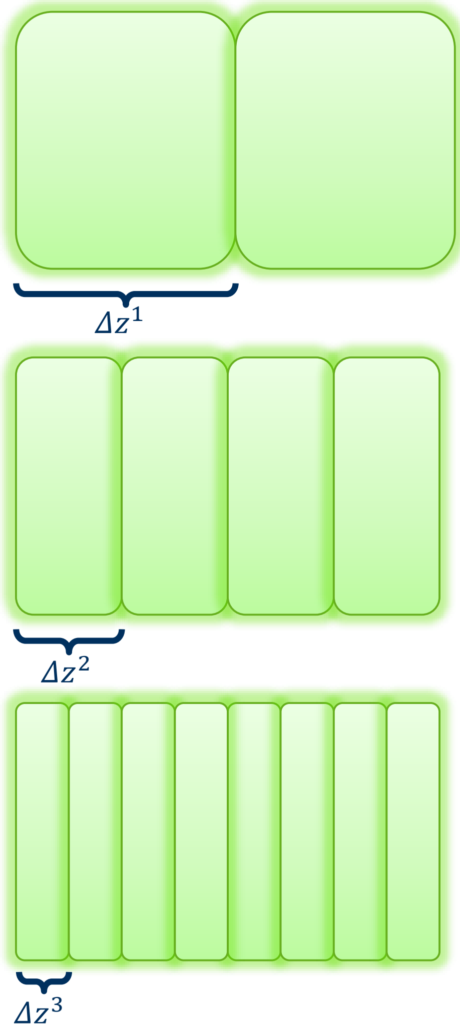
$$\dddot{\boldsymbol{I}}_{1,0} = \dddot{\boldsymbol{0}} \rightarrow \forall i = 1 \dots n: \dddot{\boldsymbol{I}}_{1,i} = \dddot{\boldsymbol{I}}_{1,i-1} + T_{1,i-1} \dddot{\boldsymbol{E}}_i,$$

$$T_{1,0} = 1 \rightarrow \forall i = 1 \dots n: T_{1,i} = T_{1,i} \cdot T_i$$

$$\dddot{\boldsymbol{I}}_{1,\infty} = \dddot{\boldsymbol{I}}_{1,n} + T_{1,n} \dddot{\boldsymbol{I}}_\infty$$

Iterated Blending – Layer Width Variation

- The result of iterated blending should not depend on subdivision into layers.
- How to choose T_i and $\ddot{\boldsymbol{E}}_i$ in dependence of layer depth Δz_i?
- Ansatz: $O_i = o_i \cdot \Delta z_i$, $\ddot{\boldsymbol{E}}_i = \ddot{\boldsymbol{\varepsilon}}_i \cdot \Delta z_i$
- Validation:
  - 2 layers with $o_i = \frac{1}{2}, \varepsilon_i = 1, \Delta z_i = 1$:
    $T_i = 1 - O_i = \frac{1}{2}, E_i = 1$ ➔ $I_{1,2} = E_1 + T_1 E_2 = 1\frac{1}{2}$
  - 4 layers with $o_i = \frac{1}{2}, \varepsilon_i = 1, \Delta z_i = \frac{1}{2}$:
    $T_i = 1 - O_i = \frac{3}{4}, E_i = \frac{1}{2}$
    ➔ $I_{1,4} = \frac{1}{2} + \frac{3}{4}\left(\frac{1}{2} + \frac{3}{4}\left(\frac{1}{2} + \frac{3}{4}\frac{1}{2}\right)\right) \approx 1.367$
- This does not work as the result should not depend on the chosen sampling density



Iterated Blending – As a sum

Computer Graphics and Visualization

- Blending: $I_{0,\infty} = E_1 + T_1\big(E_2 + T_2(E_3 + T_3(\ldots + T_{n-1}E_n))\big) + T_1 \cdot \cdots \cdot T_n I_\infty$
- expanding:

$$I_{0,\infty} = E_1 + T_1E_2 + T_1T_2E_3 + T_1T_2T_3E_4 + \cdots + T_1 \cdot \cdots \cdot T_n I_\infty$$

- This can be written as a sum of products:

$$I_{0,\infty} = \sum_{i=1}^{n}\left(\prod_{j=1}^{i-1} T_j\right)E_i + \left(\prod_{j=1}^{n} T_j\right)I_\infty$$

- The product can be converted to a sum when transformed to log space:

$$T_{1,n} = \exp\left(\log\prod_{j=1}^{n} T_j\right) = \exp\left(\sum_{j=1}^{n}\log T_j\right)$$

- If $T_j \in [0,1]$ then $\log T_j \in [-\infty, 0]$ ➔ define $\Omega_j = -\log T_j \geq 0$
- **Iterated blending as a sum**:

$$I_{1,\infty} = \sum_{i=1}^{n} T_{1,i-1}E_i + T_{1,n}I_\infty, \qquad T_{1,k} = \exp\left(-\sum_{j=1}^{k}\Omega_j\right)$$

Volume-Rendering Integral

$$I_{1,\infty} = \sum_{i=1}^{n} T_{1,i-1} E_i + T_{1,n} I_\infty, \qquad T_{1,k} = \exp\left(-\sum_{j=1}^{k} \Omega_j\right)$$

- A continuous version with integrals instead of sums can be derived with the following replacements with maximum z value $z_{\max}$:

$$i \to z \in [z_{\min}, z_{\max}]$$
$$E_i \to \varepsilon(z) = \frac{\partial E}{\partial z}(z)$$
$$\Omega_i \to \omega(z) = \frac{\partial \Omega}{\partial z}(z)$$

- The viewing ray is parameterized by the depth $z = z_{\min} \dots z_{\max}$ and we arrive at the **Volume-Rendering Integral**:

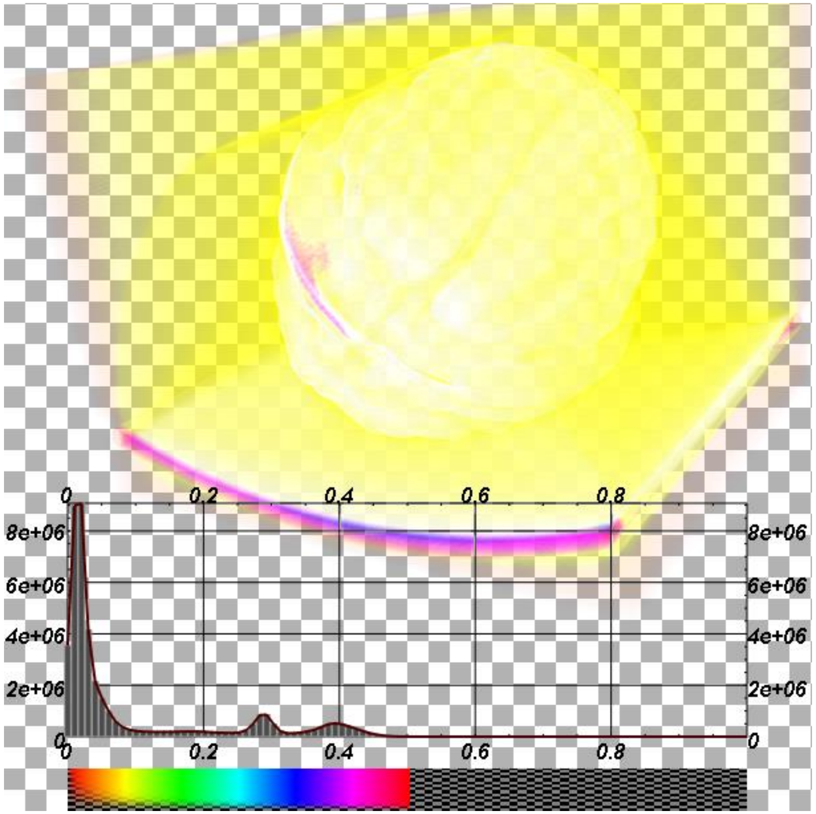
$$\dddot{\boldsymbol{I}}_{0,\infty} = \int_{z_{\min}}^{z_{\max}} T(z_{\min}, z) \dddot{\boldsymbol{\varepsilon}}(z) dz + T(z_{\min}, z_{\max}) \dddot{\boldsymbol{I}}_\infty,$$
$$T(a,b) = e^{-\int_a^b \omega(\tilde{z}) d\tilde{z}}$$

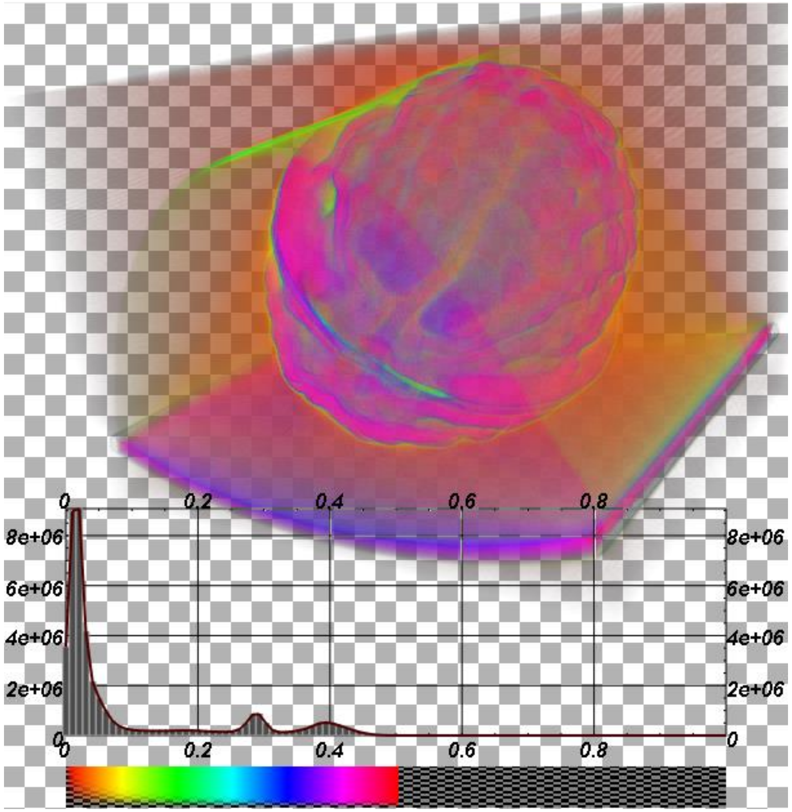
Volume Density Optical Model

Density or Particle Interpretation

- First idea: volume is filled with particles that absorb and emit light. Emission and Absorption are proportional to particle density
- Peter Williams and& Nelson Max proposed 1992 a continous model where emission and absorption are derived from optical density ω and chromaticity $\ddot{\boldsymbol{c}}$
- $\omega(z)$ … is called optical density and describes how much light is absorbed per path length dz. Typically, assumed to be a wavelength independent scalar.
- $\ddot{\boldsymbol{\varepsilon}}(z) = \omega(z)\ddot{\boldsymbol{c}}(z)$ … is wavelength dependent emission per path length (RGB) and proportional to optical density and chromaticity

Emission





- Figures show difference between defining emission independent of optical density (left) and with multiplying ω, i.e. $\ddot{\boldsymbol{\varepsilon}} = \omega \cdot \ddot{\boldsymbol{c}}(S)$ (right)
- Notice that emission becomes too strong on left side

VR Integral – Constant Case

Computer Graphics and Visualization

Volume-Rendering Integral:

- $\omega(z)$ … absorption strength per path length
- $\dddot{\boldsymbol{\varepsilon}}(z) = \omega(z)\dddot{\boldsymbol{c}}(z)$ … emission per path length (RGB)
- $\Omega(a,b) = \int_a^b \omega(\tilde{z})d\tilde{z}$ … absorption strength per layer from a to b
- $T(a,b) = \exp\big(-\Omega(a,b)\big)$ … transparency per layer
- $O(a,b) = 1 - \exp\big(-\Omega(a,b)\big)$ … opacity per layer
- $\dddot{\boldsymbol{E}}(a,b) = \int_a^b T(a,z)\dddot{\boldsymbol{\varepsilon}}(z)dz$ … emission per layer
- $\dddot{\boldsymbol{I}}_{0,\infty} = \int_0^\infty T(0,z)\dddot{\boldsymbol{\varepsilon}}(z)dz + T(0,\infty)\dddot{\boldsymbol{I}}_\infty$ … intensity along viewing ray

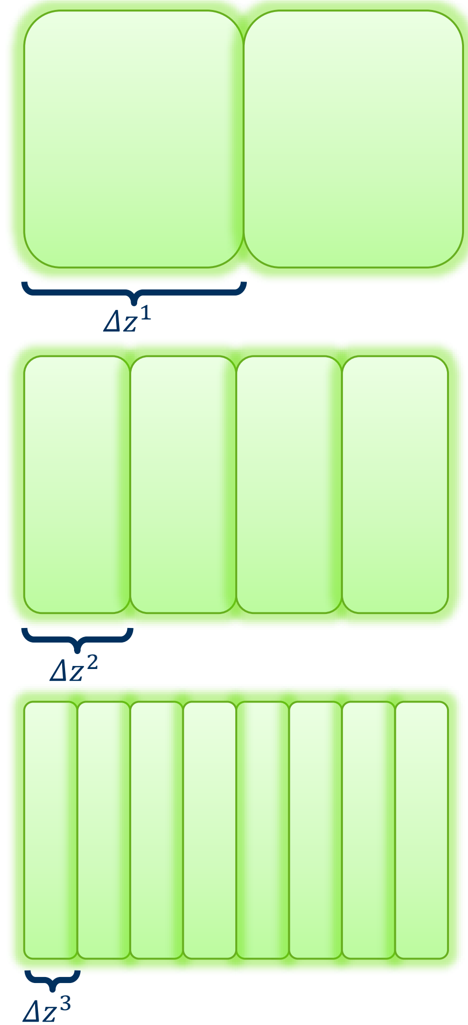
Constant Case with layer depth Δz (see exercise)

$$T(\Delta z) = e^{-\omega_0 \Delta z}$$

$$\dddot{\boldsymbol{E}}(\Delta z) = \frac{\dddot{\boldsymbol{\varepsilon}}_0}{\omega_0}\overbrace{\left(1 - e^{-\omega_0 \Delta z}\right)}^{O(\Delta z)} = O(\Delta z)\dddot{\boldsymbol{c}}_0, \ \lim_{\omega_0 \to 0} \dddot{\boldsymbol{E}}(\Delta z) = \Delta z \cdot \dddot{\boldsymbol{\varepsilon}}_0$$

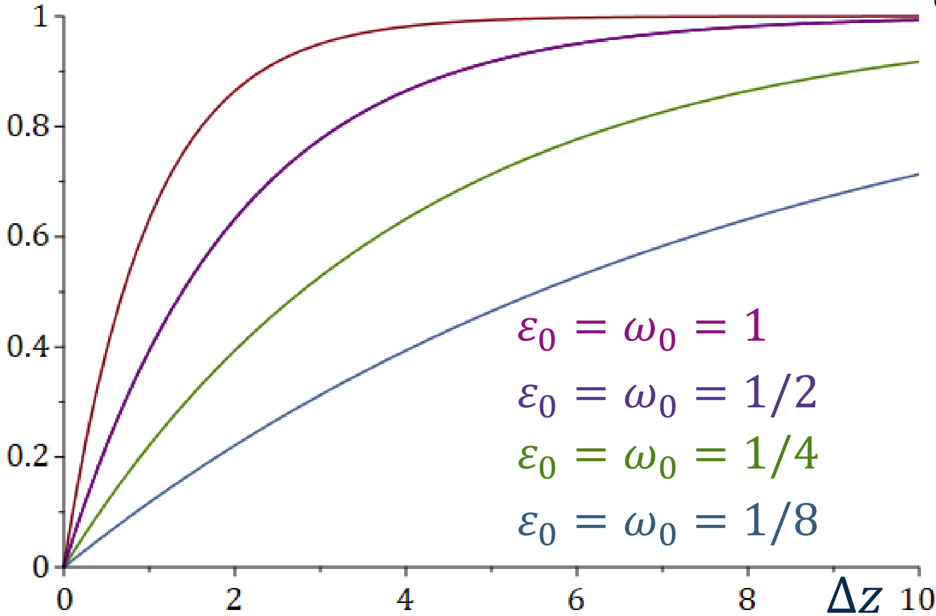
VR Integral – Discretization

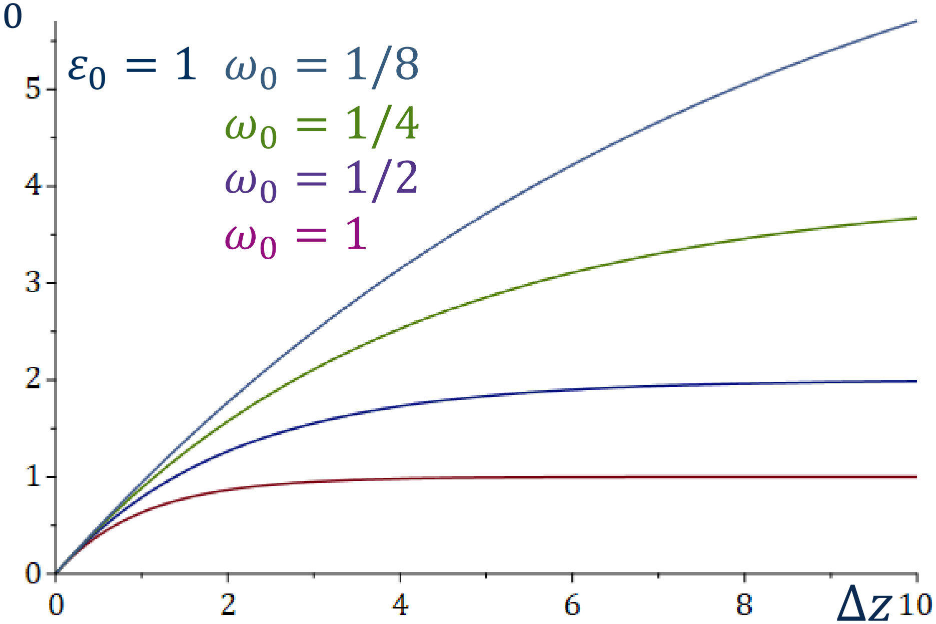
- compute contribution of a layer from a to b from
  $\ddot{\boldsymbol{E}}(a,b) = \int_a^b T(a,z)\ddot{\boldsymbol{\varepsilon}}(z)dz\,,\ T(a,b) = e^{-\int_a^b \omega(\tilde{z})d\tilde{z}}$
- Constant case: $\ddot{\boldsymbol{\varepsilon}}(z) \equiv \ddot{\boldsymbol{\varepsilon}}_0$ and $\omega(z) \equiv \omega_0$
  $$\ddot{\boldsymbol{E}}(a,b) = \ddot{\boldsymbol{\varepsilon}}_0 \int_a^b e^{-\omega_0(z-a)}\,dz\,, T(a,b) = e^{-\omega_0(b-a)}$$
- with $\Delta z = b - a$ we get
  $$\ddot{\boldsymbol{E}}(\Delta z) = \frac{\ddot{\boldsymbol{\varepsilon}}_0}{\omega_0}\left(1 - e^{-\omega_0 \Delta z}\right), \qquad T(\Delta z) = e^{-\omega_0 \Delta z}$$
- validation for $\varepsilon_0 \equiv 1$ and $\omega_0 \equiv 1$
  - 2 layers with $\Delta z \equiv 1$: $E_i = 1 - \frac{1}{e}, T_i = \frac{1}{e}$
    ➔ $I_{1,2} = E_1 + T_1 E_2 = \left(1 + \frac{1}{e}\right)\left(1 - \frac{1}{e}\right) = 1 - \frac{1}{e^2}$
  - 4 layers with $\Delta z = \frac{1}{2}$: $E_i = 1 - \frac{1}{\sqrt{e}}, T_i = \frac{1}{\sqrt{e}}$ ✔
    ➔ $I_{1,2} = \left(1 + \frac{1}{\sqrt{e}}\right)\left(1 - \frac{1}{\sqrt{e}}\right) = 1 - \frac{1}{e}$ ➔ $I_{1,4} = 1 - \frac{1}{e^2}$



Constant Case –Intensity Range

$$E(\Delta z) = \frac{\varepsilon_0}{\omega_0}\left(1 - e^{-\omega_0 \Delta z}\right)$$

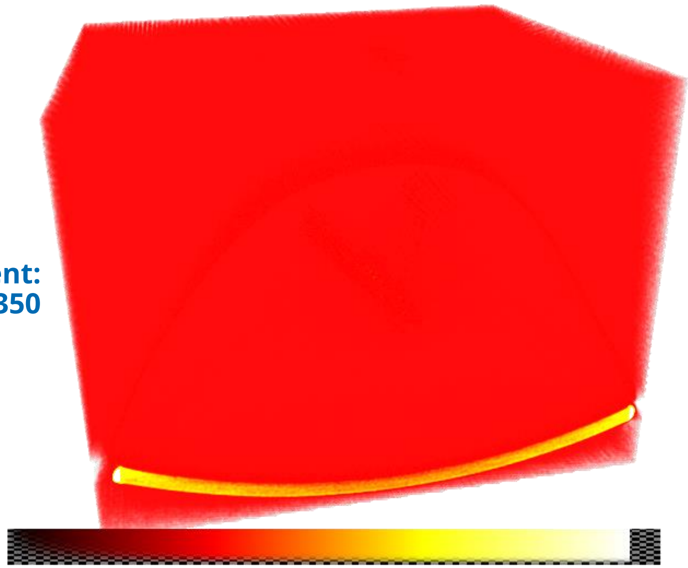




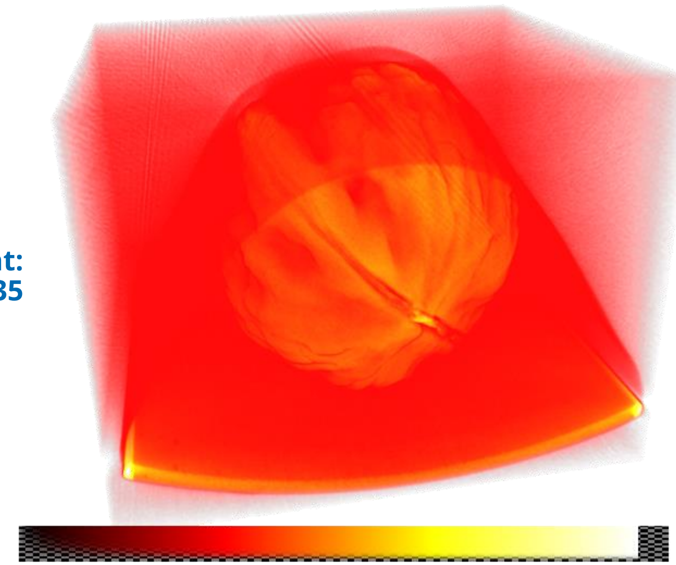
- If we choose ε_0 proportional to ω_0 (left plot) then the emitted intensity $E(\Delta z)$ converges for $\Delta z \rightarrow \infty$ always to 1.
- If ε_0 is greater than ω_0 (right plot) then $E(\Delta z)$ becomes larger than 1
- to have pixel values in [0,1] one sets: $\varepsilon = \omega \cdot c$ with $c \in [0,1]$
- this makes constant case numerically stable: $\ddot{\boldsymbol{E}}(\Delta z) = \ddot{\boldsymbol{c}}_0\left(1 - e^{-\omega_0 \Delta z}\right)$

Is VolRen scale invariant? – no

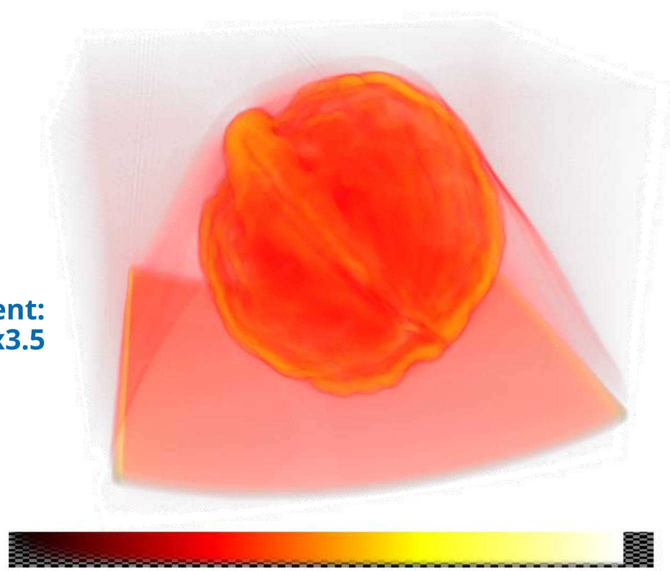
extent: 400x300x350



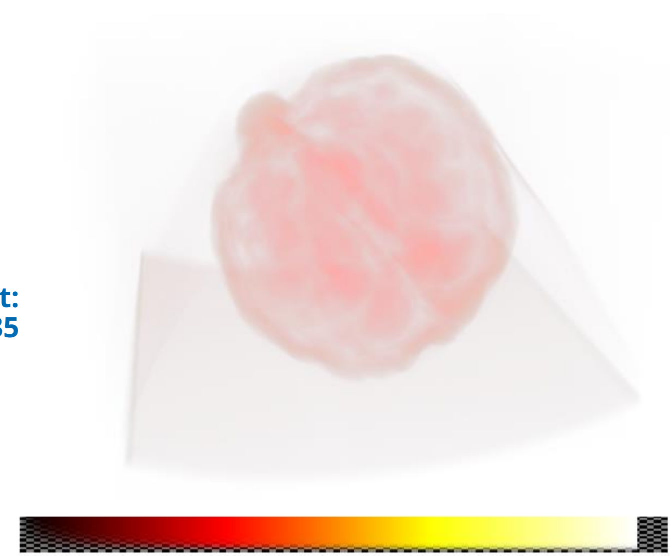
extent: 40x30x35



extent: 4x3x3.5



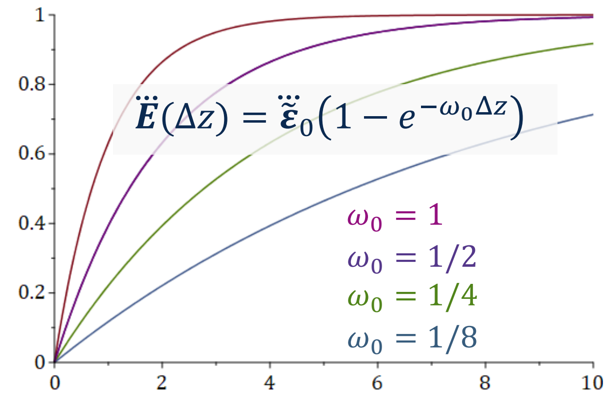
extent: 0.4x0.3x0.35



VR-Integral – Scale Adaptation

Computer Graphics and Visualization

- If we increase/decrease size of volume, volume rendering integral yields more opaque/transparent results
- To scale the volume, one can simply multiply the differential path length with a factor s_V in order to integrate over scaled length:



$$\ddot{\boldsymbol{I}}_{0,\infty} = \int_0^{z_{\max}} T(0,z)\ddot{\boldsymbol{\varepsilon}}(z) \cdot s_V dz + T(0,\infty)\ddot{\boldsymbol{I}}_\infty, \qquad T(a,b) = e^{-\int_a^b \omega(\tilde{z}) \cdot s_V d\tilde{z}}$$

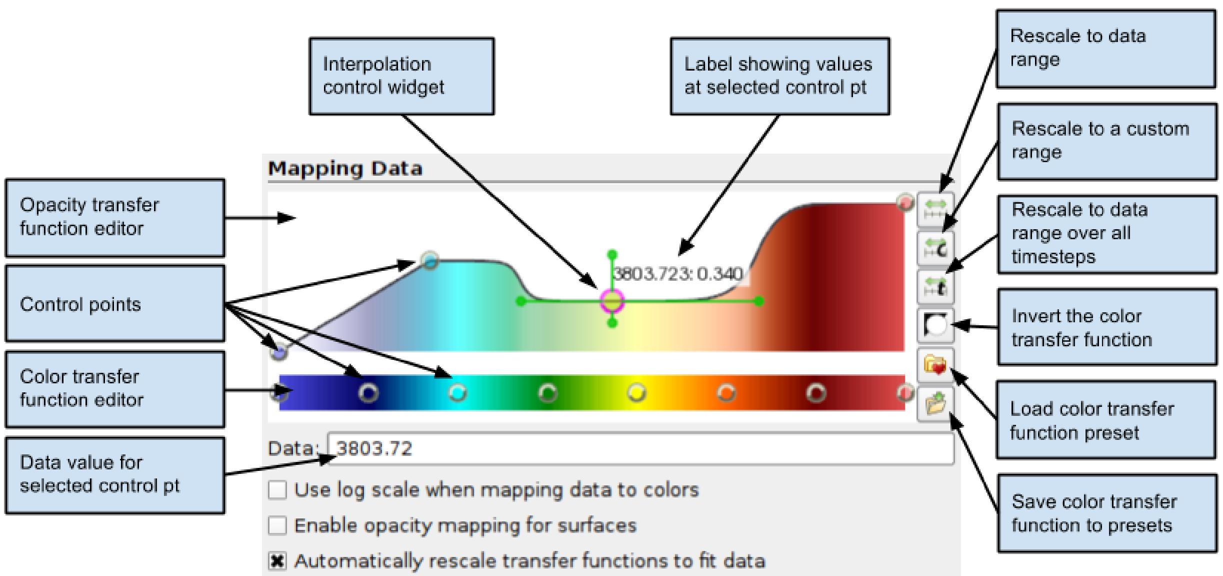
- This results in a joint scaling of $\ddot{\boldsymbol{\varepsilon}}(z)$ and $\omega(z)$ by s_V
- The optimal scale depends on the value distribution inside the Volume. From total / per value S voxel counts $\#/\#_S$ and transfer function $\omega(S)$ one can estimate the average value $\overline{\omega} = \frac{1}{\#}\sum_S \#_S \omega(S)$
- For expected opacity of $\hat{O}$ and bounding box diagonal d, one can estimate s_V through constant case approximation:
  $\hat{O} = 1 - e^{-s_V \overline{\omega} d} \Rightarrow \tilde{s}_V(\hat{O}, \overline{\omega}) = \frac{\log(1-\hat{O})}{\overline{\omega} d}$. E.g. $\tilde{s}_V\left(95\%, \frac{1}{8}\right) \approx 24/d$

Direct Volume Rendering

TRANSFER FUNCTIONS PART 1

Transfer Function Design

- Let $S \in [S_{\min}, S_{\max}]$ be the scalar attribute of the volume dataset
- In the simplest approach a transfer function maps the scalar values S to an chromaticity $\ddot{\boldsymbol{c}}(S)$ and opacity $O(S)$
- Based on volume extent opacity is converted to absorption strength $\omega(S)$ per traveled length and emission strength $\ddot{\boldsymbol{\varepsilon}}(S)$ per traveled length is computed according to $\ddot{\boldsymbol{\varepsilon}}(S) = \omega(S) \cdot \ddot{\boldsymbol{c}}(S)$.
- typical editors are similar to curve editors and use control points



Paraview-Editor (https://blog.kitware.com/using-the-color-map-editor-in-paraview-the-basics)

Hounsfield Scale

- Scalar values of volumetric CT images measure the linear attenuation coefficient μ of x-ray radiation
- Values can be scaled according to Hounsfield units:
  - number format: 16Bit signed integer with 12 significant bits
  - encoding range: $[-1024, 3071]$
  - scale is linear and based on μ values for air and water:

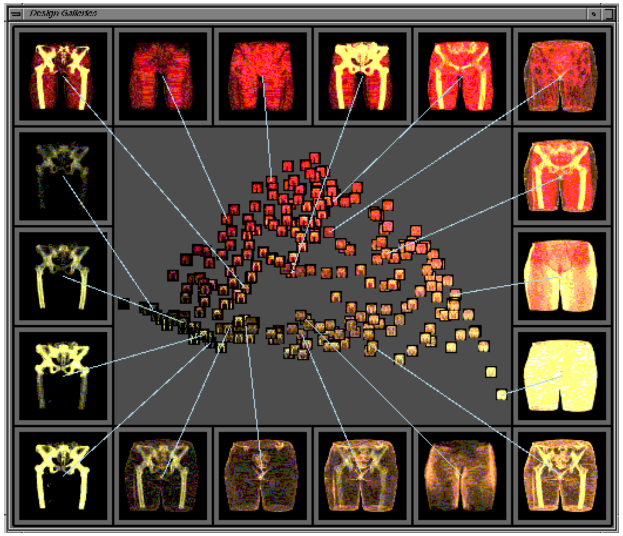
$$v_{\mathrm{HU}}(\mu) = 1000 \times \frac{\mu - \mu_{\mathrm{water}}}{\mu_{\mathrm{water}} - \mu_{\mathrm{air}}}$$

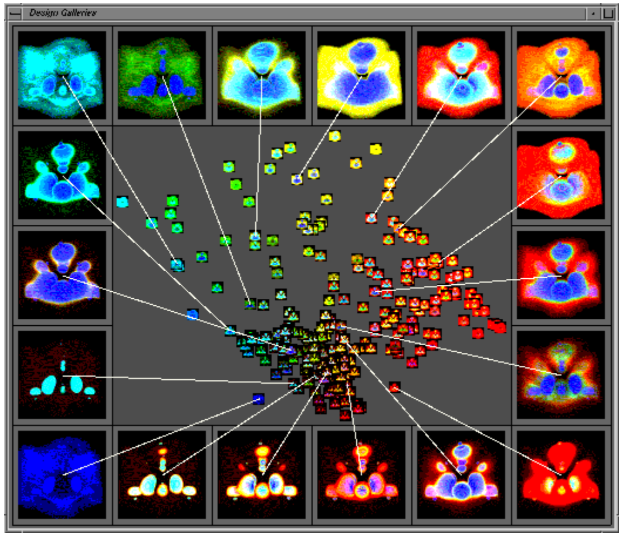
  - Some values / value ranges:
    - air: -1000, water: 0
    - lung: -700 … -600, fatt: -120 … -90, blood: +13 … +50,
    - soft tissue: +100 … +300, bone: +1800 … +1900
  - due to noise and overlapping ranges, different soft tissue organs cannot be segmented based only on scalar values
  - Bit depth reduction to 8bit unsigned ints: $v_{\mathrm{8bit}} = \left\lfloor 256 \frac{v_{\mathrm{HU}}+1024}{4096} \right\rfloor$

Sir Godfrey Newbold Hounsfield

Transfer Function Design Galleries

- Design Galleries provide a simplified user interface:
  - Parameterize transfer function with about 20-30 curve parameters
  - sample parameter space randomly and generate volume rendering for each sample
  - choose Design Gallery as a subset of samples so that their volume rendering differ maximally
  - show the gallery to the user and ask for one or more samples
  - iterate with local sampling of the parameter space

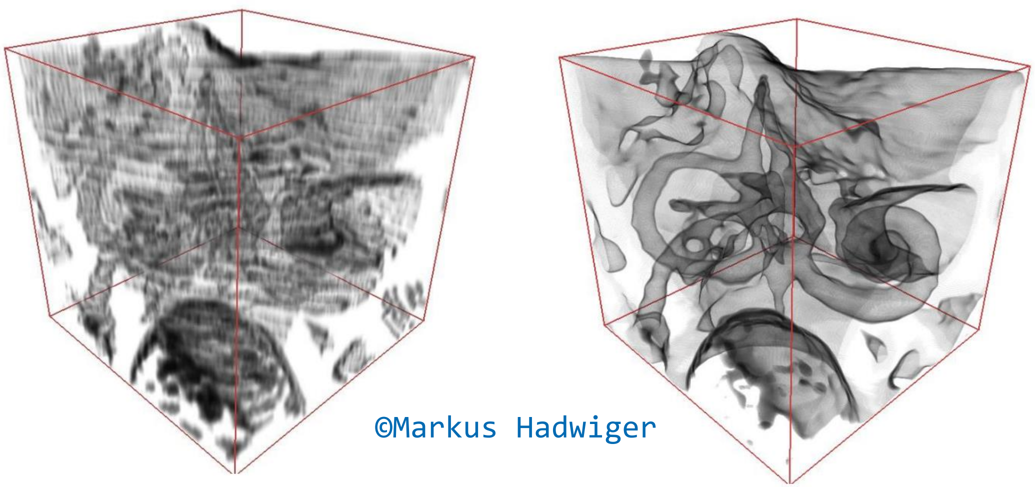




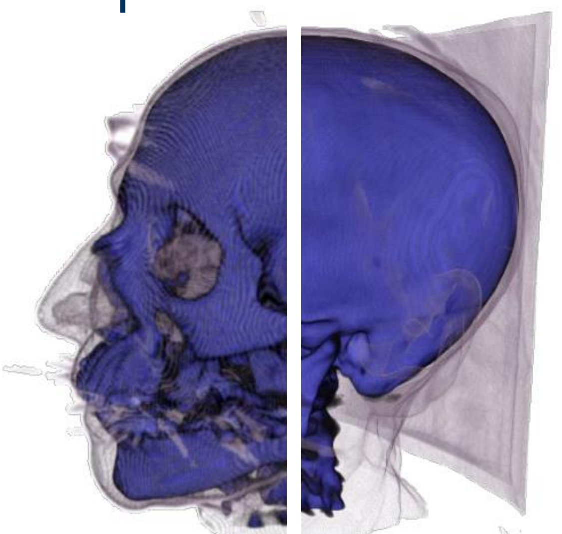
Marks, Joe, et al. "Design galleries: A general approach to setting parameters for computer graphics and animation." *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 1997. acm-link

Transfer Function – Pre- vs Post-Interpolation

- One can apply the transfer function to the voxel values resulting in a rgba volume. This is called pre-interpolation as the rgba values are interpolated afterwards
- In post-interpolation one first interpolates the scalar values and then applies the transfer function
- For high frequency transfer functions pre-interpolation yields significant artefacts ➔ use post-interpolation



pre-classification post-classification



pre-interpolation post-interpolation

Transfer Function – Pre-integration

- During raycasting emission intensity and absorption probability are a function of depth z: $\boldsymbol{\varepsilon}\big(S(z)\big), \omega\big(S(z)\big)$
- Even for a linear scalar function

$$S(z) = \frac{z_1 - z}{\Delta z} S_0 + \frac{z - z_0}{\Delta z} S_1, \qquad \Delta z = z_1 - z_0$$

  both functions can vary significantly & non-linearly in z
- But for linear functions the volume rendering integral only depends on the three parameters S_0, S_1 and Δz.
- To show this we change the integration variable from z to S: $dS(z) = \frac{\Delta S}{\Delta z} dz,\ \Delta S = S_1 - S_0$:

$$\ddot{\boldsymbol{E}}(S_0, S_1, \Delta z) = \int_{z_0}^{z_1} T(z_0, z)\ddot{\boldsymbol{\varepsilon}}(z)dz = \frac{\Delta z}{\Delta S}\int_{S_0}^{S_1} T(S_0, S, \Delta z)\ddot{\boldsymbol{\varepsilon}}(S)dS$$

$$T(S_0, S_1, \Delta z) = e^{-\int_{z_0}^{z_1} \omega(\tilde{z})d\tilde{z}} = e^{-\frac{\Delta z}{\Delta S}\int_{S_0}^{S_1} \omega(\tilde{S})d\tilde{S}}$$

Transfer Function – Pre-integration

Computer Graphics and Visualization

- Transfer function is typically defined over discretization of S into n values: $\forall i = 0 \ldots n-1$: $S_i = i \cdot \delta S,\ \delta S = \frac{1}{n-1}$
- For the transparency integral one can work with a 1D integral table of the antiderivative $\Omega(S_i) = \int_0^{S_i} \omega(\tilde{S}) d\tilde{S}$:

$$T_{ij} = T\left(S_i, S_j, \Delta z\right) = e^{-\frac{\Delta z}{S_j - S_i} \int_{S_i}^{S_j} \omega(\tilde{S}) d\tilde{S}} = e^{-\frac{\Delta z}{S_j - S_i} \cdot \left(\Omega(S_j) - \Omega(S_i)\right)}$$

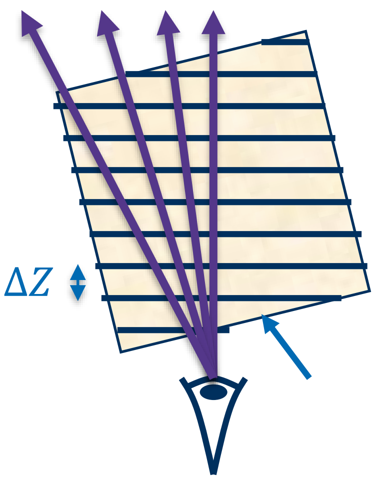
- Special case for $S_i = S_j$: $T_{ii} = e^{-\omega(S_i)\Delta z}$
- The table $\Omega_i = \Omega(S_i)$ can be computed in $O(n)$:

$$\Omega_0 = 0, \Omega_{i+1} = \Omega_i + \int_{S_i}^{S_{i+1}} \omega\left(\tilde{S}\right) d\tilde{S} \approx \Omega_i + \omega\left(\frac{S_i + S_{i+1}}{2}\right) \delta S$$

- Summary: $T_{ij}(\Delta z) = \begin{cases} \exp[-\omega(S_i) \cdot \Delta z] & i = j \\ \exp\left[-\frac{\Delta z}{(j-i) \cdot \delta S} \left(\Omega_j - \Omega_i\right)\right] & i \neq j \end{cases}$

Transfer Function – Pre-integration

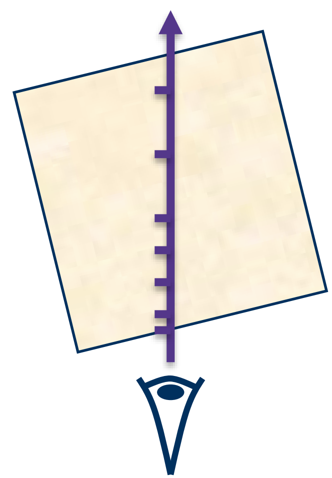
- For the emission integral the trick to integrate independent of Δz does not work.
- Depending on the rendering algorithm one discretizes Δz into m values: $\Delta z_{k=0\ldots m-1}$



Texture-Slicing

$$\Delta z_k = \left(1 + \frac{k}{m-1}\right)\Delta Z$$

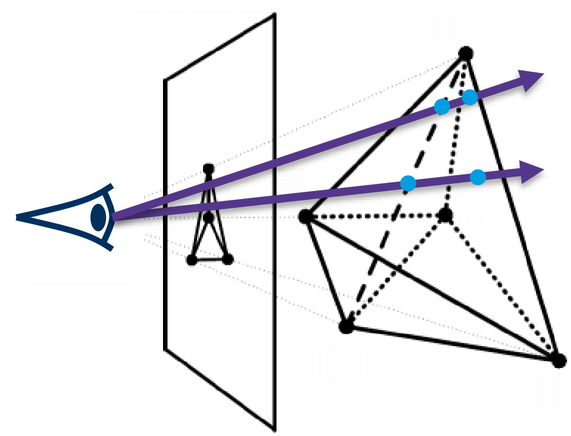
$m \approx 5$



Ray Casting

$$\Delta z_k = 2^k \cdot \Delta z_{\min}$$

$m \approx 5$



Projektion

$$\Delta z_k = \frac{k}{m-1} \cdot \Delta z_{\max}$$

$m \approx 20$

Transfer Function – Pre-integration

- For emission a 3D pre-integration lookup is necessary:

$$\dddot{\boldsymbol{E}}_{ijk} = \dddot{\boldsymbol{E}}\left(S_i, S_j, \Delta z_k\right) = \frac{\Delta z_k}{S_j - S_i} \int_{S_i}^{S_j} T(S_i, S, \Delta z_k) \dddot{\boldsymbol{\varepsilon}}(S) dS$$

- Special case for $i = j$: $\dddot{\boldsymbol{E}}_{iik} = \dddot{\boldsymbol{c}}(S_i)\left(1 - e^{-\omega(S_i)\Delta z}\right)$
- 2D antiderivative $\dddot{\boldsymbol{\Xi}}_{ik} = \int_0^{S_i} T\left(0, \tilde{S}, \Delta z_k\right) \dddot{\boldsymbol{\varepsilon}}\left(\tilde{S}\right) d\tilde{S}$ table:

$$\dddot{\boldsymbol{E}}_{ijk} = \frac{\Delta z_k}{S_j - S_i} \frac{\dddot{\boldsymbol{\Xi}}_{jk} - \dddot{\boldsymbol{\Xi}}_{ik}}{T(0, S_i, \Delta z_k)},$$

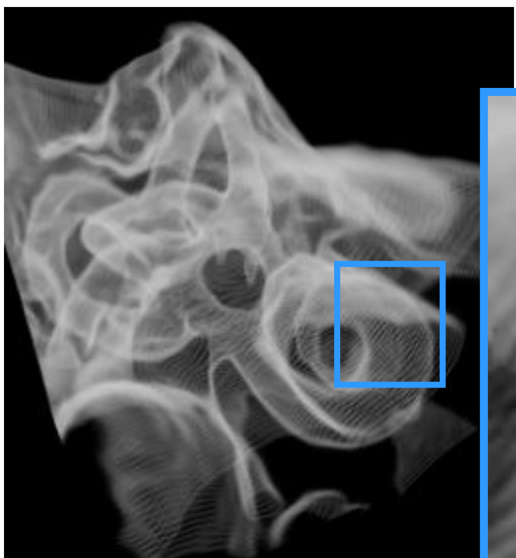
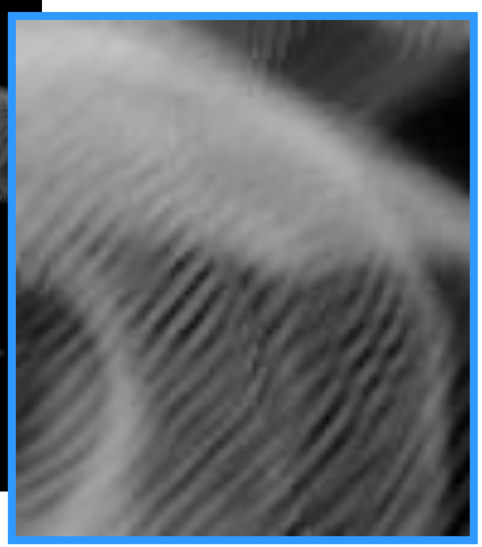
  where we define $T(0,0,\Delta z) := 1$.
- Incremental computation of $\dddot{\boldsymbol{\Xi}}_{ik}$:

$$\dddot{\boldsymbol{\Xi}}_{0k} = \dddot{\boldsymbol{0}}, \dddot{\boldsymbol{\Xi}}_{(i+1)k} = \dddot{\boldsymbol{\Xi}}_{ik} + \int_{S_i}^{S_{i+1}} T\left(0, \tilde{S}, \Delta z_k\right) \dddot{\boldsymbol{\varepsilon}}\left(\tilde{S}\right) d\tilde{S}$$

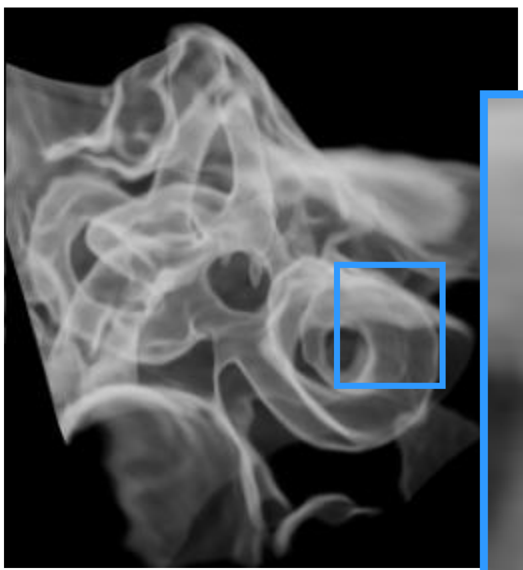
$$\approx \dddot{\boldsymbol{\Xi}}_{ik} + T\left(0, S_{i+\frac{1}{2}}, \Delta z_k\right) \dddot{\boldsymbol{\varepsilon}}\left(S_{i+\frac{1}{2}}\right) \delta S$$

Transfer Function – Pre-integration

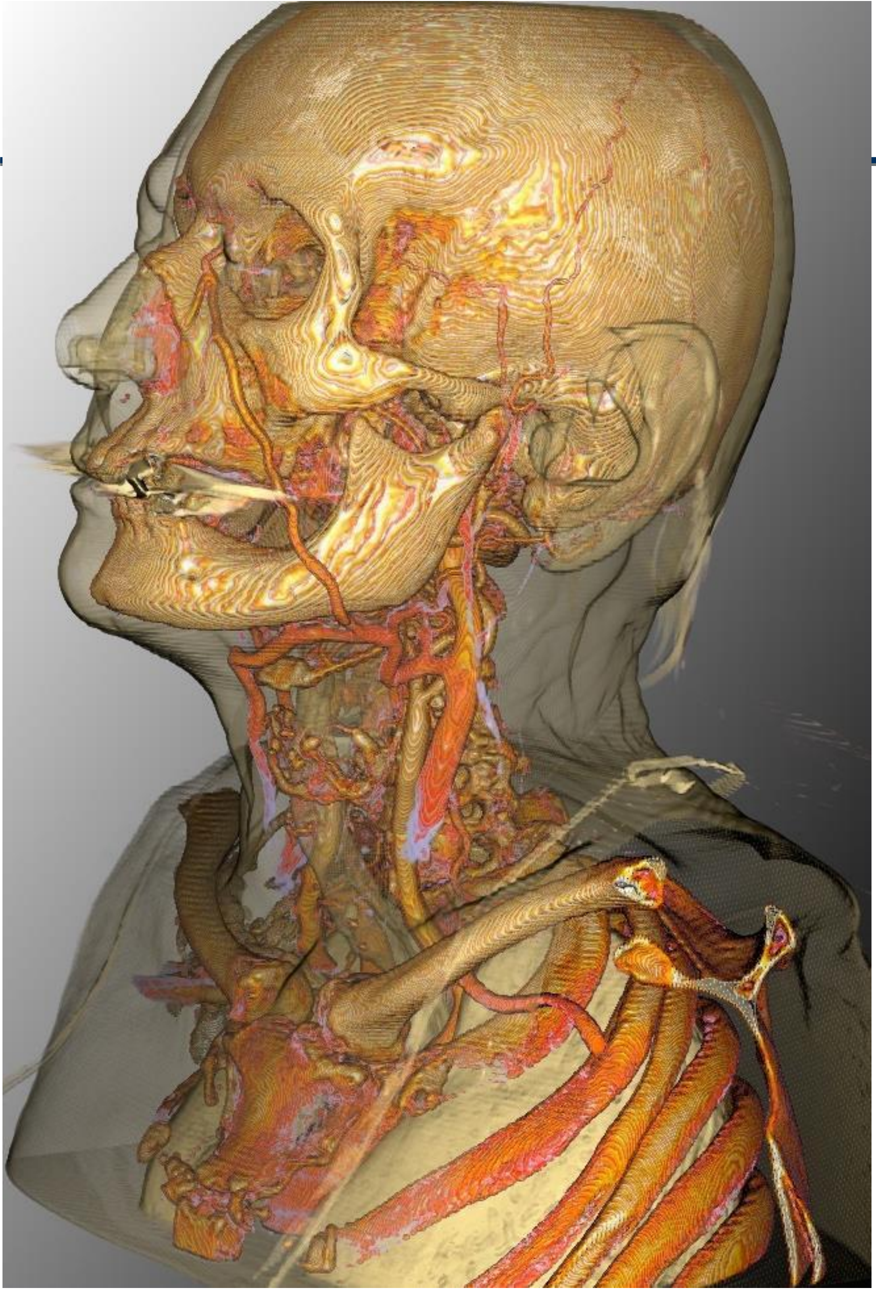
- Precomputation runtime and space consumption for n scalar values S_i and m step widths Δz_k:
  - Ω_i … $O(n)$
  - $\ddot{\Xi}_{ik}$ … $O(m \cdot n)$
- Per table entry runtime:
  - $T_{ij}(\Delta z)$ … $O(1)$
  - $\ddot{\boldsymbol{E}}_{ijk} = \frac{\Delta z_k}{S_j - S_i} \frac{\ddot{\Xi}_{jk} - \ddot{\Xi}_{ik}}{T(0, S_i, \Delta z_k)}$ … $O(1)$
- Overall runtime: $O(m \cdot n^2)$

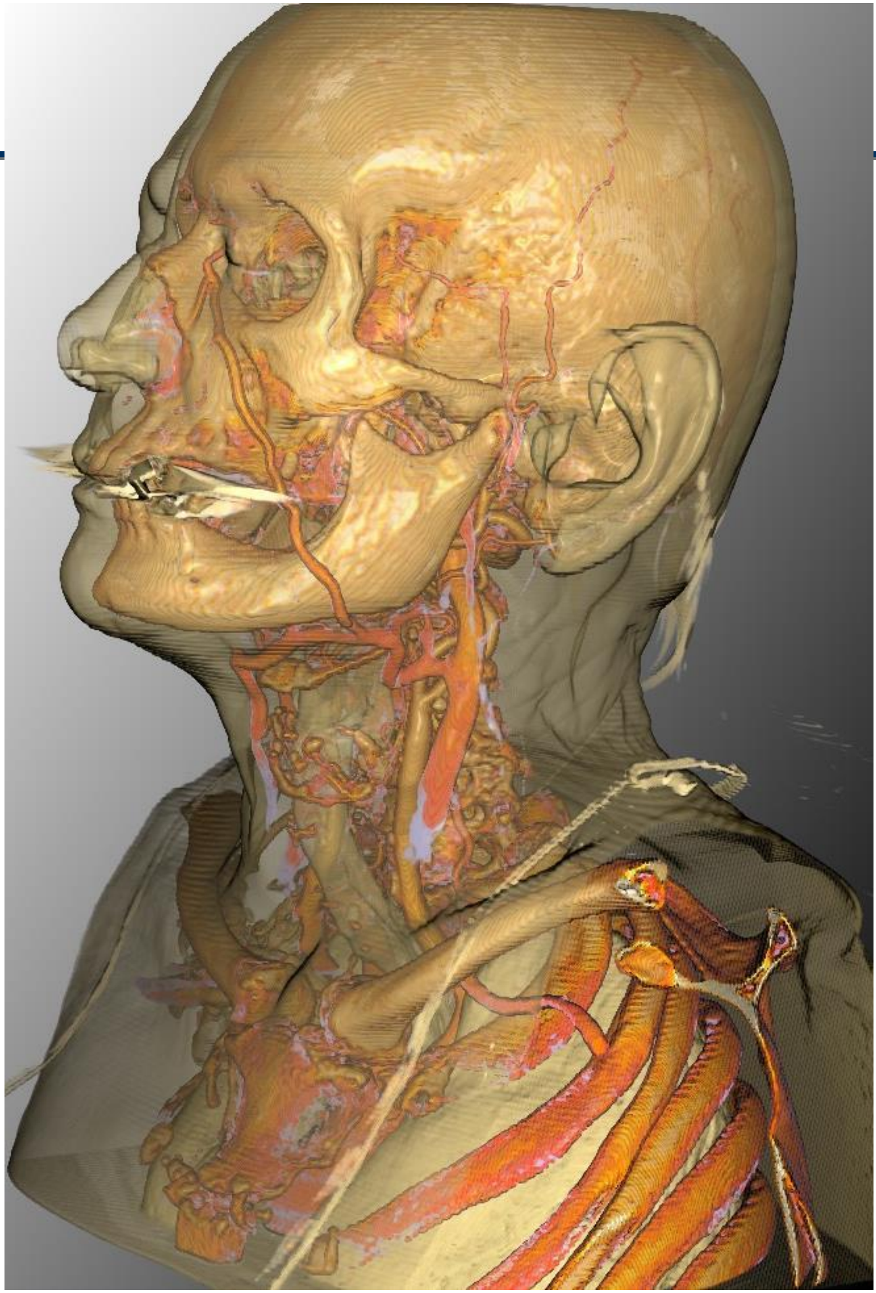
no pre-integration




with pre-integration



no pre-integration



with pre-integration

Transfer Function – Pre-integration

- Pre-integration provides fast access to the volume rendering integral for the case where S varies linearly
- In the simplest implementation one works with a 3D lookup function stored in a 3D RGBA texture, but changes in the transfer function demand for long re-computation times of the 3D lookup table
- Pre-integration only works for 1D transfer functions