CS 247 – Scientific Visualization
Lecture 17: Volume Visualization, Pt. 4

Markus Hadwiger, KAUST

Reading Assignment #9 (until Apr 2)

Read (required):

- Real-Time Volume Graphics, Chapter 7 (GPU-Based Ray Casting)
- Real-Time Volume Graphics, Chapter 4.5 – 4.8

Quiz #2: Apr 2

Organization

- First 30 min of lecture
- No material (book, notes, ...) allowed

Content of questions

- Lectures (both actual lectures and slides)
- Reading assignments (except optional ones)
- Programming assignments (algorithms, methods)
- Solve short practical examples

Opacity Correction

Volume Rendering Integral Summary

Volume rendering integral
for *Emission Absorption* model

true emission **true *absorption***

$$I(s) = I(s_0)\, e^{-\tau(s_0,s)} + \int_{s_0}^{s} q(\tilde{s})\, e^{-\tau(\tilde{s},s)}\, \mathrm{d}\tilde{s}$$

$$\tau(s_1, s_2) = \int_{s_1}^{s_2} \kappa(s)\, ds.$$

Iterative/recursive numerical solutions:

Back-to-front compositing

$$C'_i = C_i + (1 - A_i)C'_{i-1}$$

Front-to-back compositing

$$C'_i = C'_{i+1} + (1 - A'_{i+1})C_i$$
$$A'_i = A'_{i+1} + (1 - A'_{i+1})A_i$$

here, all colors are *associated* colors!

Opacity Correction

Simple compositing only works as far as the opacity values are correct… and they depend on the sample distance!

$$T_i = e^{-\int_{s_i}^{s_i+\Delta t} \kappa(t)\,dt} \approx e^{-\kappa(s_i)\Delta t} = e^{-\kappa_i \Delta t}$$

$$A_i = 1 - e^{-\kappa_i \Delta t}$$

$$\tilde{T}_i = T_i^{\left(\frac{\Delta \tilde{t}}{\Delta t}\right)}$$

$$\tilde{A}_i = 1 - (1 - A_i)^{\left(\frac{\Delta \tilde{t}}{\Delta t}\right)}$$

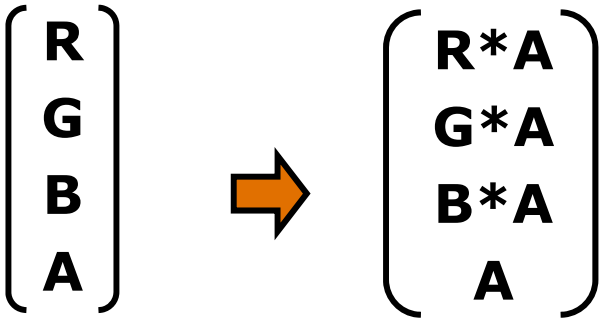
opacity correction formula

beware that usually this is done *for each different scalar value* (every transfer function entry), not actually at spatial positions/intervals *i*

Associated Colors

Associated (or *"opacity-weighted"* colors) are often used in compositing equations

Every color is *pre-multiplied* by its corresponding opacity



Our compositing equations assume associated colors!

Important: After opacity correction (updating all opacities accordingly), all *associated colors* must also be updated accordingly! (or combined/multiplied correctly on-the-fly!)

Interlude: “Self-Absorption” (1)

Our previous derivation of the discretization of the volume rendering integral skipped over a small but important detail:

Emission and absorption combine in the same segment (interval)!

$$I(s_n) = I_0\, e^{-\int_{s_0}^{s_n} \kappa(t)\,\mathrm{d}t} + \sum_{i=0}^{n-1} I(s_i, s_{i+1})\, e^{-\int_{s_{i+1}}^{s_n} \kappa(t)\,\mathrm{d}t}.$$

$$I(s_i, s_{i+1}) = \int_{s_i}^{s_{i+1}} q(s)\, e^{-\int_{s}^{s_{i+1}} \kappa(t)\,\mathrm{d}t}\,\mathrm{d}s,$$

Interlude: “Self-Absorption” (2)

Piecewise constant approximation, *but* correct integration

$$I(s_i, s_{i+1}) = \int_{s_i}^{s_{i+1}} q(s)\, e^{-\int_{s}^{s_{i+1}} \kappa(t)\,\mathrm{d}t}\,\mathrm{d}s,$$

not piecewise constant q, κ

$$\bar{I}(s_i, s_{i+1}) = \int_{s_i}^{s_{i+1}} q_i\, e^{-\kappa_i \cdot (s_{i+1} - s)}\,\mathrm{d}s,$$

piecewise constant q, κ

$$q(s) = q_i \quad \forall s \in [s_i, s_{i+1})$$
$$\kappa(s) = \kappa_i \quad \forall s \in [s_i, s_{i+1})$$

$$I(s_i, s_{i+1}) \approx \bar{I}(s_i, s_{i+1})$$

Interlude: “Self-Absorption” (3)

Piecewise constant approximation, *but* correct integration

$$\bar{I}(s_i, s_{i+1}) = q_i e^{-\kappa_i s_{i+1}} \int_{s_i}^{s_{i+1}} e^{\kappa_i s} \, \mathrm{d}s.$$

$$= q_i e^{-\kappa_i s_{i+1}} \left(\frac{1}{\kappa_i} e^{\kappa_i s_{i+1}} - \frac{1}{\kappa_i} e^{\kappa_i s_i} \right)$$

$$= \frac{q_i}{\kappa_i} \left(1 - e^{-\kappa_i \cdot (s_{i+1} - s_i)} \right),$$

$$= \frac{q_i}{\kappa_i} \left(1 - e^{-\kappa_i \Delta t} \right).$$

Associated Colors in Volume Rendering

Standard emission-absorption optical model

- Only *one kind of particle*: the same particles that absorb light, emit light
- Aha! Therefore lower absorption means lower emission as well

Light observed from (in front of) segment *i* (without any light behind it):

$$C_i = \frac{q_i}{\kappa_i}\left(1 - e^{-\kappa_i \Delta t}\right) = \hat{C}_i A_i$$

$$q_i := \hat{C}_i \kappa_i$$

$$A_i := 1 - e^{-\kappa_i \Delta t}$$

$$\lim_{\kappa_i \to 0} q_i \frac{\left(1 - e^{-\kappa_i \Delta t}\right)}{\kappa_i} = \lim_{\kappa_i \to 0} \hat{C}_i \left(1 - e^{-\kappa_i \Delta t}\right) = 0$$

$$\lim_{\kappa_i \to \infty} q_i \frac{\left(1 - e^{-\kappa_i \Delta t}\right)}{\kappa_i} = \lim_{\kappa_i \to \infty} \hat{C}_i \left(1 - e^{-\kappa_i \Delta t}\right) = \hat{C}_i$$

hold $\hat{C}_i$ fixed! (as a fixed ratio)

$$q_i := \hat{C}_i \kappa_i$$

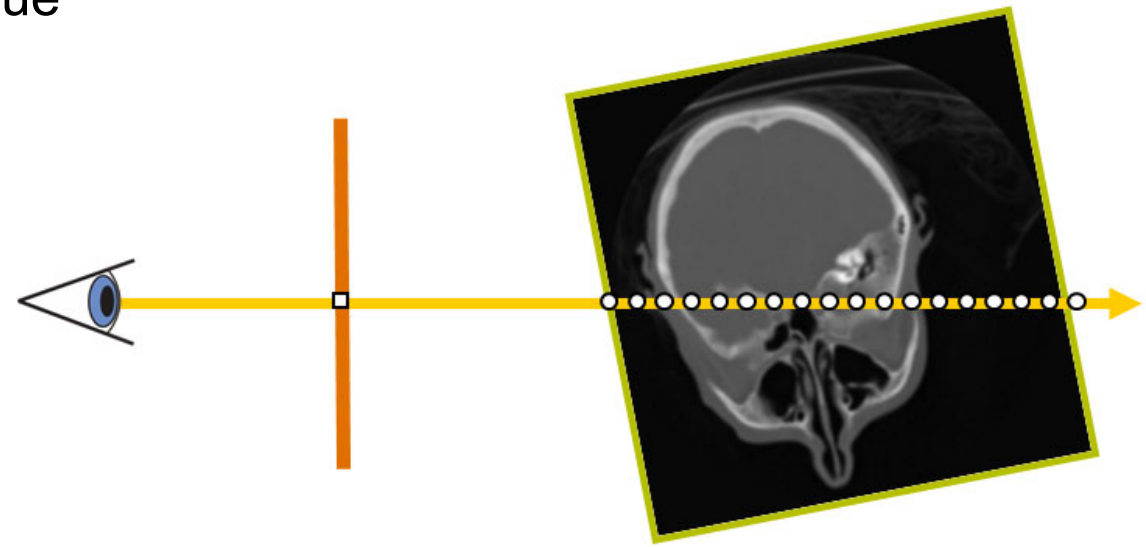
Implementation

Implementation

Ray setup

Loop over ray

- Resample scalar value
- Classification
- Shading
- Compositing

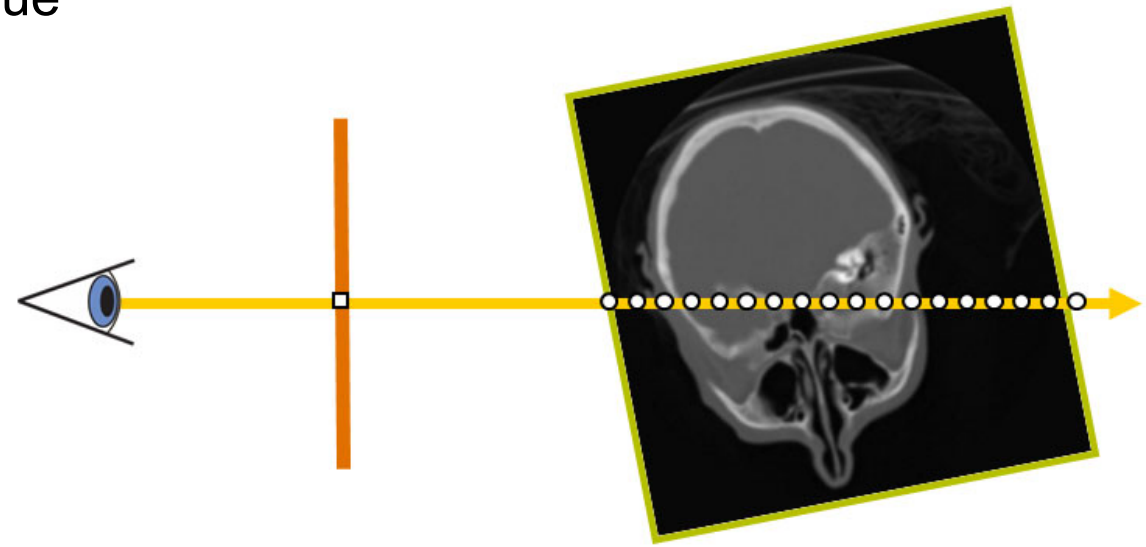


Implementation

Ray setup

Loop over ray

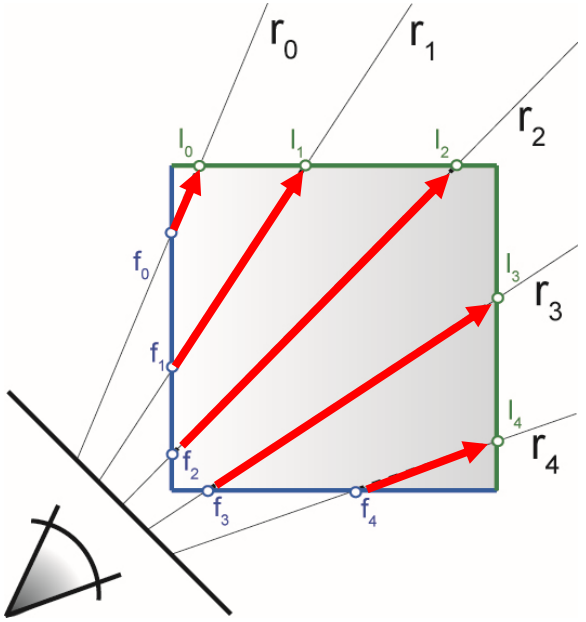
- Resample scalar value
- Classification
- Shading
- Compositing



Ray Setup

Two main approaches:

- Procedural ray/box intersection
  [Röttger et al., 2003], [Green, 2004]
- Rasterize bounding box
  [Krüger and Westermann, 2003]

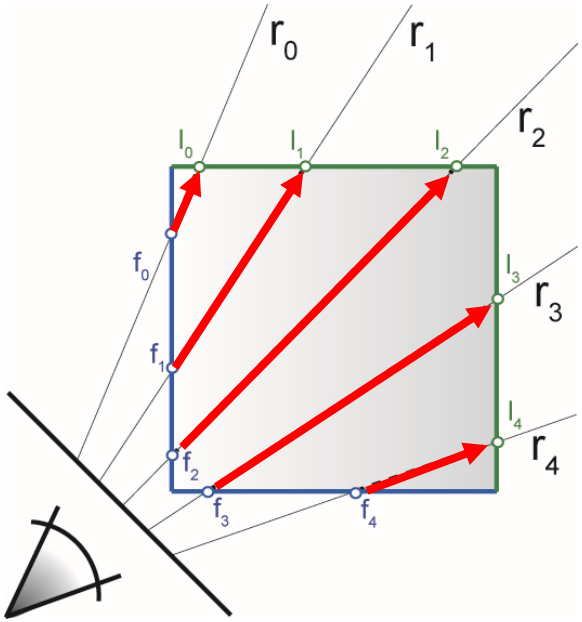


Some possibilities

- Ray start position and exit check
- Ray start position and exit position
- Ray start position and direction vector

Procedural Ray Setup/Termination

- Everything handled in the fragment shader / CUDA kernel
- Procedural ray / bounding box intersection

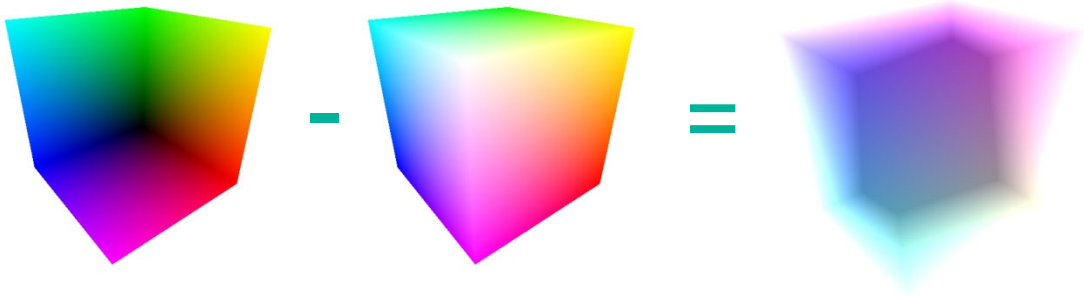


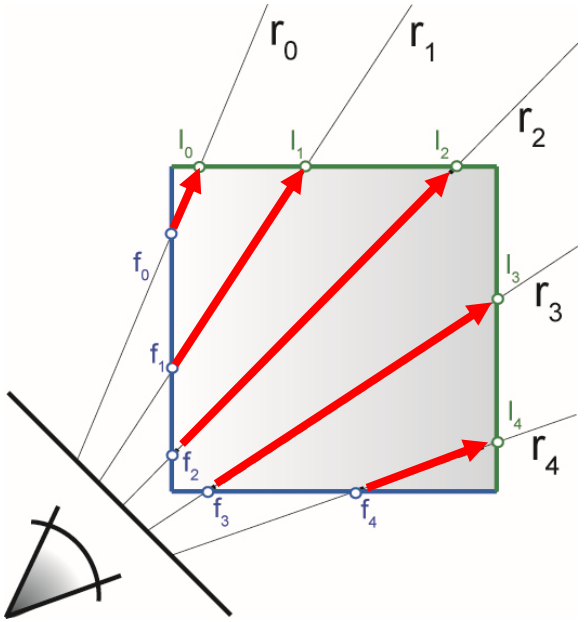
- Ray is given by camera position and volume entry position
- Exit criterion needed

- Pro: simple and self-contained
- Con: full computational load per-pixel/fragment

Rasterization-Based Ray Setup

- Fragment == ray
- Need ray start pos, direction vector
- Rasterize bounding box

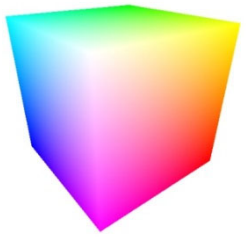


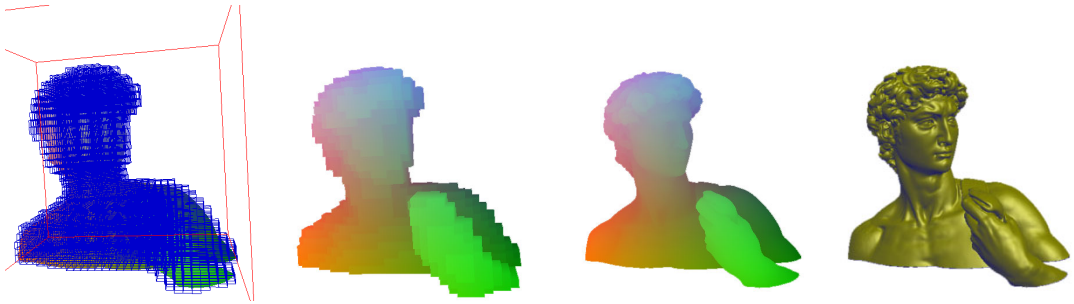


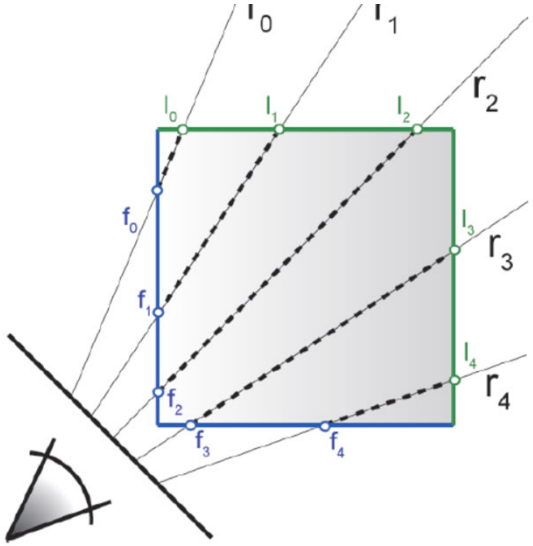
- Identical for orthogonal and perspective projection!

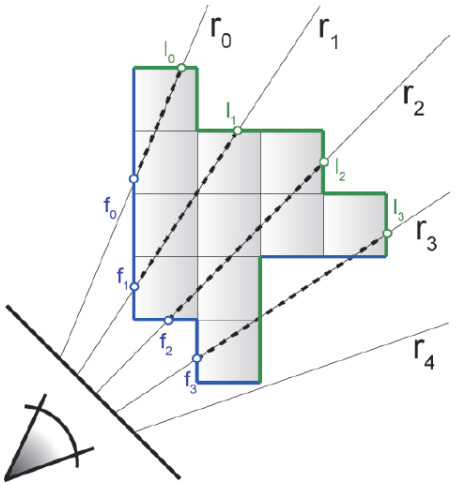
Object-Order Empty Space Skipping

Modify initial rasterization step





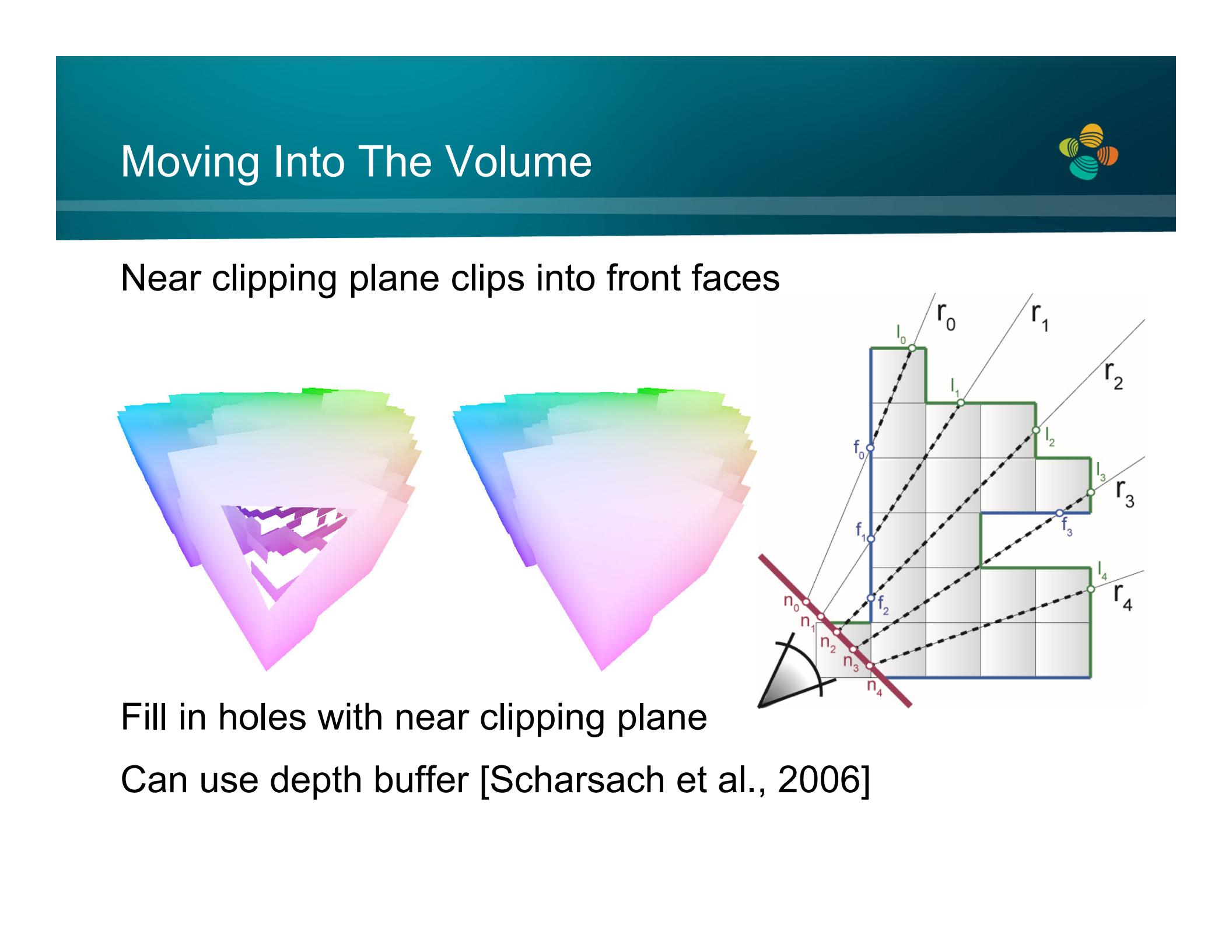




rasterize bounding box

rasterize “tight" bounding geometry

Moving Into The Volume



Near clipping plane clips into front faces

Fill in holes with near clipping plane

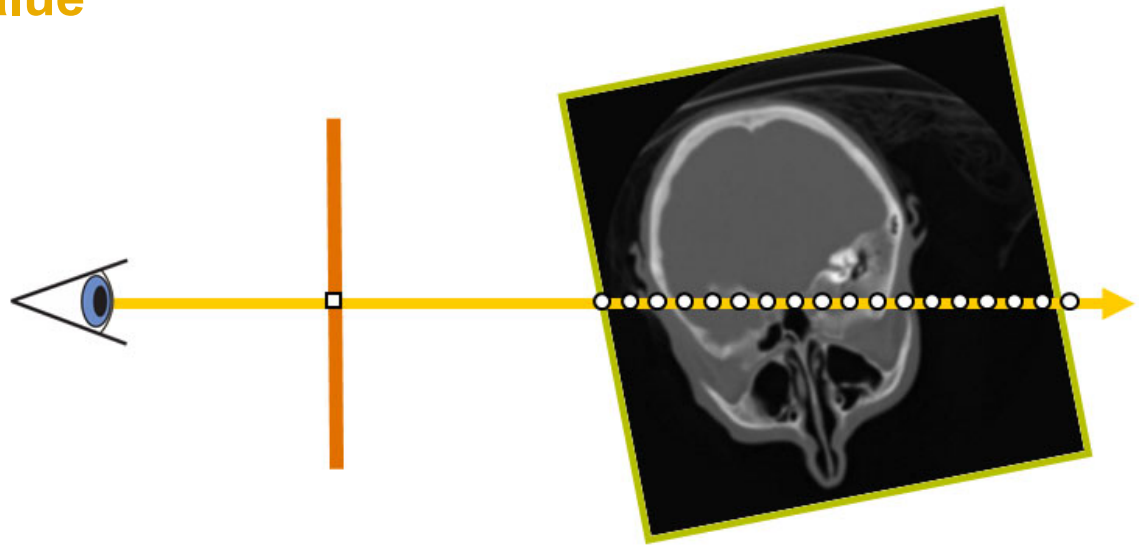
Can use depth buffer [Scharsach et al., 2006]

Implementation

Ray setup

Loop over ray

- **Resample scalar value**
- **Classification**
- Shading
- Compositing



Classification – Transfer Functions

During Classification the user defines the ***"look"*** of the data.

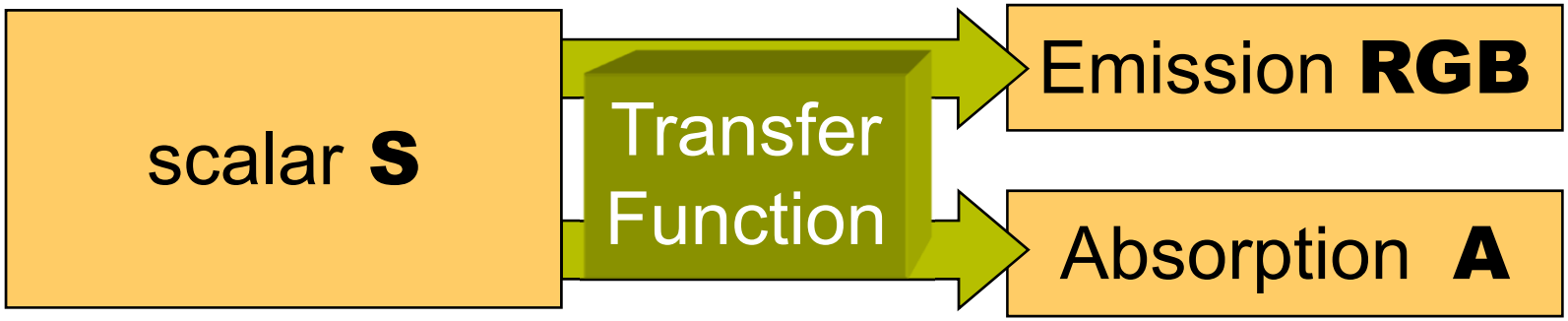
- Which parts are transparent?
- Which parts have what color?

Classification – Transfer Functions

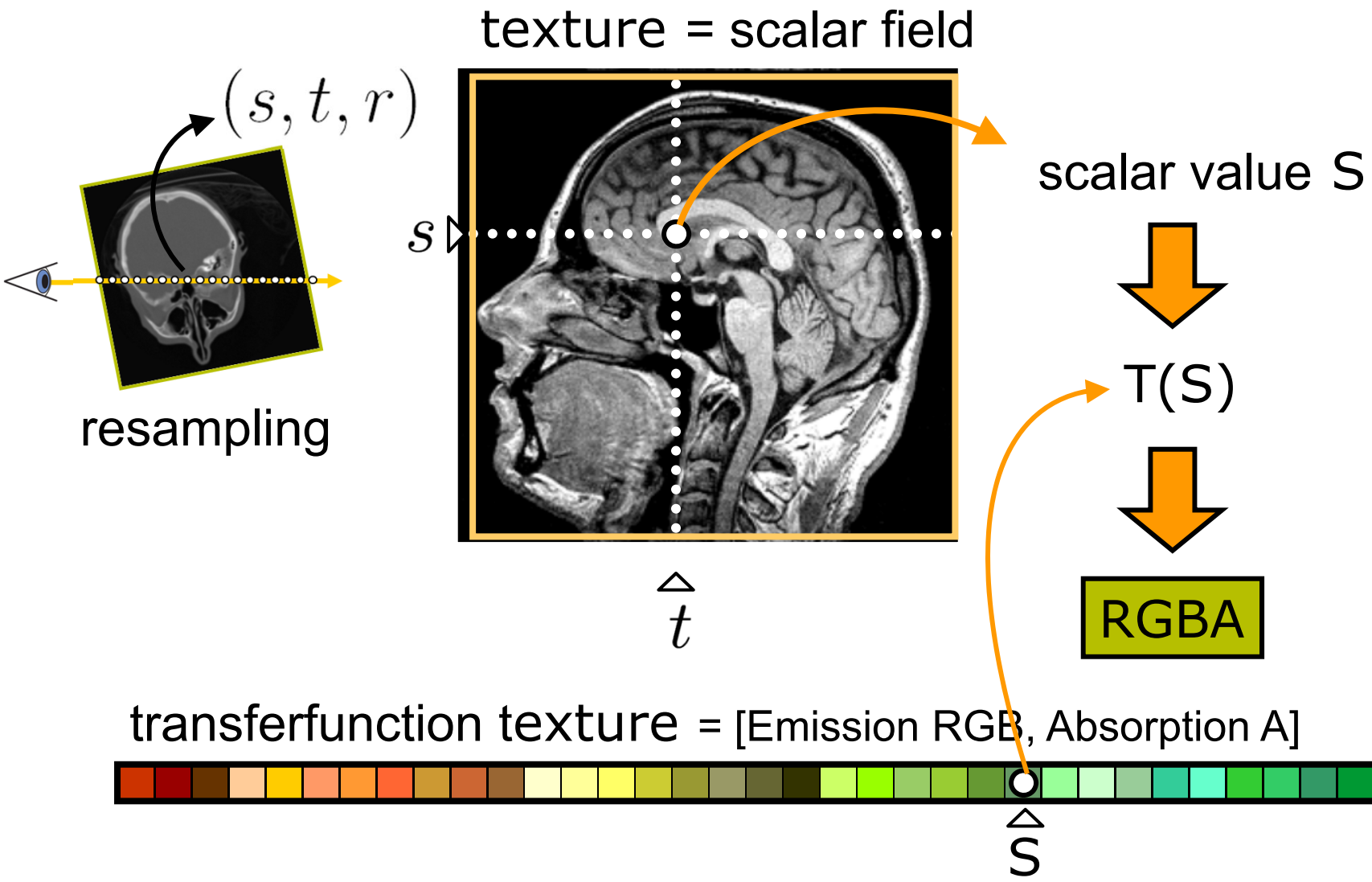
During Classification the user defines the ***"look"*** of the data.

- Which parts are transparent?
- Which parts have what color?

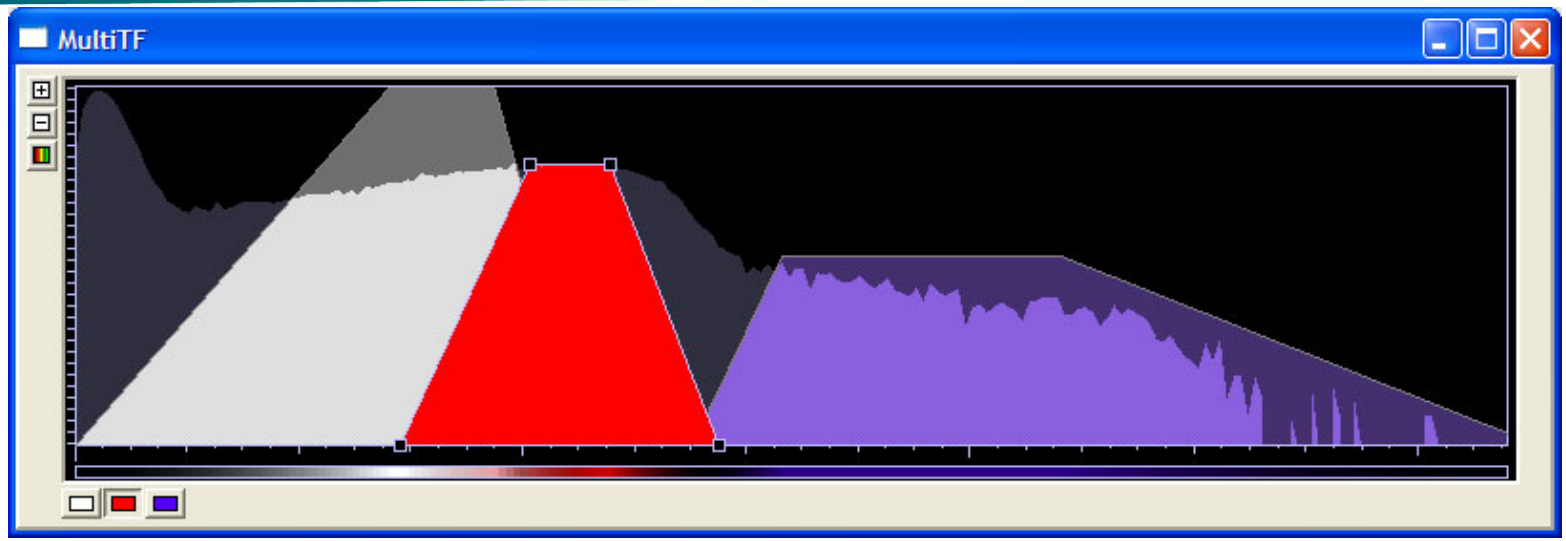
The user defines a ***transfer function.***

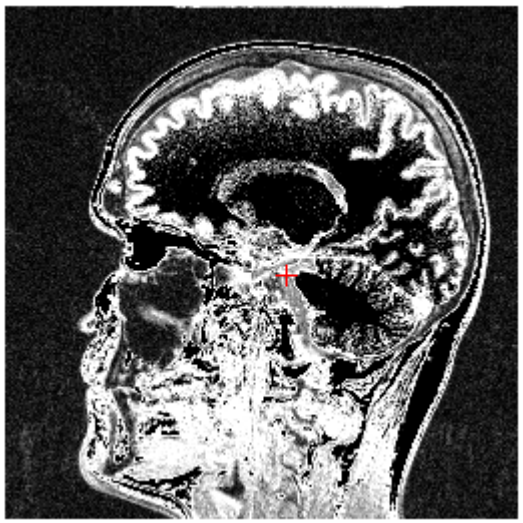


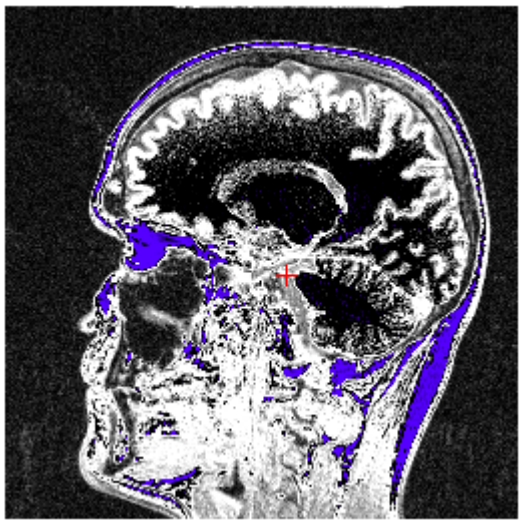
1D Transfer Functions

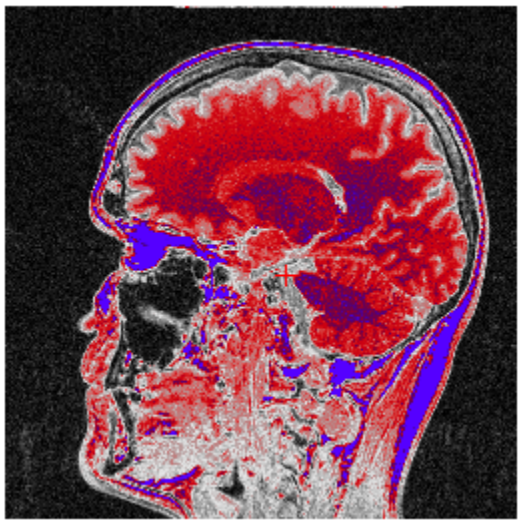


1D Transfer Functions









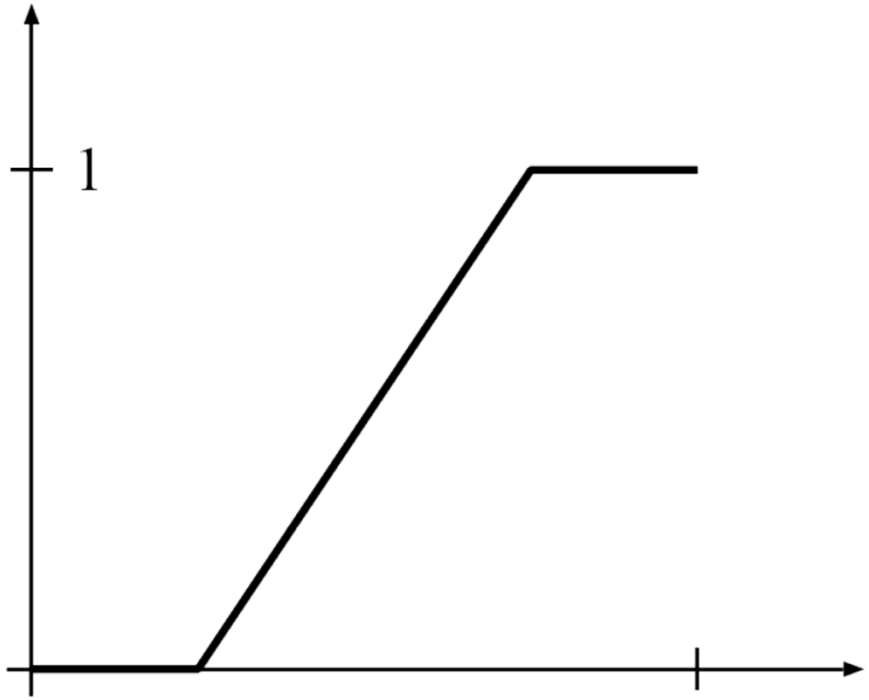
Applying Transfer Function: Cg Example

```
// Cg fragment program for post-classification
// using 3D textures
float4 main (float3 texUV : TEXCOORD0,
             uniform sampler3D volume_texture,
             uniform sampler1D transfer_function) :
   COLOR
{
   float index = tex3D(volume_texture, texUV);
   float4 result = tex1D(transfer_function, index);
   return result;
}
```

Windowing Transfer Function

Map input scalar range to output intensity range

- Select scalar range of interest
- Adjust contrast



Thank you.

Thanks for material

- Helwig Hauser
- Eduard Gröller
- Daniel Weiskopf
- Torsten Möller
- Ronny Peikert
- Philipp Muigg
- Christof Rezk-Salama