Real-Time Visualization

02: Real-Time Volume Graphics 2

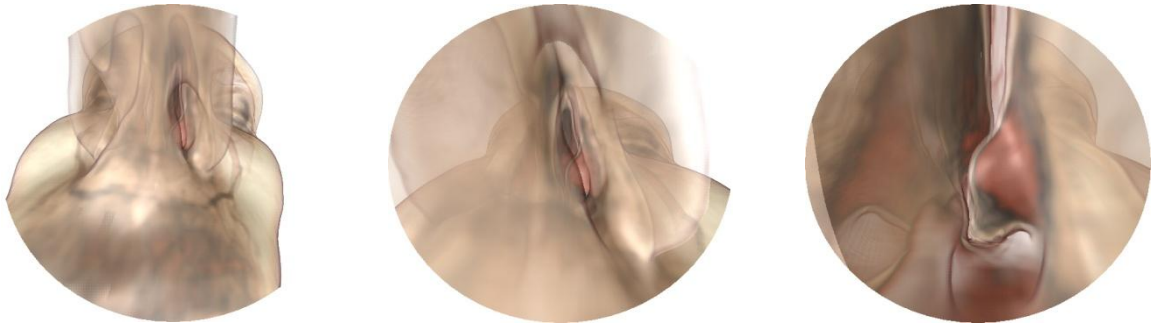
Stefan Bruckner

Lectures

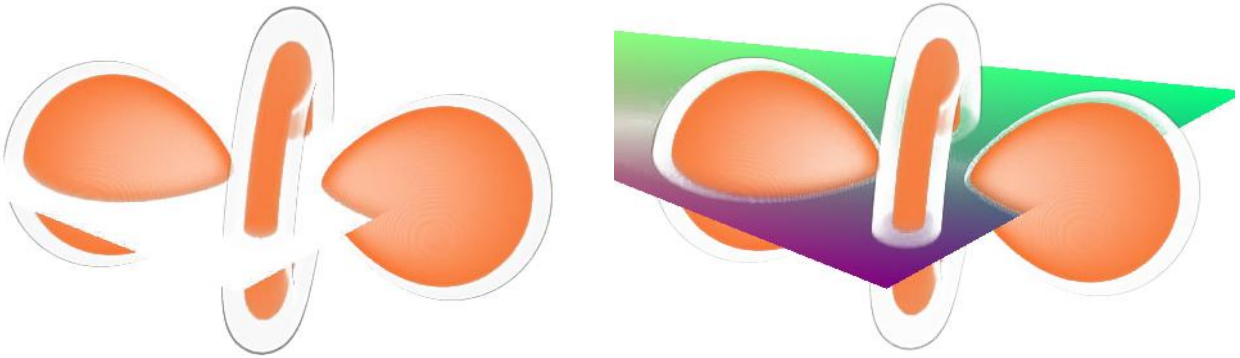
1. **Introduction:** Volume rendering basics, GPU architecture, OpenGL, CUDA, case studies, … (03.05.)
2. **Real-Time Volume Graphics 1:** GPU ray-casting, optimizations, memory management, OpenGL vs. CUDA, … (10.05.)
3. **Real-Time Volume Graphics 2:** Advanced illumination, filtering, derivatives, advanced transfer functions, … (17.05.)
4. **Real-Time Computations on GPUs:** Fluid simulation, level-set deformation, … (24.05.)
5. **Volumetric Special Effects:** combining simulation and ray-casting, integration in game engine scenes, ... (31.05.)
6. **Final event:** Project presentations (28.06.)

Enhancements (1)

Allow viewpoint inside the volume



Intersect polygonal geometry



Enhancements (2)

Starting position computation

- Ray start position image

Ray length computation

- Ray length image

Render polygonal geometry

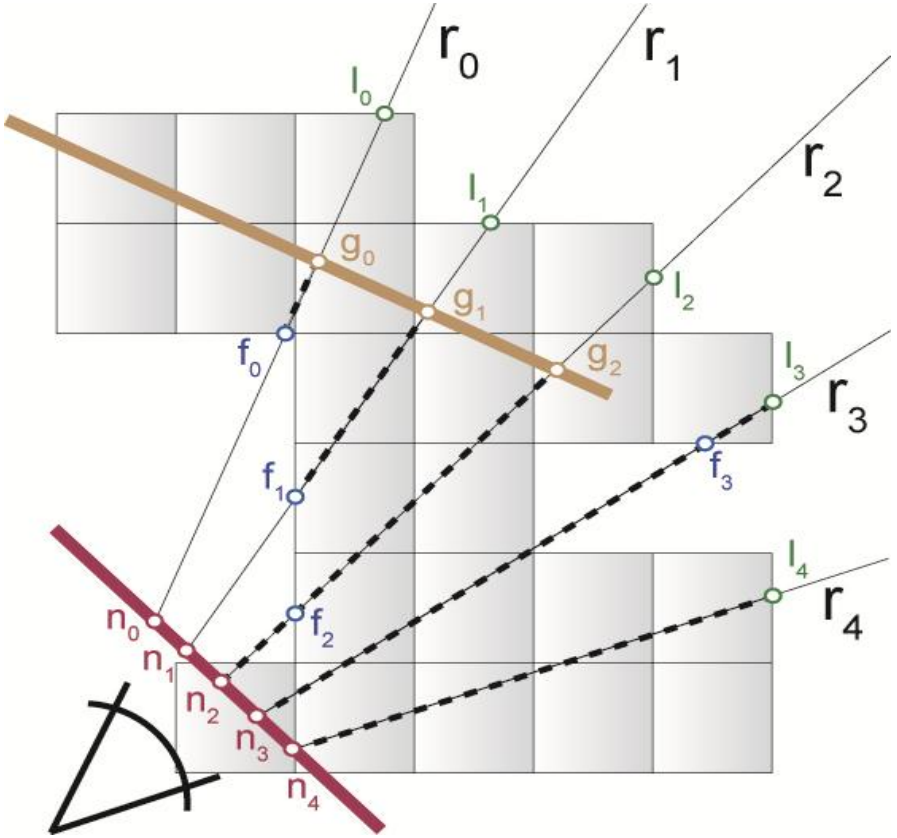
- Modified ray length image

Raycasting

- Compositing buffer

Blending

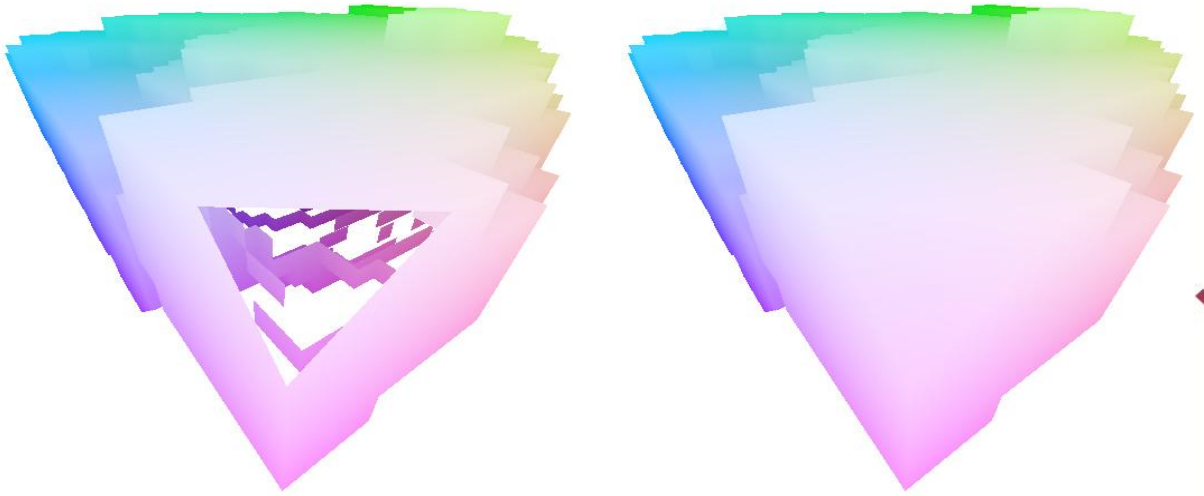
- Final image

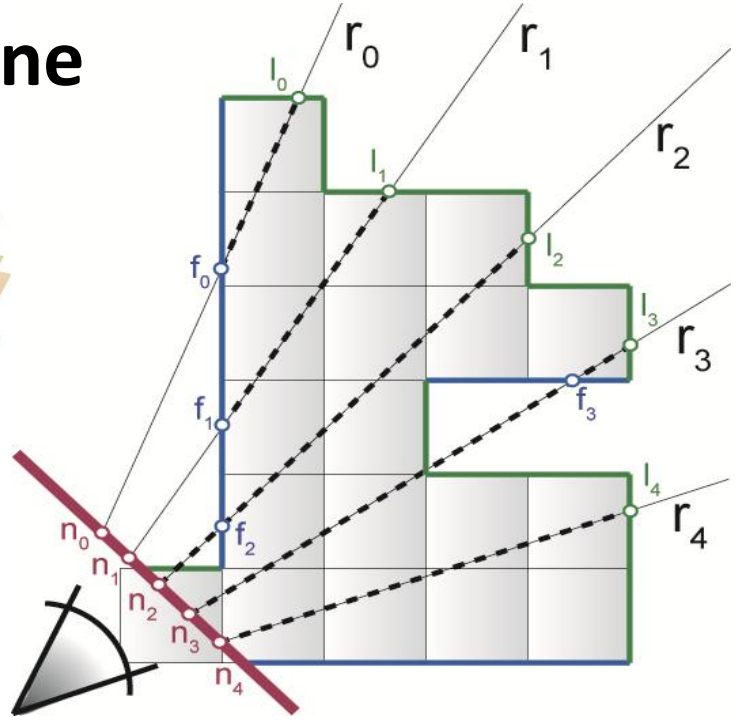


Moving Into The Volume (1)

Near clipping plane clips into front faces

Fill in holes with near clipping plane

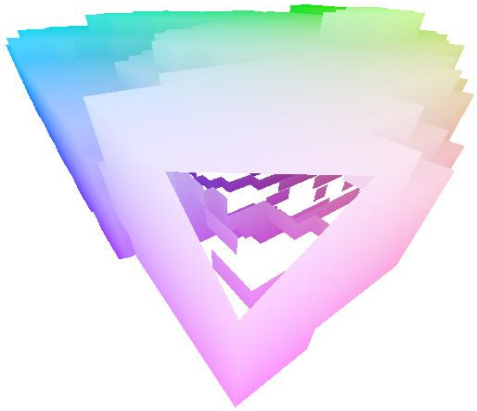




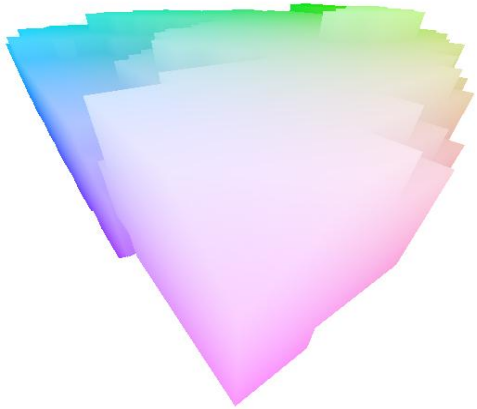
- Can use depth buffer [Scharsach et al., 2006]

Moving Into The Volume (2)

1. **Rasterize near clipping plane**
   - Disable depth buffer, enable color buffer
   - Rasterize entire near clipping plane



2. **Rasterize nearest back faces**
   - Enable depth buffer, disable color buffer
   - Rasterize *nearest back faces* of active bricks



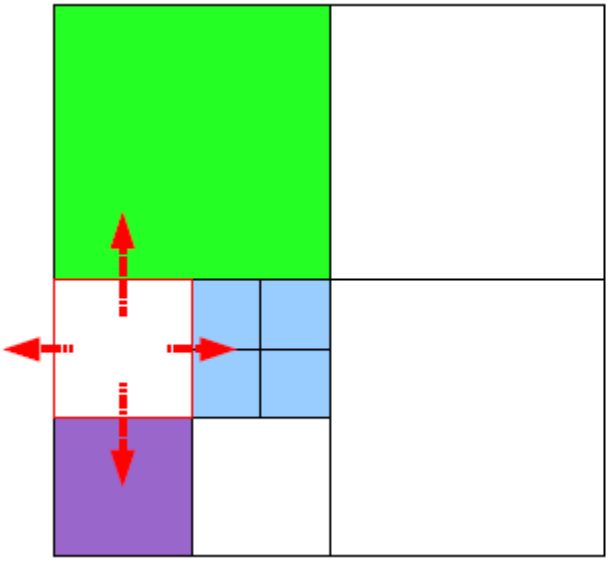
3. **Rasterize nearest front faces**
   - Enable depth buffer, enable color buffer
   - Rasterize *nearest front faces* of active bricks

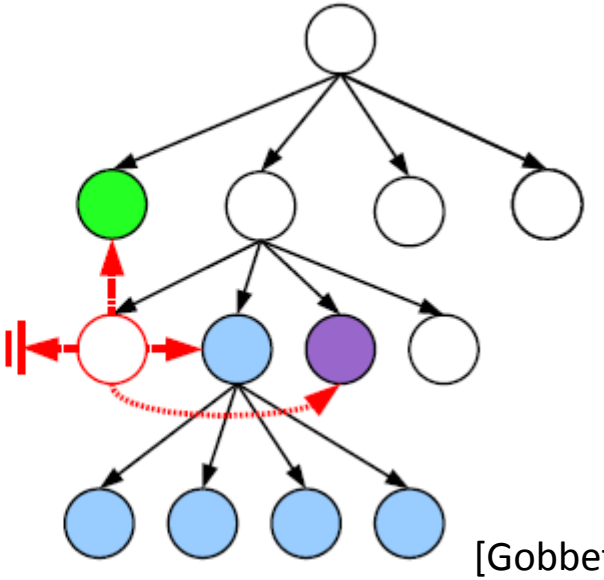
Single-Pass Octree Ray-Casting (1)

Determine active octree front/cut

Use stack-less kD tree / octree traversal

Download only active octree nodes to GPU





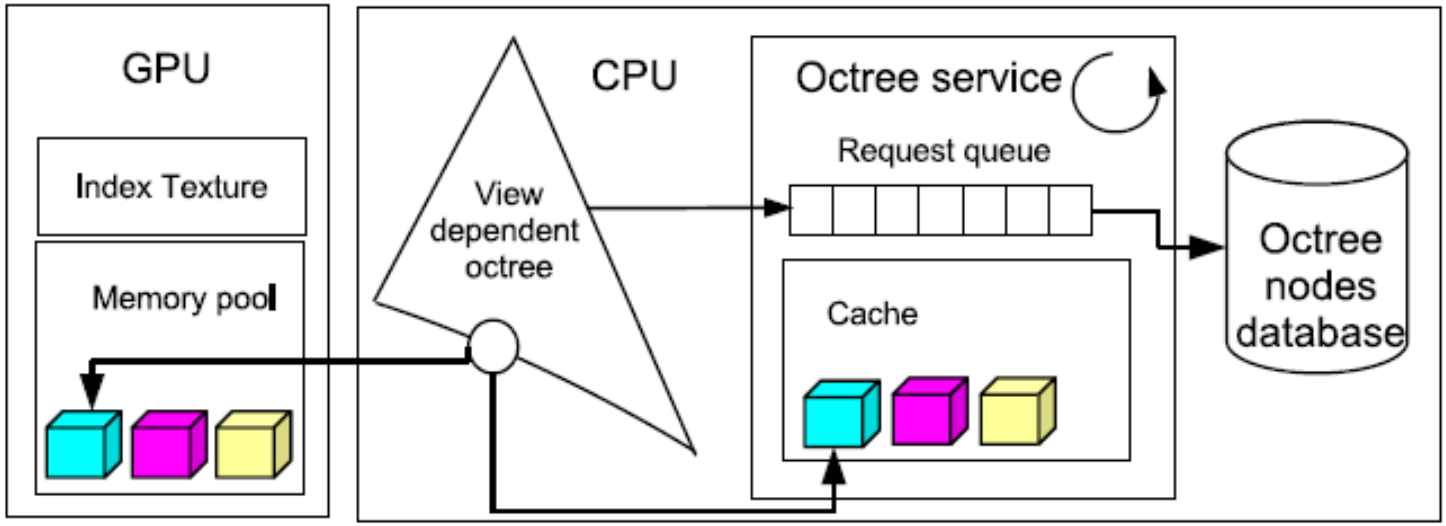
[Gobbetti et al., 2008]

Single-Pass Octree Ray-Casting (2)

Out-of-core rendering

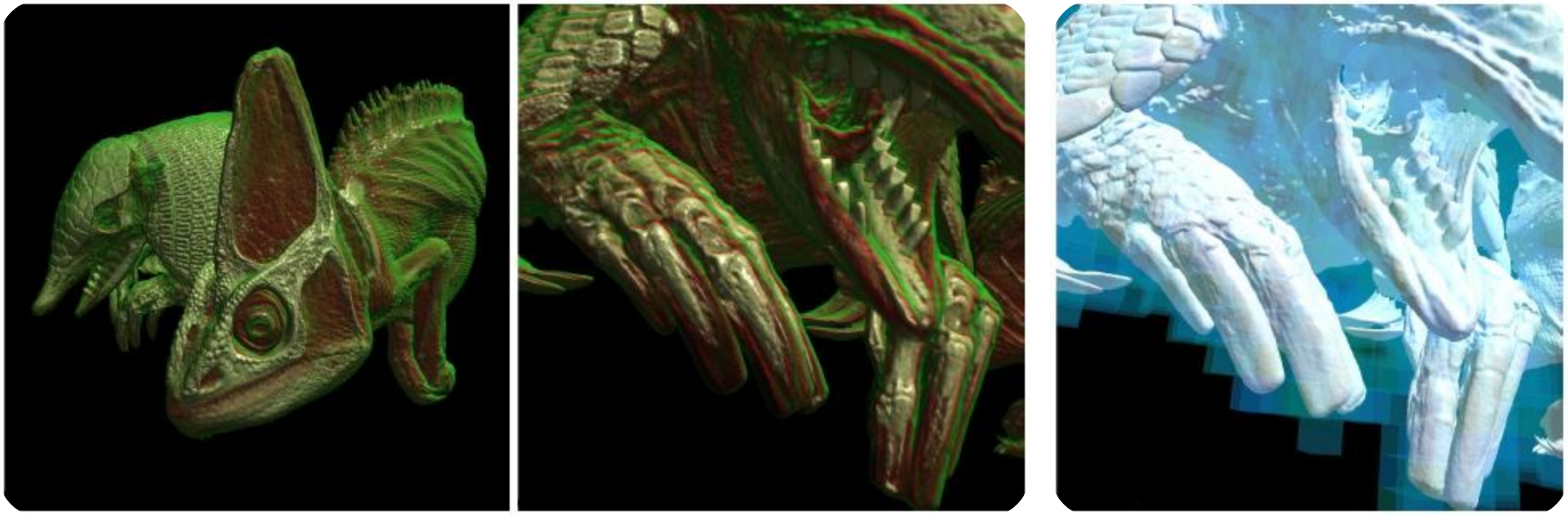
Visibility culling using occlusion queries

Encode active octree cut in index texture



[Gobbetti et al., 2008]

Single-Pass Octree Ray-Casting (3)



[Gobbetti et al., 2008]

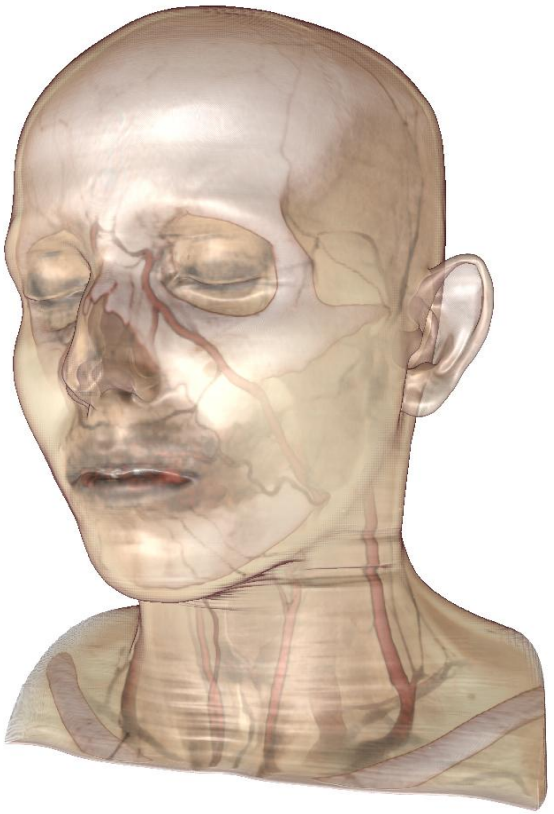
Hybrid Ray-Casting (1)

Isosurface rendering

- Find isosurface first
- Semi-transparent shading provides surface information

Additional unshaded DVR

- Render volume behind the surface with unshaded DVR
- Isosurface is starting position
- Start with (1.0 - iso_opacity)



Hybrid Ray-Casting (2)

Hiding sampling artifacts (similar to interleaved sampling)

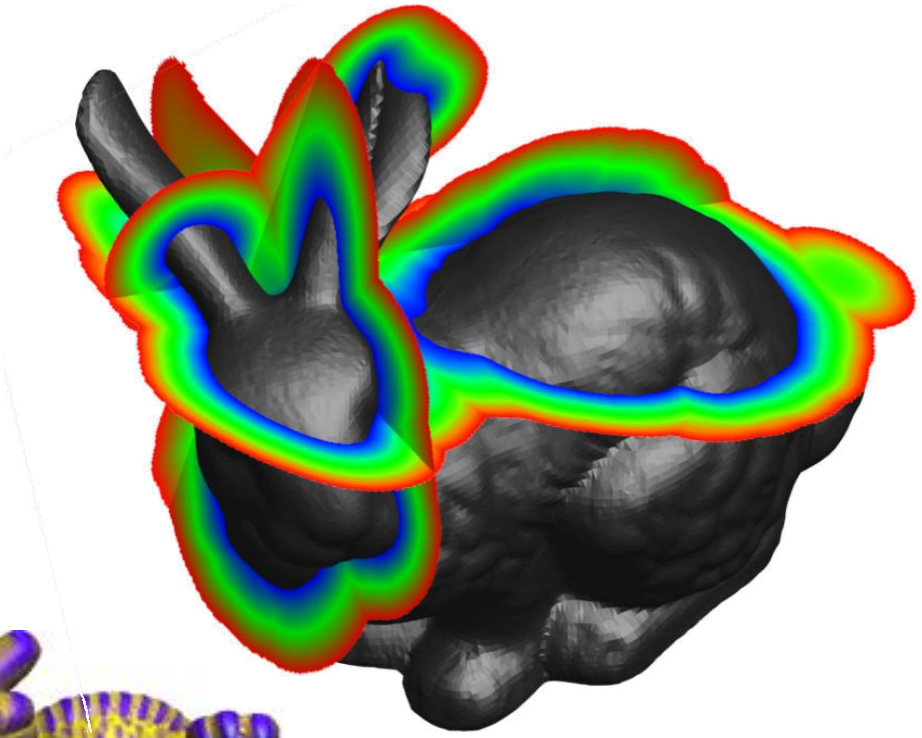
Signed Distance Fields

- Volume / scalar field for “objects”
- Signed distance to surface

$$\phi : \mathbb{R}^3 \mapsto \mathbb{R}$$

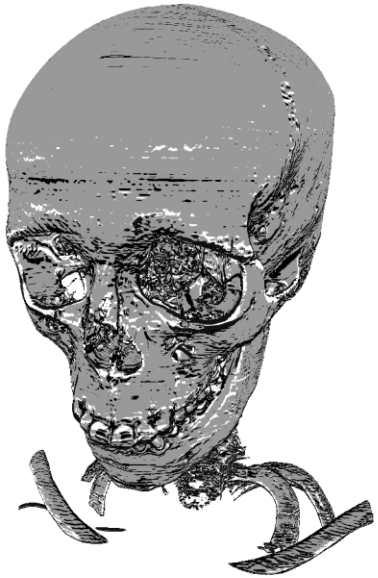
- Implicit surface: zero level set

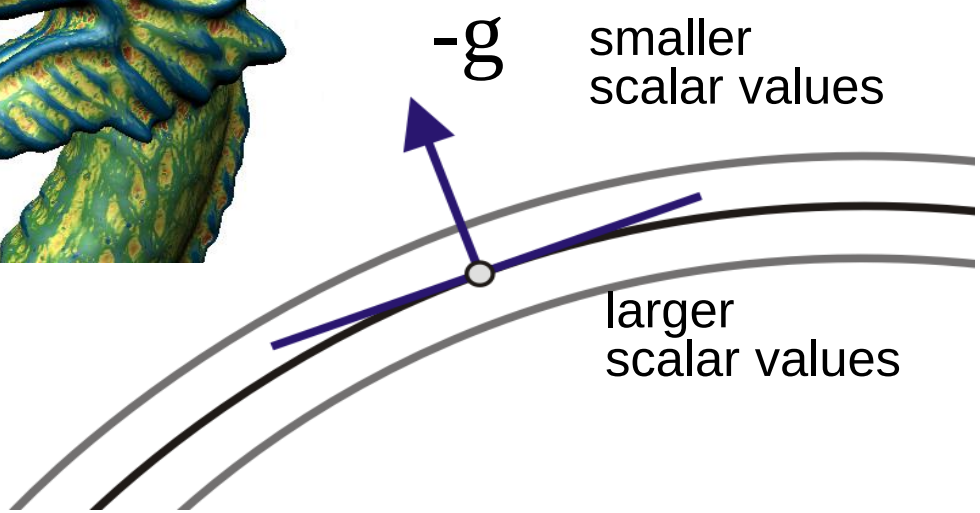
$$S = \{\mathbf{x} | \phi(\mathbf{x}) = 0\}$$



Curvature-Based Isosurface Illustration

- Curvature measure color mapping
- Curvature directions; ridges and valleys



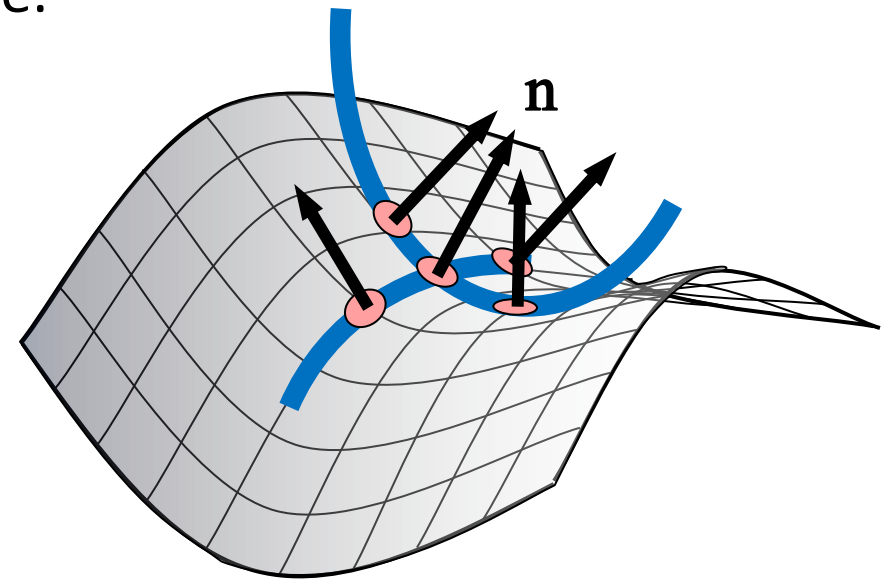


- Implicit surface curvature
- Isosurface through a point

Curvature

How do small positional changes on the surface change the normal vector?

- "Derivative" of normal
- First and second principal curvature:
  maximum: κ_1 minimum: κ_2
- Curvature directions
- Curvature magnitudes



courtesy of Gordon Kindlmann

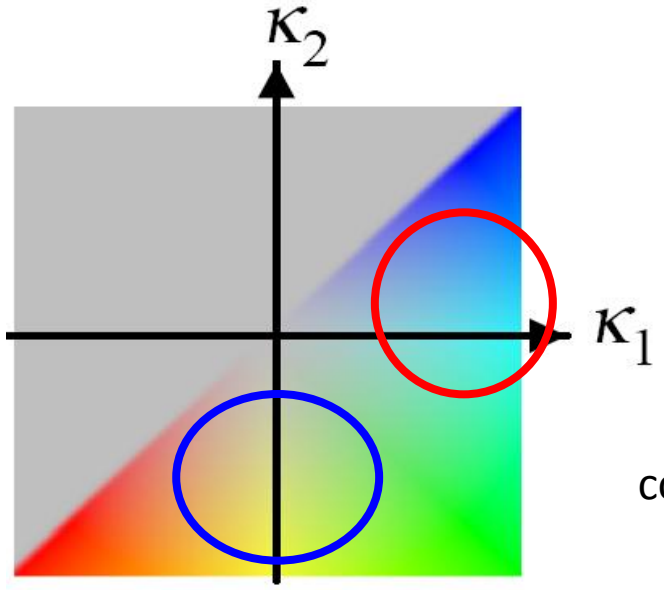
Principal Curvature Domain

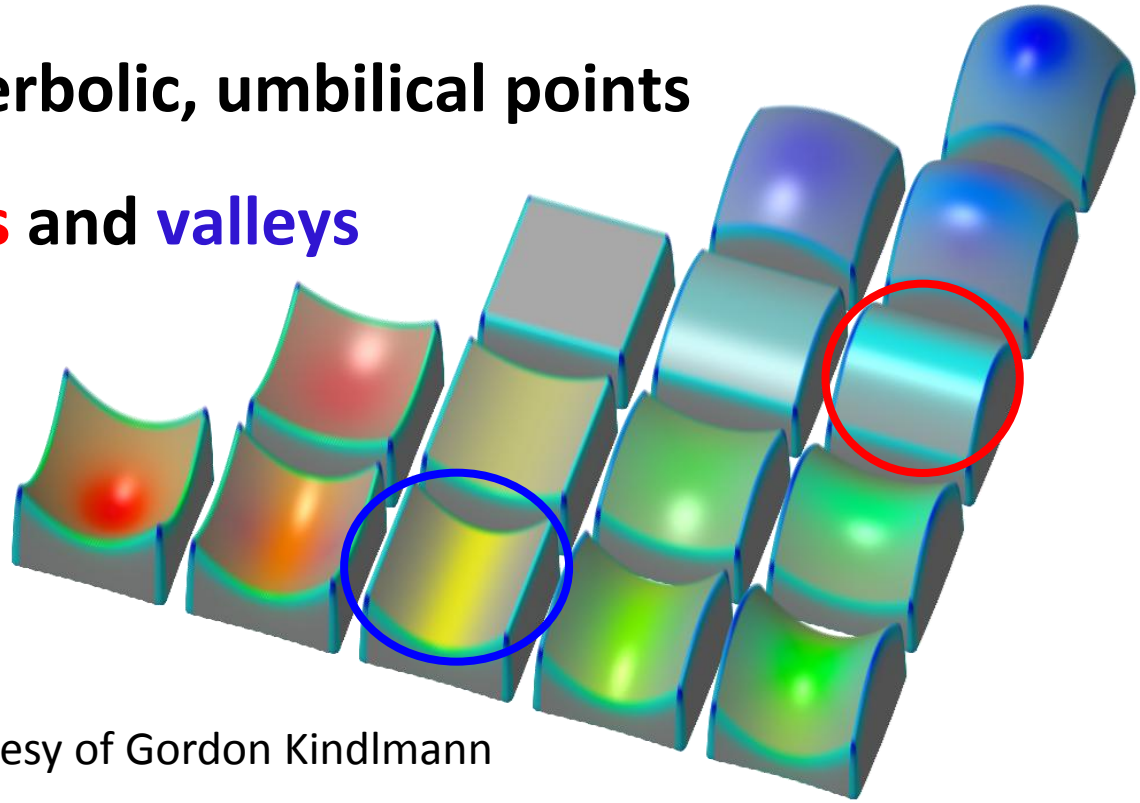
Maximum/minimum principal curvature magnitude

Identification of different shapes in 2D domain

Elliptical, parabolic, hyberbolic, umbilical points

Feature lines: e.g., ridges and valleys





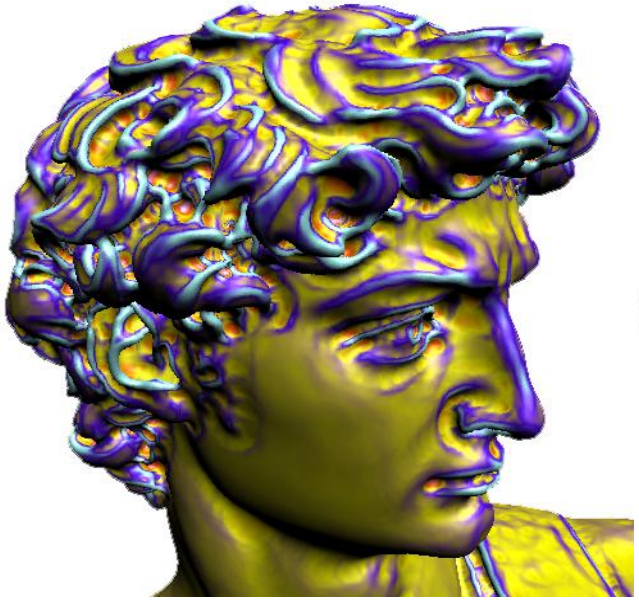
courtesy of Gordon Kindlmann

Deferred Isosurface Shading

Shading is expensive

- Compute surface intersection image from volume
- Compute derivatives and shading in image space

intersection image



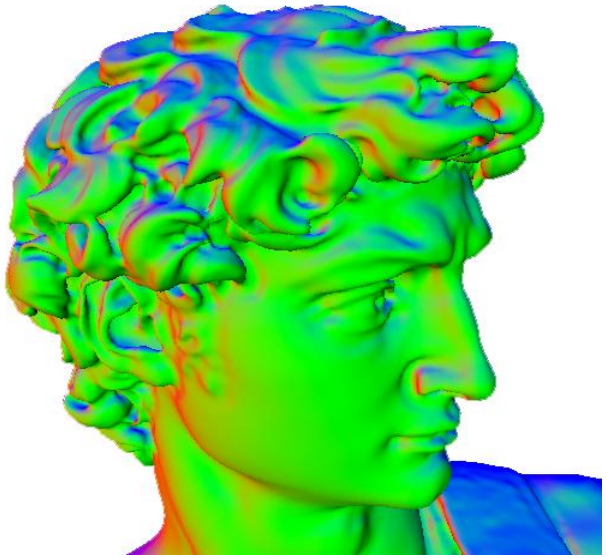
curvature color coding

ridges and valleys

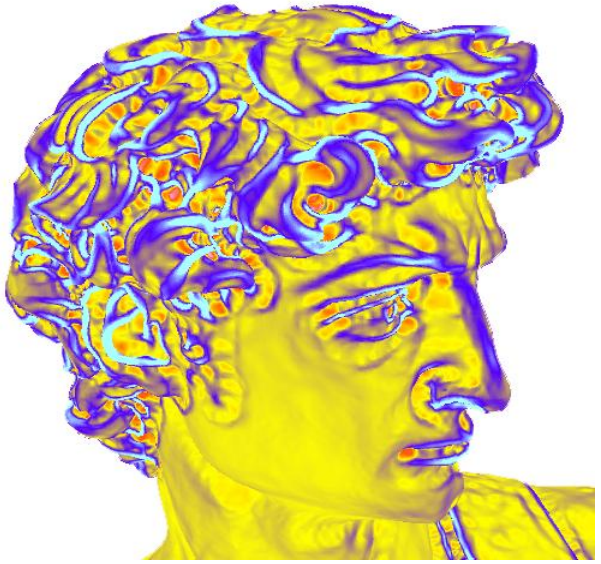
Implicit Curvature via Convolution

Computed from first and second derivatives

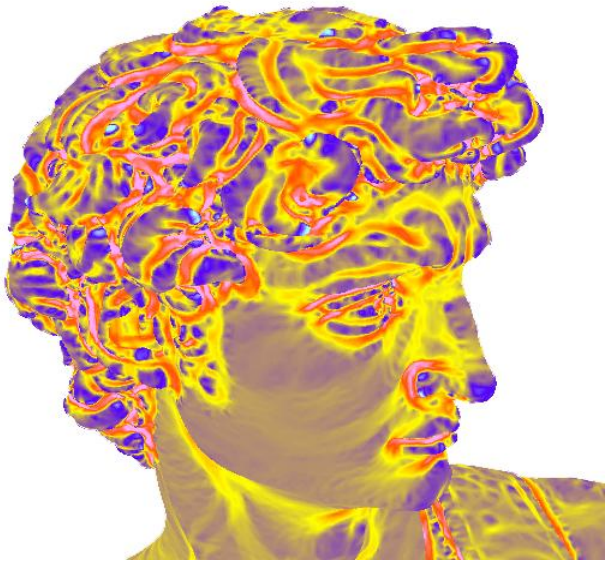
- Can use fast texture-based tri-cubic filters in shader
- Can use deferred computation and shading



first derivative



maximum curvature



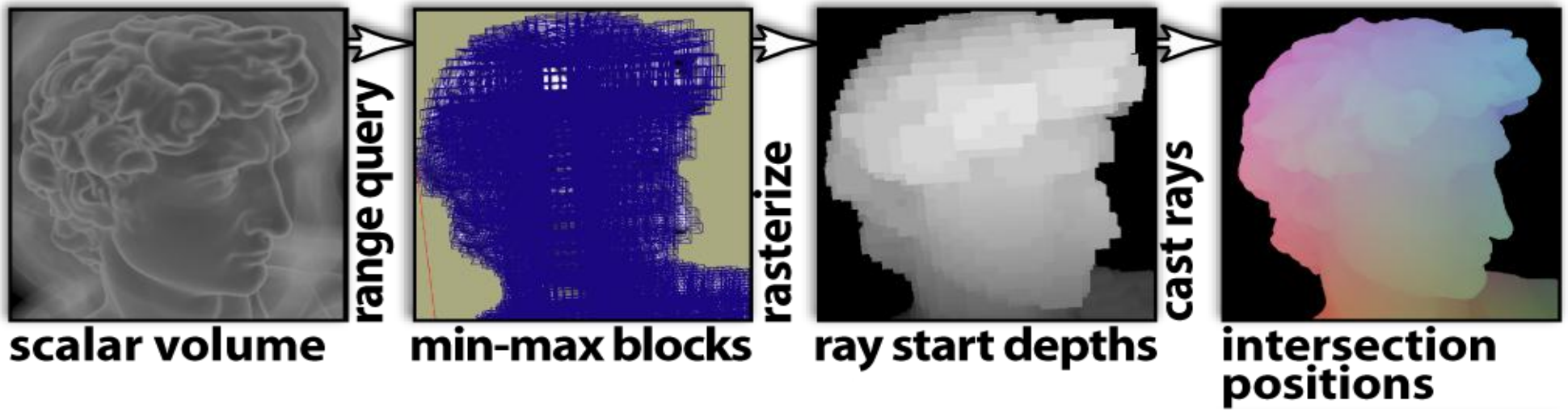
minimum curvature

Pipeline Stage #1: Ray-Casting

Rasterize faces of active min-max blocks

Cast into the volume; stop when isosurface hit

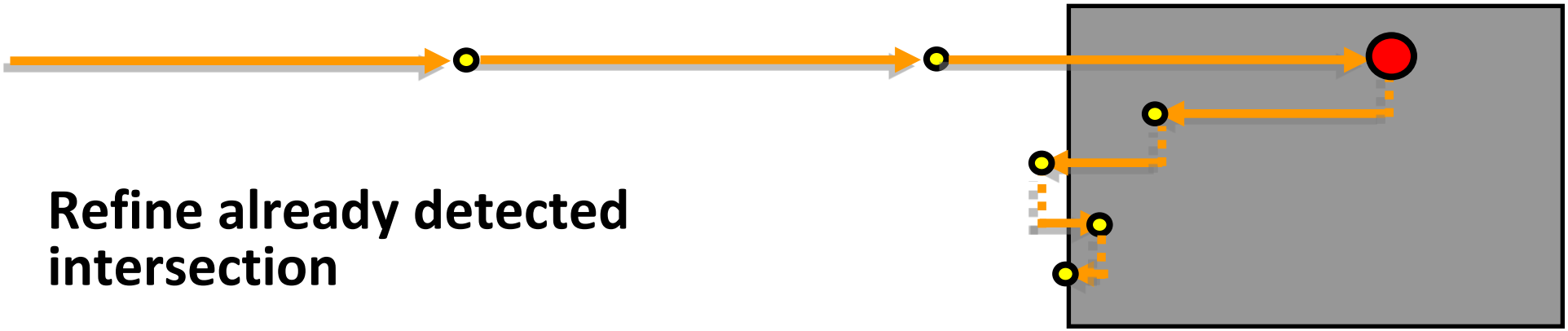
Refine isosurface hit positions (root search)



Intersection Refinement (1)

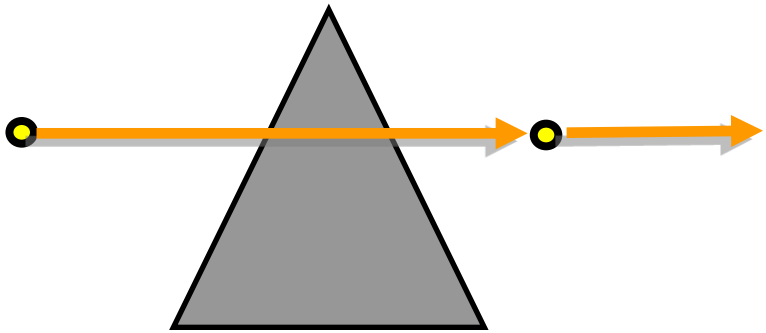
Fixed number of bisection steps

- Virtually no impact on performance



Refine already detected intersection

Handle problems with small features / at silhouettes with adaptive sampling



Intersection Refinement (2)

without refinement

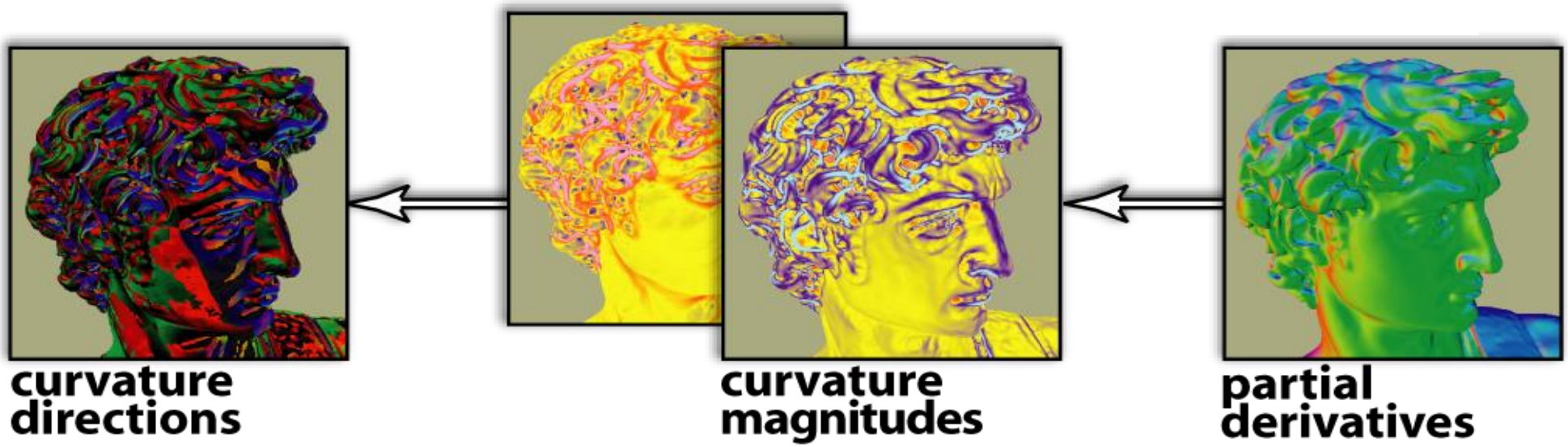
with refinement

sampling rate 1/5 voxel (no adaptive sampling)

Pipeline Stage #2: Differential Properties

Basis for visualization of surface shape

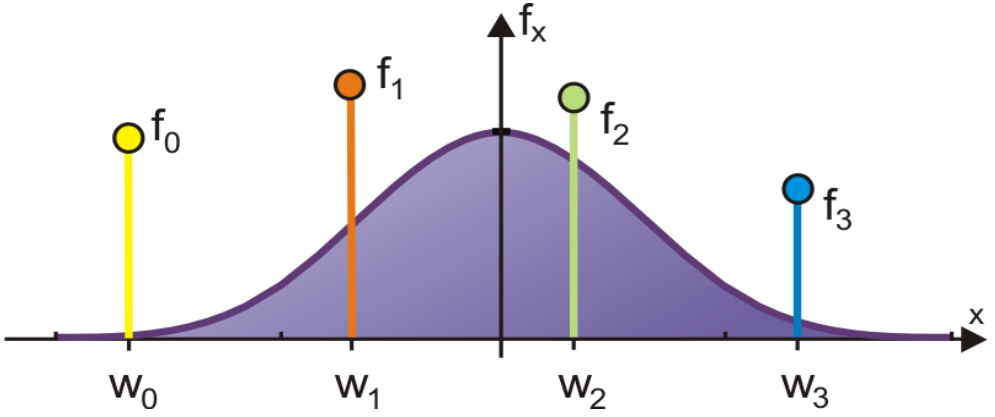
- First and second derivatives (gradient, Hessian)
- From these: curvature information, …



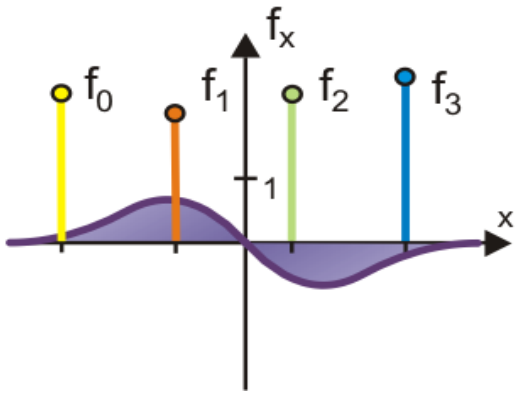
Value and Derivative Reconstruction

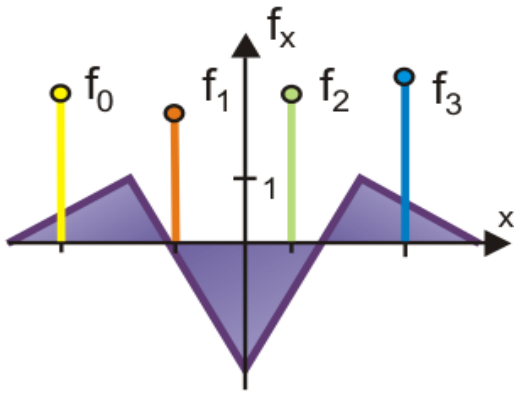
Cubic B-spline and derivatives

- Use 1D kernels and tensor product for tri-cubic filtering
- Well-suited for curvature computation [Kindlmann et al., 2003]



Expensive convolution?





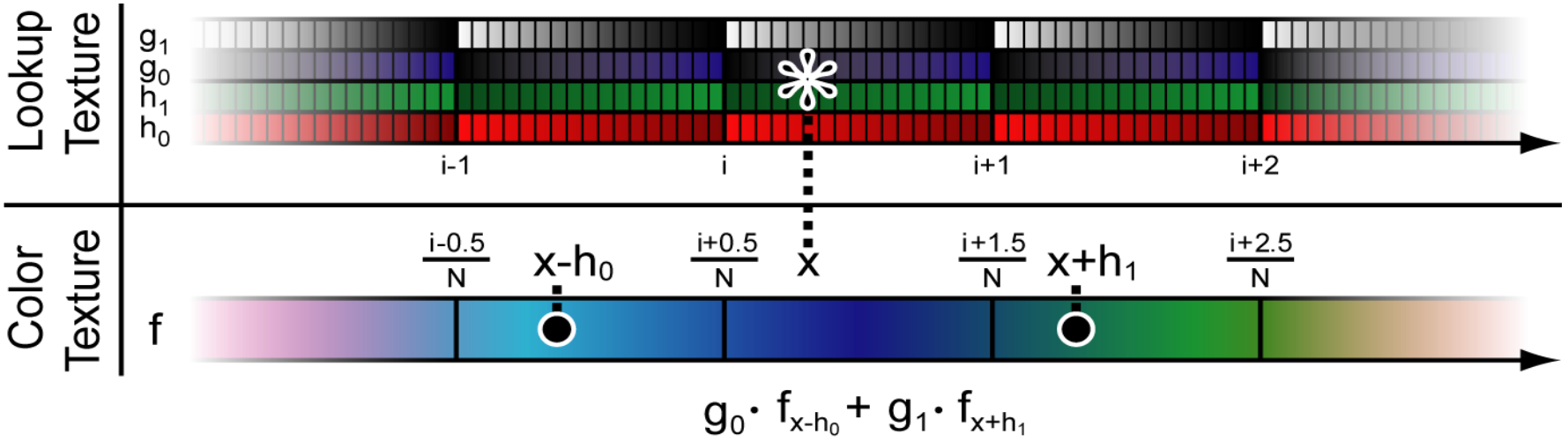
Fast Tri-Cubic Filtering on GPUs

Usually 64 nearest-neighbor lookups

But 8 tri-linear lookups suffice for tri-cubic B-spline

Kernels are transformed into 1D textures

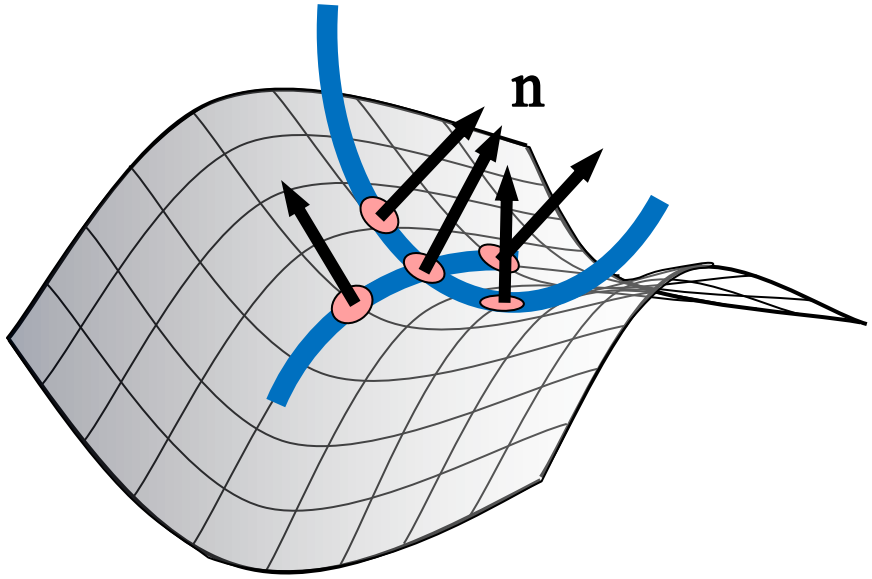
[Sigg and Hadwiger, 2005] (GPU Gems 2)



Curvature Computation

Build on gradient and Hessian matrix

Hessian contains curvature information



- Transform Hessian into tangent space
- Curvature magnitudes: eigenvalues of 2x2 matrix
- Curvature directions: eigenvectors of 2x2 matrix

Pipeline Stage #3: Shading

Build on previous images

- Position in object space
- Gradient
- Principal curvature magnitudes and directions


curvature flow

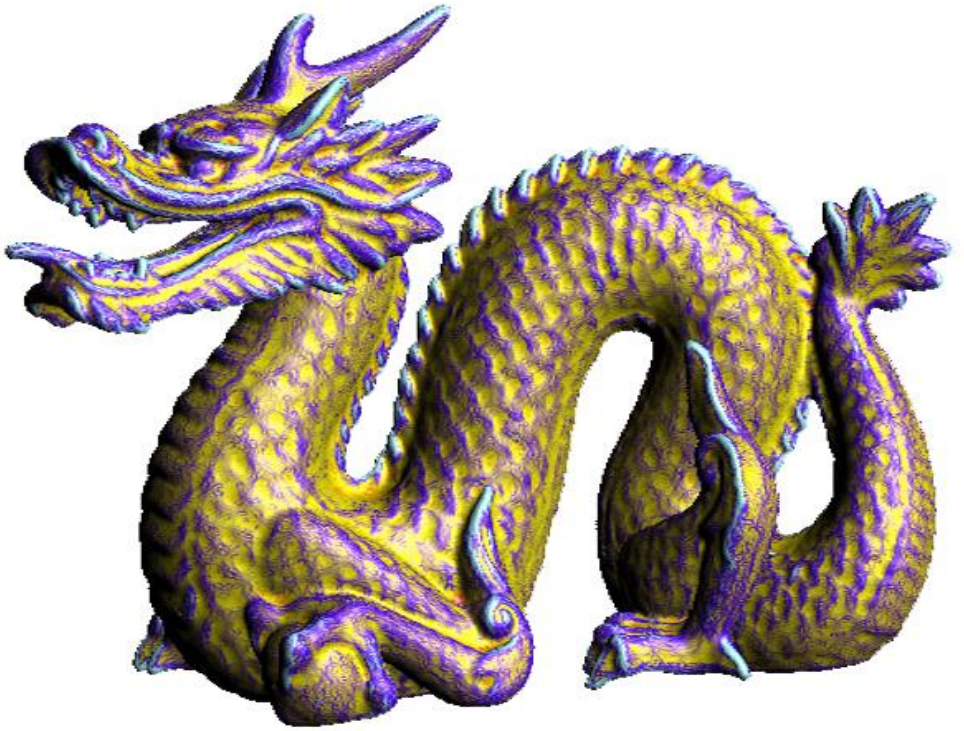

ridges+valleys


curvature mapping

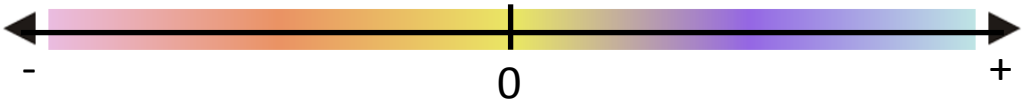

tone shading

Color Coding Scalar Curvature Measures

1D color lookup table



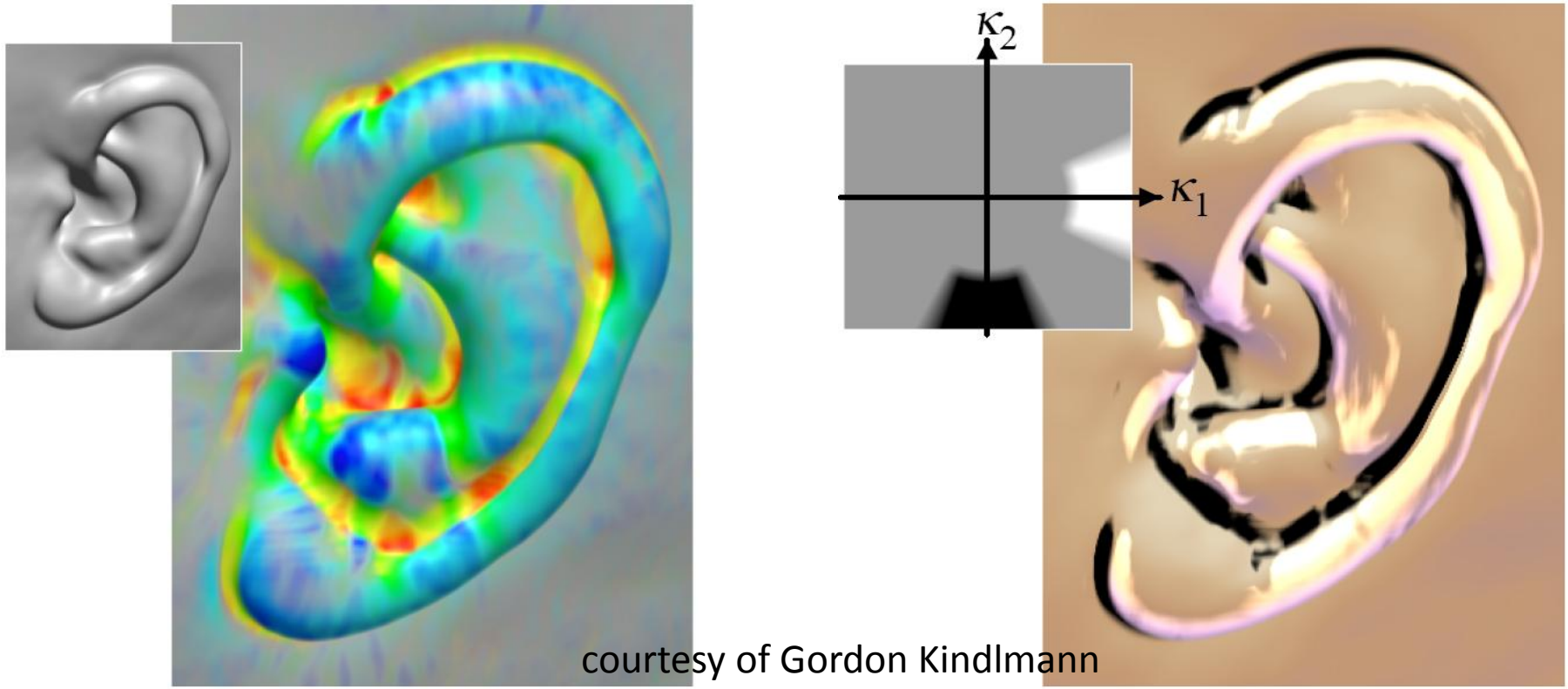
maximum principal curvature



Curvature Transfer Functions (1)

Color coding of curvature domain

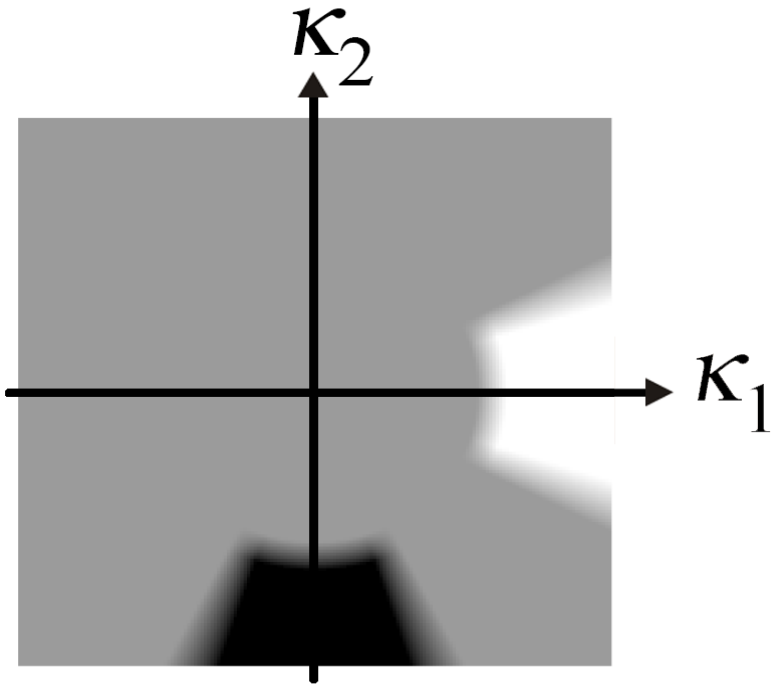
Paint features: ridge and valley lines



courtesy of Gordon Kindlmann

Curvature Transfer Functions (2)

2D lookup table in domain of principal curvatures

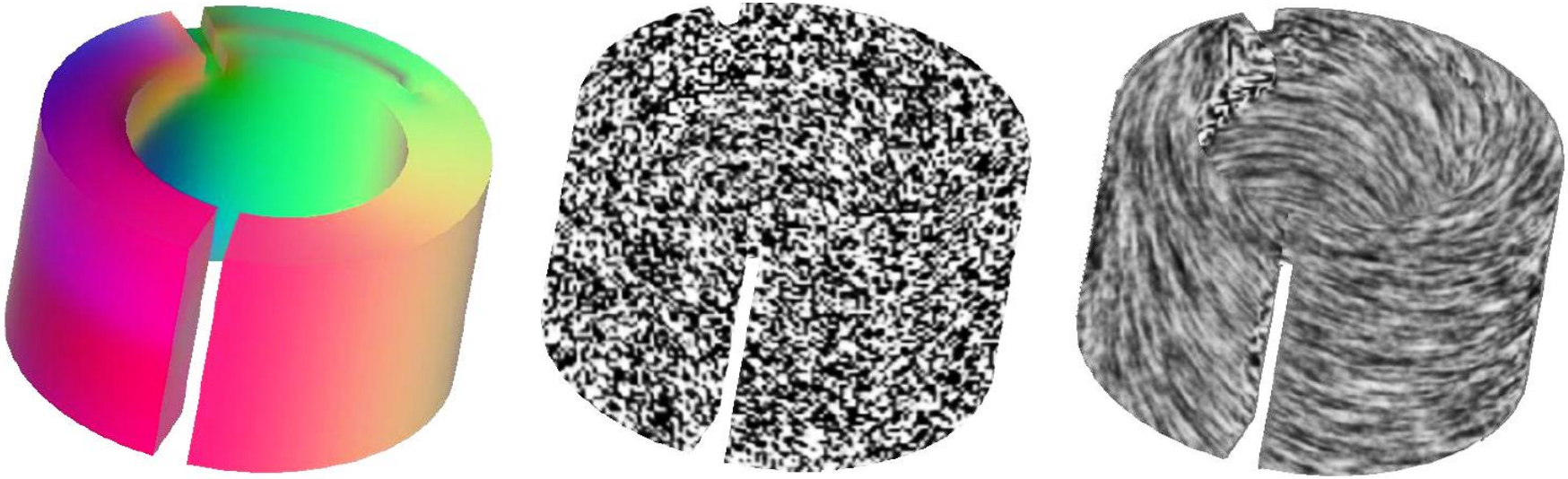


ridges and valleys, plus contours:

Visualizing Curvature Directions (1)

Use 3D vector field visualization on curved surfaces [van Wijk, 2003], [Laramee et al., 2003]

- Project 3D vectors to screen space
- Advect dense noise textures in screen space



Visualizing Curvature Directions (2)

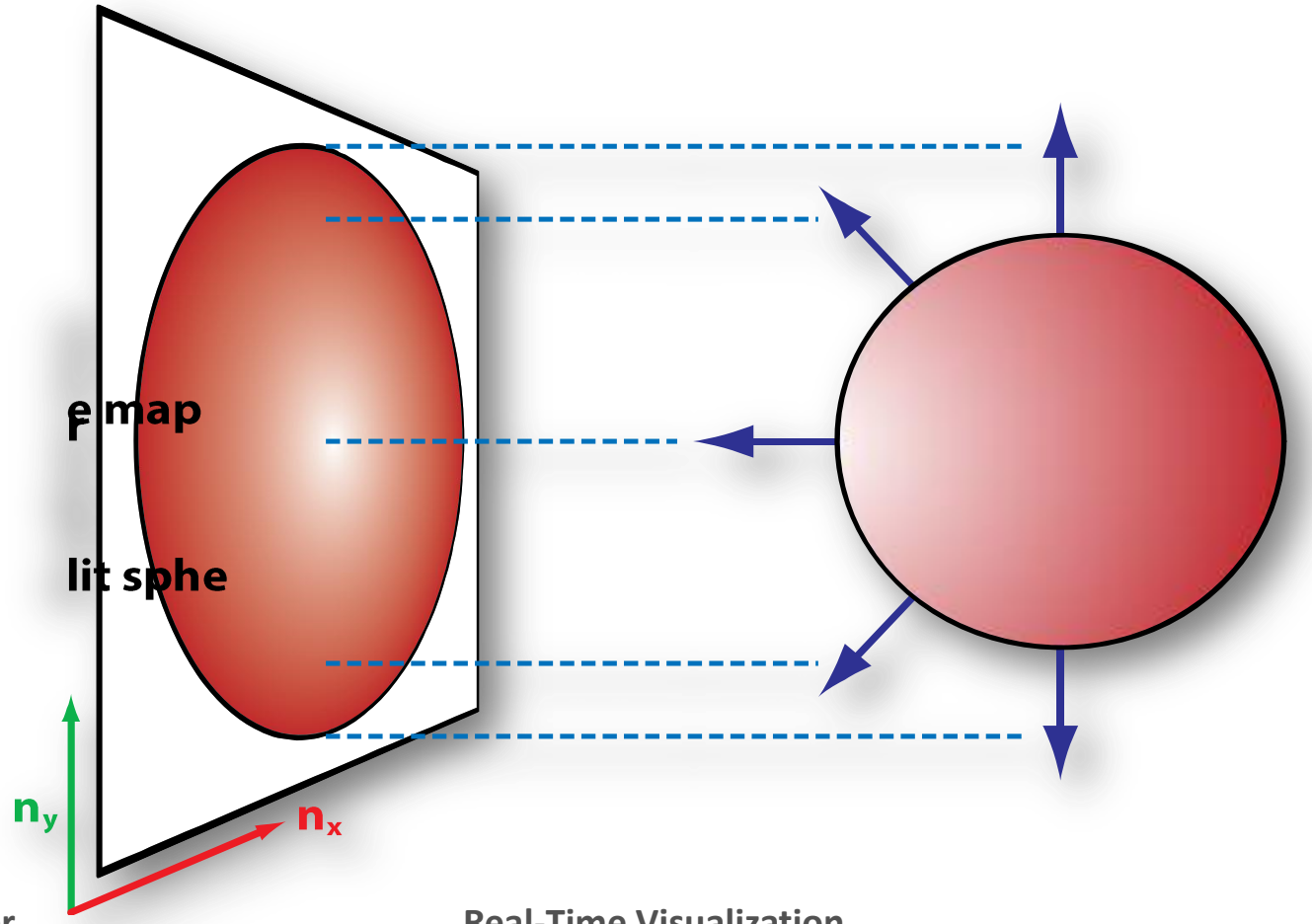
Lit Sphere Maps [Sloan et al. 1998] (1)

Use an image of a sphere under orthographic projection to shade another object

- Like environment mapping, but eye-space normal is used instead of reflection vector
- Light sources appear to be fixed to the camera
- Flexible image-based illumination, captures many different rendering styles

Lit Sphere Maps [Sloan et al. 1998] (2)

Use a sphere map indexed by the eye-space normal to determine the color of a point

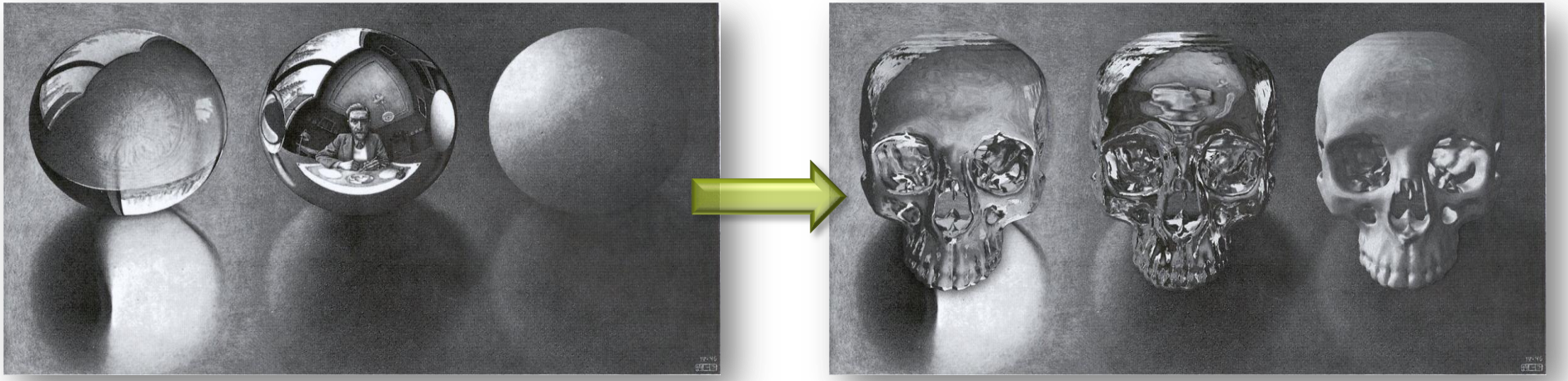


Lit Sphere Maps [Sloan et al. 1998] (3)

Easy to obtain – lighting studies are frequently performed using spheres

Sloan et al. describe simple extraction process from existing works of art

Intuitive representation, can be directly displayed to the user as a preview



Style Transfer Functions (1)

Use lit sphere maps to allow data-dependent illustrative shading for volume rendering

- One lit sphere maps represents one specific rendering style
- Transfer function is defined over styles instead of colors
- Combines the power of data-dependent lighting with the flexibility of lit sphere maps

Style Transfer Functions (2)

Regular Transfer Function

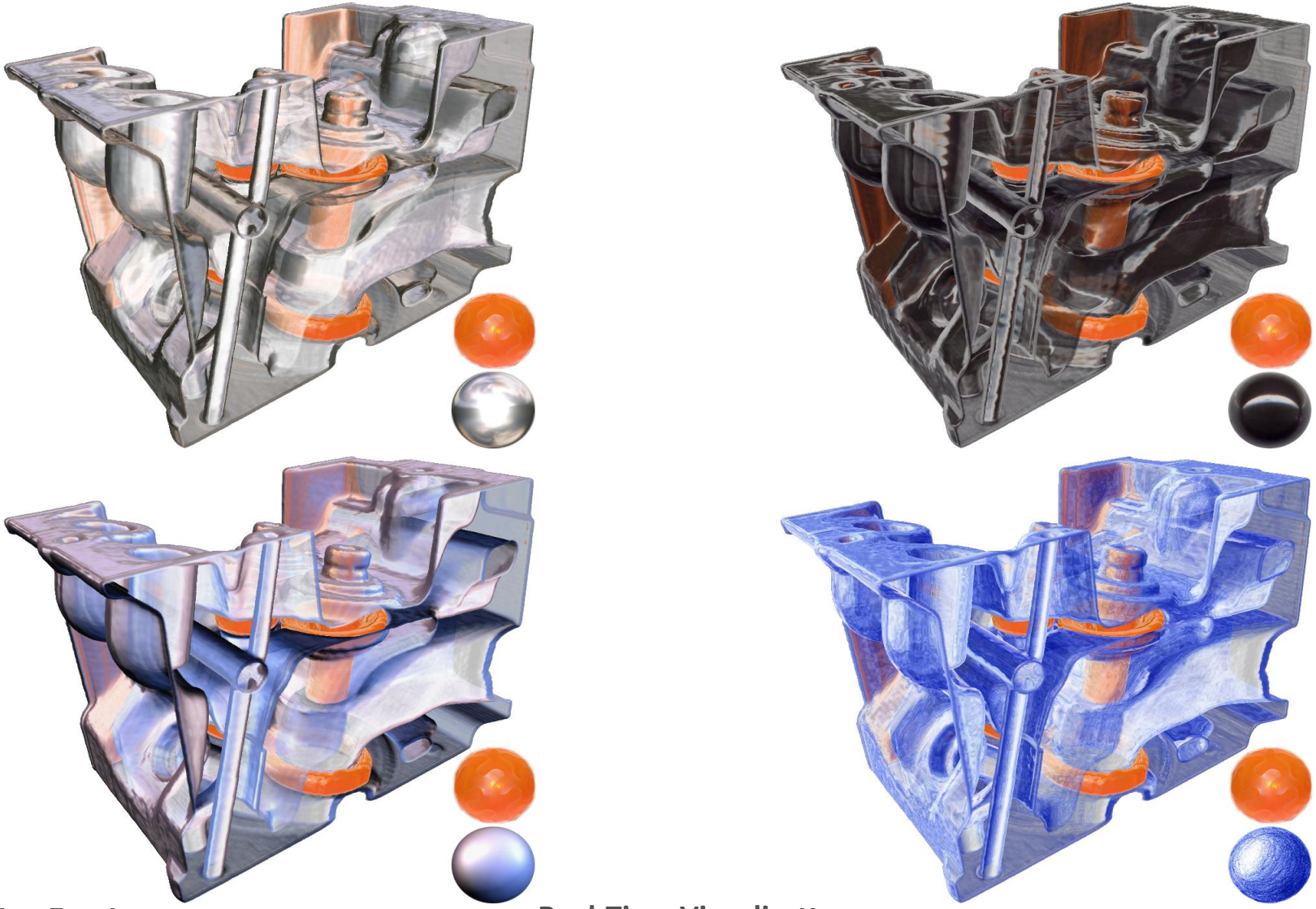
Style Transfer Function

Style Transfer Functions (3)

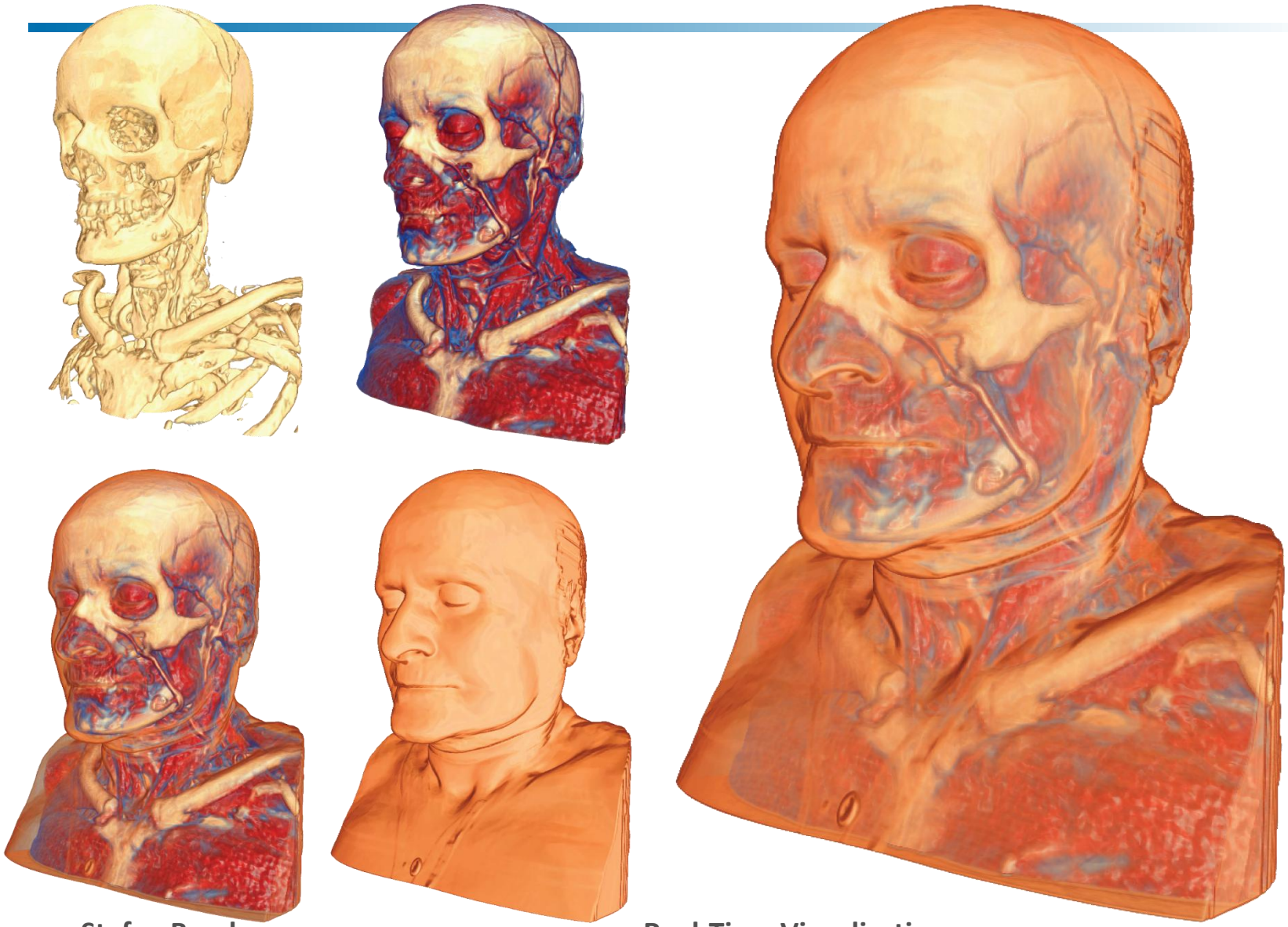
Replace color nodes in transfer function by 2D lit sphere maps

- Essentially a 3D transfer function of data value and eye-space normal: **stf**(s,n_x,n_y)
- Prohibitive storage requirements – split up into two functions: **sf**(**tf**$(s))(n_x,n_y)$
- Linear blending between styles – complex transitions possible through intermediate styles

Style Transfer Functions (4)



Style Transfer Functions (5)



Style Contours (1)

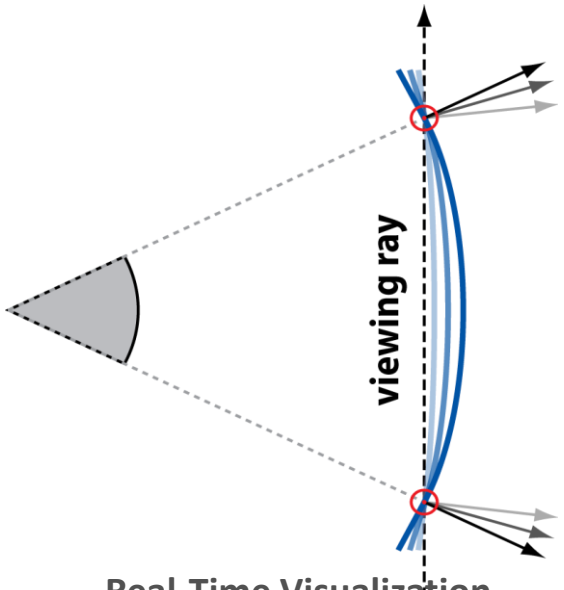
Contours are a frequent stylistic element in illustrations

- Contour appearance should be derived from lit sphere map
- Apparent contour thickness varies based on curvature
- Solution by [Kindlmann et al. 2003]: use normal curvature along the view direction to modulate contour threshold

Style Contours (2)

Kindlmann's approach requires expensive reconstruction of 2nd order derivatives

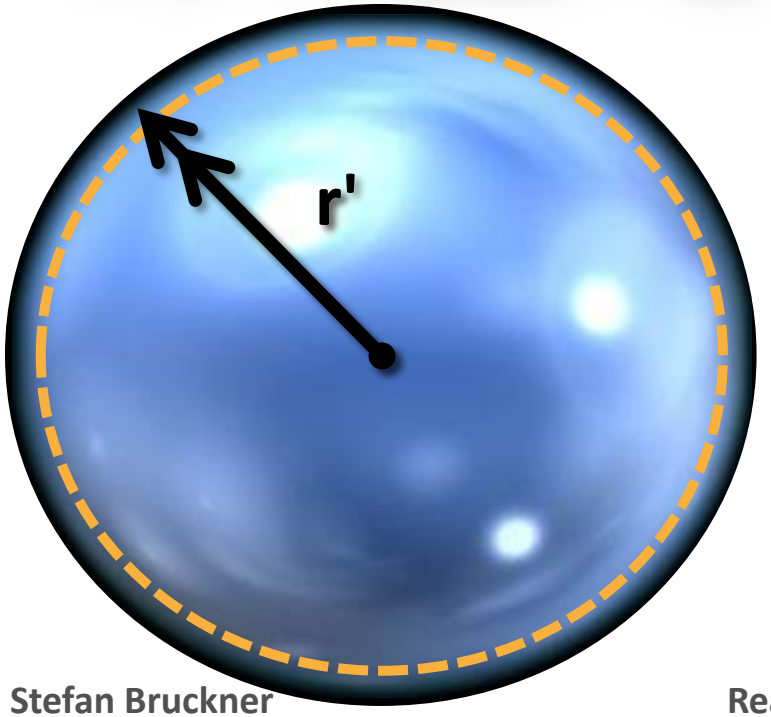
Simple approximation: angle between the gradient direction at two subsequent sample locations along a ray divided by step size



Style Contours (3)

Instead of simple threshold, push lit sphere lookup coordinates outwards along the radius based on fuzzy "contourness" criterion

$$\text{if } |n \cdot v| \leq \sqrt{k_v(2-k_v)}$$



$$\delta = 1 - \min(1, \frac{\sqrt{k_v(2-k_v)} - |n \cdot v|}{\sqrt{k_v(2-k_v)}})$$

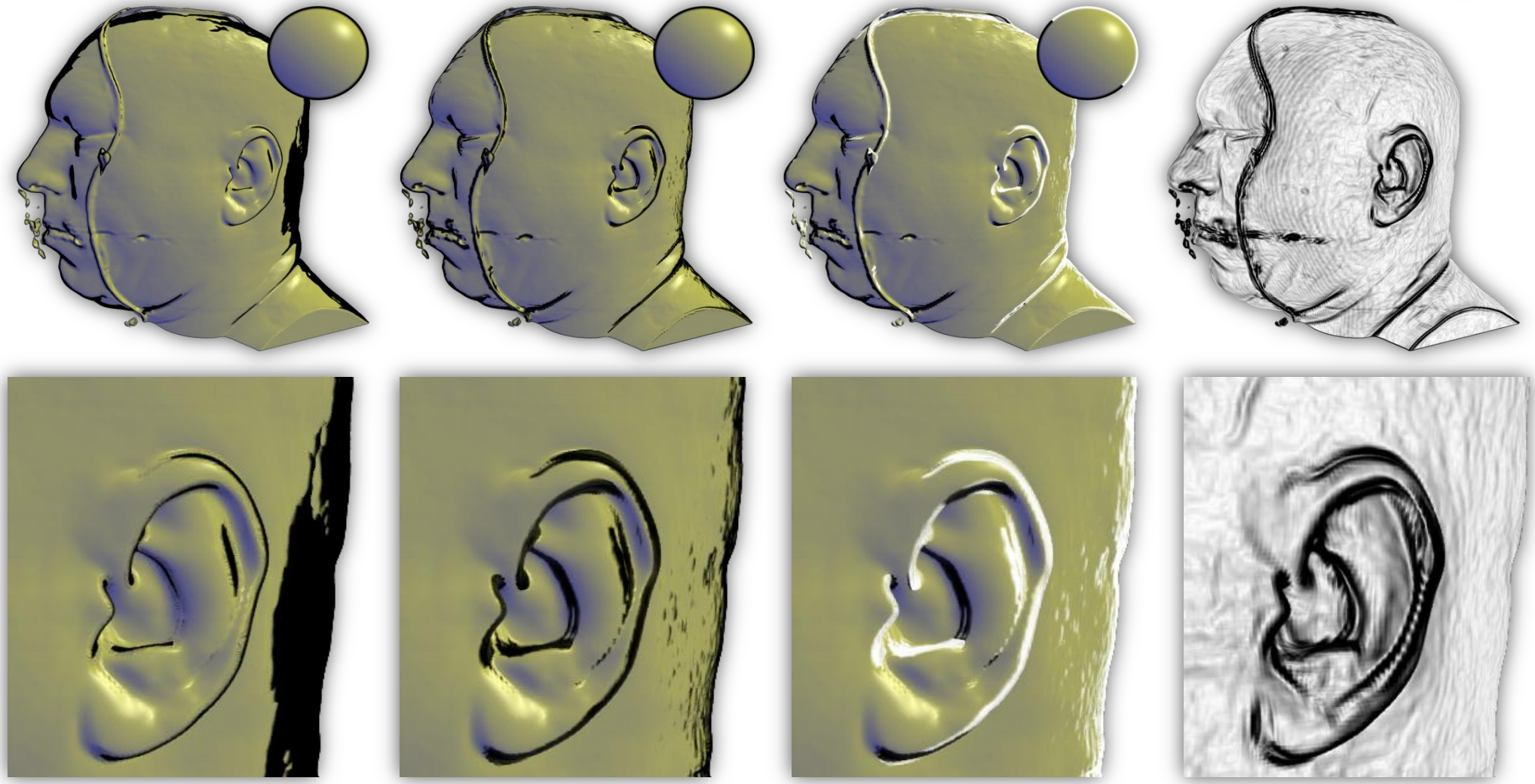
$$r' = \min(1, \frac{r}{\delta})$$

Style Contours (4)

normal contours

thickness-controlled contours

curvature image



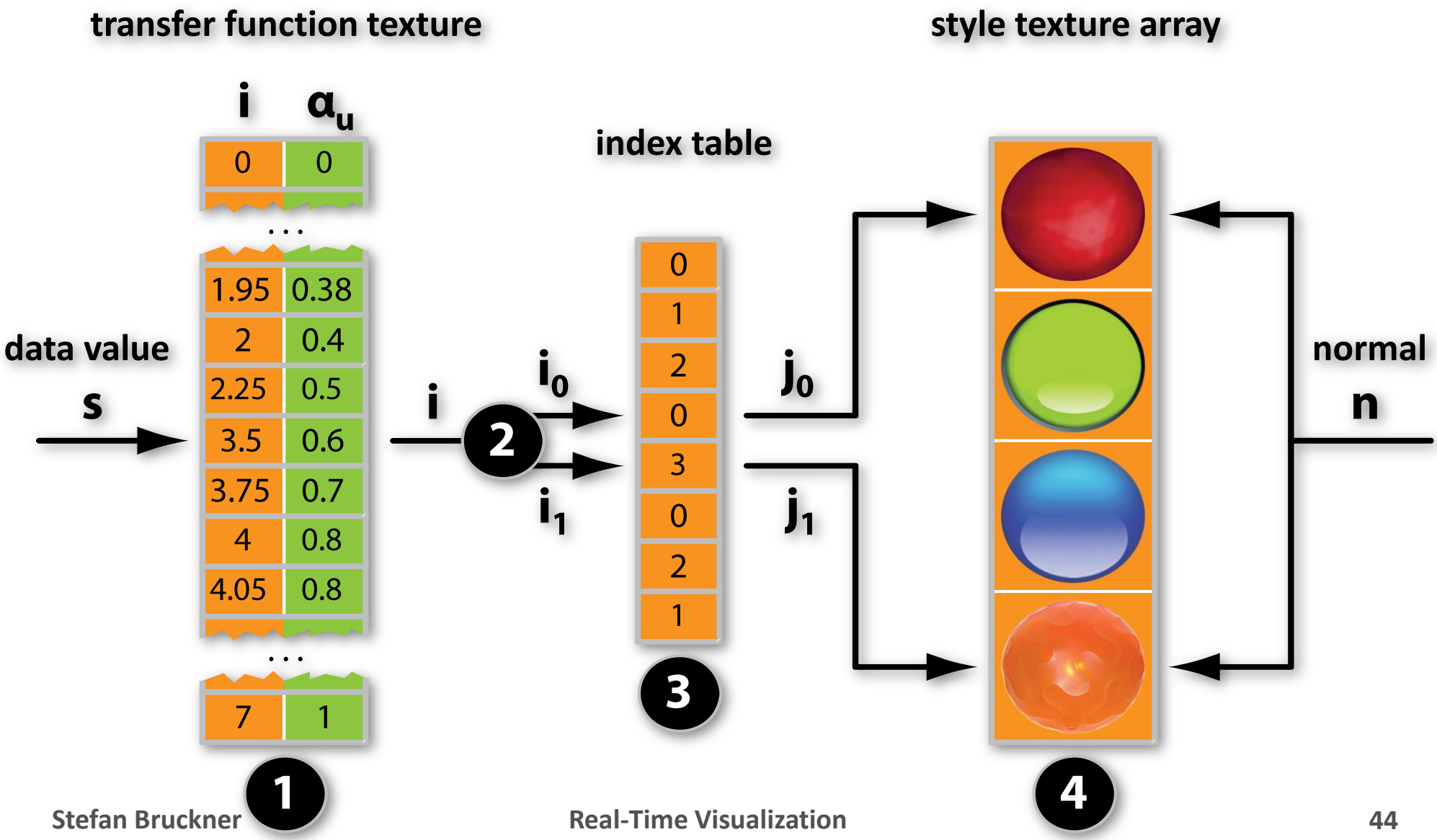
Implementation (1)

Easy integration into existing GPU-based ray casting algorithms

Style transfer function lookups require three textures

- 1D transfer function texture
  - Stores node indices and opacities
- 1D index table
  - Maps node indices to style indices
- 2D x N style texture array
  - Stores set of styles (lit sphere maps)

Implementation (2)



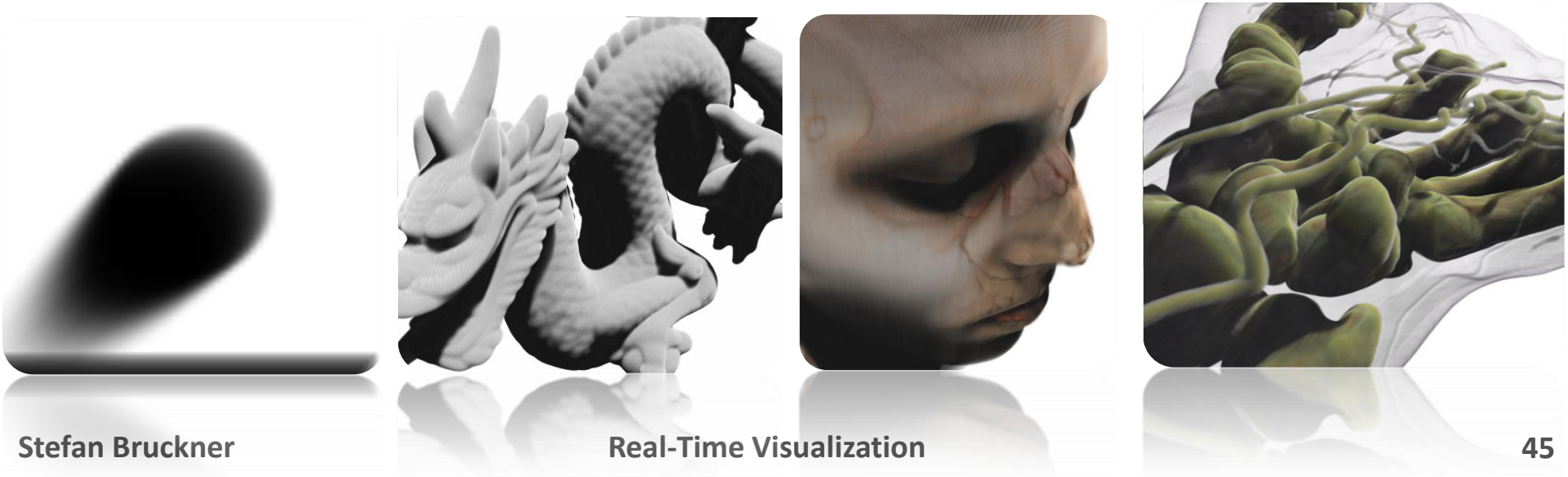
Global Volume Illumination

Local illumination might sufficient for many application areas in scientific visualization (e.g medicine)

- Not sufficient for visual arts/photorealism

Appearance of many common objects is dominated by scattering effects

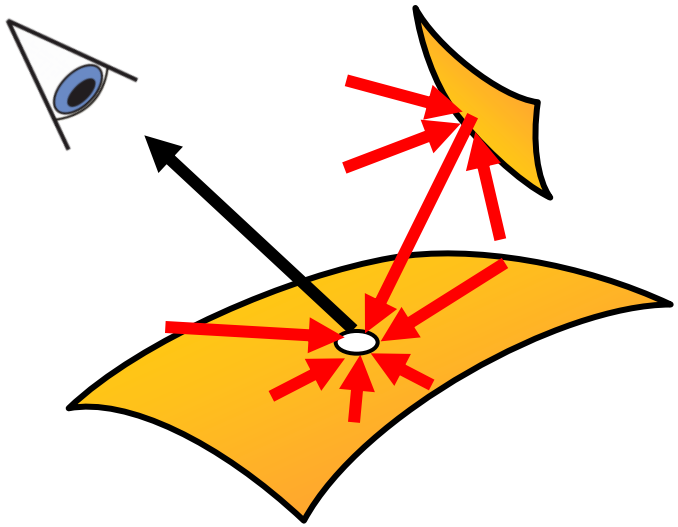
- Smoke, clouds, wax, skin, …



Surface vs. Volume Illumination

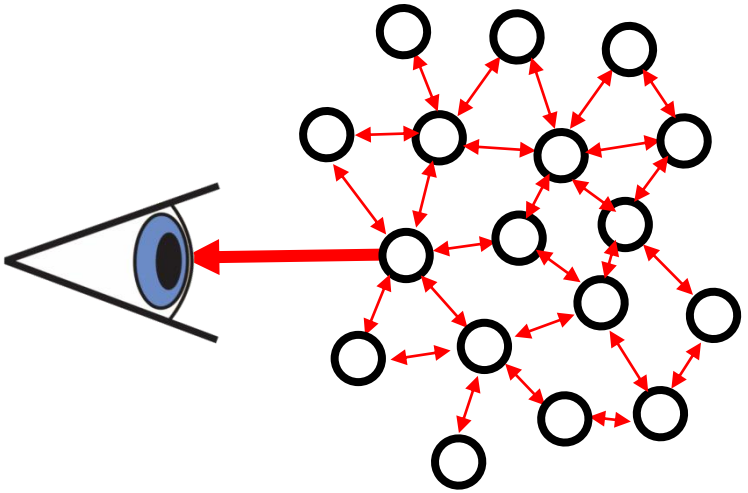
Surface Lighting

- Light transport in vacuum
- Lighting calculation is performed at surface points
- Reflectivity from BRDF



Volume Lighting

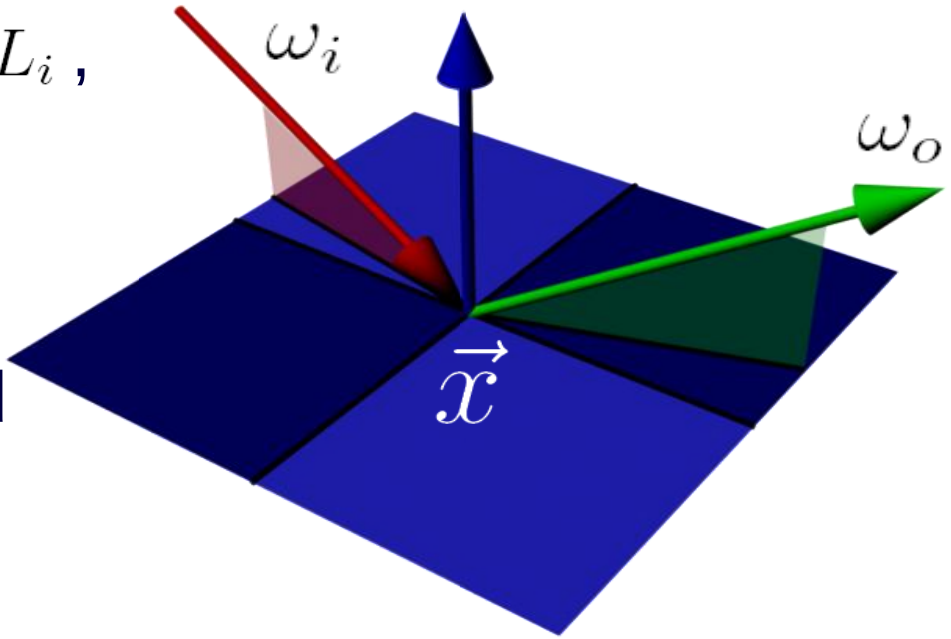
- Light transport in participating medium
- Lighing calculation at very point
- Scattering from phase function



Surface Illumination

The incoming radiance L_i ,
at a point $\vec{x}$
from direction
$\omega_i = (\theta_i, \phi_i)$

will partially be reflected
into direction
$\omega_o = (\theta_o, \phi_o)$

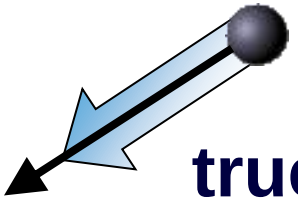


To obtain the radiance at $\vec{x}$, we must account for all possible incoming directions:

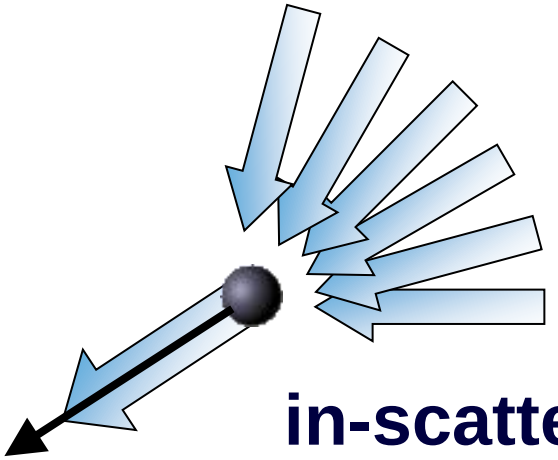
$$L_o(\vec{x}, \vec{\omega_o}) = \int_\Omega f(\vec{x}, \vec{\omega_i} \rightarrow \vec{\omega_o}) L_i(\vec{x}, \vec{\omega_i}) \cos\theta_i \, d\omega_i$$

Physical Model of Radiative Transfer

Increase



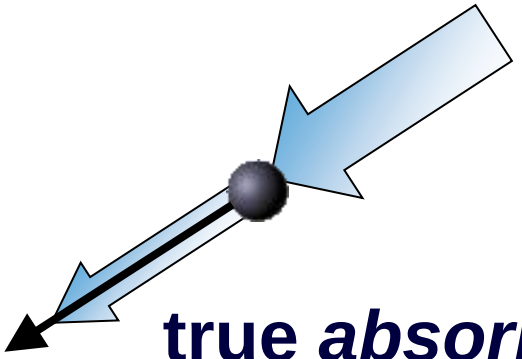
true emission



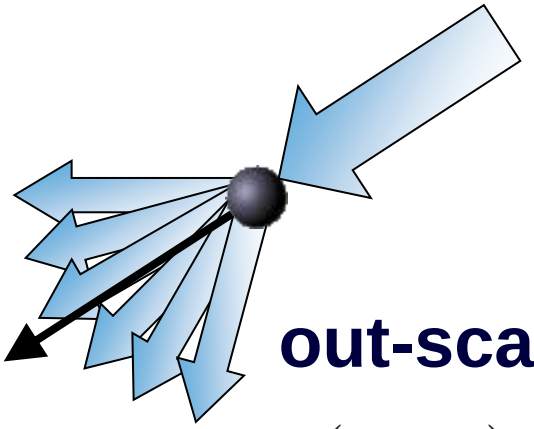
in-scattering

$$\eta(\mathbf{x},\omega) = q(\mathbf{x},\omega) + j(\mathbf{x},\omega)$$

Decrease



true *absorption*



out-scattering

$$\chi(\mathbf{x},\omega) = \kappa(\mathbf{x},\omega) + \sigma(\mathbf{x},\omega)$$

Scattering

In-scattering

Out-scattering coefficient

Phase function

Incoming radiance

$$j(\mathbf{x},\omega) = \frac{1}{4\pi}\int_{\text{sphere}} \sigma(\mathbf{x},\omega')p(\mathbf{x},\omega',\omega)I(\mathbf{x},\omega')\mathrm{d}\omega'$$

Volume Rendering Equation

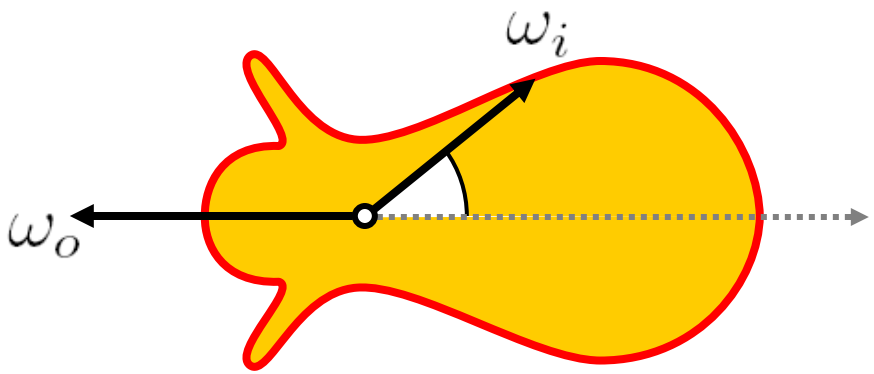
$$\omega \cdot \nabla_{\mathbf{x}} I(\mathbf{x},\omega) = -(\kappa(\mathbf{x},\omega)+\sigma(\mathbf{x},\omega))I(\mathbf{x},\omega) + q(\mathbf{x},\omega) + \int_{\text{sphere}} \sigma(\mathbf{x},\omega')p(\mathbf{x},\omega',\omega)I(\mathbf{x},\omega')\ \mathrm{d}\omega'$$

Phase Functions

For surfaces, the BRDF describes the probability of light being reflected from one direction on the hemisphere into another direction.

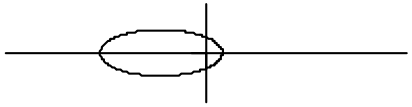
Analogously, for volumes, the phase function describes the probability of light being scattered from direction w_i into direction w_o

$$p(\mathbf{x}, \omega_o, \omega_i)$$



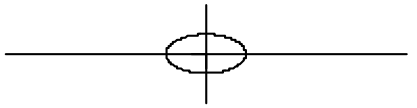
Henyey-Greenstein Phase Function

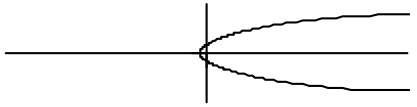
Empirical phase function



$g = -0.3$

$$p(\cos\theta) = \frac{1}{4\pi} \frac{1-g^2}{\left(1+g^2-2g\cos\theta\right)^{3/2}}$$





$g = 0.6$

$$2\pi \int_0^{\pi} p(\cos\theta)\cos\theta \, d\theta = g$$

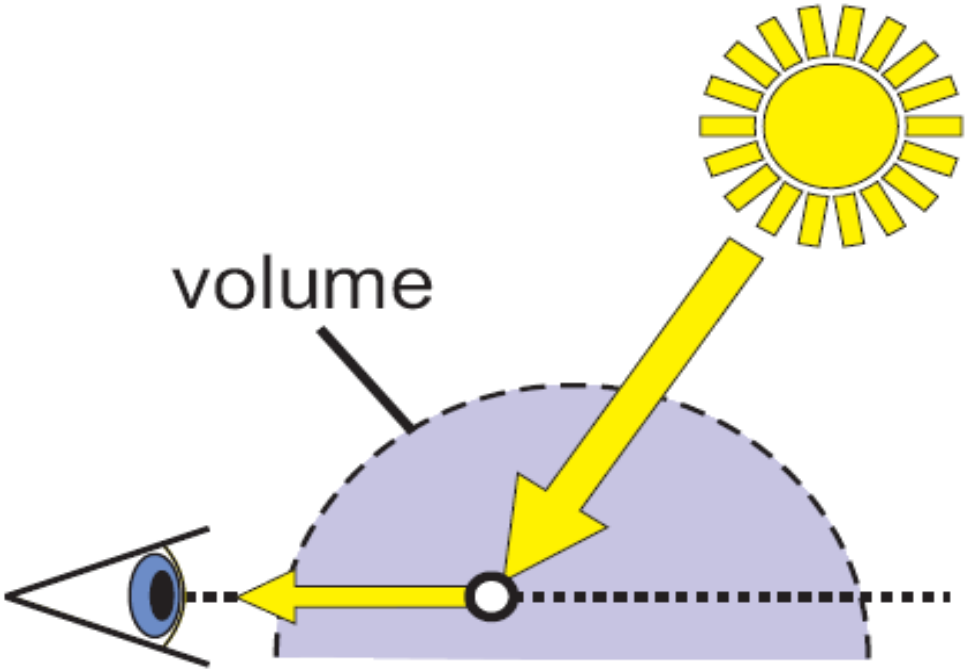
g: average phase angle

Volume Illumination – Attenuation (1)

Previously: External light is not attenuated

Single scattering without attenuation

- Light reaches every point unimpededly
- Light is scattered once before it reaches the eye

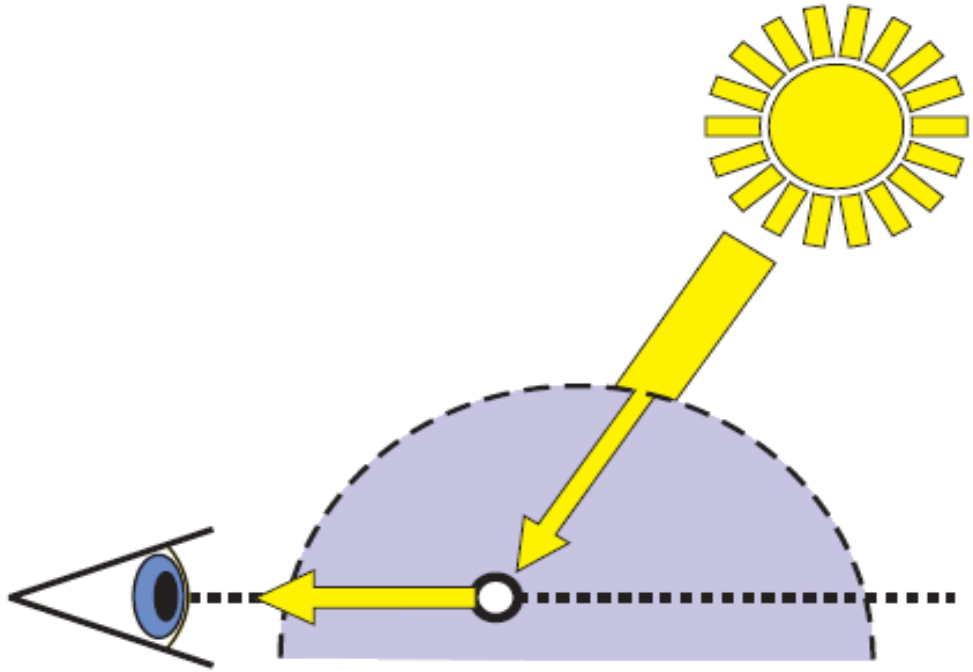


Volume Illumination – Attenuation (2)

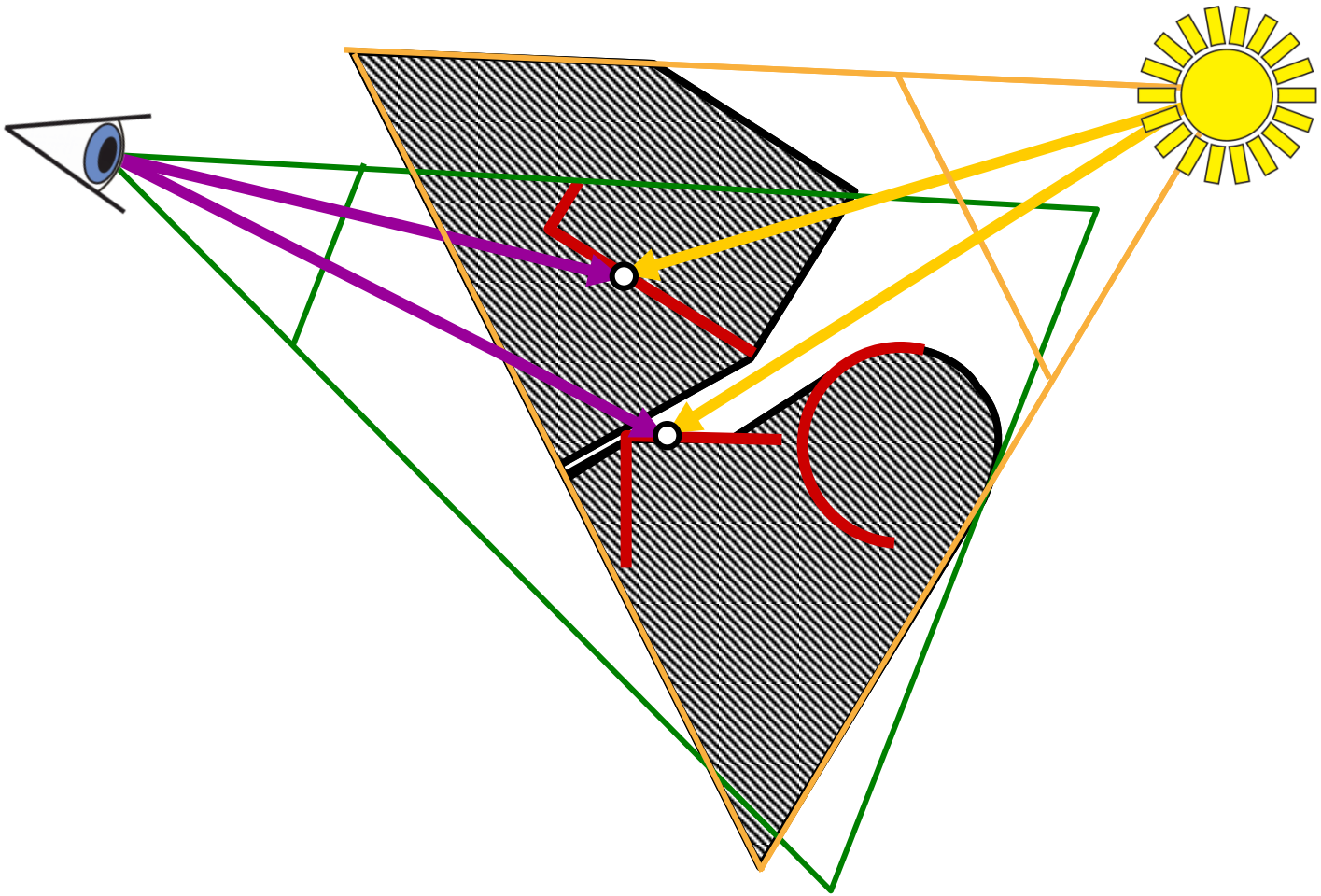
Now: Attenuation of light by the volume

Single scattering with attenuation

- Light is attenuated along its way through the volume
- Light is scattered once before it reaches the eye



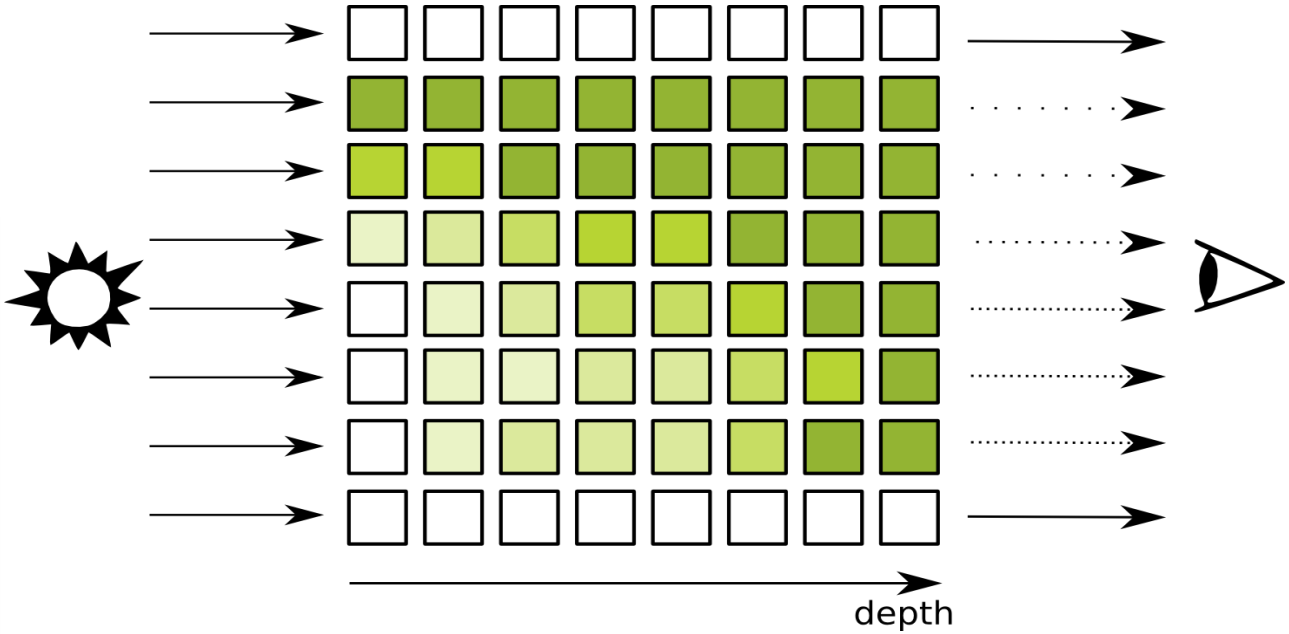
Surface Shadows



Volumetric Shadows (1)

Geometric soft shadows are due to area light source

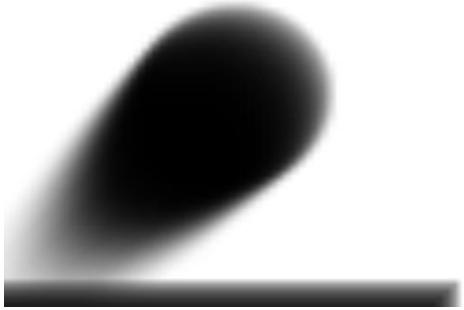
Volumetric shadows are due to continuous absorption along rays

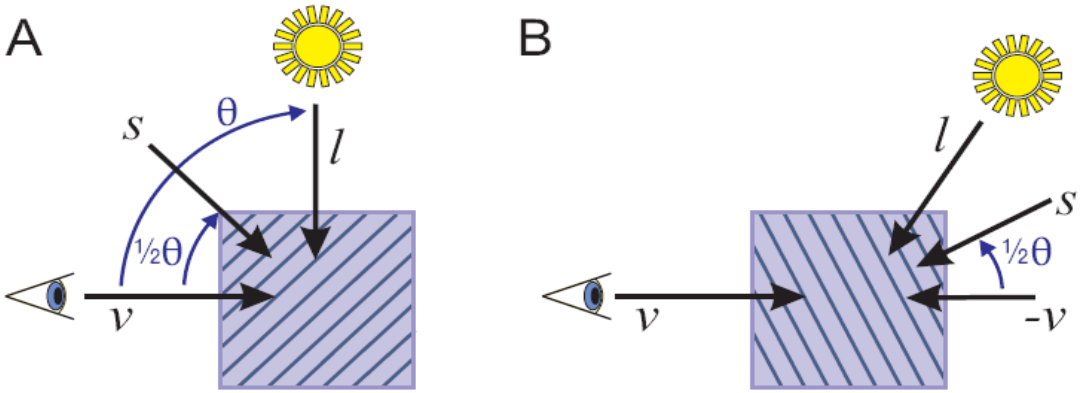


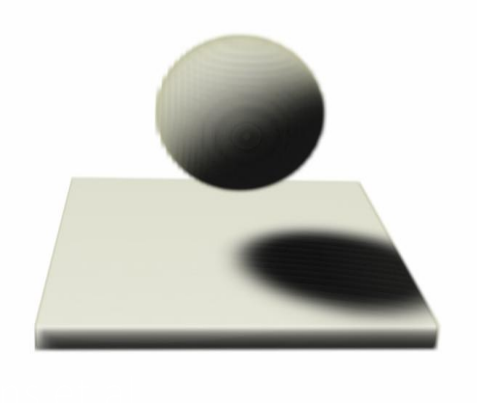
Volumetric Shadows (2)

Approaches in real-time volume rendering

- Create full “shadow volume” with accumulated absorption per voxel
- Half-angle slicing
- Deep shadow maps



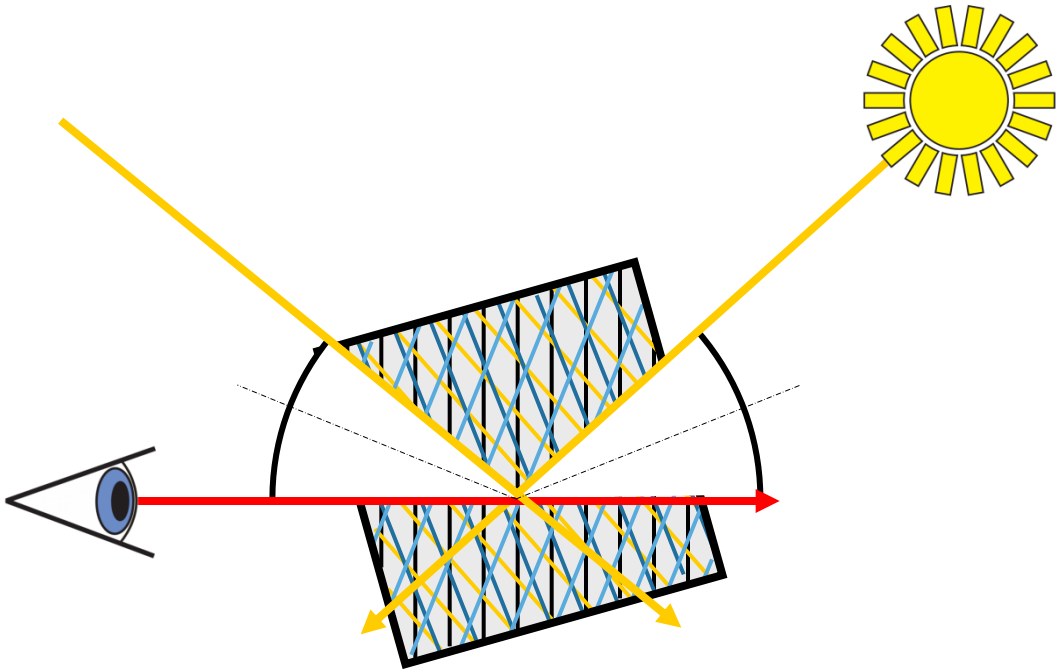




Half-Angle Slicing

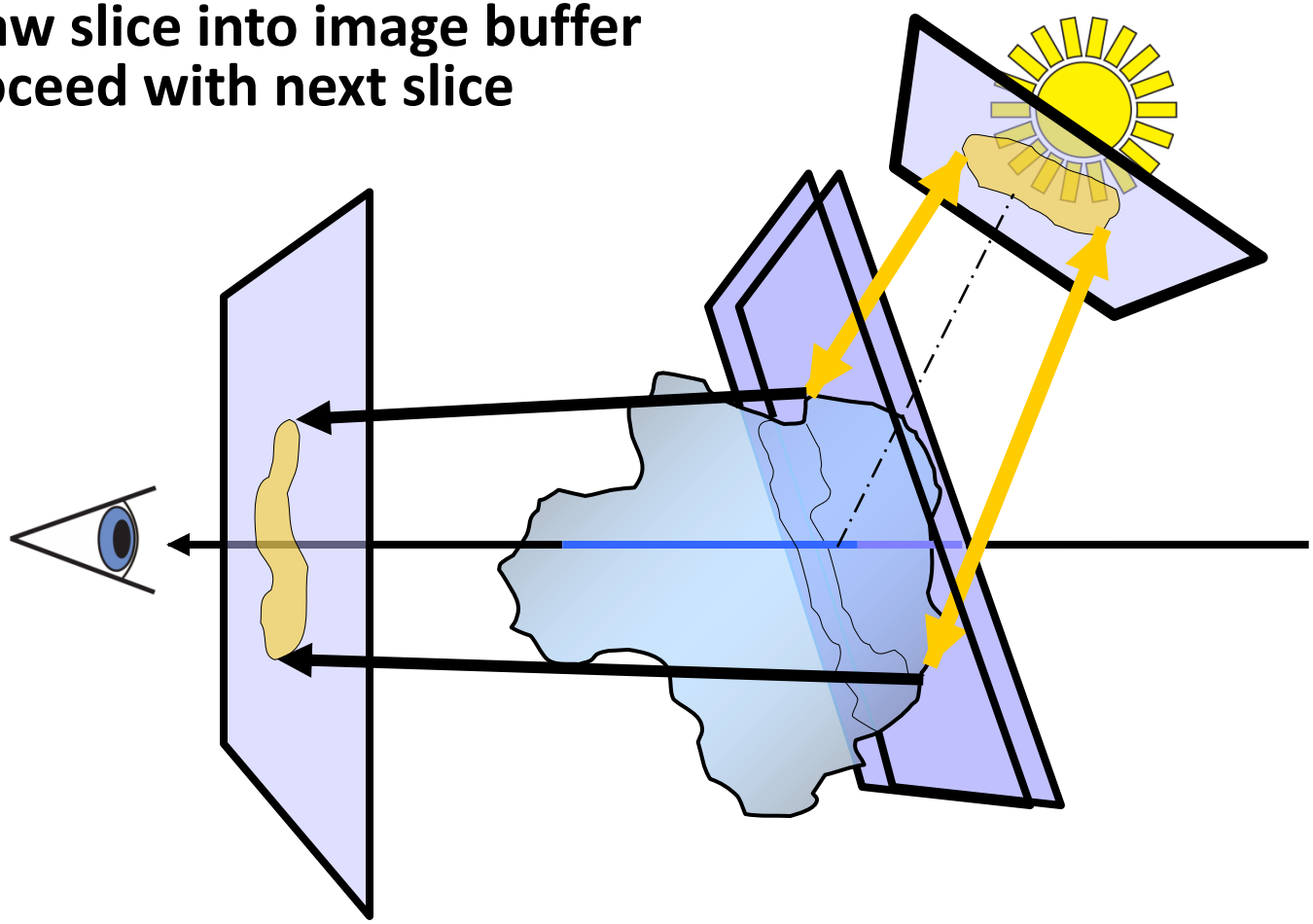
Update image and shadow buffer slice-by-slice

- Need proxy geometry, that can be rendered from two different views



Light Source Attenuation

1. Draw slice into light buffer
2. Draw slice into image buffer
3. Proceed with next slice



Light Source Attenuation: Example

surface-based shading

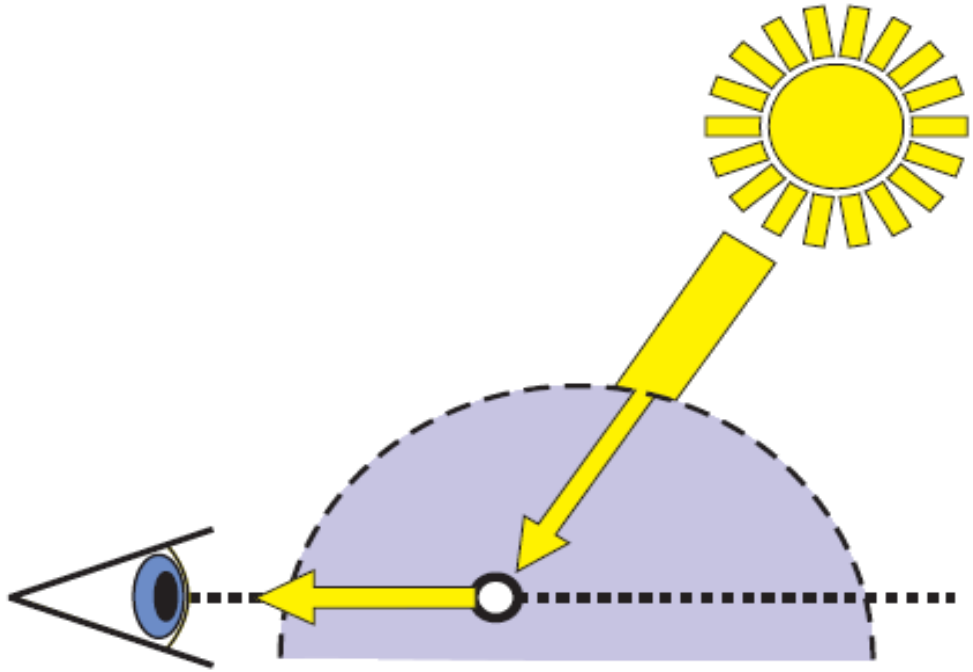
light attenuation

Volume Illumination – Scattering (1)

Previously: Attenuation of light by the volume

Single scattering with attenuation

- Light is attenuated along its way through the volume
- Light is scattered once before it reaches the eye

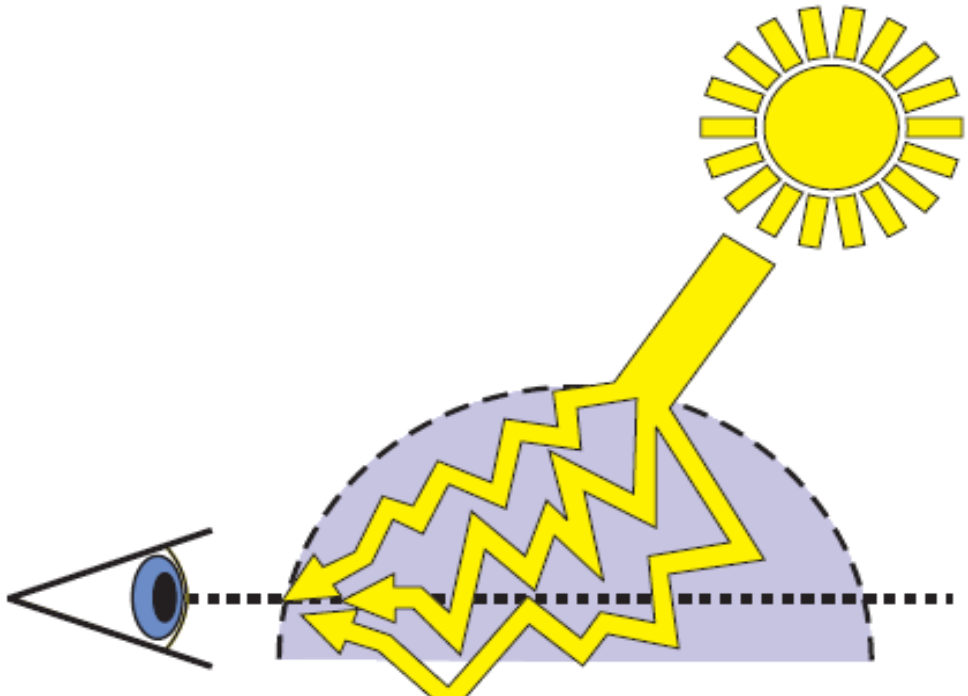


Volume Illumination – Scattering (2)

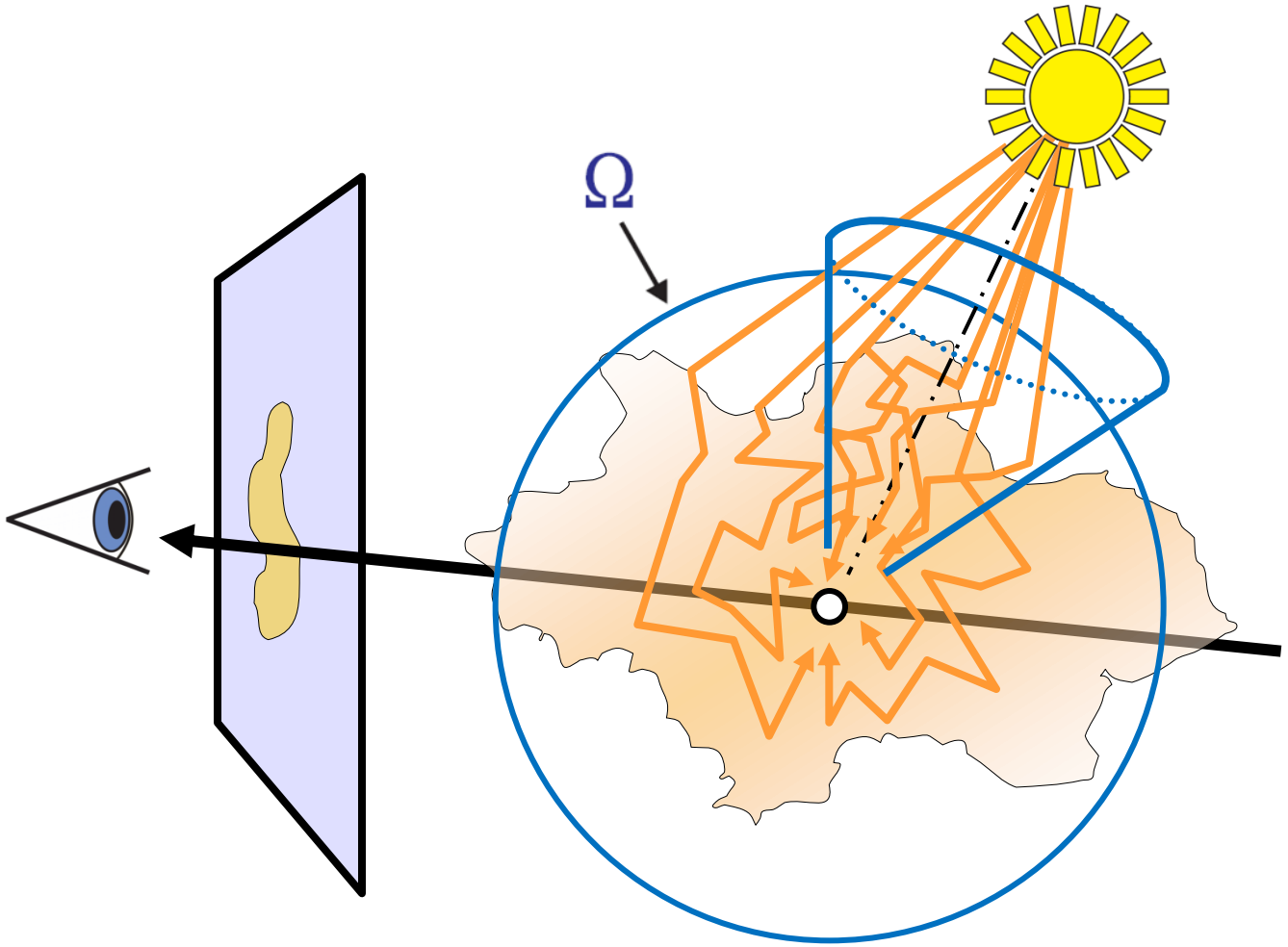
Now: Light is scattered inside the volume

Multiple scattering

- Light is scattered multiple times before it reaches the eye



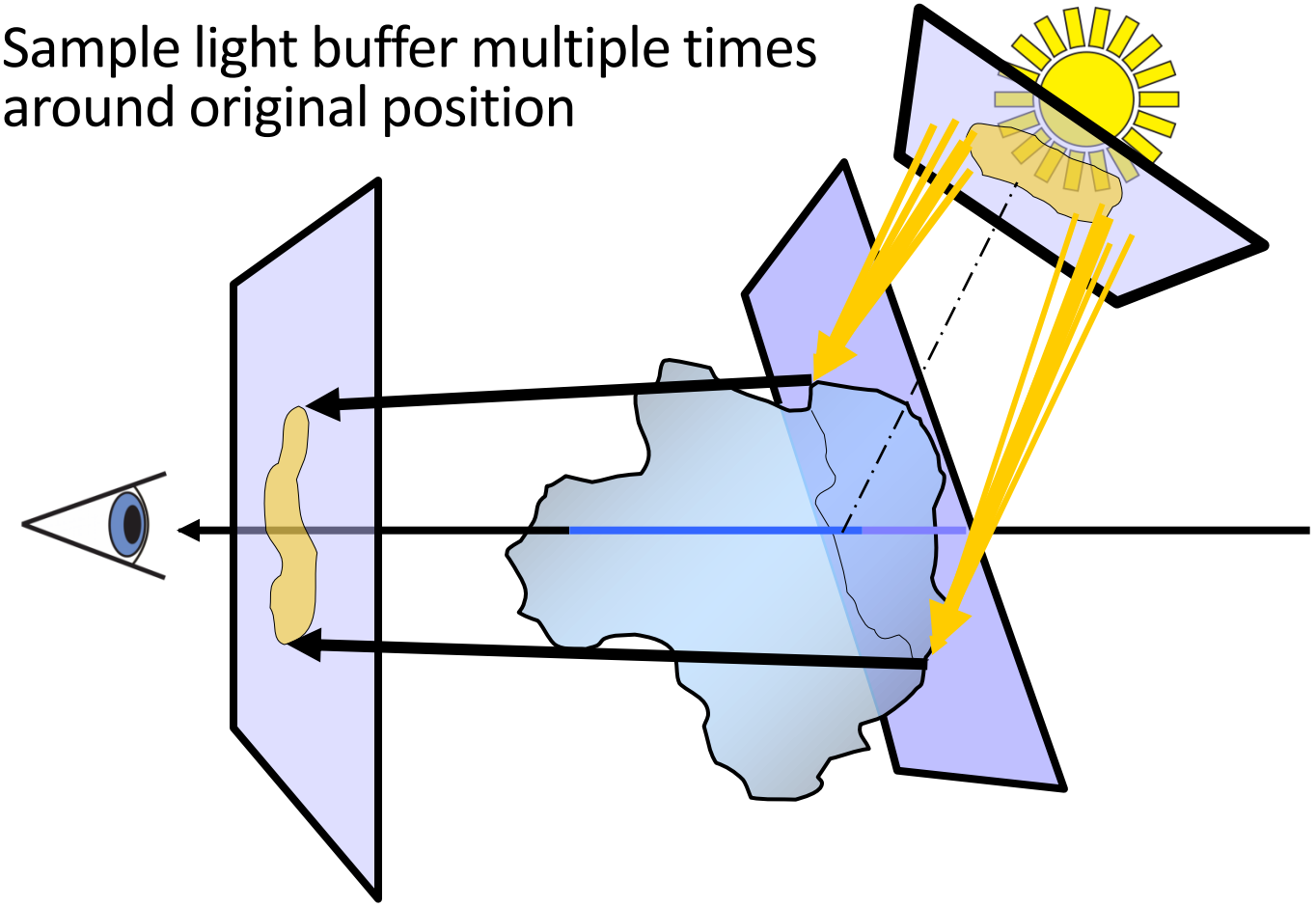
Volumetric Scattering (1)



Volumetric Scattering (2)

Draw slice into image buffer

- Sample light buffer multiple times around original position

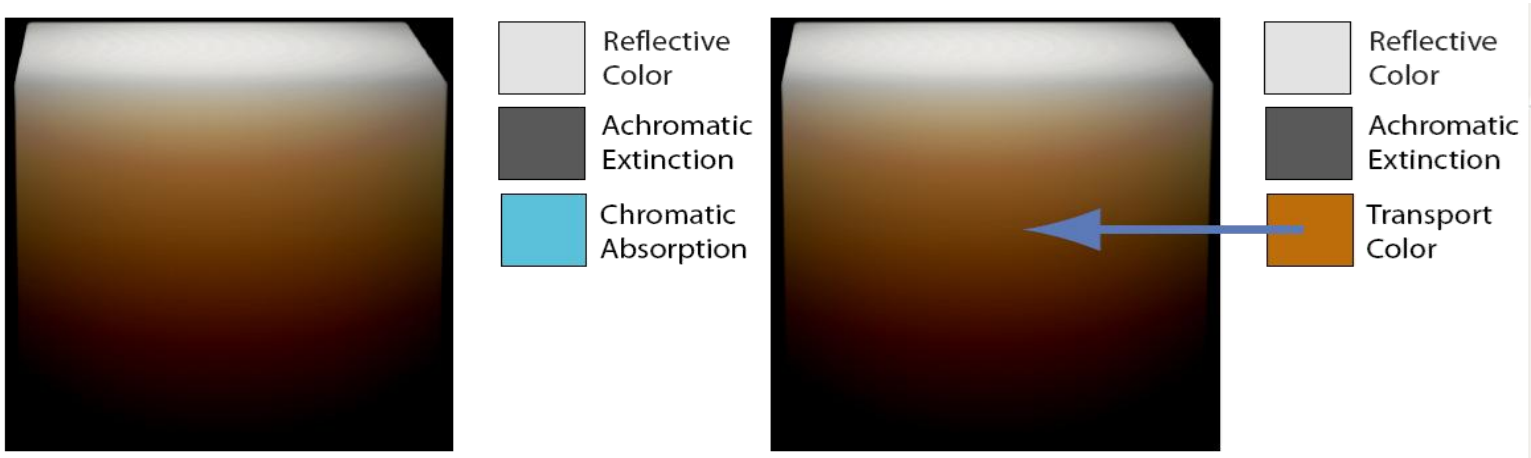


Volumetric Scattering (3)

Scattering part:

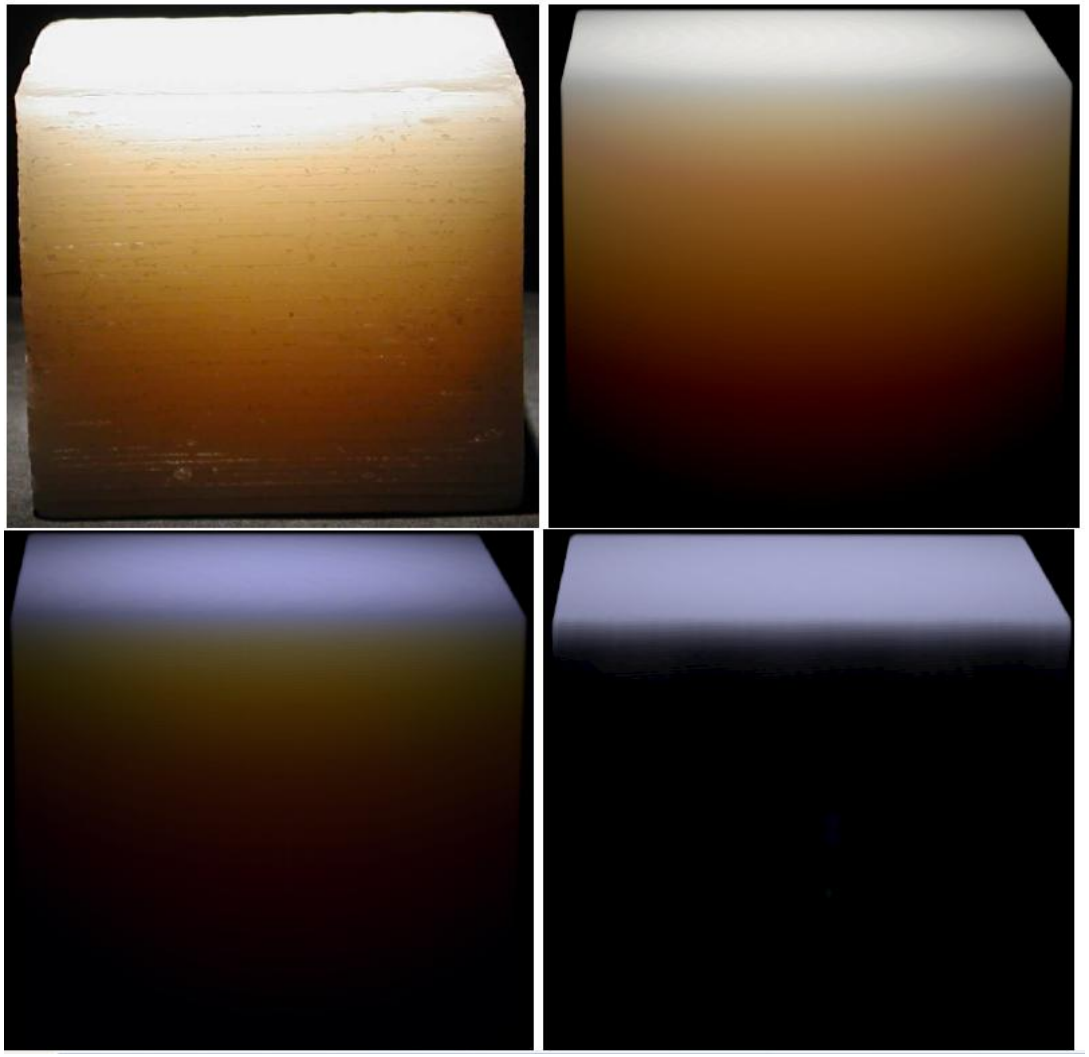
$$j(\mathbf{x},\omega) = \frac{1}{4\pi}\int_{\text{sphere}} \sigma(\mathbf{x},\omega')p(\mathbf{x},\omega',\omega)I(\mathbf{x},\omega')\mathrm{d}\omega'$$

Chromatic out-scattering term $\sigma(\mathbf{x},\omega')$ can be used to change color of light as it travels throught the volume



Volumetric Scattering (4)

Photograph of real wax block



Volumetric Scattering + Chromatic Attenuation

Bright blue reflective color

Direct attenuation only

Phase Function Examples

Henyey-Greenstein

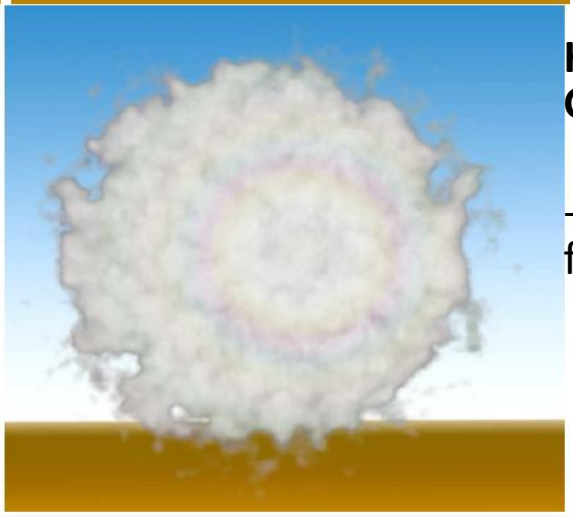
Lit from behind

Henyey-Greenstein

Lit from front

Henyey-Greenstein

Lit 45 degrees from above



Henyey-Greenstein

+ Mie Phase function

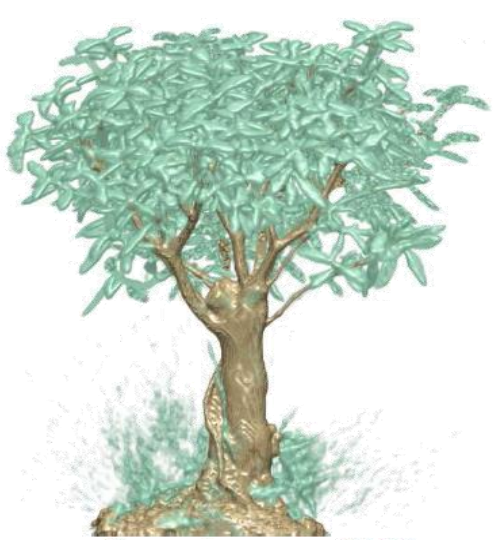
Examples (1)

Examples (2)

Direct light only | Achromatic indirect light | Chromatic indirect light

Examples (3)

Surface Shading



Direct Light + Shadows

Direct Light+ Indirect Light

Direct Light+ Indirect Light

Surface shading on the leaves only

Alternative Approach (1)

Kniss et al.'s proposed solution: render each slice twice

1. Project slice into view buffer
2. Project slice into light buffer

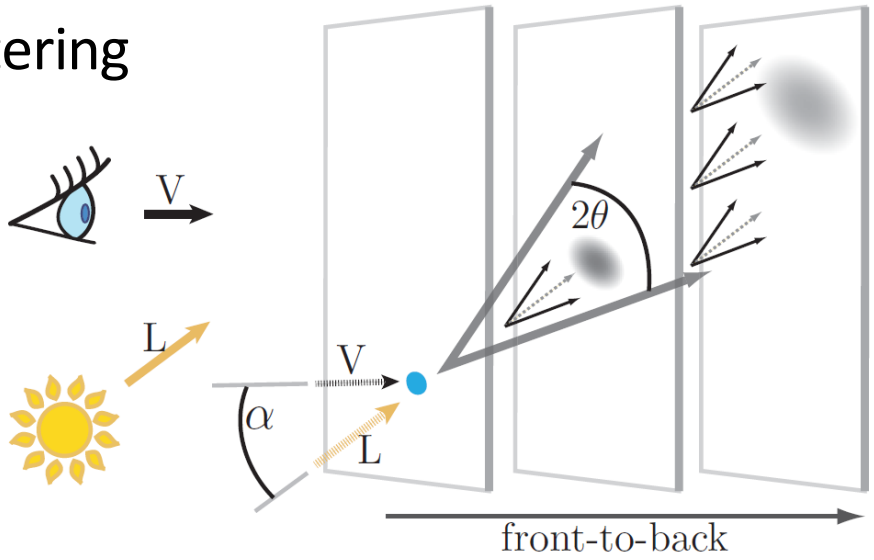
The same volume locations are sampled in both cases, but projected into different images

Can be expensive: volume sampling, transfer function lookup, etc.

Alternative Approach (2)

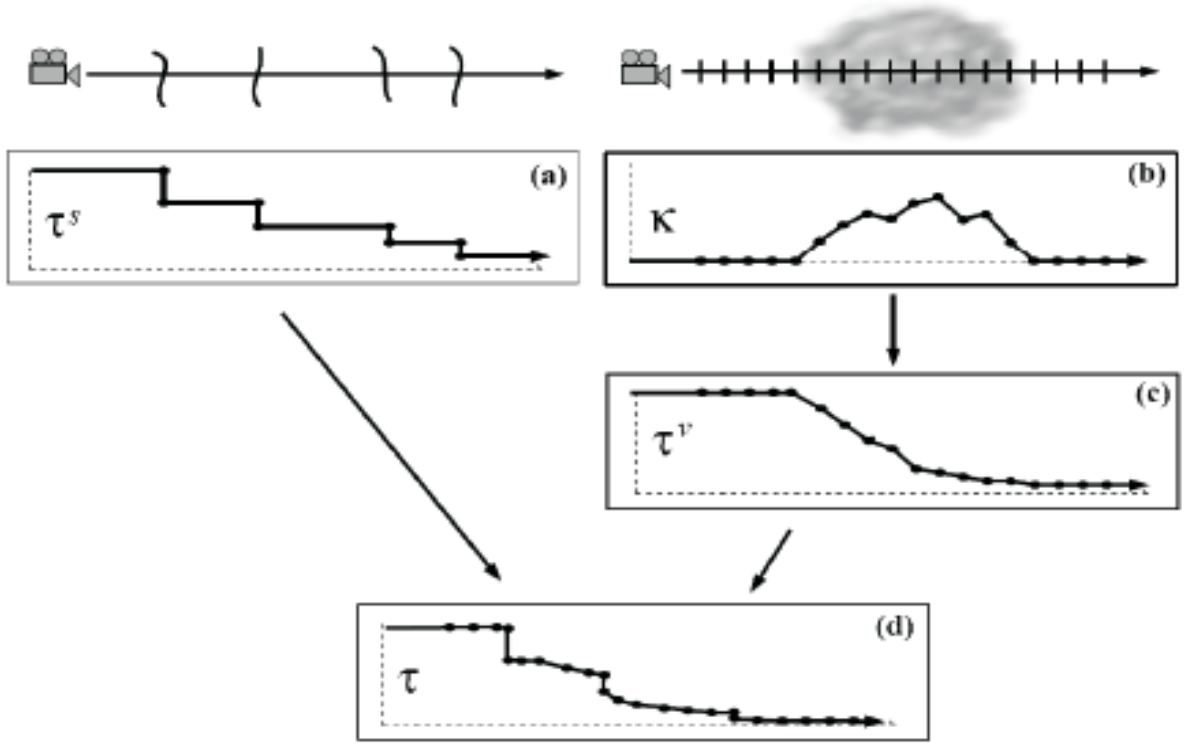
If we limit light source to the eye plane (i.e., angle < 90 degrees), we can do it in one pass [Šoltészova et al. 2010]

- Volume, transfer function, etc. only seed to be sampled once for every sice
- Directly sample previous slice using an offset corresponding to light direction, light buffer as a second render target
- Incremental convolution for scattering



Deep Shadow Maps

Linked list of visibility function for each pixel in the shadow map

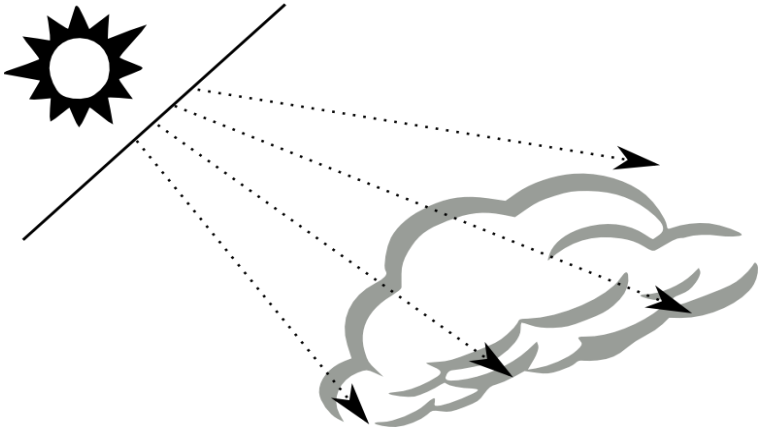


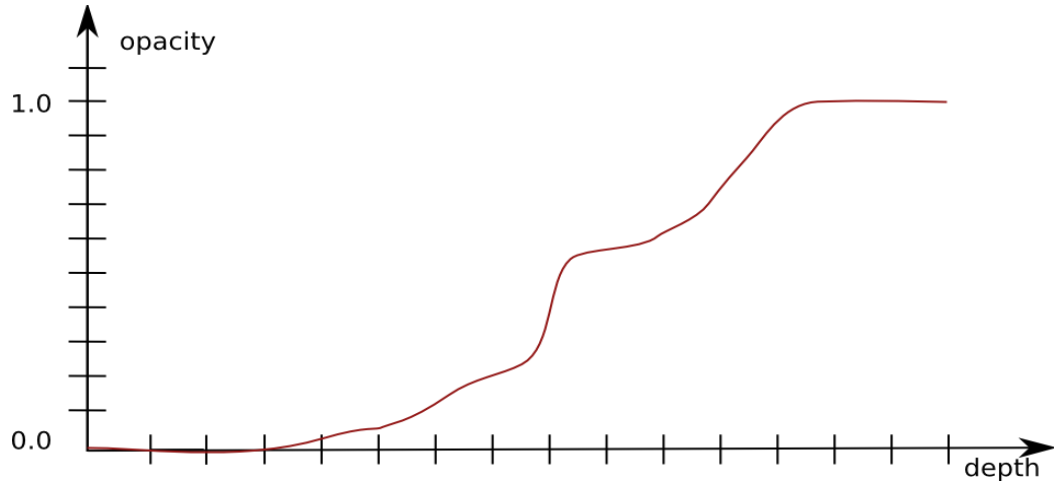
GPU Deep Shadow Maps for Volumes

GPU ray-casting from light source

Compress on the fly, store “list” in 3D texture

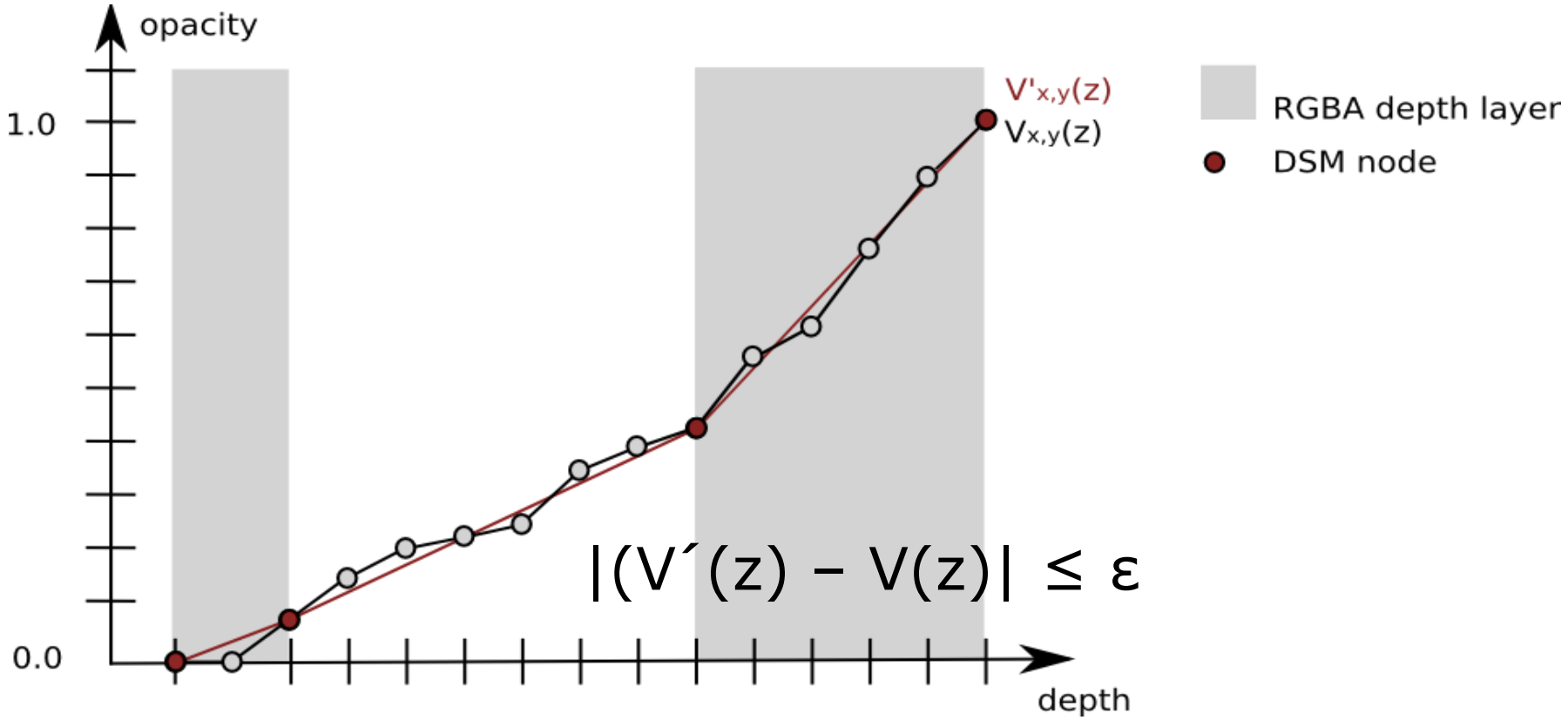
Attenuation function: Vx,y(z) = opacity



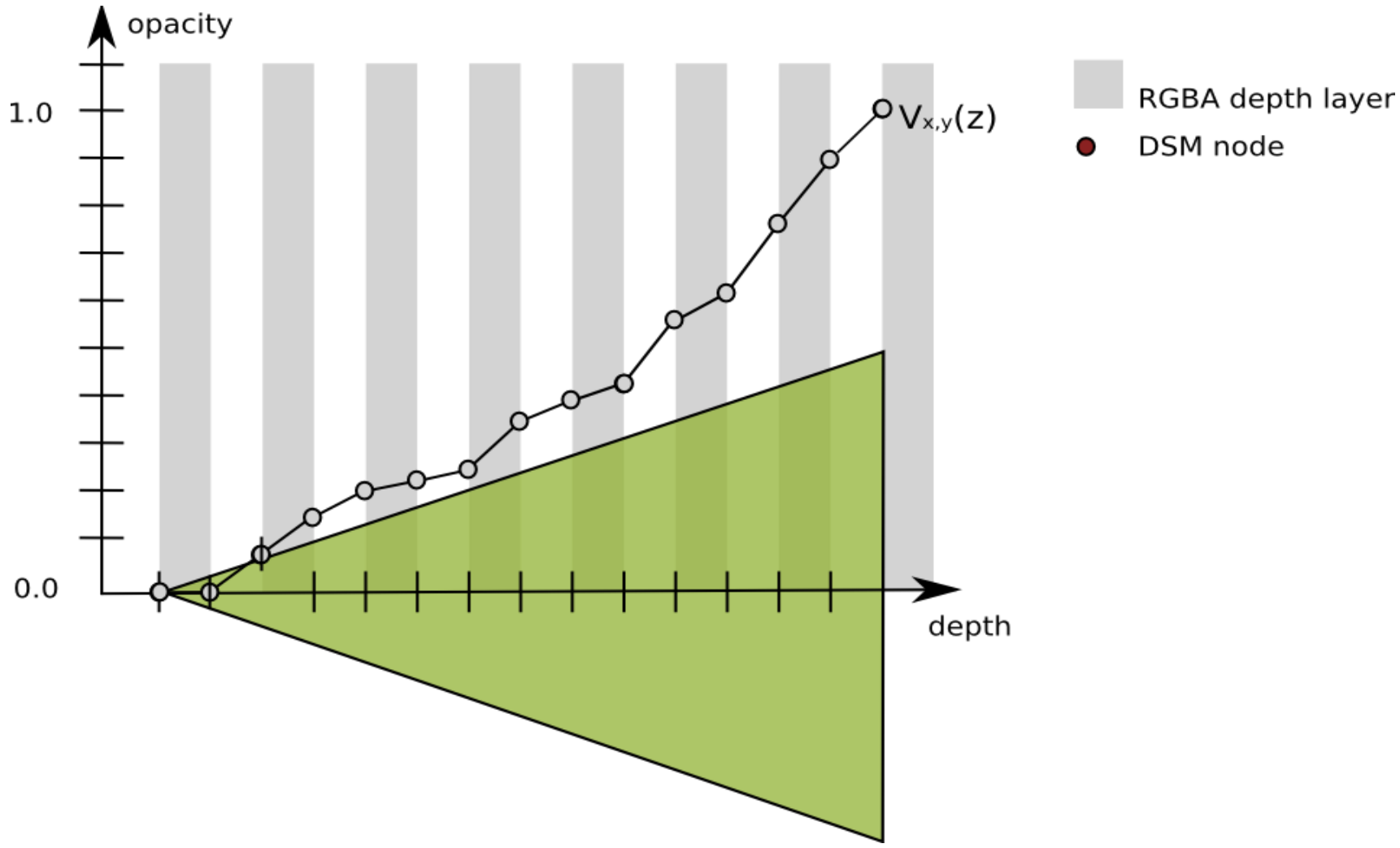


DSM Compression (1)

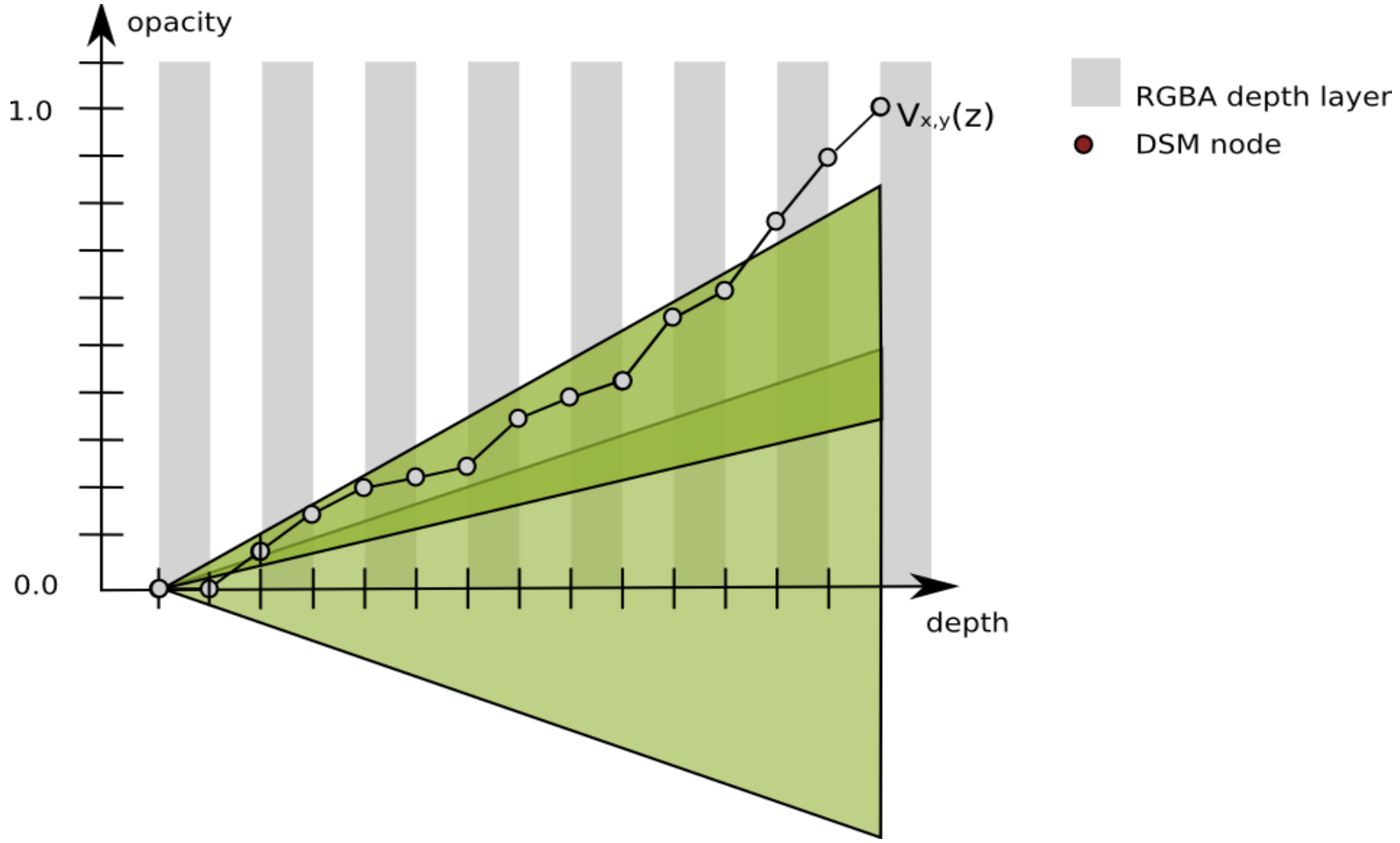
- Look at each successive sample;
  compare with linear interpolation of running segment



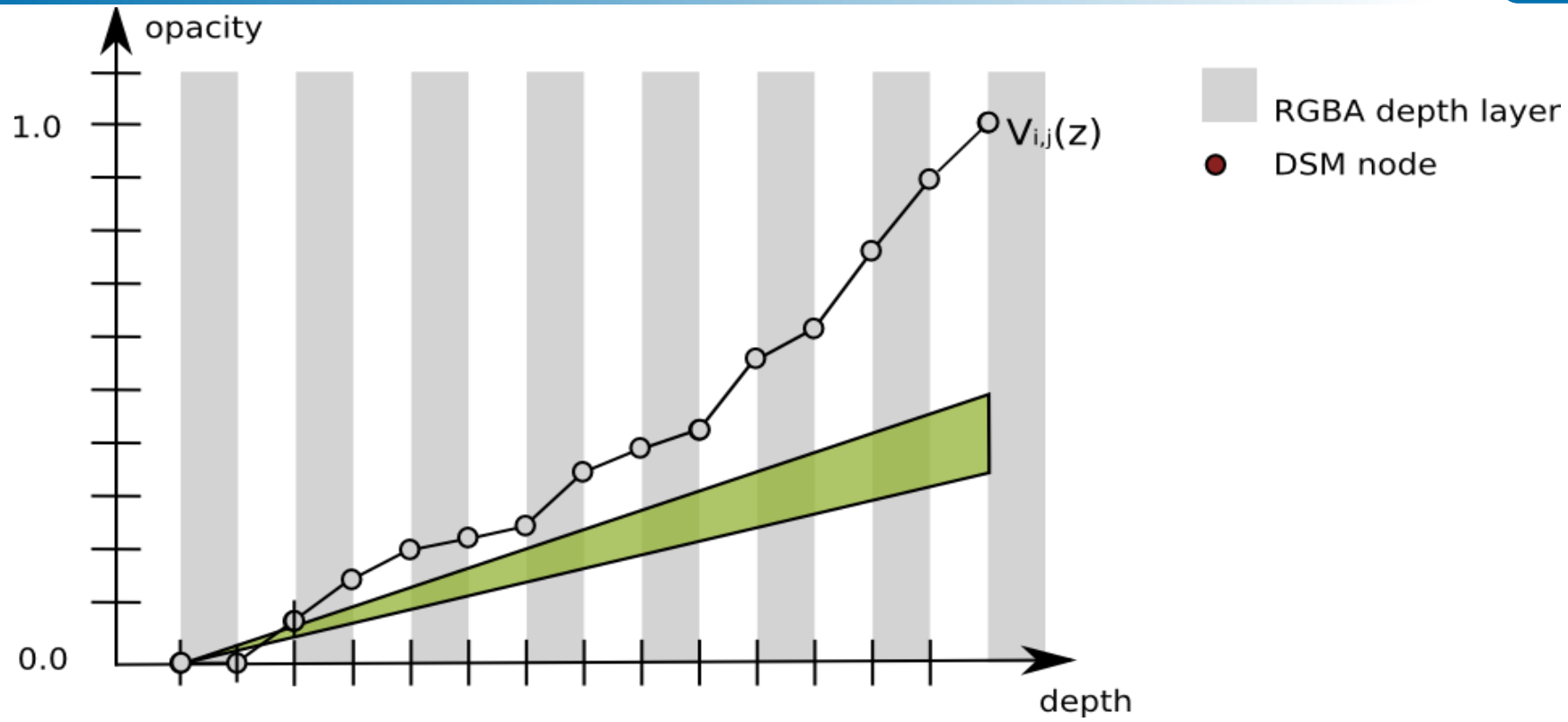
DSM Compression (2)



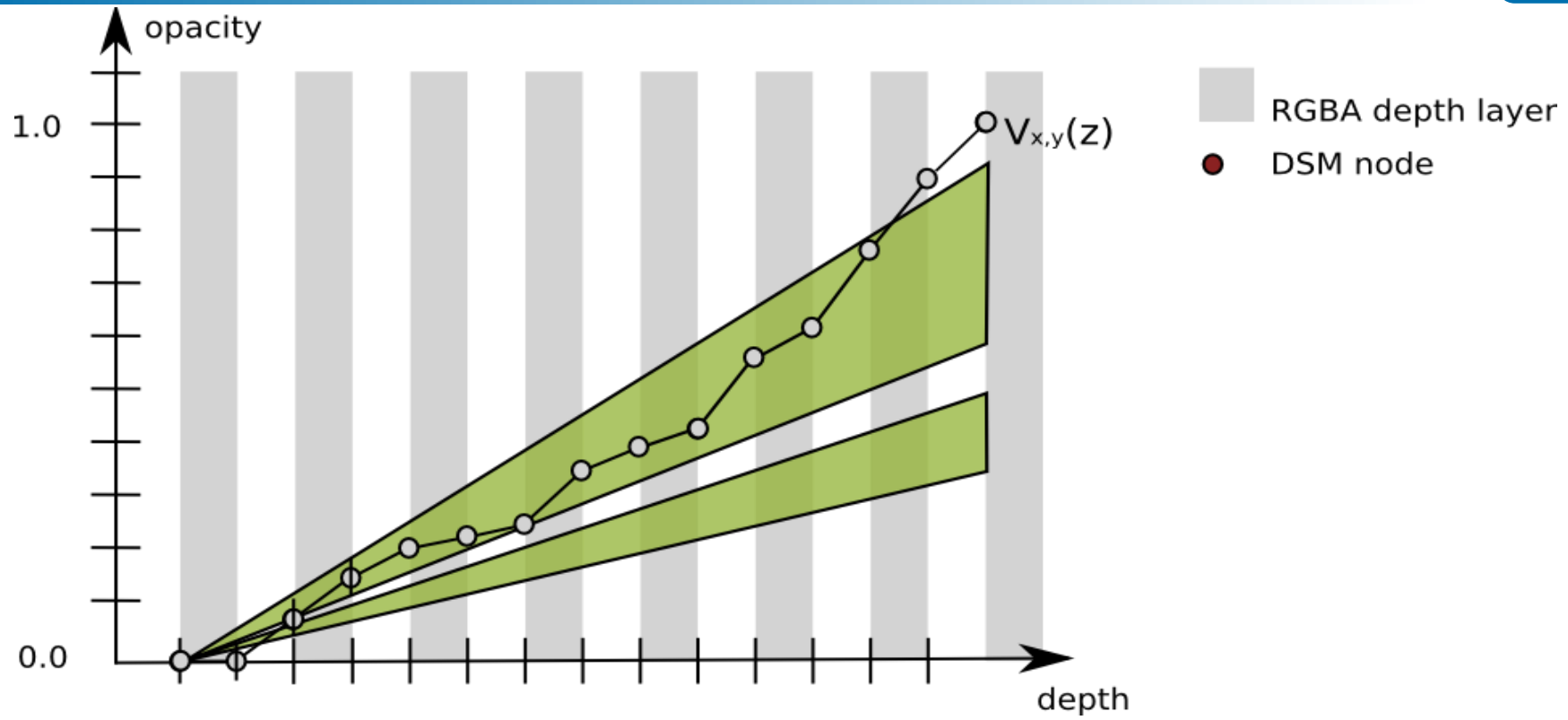
DSM Compression (3)



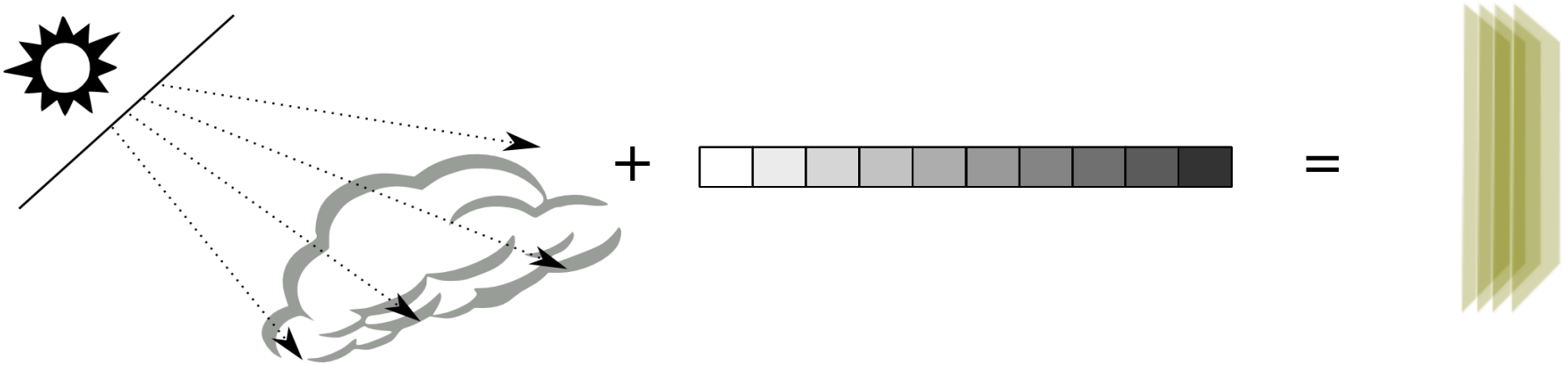
DSM Compression (4)

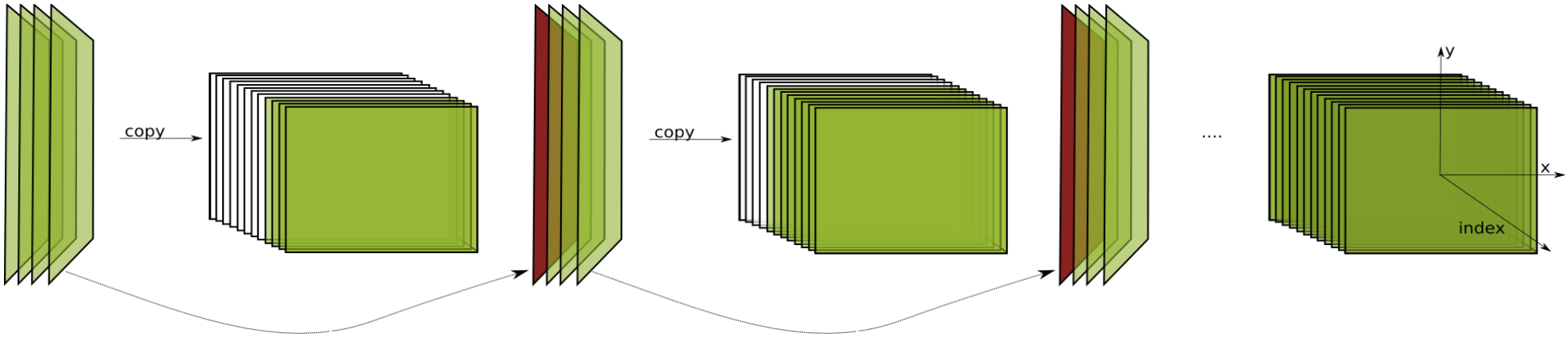


DSM Compression (5)



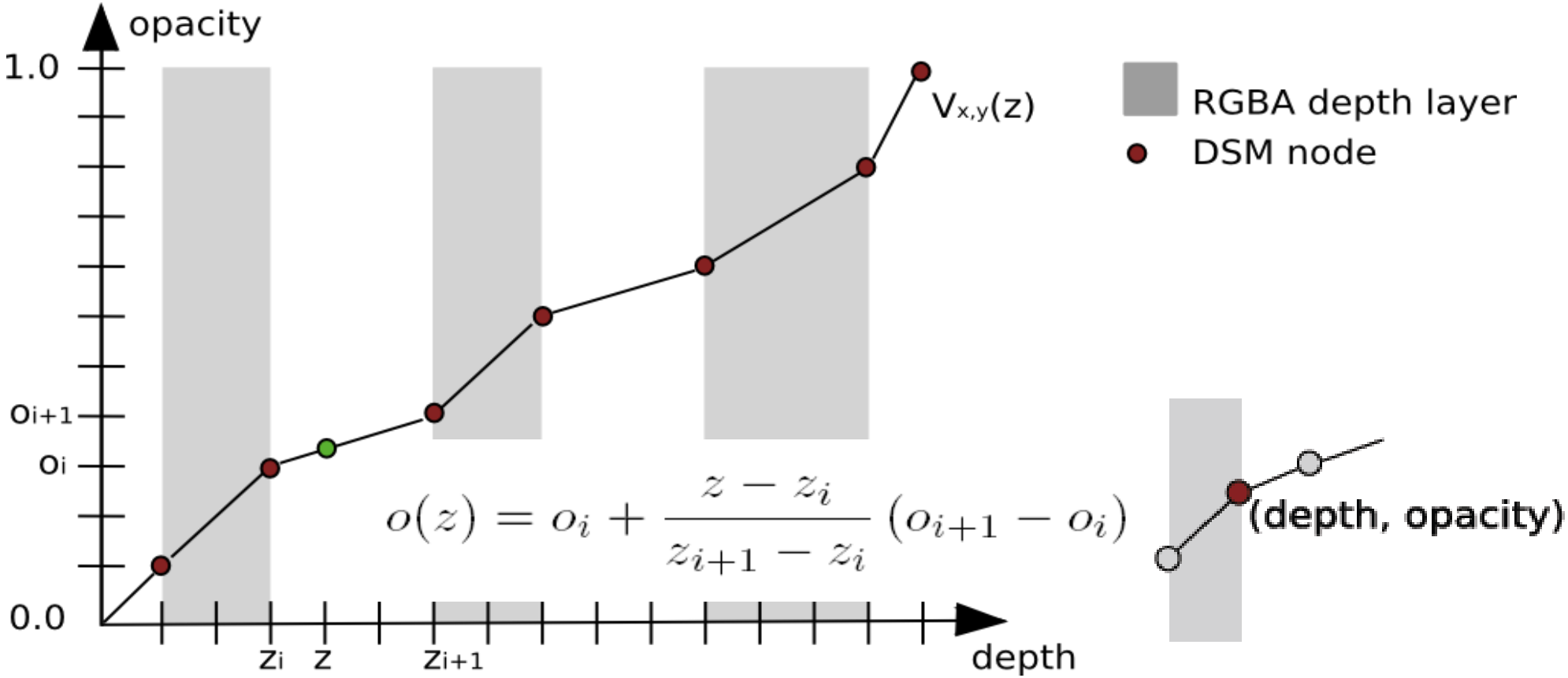
GPU DSM Construction





DSM Rendering

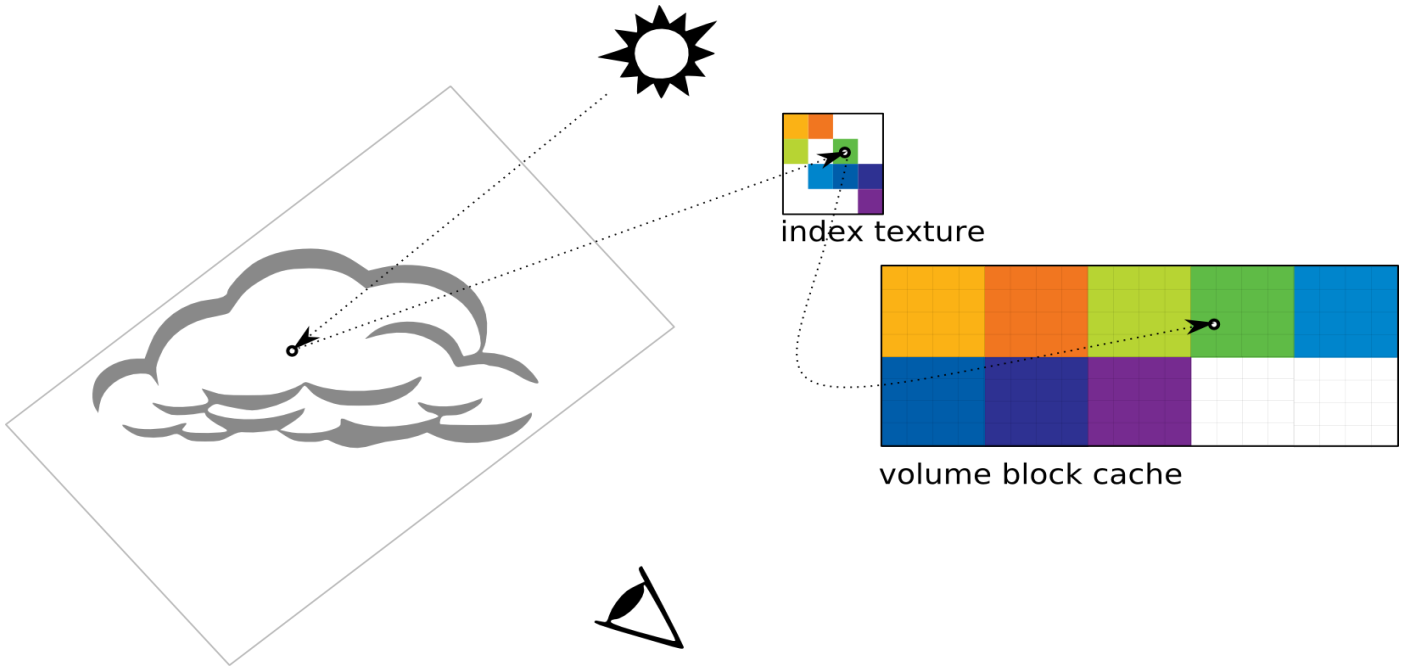
Need search + lerp in DSM for each sample



DSM Memory Management (1)

DSM Construction

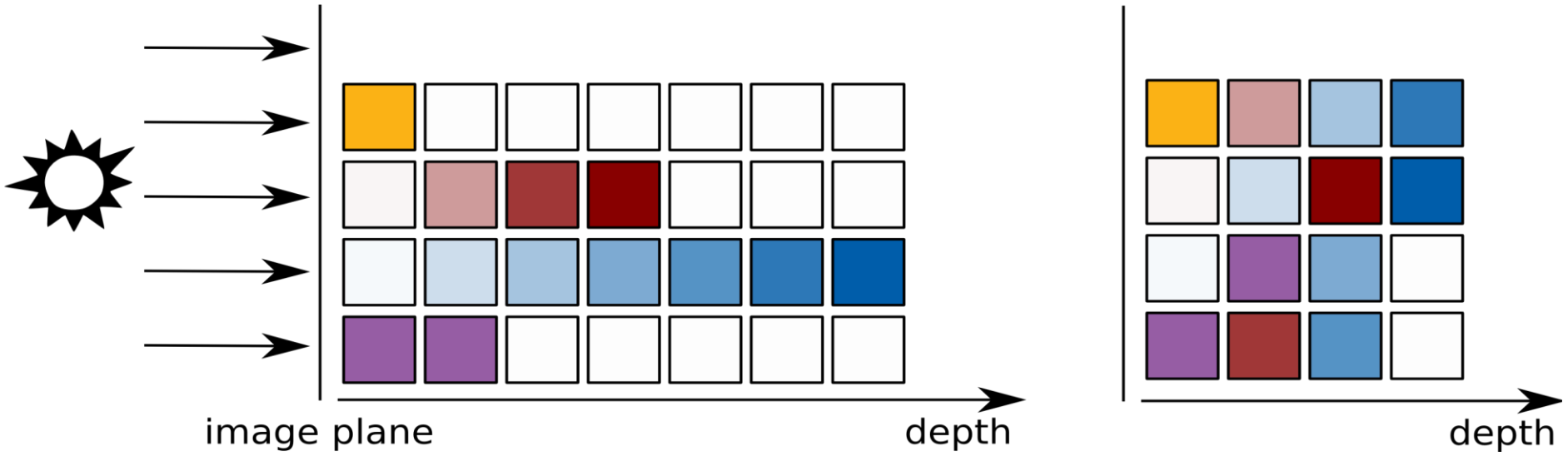
- Bricking scheme for volume (density)
- Address translation (index) texture



DSM Memory Management (2)

DSM Construction

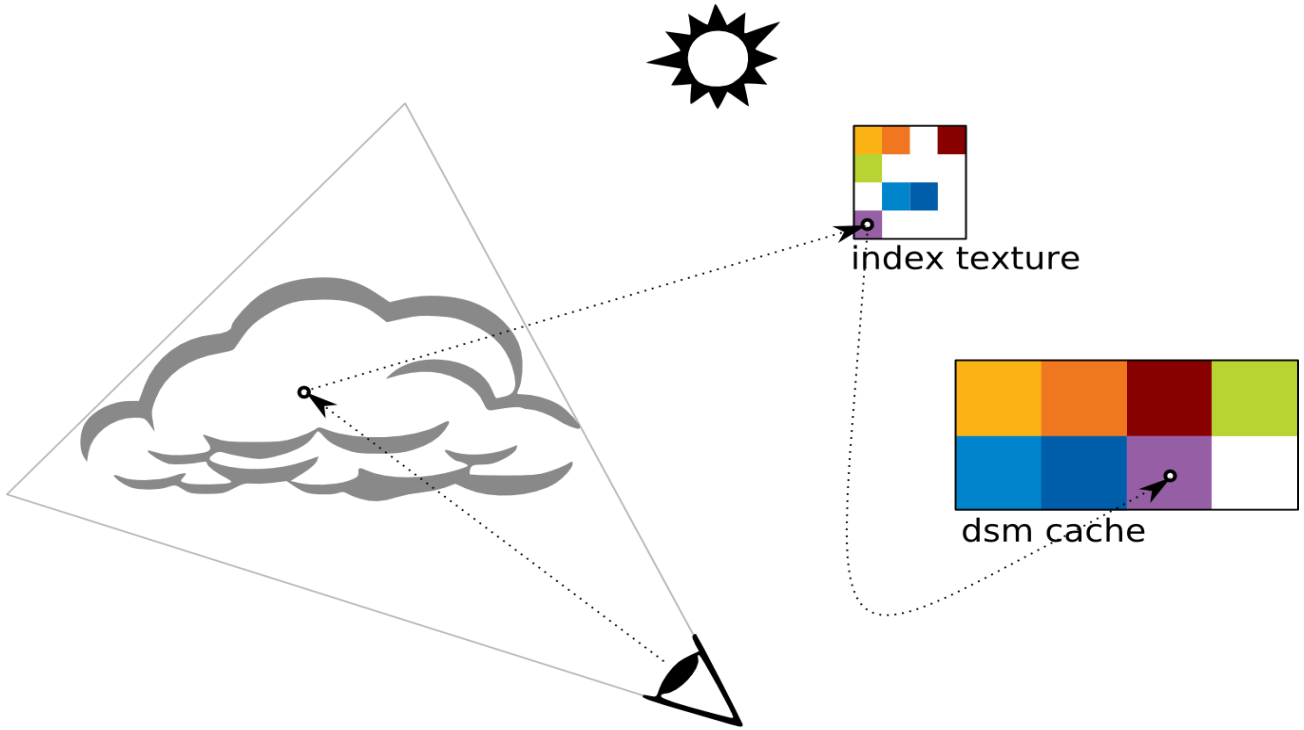
- Optionally pack DSM node layers



DSM Memory Management (3)

DSM Rendering

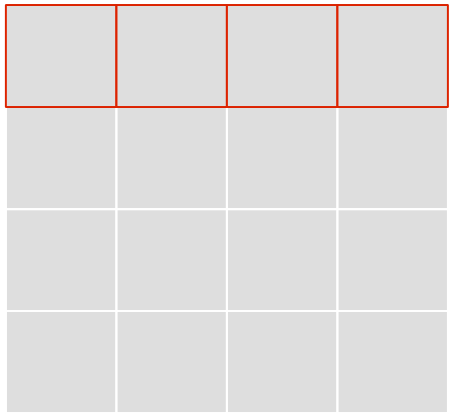
- Access second index texture for DSM

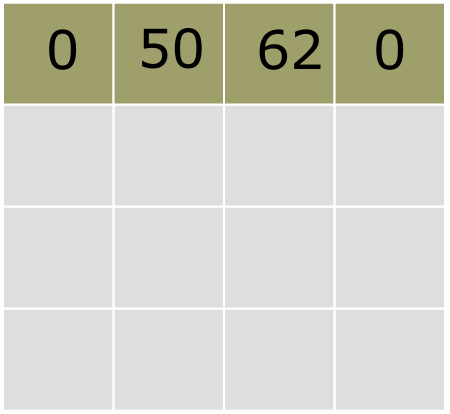


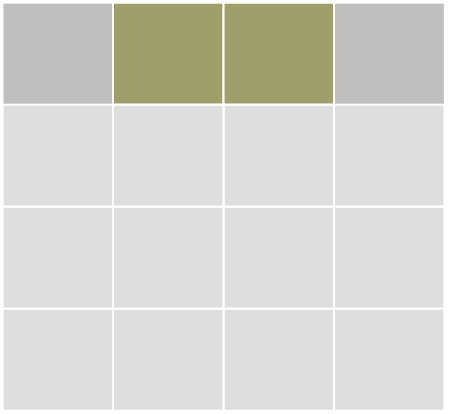
DSM Tiling

Construct DSM tile by tile in shadow map plane

- Occlusion query determines whether to stop depth progression
- On-demand allocation of bricks (memory)

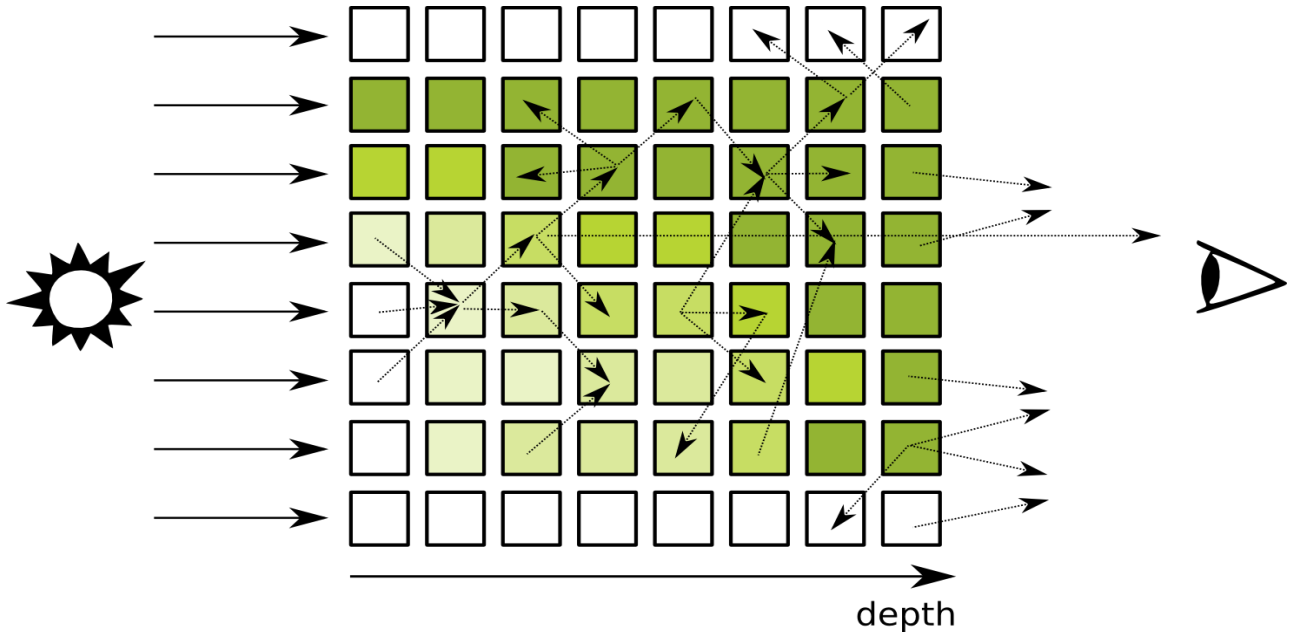






Scattering (1)

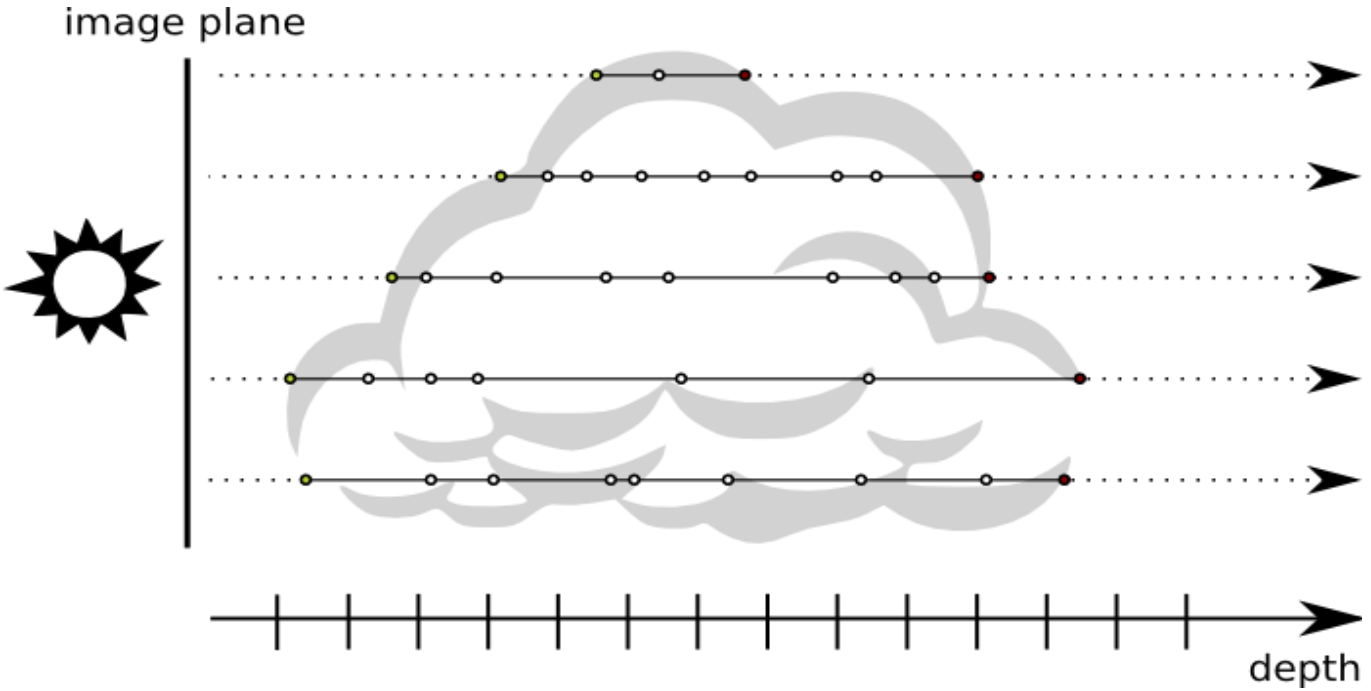
Each sample in the volume potentially scatters to all other samples



Scattering (2)

Real-time forward scattering approximations need access to light at previous coherent depth

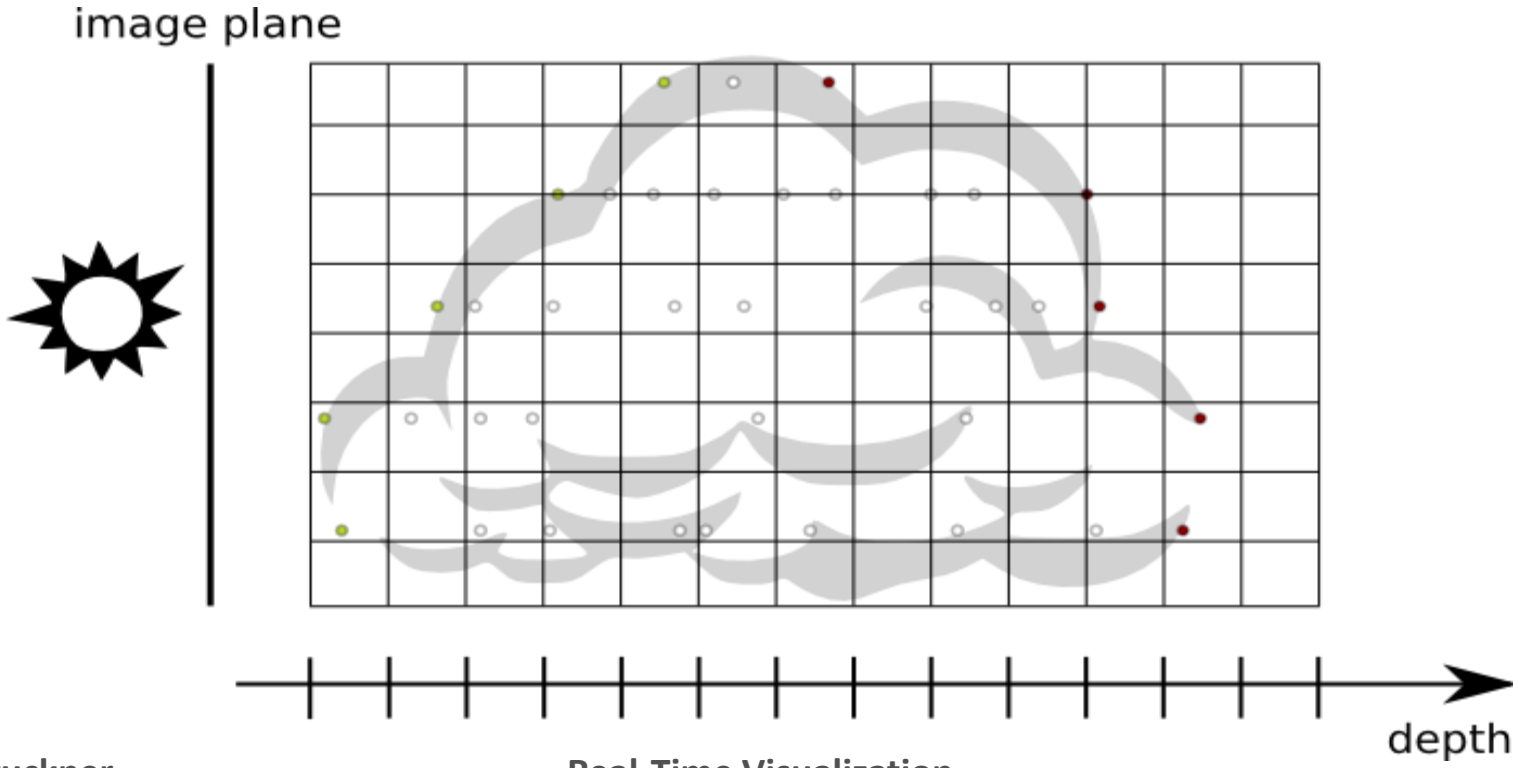
- Problem: DSM is sampled irregularly



Scatter Volume

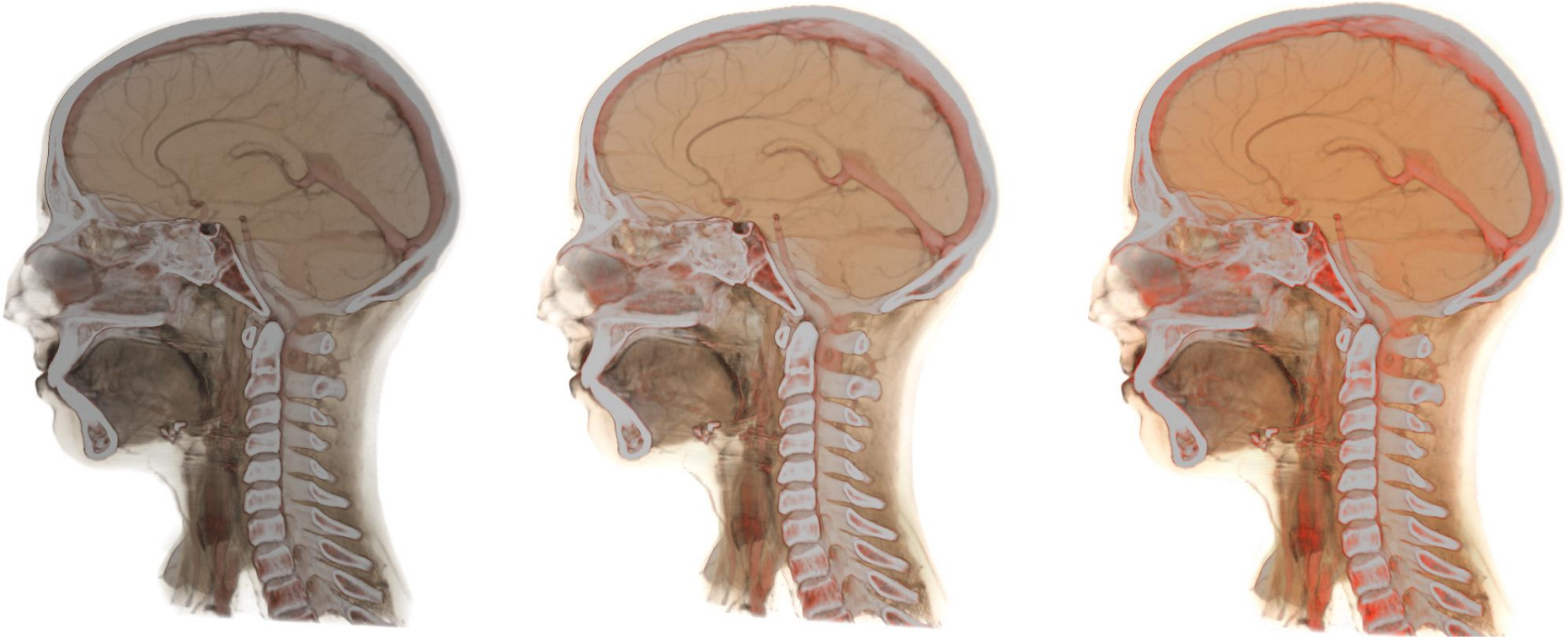
Low-resolution regular representation of DSM just for indirect lighting contribution

- Two shadow/scattering construction passes

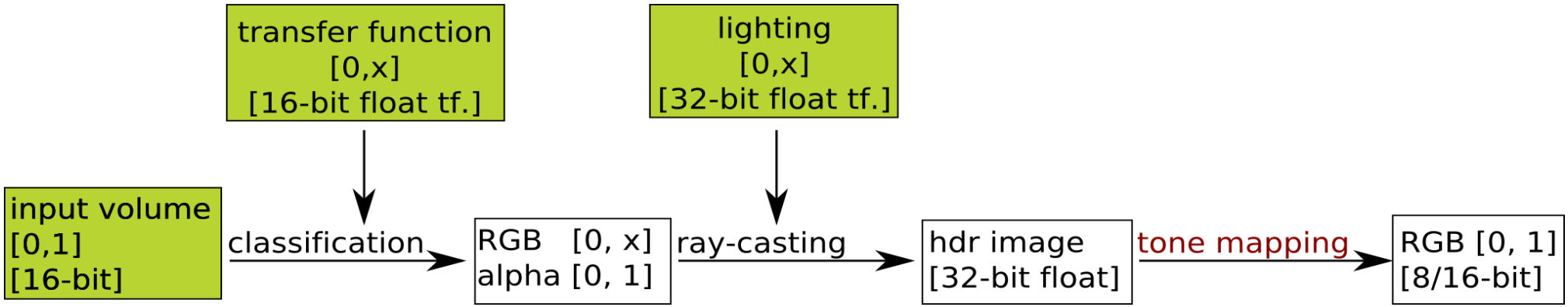


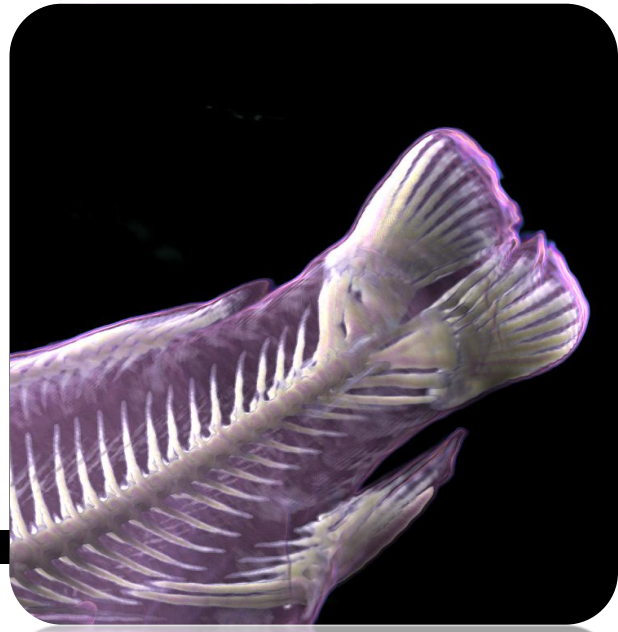
HDR Volume Rendering (1)

Ray-casting in HDR plus tone mapping pass



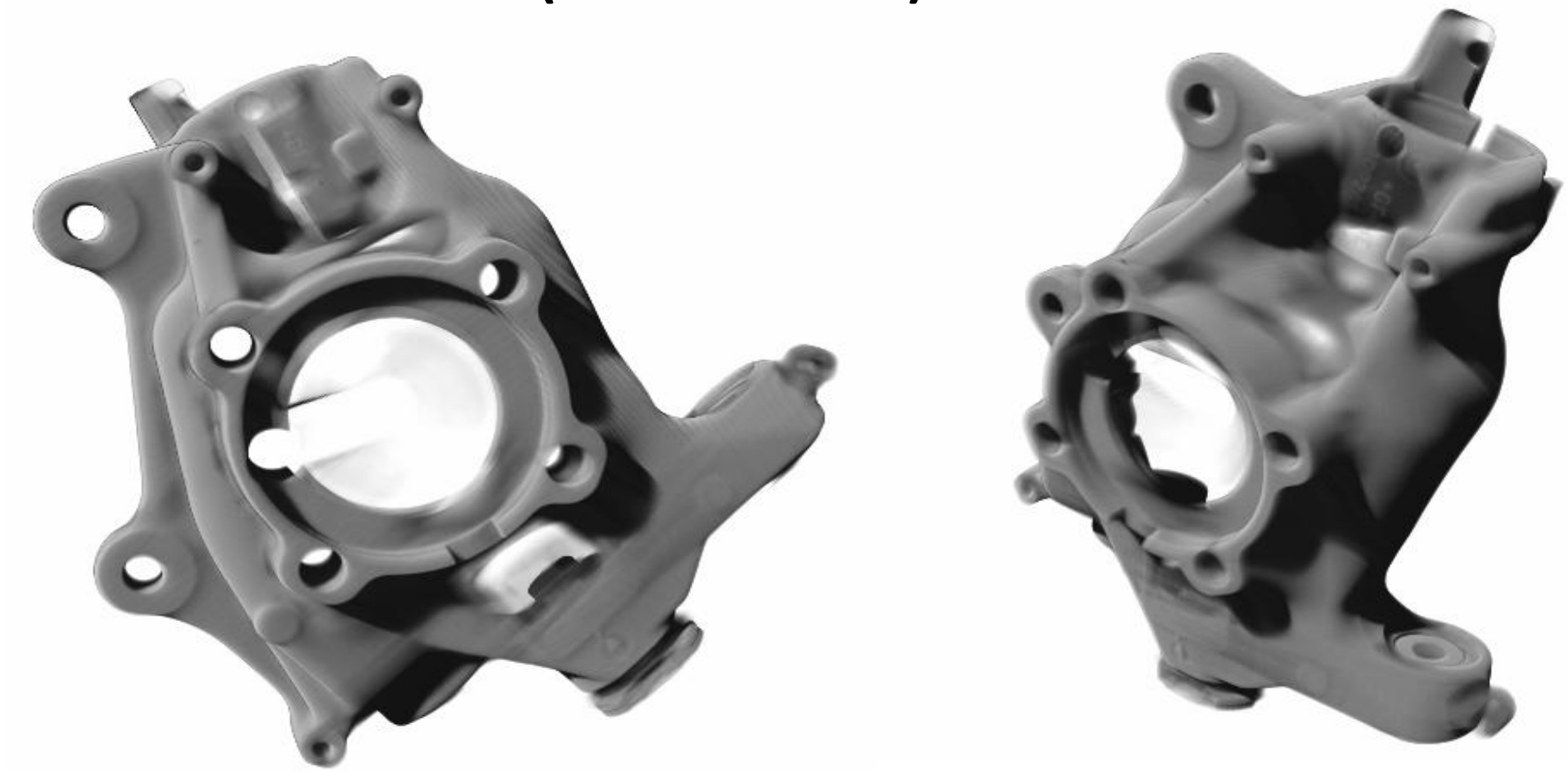
HDR Volume Rendering (2)





More Results (1)

Industrial CT scan (300x425x512)

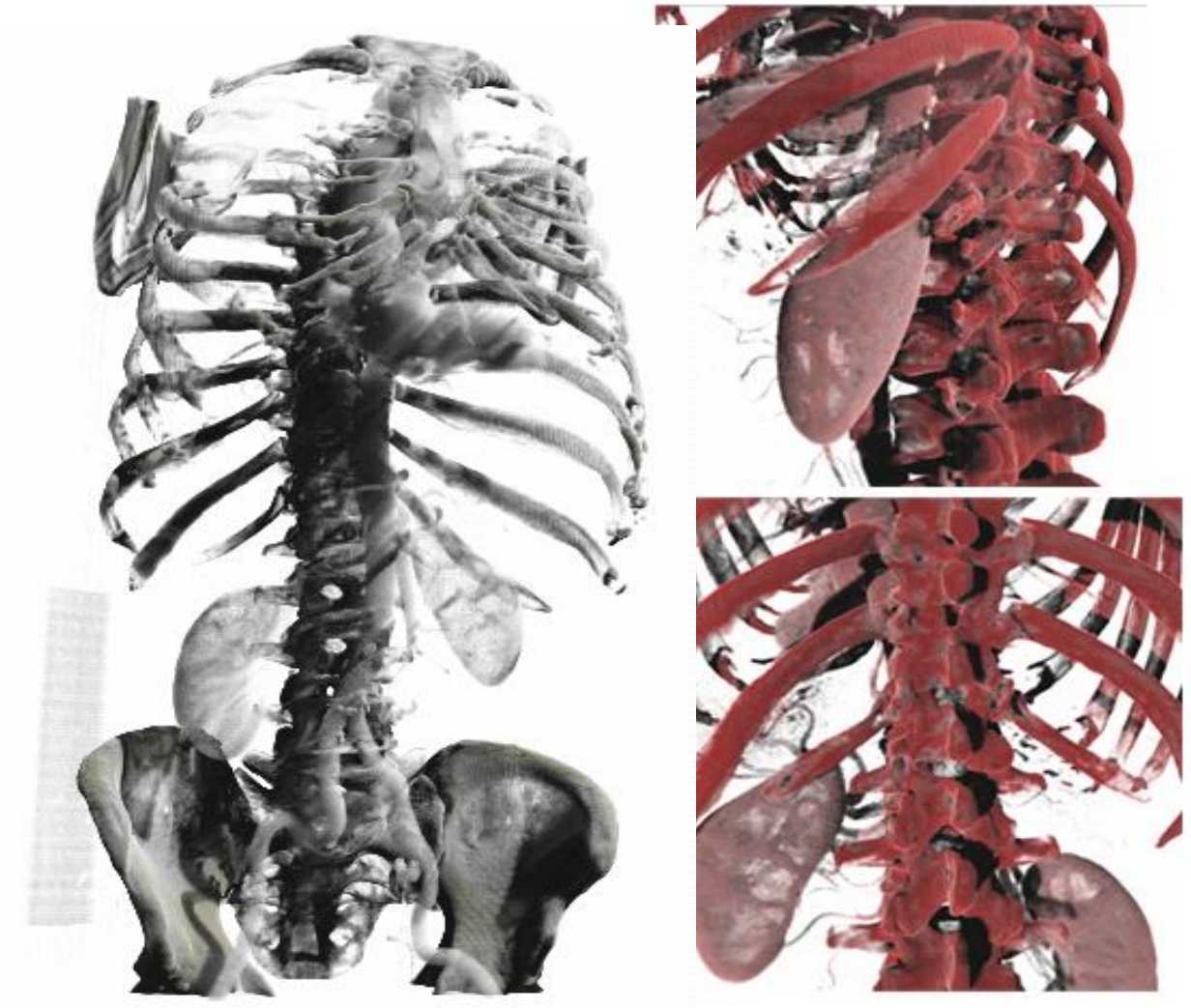


More Results (2)

Human torso (512x512x1112)

Performance on Geforce 8

- Rendering: 6 - 8 fps
- Move light: 2 - 3 fps



Thank You!

- Markus Hadwiger
- Christof Rezk-Salama
- Henning Scharsach
- Andrea Kratz
- Gordon Kindlmann
- Enrico Gobbetti et al.