

CS 247 – Scientific Visualization

Lecture 18: Volume Rendering, Pt. 5 [preview]

Markus Hadwiger, KAUST



Reading Assignment #10 (until Apr 4)

Read (required):

- Real-Time Volume Graphics, Chapter 7 (GPU-Based Ray Casting)
- Paper:

Markus Hadwiger, Ali K. Al-Awami, Johanna Beyer, Marco Agus, and Hanspeter Pfister

SparseLeap: Efficient Empty Space Skipping for Large-Scale Volume Rendering, IEEE Scientific Visualization 2017,

http://vccvisualization.org/publications/2017_hadwiger_sparseleap.pdf

http://vccvisualization.org/publications/2017_hadwiger_sparseleap.mp4

Read (optional):

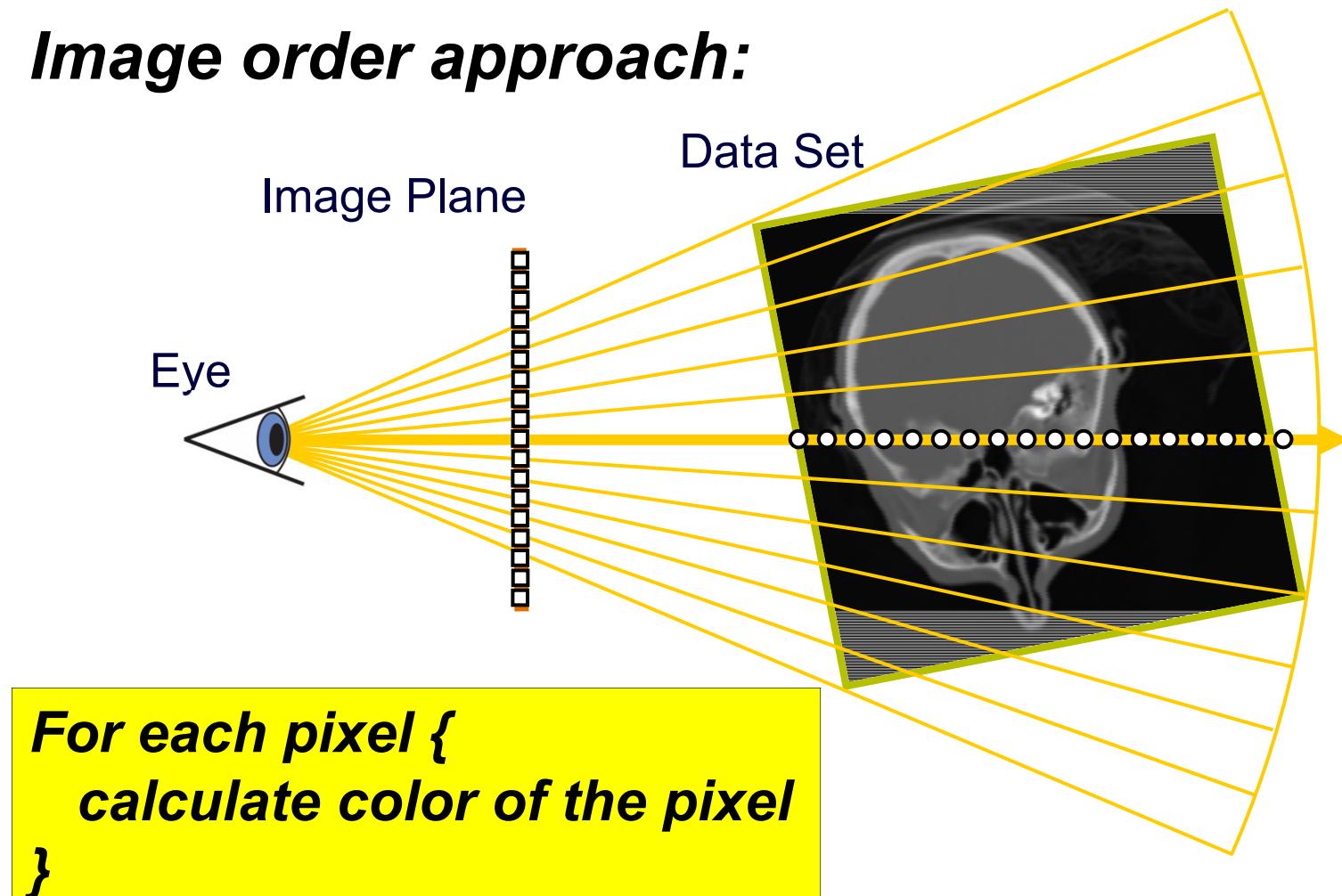
- Real-Time Volume Graphics, Chapter 6
(Global Volume Illumination)

VolVis: Image vs. Object Order

Direct Volume Rendering: Image Order



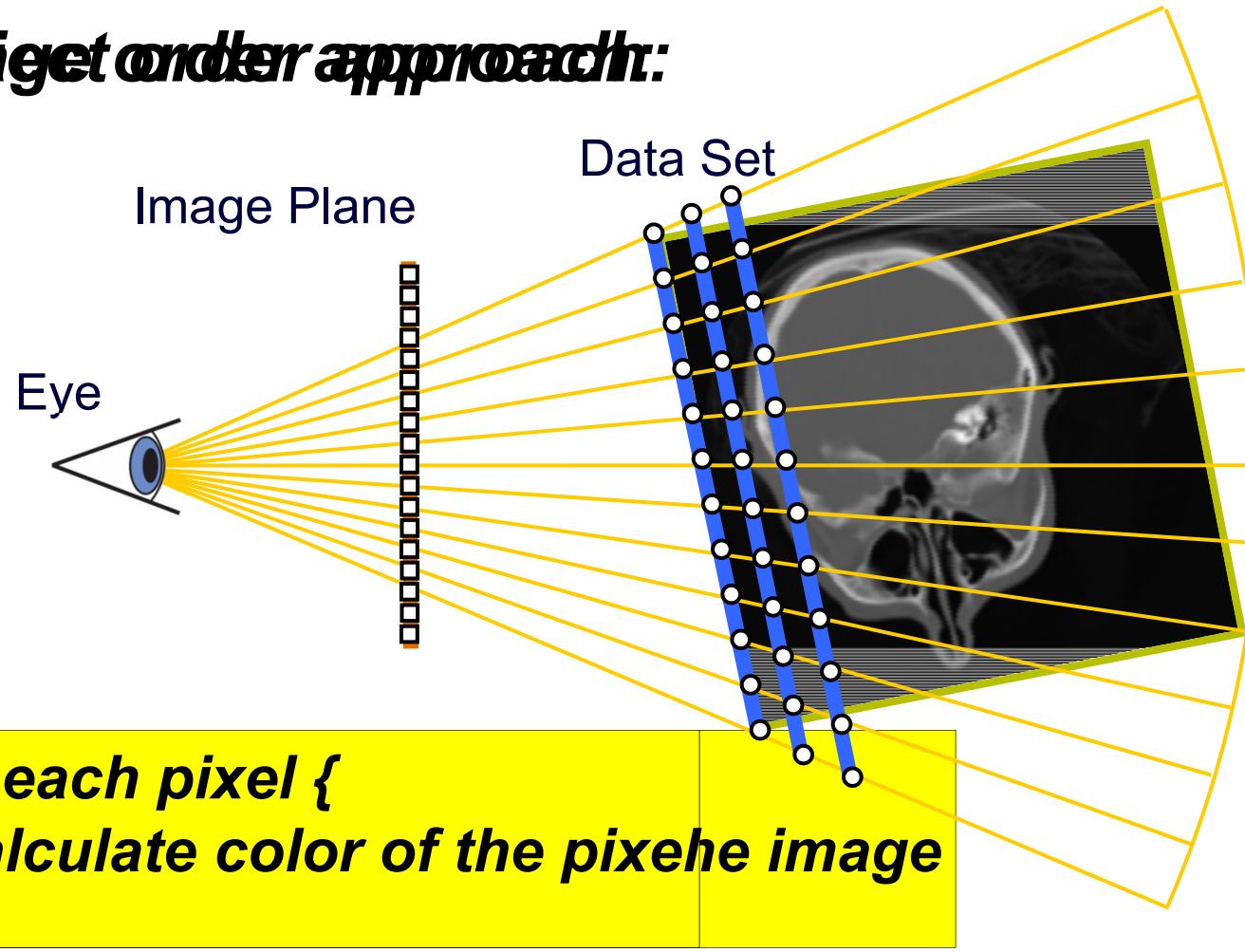
Image order approach:



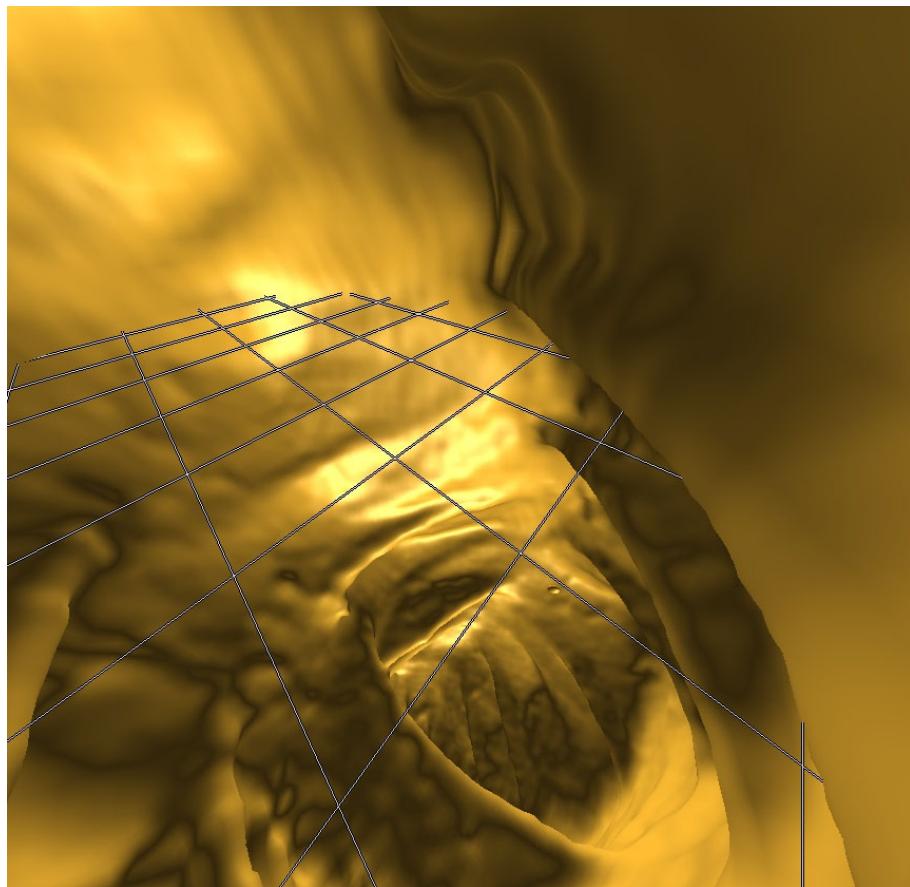
Direct Volume Rendering: Object Order



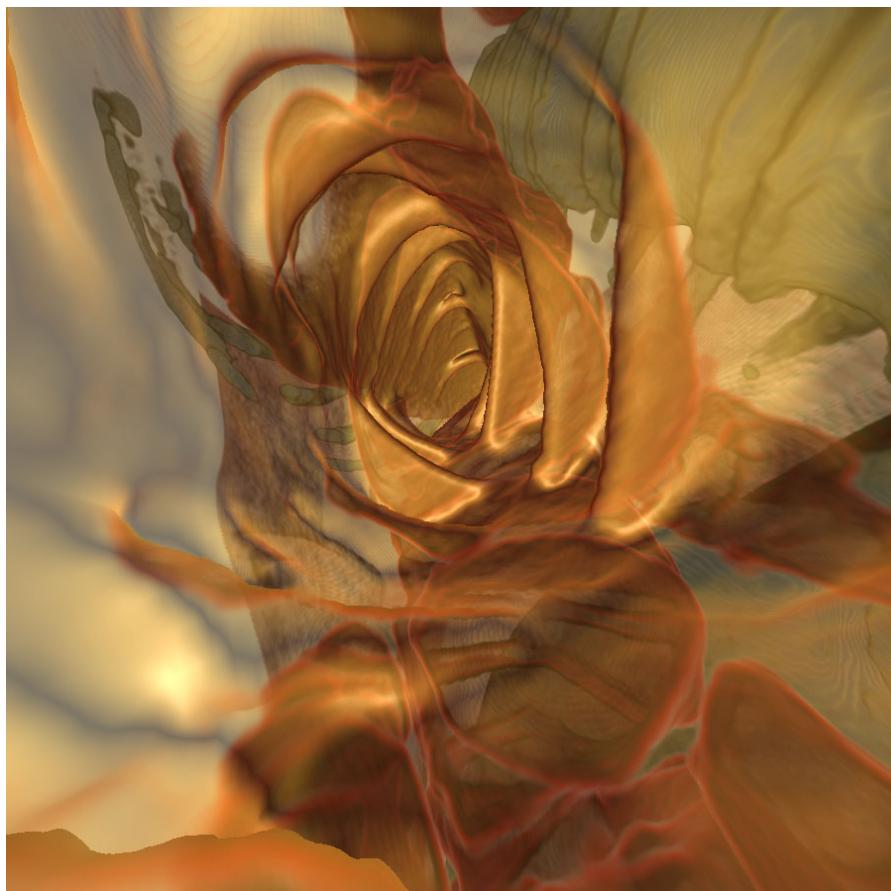
Object order approach:



Compositing



Compositing



Basic Volume Rendering Summary



Volume rendering integral
for *Emission Absorption* model

The diagram illustrates the volume rendering integral. A black sphere representing a volume element is at position s_0 . Two arrows originate from it: one pointing back along the ray path, labeled $I(s)$, and another pointing forward, labeled $q(\tilde{s}) e^{-\tau(\tilde{s}, s)}$.

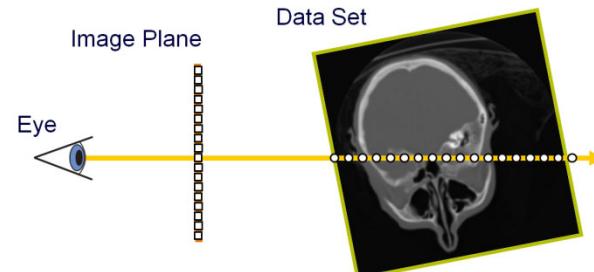
$$I(s) = I(s_0) e^{-\tau(s_0, s)} + \int_{s_0}^s q(\tilde{s}) e^{-\tau(\tilde{s}, s)} d\tilde{s}$$

Numerical solutions: ***back-to-front*** vs. ***front-to-back compositing***

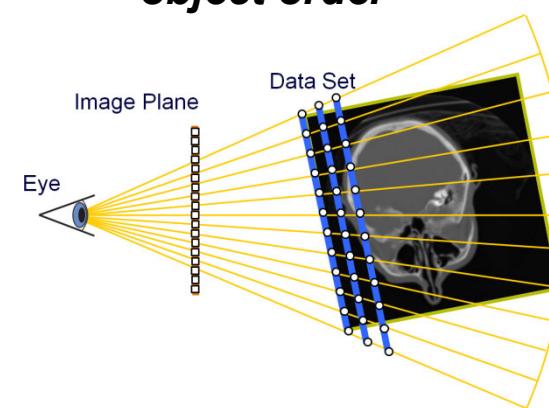
$$C'_i = C_i + (1 - A_i)C'_{i-1}$$

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

Approaches: ***image order***



vs. ***object order***



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_idx = ( __umul24( screencoord_y, cu_screensize.x ) + screencoord_x ) * 4;

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

    // 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_idx ] = __saturatef( color );
    }
}
```

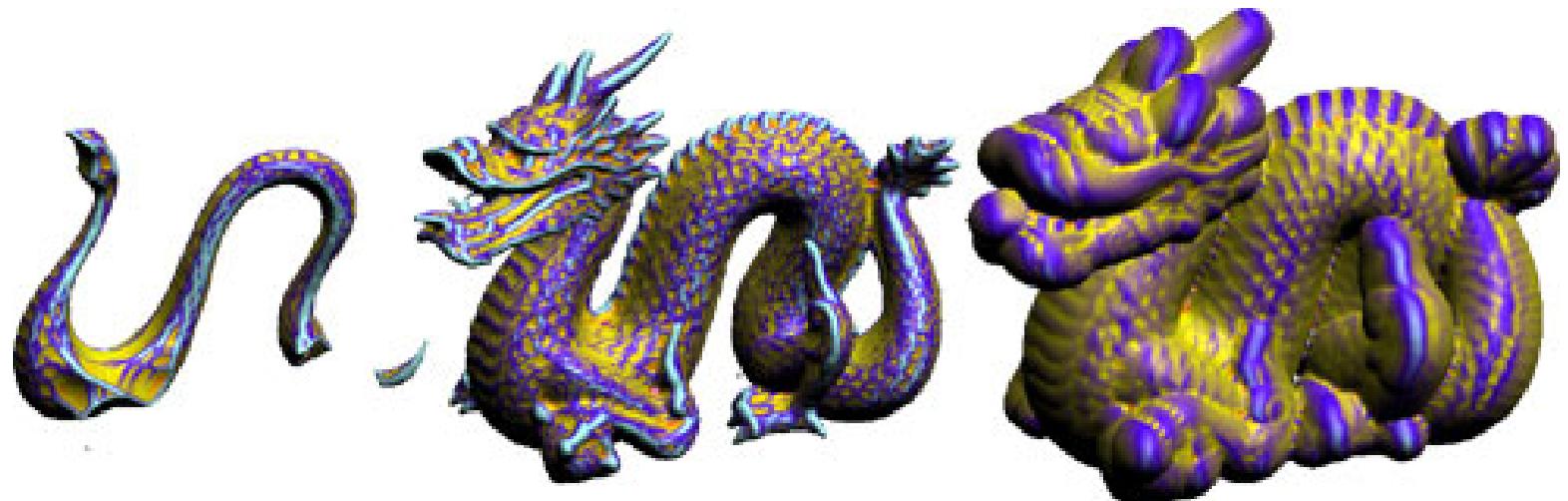
Isosurface Ray-Casting

Isosurface Ray-Casting



Isosurfaces/Level Sets

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

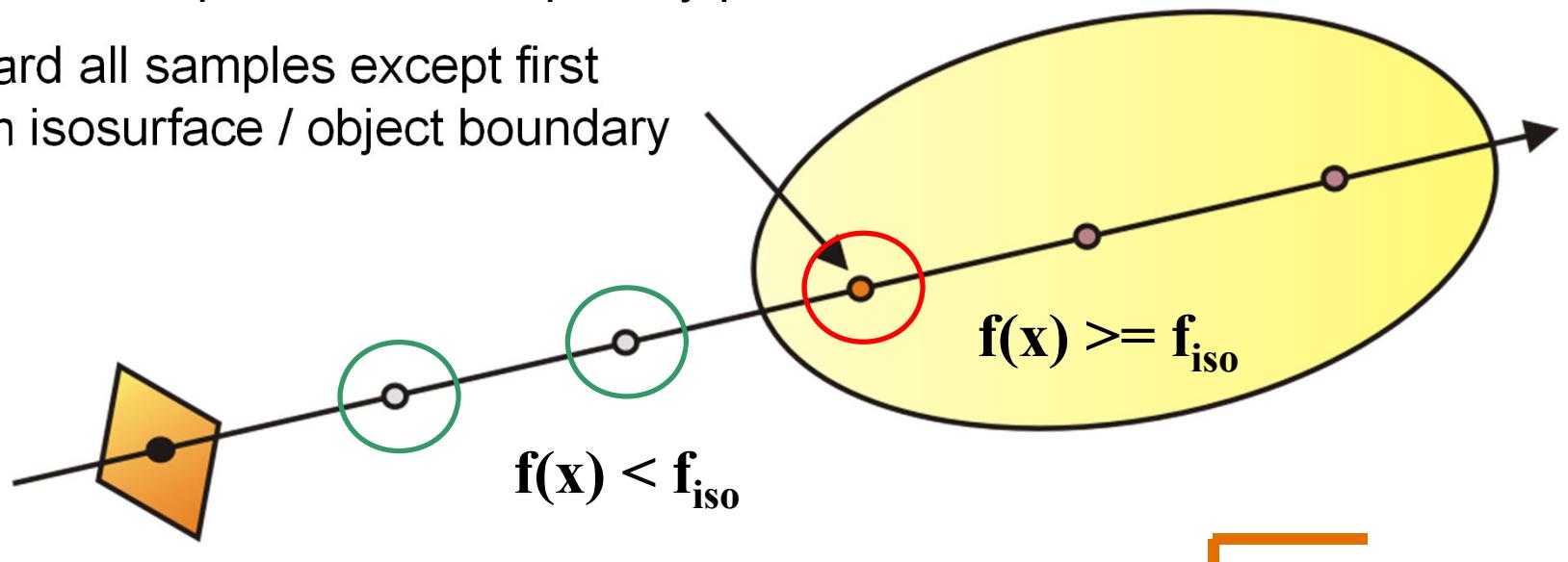


Isosurface Ray-Casting



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)

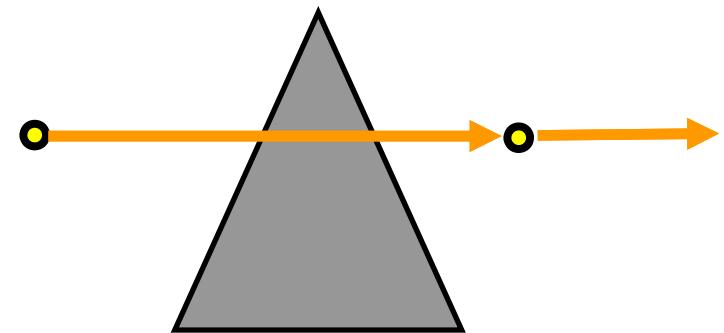
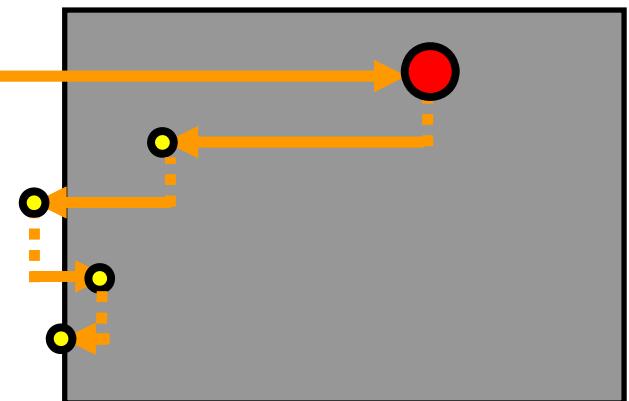


Fixed number of bisection / binary search steps

Virtually no impact on performance

Refine already detected
intersection

Handle problems with small
features / at silhouettes with
adaptive sampling



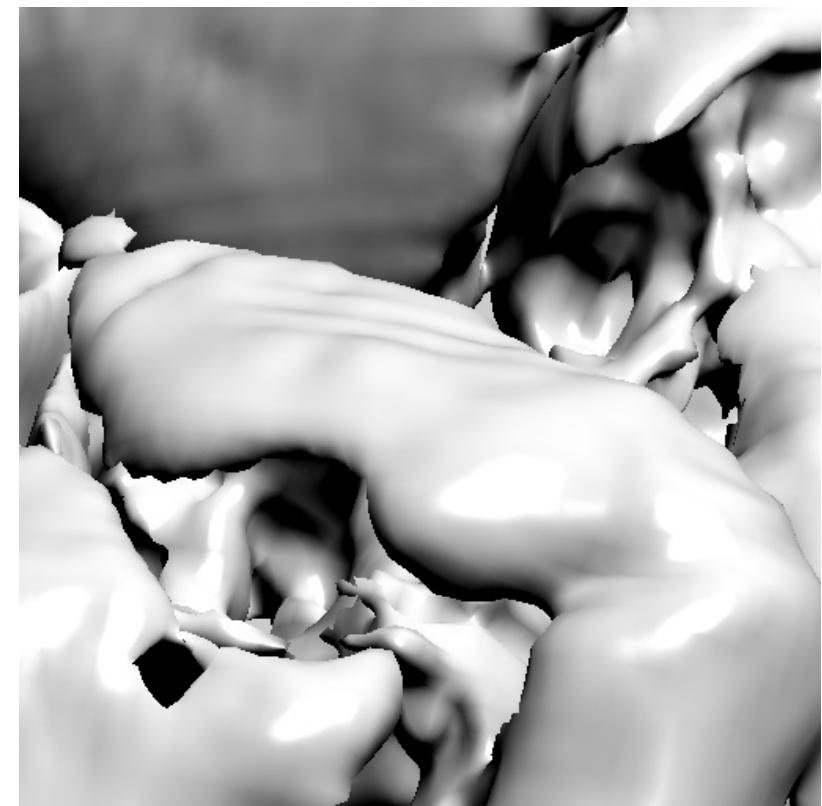
Intersection Refinement (2)



without refinement



with refinement

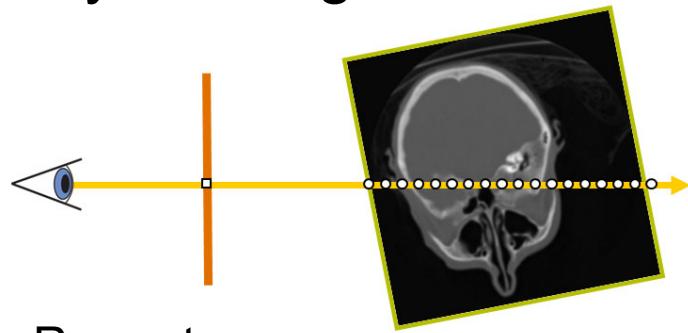


sampling distance 5 voxels (no adaptive sampling)

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 \geq isoValue (i.e., first hit)

break out of the loop

[Refine first hit location] (optional)

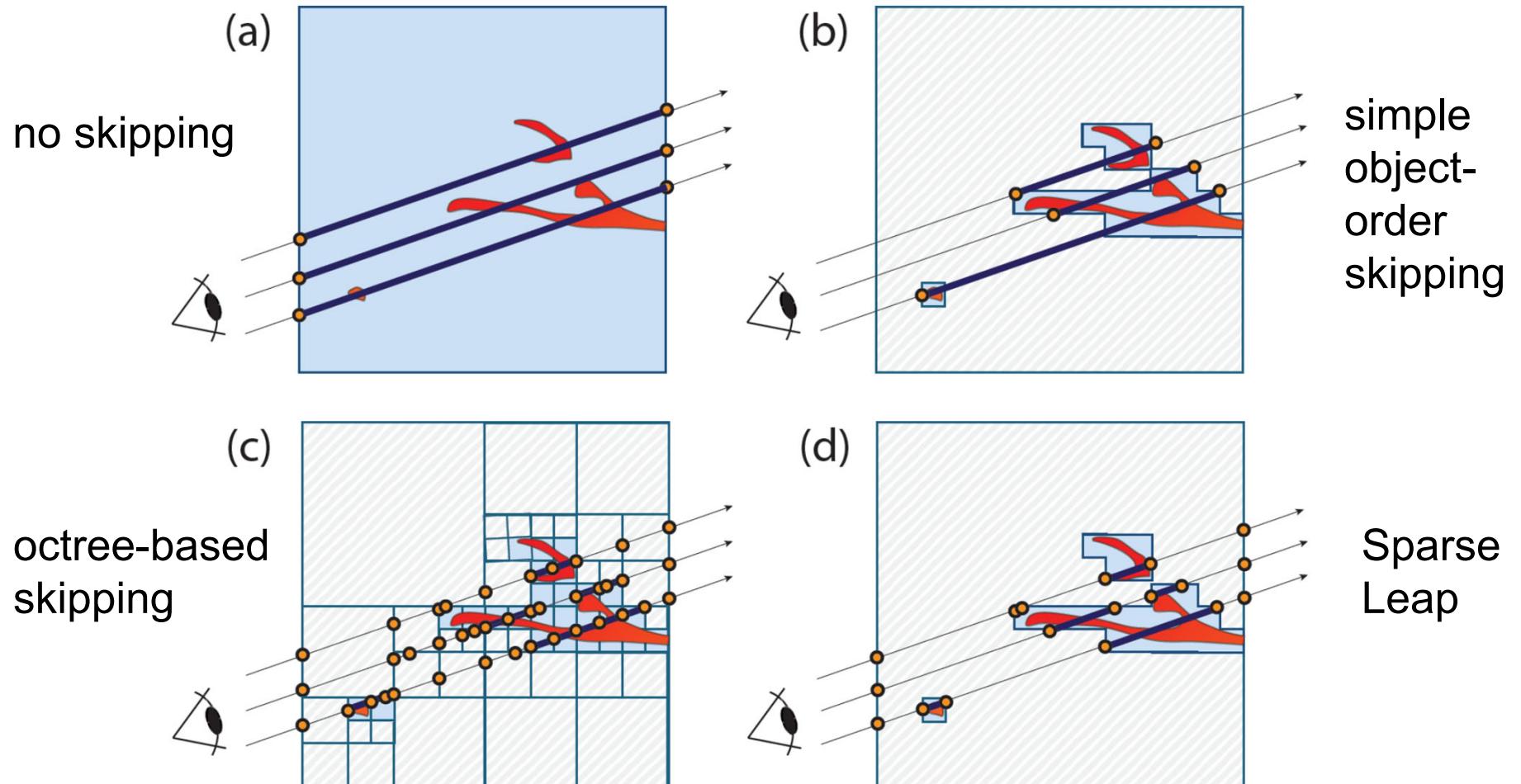
Shading

(Compositing not needed)

Empty Space Skipping



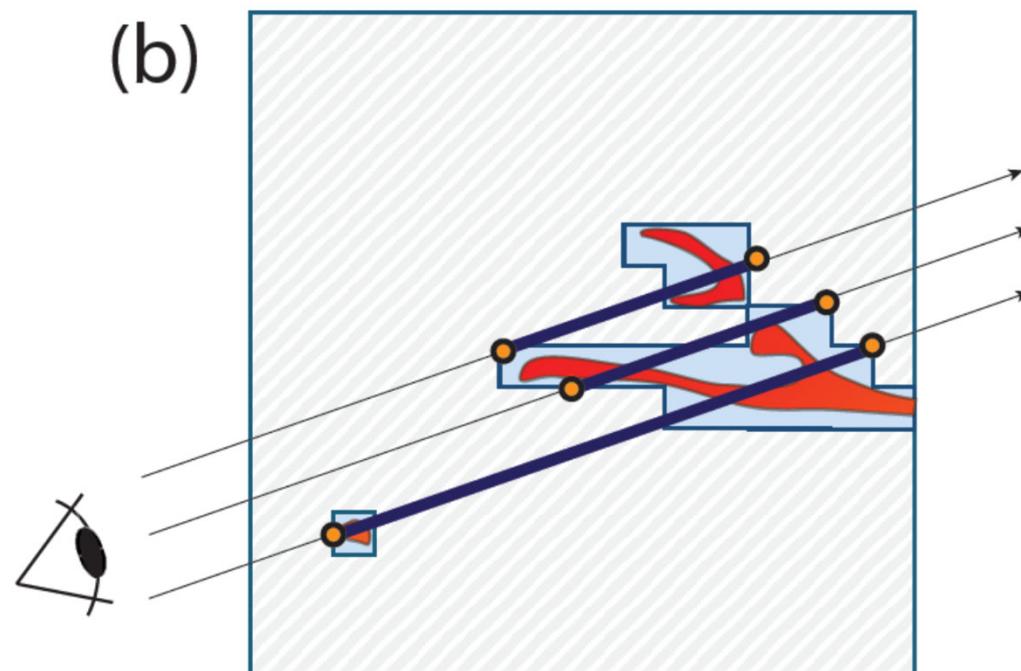
Different Approaches





Object-Order Empty Space Skipping

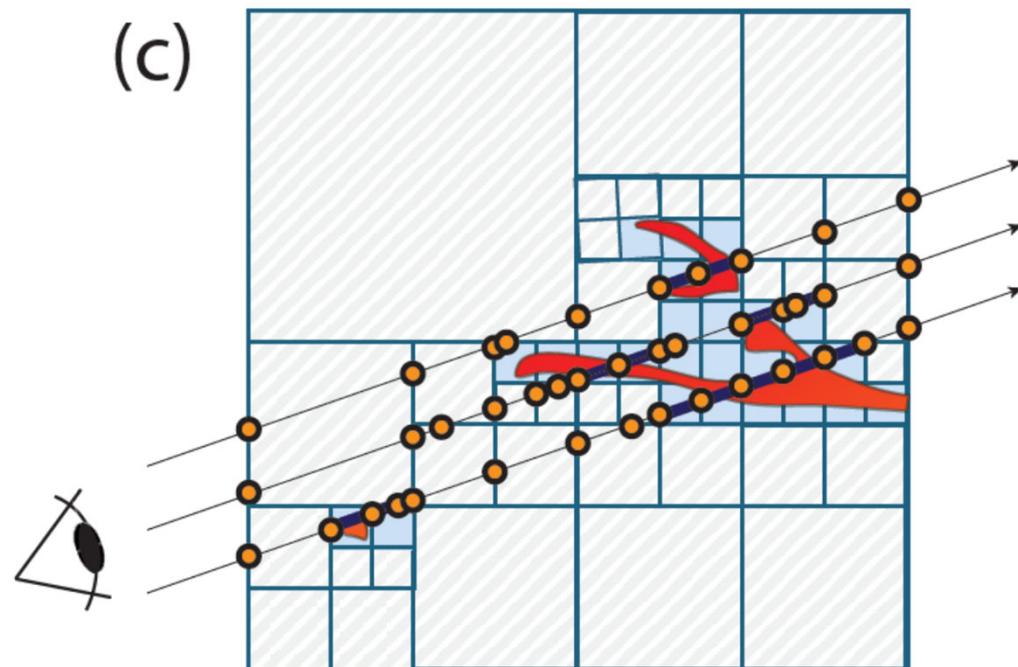
Modify initial rasterization step for ray setup





Octree-Based Empty Space Skipping

Everything is done during tree traversal along the ray



More on Transfer Functions

Classification – Transfer Functions



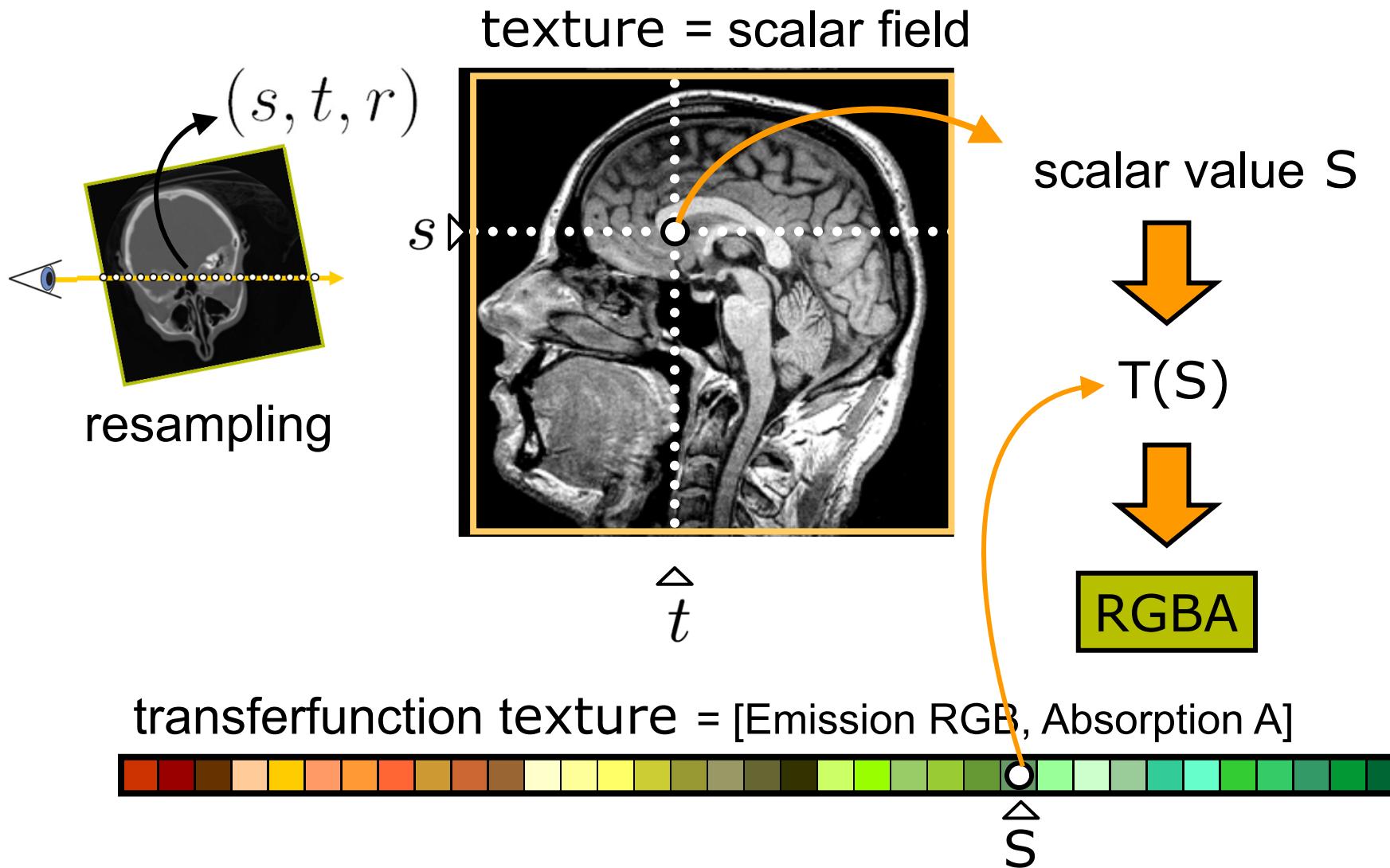
During Classification the user defines the “*look*“ of the data.

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

The user defines a *transfer function*.



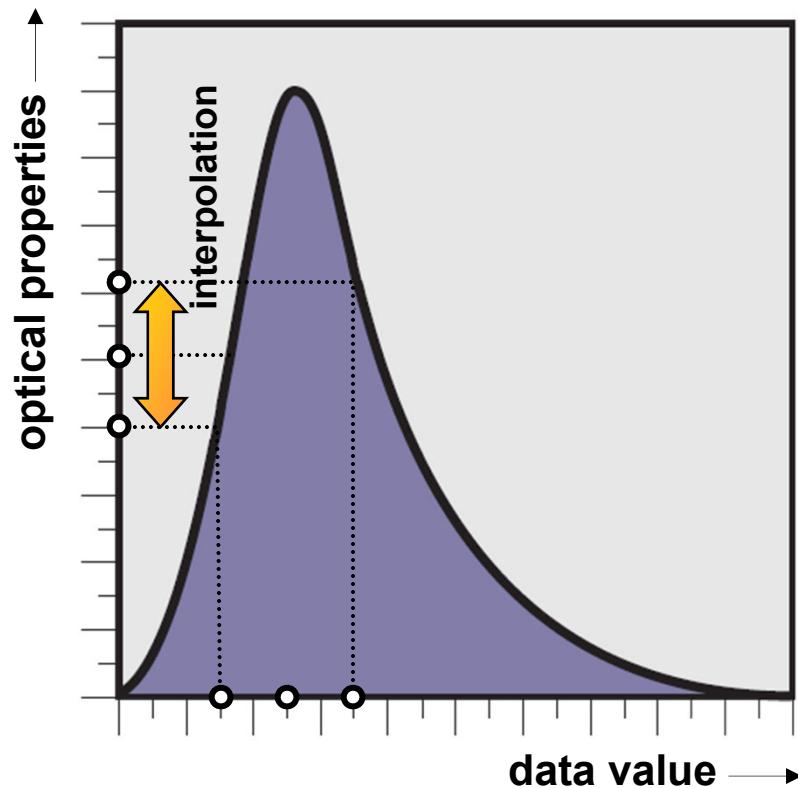
1D Transfer Functions



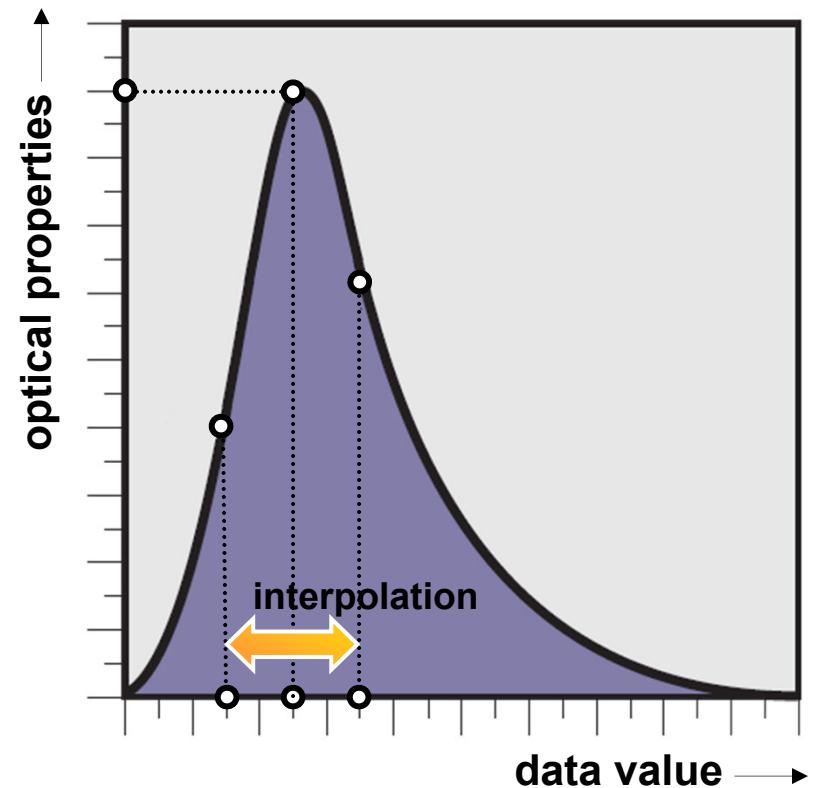
Pre- vs Post-Interpolative Classification



PRE-INTERPOLATIVE



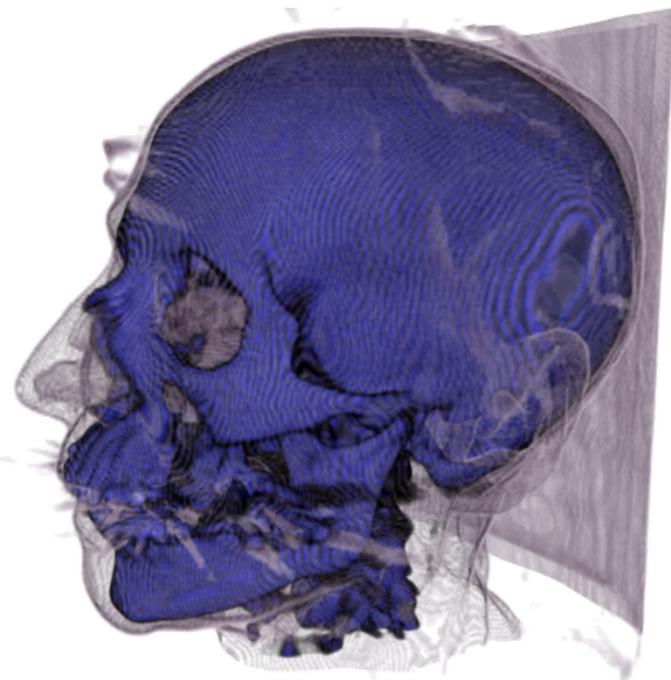
POST-INTERPOLATIVE



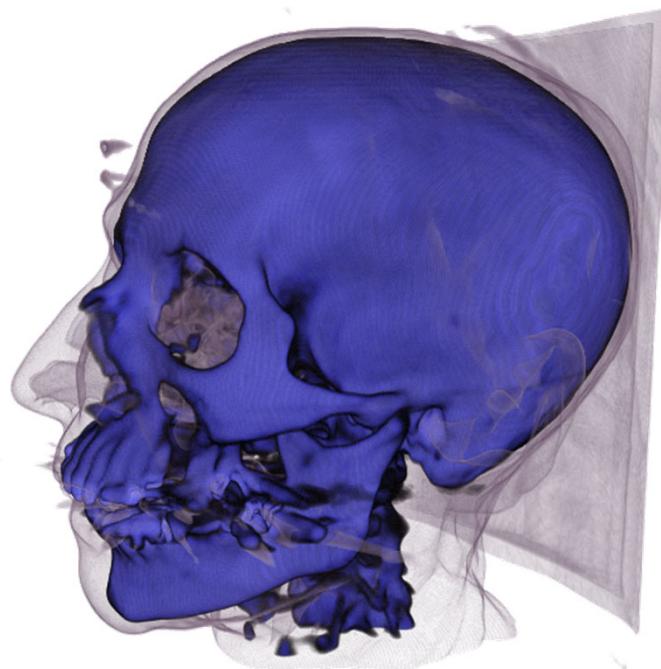
Quality: Pre- vs. Post-Classification



Comparison of image quality



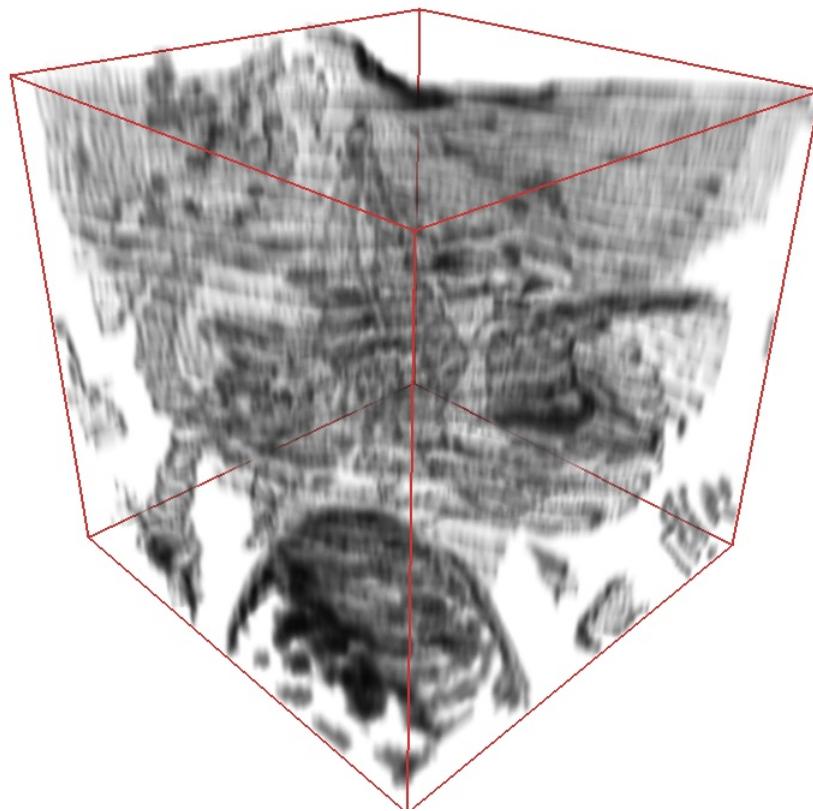
Pre-Classification



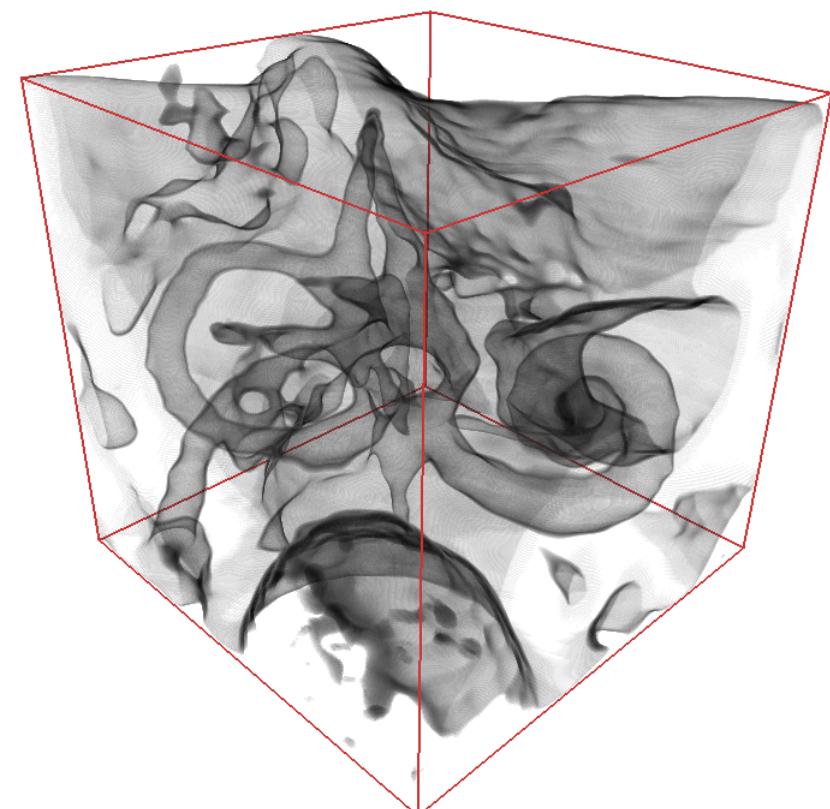
Post-Classification

same TF, same resolution, same sampling rate

Quality: Pre- vs. Post-Classification



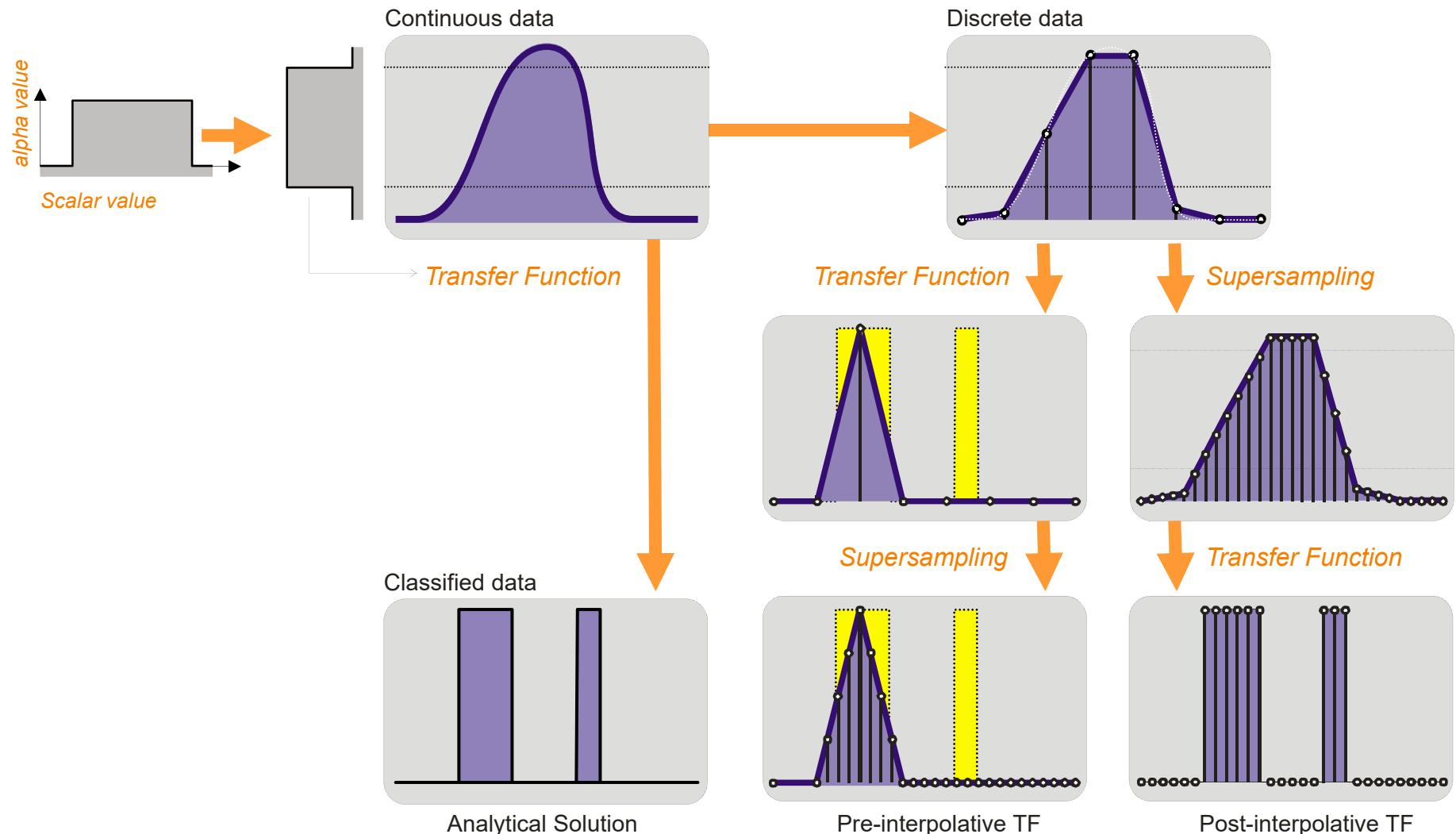
Pre-Classification



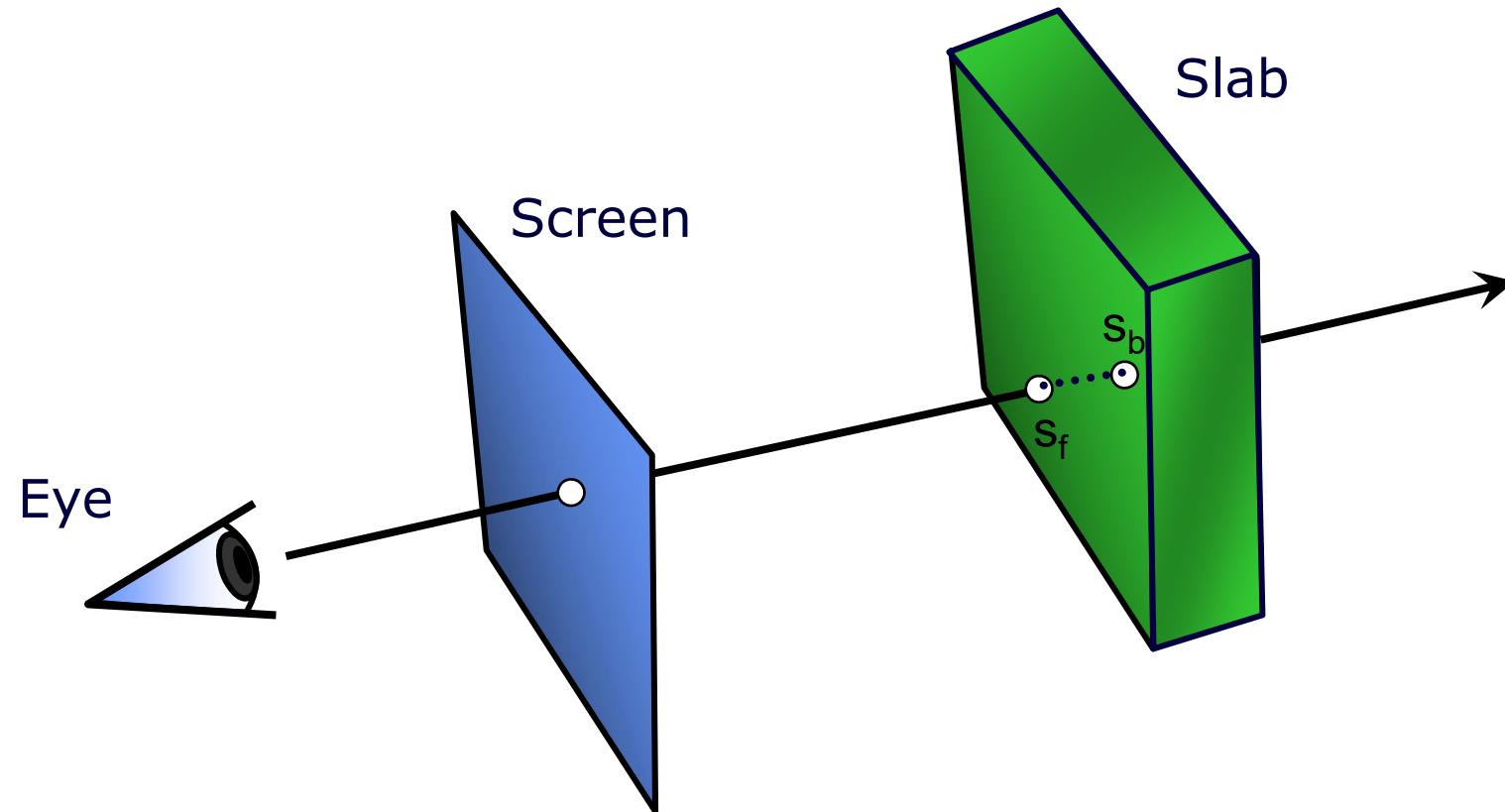
Post-Classification



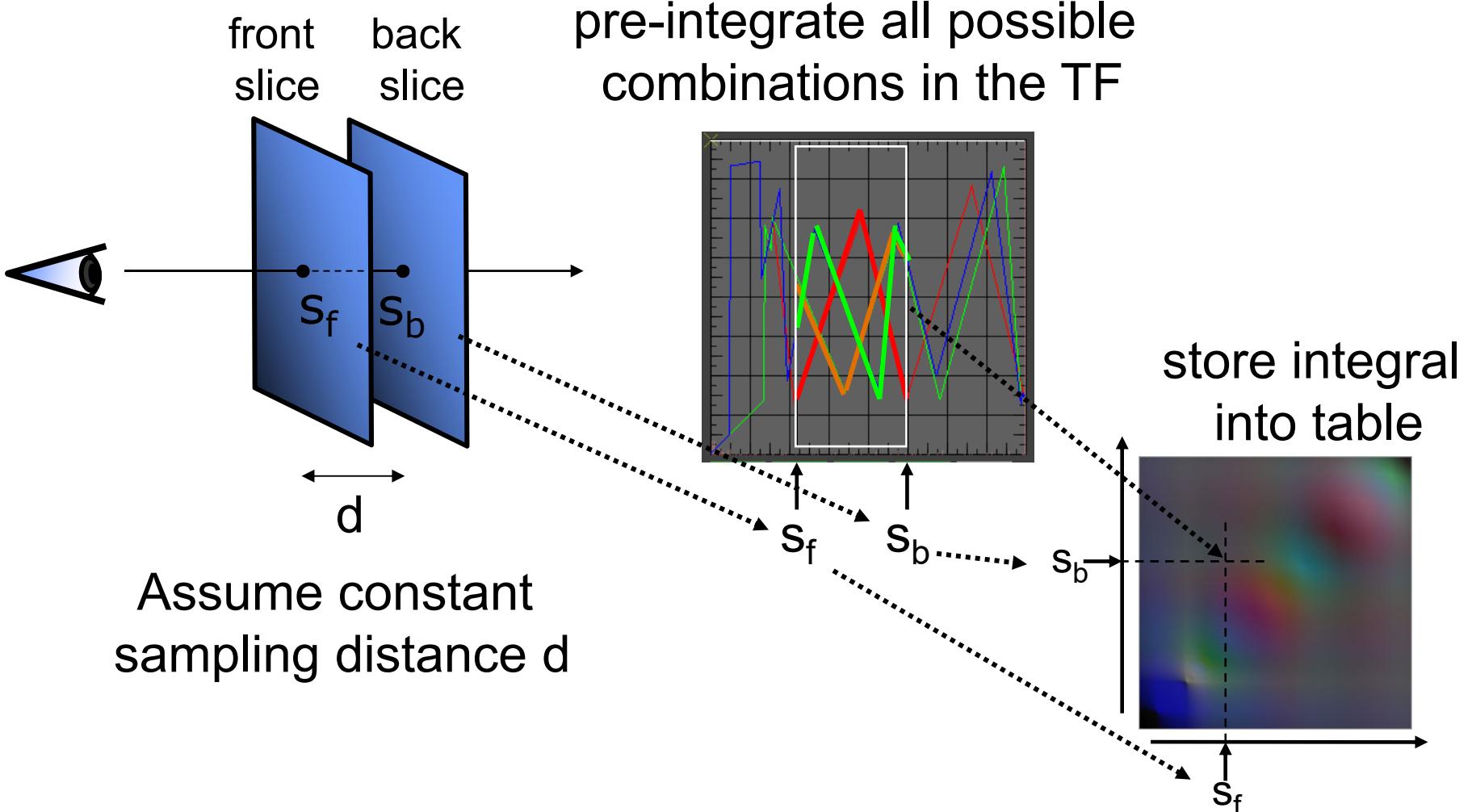
Pre- vs Post-Classification



Pre-Integrated Classification



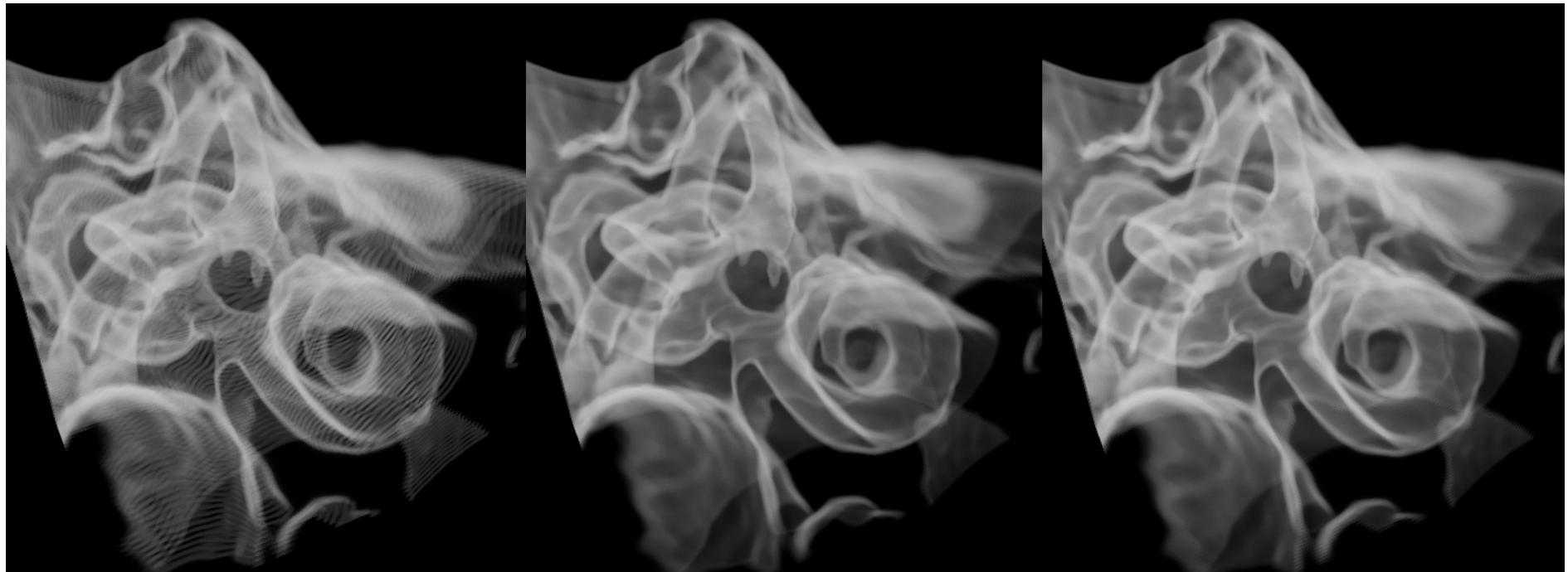
Pre-Integrated Classification





Pre-Integrated Classification

Quality comparison



128 Slices

284 Slices

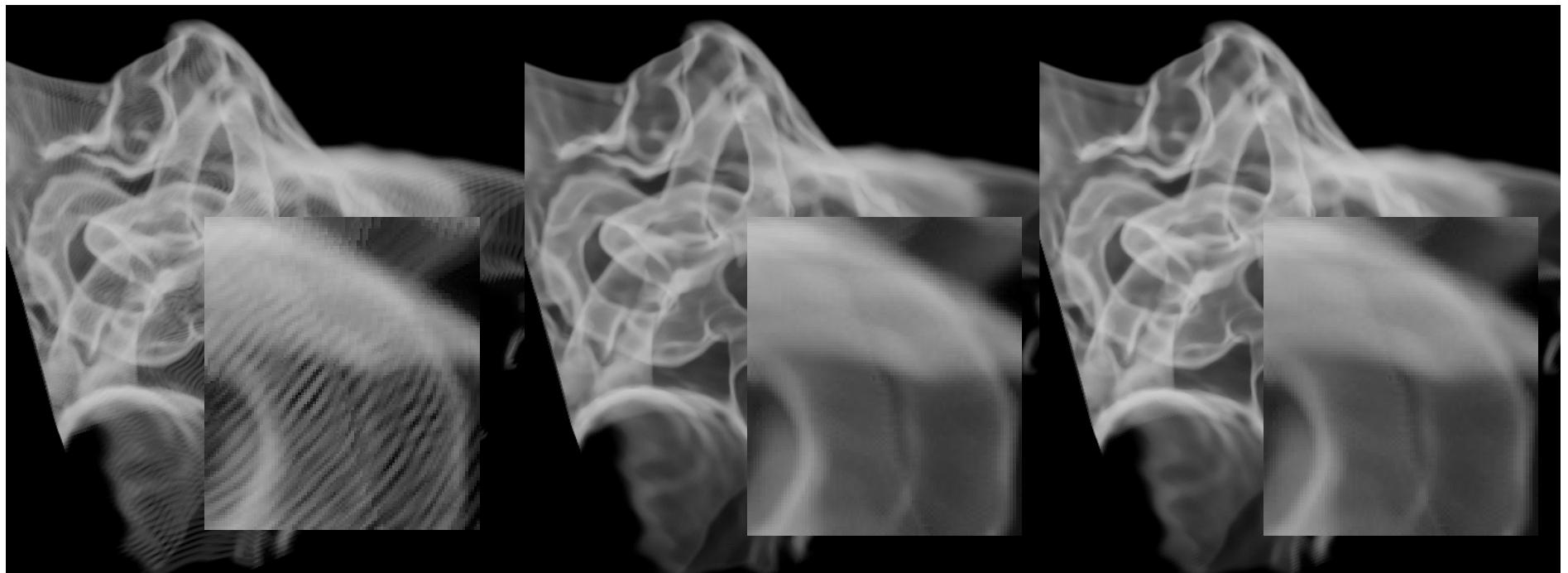
128 Slabs

© Weiskopf/Machiraju/Möller



Pre-Integrated Classification

Quality comparison



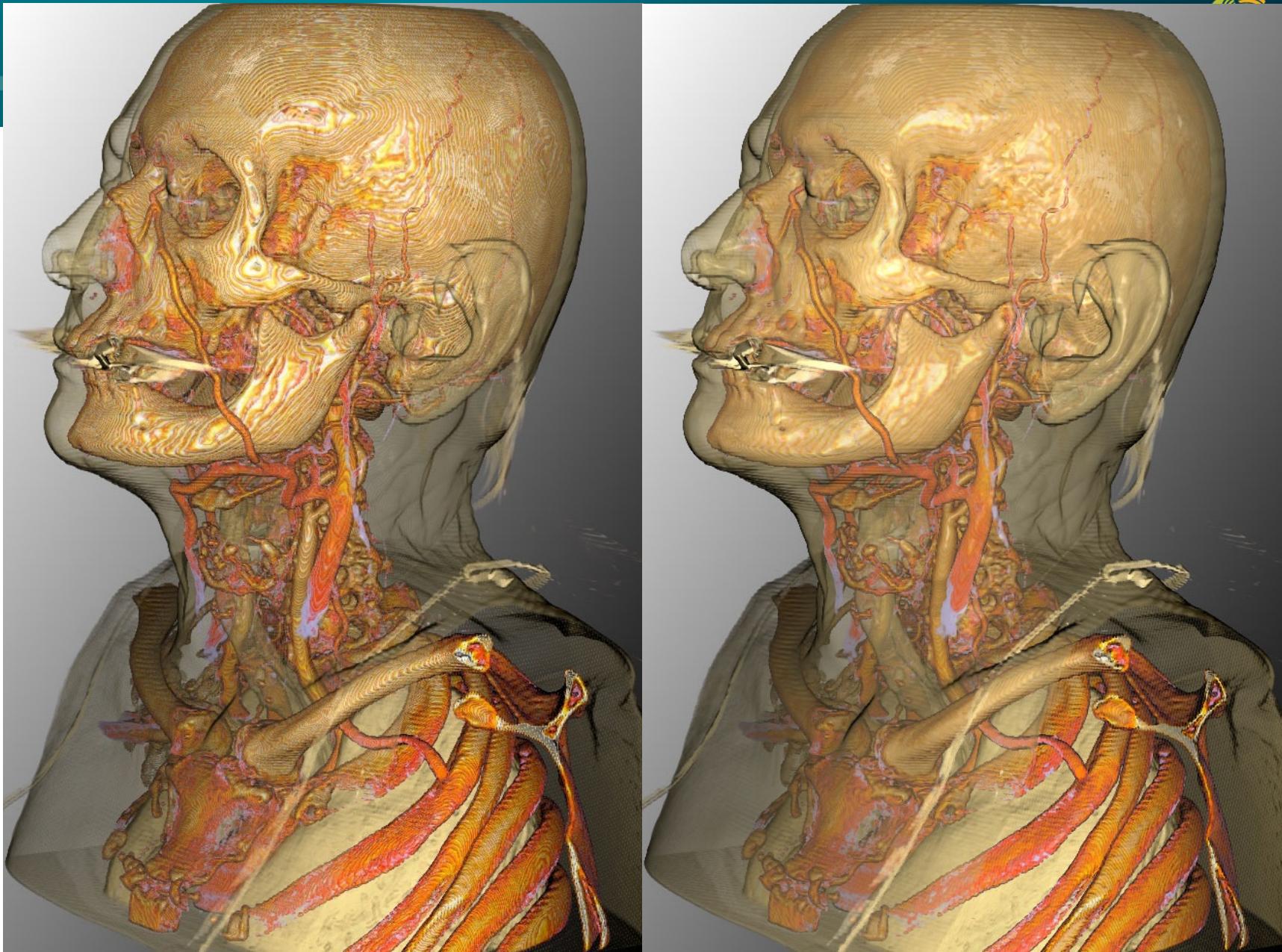
128 Slices

284 Slices

128 Slabs

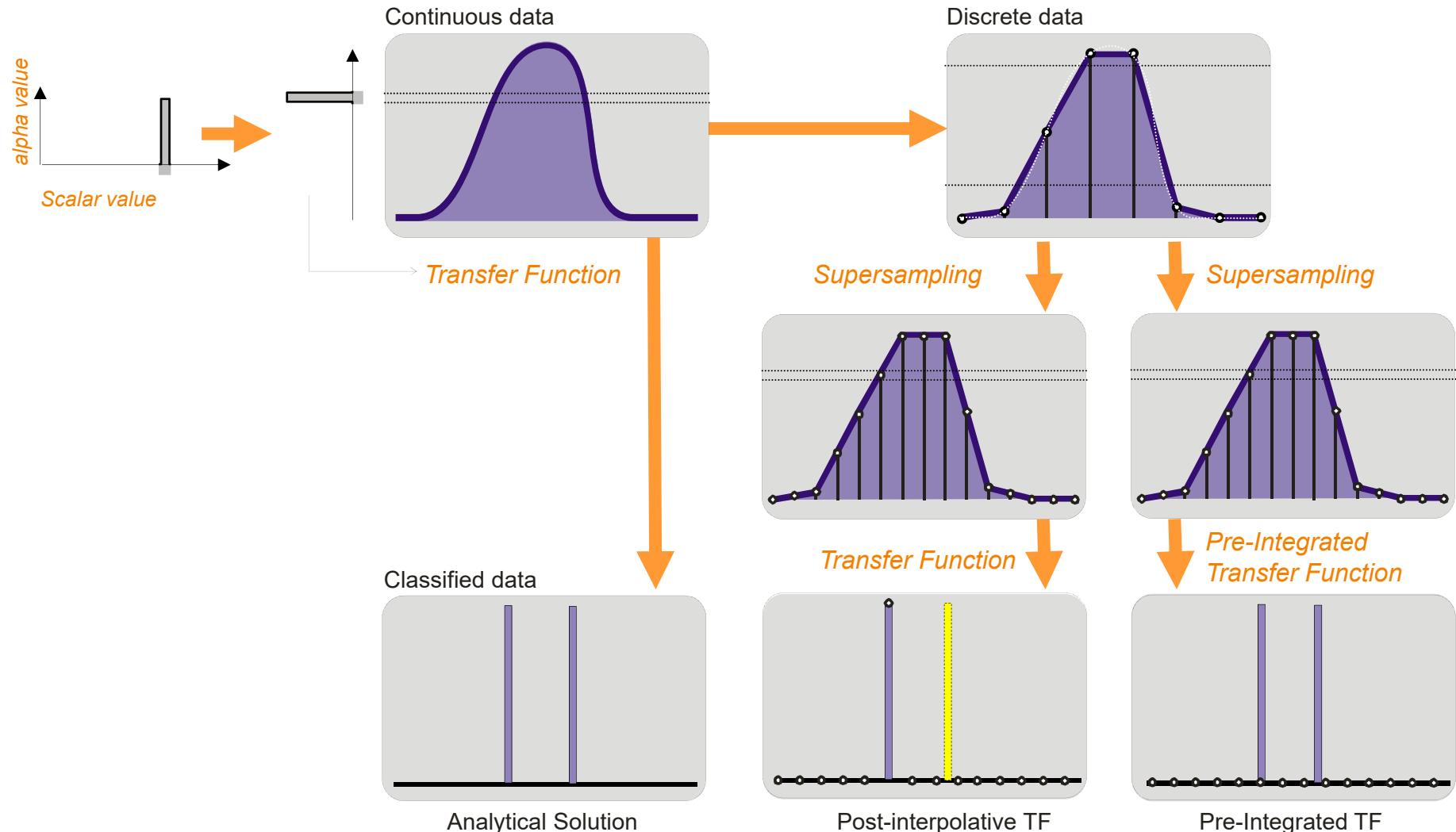
© Weiskopf/Machiraju/Möller

Pre-Integrated Classification





Post- vs. Pre-Integrated Classification



Thank you.

Thanks for material

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