King Abdullah University of Science and Technology

KAUST

CS 247 – Scientific Visualization
Lecture 21: Volume Visualization, Pt. 8

Markus Hadwiger, KAUST

Reading Assignment #12 (until Apr 27)

Read (required):

- Data Visualization book
  - Chapter 6 (Vector Visualization)
    - Beginning (before 6.1)
    - Chapters 6.2, 6.3, 6.5
- More general vector field basics (the book is not very precise on the basics)

  `https://en.wikipedia.org/wiki/Vector_field`

Read (optional):

- Paper:
  Bruno Jobard and Wilfrid Lefer
  Creating Evenly-Spaced Streamlines of Arbitrary Density,

  `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.9498`

Volume Rendering

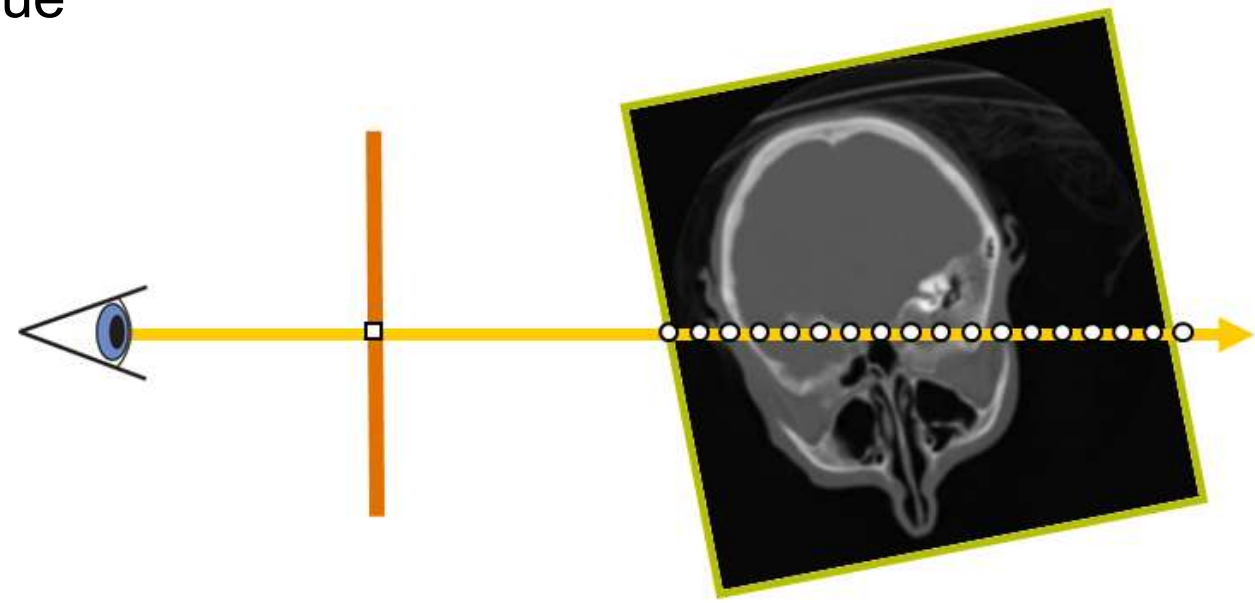
Implementation

Implementation

Ray setup

Loop over ray

- Resample scalar value
- Classification
- Shading
- Compositing

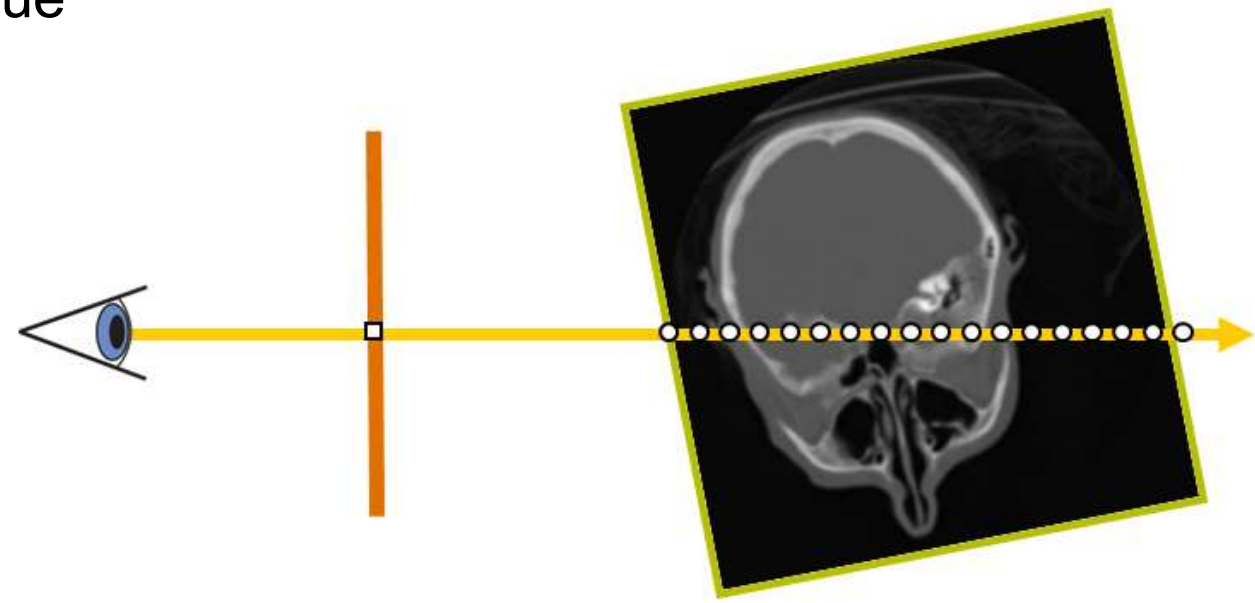


Implementation

Ray setup

Loop over ray

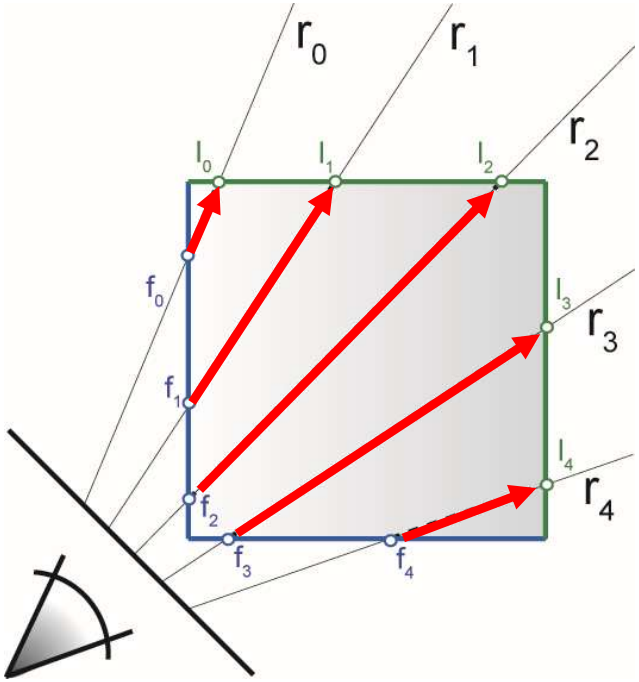
- Resample scalar value
- Classification
- Shading
- Compositing



Ray Setup

Two main approaches:

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



Some possibilities

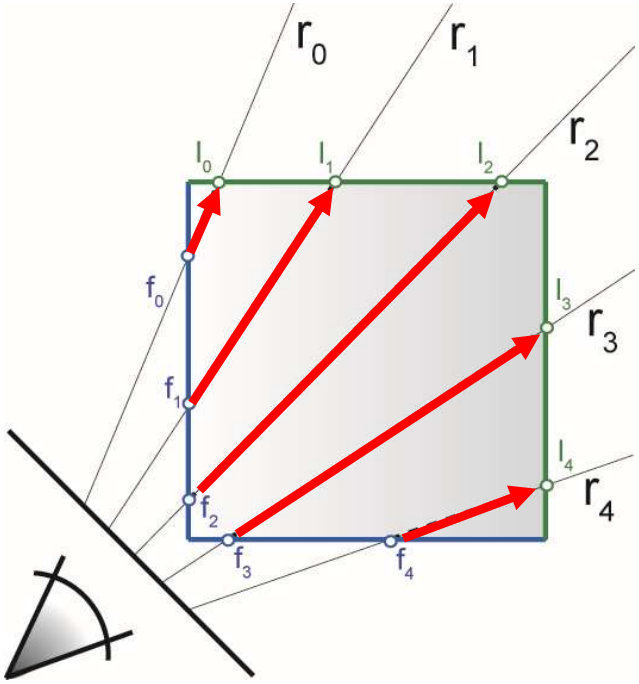
- Ray start position and exit check
- Ray start position and exit position
- Ray start position and direction vector

Procedural Ray Setup/Termination

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

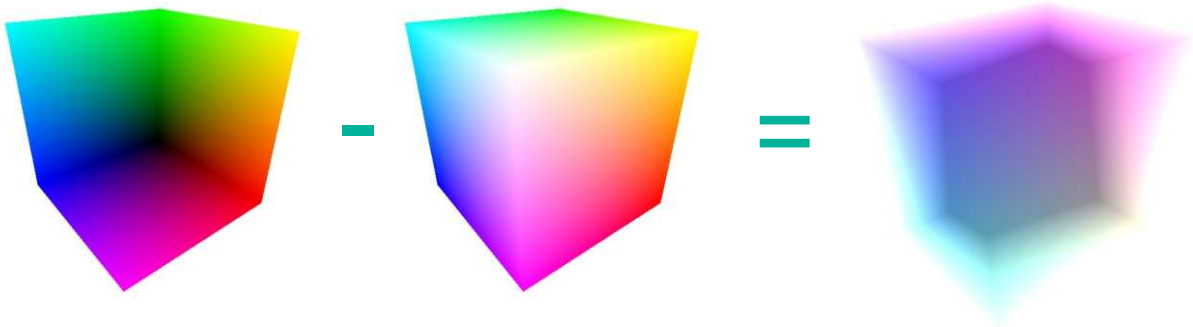
- Ray is given by camera position and volume entry position
- Exit criterion needed

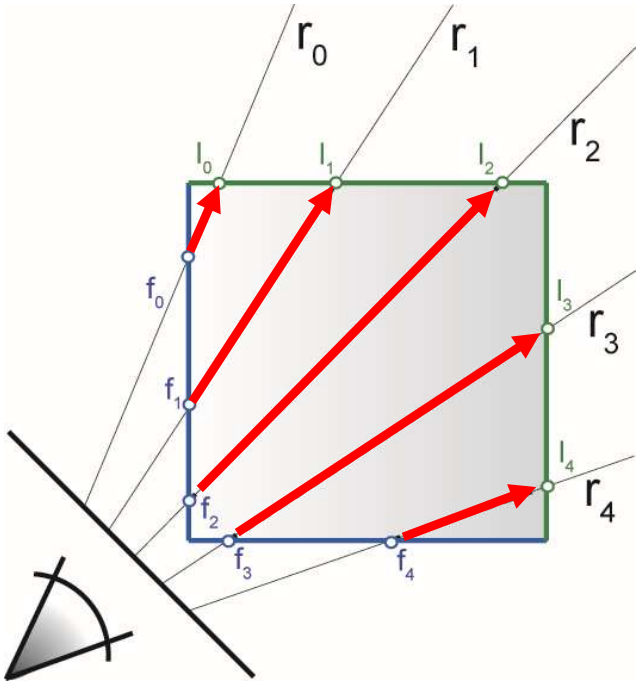
- Pro: simple and self-contained
- Con: full computational load per-pixel/fragment



Rasterization-Based Ray Setup

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

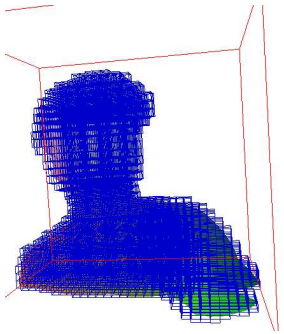


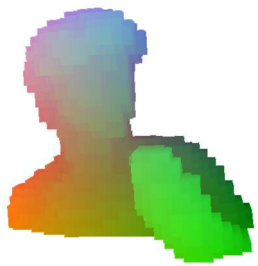


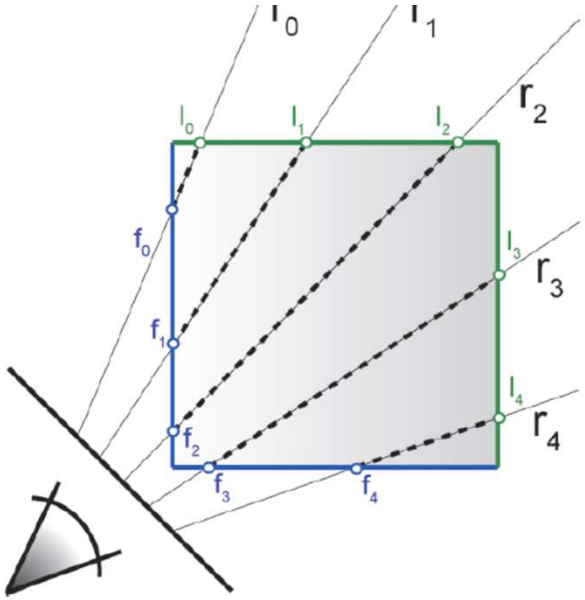
- Identical for orthogonal and perspective projection!

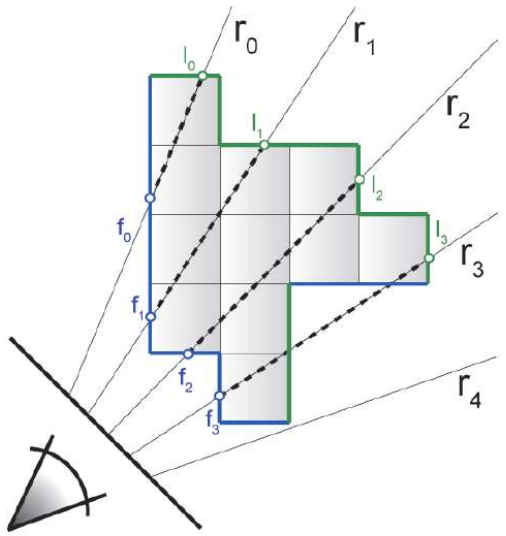
Object-Order Empty Space Skipping

Modify initial rasterization step





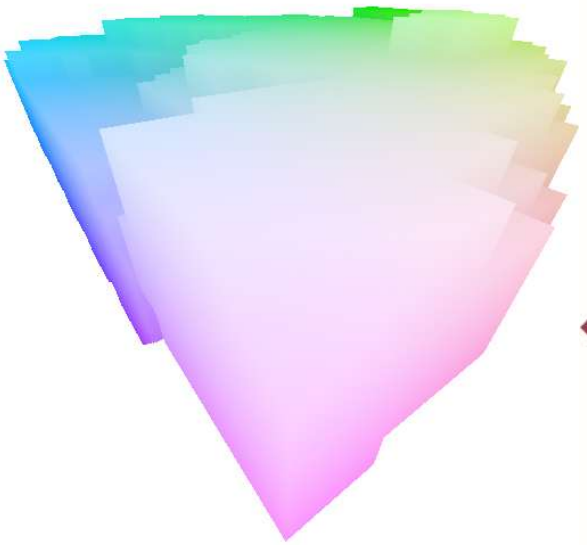




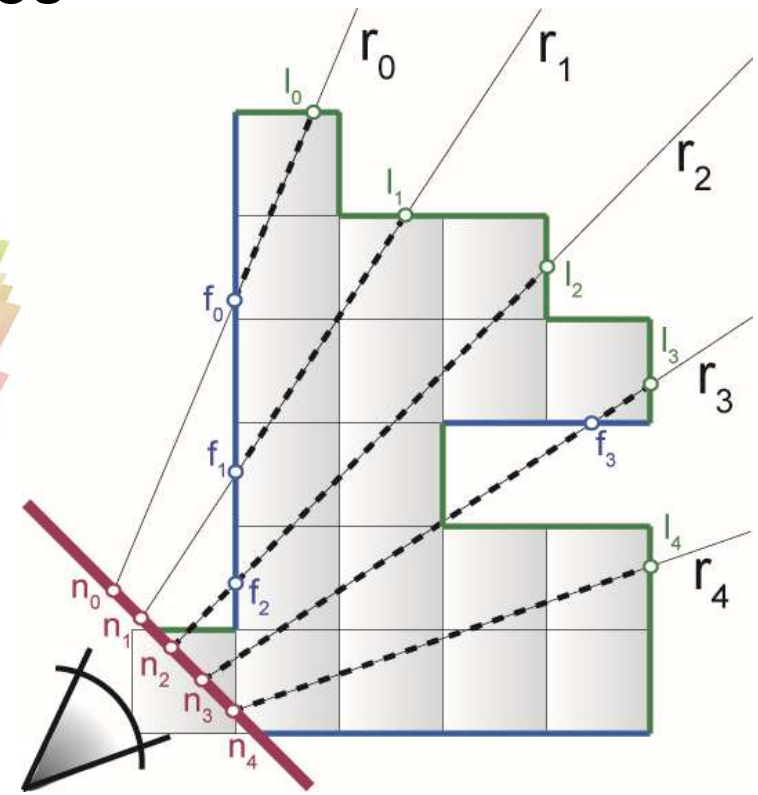
rasterize bounding box

rasterize “tight" bounding geometry

Moving Into The Volume



Near clipping plane clips into front faces



Fill in holes with near clipping plane

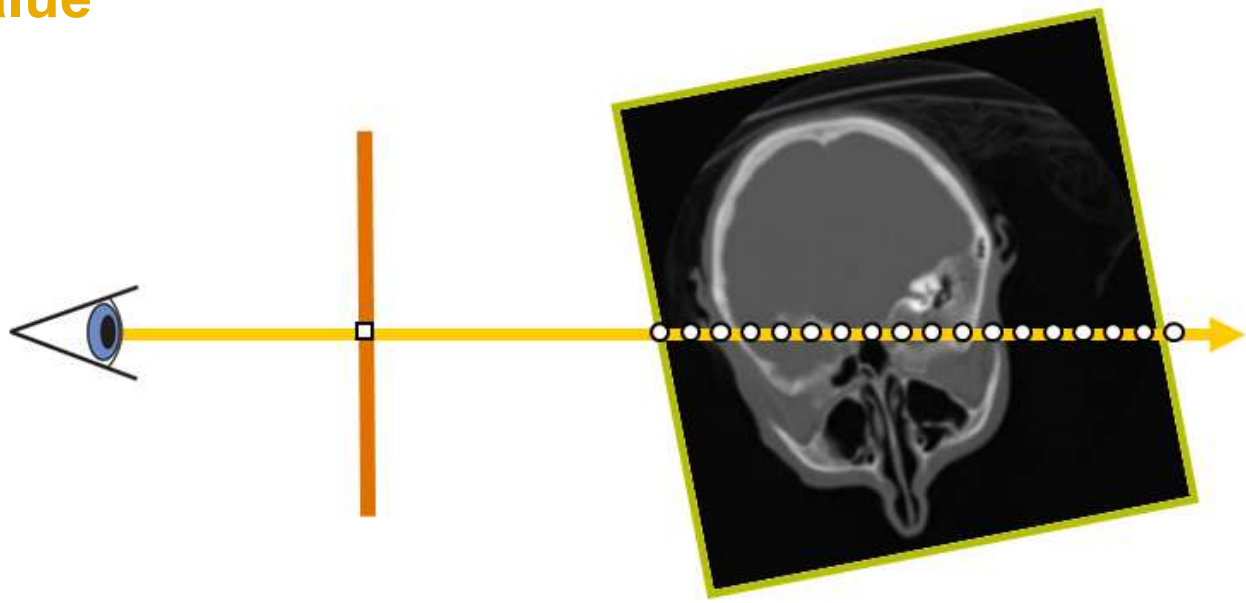
Can use depth buffer [Scharsach et al., 2006]

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.

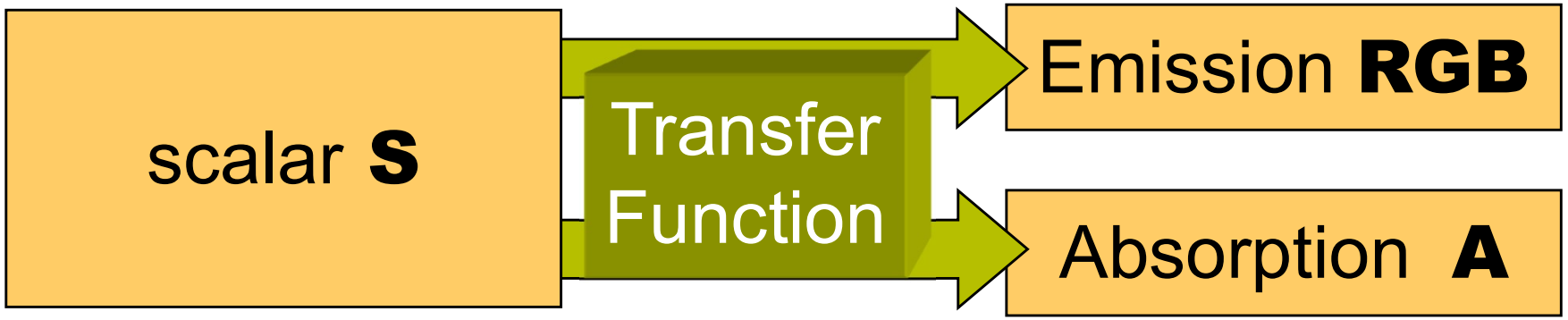
- Which parts are transparent?
- Which parts have what color?

Classification – Transfer Functions

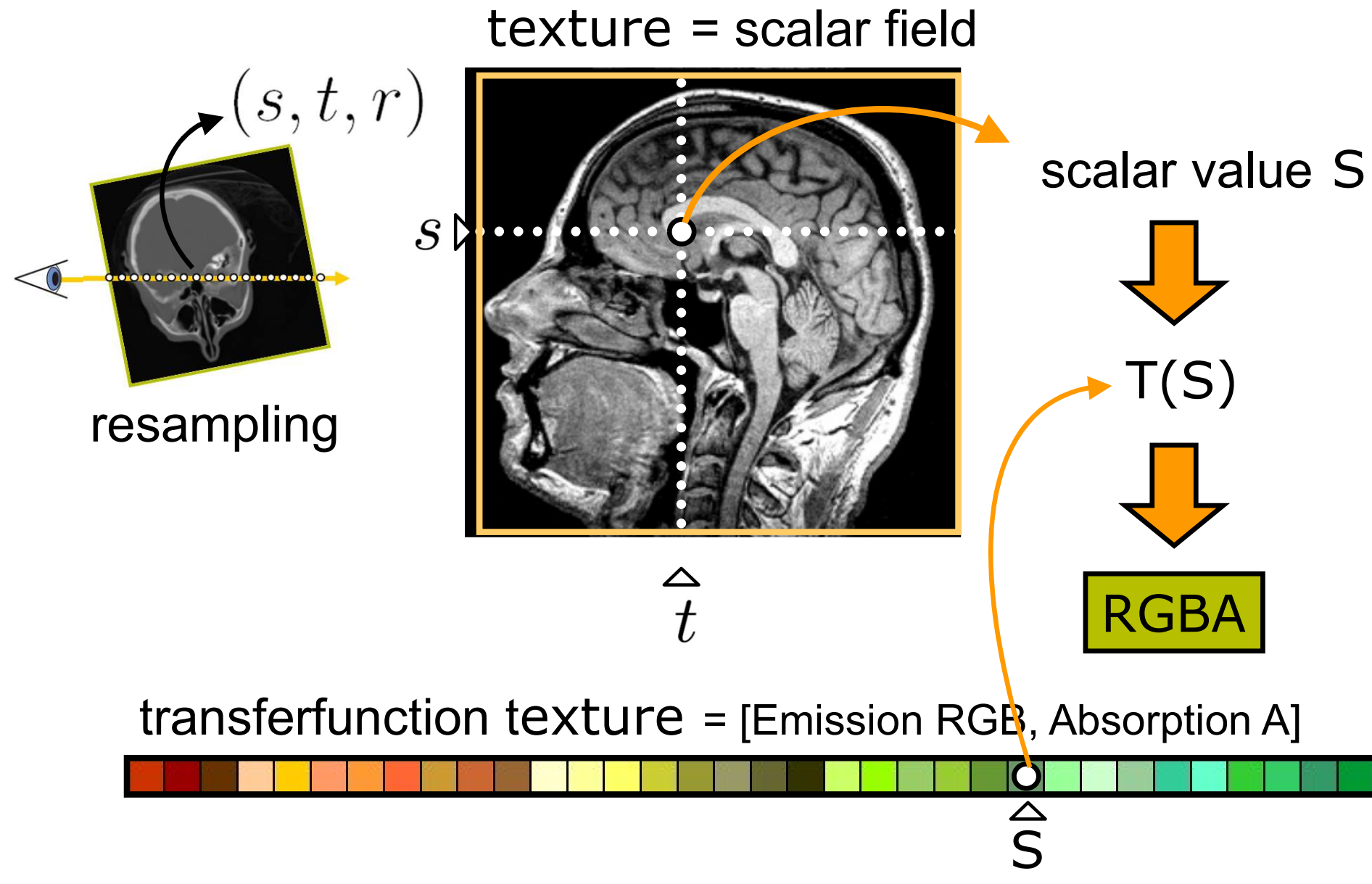
During Classification the user defines the ***"look"*** of the data.

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

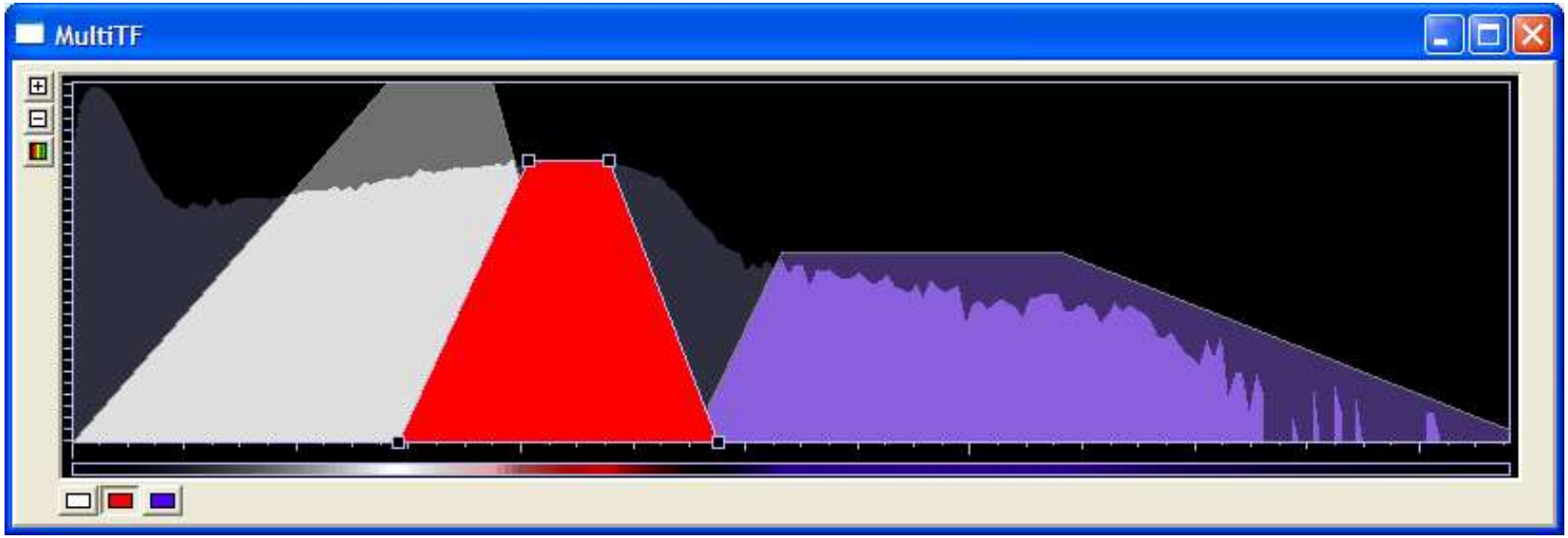
The user defines a ***transfer function.***

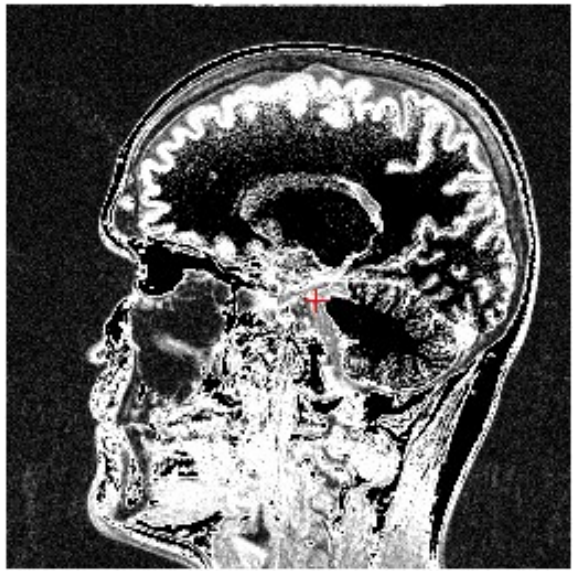


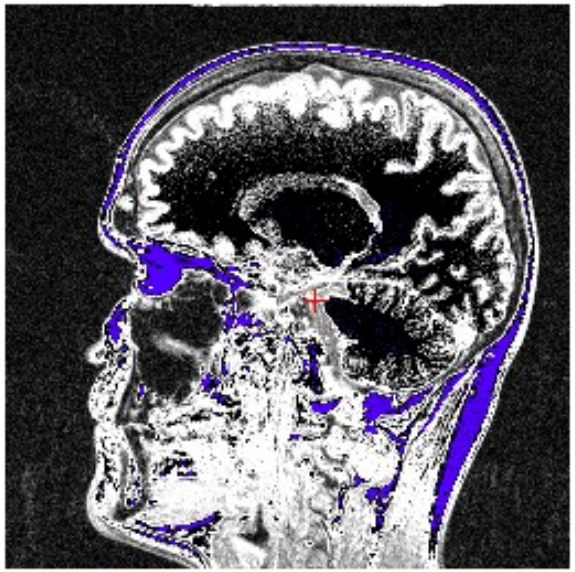
1D Transfer Functions

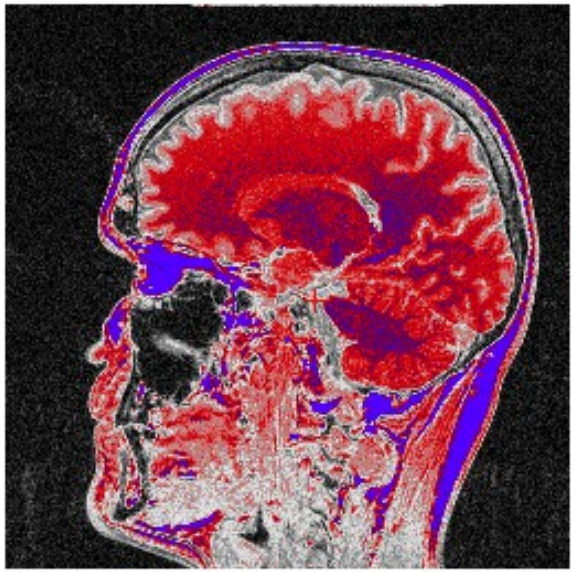


1D Transfer Functions









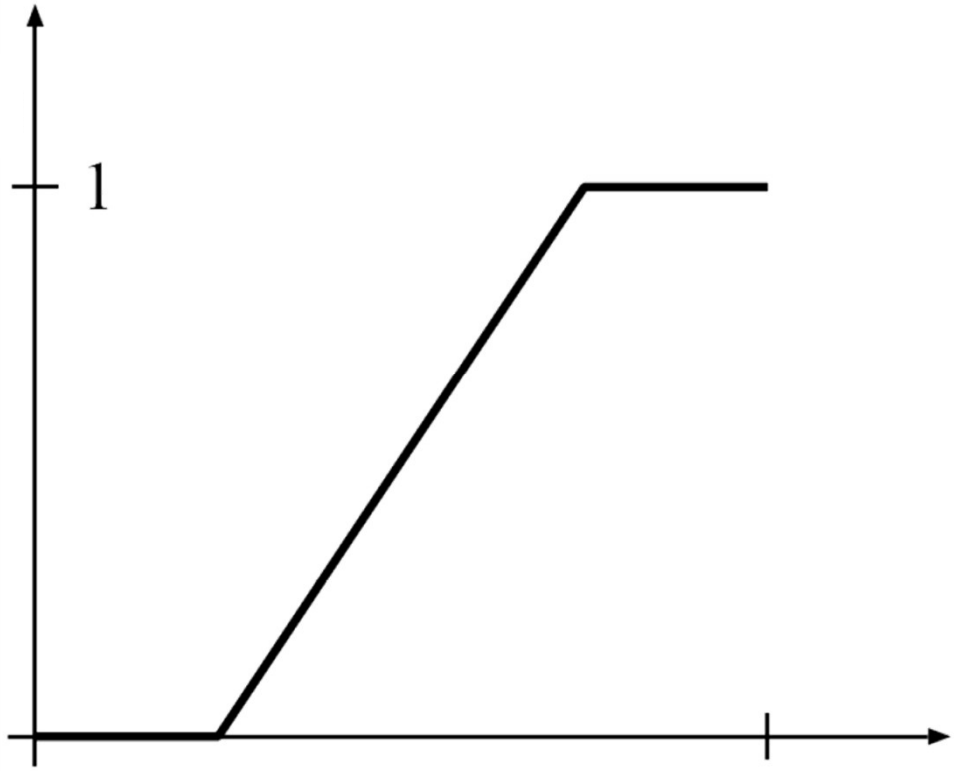
Applying Transfer Function: Cg Example

```
// 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

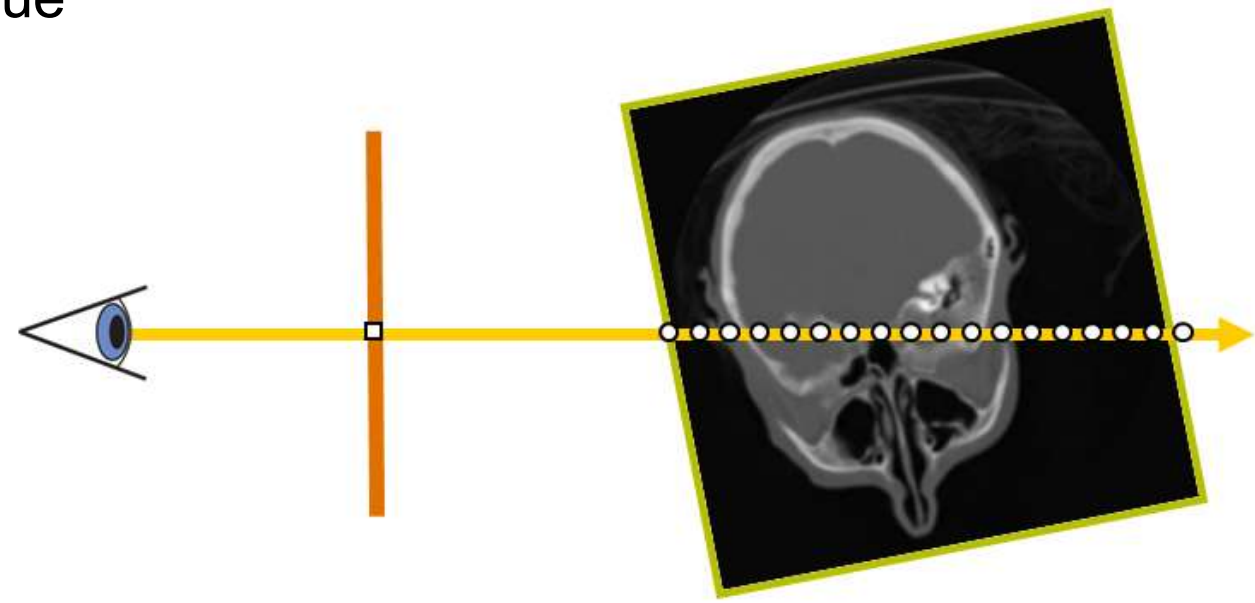


Implementation

Ray setup

Loop over ray

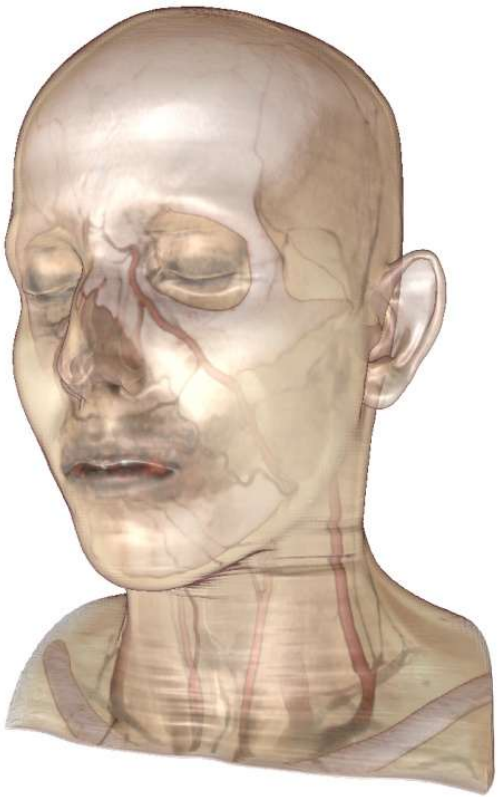
- Resample scalar value
- Classification
- **Shading**
- Compositing



Volume Shading

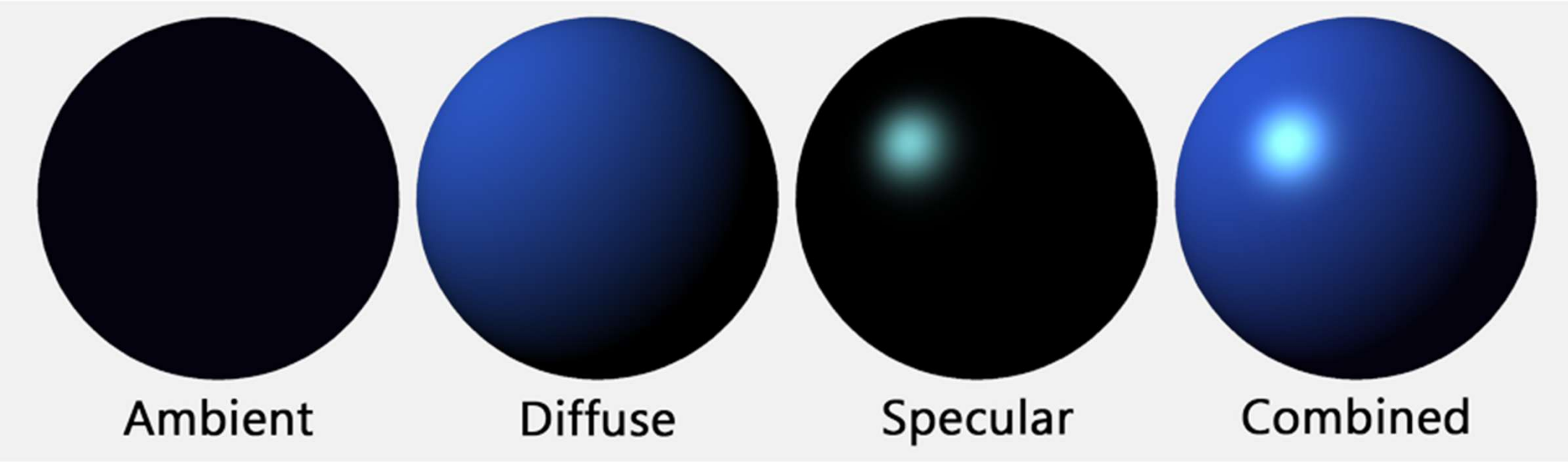
Local illumination vs. global illumination

- Gradient-based or gradient-less
- Shadows, (multiple) scattering, …



Local Illumination Model: Phong Lighting Model

$$\mathbf{I}_{\text{Phong}} = \mathbf{I}_{\text{ambient}} + \mathbf{I}_{\text{diffuse}} + \mathbf{I}_{\text{specular}}$$

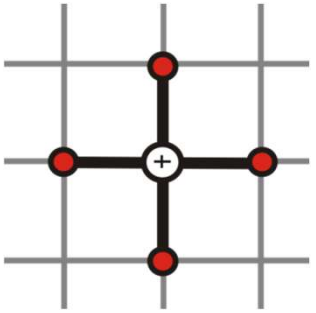


On-the-fly Gradient Estimation

$$\nabla f(x,y,z) \approx \frac{1}{2h}\begin{pmatrix} f(x+h,y,z) - f(x-h,y,z) \\ f(x,y+h,z) - f(x,y-h,z) \\ f(x,y,z+h) - f(x,y,z-h) \end{pmatrix}$$

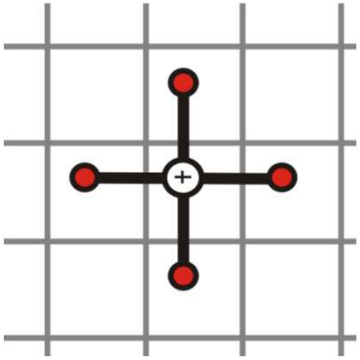
```
float3 sample1, sample2;
// six texture samples for the gradient
sample1.x = tex3D(texture,uvw-half3(DELTA,0.0,0.0)).x;
sample2.x = tex3D(texture,uvw+half3(DELTA,0.0,0.0)).x;
sample1.y = tex3D(texture,uvw-half3(0.0,DELTA,0.0)).x;
sample2.y = tex3D(texture,uvw+half3(0.0,DELTA,0.0)).x;
sample1.z = tex3D(texture,uvw-half3(0.0,0.0,DELTA)).x;
sample2.z = tex3D(texture,uvw+half3(0.0,0.0,DELTA)).x;
// central difference and normalization
float3 N = normalize(sample2-sample1);
```

On-The-Fly Gradients



Reduce texture memory consumption!

Central differences before and after linear interpolation of values at grid points yield the same results



Caveat: texture filter precision

Filter kernel methods are expensive, but:

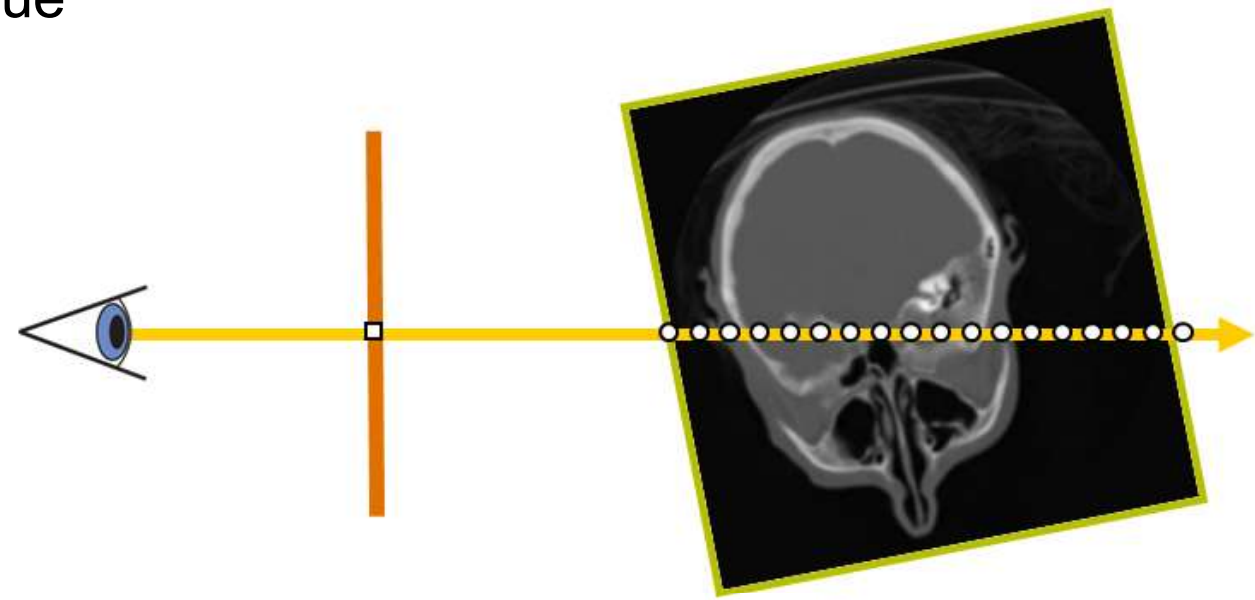
Tri-cubic B-spline kernels can be used in real-time (e.g., GPU Gems 2 Chapter “Fast Third-Order Filtering”)

Implementation

Ray setup

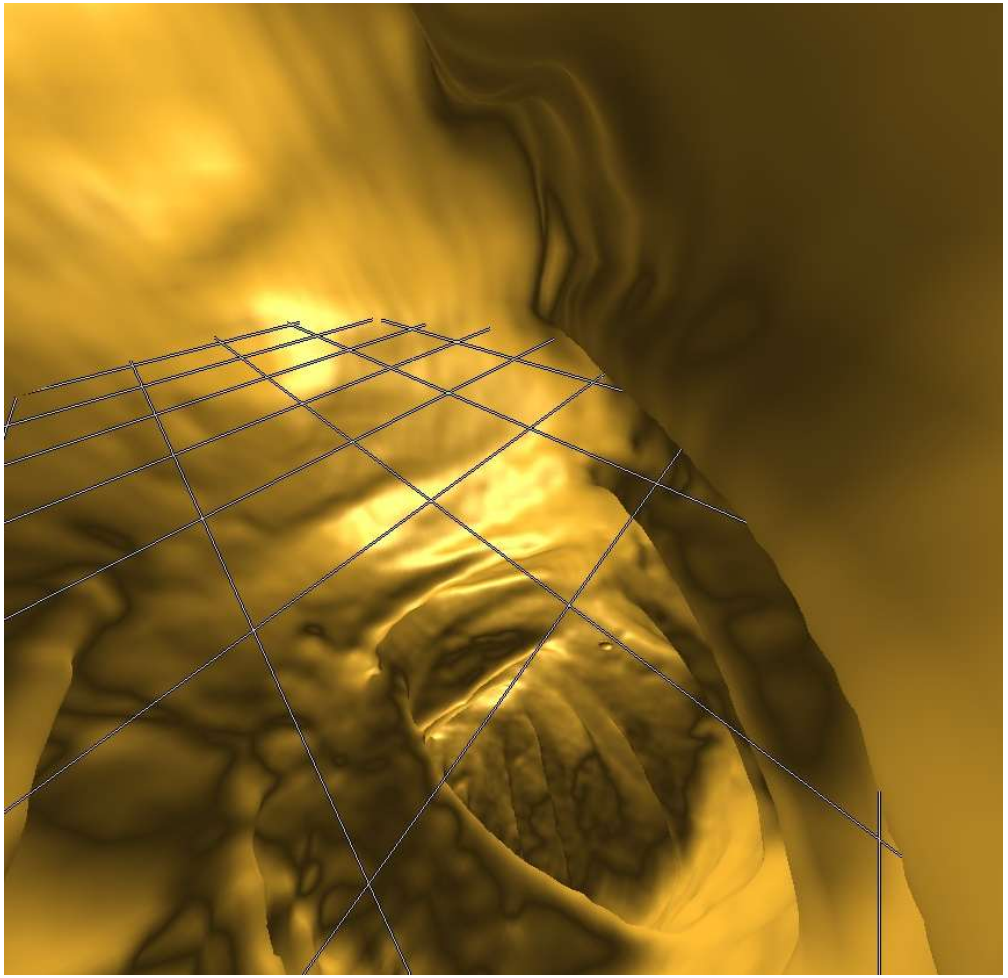
Loop over ray

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

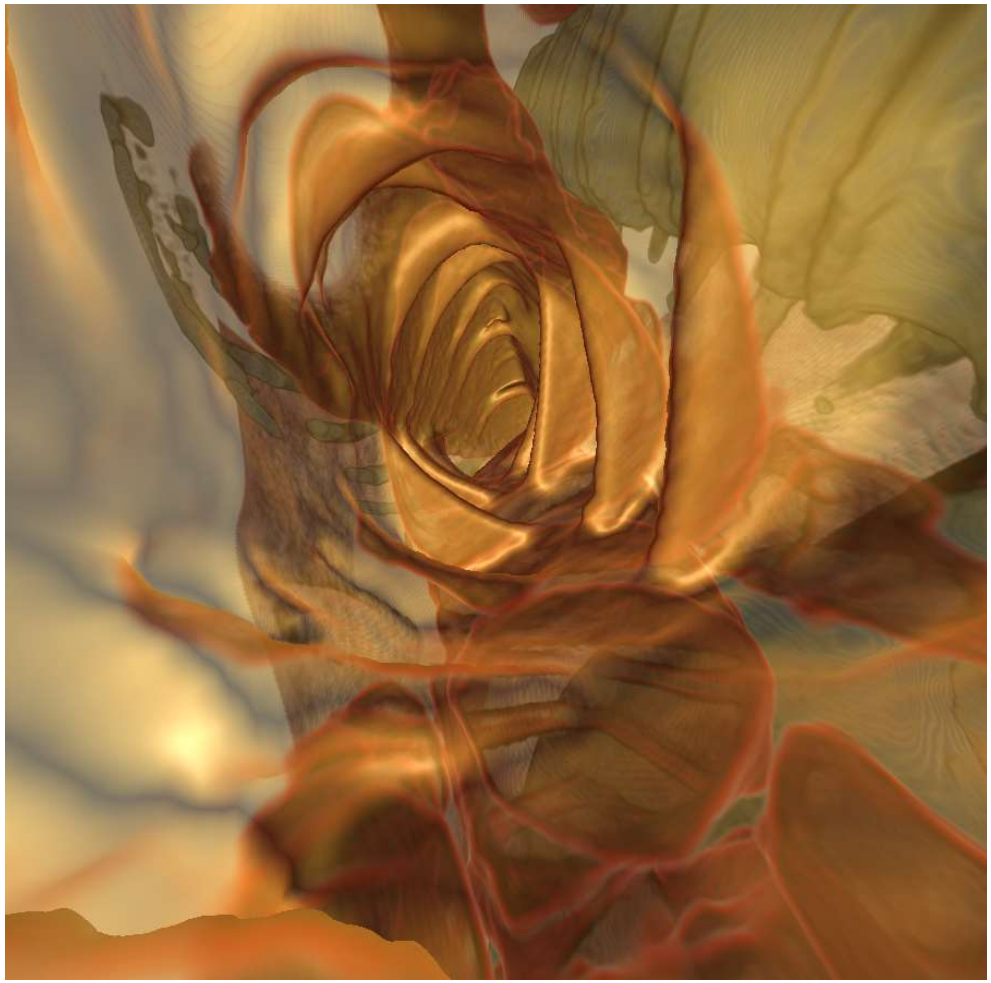


$$
\begin{aligned}
C'_i &= C'_{i+1} + (1 - A'_{i+1})C_i \\
A'_i &= A'_{i+1} + (1 - A'_{i+1})A_i
\end{aligned}
$$

Compositing



Compositing



Fragment Shader

- Rasterize front faces of volume bounding box
- Texcoords are volume position in [0,1]
- Subtract camera position
- Repeatedly check for exit of bounding box

```
// Cg fragment shader code for single-pass ray casting
float4 main(VS_OUTPUT IN, float4 TexCoord0 :  TEXCOORD0,
            uniform sampler3D SamplerDataVolume,
            uniform sampler1D SamplerTransferFunction,
            uniform float3 camera,
            uniform float stepsize,
            uniform float3 volExtentMin,
            uniform float3 volExtentMax
            ) :  COLOR
{
    float4 value;
    float scalar;
    // Initialize accumulated color and opacity
    float4 dst =  float4(0,0,0,0);
    // Determine volume entry position
    float3 position = TexCoord0.xyz;
    // Compute ray direction
    float3 direction = TexCoord0.xyz - camera;
    direction = normalize(direction);
    // Loop for ray traversal
    for (int i = 0; i < 200; i++)  // Some large number
    {
        // Data access to scalar value in 3D volume texture
        value = tex3D(SamplerDataVolume, position);
        scalar = value.a;
        // Apply transfer function
        float4 src = tex1D(SamplerTransferFunction, scalar);
        // Front-to-back compositing
        dst = (1.0-dst.a) * src + dst;
        // Advance ray position along ray direction
        position = position + direction * stepsize;
        // Ray termination:  Test if outside volume ...
        float3 temp1 = sign(position - volExtentMin);
        float3 temp2 = sign(volExtentMax - position);
        float inside = dot(temp1, temp2);
        // ...  and exit loop
        if (inside < 3.0)
            break;
    }
    return dst;
}
```

CUDA Kernel

- Image-based ray setup
  - Ray start image
  - Direction image
- Ray-cast loop
  - Sample volume
  - Accumulate color and opacity
- Terminate
- Store output

```
__global__
void RayCastCUDAKernel( float *d_output_buffer, float *d_startpos_buffer, float *d_direction_buffer )
{
    // output pixel coordinates
    dword screencoord_x = __umul24( blockIdx.x, blockDim.x ) + threadIdx.x;
    dword screencoord_y = __umul24( blockIdx.y, blockDim.y ) + threadIdx.y;

    // target pixel (RGBA-tuple) index
    dword screencoord_indx = ( __umul24( screencoord_y, cu_screensize.x ) + screencoord_x ) * 4;

    // get direction vector and ray start
    float4 dir_vec  = d_direction_buffer[ screencoord_indx ];
    float4 startpos = d_startpos_buffer[ screencoord_indx ];

    // ray-casting loop
    float4 color     = make_float4( 0.0f );
    float poscount   = 0.0f;
    for ( int i = 0; i < 8192; i++ ) {

        // next sample position in volume space
        float3 samplepos = dir_vec * poscount + startpos;
        poscount += cu_sampling_distance;

        // fetch density
        float tex_density = tex3D( cu_volume_texture, samplepos.x, samplepos.y, samplepos.z );

        // apply transfer function
        float4 col_classified = tex1D( cu_transfer_function_texture, tex_density );

        // compute (1-previous.a)*tf.a
        float prev_alpha = -color.w * col_classified.w + col_classified.w;

        // composite color and alpha
        color.xyz = prev_alpha * col_classified.xyz + color.xyz;
        color.w  += prev_alpha;

        // break if ray terminates (behind exit position or alpha threshold reached)
        if ( ( poscount > dir_vec.w ) || ( color.w > 0.98f ) ) {
            break;
        }
    }

    // store output color and opacity
    d_output_buffer[ screencoord_indx ] = __saturatef( color );
}
```

Isosurface Ray-Casting

Isosurface Ray-Casting

Isosurfaces/Level Sets

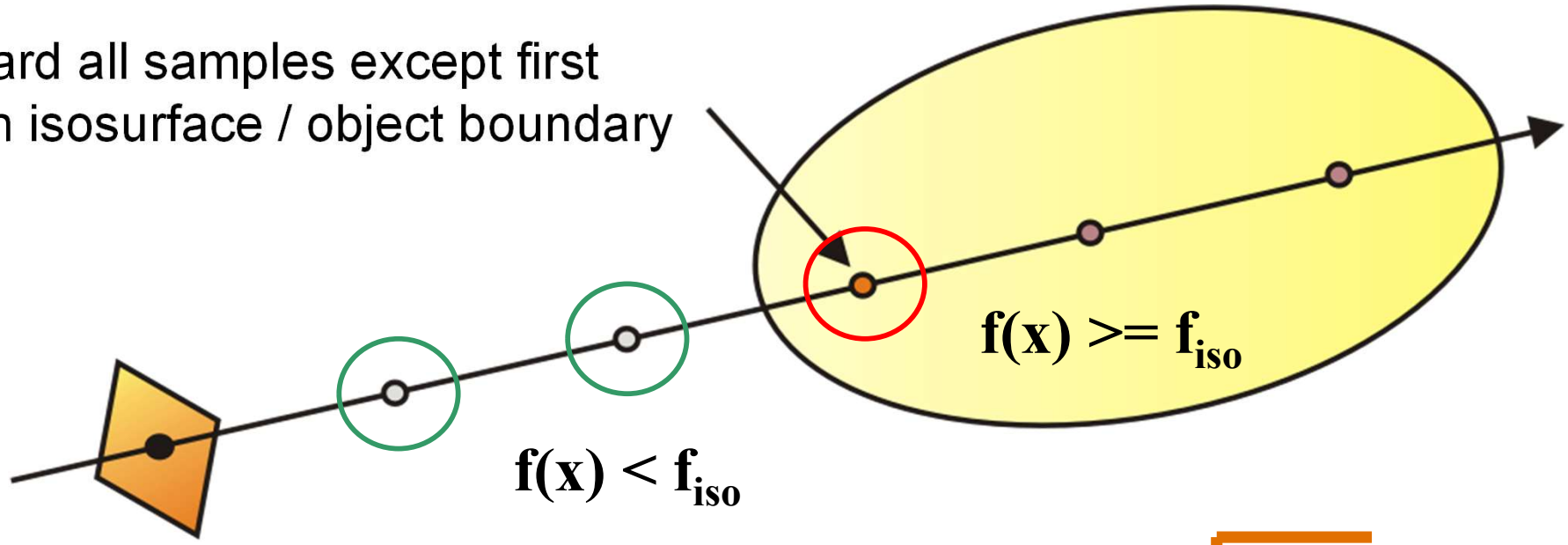
- Scanned data (fit signed distance function to points, ...)
- Signed distance fields
- CSG (constructive solid geometry) operations

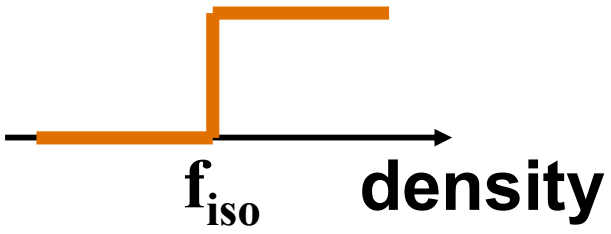
..

Isosurface Ray-Casting

Opaque isosurfaces:
only one sample contributes per ray/pixel

Discard all samples except first
hit on isosurface / object boundary





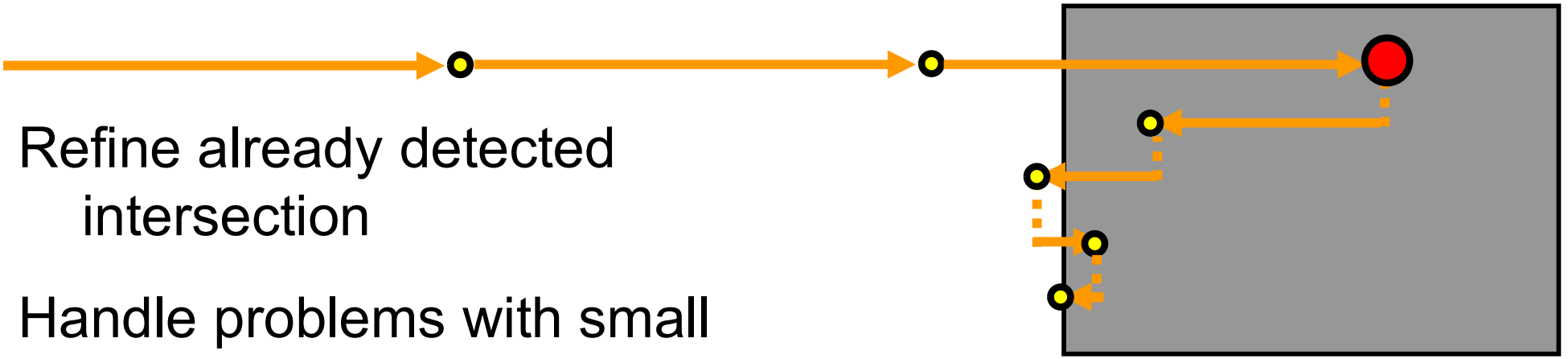
Threshold transfer function / alpha test

First hit ray casting

Intersection Refinement (1)

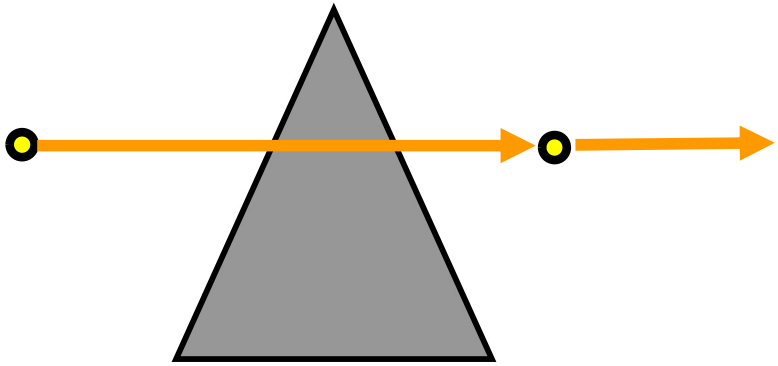
Fixed number of bisection / binary search steps

Virtually no impact on performance



Refine already detected intersection

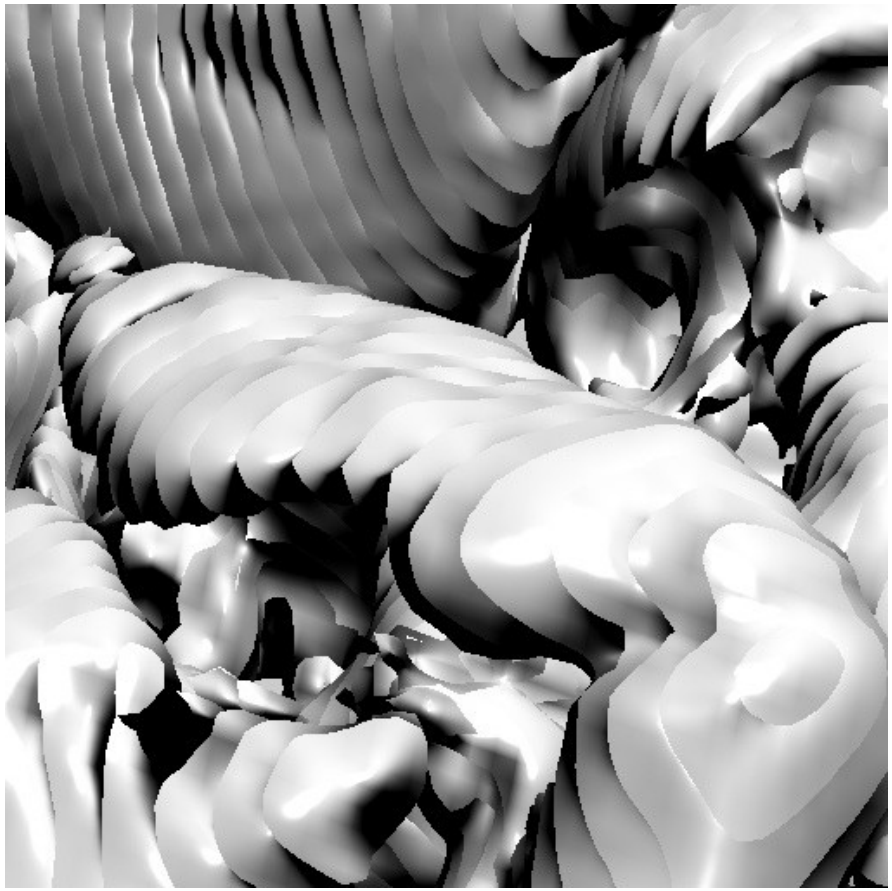
Handle problems with small features / at silhouettes with adaptive sampling



Intersection Refinement (2)

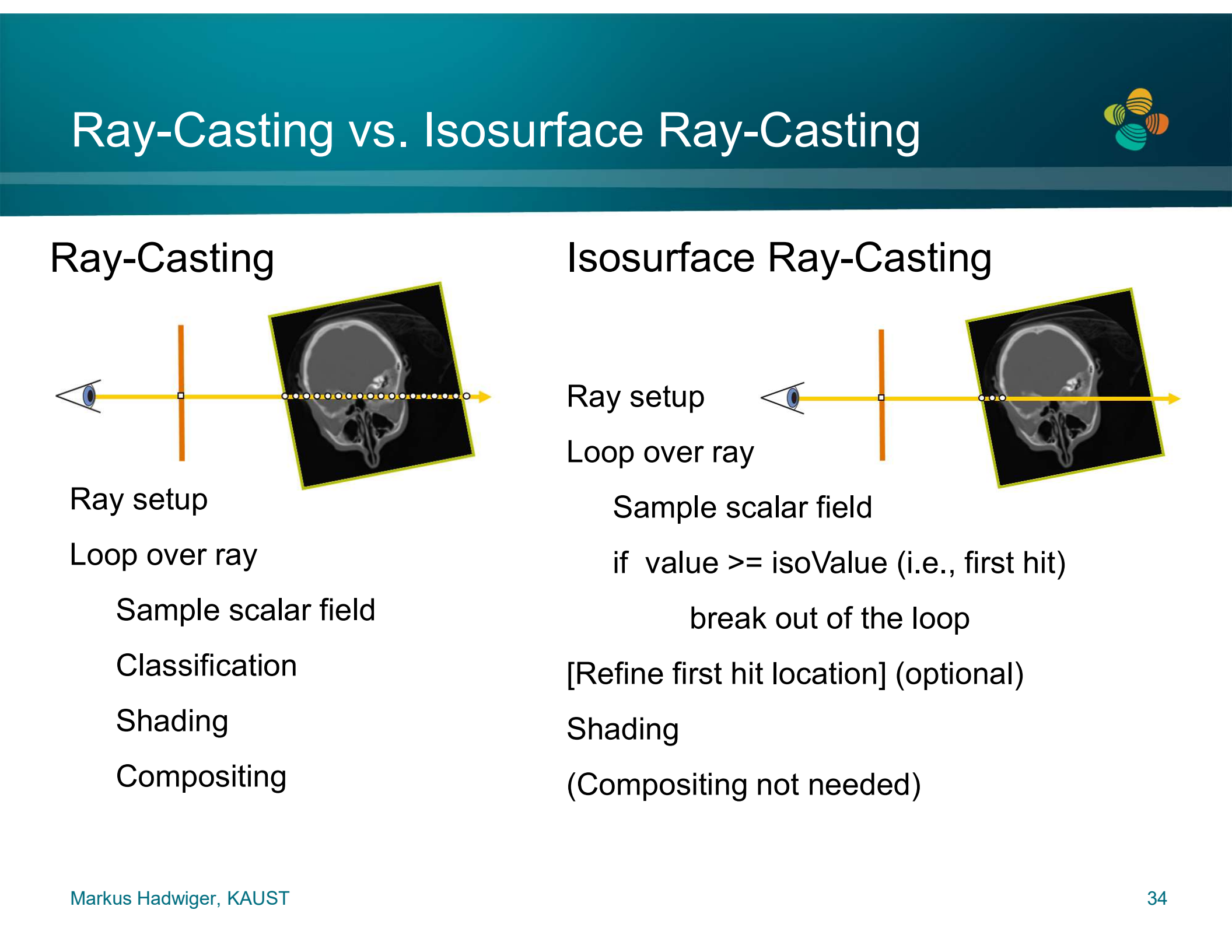
without refinement

with refinement



sampling distance 5 voxels (no adaptive sampling)

Ray-Casting vs. Isosurface Ray-Casting



Ray-Casting

Ray setup

Loop over ray

- Sample scalar field
- Classification
- Shading
- Compositing

Isosurface Ray-Casting

Ray setup

Loop over ray

- Sample scalar field
- if value >= isoValue (i.e., first hit)
  - break out of the loop

[Refine first hit location] (optional)

Shading

(Compositing not needed)

Thank you.

Thanks for material

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