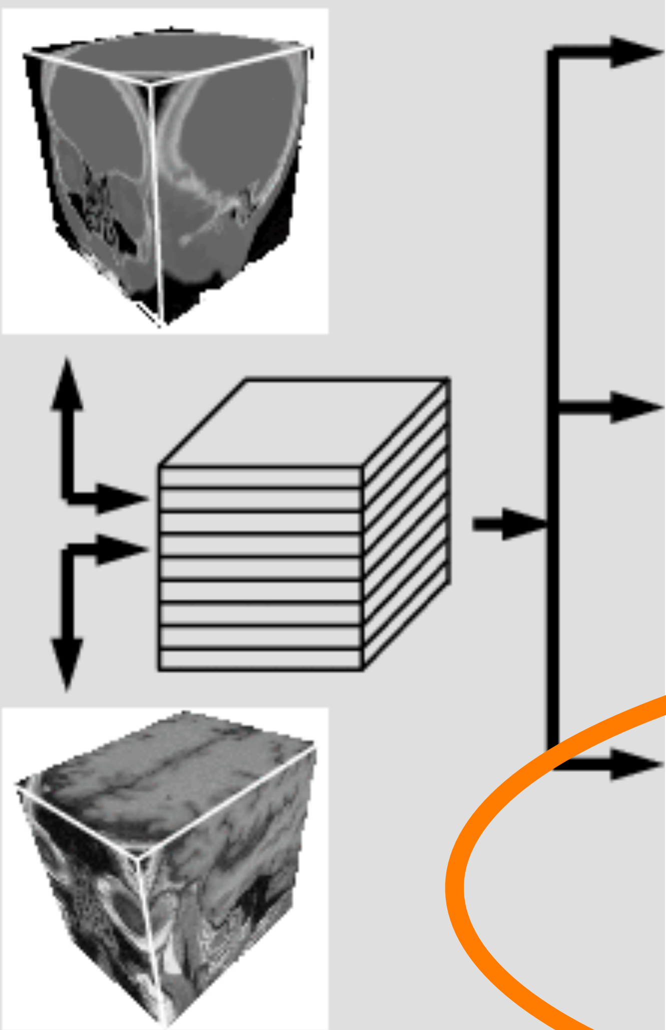
Direct Volume Rendering

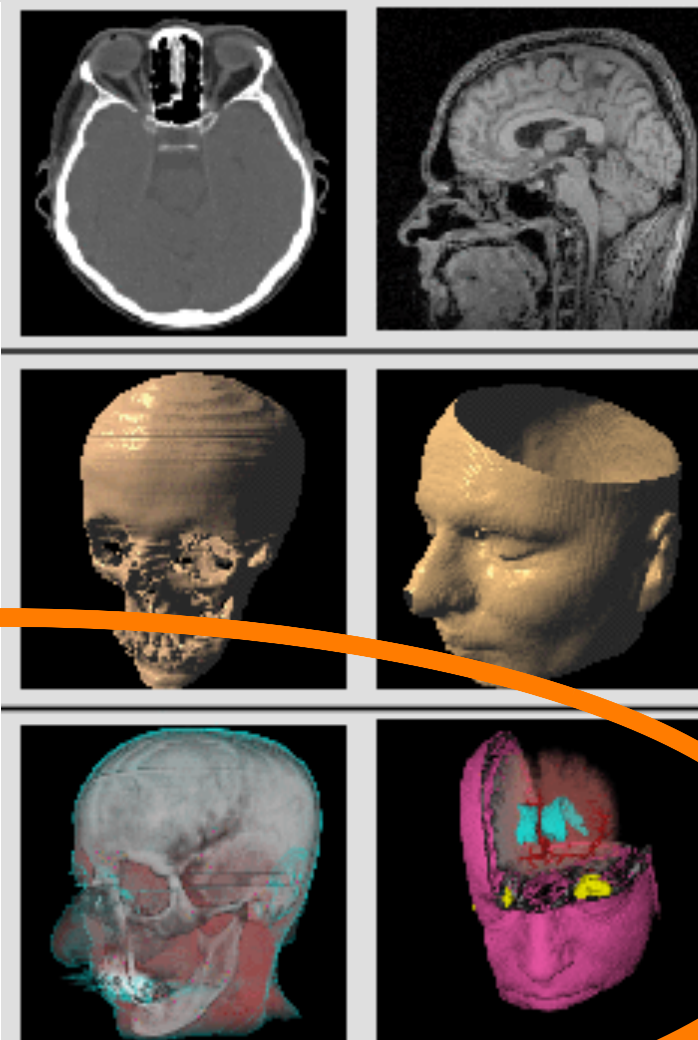
3D Image Processing

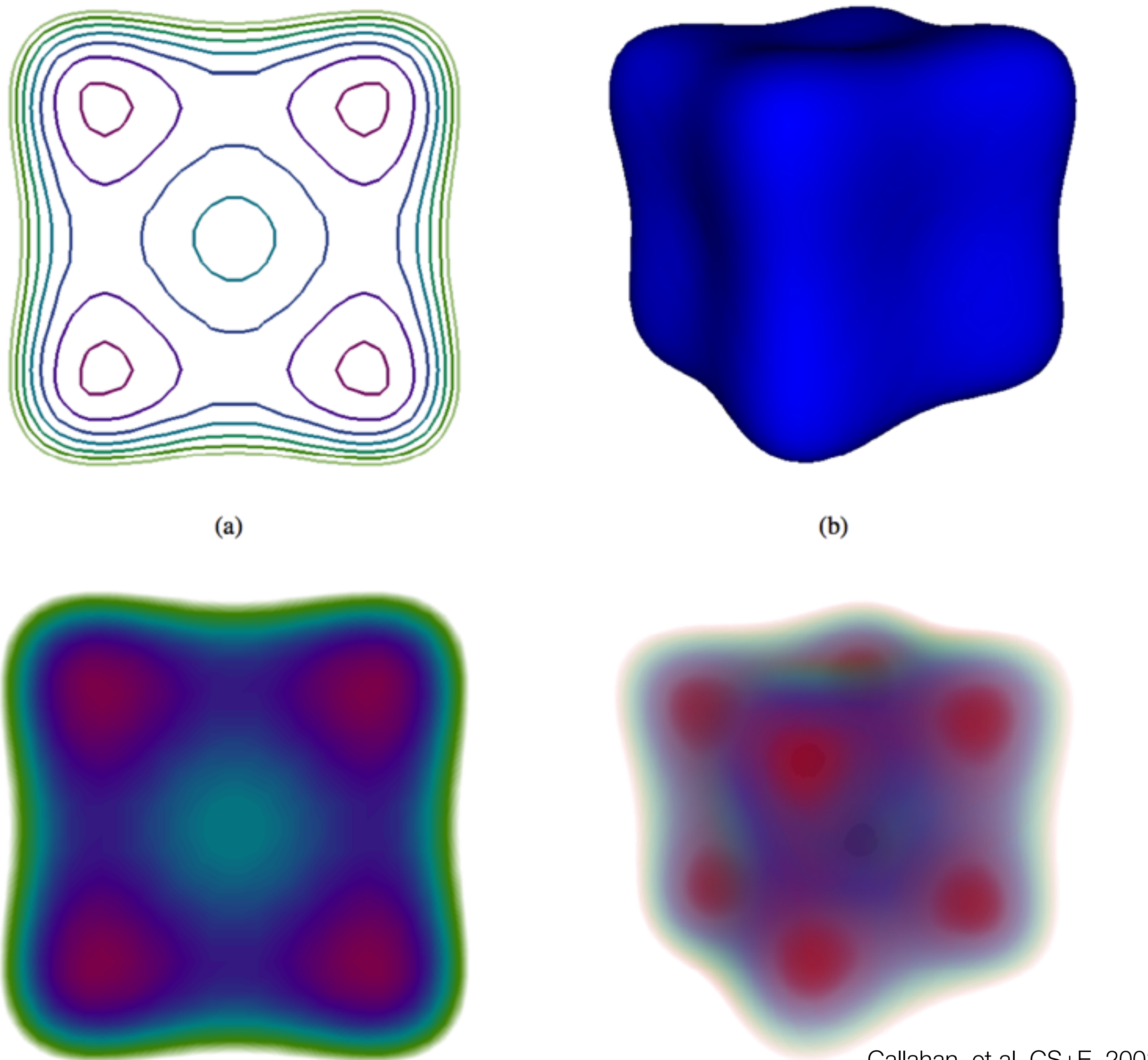
Torsten Möller / Alireza Ghane

Overview



- 2D visualization
  slice images
  (or multi-planar
  reformating MPR)
- Indirect
  3D visualization
  isosurfaces
  (or surface-shaded
  display SSD)
- **Direct**
  3D visualization
  (direct volume
  rendering DVR)



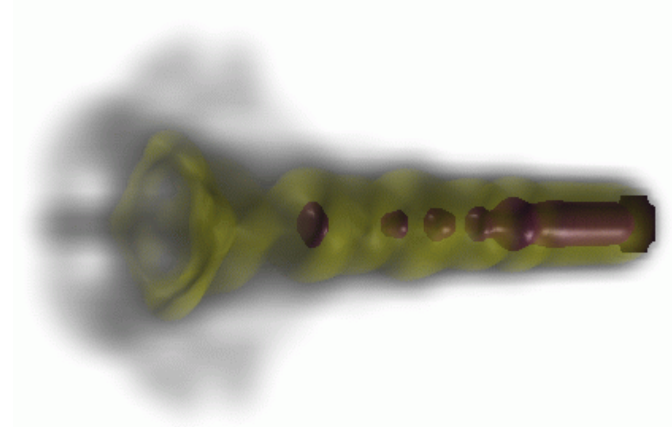


Callahan, et al. CS+E, 2008

Motivation

Motivation

Model



- The data is considered to represent a semi-transparent light-emitting medium
  - Also gaseous phenomena can be simulated
- Approaches are based on the laws of physics (emission, absorption, scattering)
- The volume data is used as a whole (look inside, see all interior structures)

Key-ideas

- Light!
- Transfer functions
- discrete data vs. continuous phenomena (i.e. interpolation)
- Projection: 3D $\Longmapsto$ 2D
- Illusion of interaction (speed!)

Overview

- Light: Volume rendering equation
- Discretized: Compositing schemes
- Ray casting
  - Acceleration techniques for ray casting
- Texture-based volume rendering
- Shear-warp factorization
- Splatting
- Fourier Volume Rendering
- Cell projection (Shirley-Tuchman)

Readings

- The Visualization Handbook:
  - Chapter 7 (Overview of Volume Rendering)
  - Chapter 8 (Volume Rendering Using Splatting)
  - Chapter 10 (Pre-Integrated Volume Rendering)
  - Chapter 11 (Hardware-Accelerated Volume Rendering)
- Engel et al: Real-time Volume Graphics
  - Chapter 1 (Theoretical Background and Basic Approaches)
  - Chapter 3 (Basic GPU-Based Volume Rendering)
  - Chapter 7 (GPU-Based Ray Casting)
  - Chapter 9 (Improving Image Quality)

Readings cont.

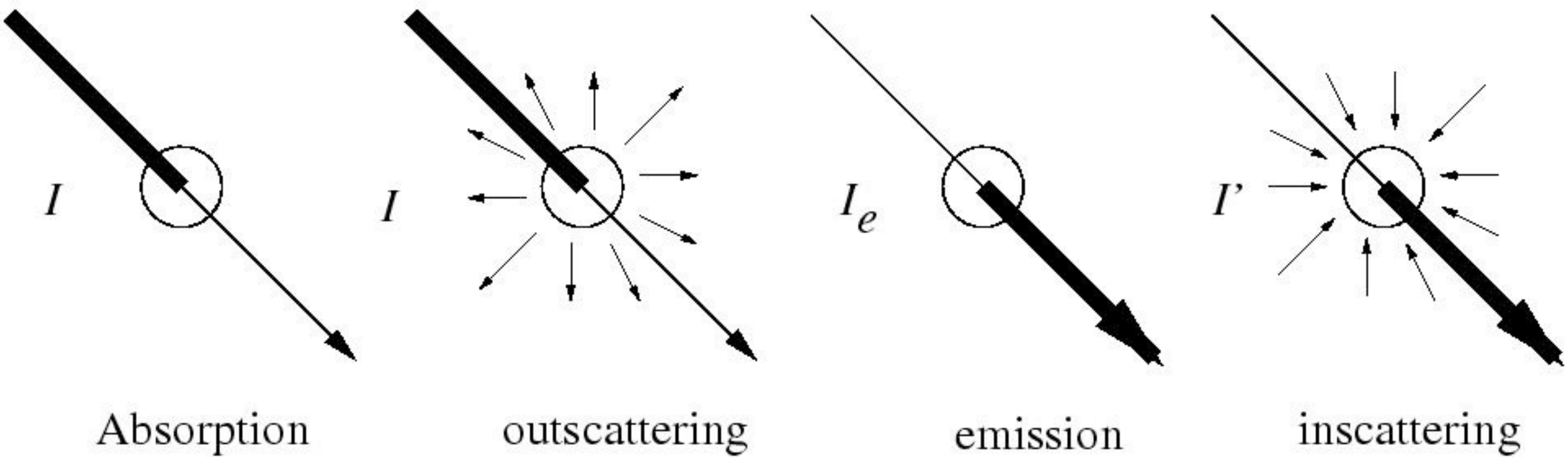
- Malzbender: “Fourier volume rendering”, ACM Transactions on Graphics (TOG), vol. 12(3), July 1993, Pages 233-250
- Totsuka, Levoy, “Frequency domain volume rendering”, SIGGRAPH '93, Pages 271-278

Volume Rendering Equation

- Goal: physical model for volume rendering
  - Emission-absorption model
  - Density-emitter model [Sabella 1988]
  - Leads to volume rendering equation
- More general approach:
  - Linear transport theory
  - Equation of transfer for radiation
  - Basis for all rendering methods
- Important aspects:
  - Absorption
  - Emission
  - Scattering
  - Participating medium

Volume Rendering Equation

- Contributions to radiation at a single position:
  - Absorption



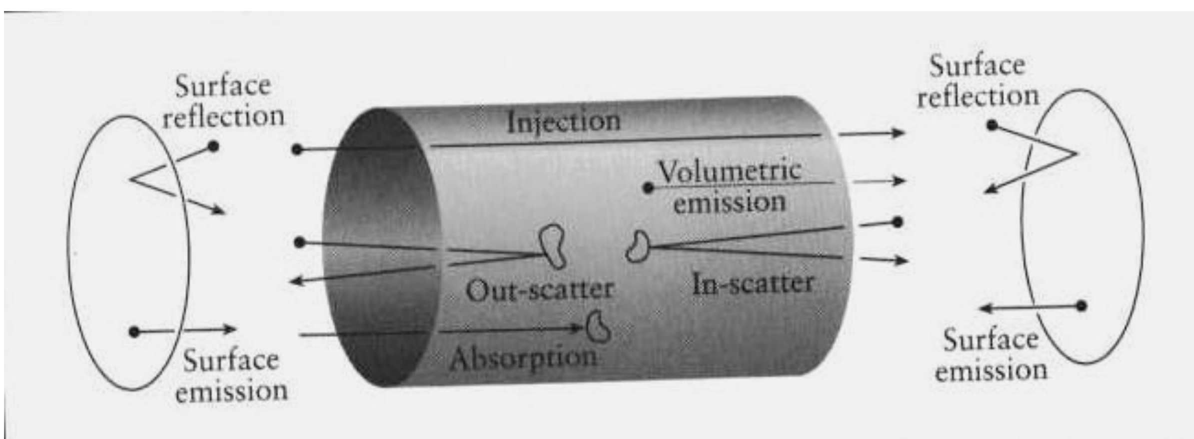
Volume Rendering Equation

- Assumptions:
  - Based on a physical model for radiation
  - Geometrical optics
- Neglect:
  - Diffraction
  - Interference
  - Wave-character
  - Polarization
- Interaction of light with matter at the macroscopic scale
  - Describes the changes of specific intensity due to absorption, emission, and scattering
- Based on energy conservation
- Expressed by equation of transfer

Steady State

- Accumulation =
  flow through boundaries
  - flow out of boundaries
  + generation within system
  - absorption within system

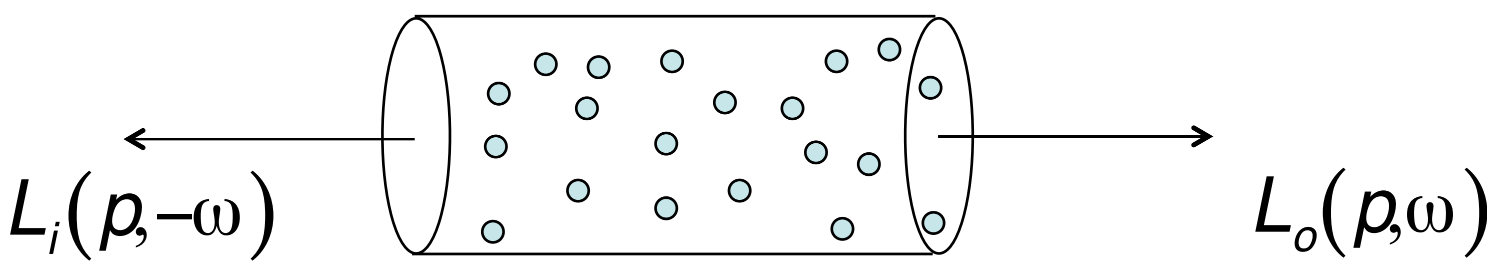
$$\textit{Streaming} + \textit{Absorbance} + \textit{Outscattering} = \textit{Emission} + \textit{Inscattering}$$



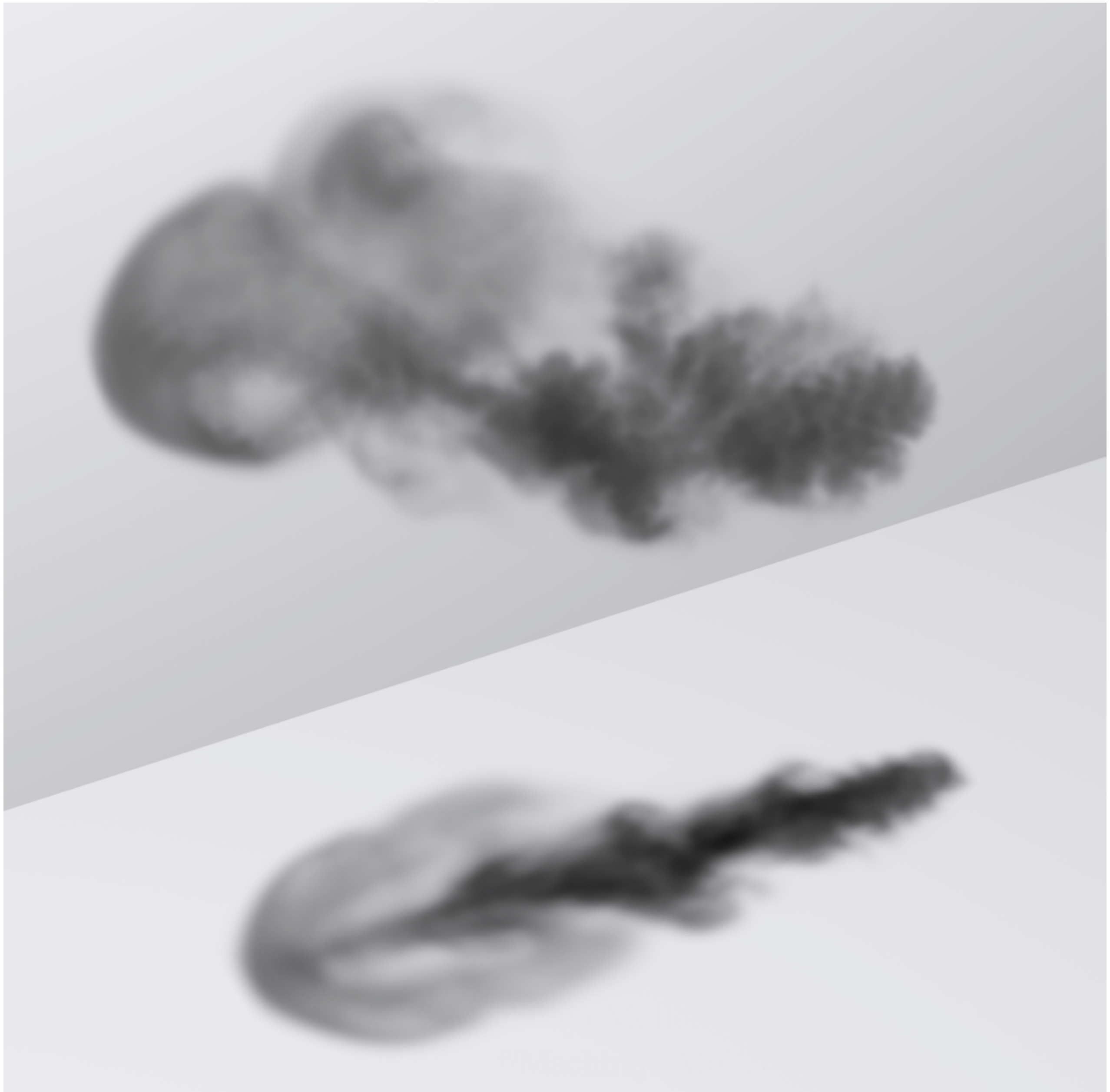
Absorption

- The reduction of radiance due to conversion of light to another form of energy (e.g. heat)
- σ_a: *absorption cross section* - probability density that light is absorbed per unit distance traveled

$$L_o(p,\omega) - L_i(p,-\omega) = dL_o(p,\omega) = -\sigma_a L_i(p,-\omega)dt$$



Absorption

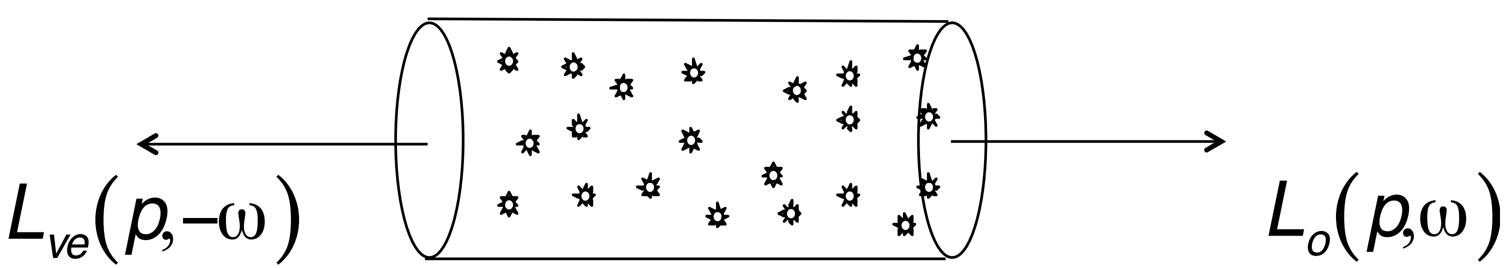


[Pharr, Humphreys, Physically Based Rendering, 2004]

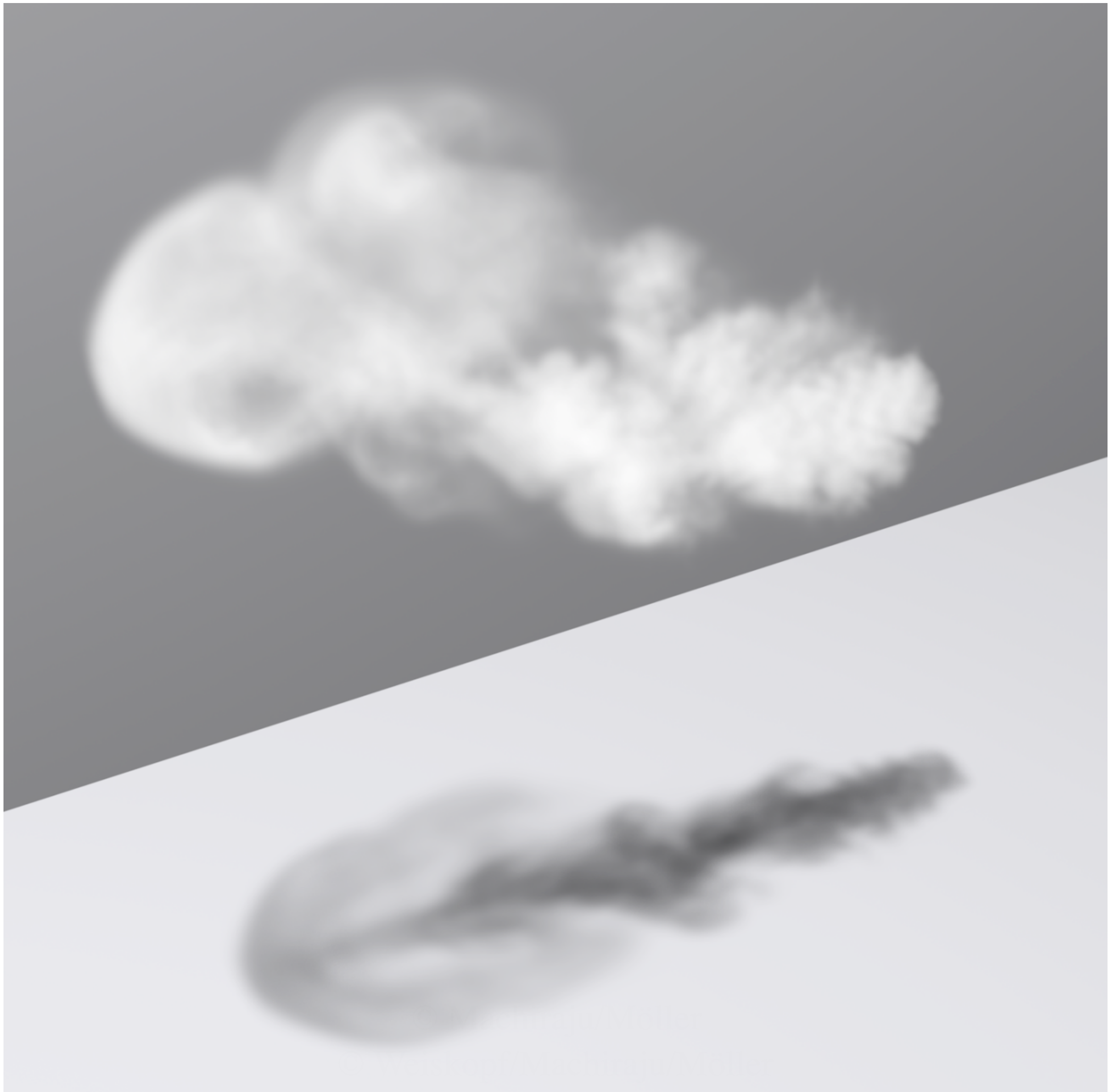
Emission

- Energy that is added to the environment from luminous particles
- L_{ve}: ***emitted light*** - not depending on incoming light!

$$dL_o(p,\omega) = L_{ve}(p,-\omega)dt$$



Emission

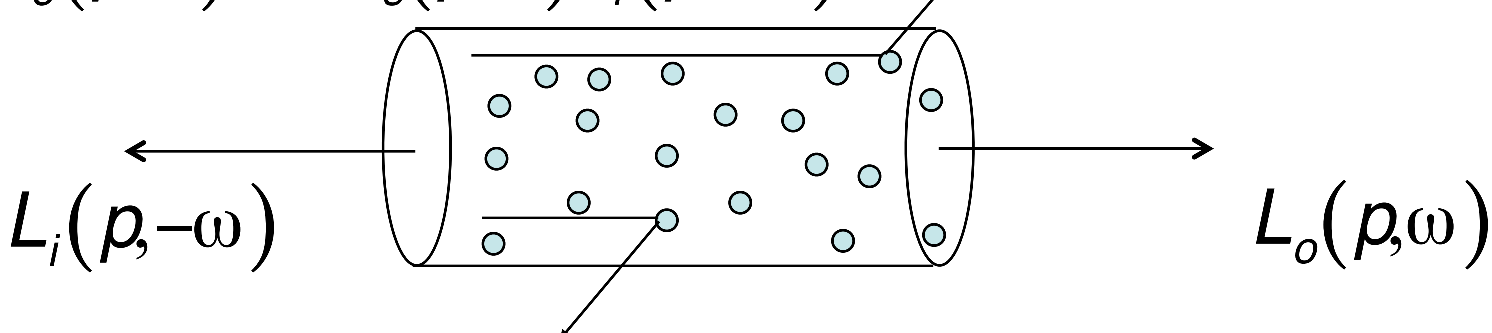


[Pharr, Humphreys, Physically Based Rendering, 2004]

Out-scattering + extinction

- Light heading in one direction is scattered to other directions due to collisions with particles
- σ_s: *scattering coefficient* - probability of an out-scattering event to happen per unit distance

$$dL_o(p,\omega) = -\sigma_s(p,\omega) L_i(p,-\omega) dt$$



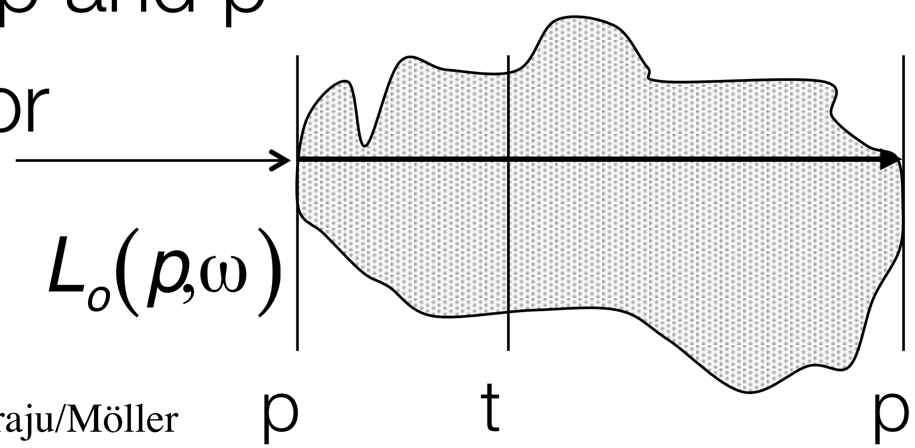
Out-scattering + extinction

- Combining absorption and out-scattering:

$$\sigma_t(p,\omega) = \sigma_s(p,\omega) + \sigma_a(p,\omega)$$

$$\frac{dL_o(p,\omega)}{dt} = -\sigma_t(p,\omega) L_i(p,-\omega)$$

- It's solution: $T_r(p \rightarrow p') = e^{-\int_0^d \sigma_t(p+t\omega,\omega)\,dt}$
  - T_r - beam transmittance
  - d - distance between p and p'
  - ω - unit direction vector



Out-scattering + extinction

- Properties of T_r:
  - In vaccum $T_r(p \rightarrow p') = 1$
  - Multiplicative $T_r(p \rightarrow p'') = T_r(p \rightarrow p') \cdot T_r(p' \rightarrow p'')$
  - Beer's law (in homogeneous medium)

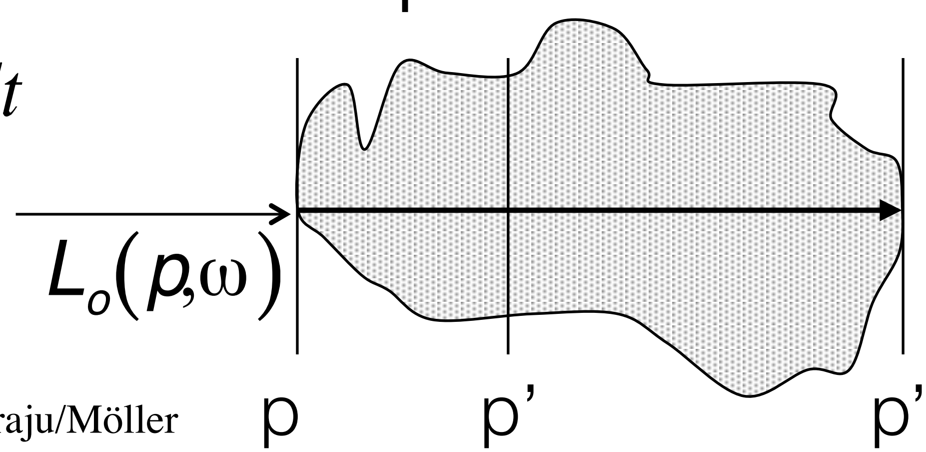
$$T_r(p \rightarrow p') = e^{-\sigma_t d}$$

- Optical thickness between two points:

$$\tau(p \rightarrow p') = \int_0^d \sigma_t(p + t\omega, \omega)\,dt$$

- Often used:

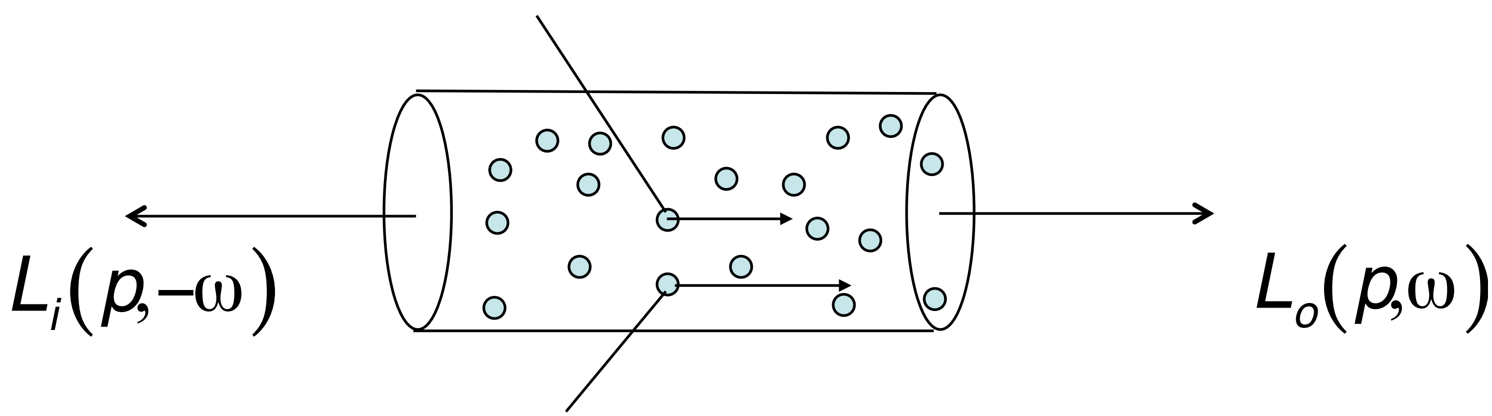
$$T_r(p \rightarrow p') \approx 1 - \tau(p \rightarrow p')$$



In-scattering

- Increased radiance due to scattering from other directions
  - Ignore inter-particle reactions
  - S - *source term*: total added radiance per unit distance

$$dL_o(p,\omega) = S(p,\omega)dt$$

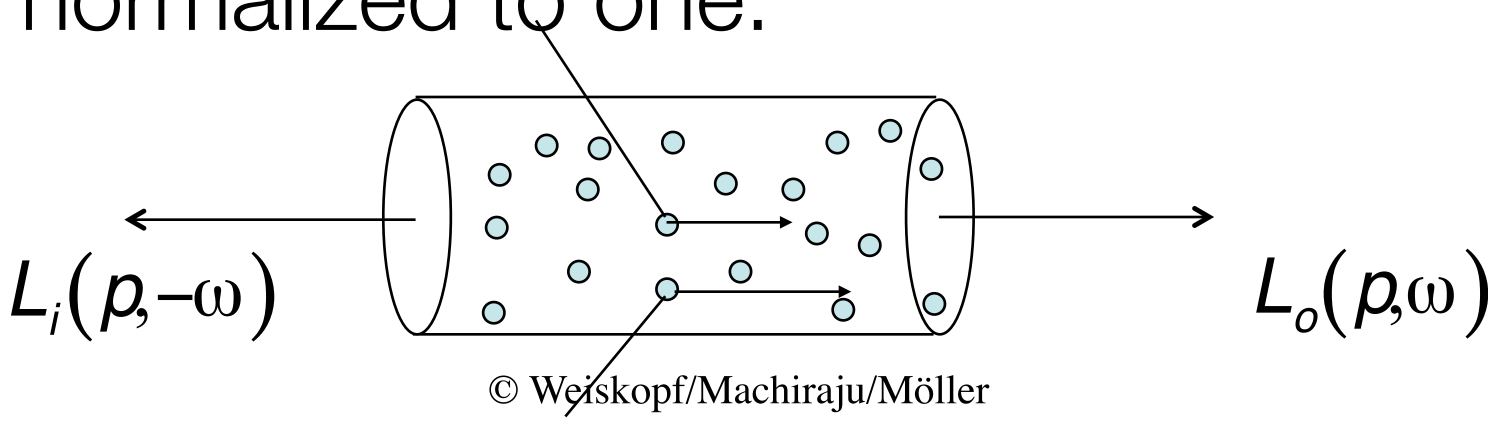


In-scattering

$$S(p,\omega) = L_{ve}(p,\omega) + \sigma_s(p,\omega)\int_{S^2} p(p,-\omega' \rightarrow \omega)L_i(p,\omega')d\omega'$$

- S determined by
  - Volume emission
  - p - *phase function*: describes angular distribution of scattered radiation (volume analog of BSDF)
- p normalized to one:

$$\int_{S^2} p(\omega \rightarrow \omega')d\omega' = 1$$



In-scattering

[Pharr, Humphreys, Physically Based Rendering, 2004]

Overview

- Light: Volume rendering equation
- Discretized: Compositing schemes
- Ray casting
  - Acceleration techniques for ray casting
- Texture-based volume rendering
- Shear-warp factorization
- Splatting
- Fourier Volume Rendering
- Cell projection (Shirley-Tuchman)

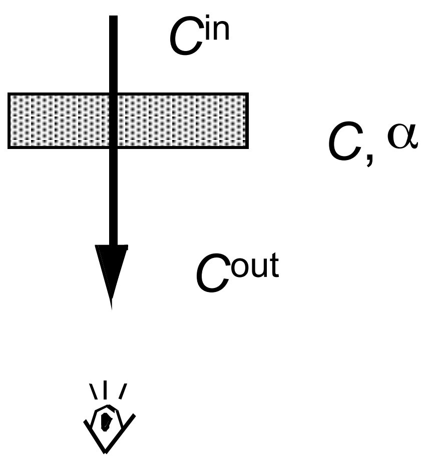
Compositing

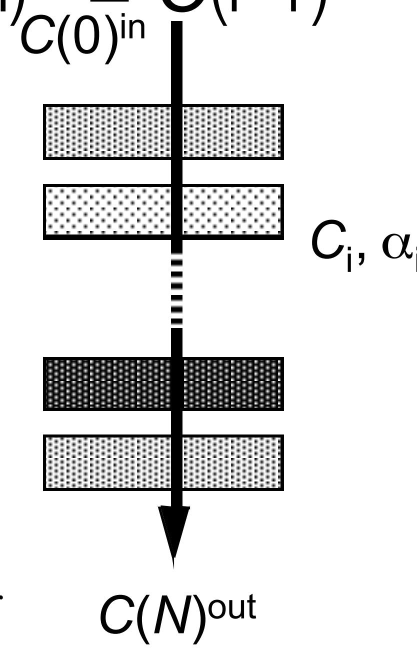
- Compositing = iterative computation of discretized volume integral
- Traversal strategies
  - Front-to-back
  - Back-to-front $C^{out} = C^{in} \times (1 - \alpha) + C$
- Directly derived from discretized integral
- Just different notation:
- Colors **C** and opacity α are assigned with transfer function

Back-to-front

- Over operator [Porter & Duff 1984]
- Used, e.g., in texture-based volume rendering
- Compositing equation:

$$C^{out} = (1 - \alpha)\, C^{in} + C \qquad C(i)^{in} = C(i-1)^{out}$$

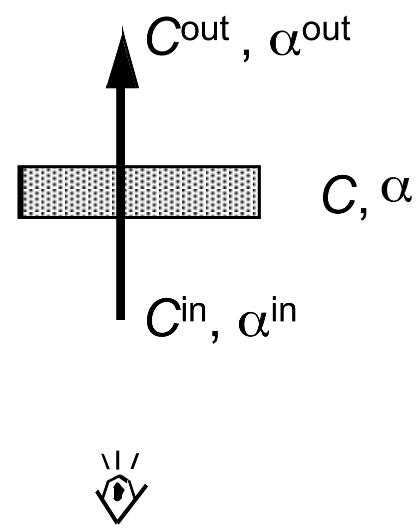




Front-to-back

- Needs to maintain α^{in}
- Most often used in ray casting
- Compositing equation:

$$C^{out} = C^{in} + (1 - \alpha^{in})\, C$$
$$\alpha^{out} = \alpha^{in} + (1 - \alpha^{in})\, \alpha$$



Compositing

- Associated colors
  - Color contributions are already weighted by their corresponding opacity
  - Also called pre-multiplied colors
- Non-associated colors: $C \rightarrow C\alpha$
  - Just substitute in compositing equations
- Yields the same results as associated colors (on a cont. level)
  - Differences occur when combined with interpolation + post-classification
- Ex.: back-to-front compositing with non-associated colors:

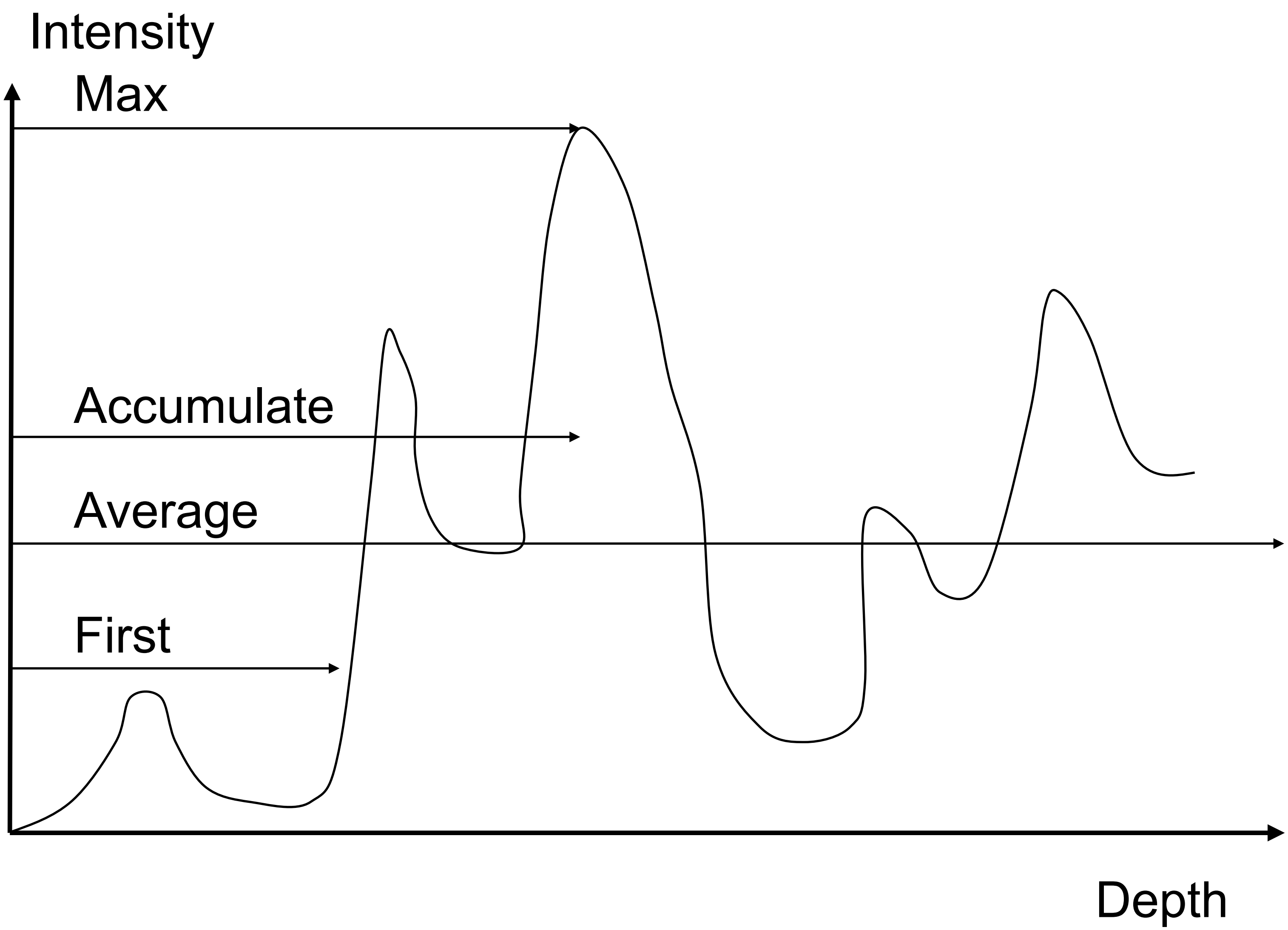
$$C^{out} = (1 - \alpha)\, C^{in} + C\alpha$$

  - Standard OpenGL blending for semi-transparent surfaces

Compositing

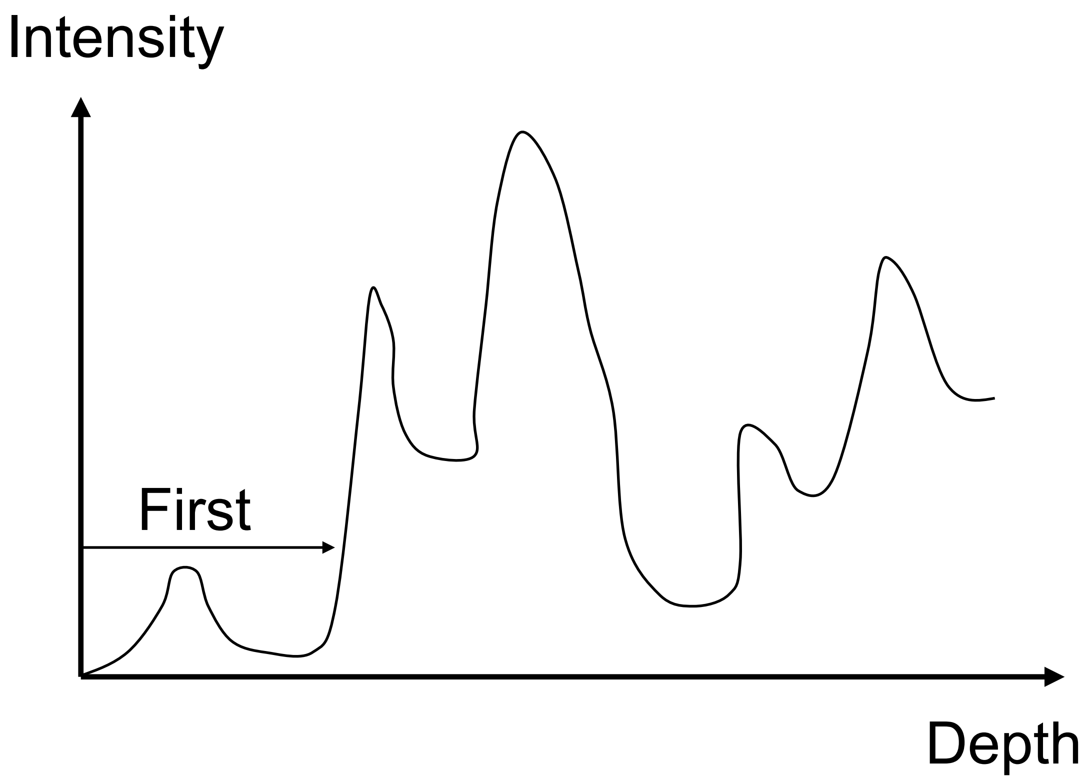
- So far: accumulation scheme
- Variations of composition schemes
  - First
  - Average
  - Maximum intensity projection

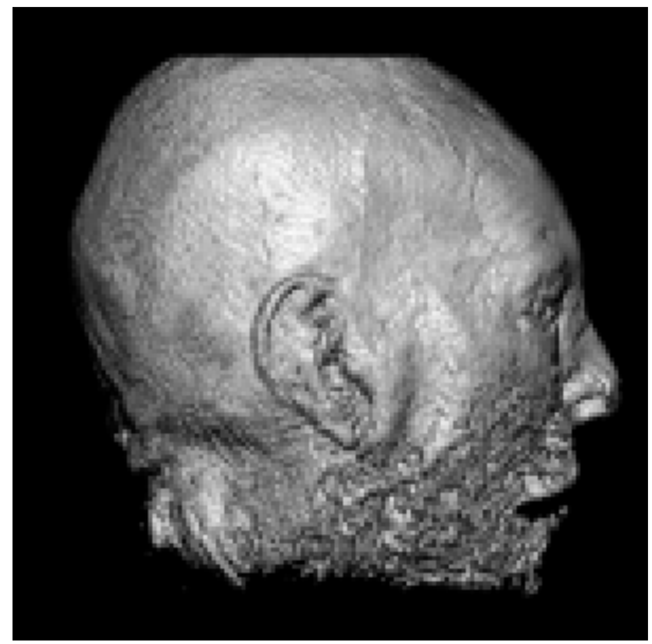
Compositing



Compositing

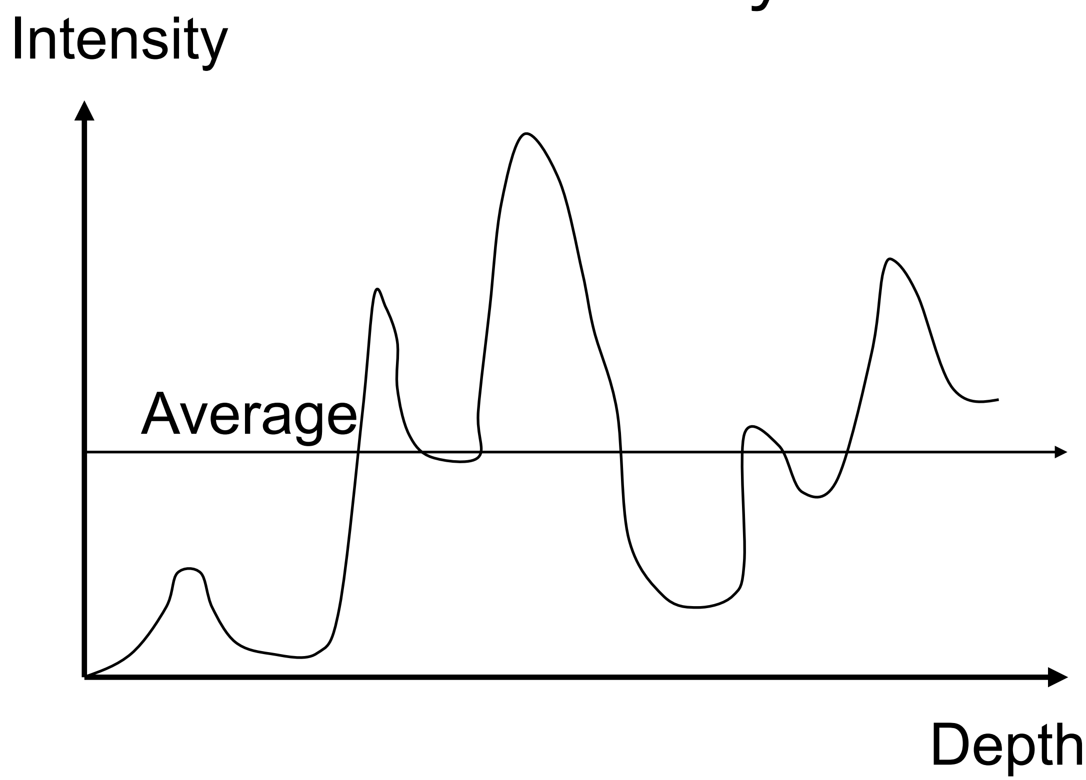
- Compositing: First
- Extracts isosurfaces

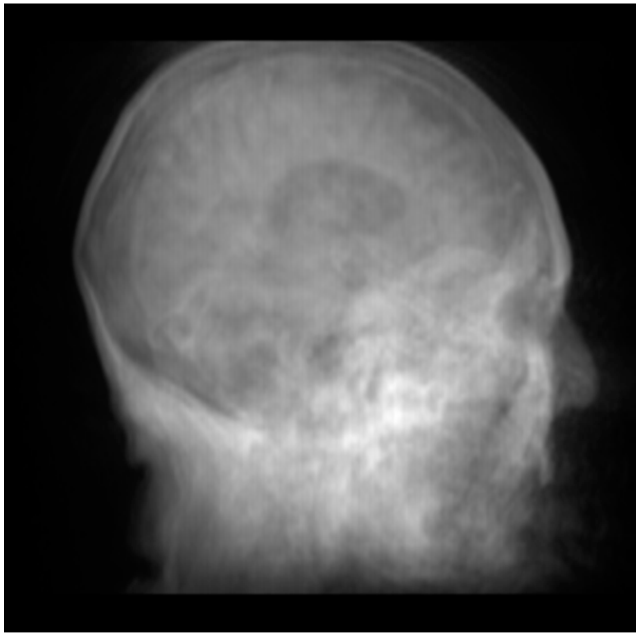




Compositing

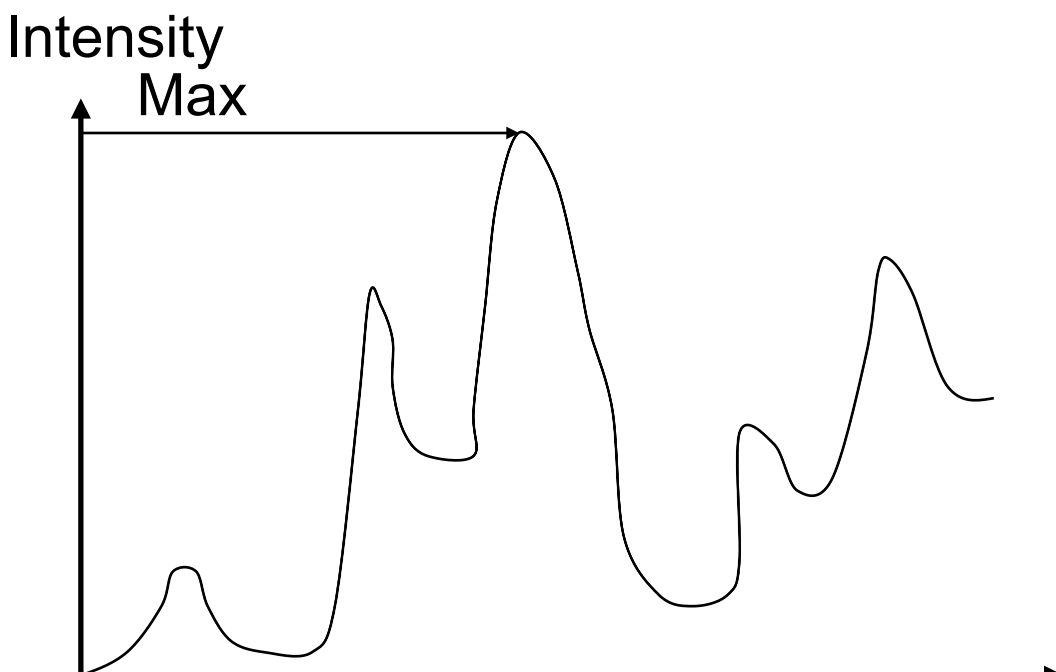
- Compositing: Average
- Produces basically an X-ray picture

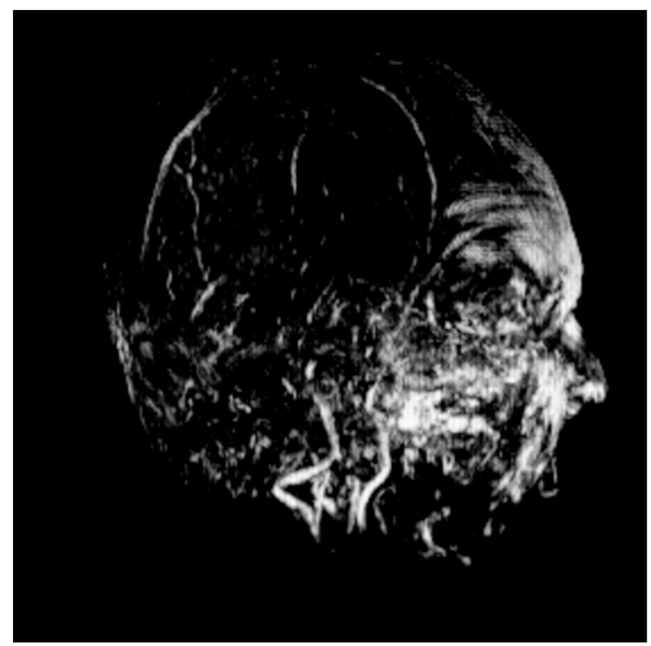




Compositing

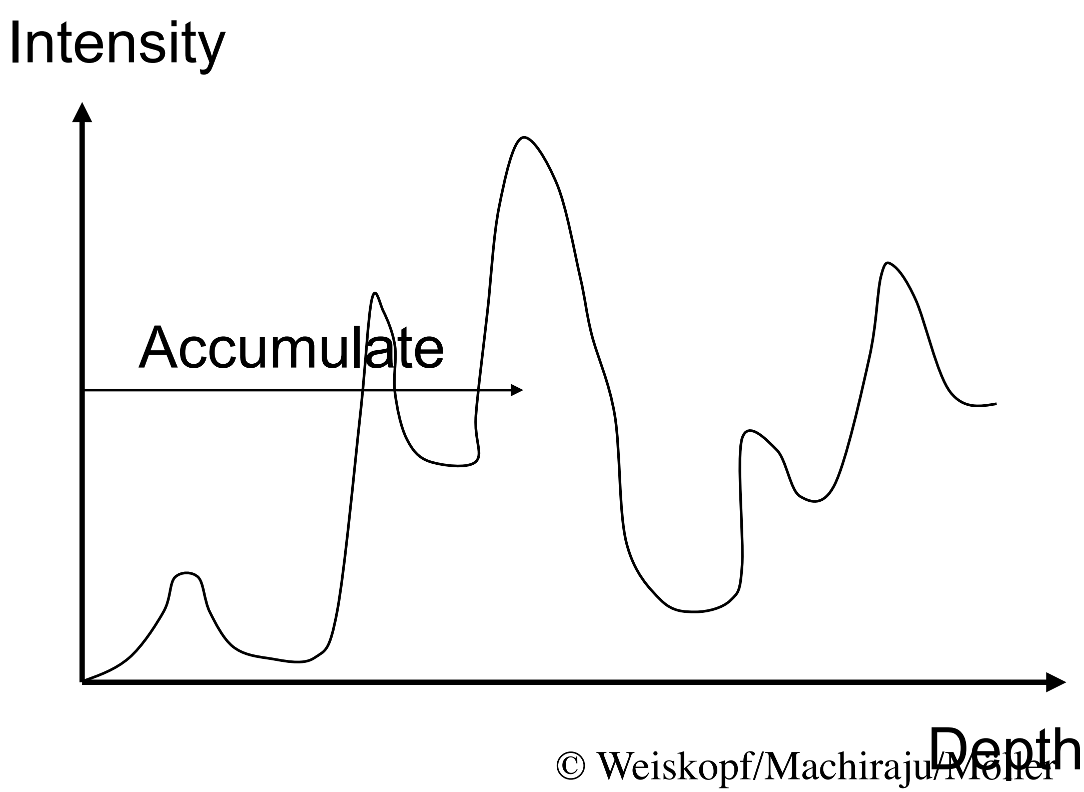
- Maximum Intensity Projection (MIP)
- Often used for MR or CT angiograms
- Good to extract vessel structures

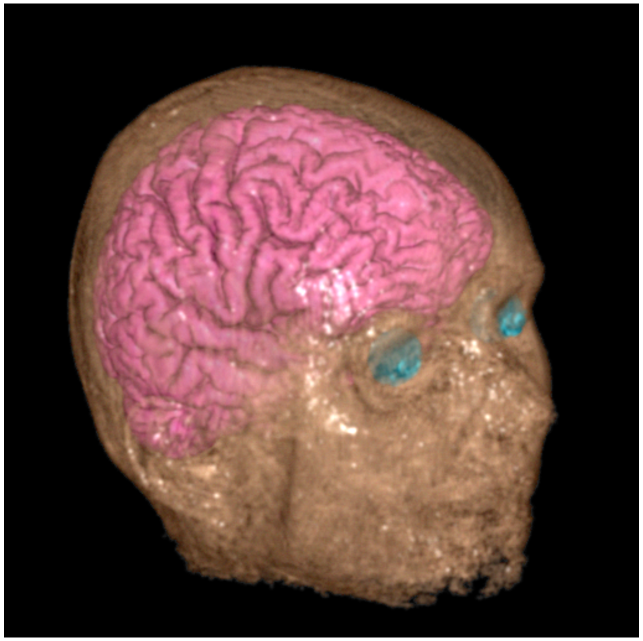




Compositing

- Compositing: Accumulate
- Emission-absorption model
- Make transparent layers visible (see classif.)





Compositing

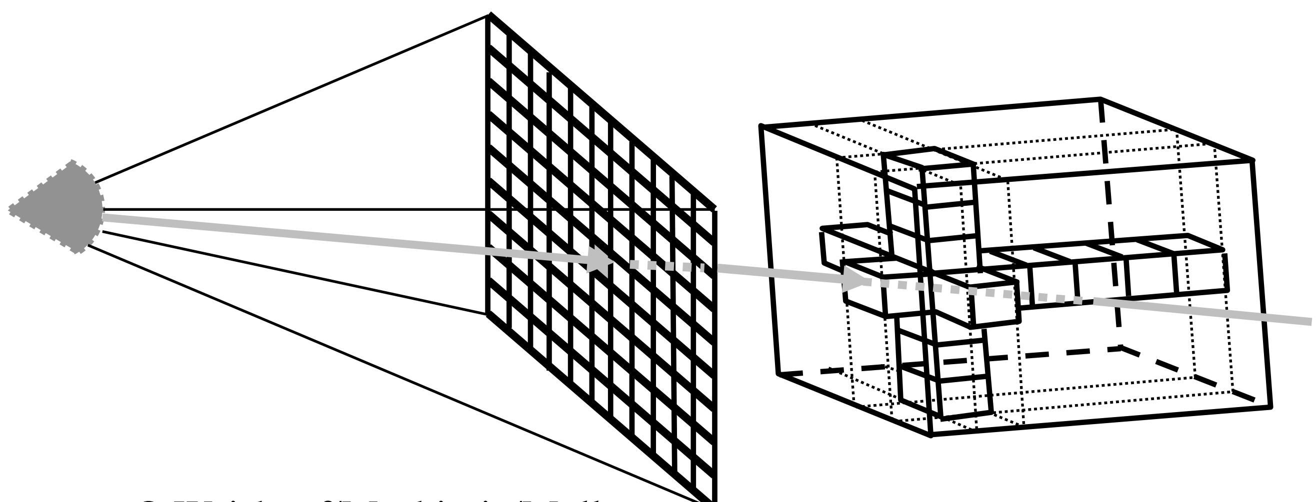
- Note: First and average are special cases of accumulate

Overview

- Light: Volume rendering equation
- Discretized: Compositing schemes
- Ray casting
  - Acceleration techniques for ray casting
- Texture-based volume rendering
- Shear-warp factorization
- Splatting
- Fourier Volume Rendering
- Cell projection (Shirley-Tuchman)

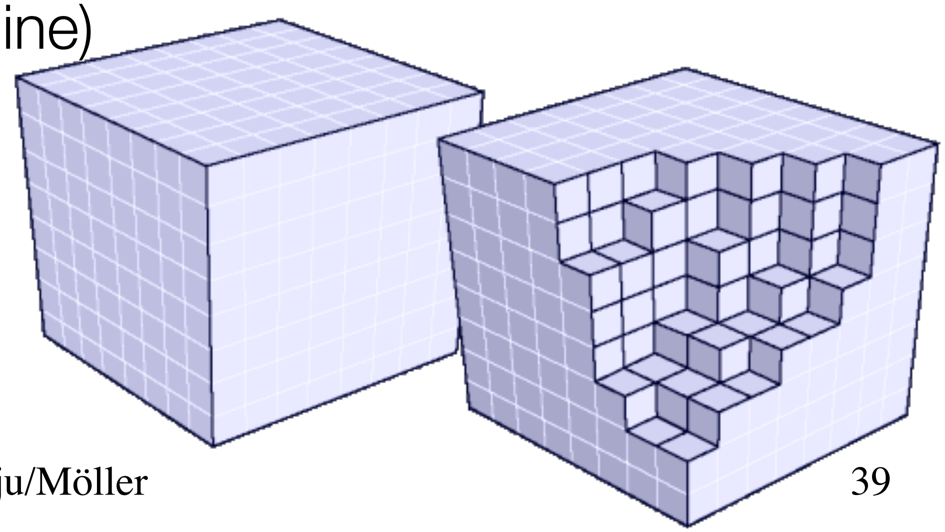
Ray Casting

- Similar to ray tracing in surface-based computer graphics
- In volume rendering we only deal with primary rays; hence: ray casting
- Natural image-order technique
- As opposed to surface graphics - how do we calculate the ray/surface intersection?



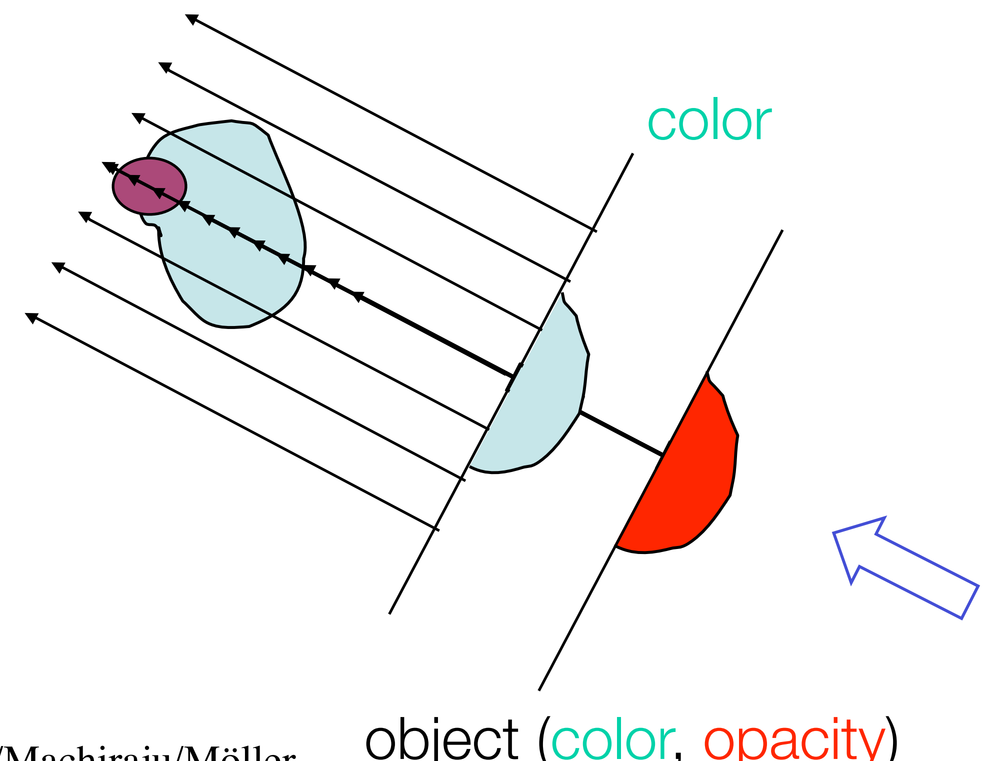
Ray Casting

- Since we have no surfaces - carefully step through volume
- A ray is cast into the volume, sampling the volume at certain intervals
- Sampling intervals are usually equidistant, but don't have to be (e.g. importance sampling)
- At each sampling location, a sample is interpolated / reconstructed from the voxel grid
- Popular filters are: nearest neighbor (box), trilinear, or more sophisticated (Gaussian, cubic spline)
- First: Ray casting in uniform grids
  - Implicit topology
  - Simple interpolation schemes

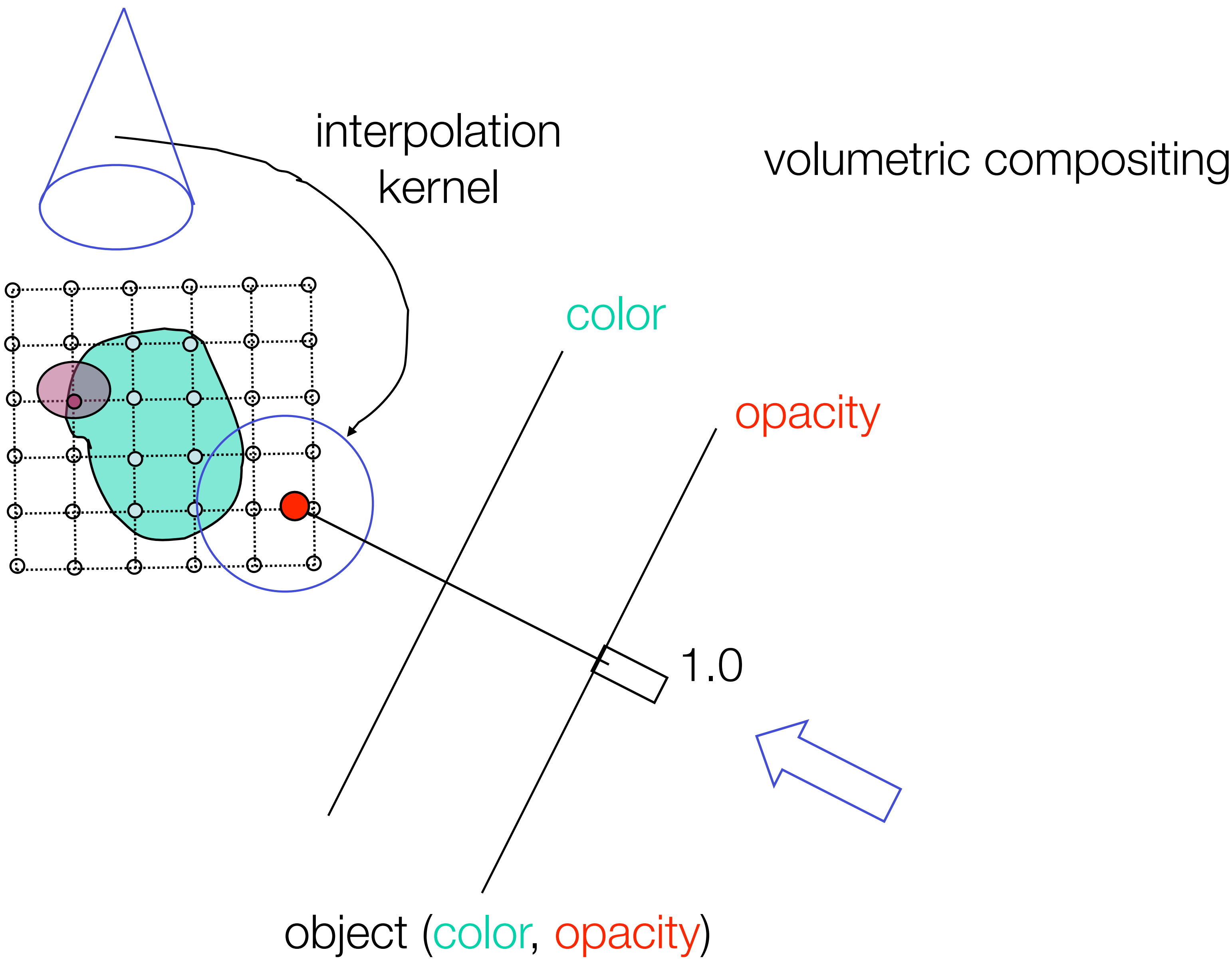


Ray Casting

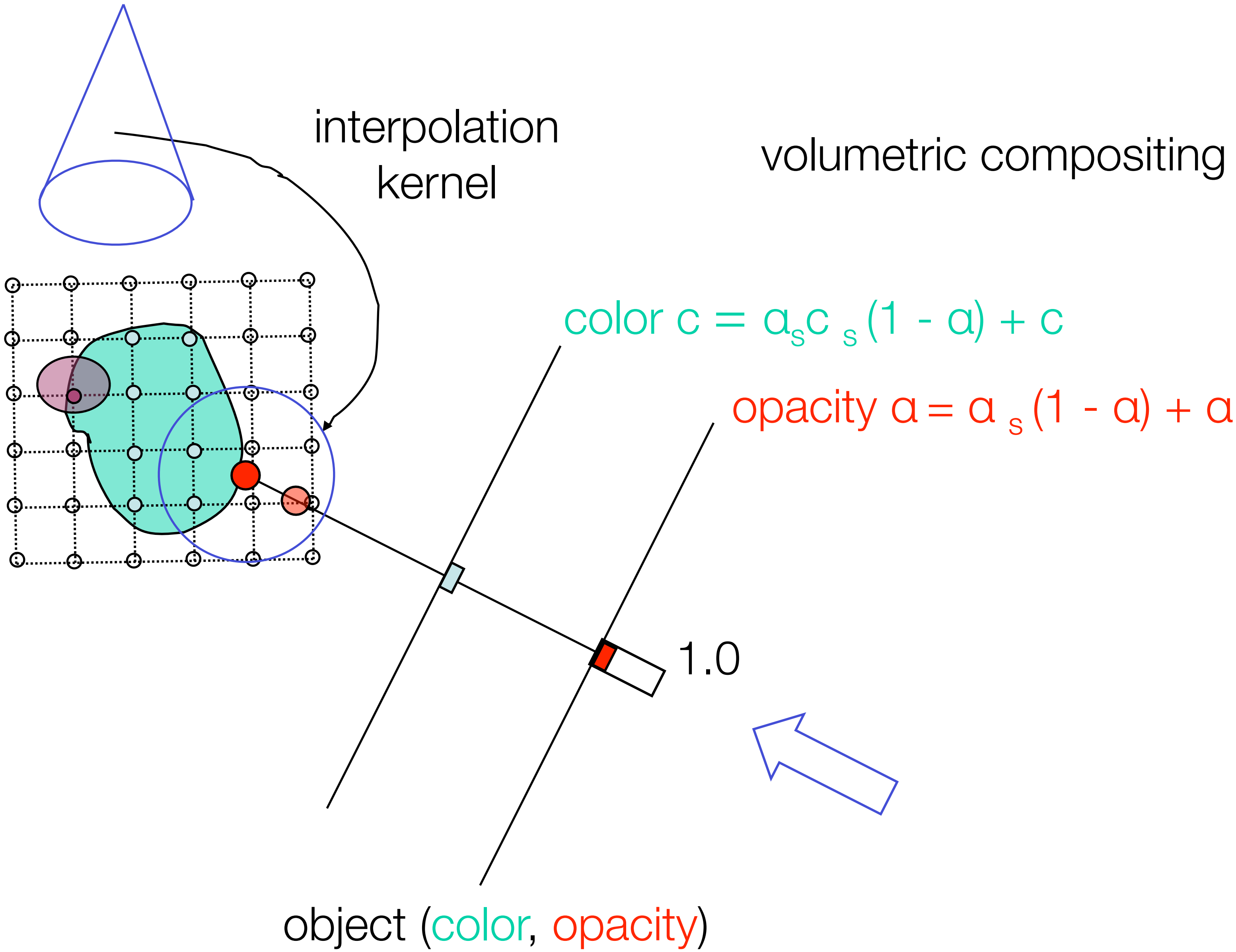
- Volumetric ray integration:
  - Tracing of rays
  - Accumulation of color and opacity along ray: compositing



Ray Casting


interpolation
kernel
volumetric compositing
color
opacity
1.0
object (color, opacity)

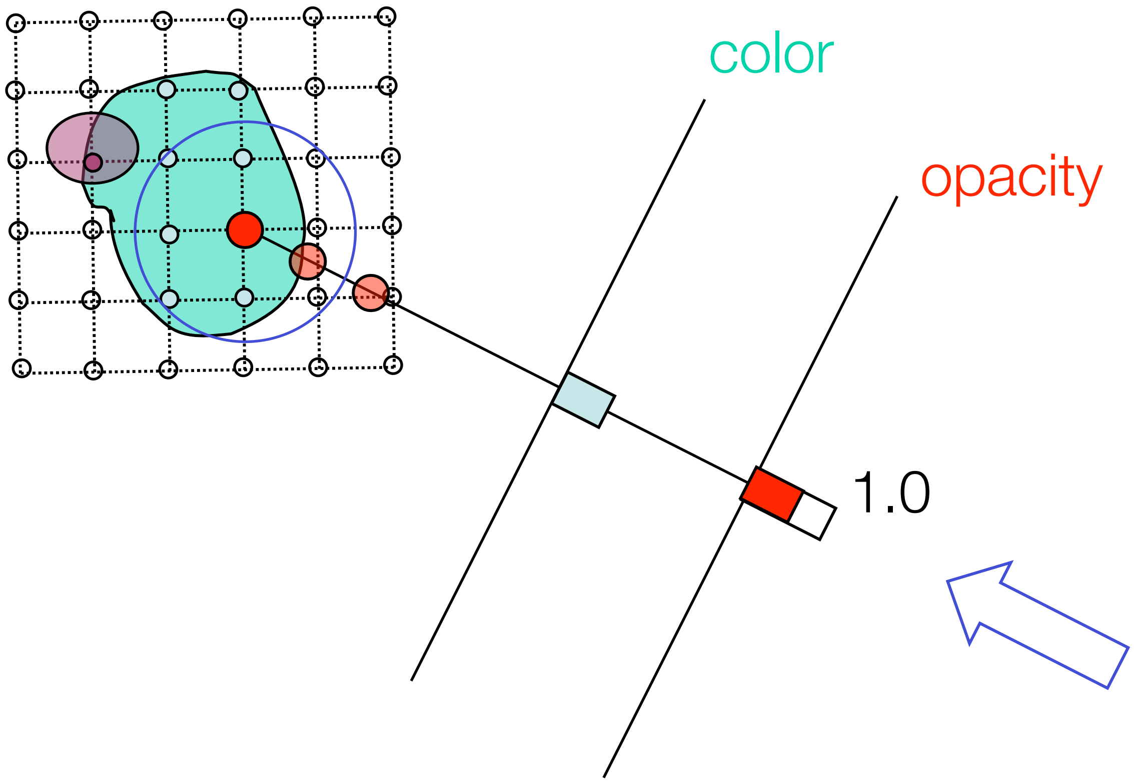
Ray Casting



interpolation
kernel
volumetric compositing
color c = αsc s (1 - α) + c
opacity α = α s (1 - α) + α
1.0
object (color, opacity)

Ray Casting

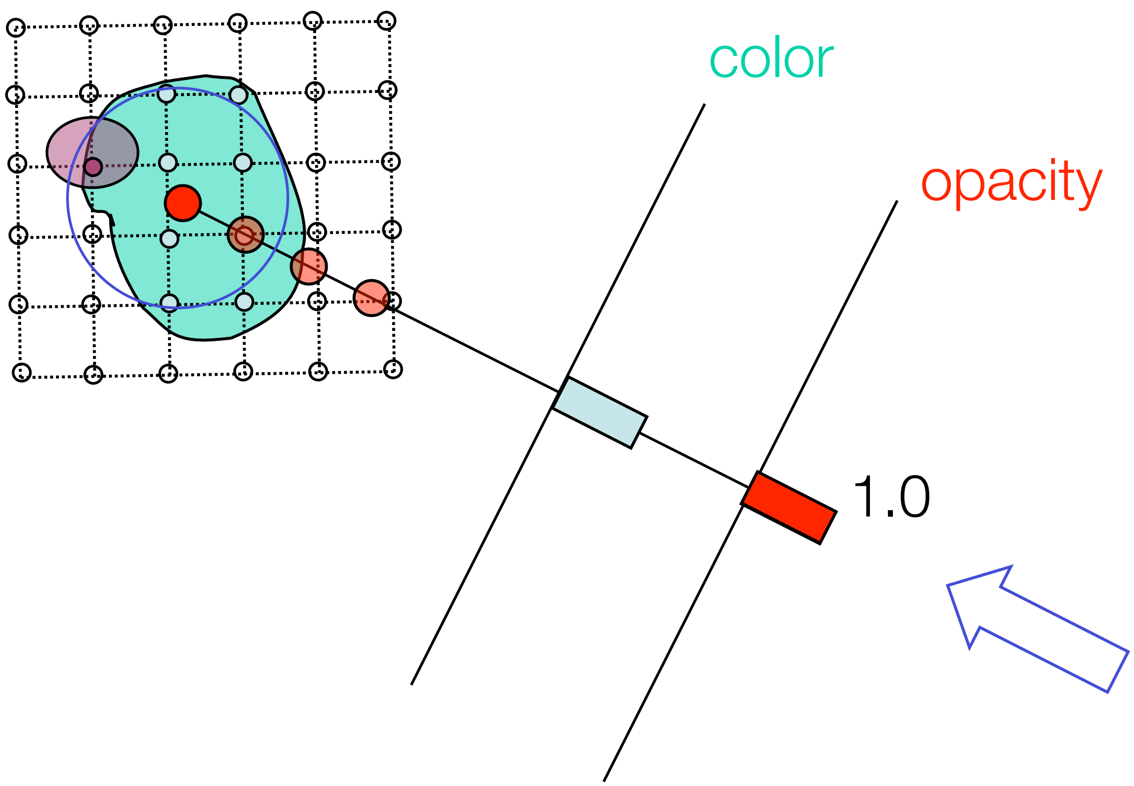
volumetric compositing



object (color, opacity)

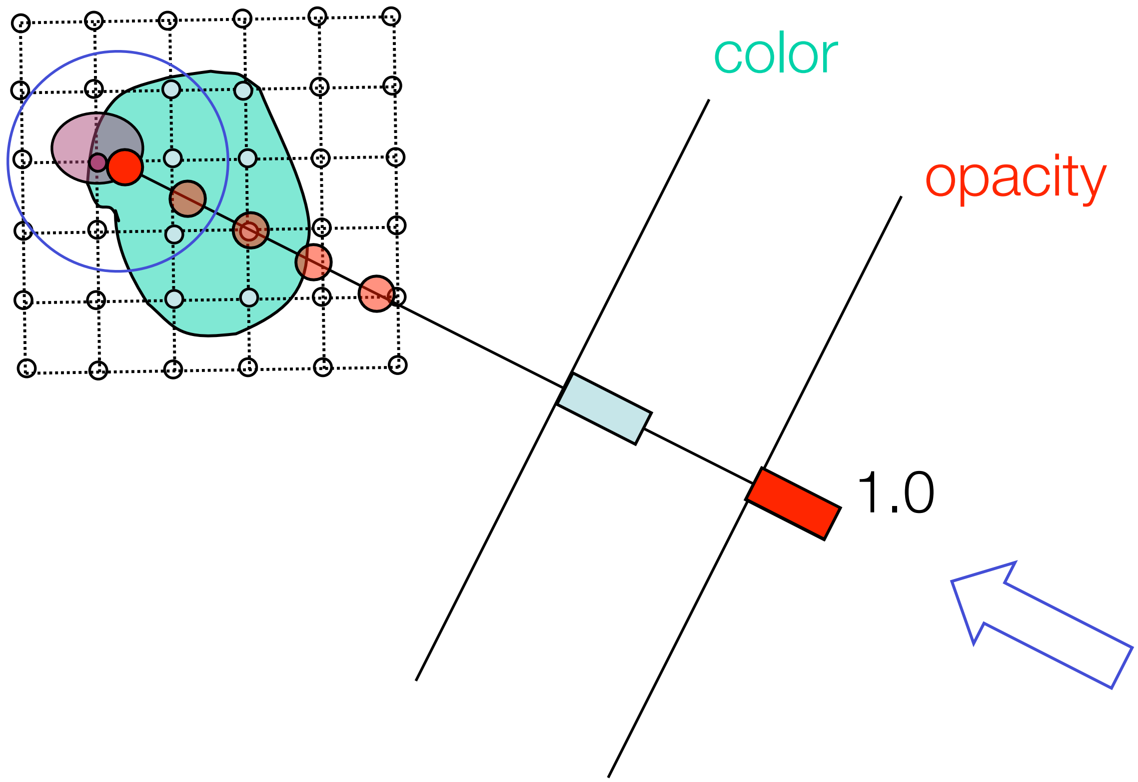
Ray Casting

volumetric compositing



Ray Casting

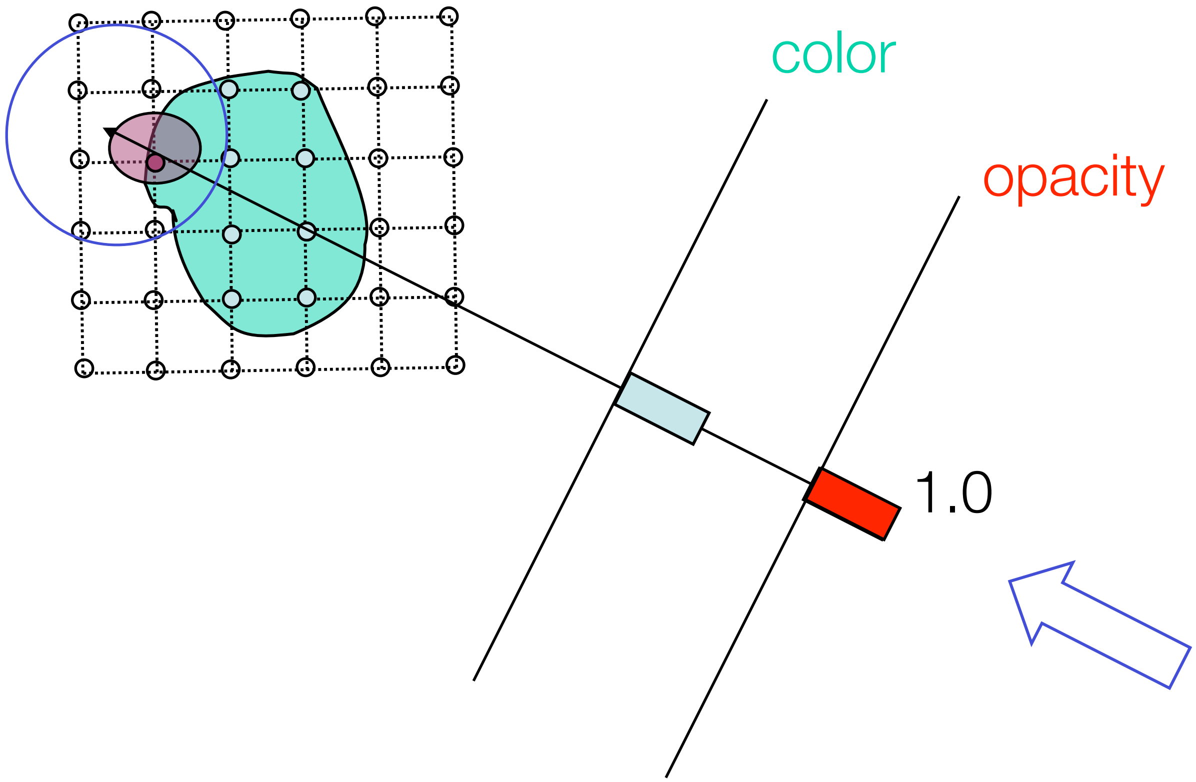
volumetric compositing



object (color, opacity)

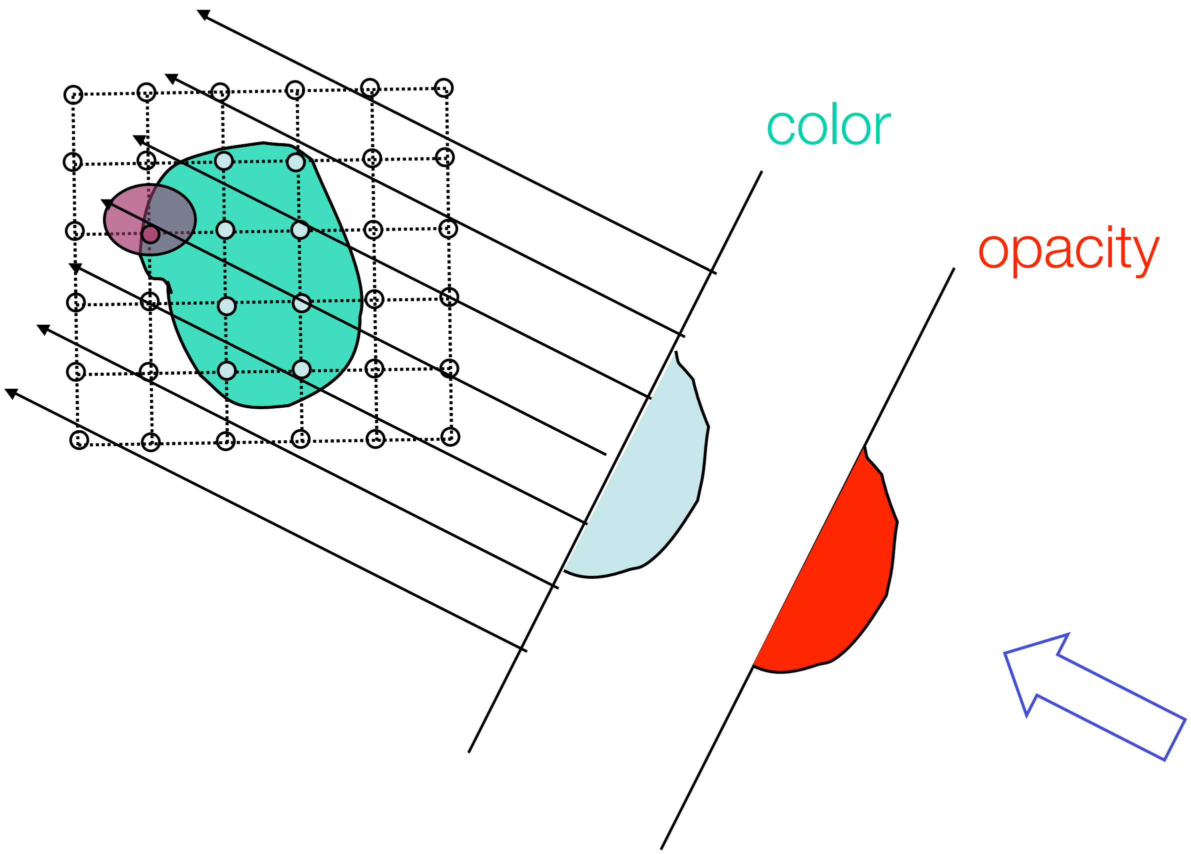
Ray Casting

volumetric compositing



Ray Casting

volumetric compositing



object (color, opacity)

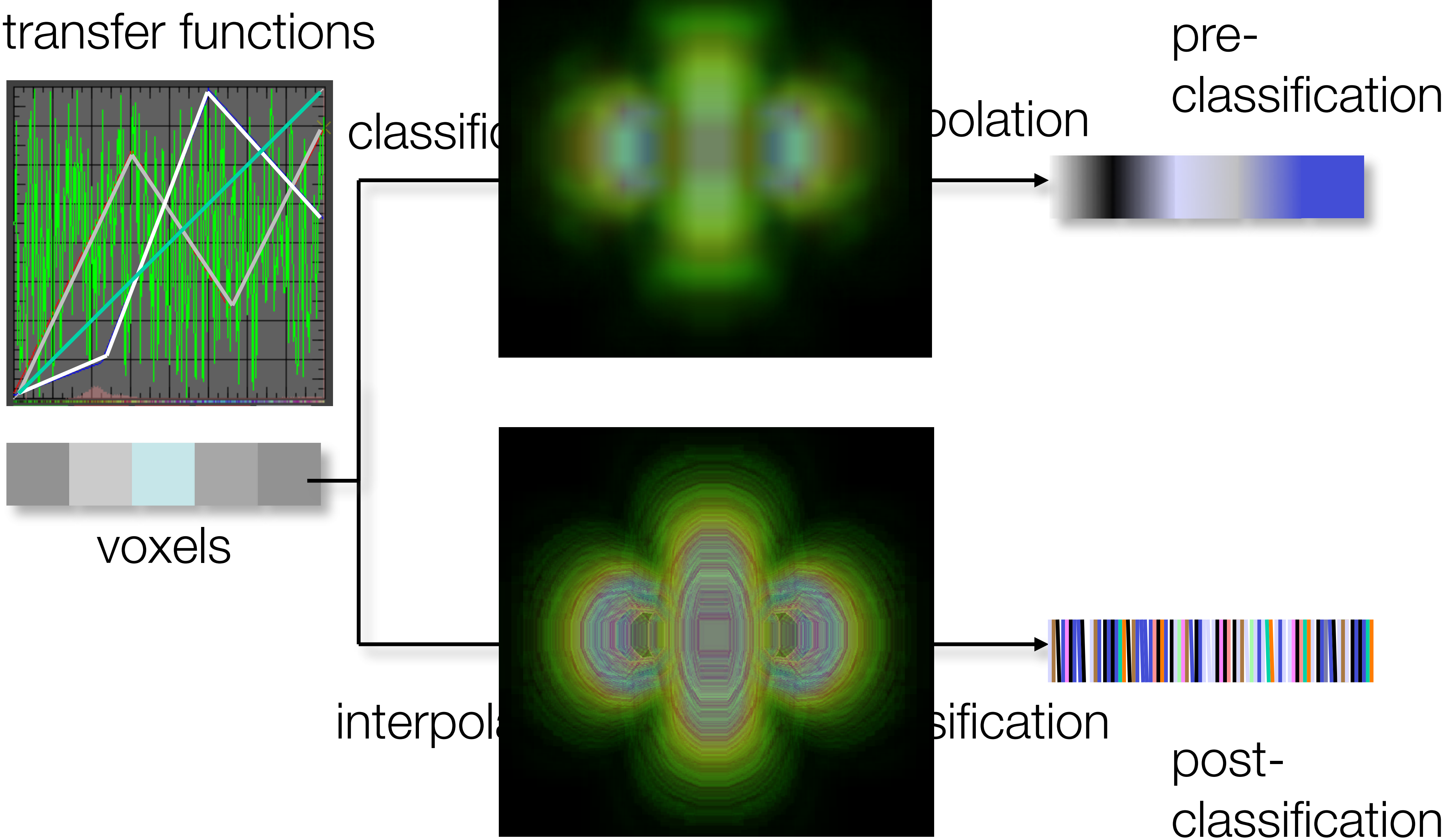
Ray Casting

- How is color and opacity at each integration step determined?
- Opacity and (emissive) color in each cell according to classification
- Additional color due to external lighting: according to volumetric shading (e.g. Blinn-Phong)
- No shadowing, no secondary effects
- Implementations
  - Traditional CPU implementation
  - straightforward, very efficient GPU implemenations
    - Fragment shader loops (Shader Model 3 GPUs)

Determining color at each step

- Pre-shading
  - Assign color values to original function values
  - Interpolate between color values
- Post-shading
  - Interpolate between scalar values
  - Assign color values to interpolated scalar values

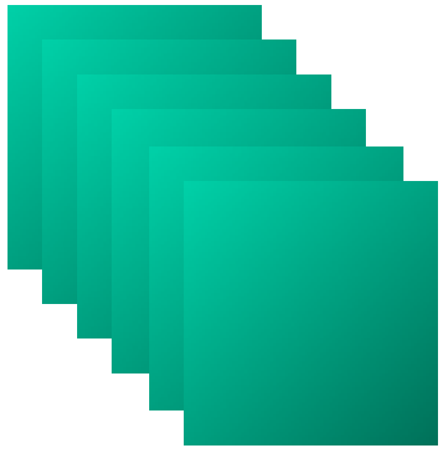
Classification

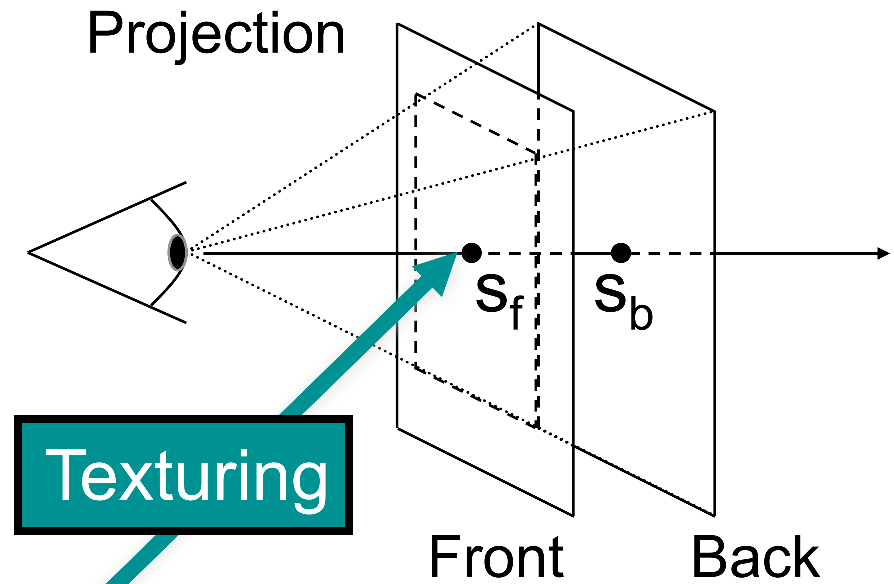


Pre-integrated Rendering

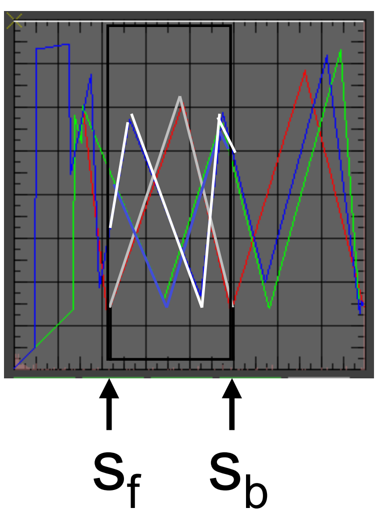
Slice-by-slice

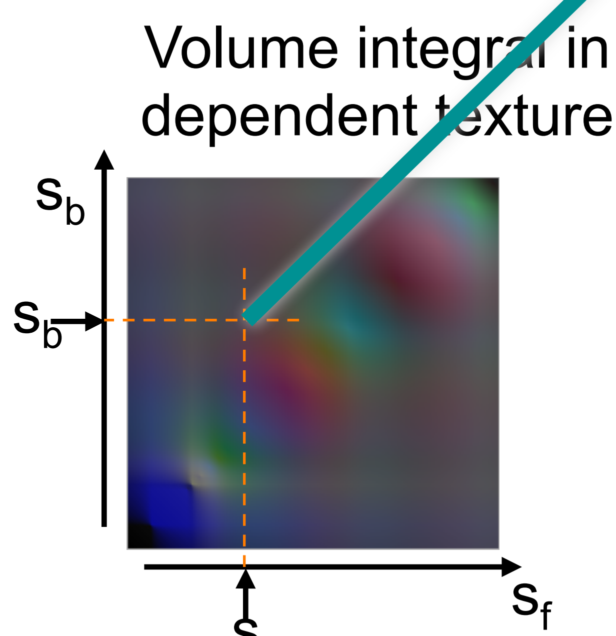
Slab-by-slab





Pre-integration of all combinations





Pre-integrated Rendering

- Assumptions:
  - Linear interp. of scalar values within a slab
  - Constant length of a slab: L
  - Only an approximation, but gives good results in most cases

- Pre-computation of all potential contrib. from a slab

$$s_L(t) = s_b + \frac{t}{L}(s_f - s_b)$$ (linear interpolation within a slab)

apply TF

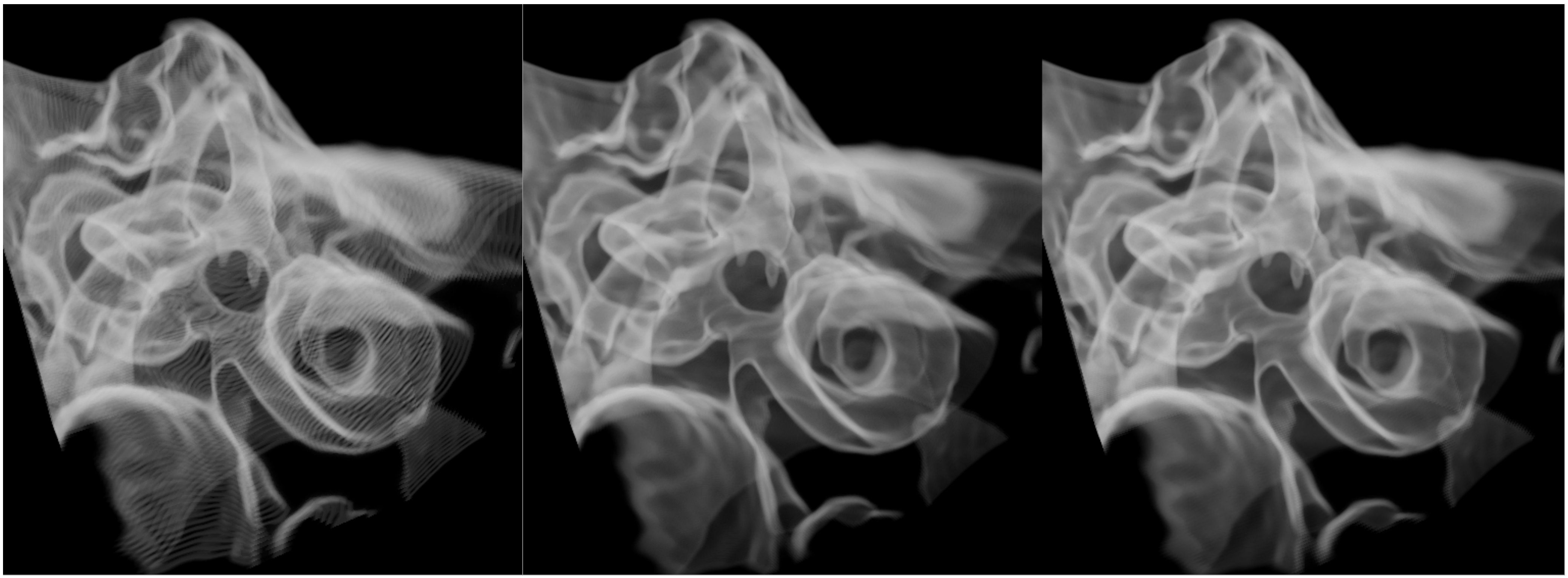
$$\theta = e^{-\int_0^L \kappa(t)\,dt} \Rightarrow \alpha = 1 - \theta$$

$$b = \int_0^L q(t)\, e^{-\int_s^L \kappa(t')\,dt'}\, dt \Rightarrow \text{RGB}$$

} pre-integrated RGBA values

Pre-integrated Rendering

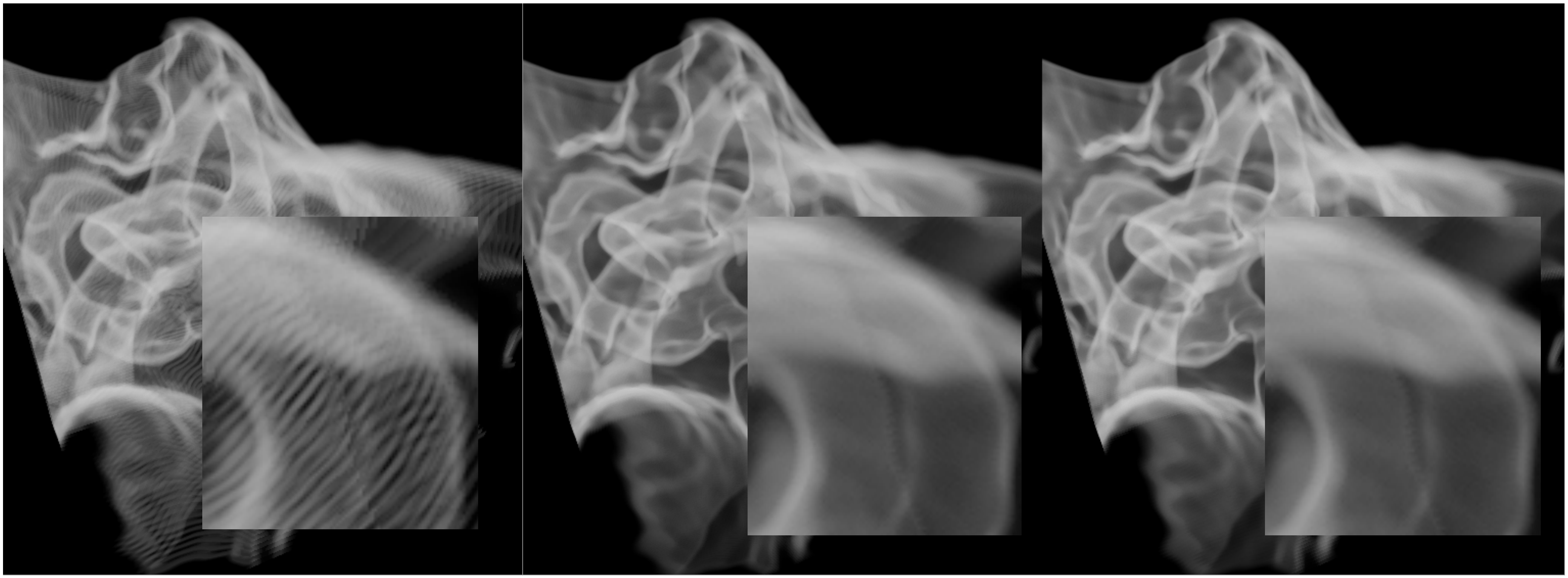
- Quality comparison



128 Slices 284 Slices 128 Slabs

Pre-integrated Rendering

- Quality comparison



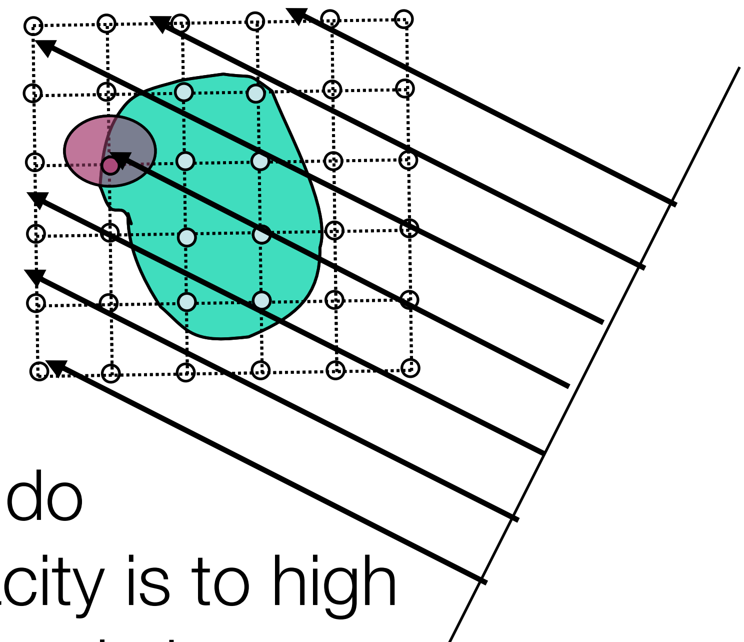
128 Slices 284 Slices 128 Slabs

Overview

- Light: Volume rendering equation
- Discretized: Compositing schemes
- Ray casting
  - Acceleration techniques for ray casting
- Texture-based volume rendering
- Shear-warp factorization
- Splatting
- Fourier Volume Rendering
- Cell projection (Shirley-Tuchman)

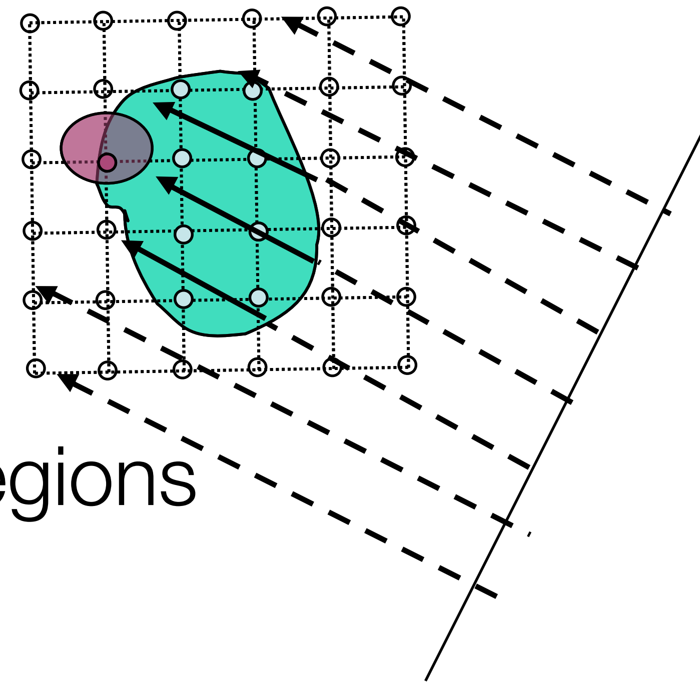
Acceleration Techniques for Ray Casting

- Problem: ray casting is time consuming
- Idea:
  - Neglect “irrelevant” information to accelerate the rendering process
  - Exploit coherence



- Early-ray termination
  - Idea: colors from faraway regions do not contribute if accumulated opacity is to high
  - Stop traversal if contribution of sample becomes irrelevant
  - User-set opacity level for termination
  - Front-to-back compositing

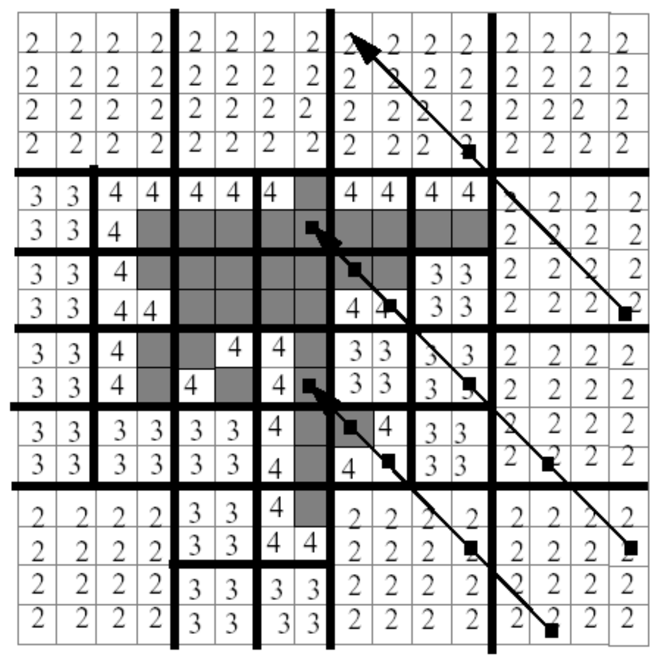
Acceleration Techniques for Ray Casting



- Space leaping
  - Skip empty cells
- Homogeneity-acceleration
  - Approximate homogeneous regions with fewer sample points

Acceleration Techniques for Ray Casting

- Hierarchical spatial data structure
  - Octree
  - Mean value and variance stored in nodes of octree



Acceleration Techniques for Ray Casting

- Modern GPUs can be used for ray casting
- Essential idea
  - Fragment shader loop
  - Implements ray marching
- Benefits from
  - High processing speed of GPUs
  - Built-in trilinear interpolation in 3D textures
- Requires Pixel Shader 3.0 compliant GPUs

Overview

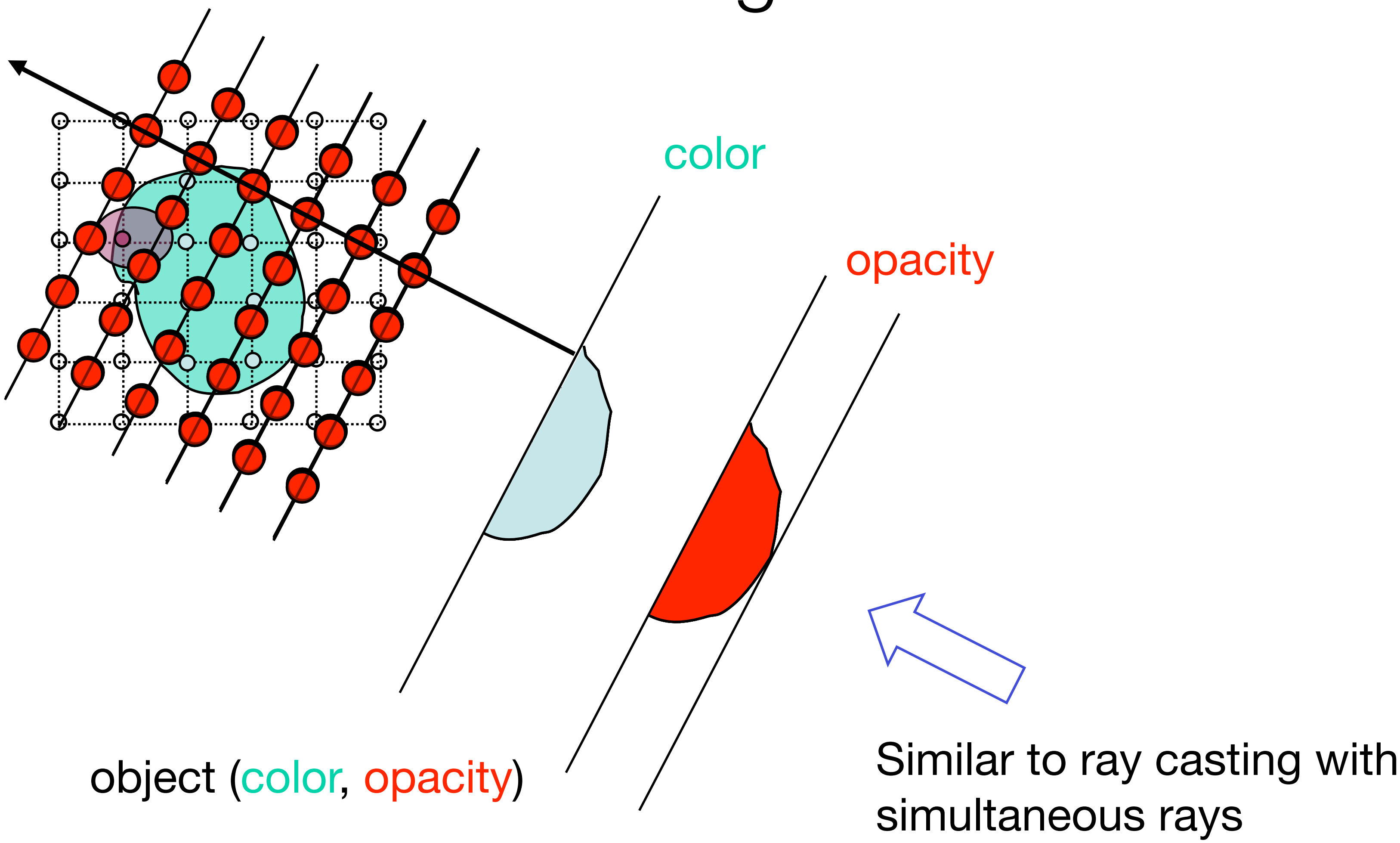
- Light: Volume rendering equation
- Discretized: Compositing schemes
- Ray casting
  - Acceleration techniques for ray casting
- Texture-based volume rendering
- Shear-warp factorization
- Splatting
- Fourier Volume Rendering
- Cell projection (Shirley-Tuchman)

Texture-Based Volume Rendering

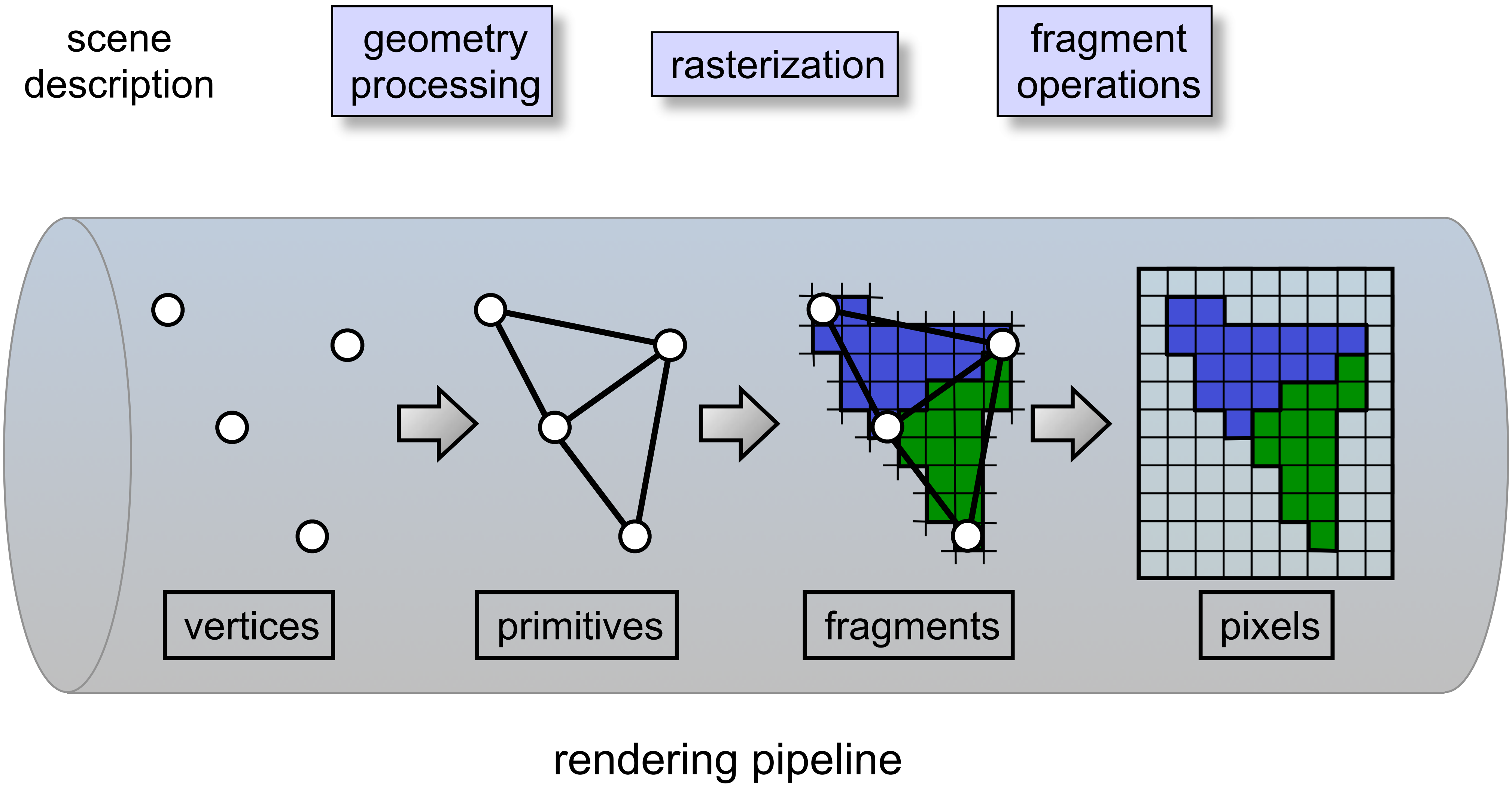
- Object-space approach
- Based on graphics hardware:
  - Rasterization
  - Texturing
  - Blending
- Proxy geometry because there are no volumetric primitives in graphics hardware
- Slices through the volume
- Supported by older graphics hardware
  - No need for (advanced) fragment shaders

Texture-Based Volume Rendering

- Slice-based rendering

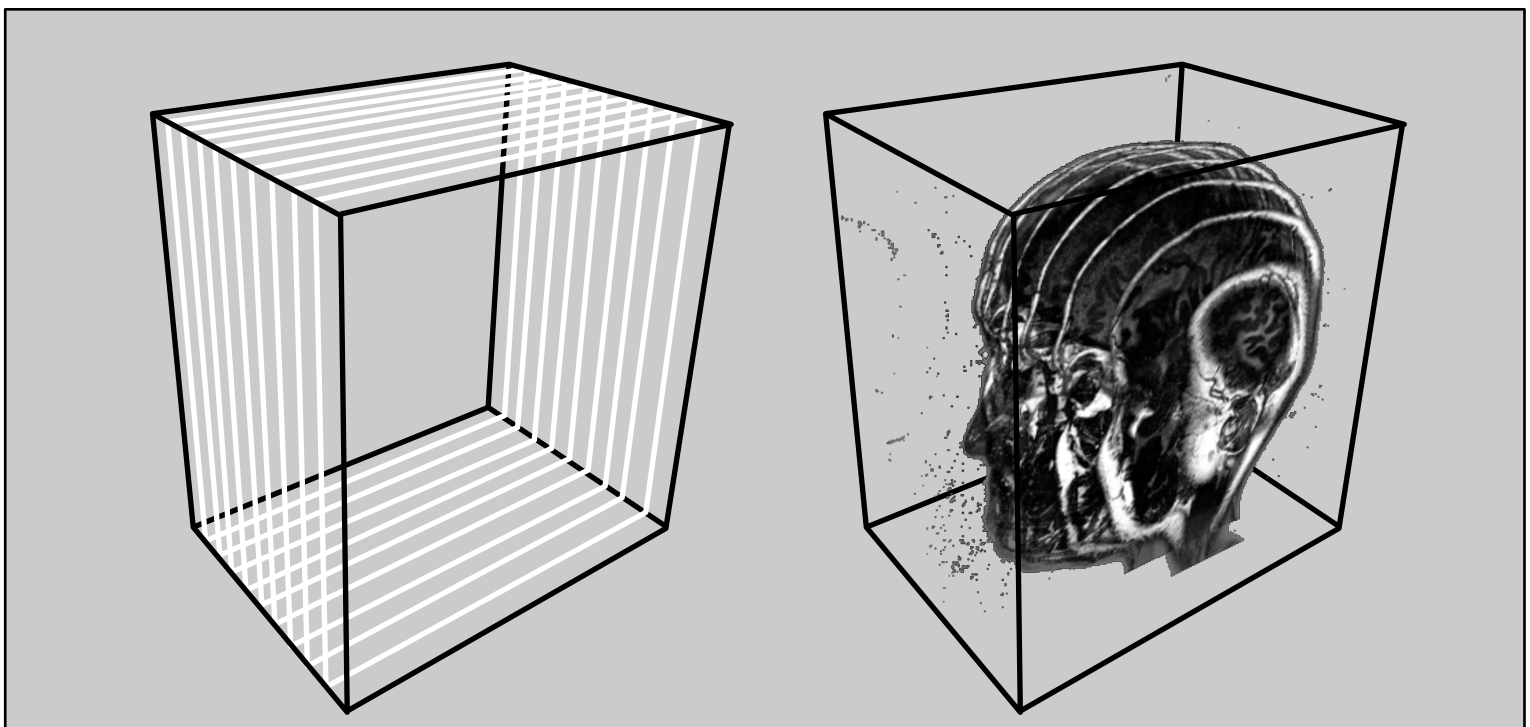


Texture-Based Volume Rendering



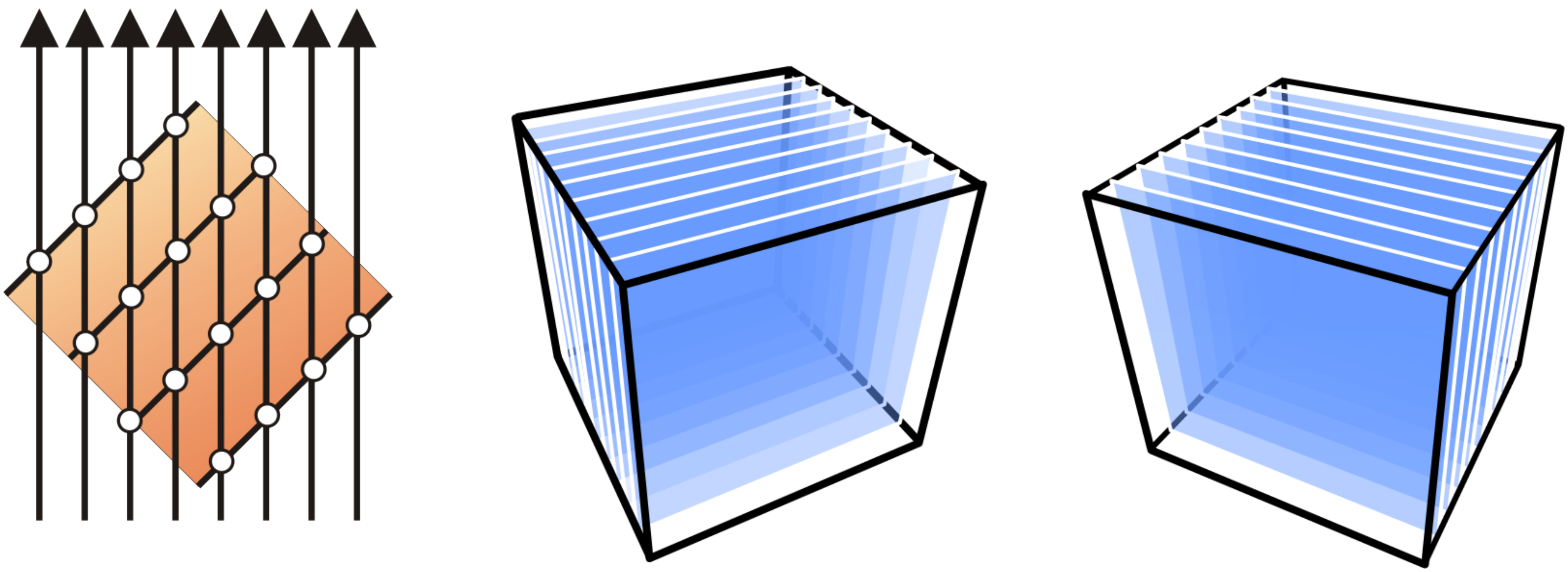
Texture-Based Volume Rendering

- Proxy geometry
  - Stack of texture-mapped slices
  - Generate fragments
  - Most often back-to-front traversal



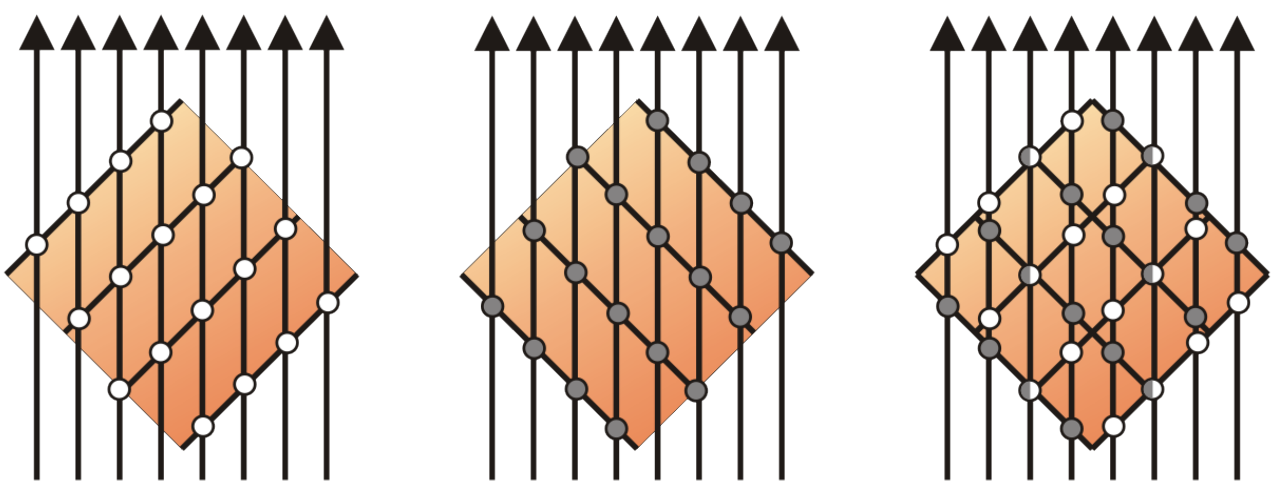
Texture-Based Volume Rendering

- 2D textured slices
  - Object-aligned slices
  - Three stacks of 2D textures
  - Bilinear interpolation



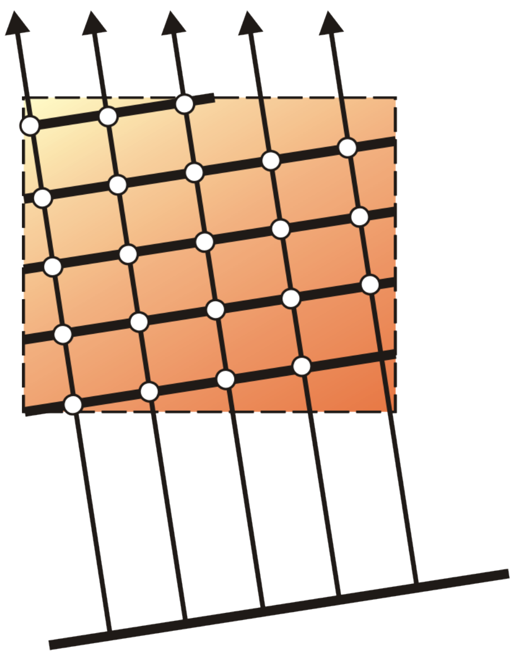
Texture-Based Volume Rendering

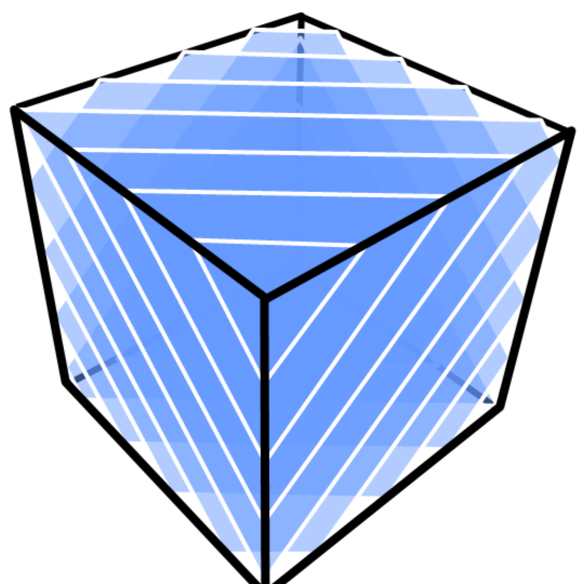
- Stack of 2D textures:
  - Artifacts when stack is viewed close to 45 degrees
    - Locations of sampling points may change abruptly

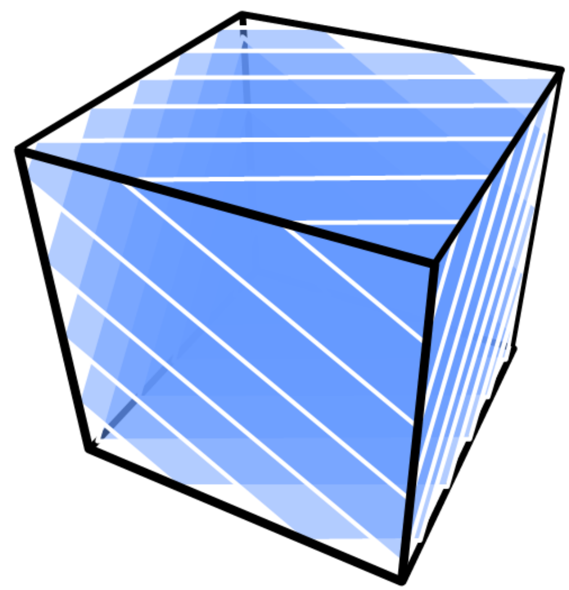


Texture-Based Volume Rendering

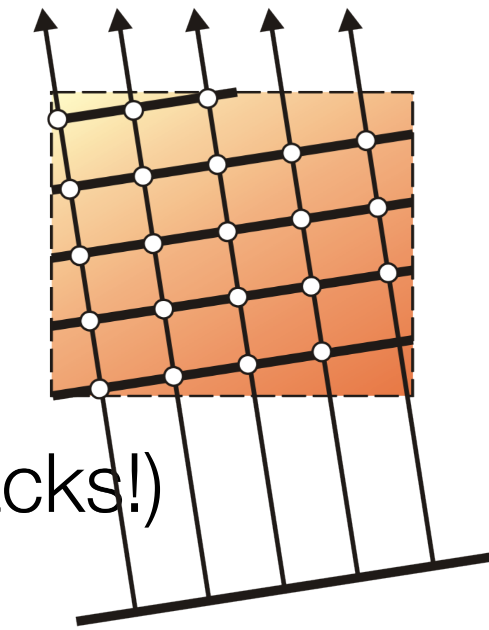
- 3D textured slices
  - View-aligned slices
  - Single 3D texture
  - Trilinear interpolation



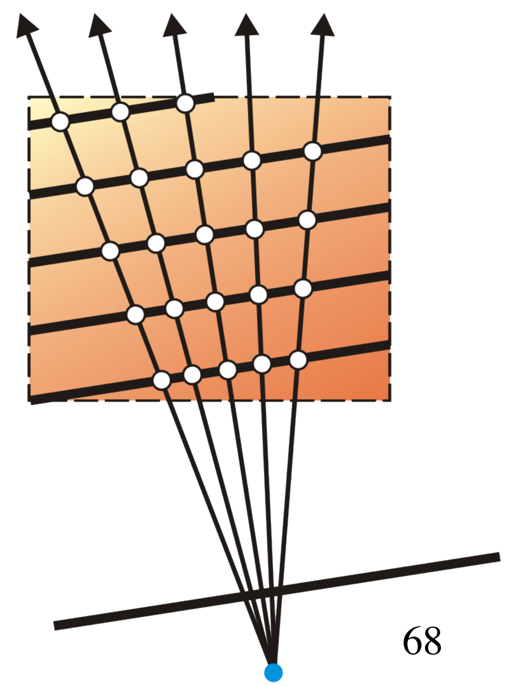




Texture-Based Volume Rendering

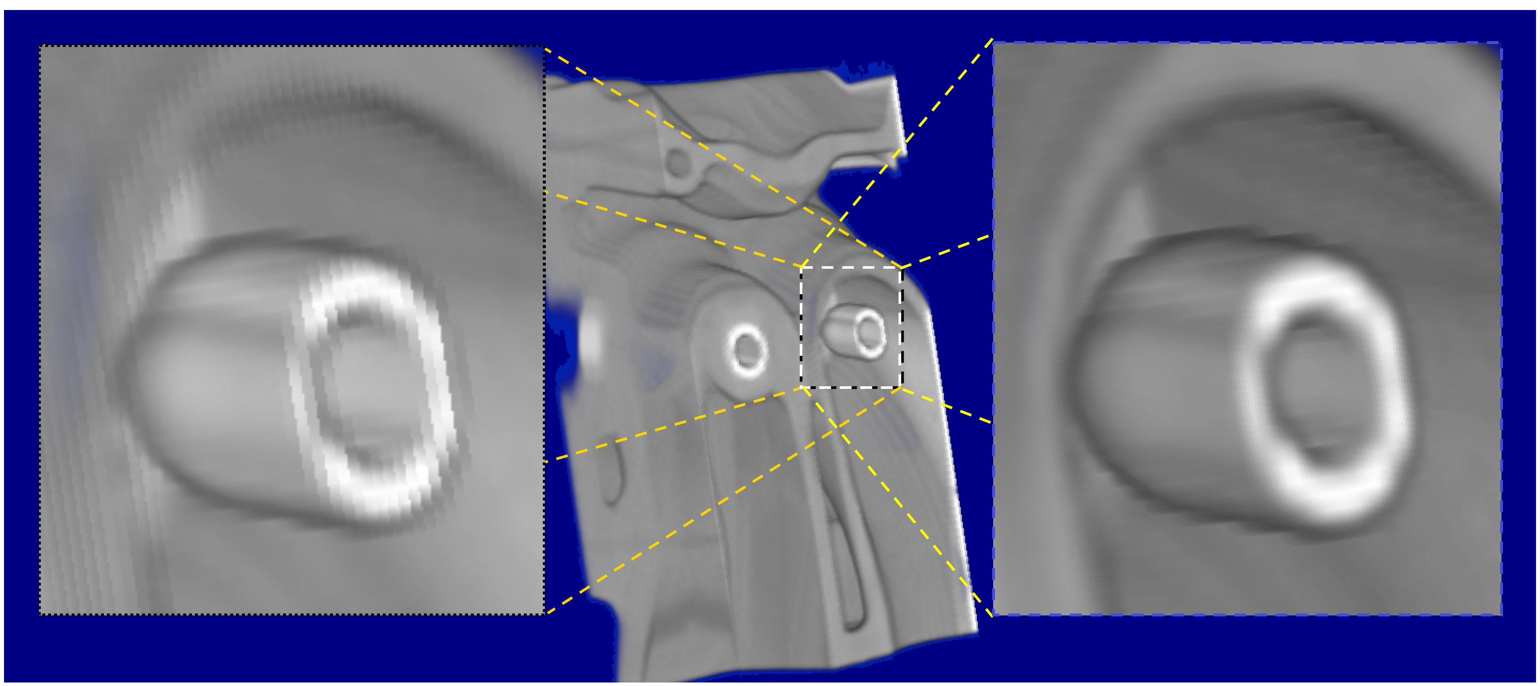


- 3D texture:
  - Needs support for 3D textures
  - Data set stored only once (not 3 stacks!)
  - Trilinear interpolation within volume
    - Slower
    - Good image quality
  - Constant Euclidean distance between slices along a light ray
    - Constant sampling distance (except for perspective projection)

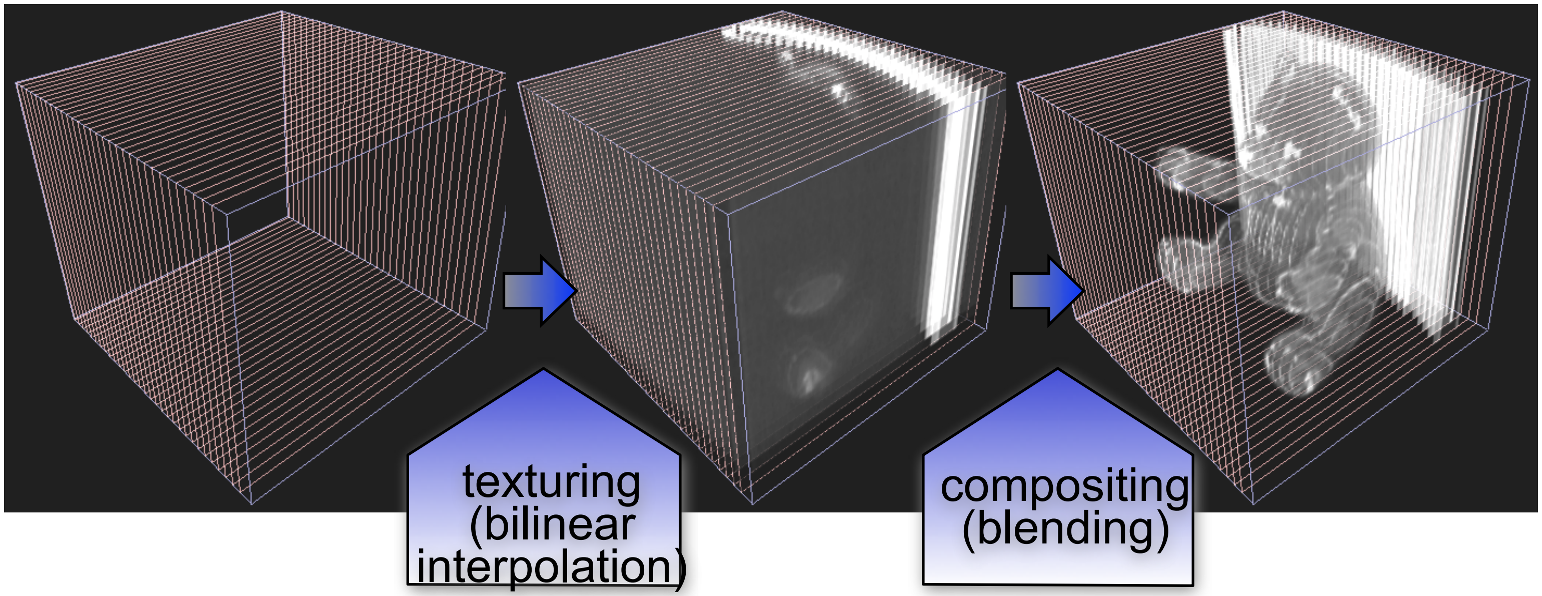


Texture-Based Volume Rendering

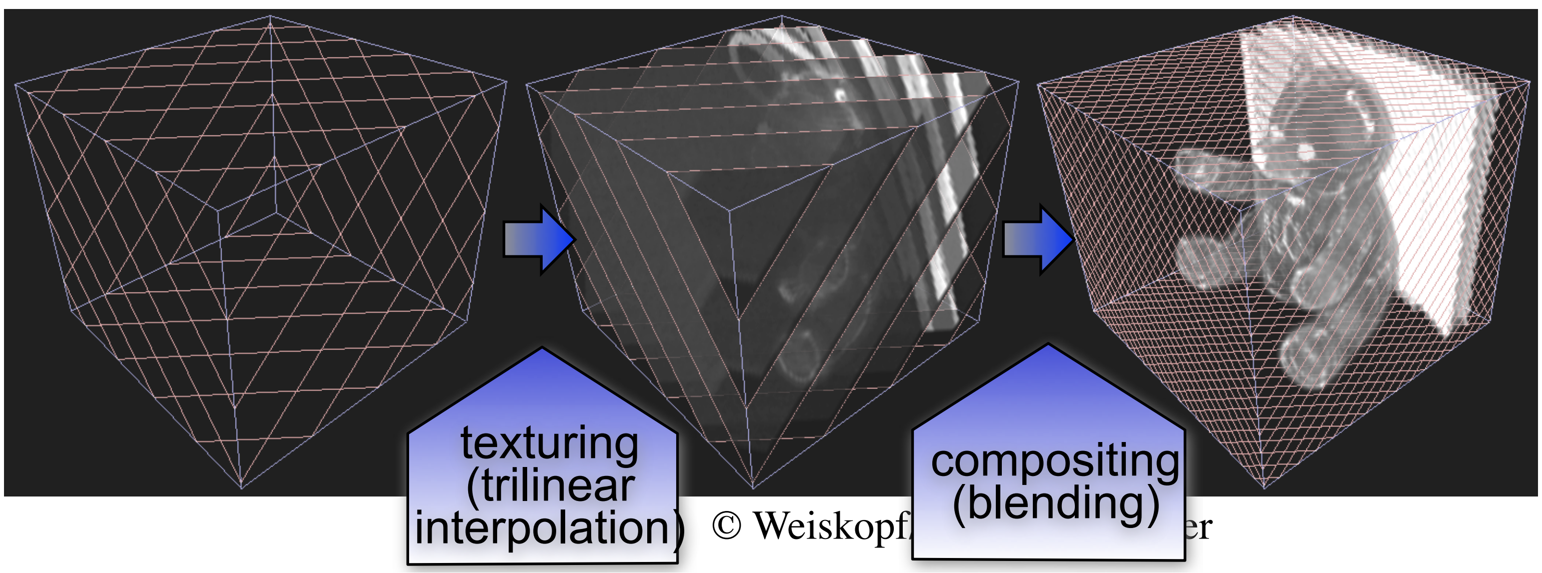
- 3D texture:
  - No artifacts due to inappropriate viewing angles
  - Increase sampling rate → more slices
    - Easy with 3D textures



Texture-Based Volume Rendering



2D textures axis-aligned



3D texture view-aligned

Texture-Based Volume Rendering

- Representation of volume data by textures
  - Stack of 2D textures
  - 3D texture
- Typical choices for texture format:
  - Luminance and alpha
    - Pre-classified (pre-shaded) gray-scale volume rendering
    - Transfer function is already applied to scalar data
    - Change of transfer func. requires complete redefinition of texture data
  - RGBA
    - Pre-classified (pre-shaded) colored volume rendering
    - Transfer function is already applied to scalar data
  - Luminance
    - Only the actual scalar data is stored
    - Best solution!

Texture-Based Volume Rendering

- Post-classification?
  - Data set represented by luminance texture (single channel)
  - Dependent texture lookup in texture for color table
  - Fragment or pixel shader program

Texture-Based Volume Rendering

- Compositing:
  - Works on fragments
  - Per-fragment operations
  - After rasterization
  - Blending of fragments via over operator
  - OpenGL code for over operator

```
glEnable (GL_BLEND);
glBlendFunc (GL_ONE, GL_ONE_MINUS_SRC_ALPHA);
```

- Generate fragments:
  - Render proxy geometry
  - Slice
  - Simple implementation: quadrilateral
  - More sophisticated: triangulated intersection surface between slice plane and boundary of the volume data set

Texture-Based Volume Rendering

- Advantages of texture-based rendering:
  - Supported by consumer graphics hardware
  - Fast for moderately sized data sets
  - Interactive explorations
  - Surface-based and volumetric representations can easily be combined
    → mixture with opaque geometries
- Disadvantages:
  - Limited by texture memory
    → Solution: bricking at the cost of additional texture downloads to the graphics board
  - Brute force: complete volume is represented by slices
  - Rasterization speed + memory access can be problematic

Overview

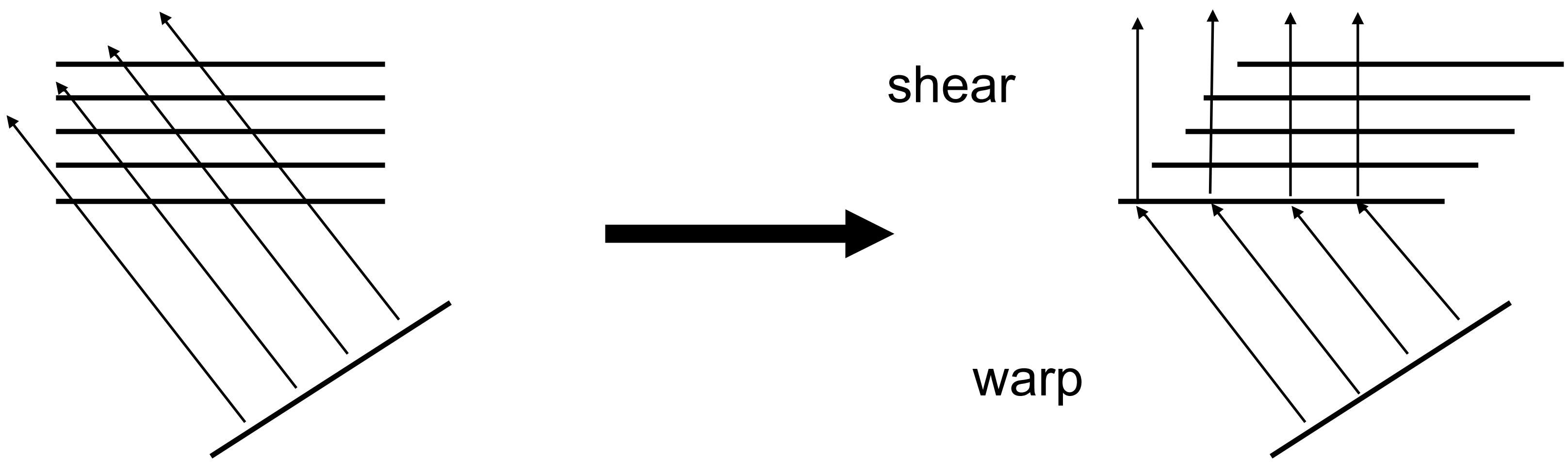
- Light: Volume rendering equation
- Discretized: Compositing schemes
- Ray casting
  - Acceleration techniques for ray casting
- Texture-based volume rendering
- Shear-warp factorization
- Splatting
- Fourier Volume Rendering
- Cell projection (Shirley-Tuchman)

Shear-Warp Factorization

- Object-space method
- Slice-based technique
- Fast object-order rendering
- Accelerated volume visualization via shear-warp factorization [Lacroute & Levoy 1994]
- CPU-based implementation

Shear-Warp Factorization

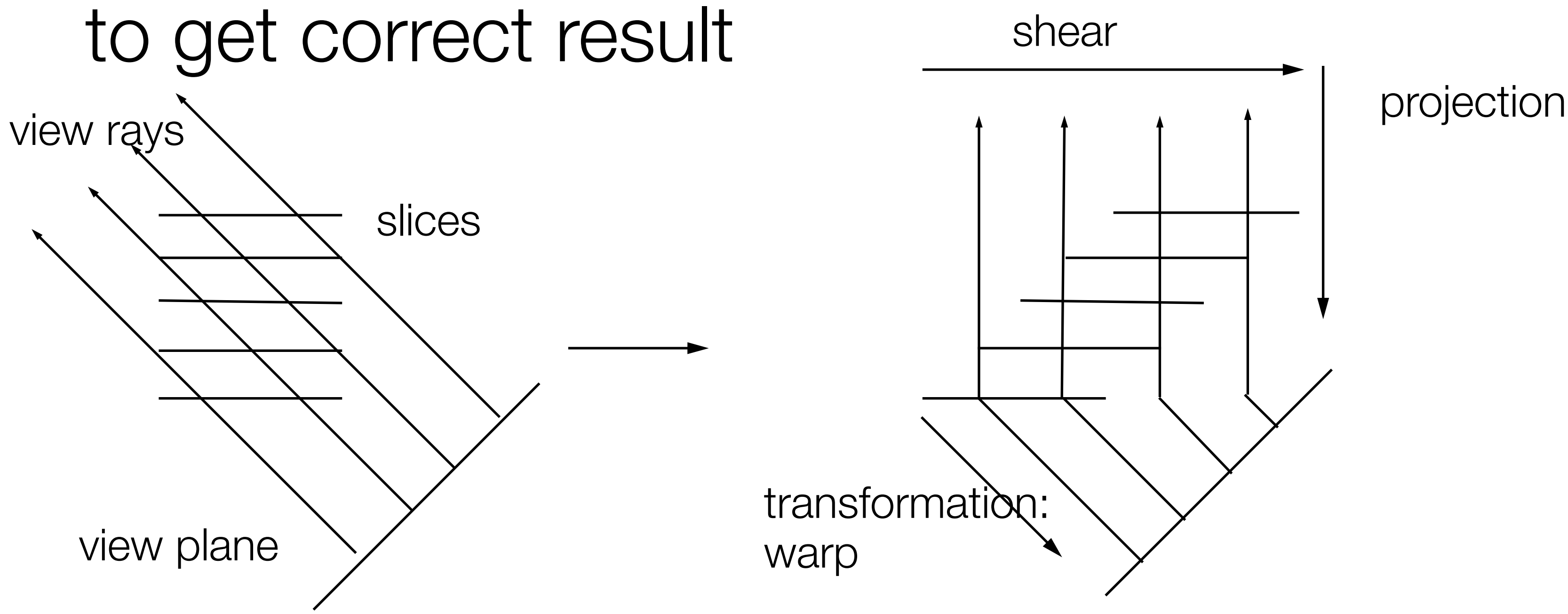
- General goal: make viewing rays parallel to each other and perpendicular to the image
- This is achieved by a simple shear



- Parallel projection (orthographic camera) is assumed

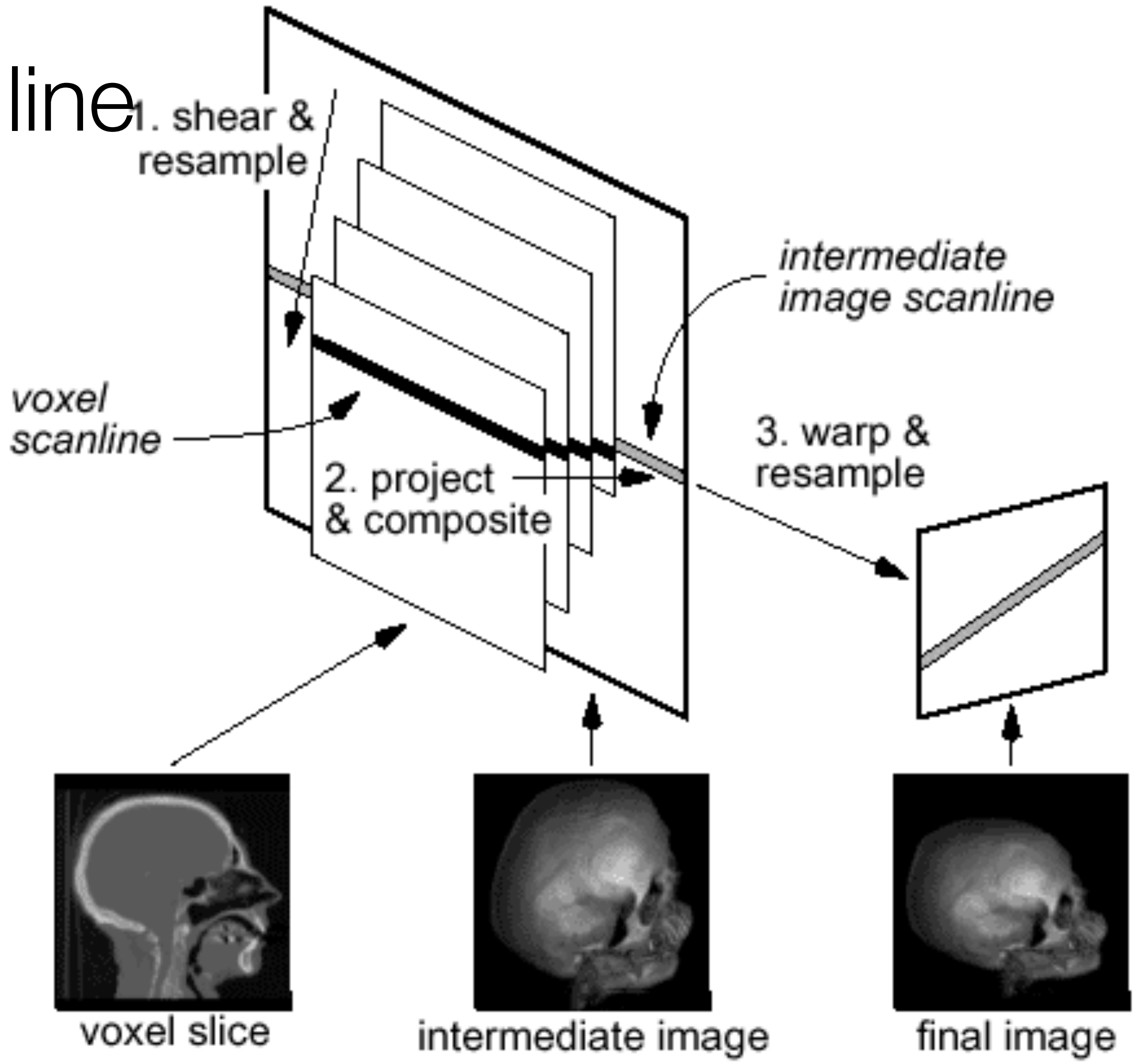
Shear-Warp Factorization

- Algorithm:
  - Shear along the volume slices
  - Projection + comp. to get intermediate image
  - Warping transformation of intermediate image to get correct result



Shear-Warp Factorization

- For one scan line



Shear-Warp Factorization

- Mathematical description of the shear-warp factorization
- Splitting the viewing transformation into separate parts

$$\mathbf{M}_{\text{view}} = \mathbf{P} \times \mathbf{S} \times \mathbf{M}_{\text{warp}}$$

  - $\mathbf{M}_{\text{view}}$ = general viewing matrix
  - $\mathbf{P}$ = permutation matrix: transposes coord. system in order to make the *z*-axis the principal viewing axis
  - $\mathbf{S}$ = transforms volume into sheared object space
  - $\mathbf{M}_{\text{warp}}$ = warps sheared object coordinates into image coordinates
- Needs 3 stacks of the volume along 3 principal axes

Shear-Warp Factorization

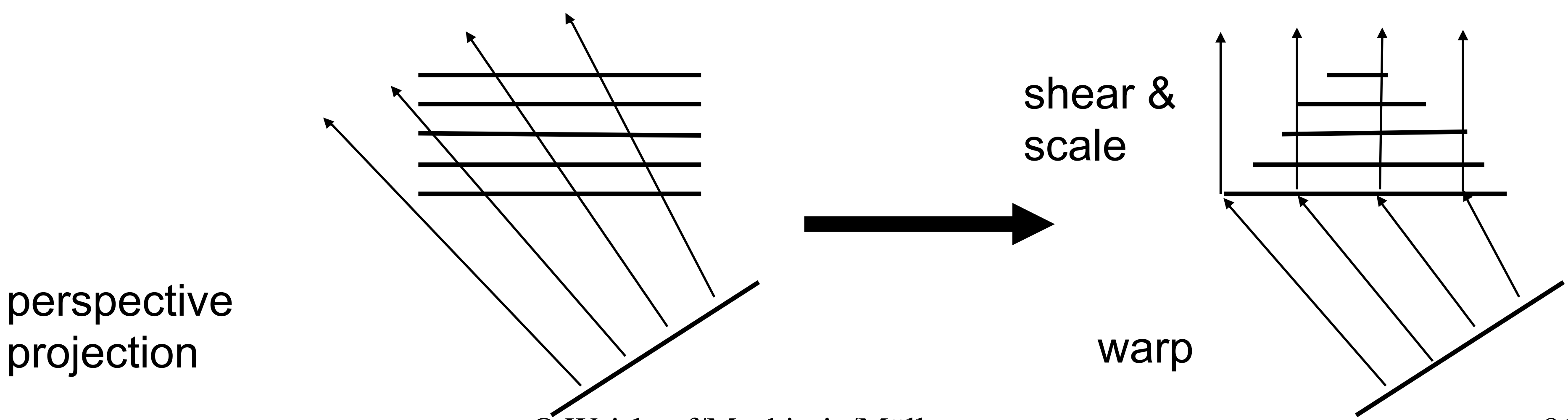
- Shear for parallel and perspective proj.

$$S_{\text{par}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ s_x & s_y & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

shear perpendicular to *z*-axis

$$S_{\text{persp}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ s'_x & s'_y & 1 & s'_w \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

shear and scale



Shear-Warp Factorization

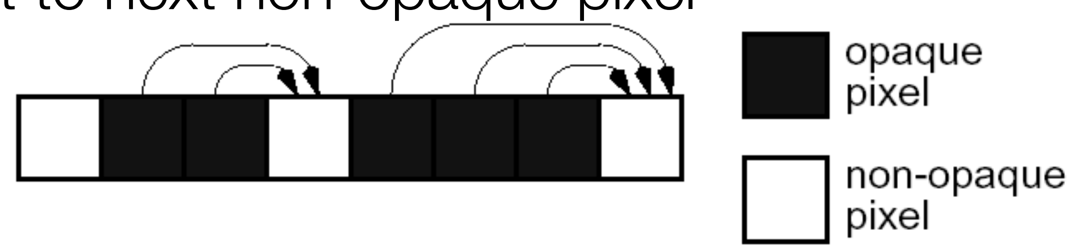
- Algorithm (detailed):
  - Transform volume to sheared object space by translation and resampling
  - Project volume into 2D intermediate image in sheared object space
    - Composite resampled slices front-to-back
  - Transform intermediate image to image space using 2D warping
- In a nutshell:
  - Shear (3D)
  - Project (3D → 2D)
  - Warp (2D)

Shear-Warp Factorization

- Three properties
  - Scan lines of pixels in the intermediate image are parallel to scan lines of voxels in the volume data
  - All voxels in a given voxel slice are scaled by the same factor
  - Parallel projections only:
    Every voxel slice has the same scale factor
- Scale factor for parallel projections
  - This factor can be chosen arbitrarily
  - Choose a unity scale factor so that for a given voxel scan line there is a one-to-one mapping between voxels and intermediate image pixels

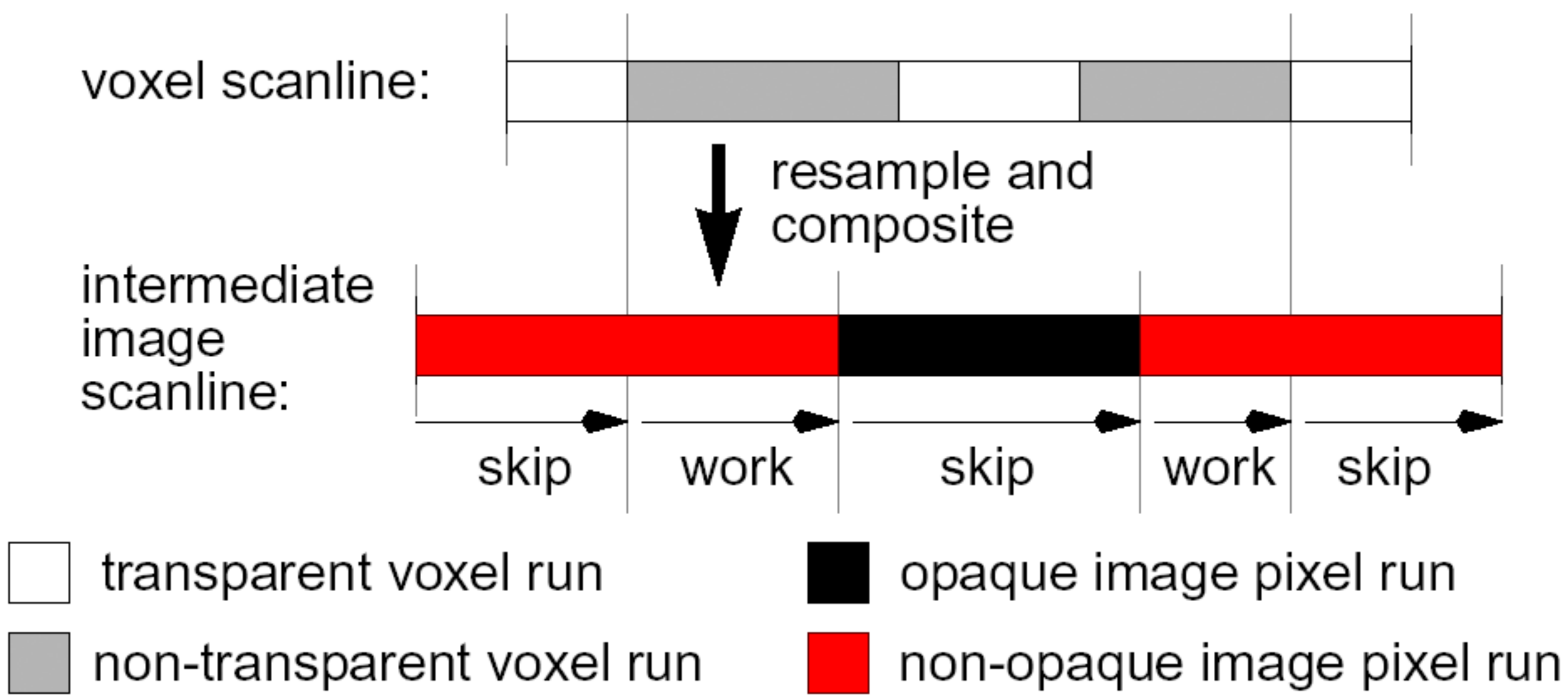
Shear-Warp Factorization

- Highly optimized algorithm for
  - Parallel projection
  - Fixed opacity transfer function
- Optimization of volume data (voxel scan lines)
  - Run-length encoding of voxel scan lines
  - Skip runs of transparent voxels
  - Transparency and opaqueness determined by user-defined opacity threshold
- Optimization in intermediate image:
  - Skip opaque pixels in intermediate image (early-ray termination)
  - Store (in each pixel) offset to next non-opaque pixel



Shear-Warp Factorization

- Combining both ideas:
  - First property (parallel scan lines for pixels and voxels): Voxel scan lines in sheared volume are aligned with pixel scan lines in intermediate
  - Both can be traversed in scan line order simultaneously



Shear-Warp Factorization

- Coherence in voxel space:
  - Each slice of the volume is only translated
  - Fixed weights for bilinear interpolation within voxel slices
  - Computation of weights only once per frame
- Final warping:
  - Works on composited intermediate image
  - Warp: affine image warper with bilinear filter
  - Often done in hardware:
    render a quadrilateral with intermediate 2D image being attached as 2D texture

Shear-Warp Factorization

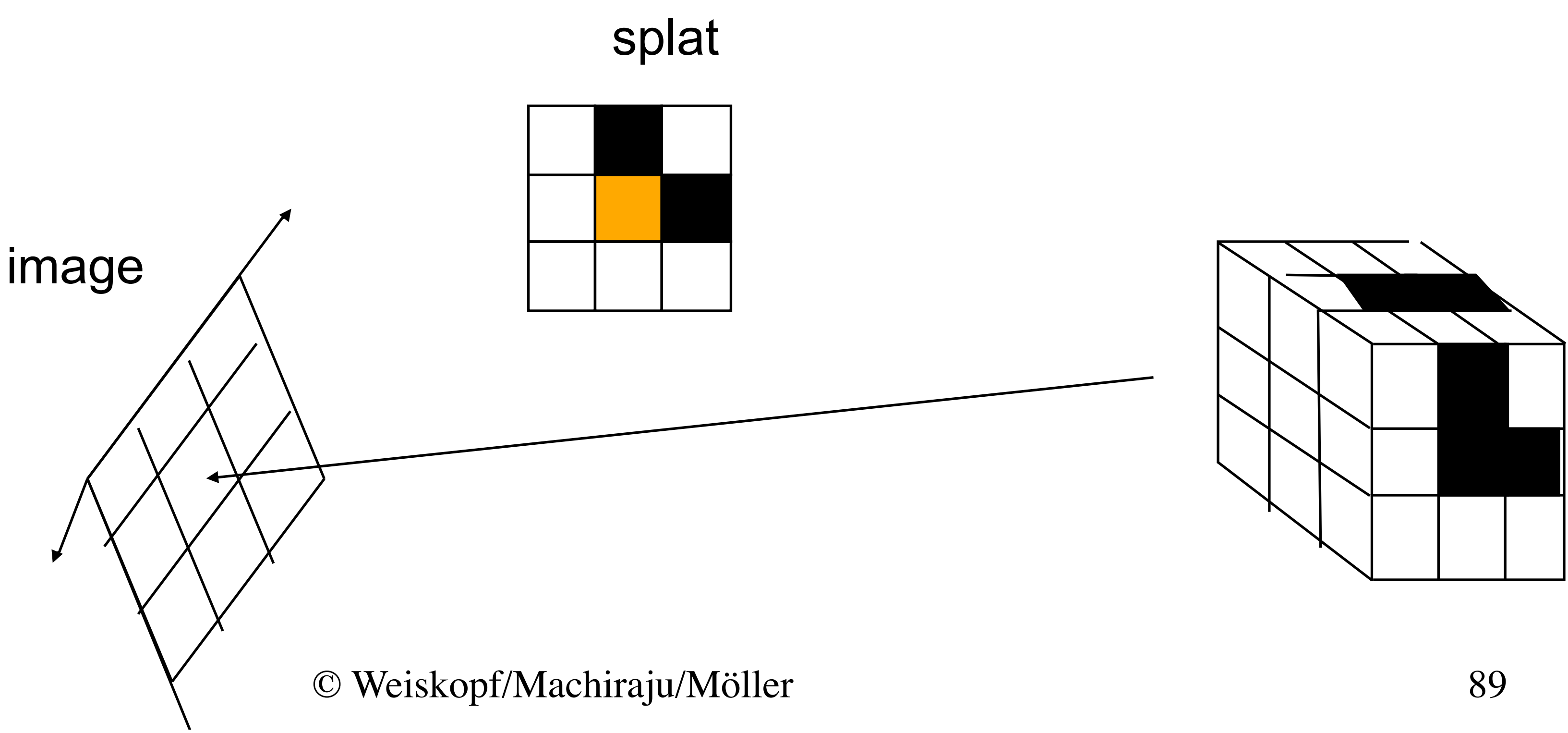
- Parallel projection:
  - Efficient reconstruction
  - Lookup table for shading
  - Lookup table for opacity correction (thickness)
  - Three RLE of the actual volume (in x, y, z)
- Perspective projection:
  - Similar to parallel projection
  - Difference: voxels need to be scaled
  - Hence more then two voxel scan lines needed for one image scan line

Overview

- Light: Volume rendering equation
- Discretized: Compositing schemes
- Ray casting
  - Acceleration techniques for ray casting
- Texture-based volume rendering
- Shear-warp factorization
- Splatting
- Fourier Volume Rendering
- Cell projection (Shirley-Tuchman)

Splatting

- Splatting [Westover 1990]
- Object-order method
- Project each sample (voxel) from the volume into the image plane



Splatting

- Ideally we would reconstruct the continuous volume (cloud) using the interpolation kernel *w* (spherically symmetric):

$$f_r(v) = \sum_k w(v - v_k) f(v_k)$$

- Analytic integral along a ray *r* for intensity (emission):

$$I(p) = \int f_r(p + r)\, dr = \int \sum_k w(p + r - v_k) f(v_k)\, dr$$

- Rewrite:

$$I(p) = \sum_k f(v_k) \times \int w(p + r - v_k)\, dr$$

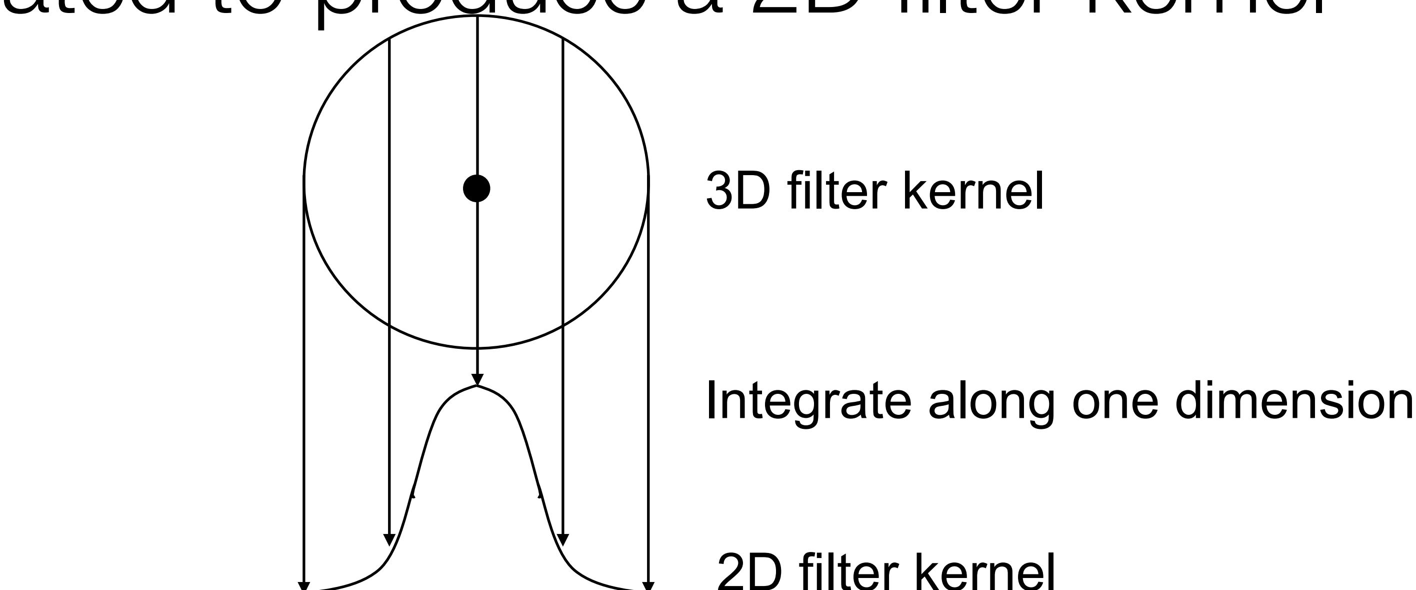
↑ splatting kernel (= "splat")

Splatting

- Discretization via 2D splats

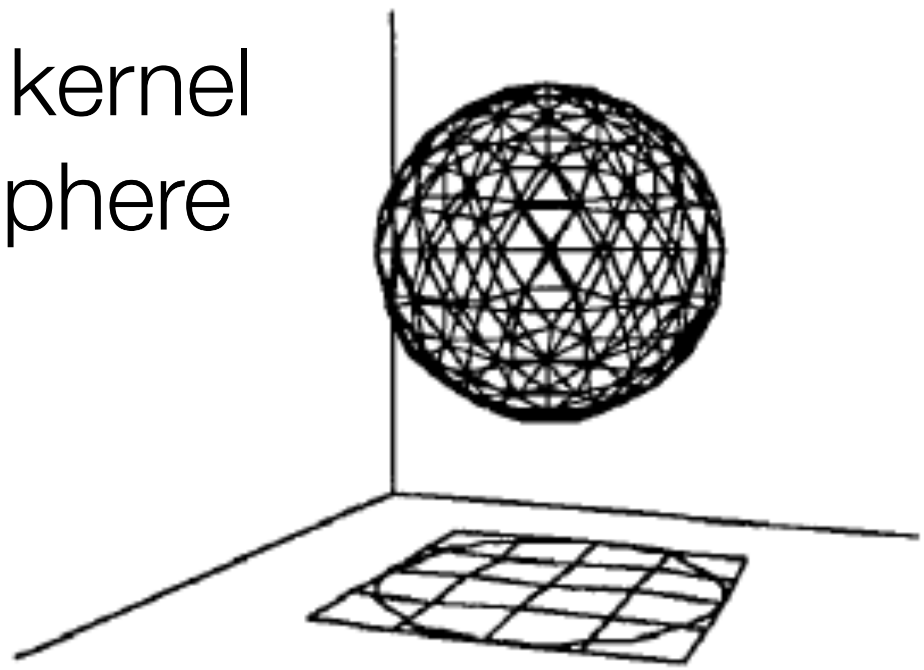
  $$\text{Splat}(x,y) = \int w(x,y,z)\,dz$$

  from the original 3D kernel
- The 3D rotationally symmetric filter kernel is integrated to produce a 2D filter kernel



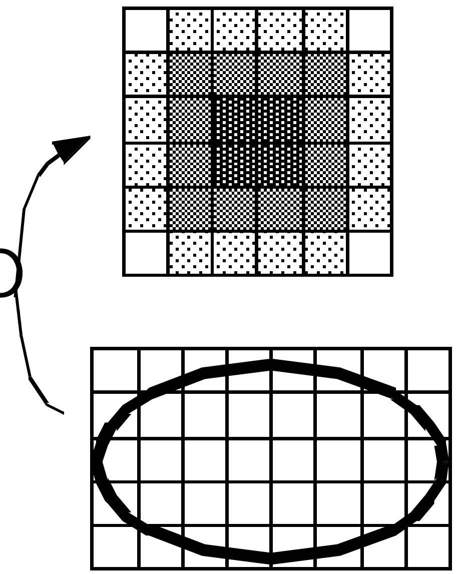
Splatting

- Draw each voxel as a cloud of points (footprint) that spreads the voxel contribution across multiple pixels
- Footprint: splatted (integrated) kernel
- Approximate the 3D kernel $h(x,y,z)$ extent by a sphere

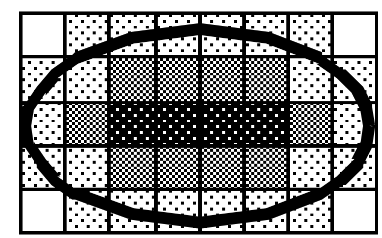


Splatting

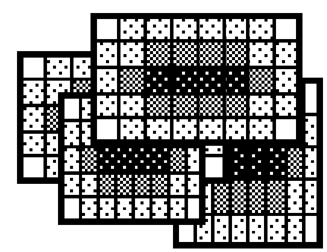
- Larger footprint increases blurring and used for high pixel-to-voxel ratio



- Footprint geometry
  - Orthographic projection: footprint is independent of the view point
  - Perspective projection: footprint is elliptical



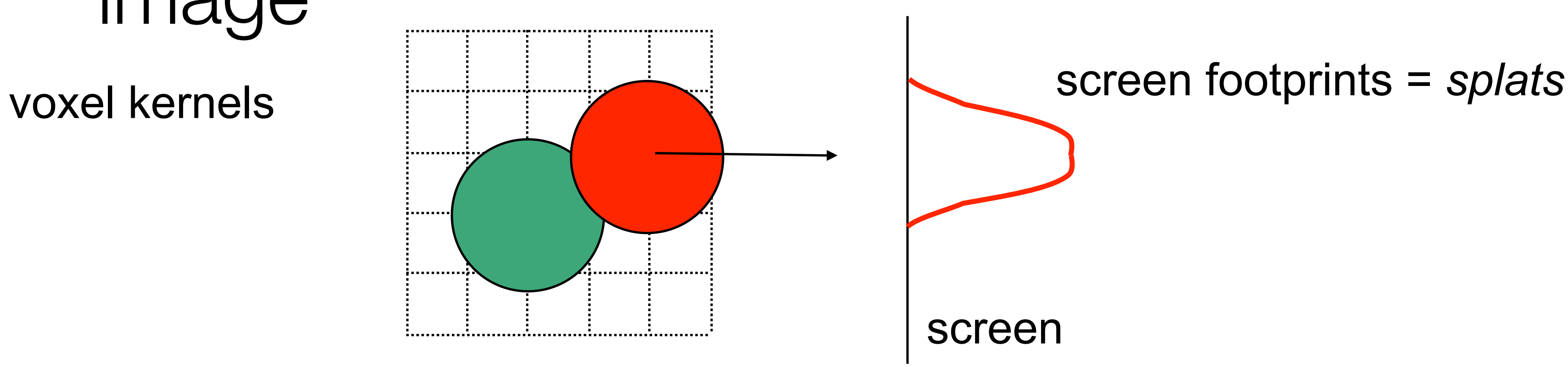
- Pre-integration of footprint



- For perspective projection: additional computation of the orientation of the ellipse

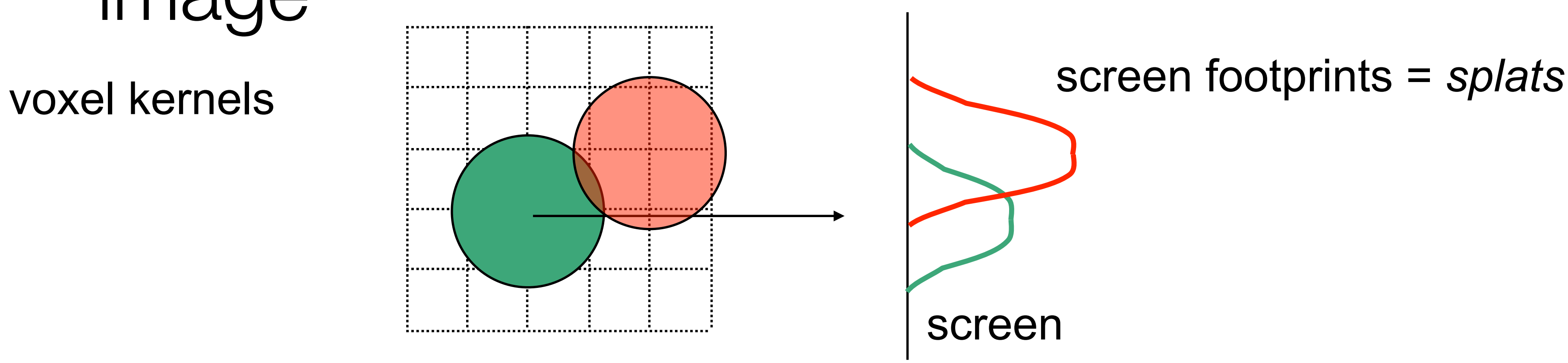
Splatting

- Volume = field of 3D interpolation kernels
  - One kernel at each grid voxel
- Each kernel leaves a 2D footprint on screen
- Weighted footprints accumulate into image



Splatting

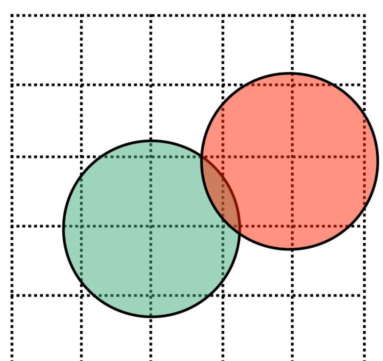
- Volume = field of 3D interpolation kernels
  - One kernel at each grid voxel
- Each kernel leaves a 2D footprint on screen
- Weighted footprints accumulate into image

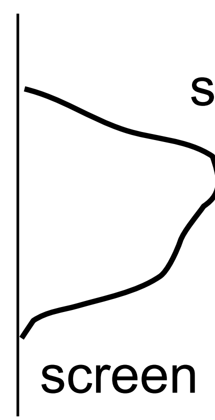


Splatting

- Volume = field of 3D interpolation kernels
  – One kernel at each grid voxel
- Each kernel leaves a 2D footprint on screen
- Weighted footprints accumulate into image

voxel kernels

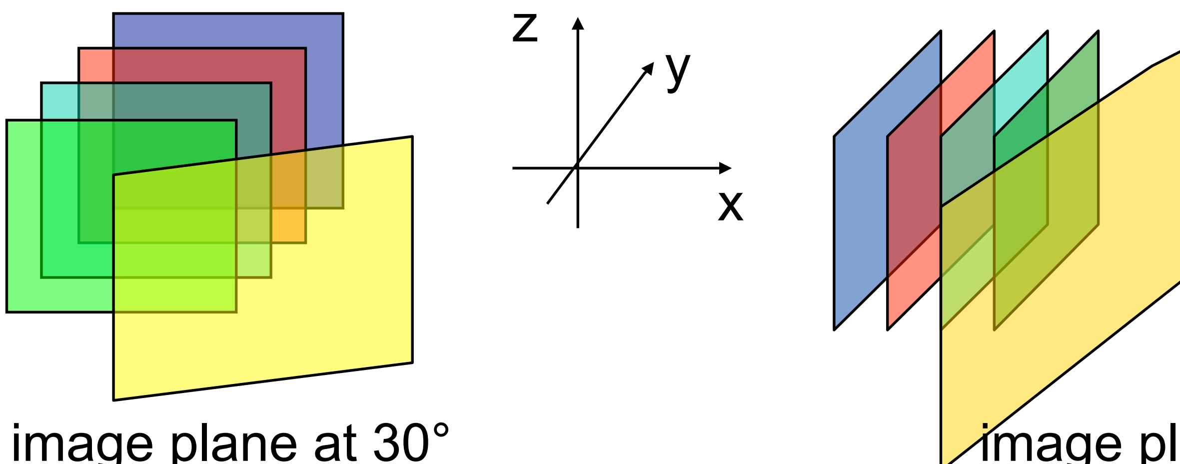




screen footprints = *splats*

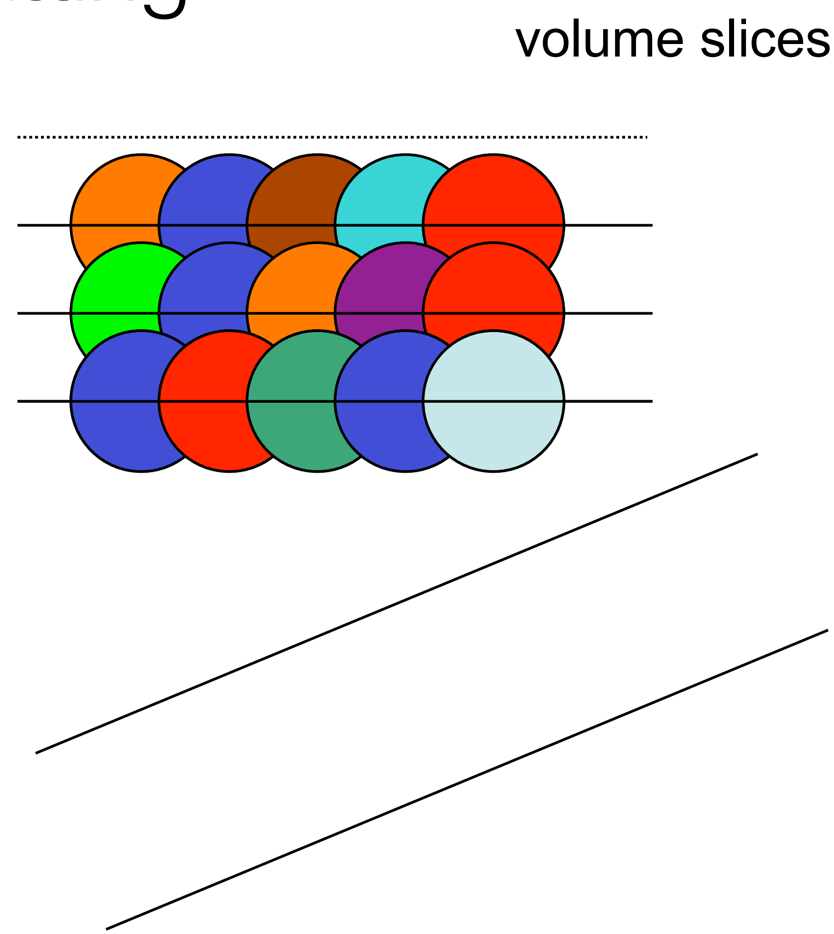
Splatting

- Voxel kernels are added within sheets
- Sheets are composited front-to-back
- Sheets = volume slices most perpendicular to the image plane (analogously to stack of slices)



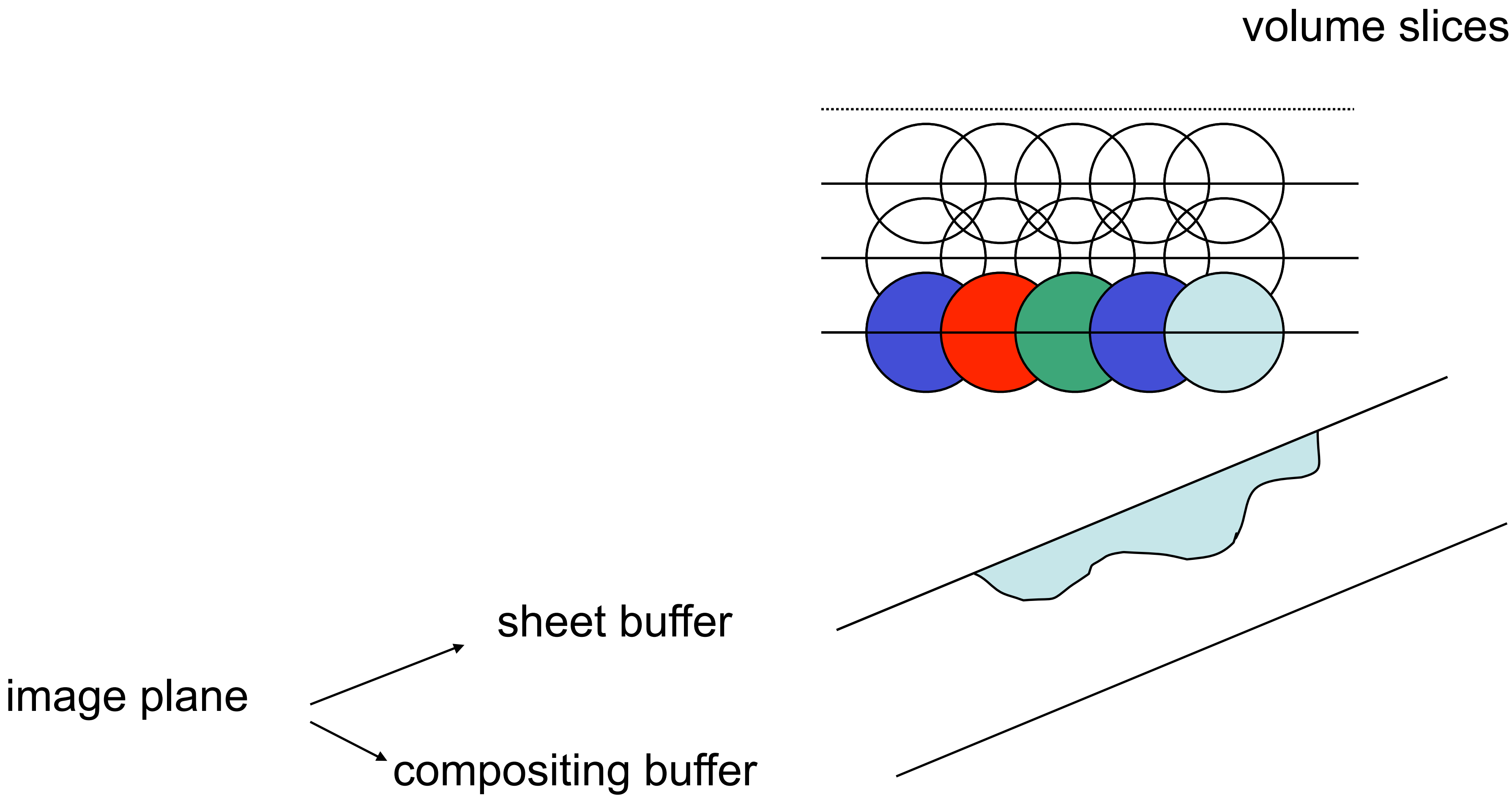
Splatting

- Core algorithm for splatting
- Volume
  - Represented by voxels
  - Slicing
- Image plane:
  - Sheet buffer
  - Compositing buffer



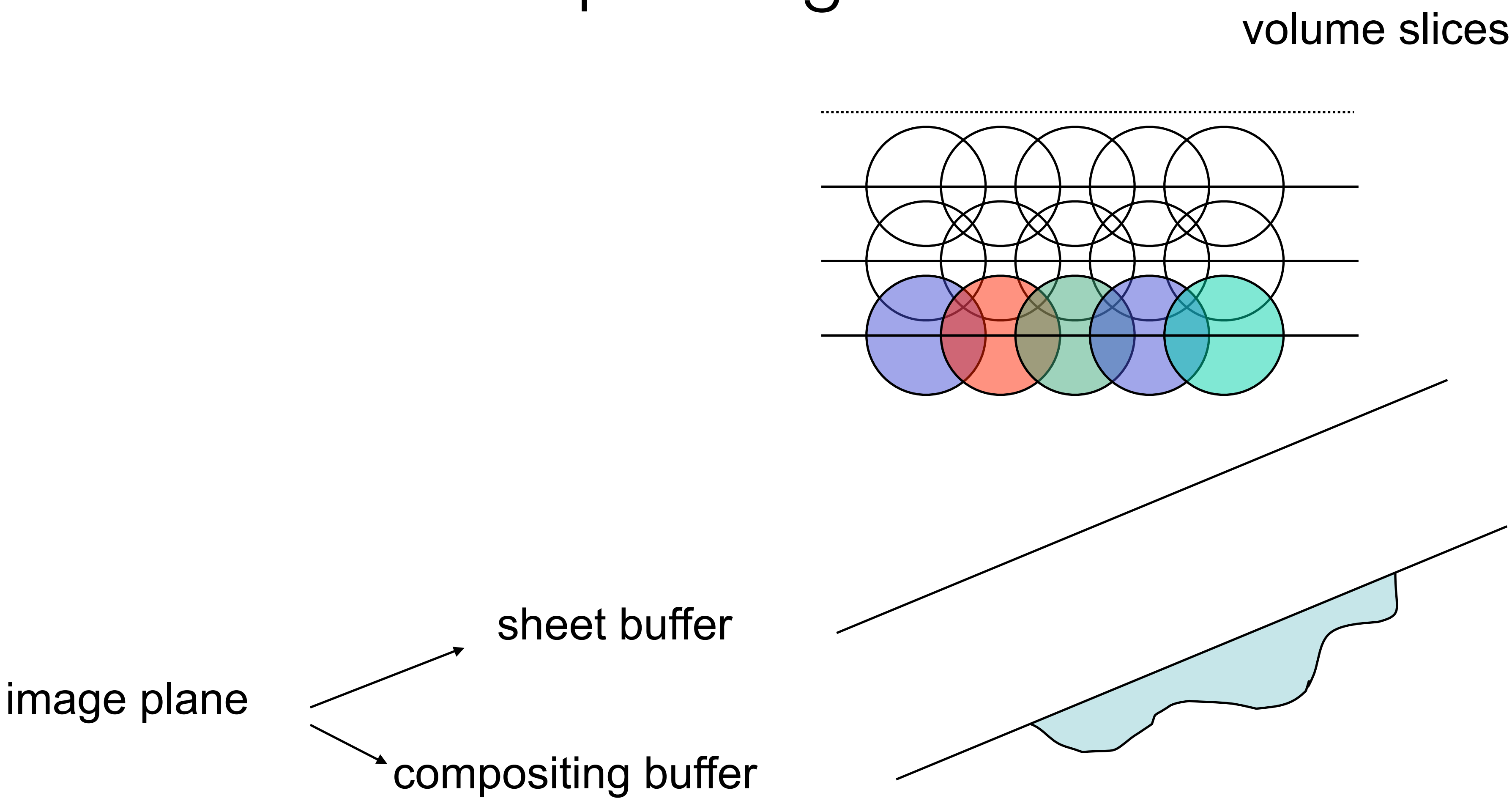
Splatting

- Add voxel kernels within first sheet



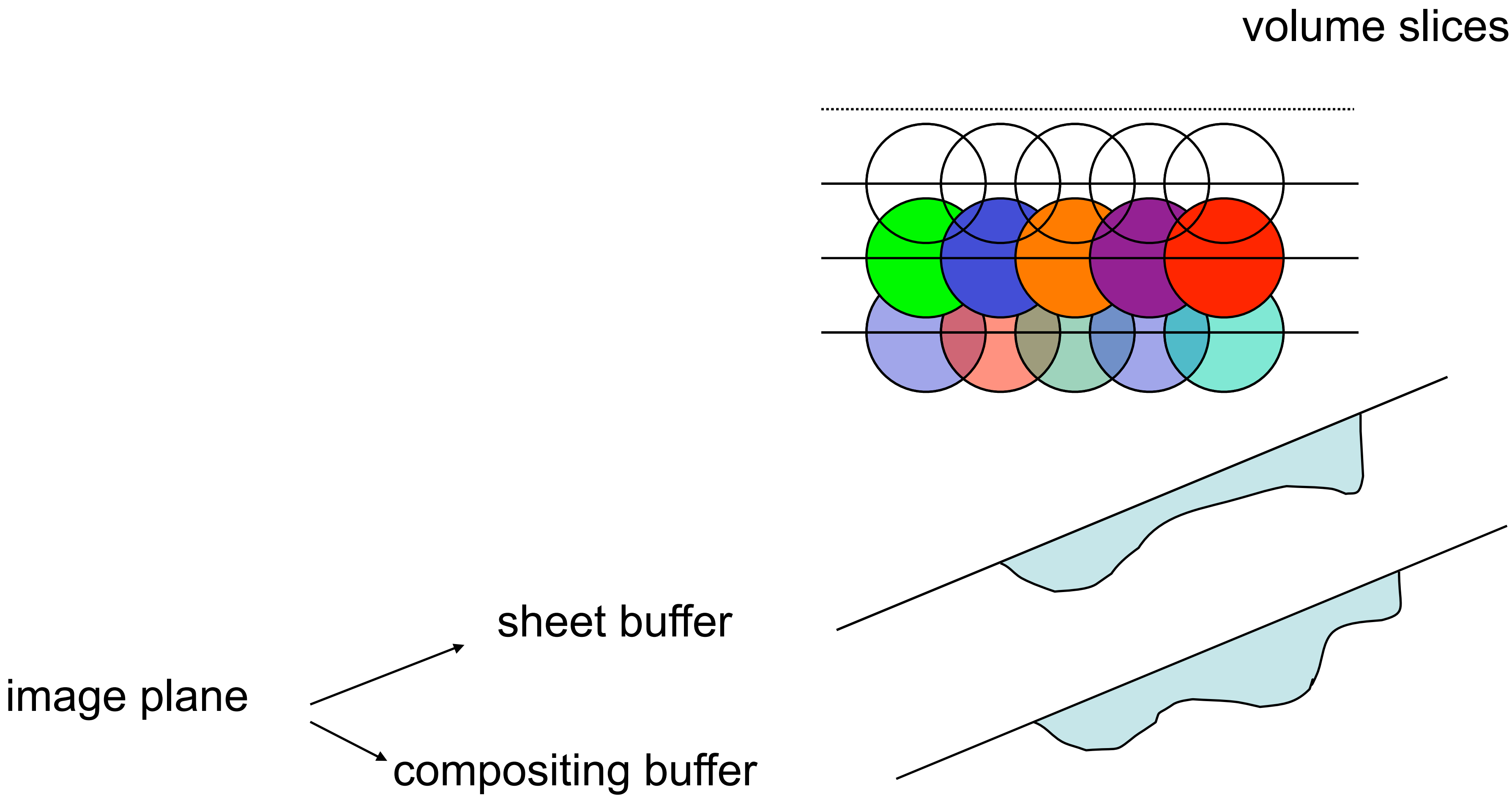
Splatting

- Transfer to compositing buffer



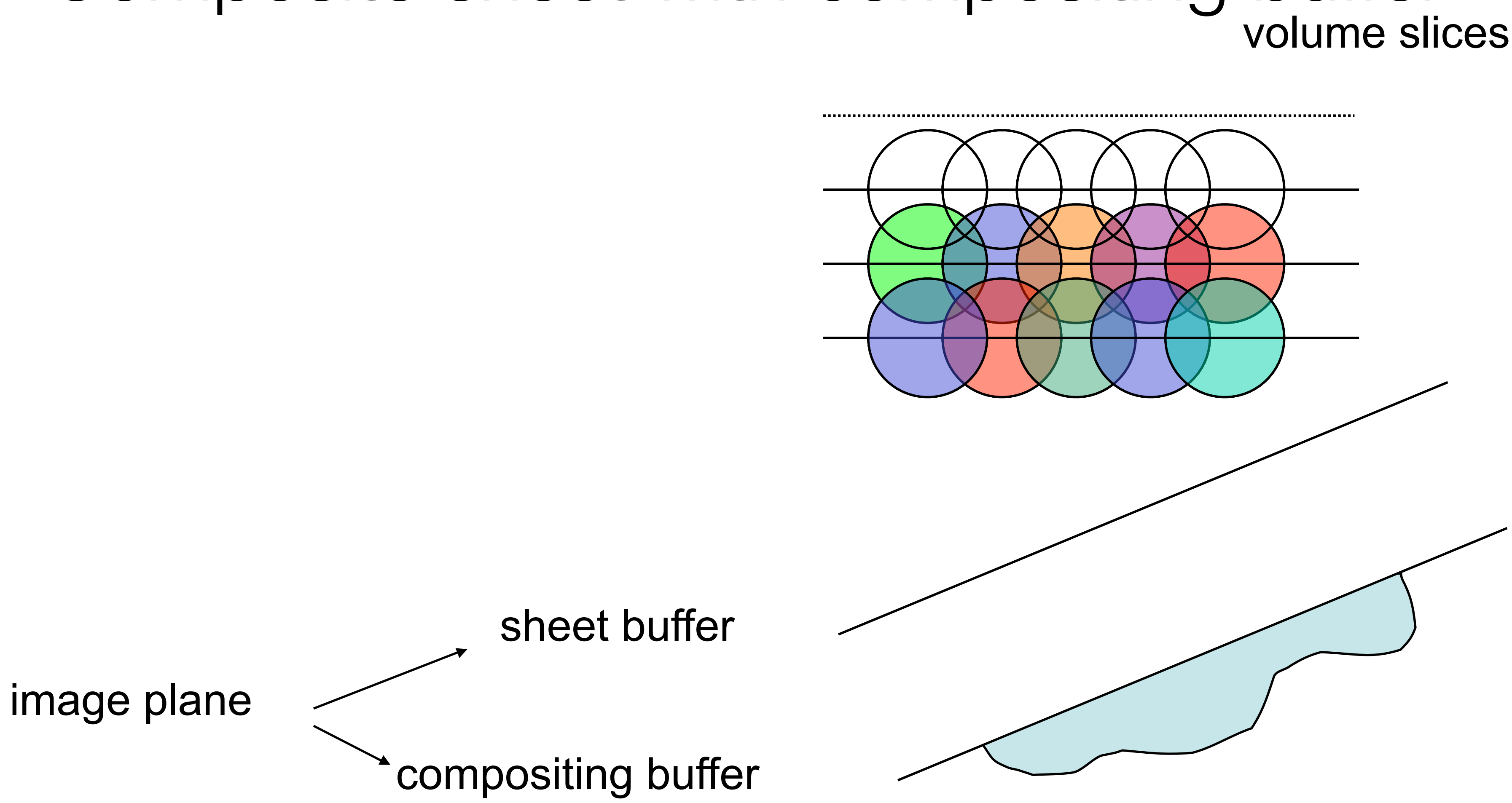
Splatting

- Add voxel kernels within second sheet



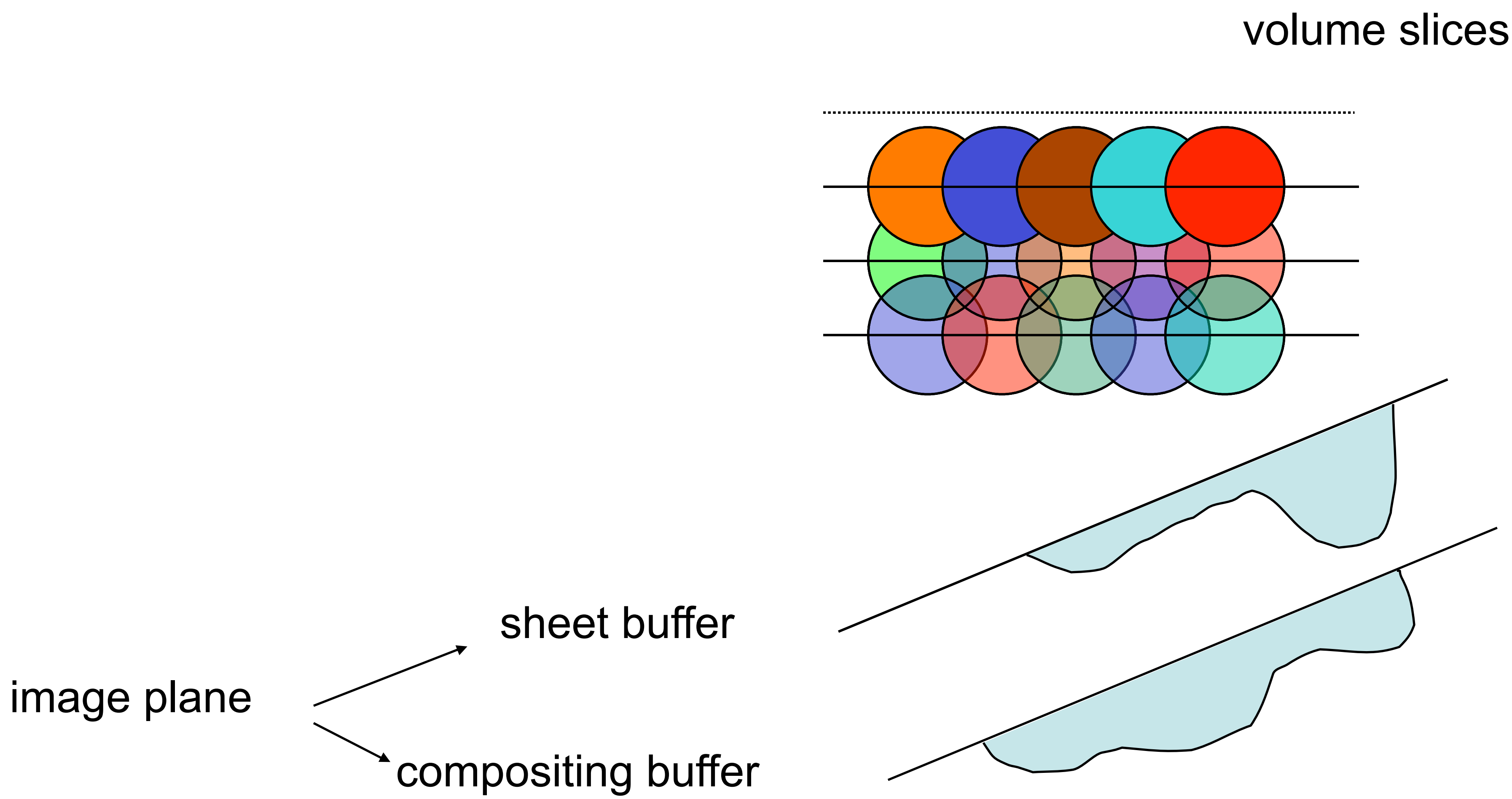
Splatting

- Composite sheet with compositing buffer



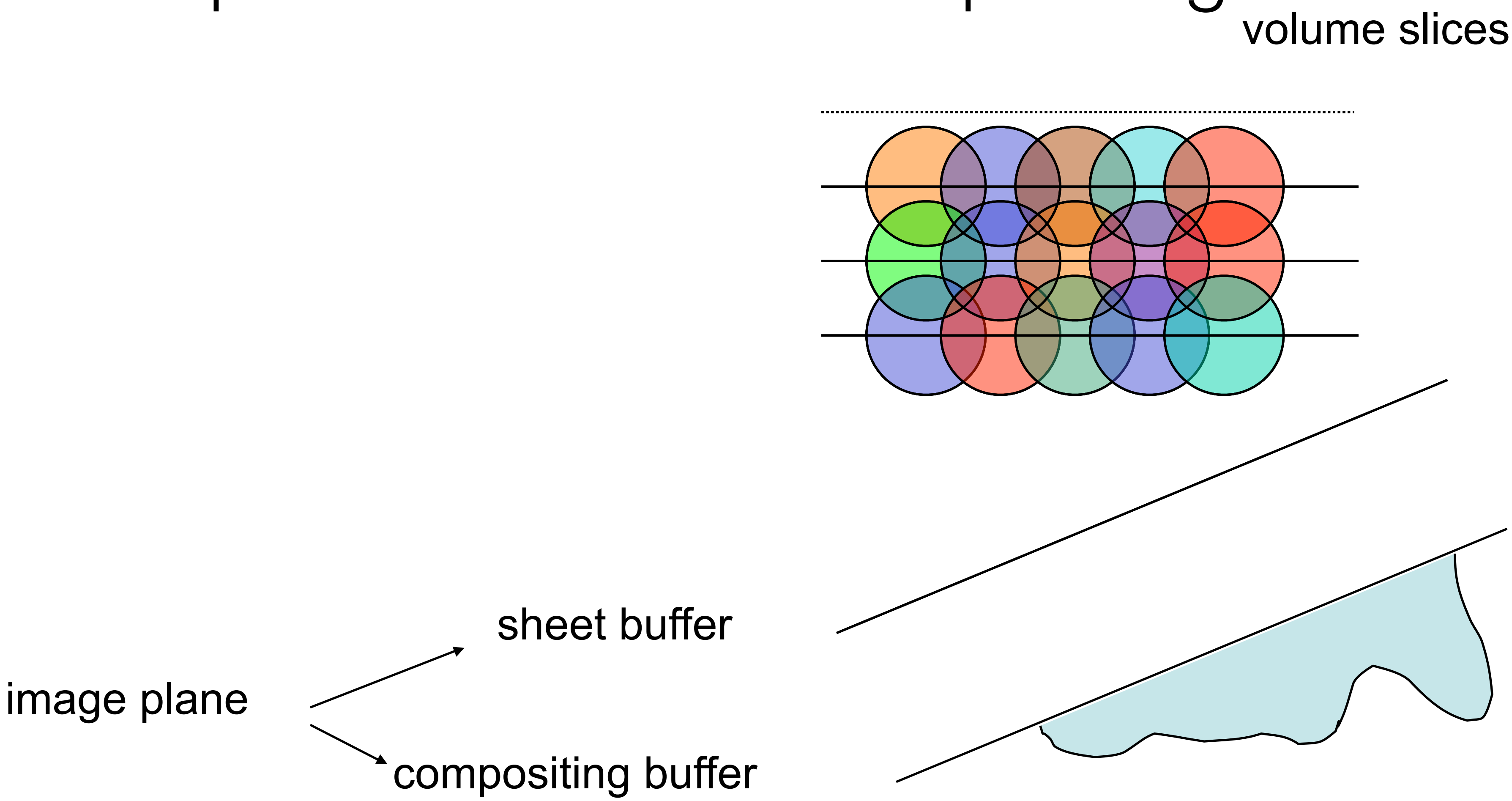
Splatting

- Add voxel kernels within third sheet



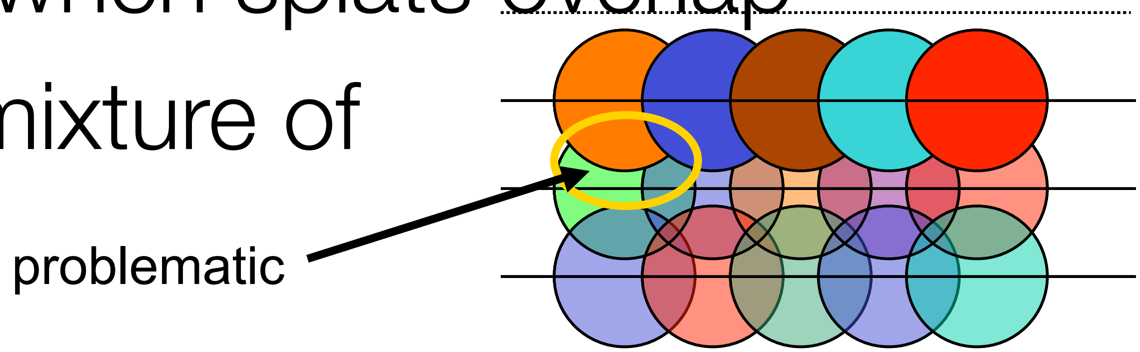
Splatting

- Composite sheet with compositing buffer

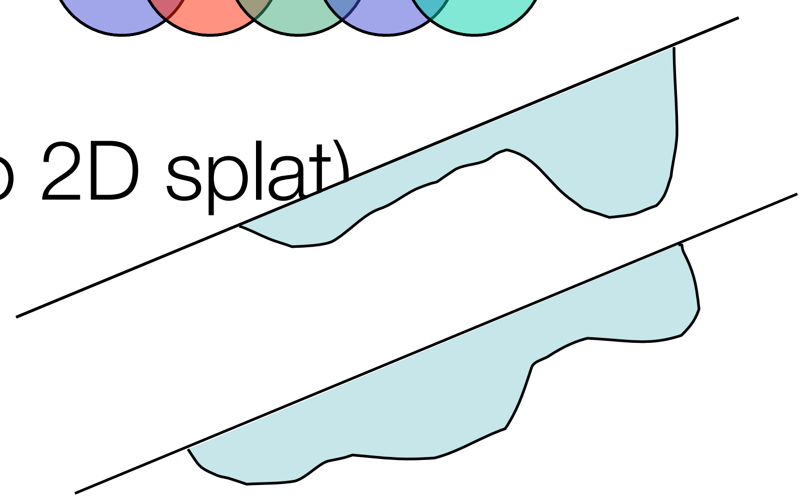


Splatting

- Inaccurate compositing
- Problems when splats overlap
- Incorrect mixture of

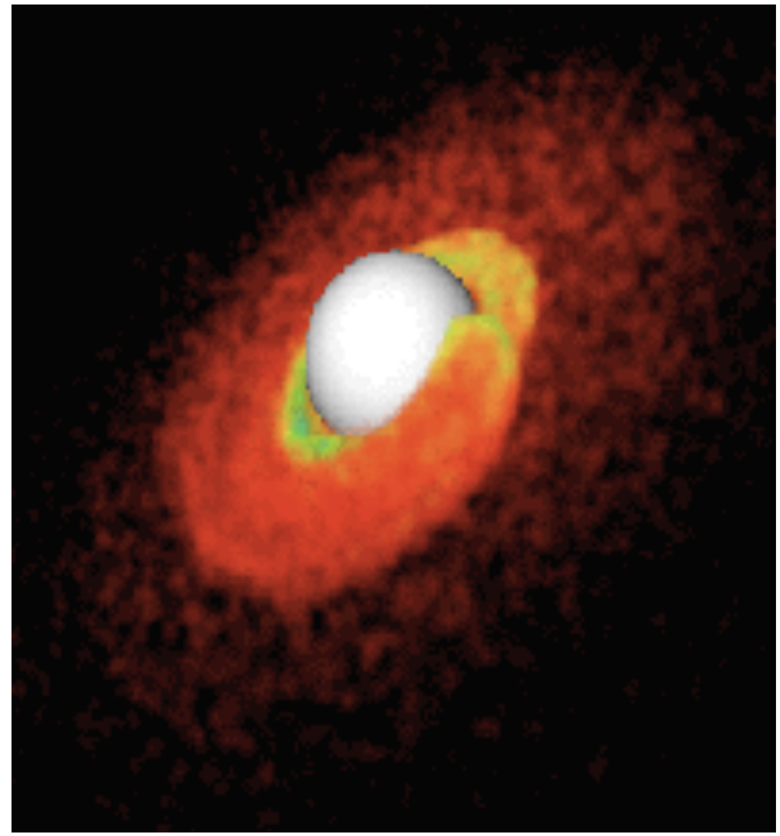


  - Integration (3D kernel to 2D splat) and
  - Compositing



Splatting

- Simple extension to volume data without grids
  - Scattered data with kernels
  - Example: SPH (smooth particle hydrodynamics)
  - Needs sorting of sample points



Overview

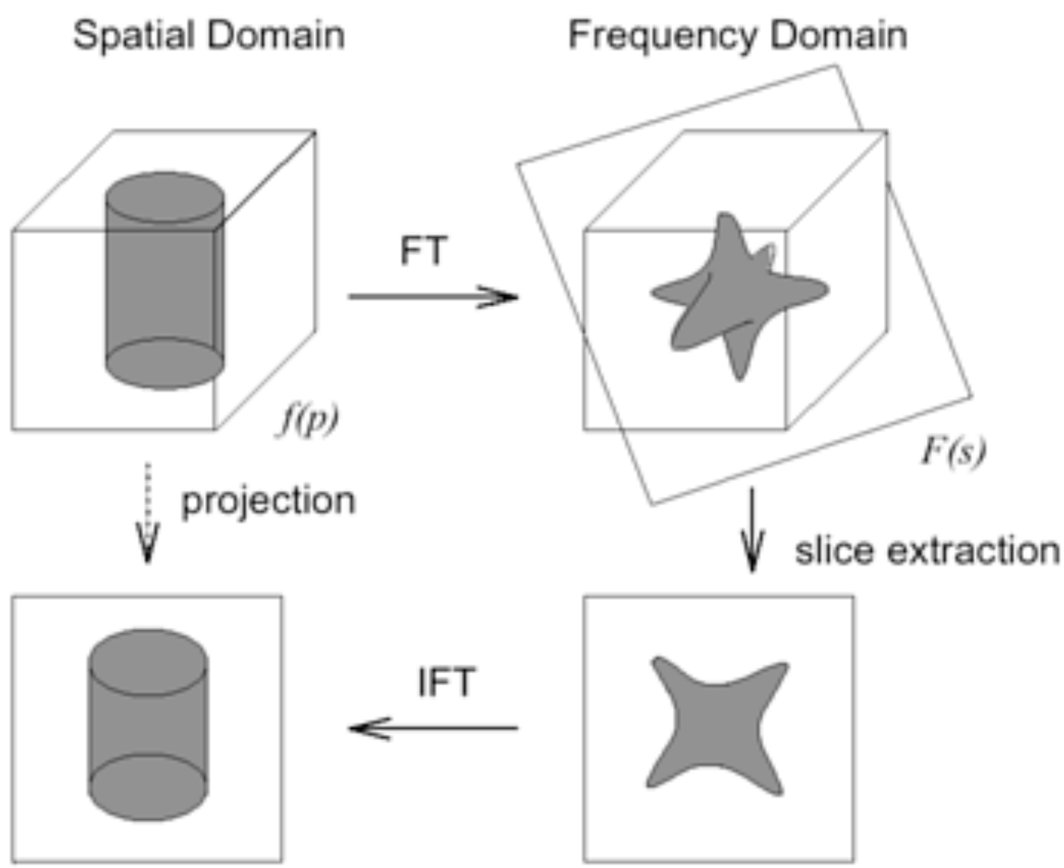
- Light: Volume rendering equation
- Discretized: Compositing schemes
- Ray casting
  - Acceleration techniques for ray casting
- Texture-based volume rendering
- Shear-warp factorization
- Splatting
- Fourier Volume Rendering
- Cell projection (Shirley-Tuchman)

Fourier Volume Rendering

- Tom Malzbender 1993
- Totsuka, Levoy 1993
- non-”traditional” method
- rendering in the Fourier domain
- based on Fourier Projection Slice Theorem
- very efficient
- lots of accuracy problems

Projection Slice Theorem

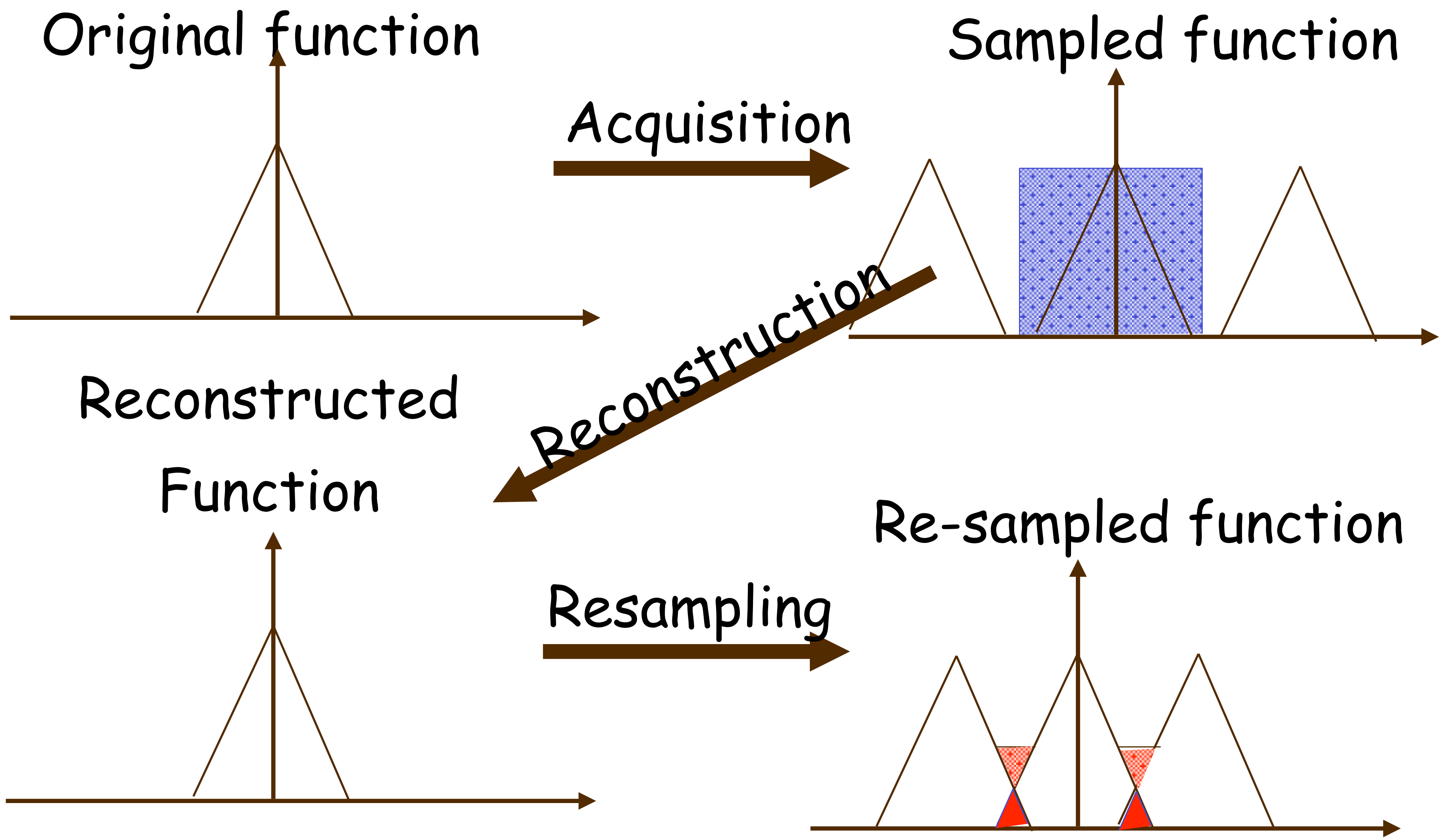
- Relates a slice of the Fourier transform to an integral in one direction in spatial domain



FVR - Basic Algorithm

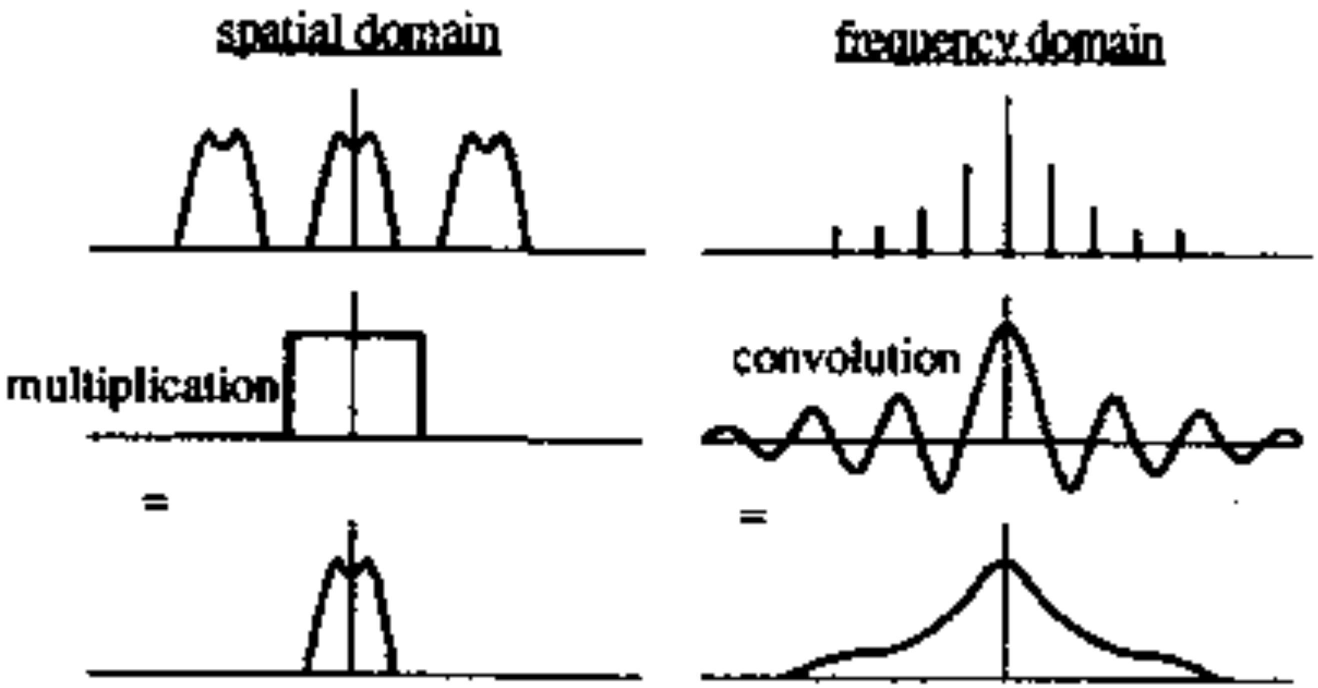
- Preprocessing:
  - pre-multiply spatial domain
  - zero-pad the volume
  - compute Fourier transform
- Actual Algorithm
  - compute viewing angle
  - extract 2D slice
  - inverse 2D Fourier transform of slice

FVR - Resampling revisited



FVR - Pre-multiplication

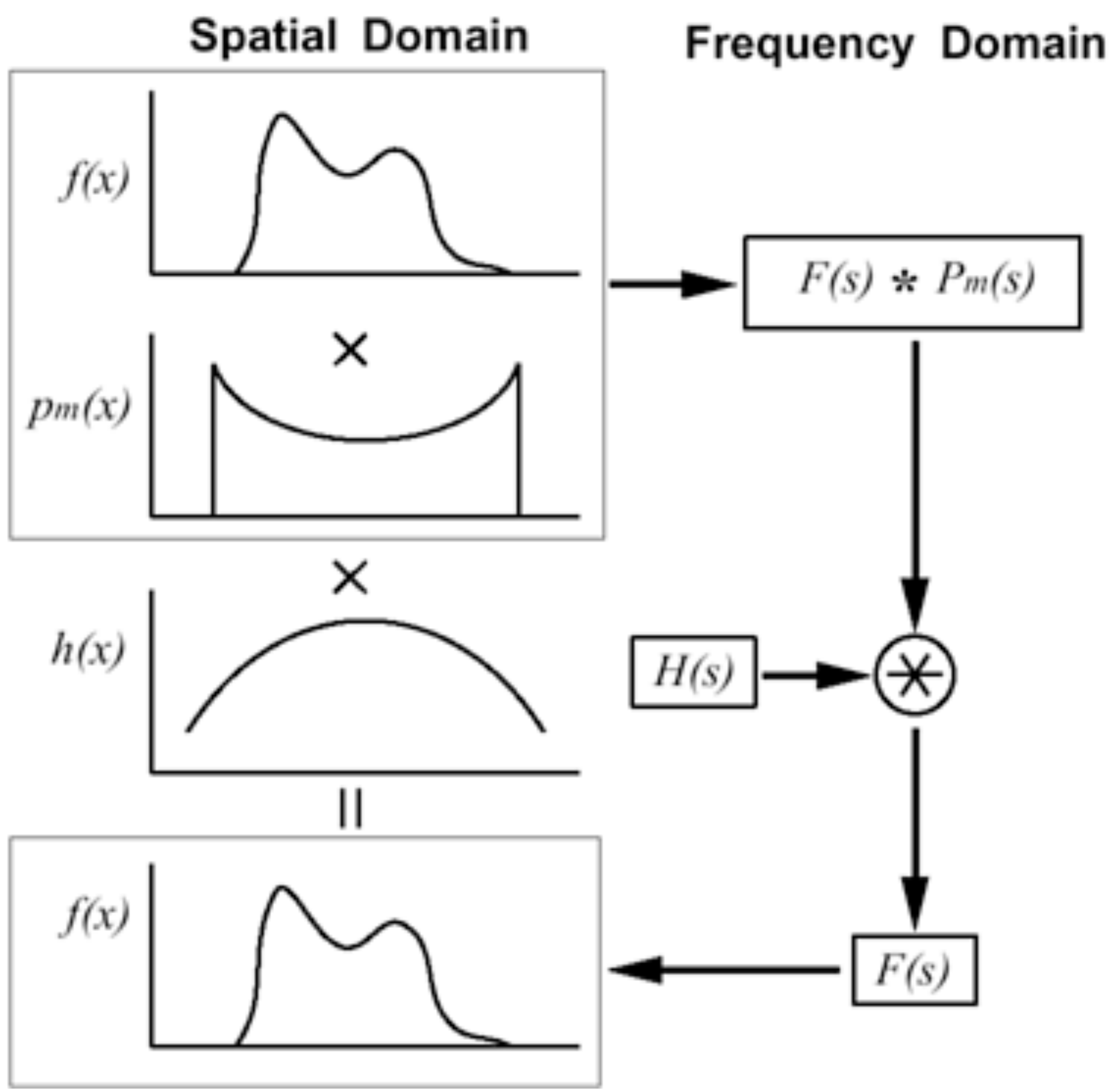
- Extracting slice requires a resampling step
- what impact has sampling in Frequency domain to the spatial domain??



FVR - Pre-multiplication (2)

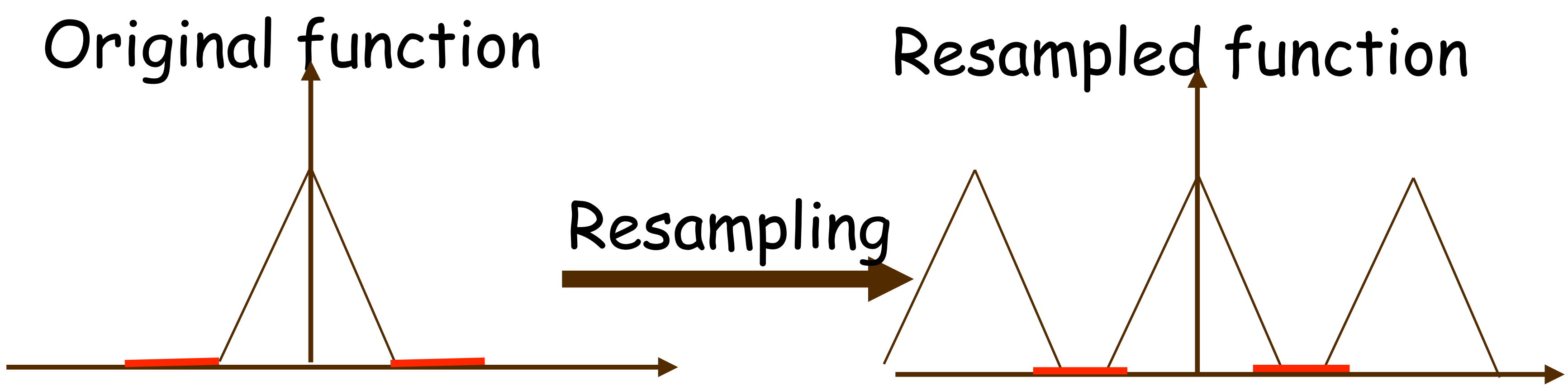
- Or mathematically:
- Reconstruction = convolution with an interpolation filter H:
- $F_h(w) = F(k)*H(s)$
- and in spatial domain:
- $f_h(x) = f(x).h(x)$

FVR - Pre-multiplication (3)



FVR - zero-padding

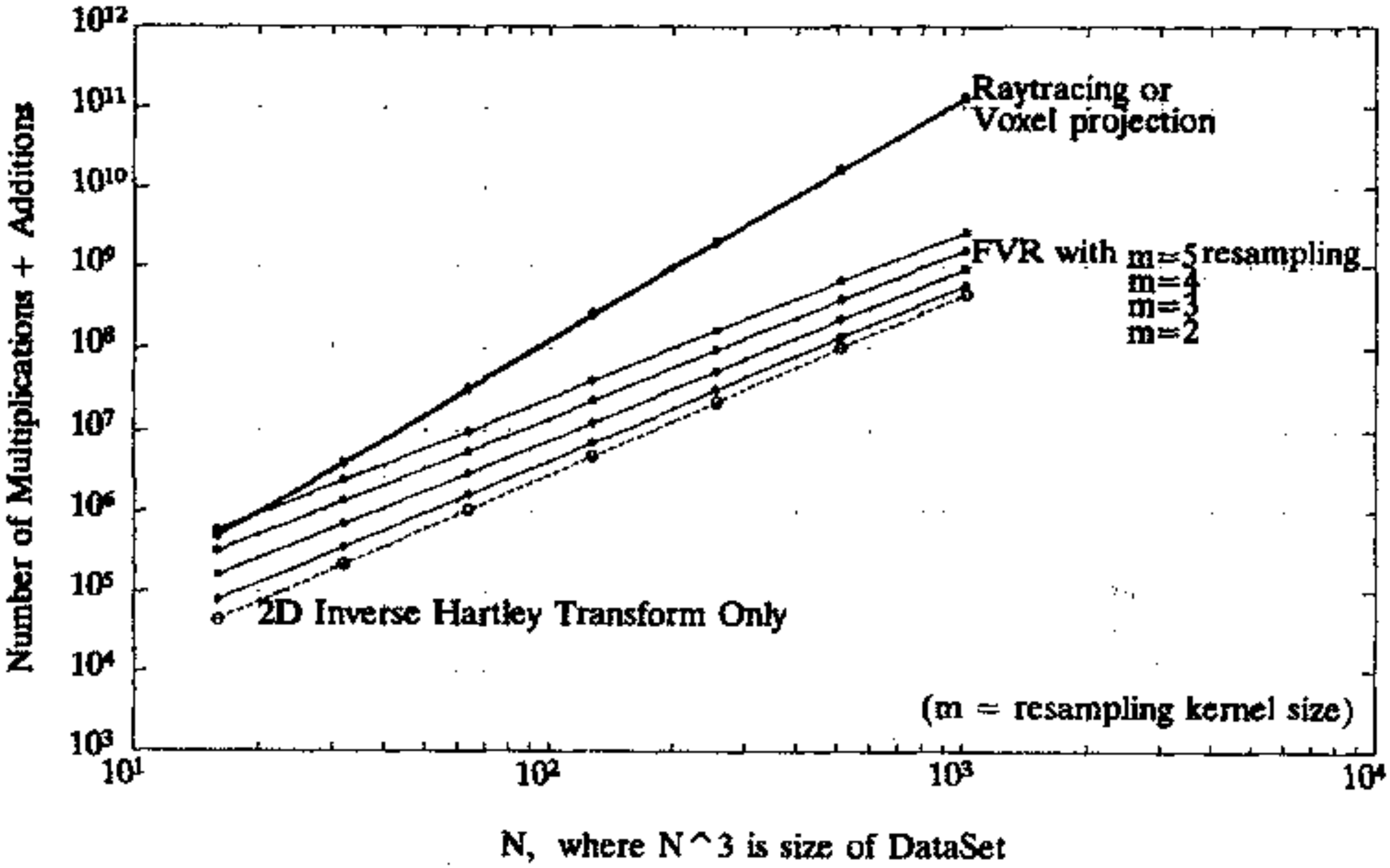
- Separates the spatial replicas further
- Decreases artifacts in spatial domain
- zero-padded function:



FVR - Efficiency

- Typical Fourier Transform = $O(N^3 * N^3)$
- Fast Fourier Tranform = $O(N^3 * \log N)$
- Hence:
  - pre-processing = $O(N^3 * \log N)$
  - slicing = $O(N^2)$
  - inverse Fourier Transform (slice) = $O(N^2 * \log N)$
- other rendering algorithm $O(N^3)$

FVR - Efficiency (2)



FVR - Depth-Shading

- Basic algorithm produces x-ray type images
- no depth information is conveyed
- depth incoding: f(x).d(x)
- Fourier Transform (with interpolation):
- F(w)*D(w)*H(w)
- Hence pre-multiply H with D!

FVR - Depth-Shading (2)

- Linear depth cueing:

$$d_l(x) = C_{cue}(V \times x) + C_{avg}$$

- Fourier Transform

$$D_l(\omega) = -\frac{C_{cue}}{i2\pi}(V \times \Delta) + C_{avg}\delta(\omega)$$

- Combined filter:

$$H'(\omega) = D_l(\omega) * H(\omega)$$

$$= -\frac{C_{cue}}{i2\pi}(V \times \nabla H(\omega)) + C_{avg} H(\omega)$$

FVR - Ambient-Shading

- Typical ambient component: $C_{amb} L_{amb} O_c$
  - C - color
  - L - constant
  - O - object color
- approximation: $C_{amb} L_{amb} f(x)$
- Fourier transform:

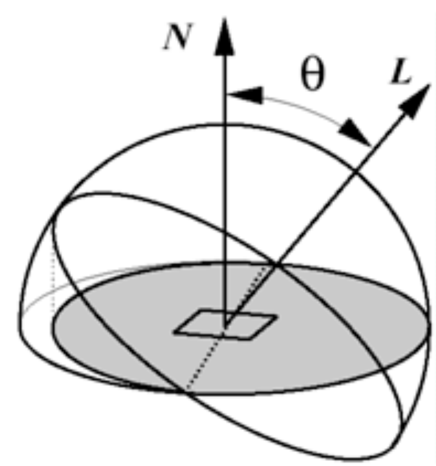
$$C_{amb} L_{amb} F\{f(x) \times p_m(x)\} * H(\omega)$$

FVR - Diffuse-Shading

- Typical diffuse component:

$$C_{dif} L_{dif} O_c \max\left(0, N \times L\right)$$

  – N - normal vector
  – L - light vector
- doesn't have a simple Fourier transform
- approximation - illumination by hemisphere:



$$C_{dif} L_{dif} \frac{1}{2} \left|\nabla f\left(x\right)\right| \left(1 + \frac{\nabla f\left(x\right) \times L}{\left|\nabla f\left(x\right)\right|}\right)$$

FVR - Diffuse-Shading (2)

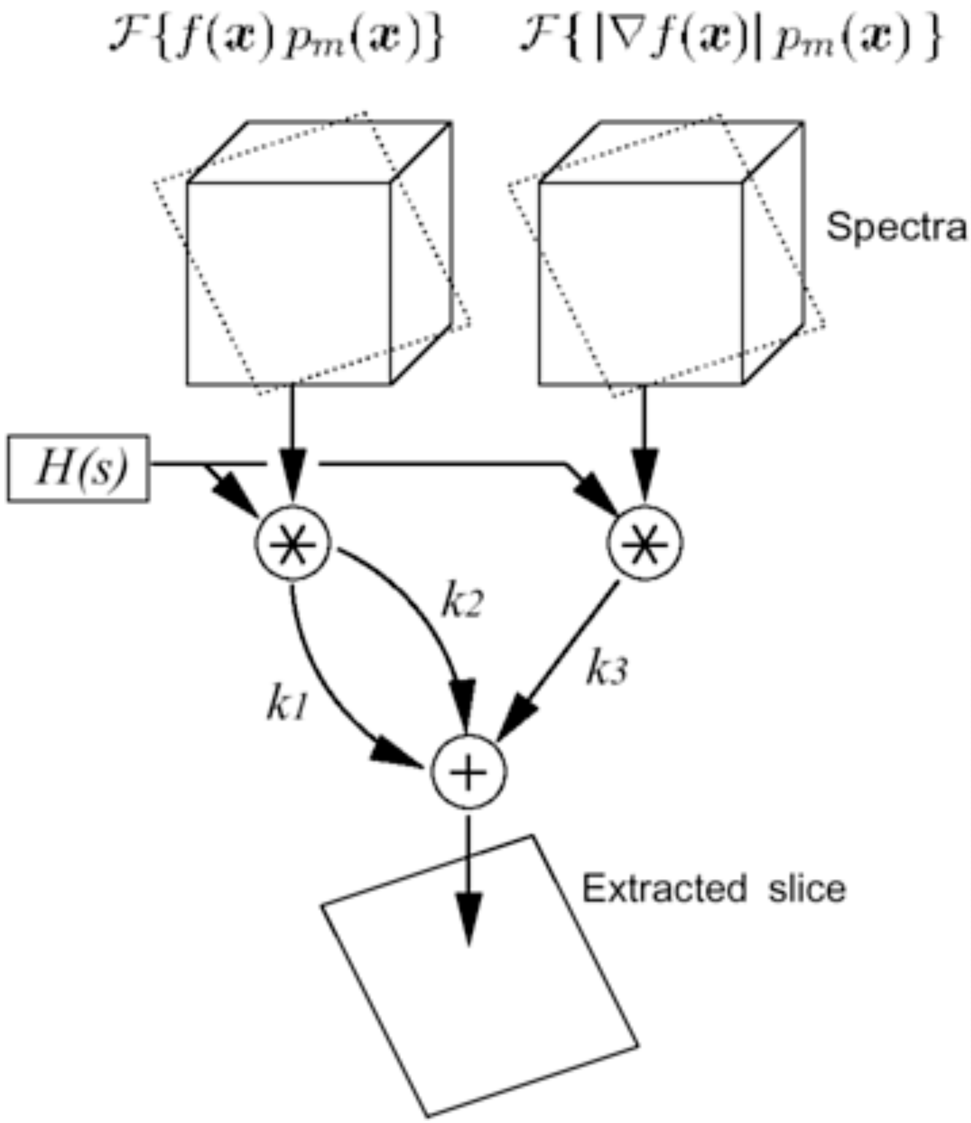
- approximation:

$$C_{dif} L_{dif} \, \frac{1}{2} \left| \nabla f(x) \right| \left(1 + \frac{\nabla f(x) \times L}{\left| \nabla f(x) \right|} \right)$$

- Fourier transform:

$$C_{dif} L_{dif} \left(\begin{array}{l} \frac{1}{2} F\left\{ \left| \nabla f(x) \right| \times p_m(x) \right\} * H(\omega) + \\ + i\pi(\omega \times L) F\left\{ f(x) \times p_m(x) \right\} * H(\omega) \end{array} \right)$$

FVR - Diffuse-Shading (3)

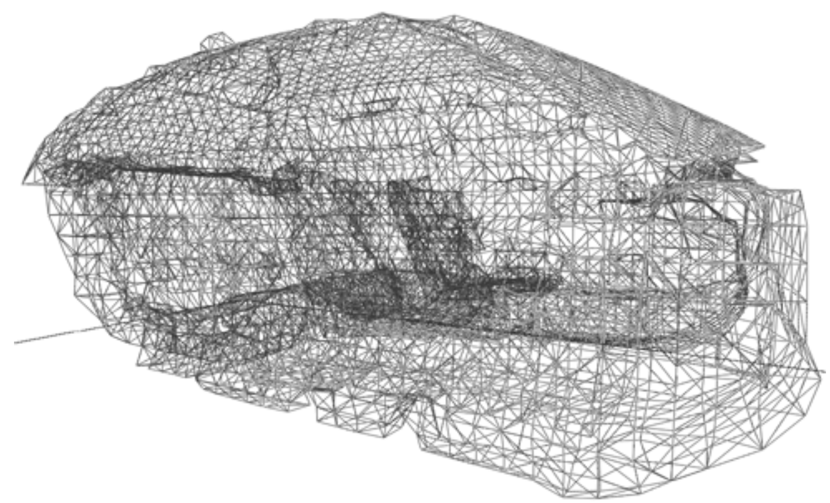


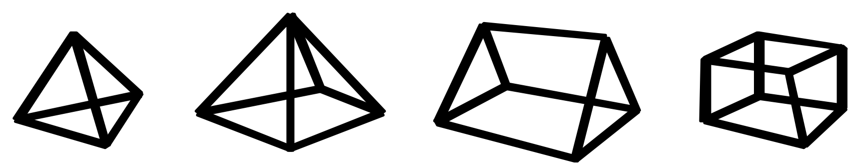
Overview

- Light: Volume rendering equation
- Discretized: Compositing schemes
- Ray casting
  - Acceleration techniques for ray casting
- Texture-based volume rendering
- Shear-warp factorization
- Splatting
- Fourier Volume Rendering
- Cell projection (Shirley-Tuchman)

Cell Projection

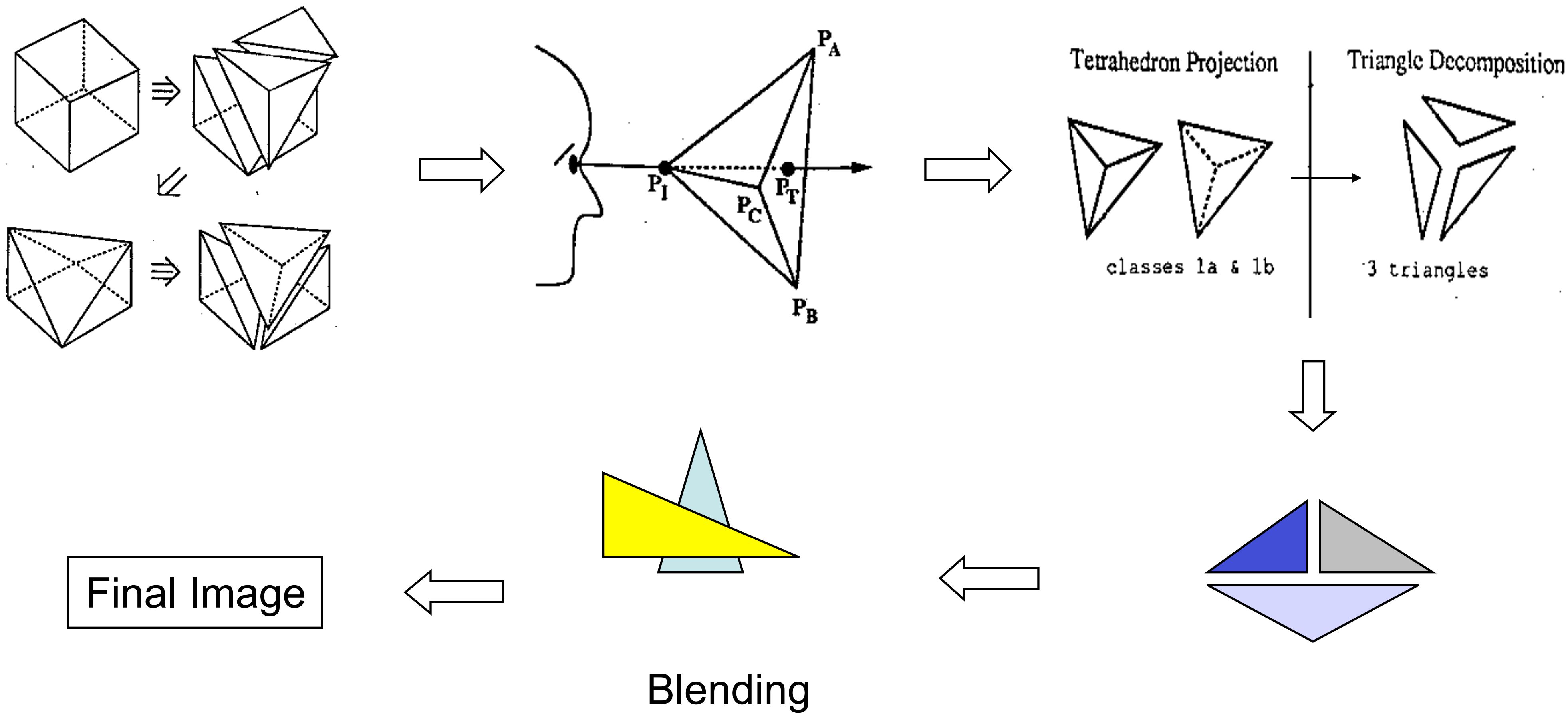
- For unstructured grids
- Alternative to ray casting (Garrity's alg.)
- Projected Tetrahedra (PT) algorithm
  [P. Shirley, A. Tuchman: A polygonal approximation to direct scalar volume rendering, Volvis 1990, p. 63-70]





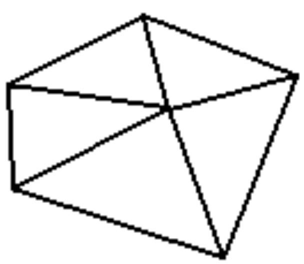
Cell Projection

- Basic idea

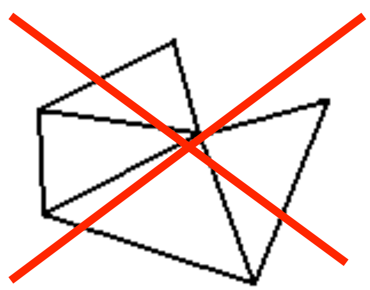


Cell Projection

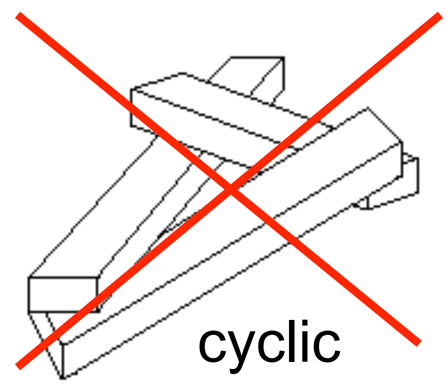
- Spatial sorting for all tetrahedra in a grid
  - Back-to-front or front-to-back strategies possible
  - Compositing is not commutative
- MPVO algorithm: Meshed Polyhedra Visibility Ordering [P. Williams: Visibility Ordering Meshed Polyhedra, ACM Transactions on Graphics, 11(2), 1992, p. 103-126]
  - Only for acyclic, convex grids



convex



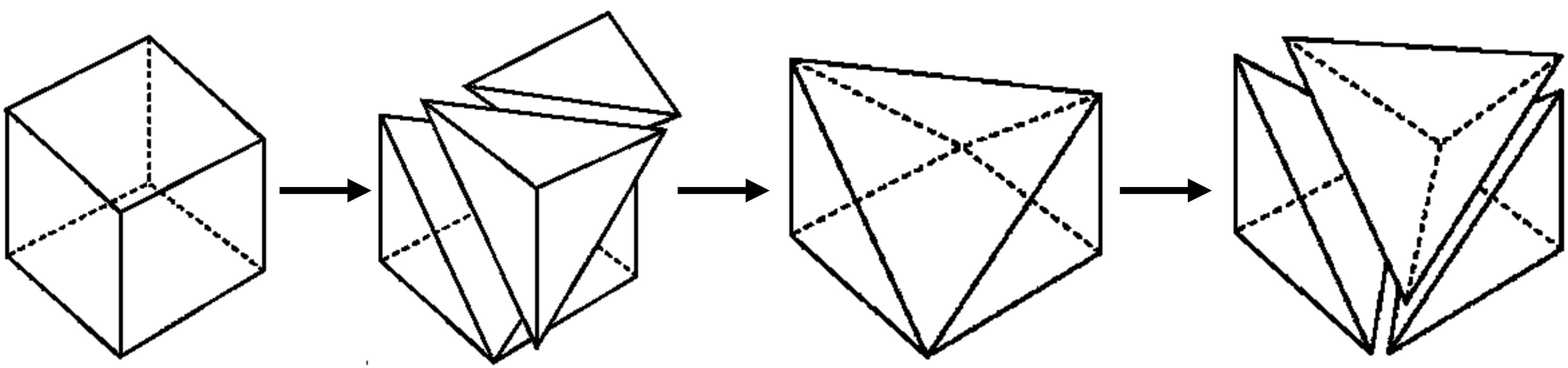
non-convex



cyclic

Cell Projection

- Decomposition of non-tetrahedral unstructured grids into tetrahedra
  - PT can be applied for all types of unstructured grids



Cell Projection

- Alternative to working directly on unstructured grids
  - Resampling approaches, adaptive mesh refinement (AMR)