Isabel Navazo

Pere-Pau Vázquez

Scientific Visualization 2020 – 2021

FUNDAMENTAL BACKGROUND

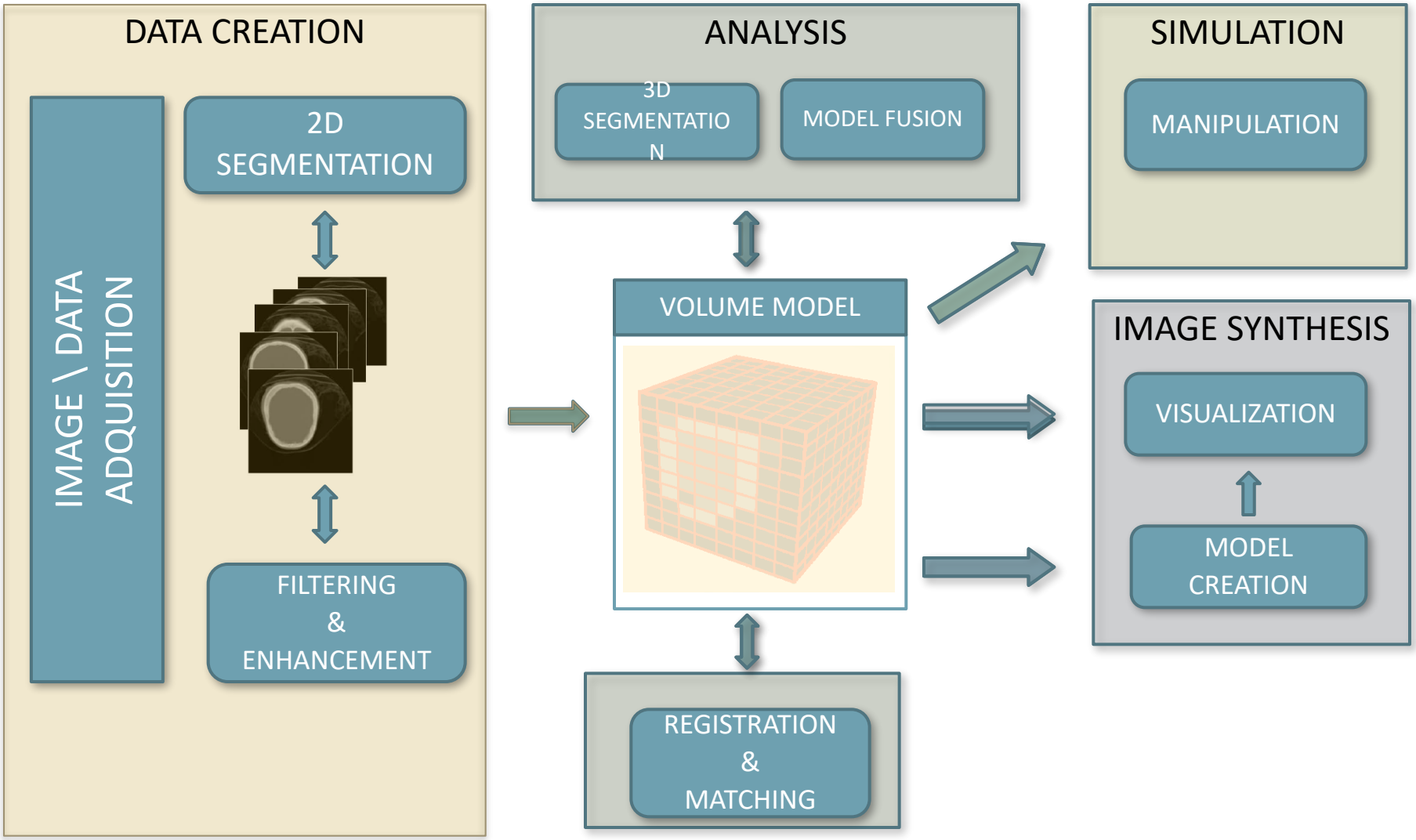
About the slides

- Contents coming from different sources, including:
  - Christof Rezk-Salama, Markus Hadwiger, Patric Ljung, Timo Ropinski (EG Volume Rendering Course, 2006), Travis Gorking, Eva Monclús …

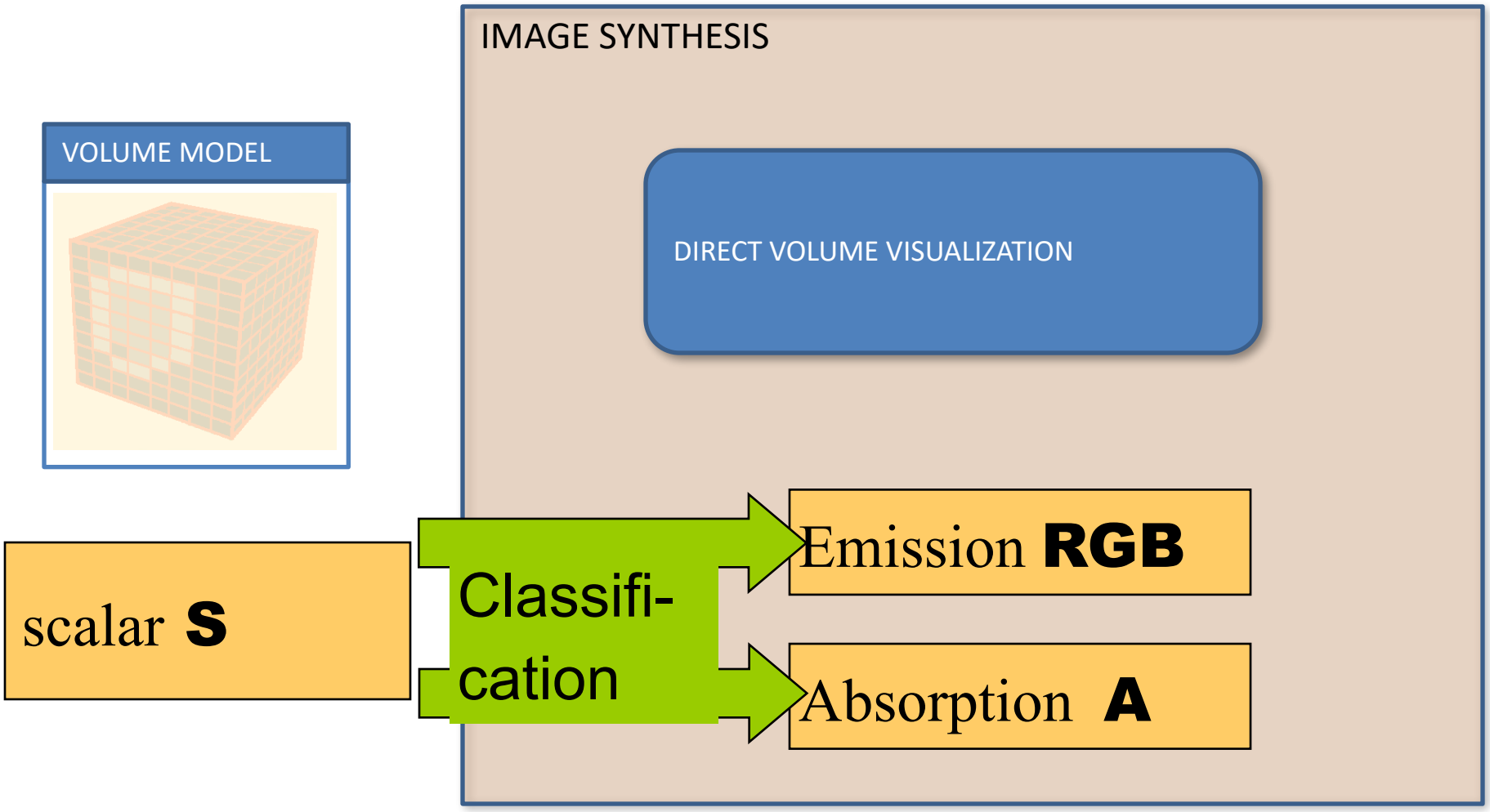
Outline

- Volume rendering equation
- Basic Approaches: Direct volume rendering, 2D and 3D textures

Volume Graphics Pipeline

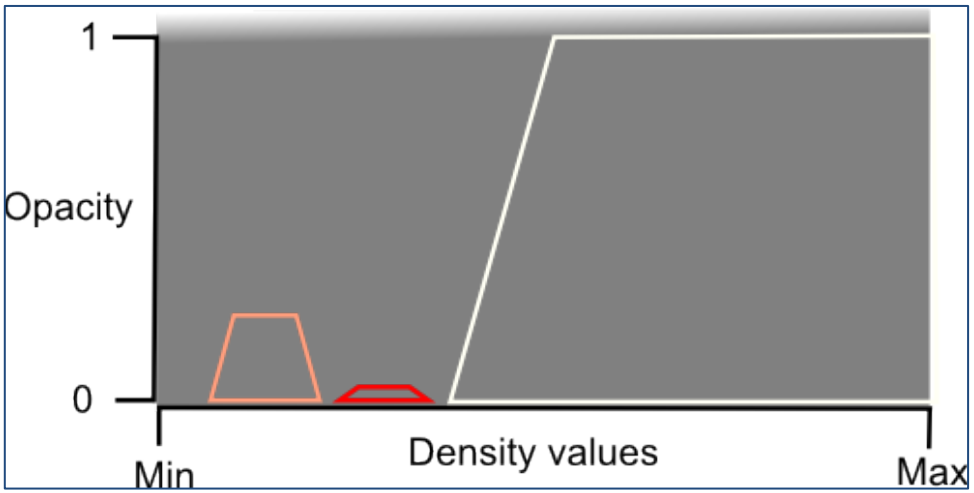


Direct Volume Rendering



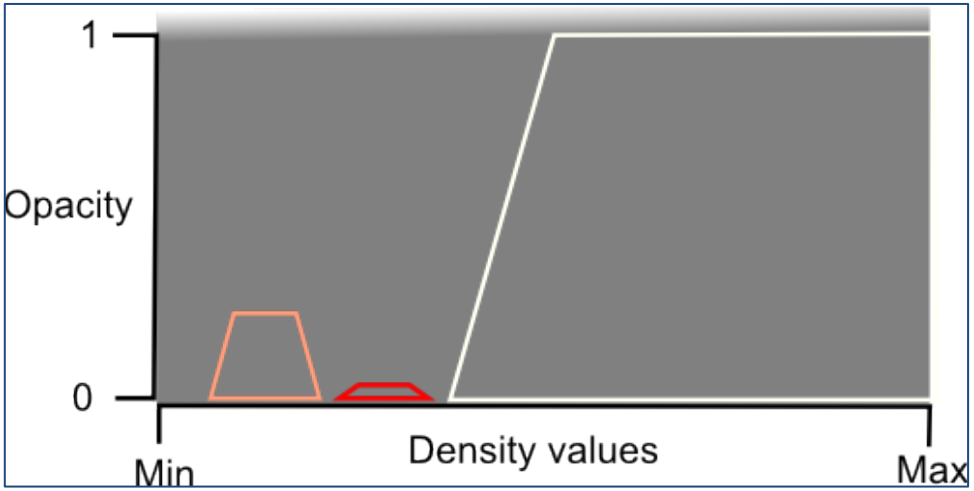
Classification

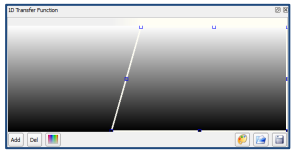
- Specifies the visual appearance of volume data: the Transfer Function

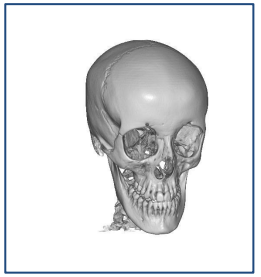


Classification

- Specifies the visual appearance of volume data: the Transfer Function

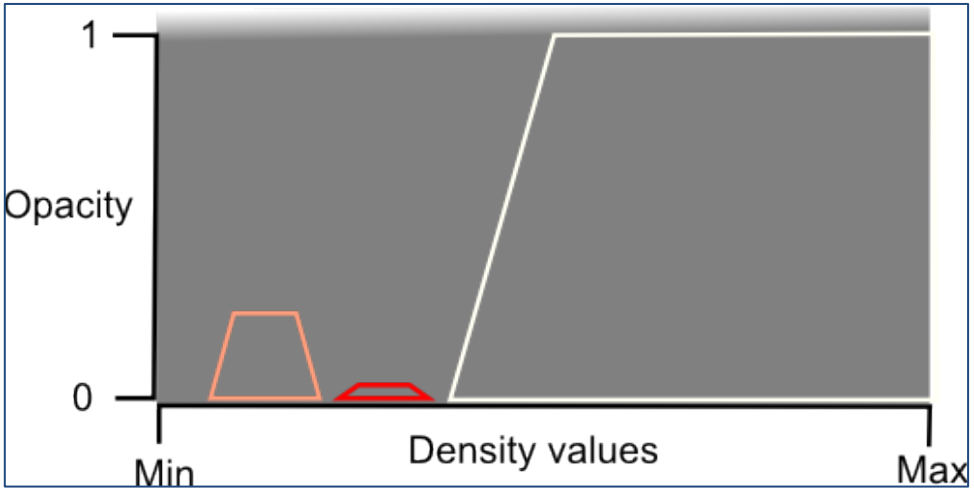


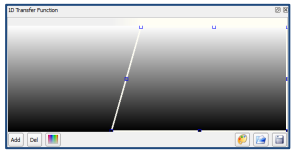


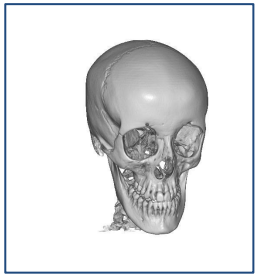


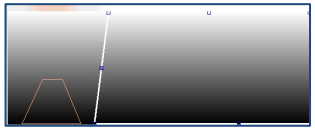
Classification

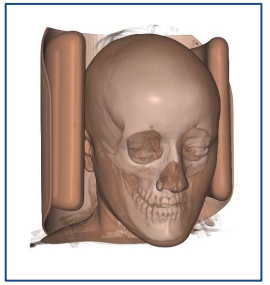
- Specifies the visual appearance of volume data: the Transfer Function





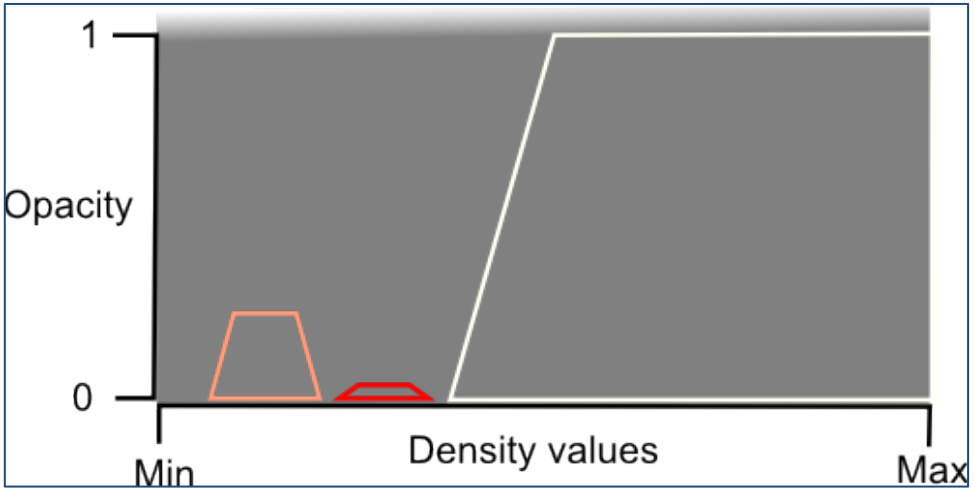


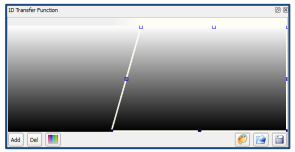


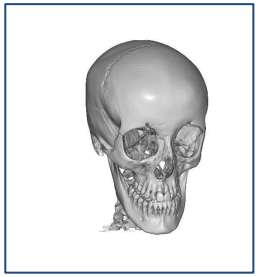


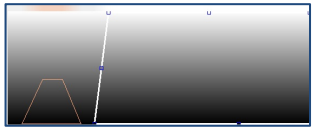
Classification

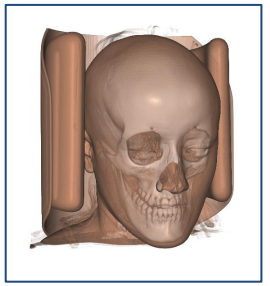
- Specifies the visual appearance of volume data: the Transfer Function

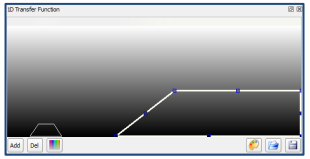


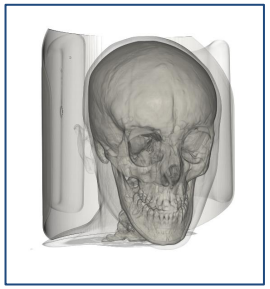








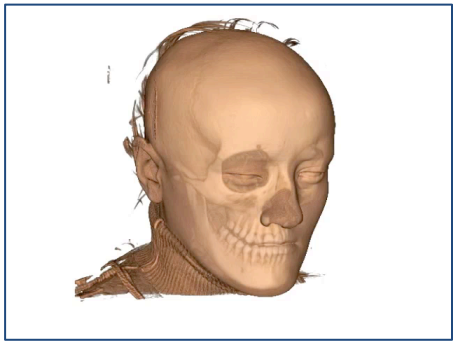




Classification

- Some considerations: pros & cons

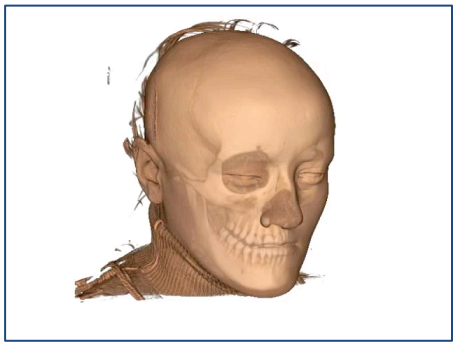
- FT gives good results in many applications
- Focuses on “materials” not on anatomy structures
    - Some structures has the same “material” (for acquisition devises features)
    - It is not possible “identify” all structures
    - Fuzzy boundaries (densities)
- *Not semantic information only visual perception*
- Sometimes is difficult to design the needed TF

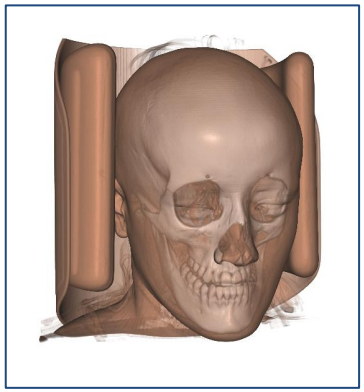


Classification

- Some considerations: pros & cons

- FT gives good results in many applications
- Focuses on “materials” not on anatomy structures
    - Some structures has the same “material” (for acquisition devises features)
    - It is not possible “identify” all structures
    - Fuzzy boundaries (densities)
- *Not semantic information only visual perception*
- Sometimes is difficult to design the needed TF

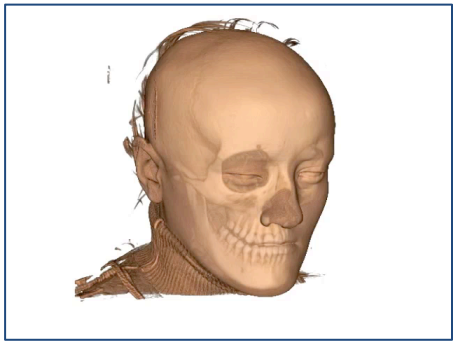


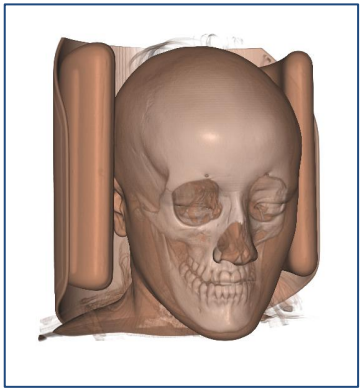


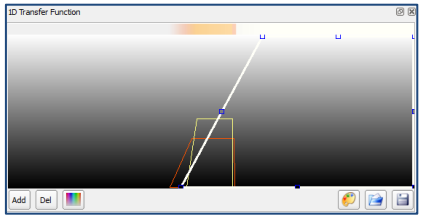
Classification

- Some considerations: pros & cons

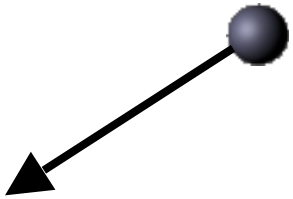
- FT gives good results in many applications
- Focuses on “materials” not on anatomy structures
  - Some structures has the same “material” (for acquisition devises features)
  - It is not possible “identify” all structures
  - Fuzzy boundaries (densities)
- *Not semantic information only visual perception*
- Sometimes is difficult to design the needed TF



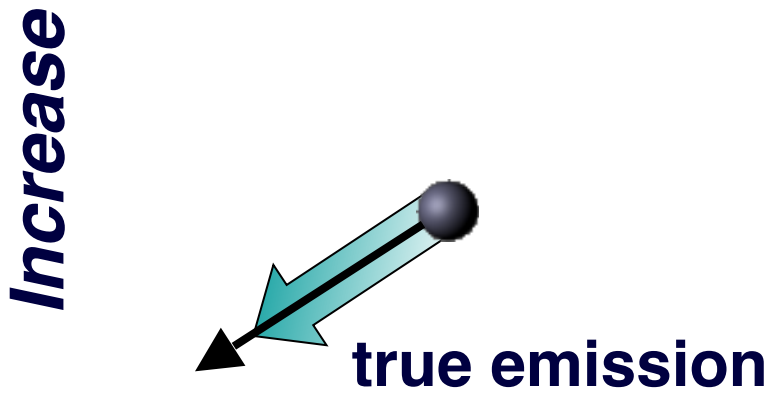




Physical Model of Radiative Transfer

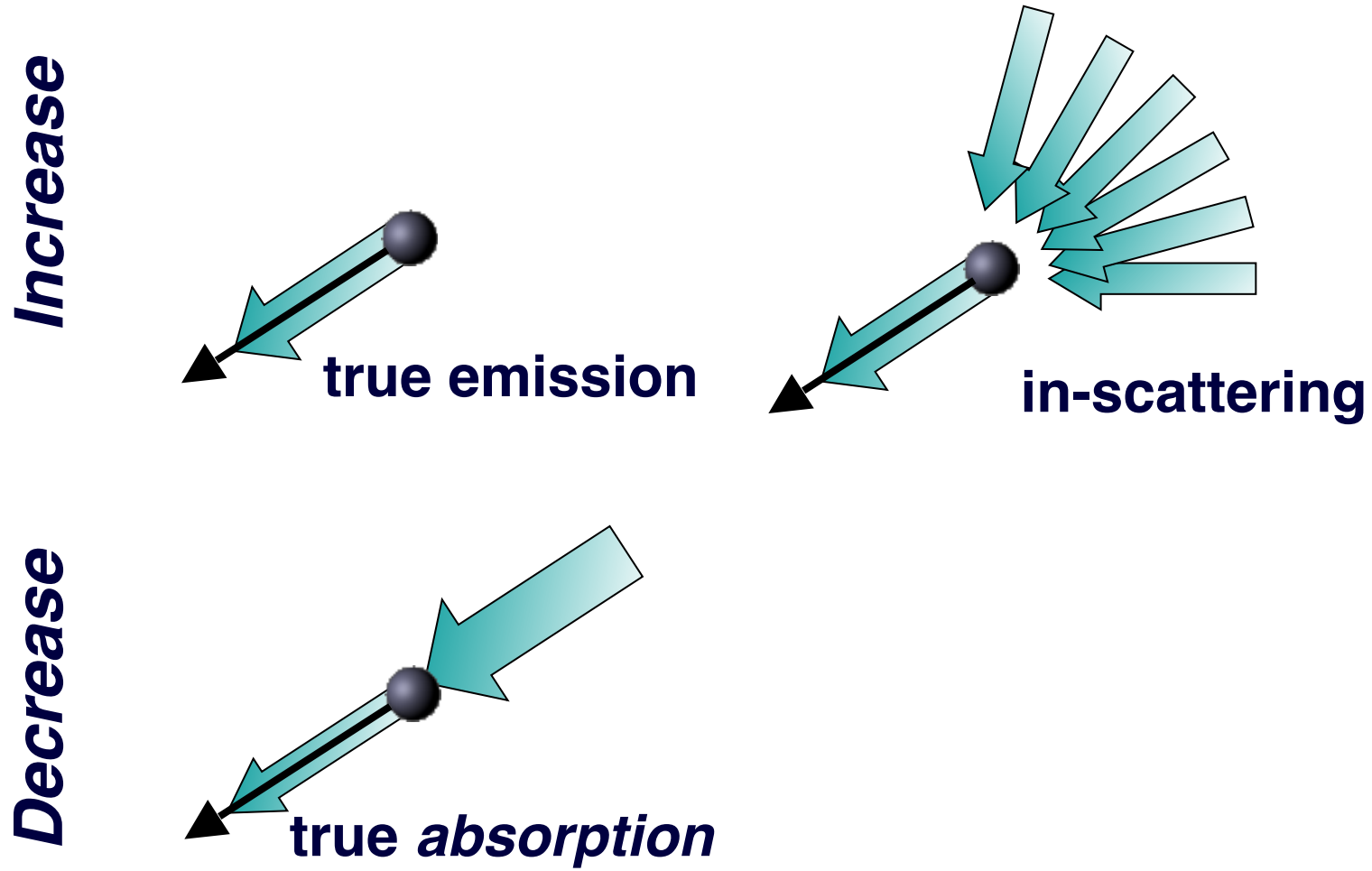


Physical Model of Radiative Transfer

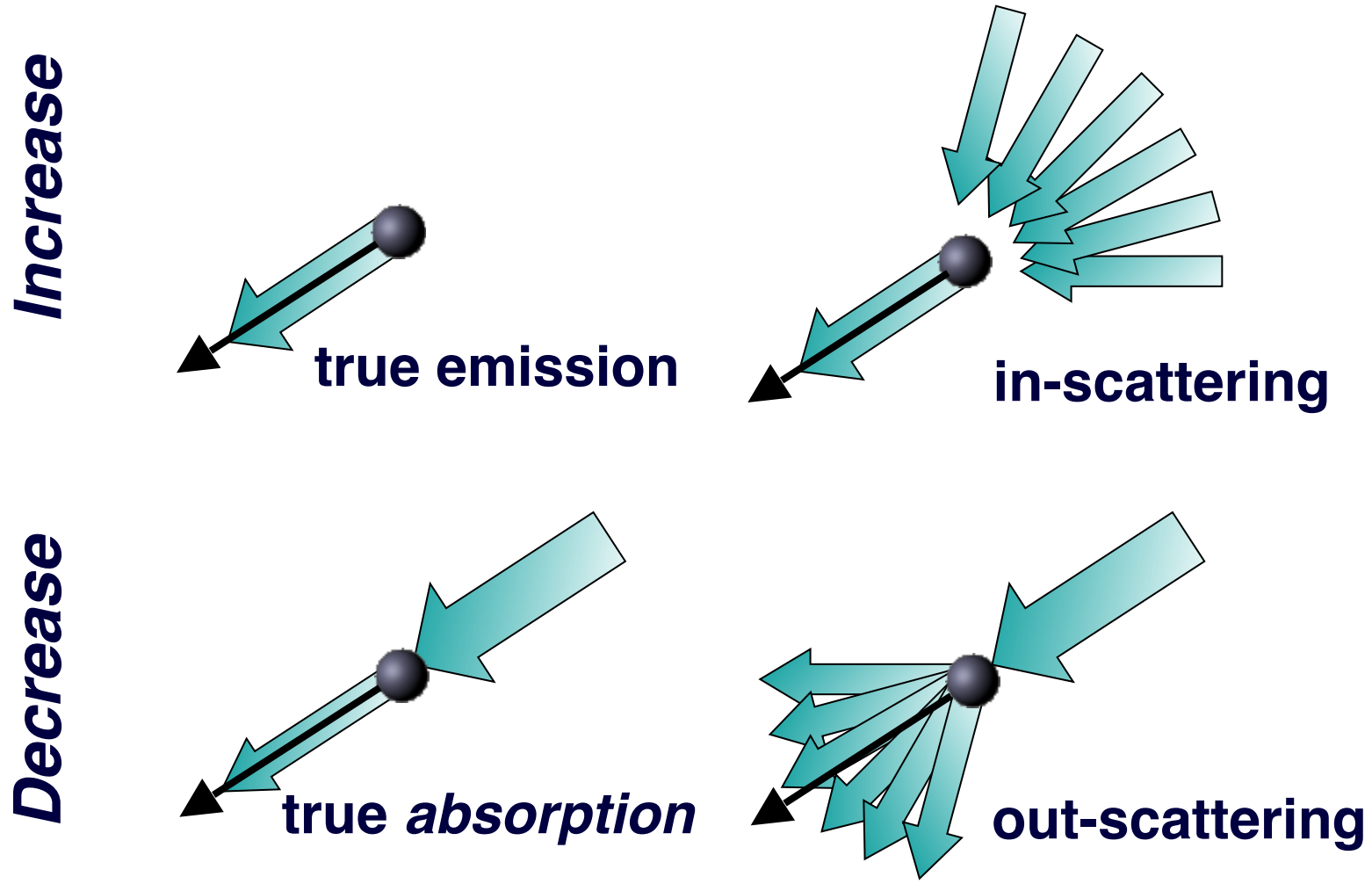


Physical Model of Radiative Transfer

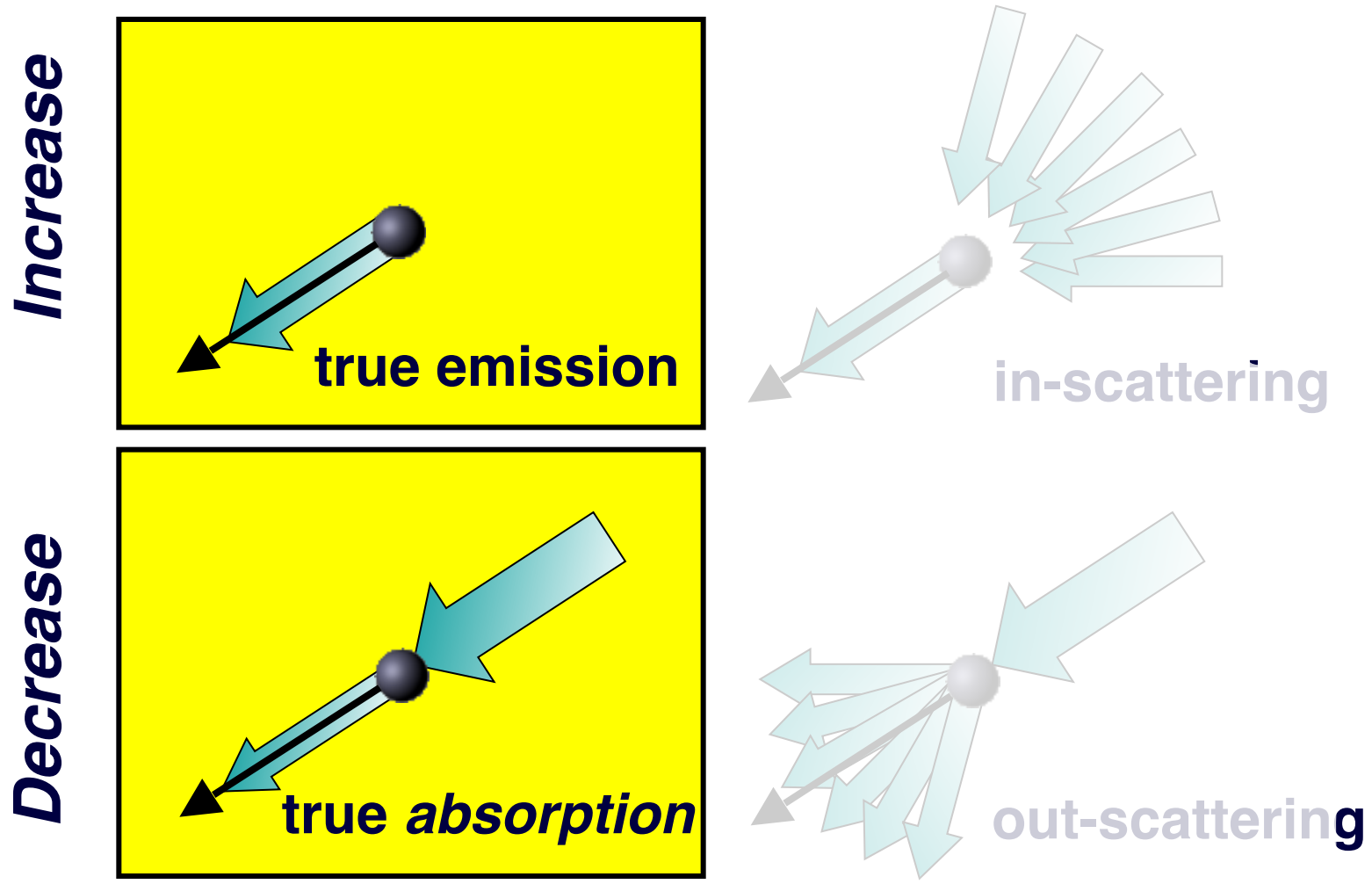
Physical Model of Radiative Transfer



Physical Model of Radiative Transfer



Physical Model of Radiative Transfer



Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering

Ray Integration

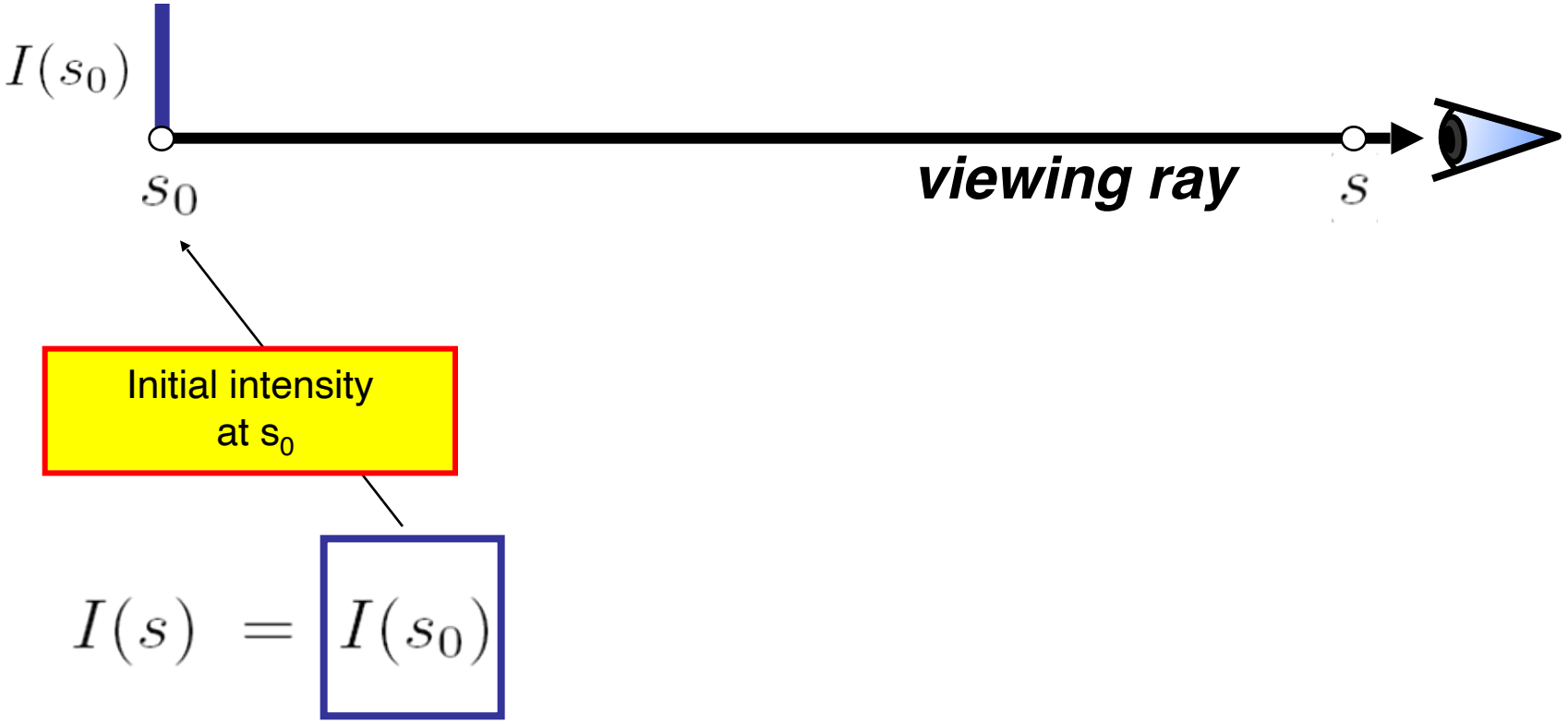
How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering

Ray Integration

How do we determine the radiant energy along the ray?

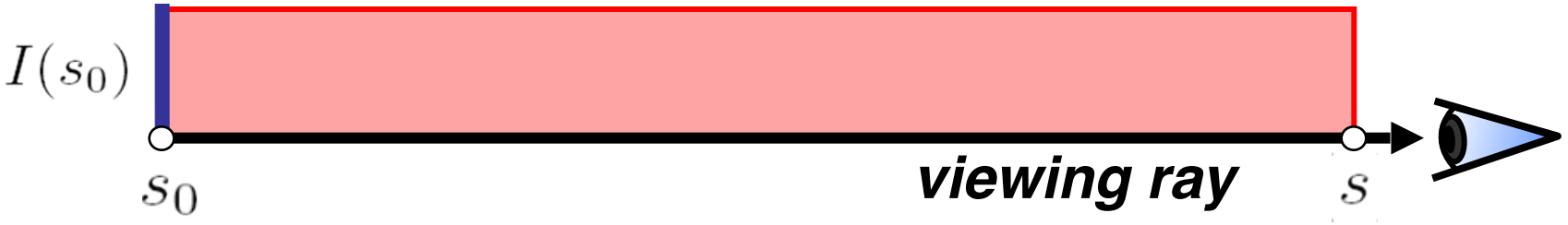
Physical model: emission and absorption, no scattering



Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



Initial intensity at s_0

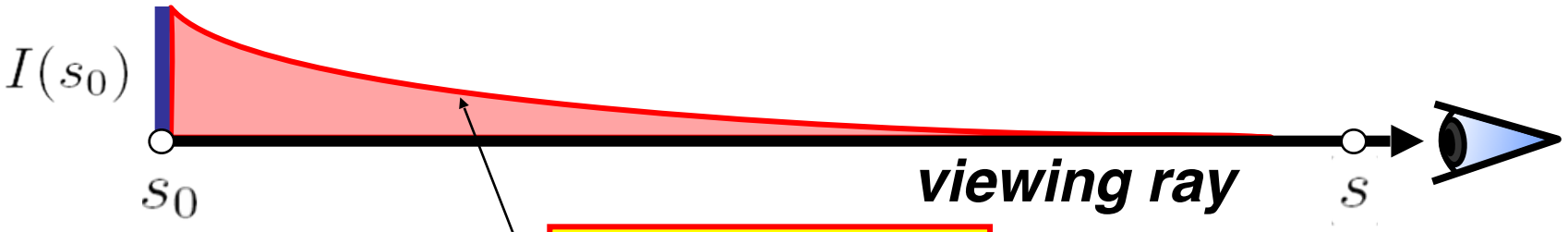
$$I(s) = I(s_0)$$

Without absorption all the initial radiant energy would reach the point ***s.***

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



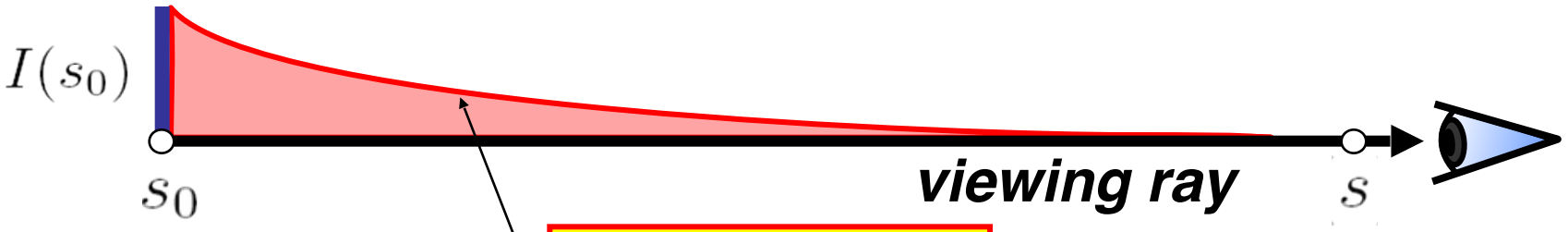
Absorption along the ray segment s_0 - s

$$I(s) = I(s_0)\, e^{-\tau(s_0,s)}$$

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



Absorption along the ray segment s_0 - s

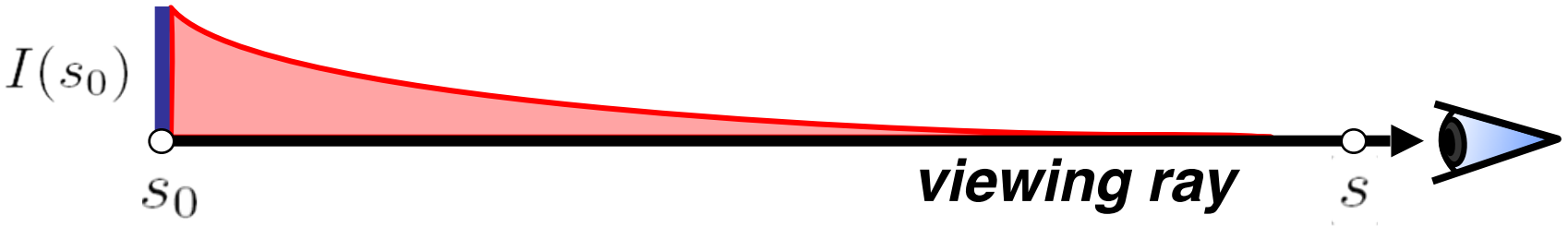
$$I(s) = I(s_0)\, e^{-\tau(s_0,s)}$$

transparency

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



$$I(s) = I(s_0)\, e^{-\tau(s_0,s)}$$

transparency

Extinction τ

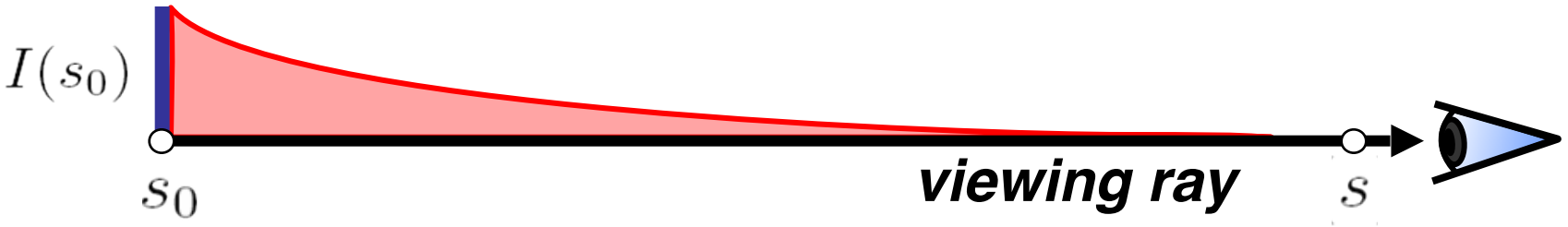
Absorption κ

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

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering

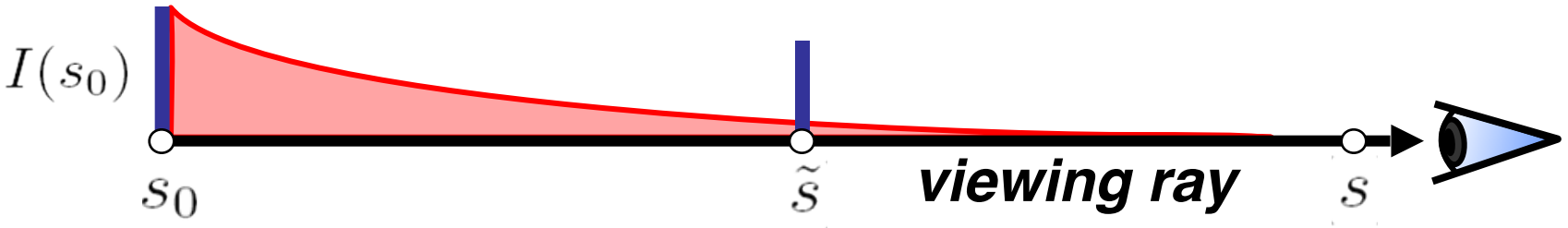


$$I(s) = I(s_0)\, e^{-\tau(s_0,s)}$$

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



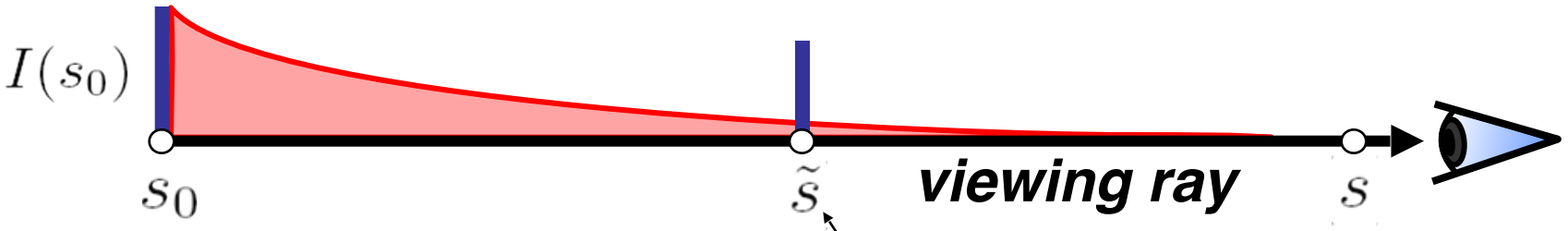
One point $\tilde{s}$ along the viewing ray emits additional radiant energy.

$$I(s) = I(s_0)\, e^{-\tau(s_0,s)}$$

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



One point $\tilde{s}$ along the viewing ray emits additional radiant energy.

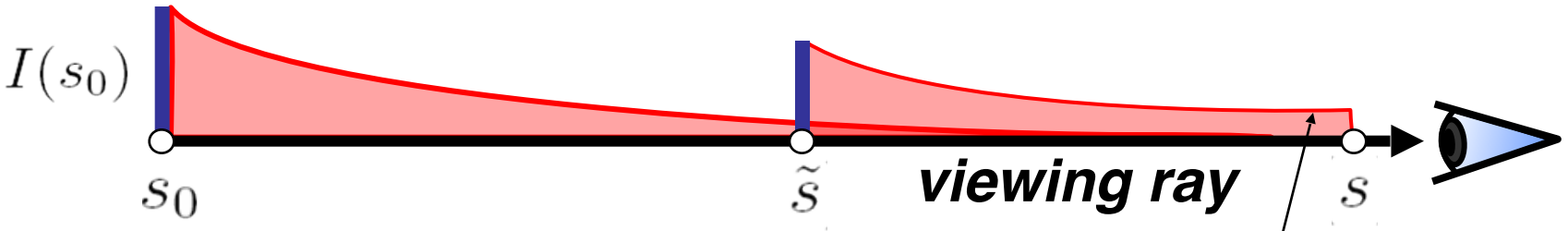
Active emission at point $\tilde{s}$

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

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



One point $\tilde{s}$ along the viewing ray emits additional radiant energy.

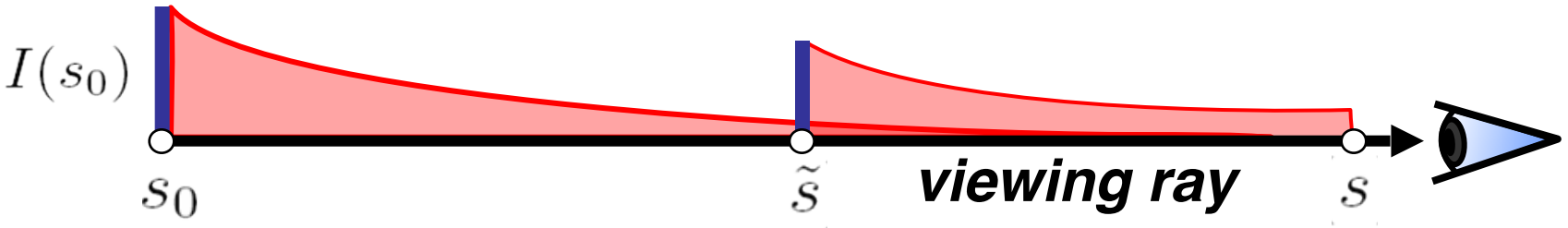
Absorption along the distance $s - \tilde{s}$

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

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



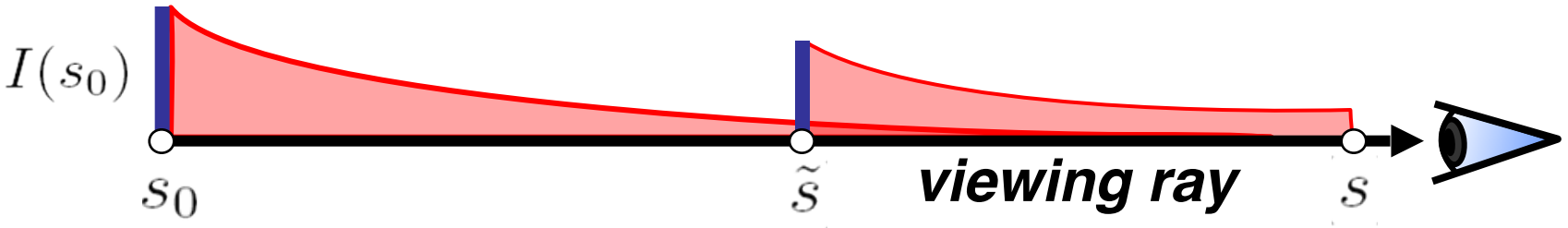
Every point $\tilde{s}$ along the viewing ray emits additional radiant energy

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

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



Every point $\tilde{s}$ along the viewing ray emits additional radiant energy

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

Ray Casting

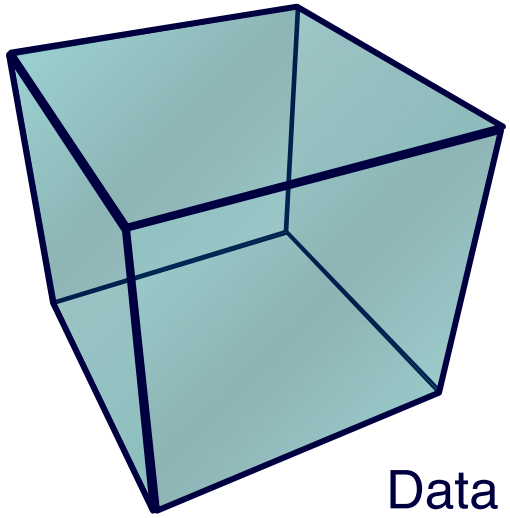
$$I(s) = + \int_{s_0}^{s} q(\tilde{s})\, e^{-\tau(\tilde{s},s)}\, d\tilde{s}$$

- Software Solution

Ray Casting

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

- Software Solution

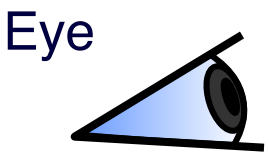


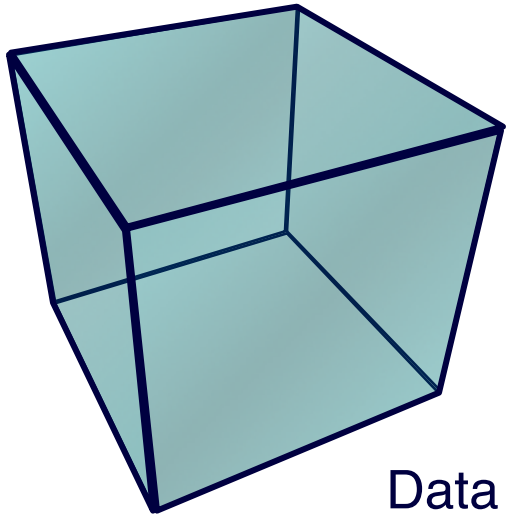
Data Set

Ray Casting

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

- Software Solution

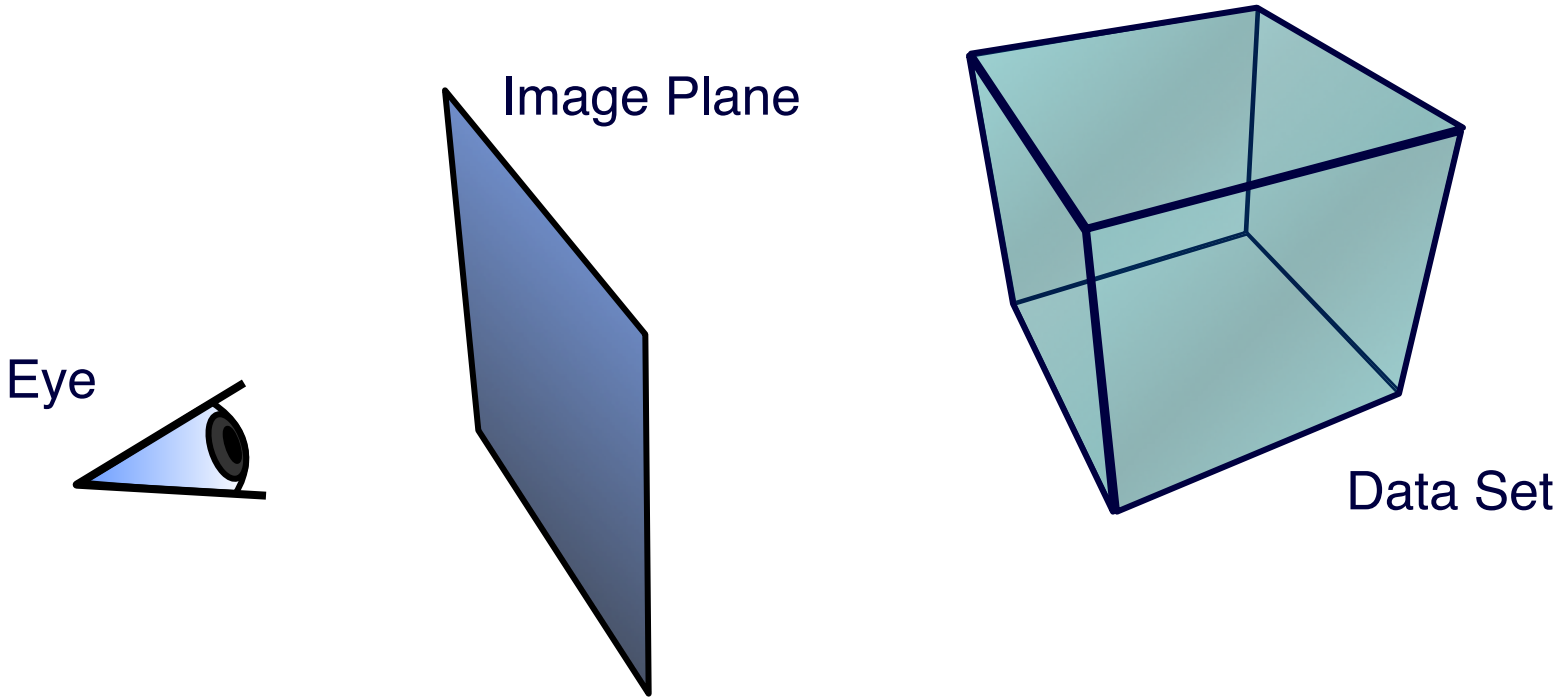




Ray Casting

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

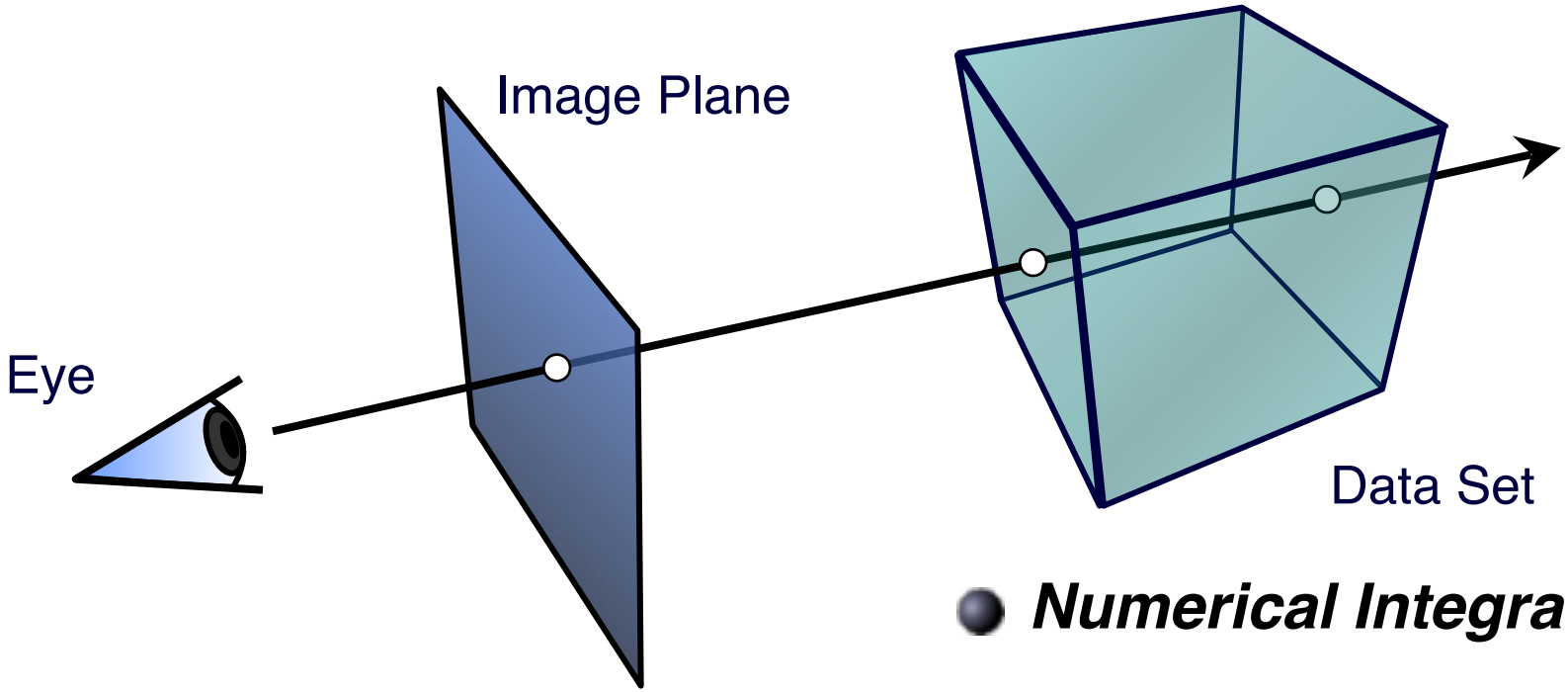
- Software Solution



Ray Casting

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

- Software Solution

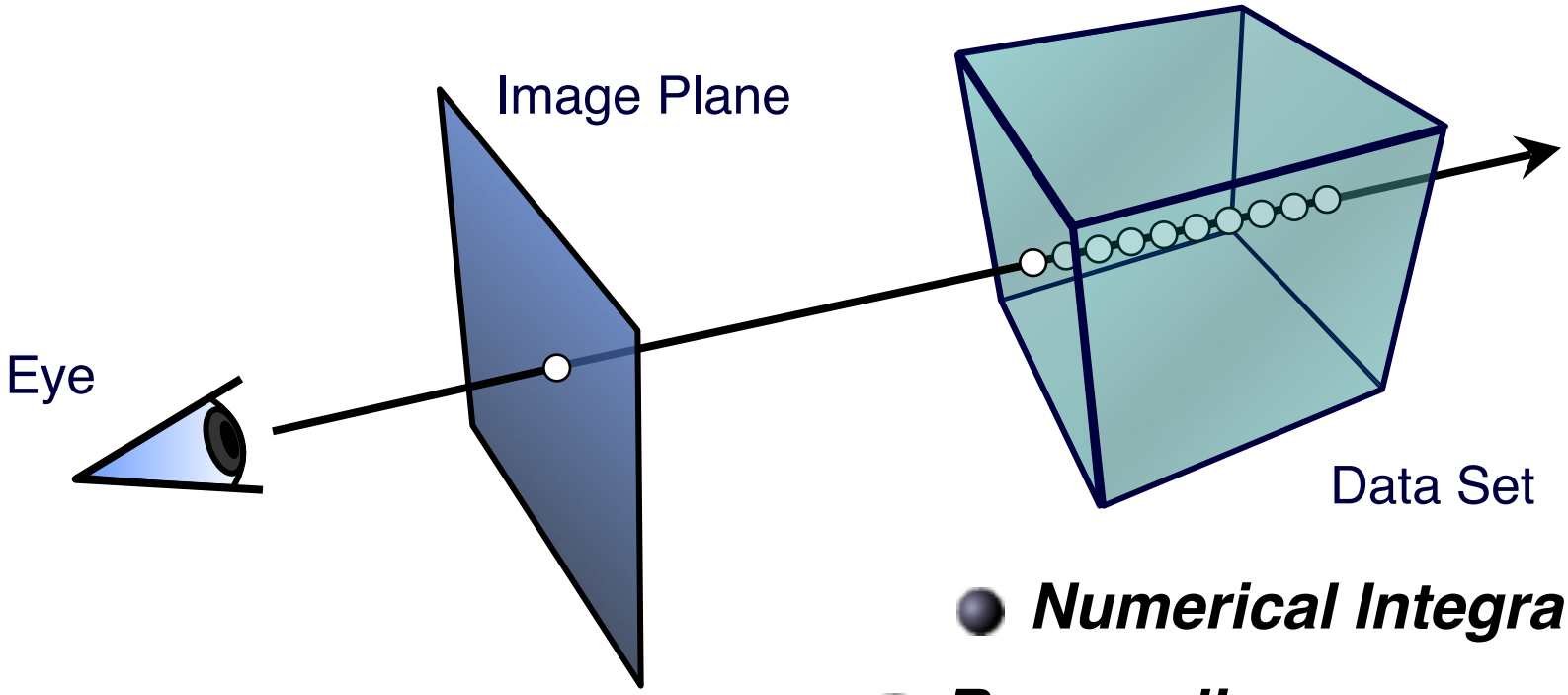


- ***Numerical Integration***

Ray Casting

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

- Software Solution



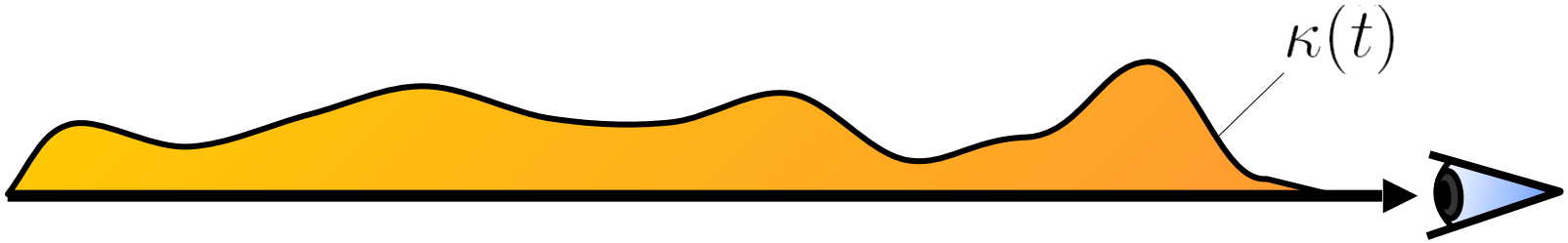
- ***Numerical Integration***
- ***Resampling***

Numerical Solution

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

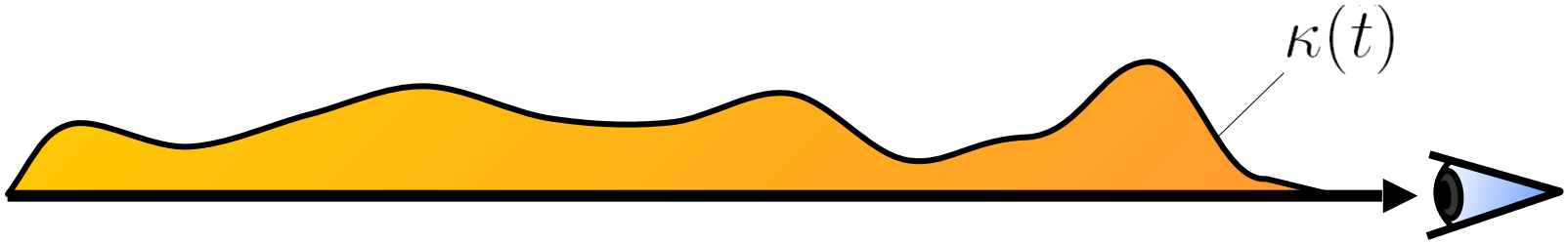
Numerical Solution

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



Numerical Solution

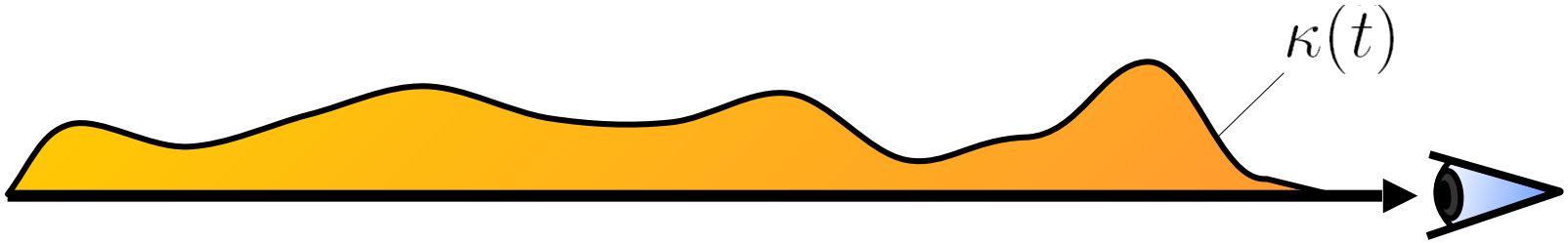
$$I(s) = + \int_{s_0}^{s} q(\tilde{s})\, e^{-\tau(\tilde{s},s)}\, d\tilde{s}$$



Extinction: $\tau(0,t) = \int_0^t \kappa(\hat{t})\, d\hat{t}$

Numerical Solution

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



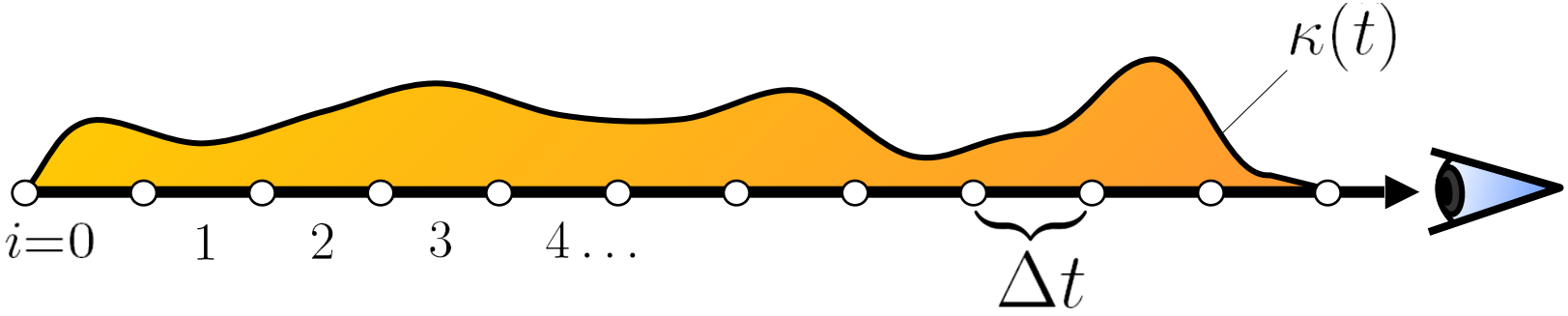
Extinction: $\tau(0,t) = \int_0^t \kappa(\hat{t})\, d\hat{t}$

Approximate Integral by Riemann sum:

$$\tau(0,t) \approx$$

Numerical Solution

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



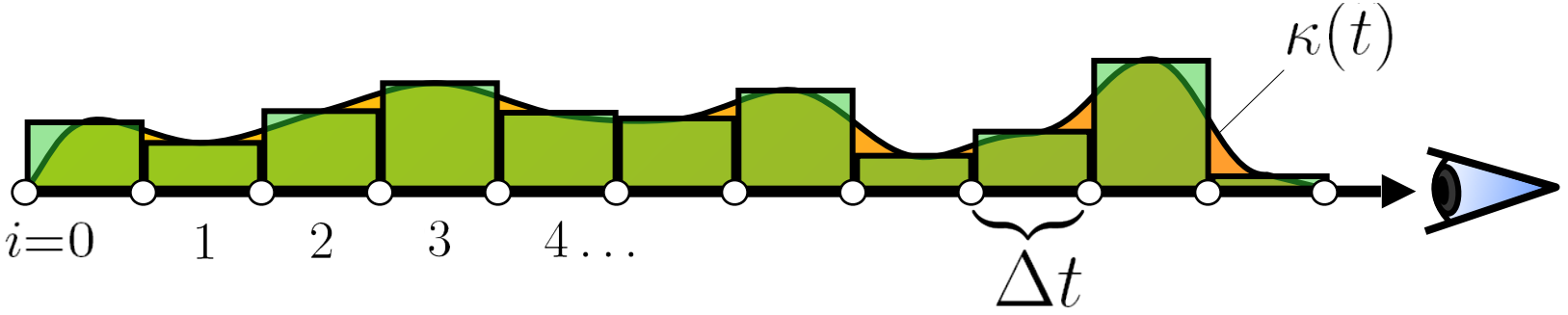
Extinction: $\tau(0,t) = \int_0^t \kappa(\hat{t})\, d\hat{t}$

Approximate Integral by Riemann sum:

$$\tau(0,t) \approx$$

Numerical Solution

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



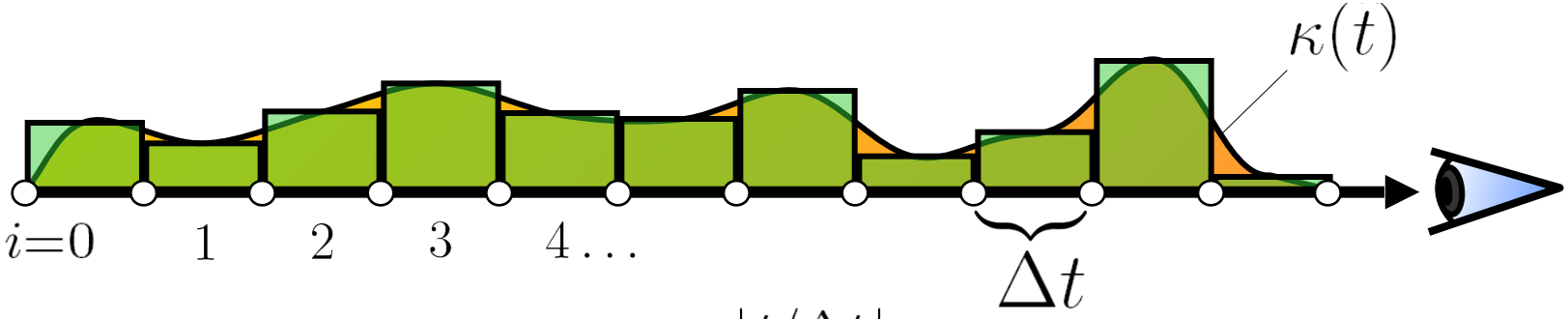
Extinction: $\tau(0,t) = \int_0^t \kappa(\hat{t})\, d\hat{t}$

Approximate Integral by Riemann sum:

$$\tau(0,t) \approx \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

Numerical Solution

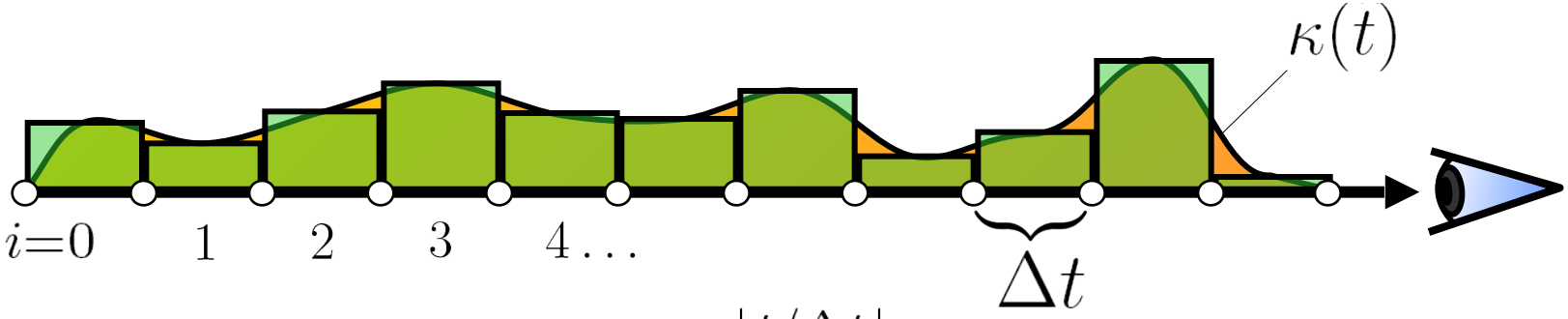
$$I(s) = + \int_{s_0}^{s} q(\tilde{s})\, e^{-\tau(\tilde{s},s)}\, d\tilde{s}$$



$$\tau(0,t) \approx \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

Numerical Solution

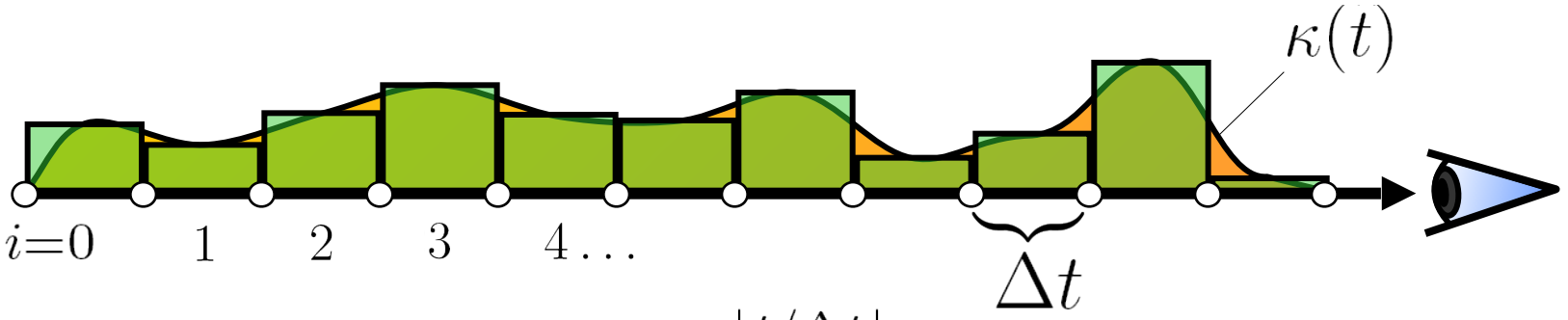
$$I(s) = + \int_{s_0}^{s} q(\tilde{s})\, e^{-\tau(\tilde{s},s)}\, d\tilde{s}$$



$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

Numerical Solution

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

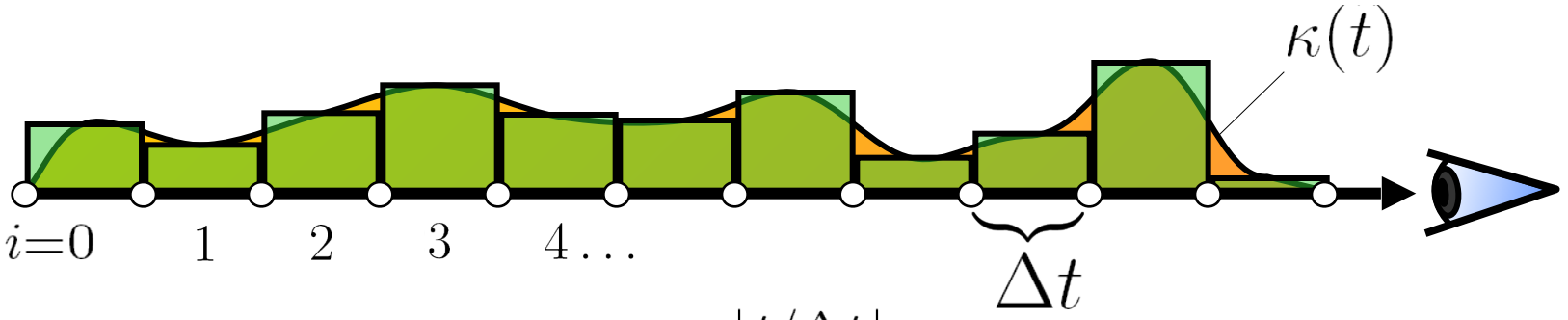


$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

$$e^{-\tilde{\tau}(0,t)} = e^{-\sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t}$$

Numerical Solution

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

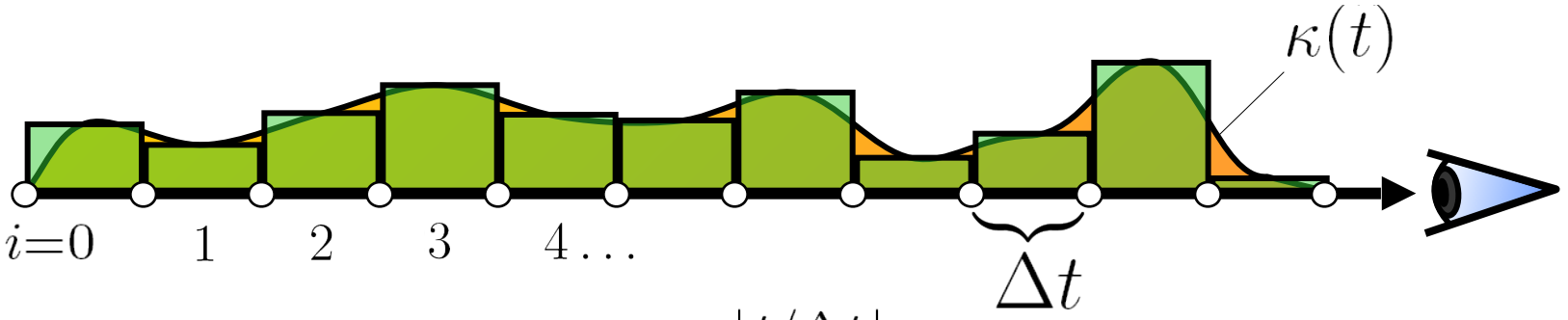


$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

$$e^{-\tilde{\tau}(0,t)} = e^{-\sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t}$$

Numerical Solution

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

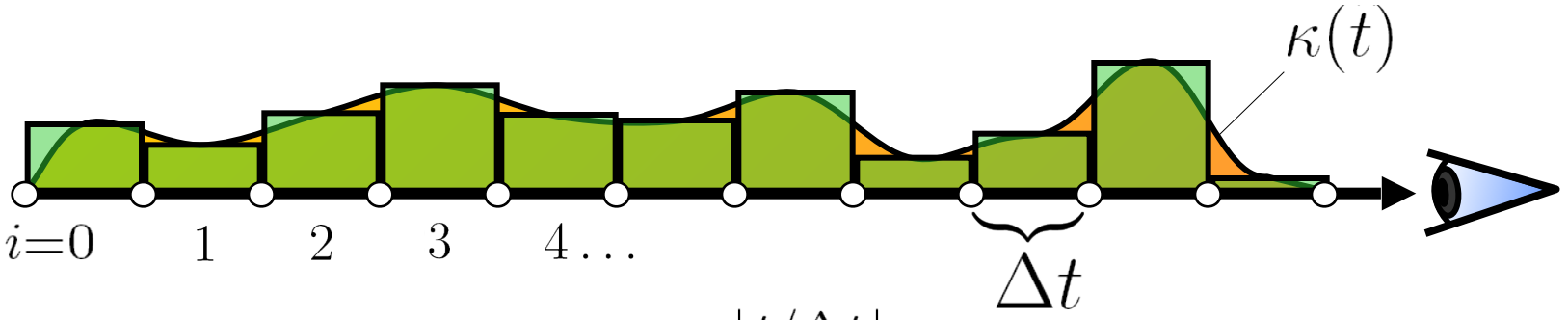


$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Numerical Solution

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



$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

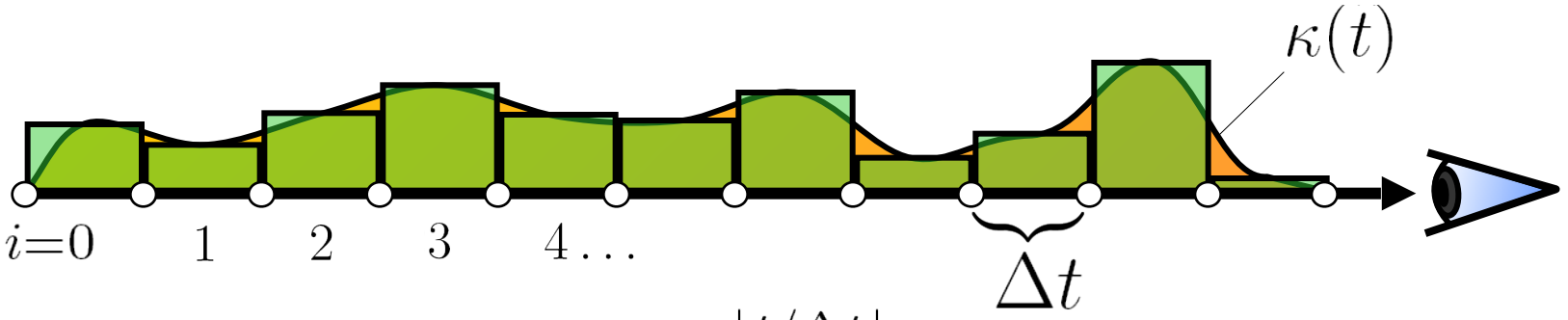
$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Now we introduce opacity:

$$A_i = 1 - e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Numerical Solution

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



$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

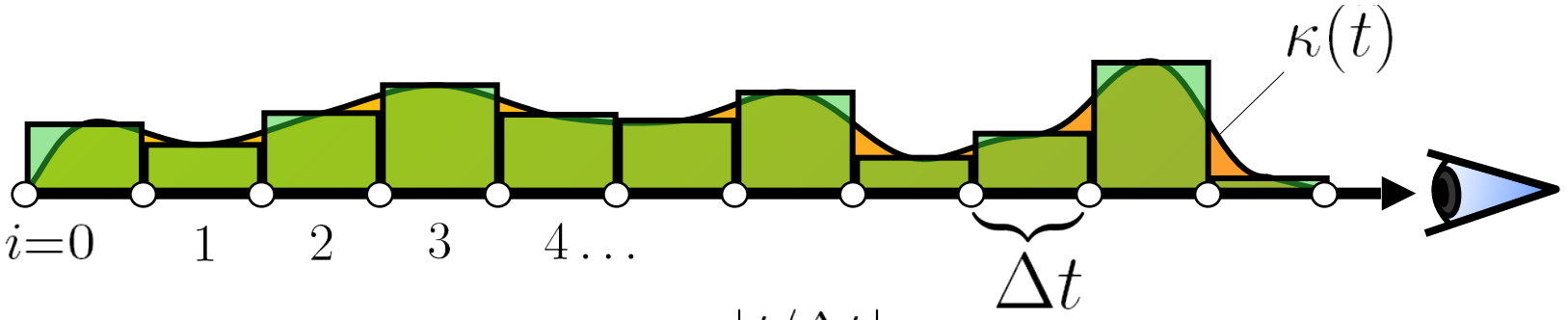
$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Now we introduce opacity:

$$1 - A_i = e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Numerical Solution

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



$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

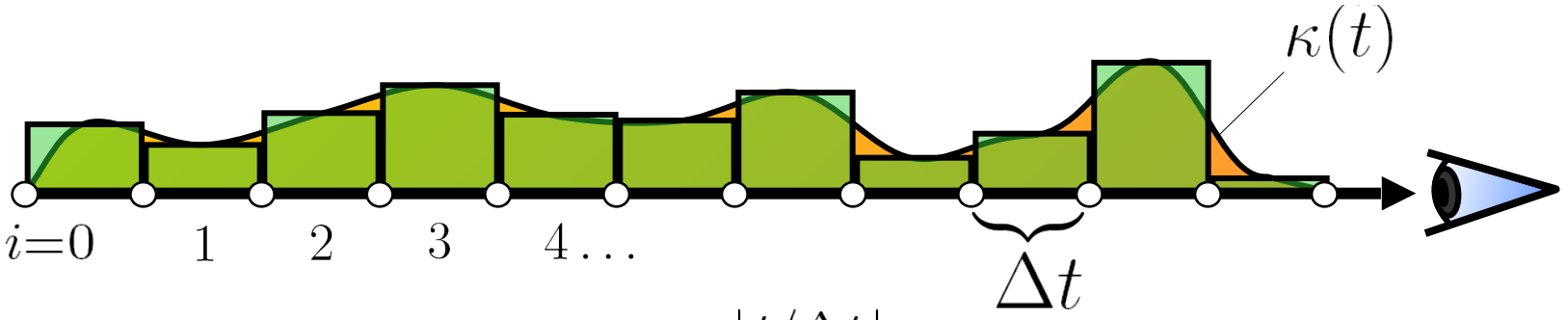
$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Now we introduce opacity:

$$1 - A_i = e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Numerical Solution

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



$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

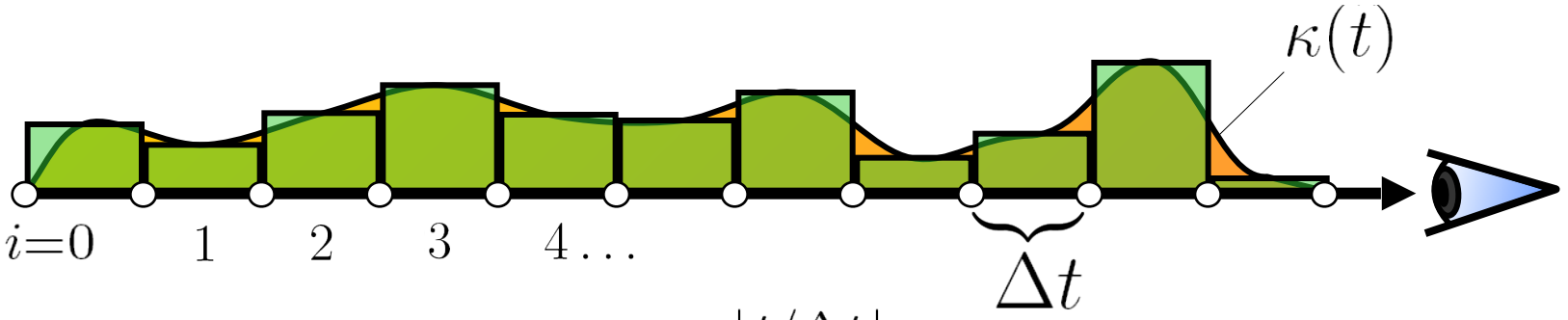
$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

Now we introduce opacity:

$$1 - A_i = e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Numerical Solution

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



$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

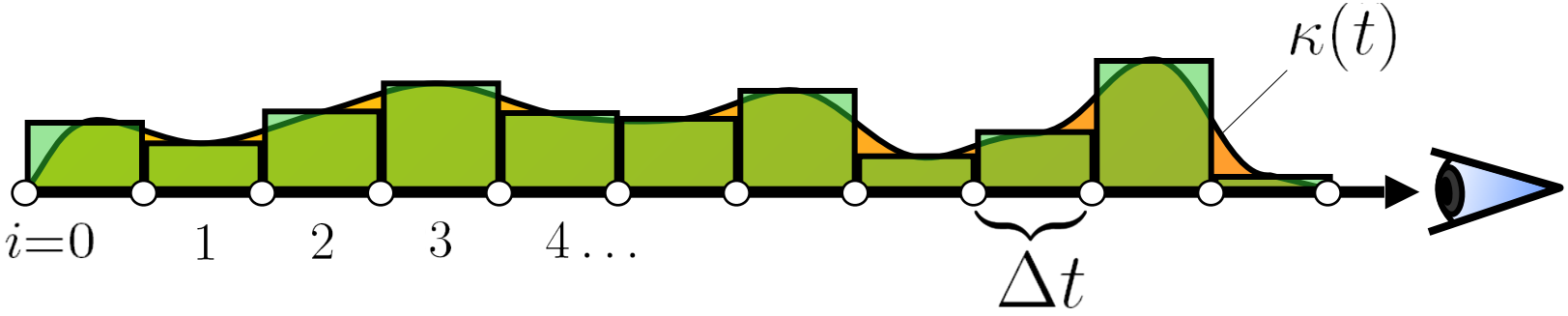
$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

Now we introduce opacity:

$$1 - A_i = e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Numerical Solution

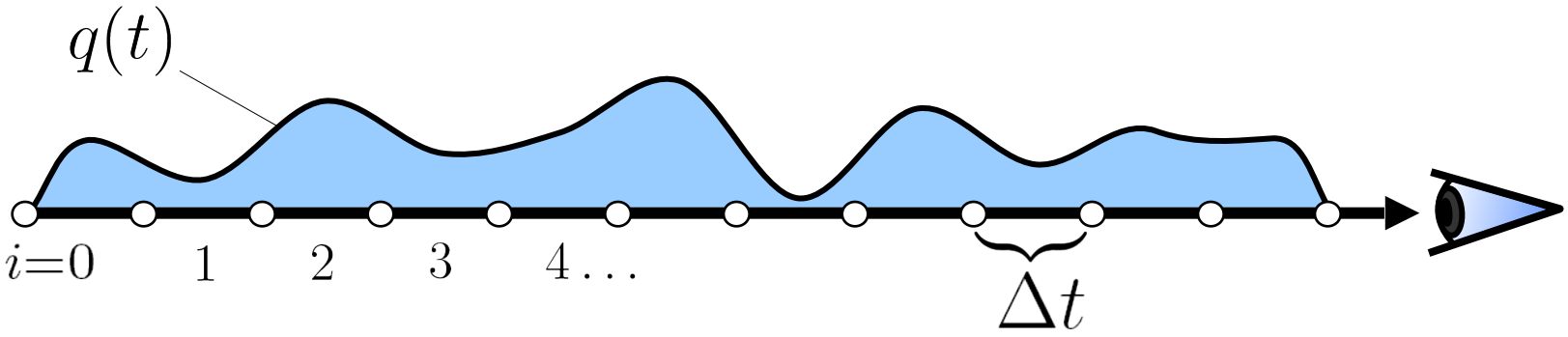
$$I(s) = + \int_{s_0}^{s} q(\tilde{s})\, e^{-\tau(\tilde{s},s)}\, d\tilde{s}$$



$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

Numerical Solution

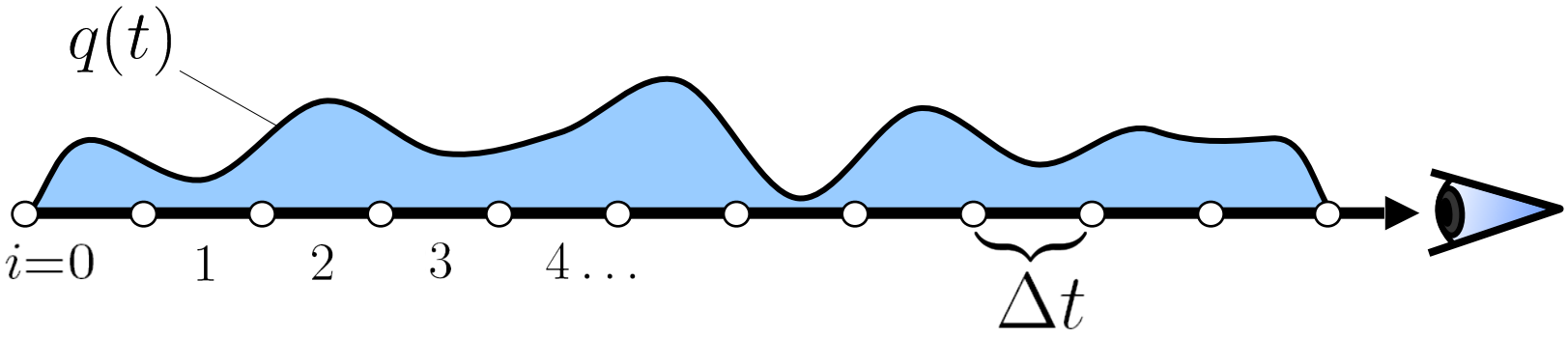
$$I(s) = + \int_{s_0}^{s} q(\tilde{s})\, e^{-\tau(\tilde{s},s)}\, d\tilde{s}$$



$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

Numerical Solution

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

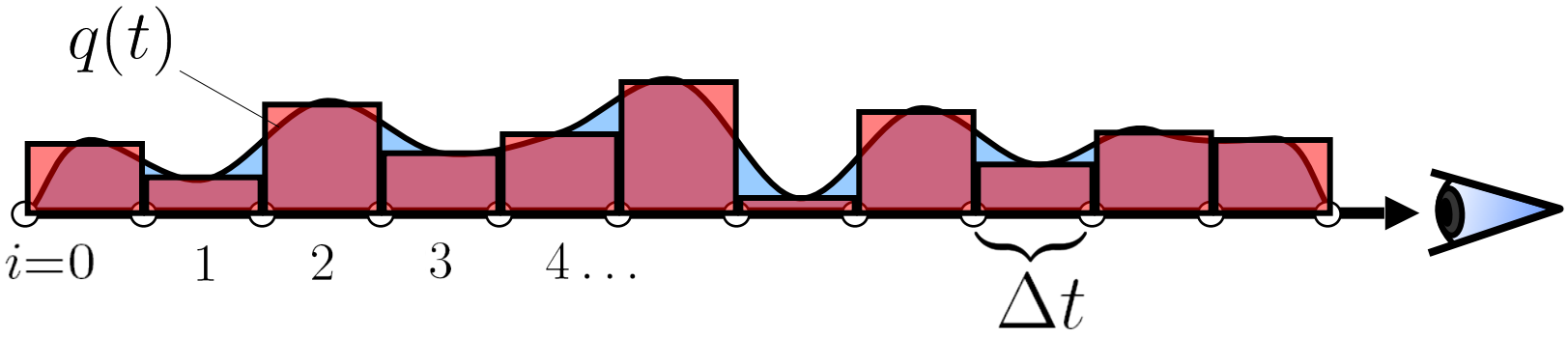


$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

$$q(t) \approx$$

Numerical Solution

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

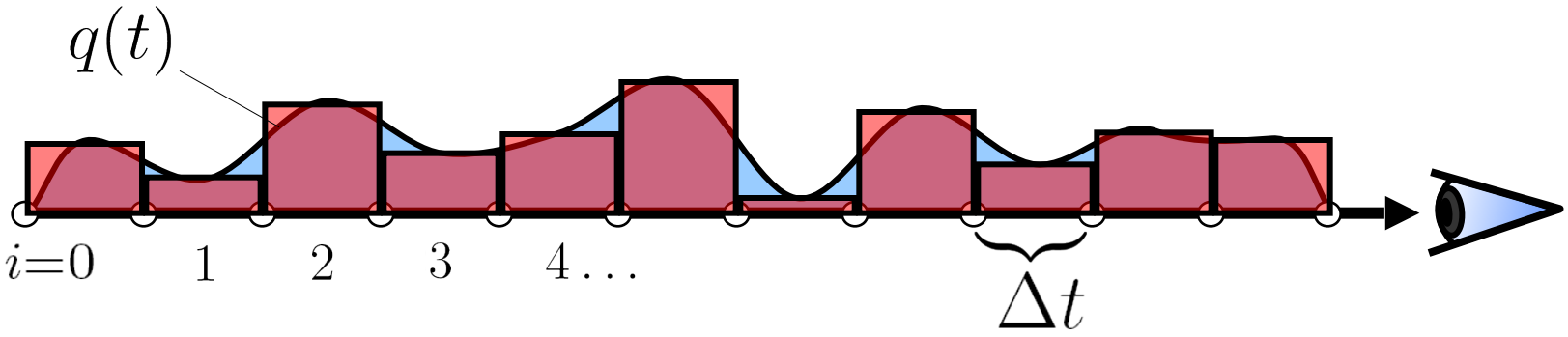


$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

$$q(t) \approx C_i = c(i \cdot \Delta t)\, \Delta t$$

Numerical Solution

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



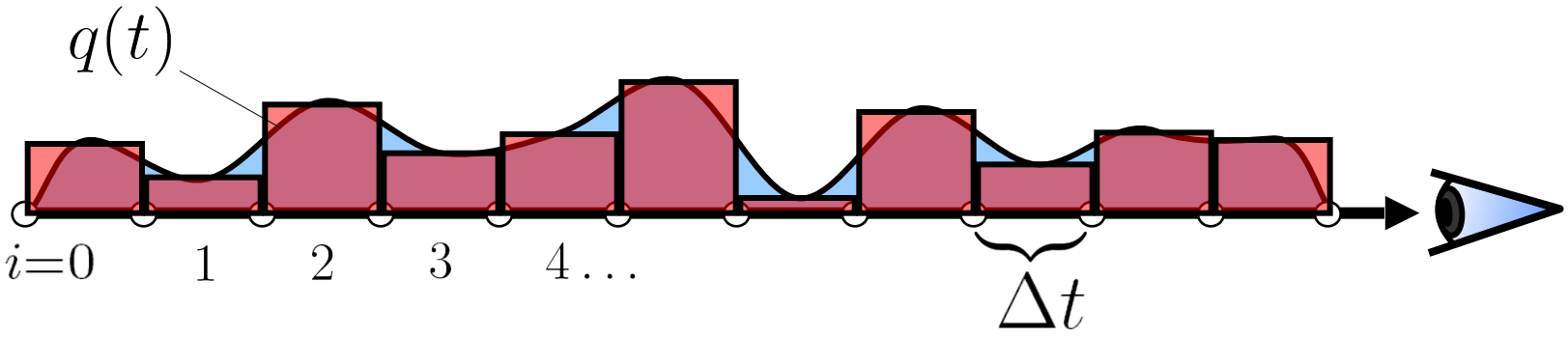
$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

$$q(t) \approx C_i = c(i \cdot \Delta t)\, \Delta t$$

$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i\, e^{-\tilde{\tau}(0,t)}$$

Numerical Solution

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



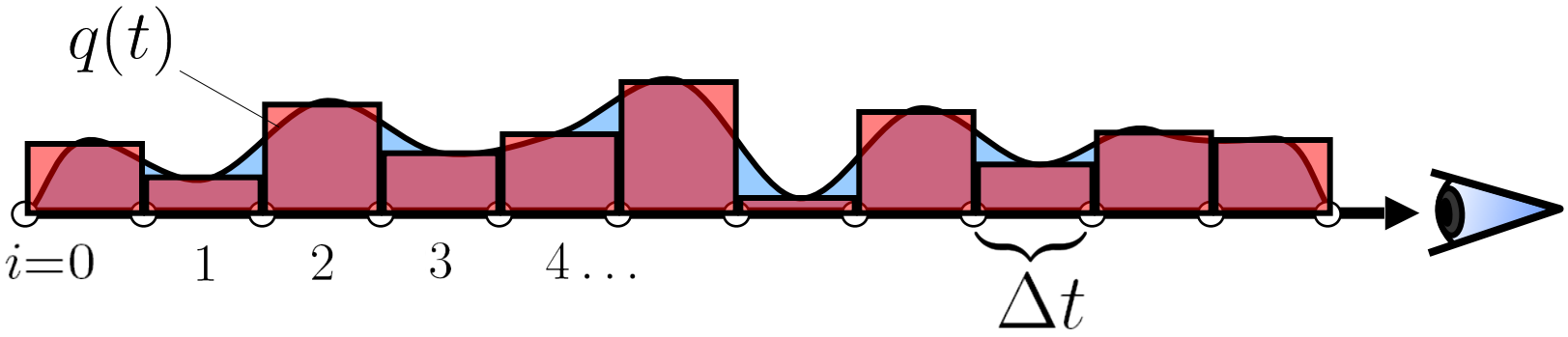
$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

$$q(t) \approx C_i = c(i \cdot \Delta t)\, \Delta t$$

$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i\, e^{-\tilde{\tau}(0,t)}$$

Numerical Solution

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



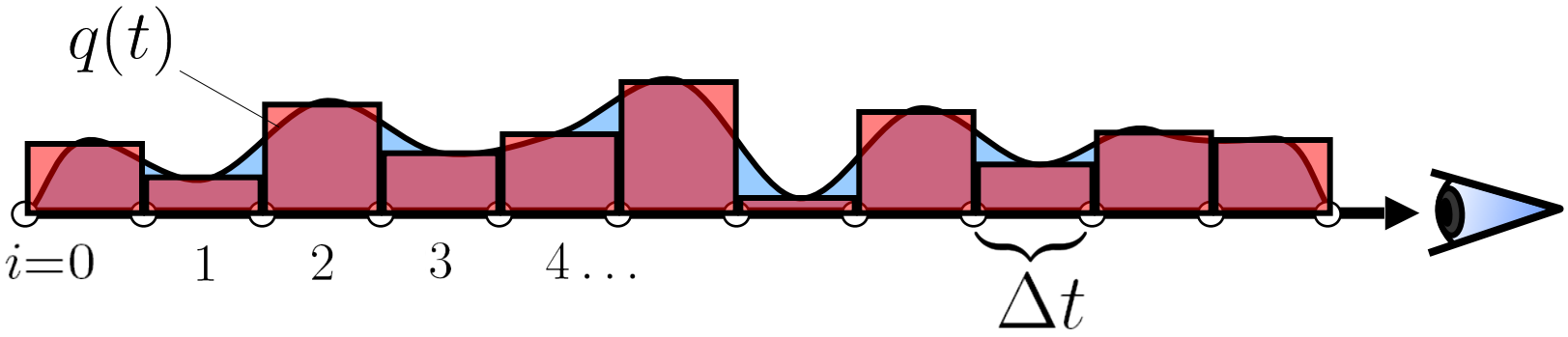
$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

$$q(t) \approx C_i = c(i \cdot \Delta t)\, \Delta t$$

$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \prod_{j=0}^{i-1} (1 - A_j)$$

Numerical Solution

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



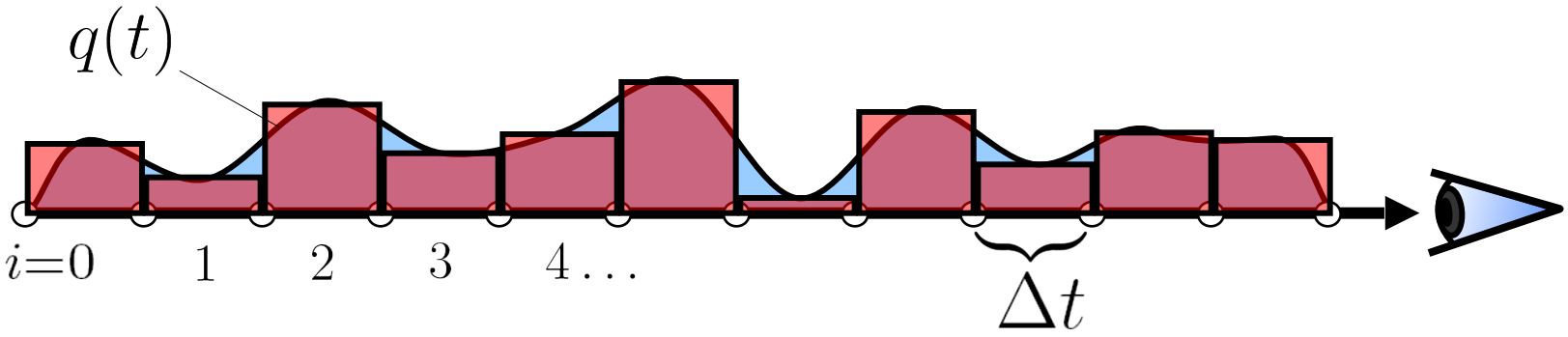
$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

$$q(t) \approx C_i = c(i \cdot \Delta t)\, \Delta t$$

$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \prod_{j=0}^{i-1} (1 - A_j)$$

Numerical Solution

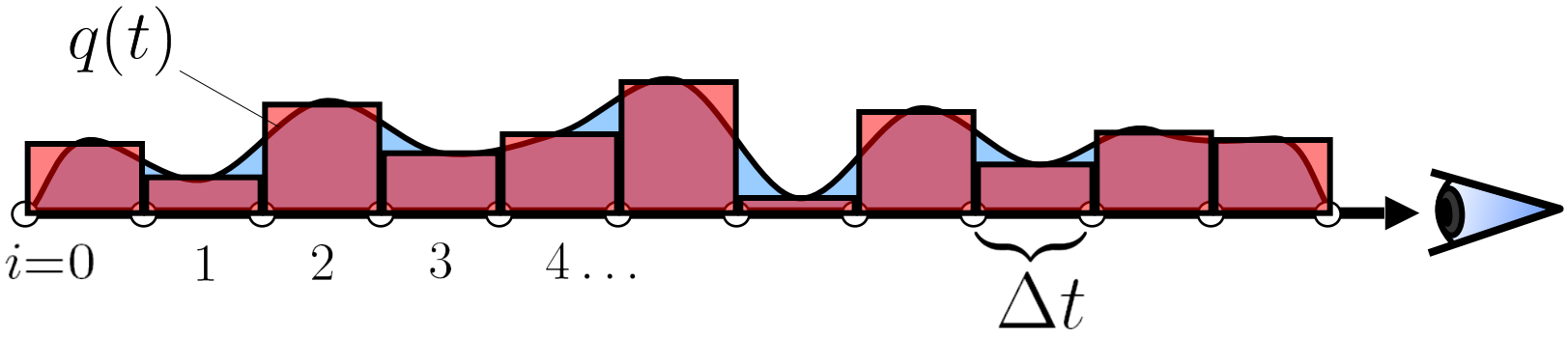
$$I(s) = + \int_{s_0}^{s} q(\tilde{s})\, e^{-\tau(\tilde{s},s)}\, d\tilde{s}$$



$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \prod_{j=0}^{i-1} (1 - A_j)$$

Numerical Solution

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



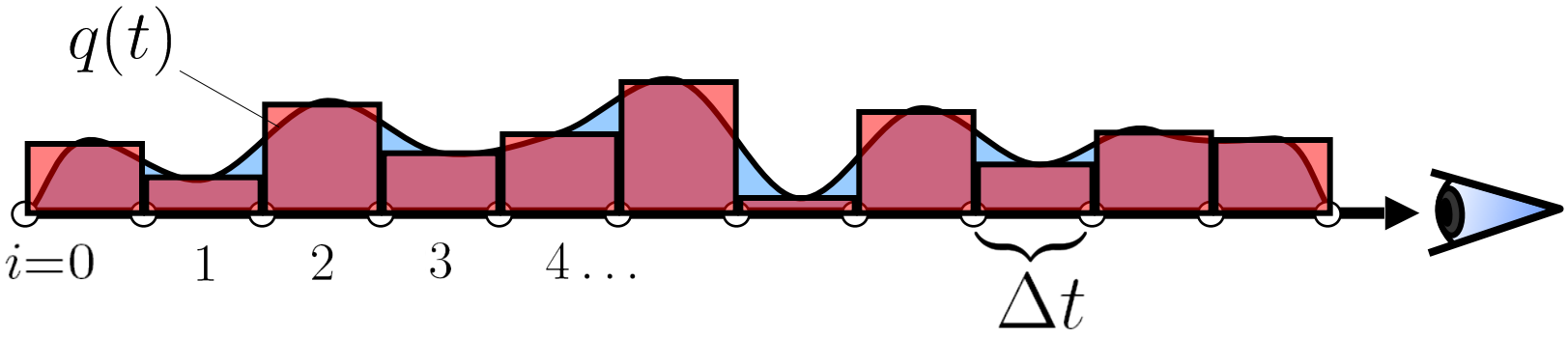
$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \prod_{j=0}^{i-1} (1 - A_j)$$

can be computed recursively

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

Numerical Solution

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



$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \prod_{j=0}^{i-1} (1 - A_j)$$

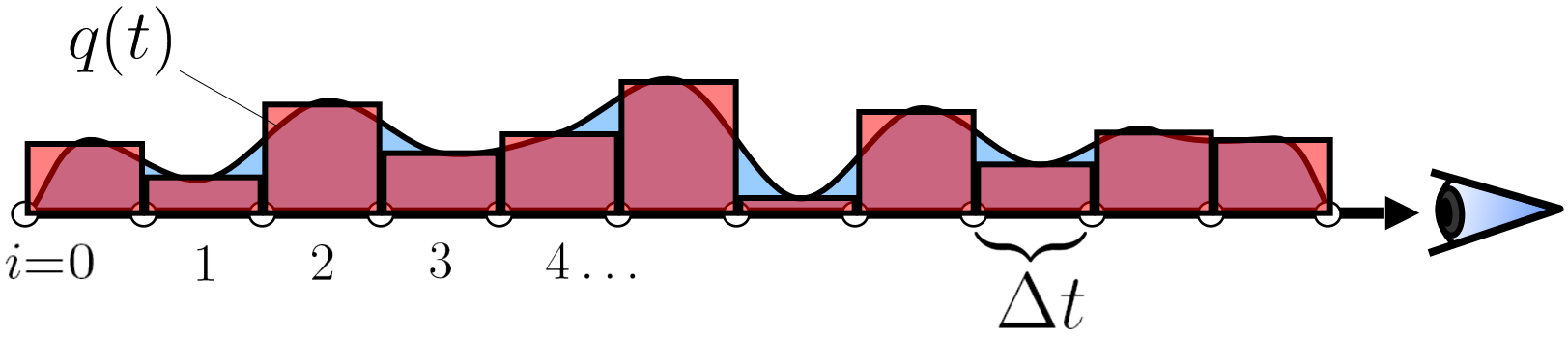
can be computed recursively

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

Radiant energy observed at position ***i***

Numerical Solution

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



$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \prod_{j=0}^{i-1} (1 - A_j)$$

can be computed recursively

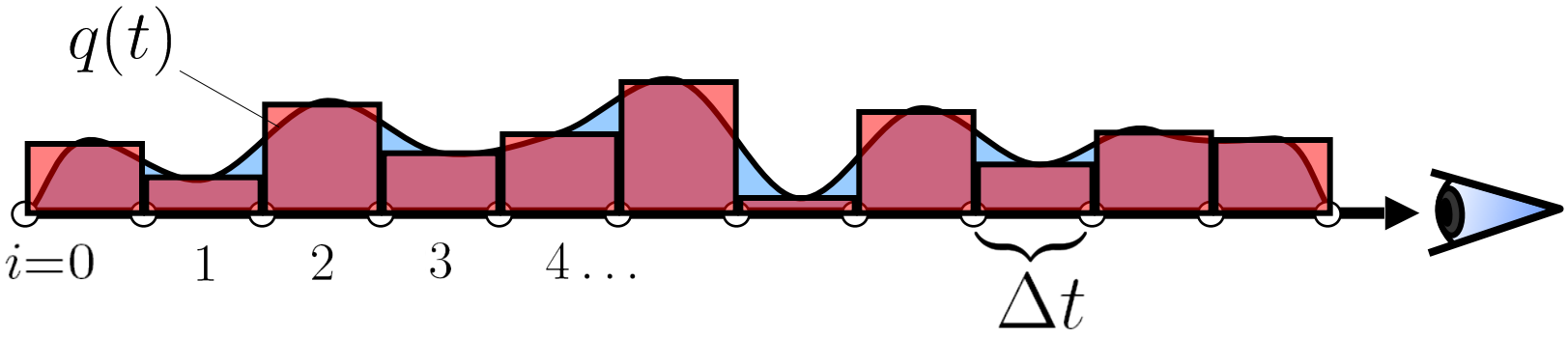
$$C'_i = C_i + (1 - A_i) C'_{i-1}$$

Radiant energy observed at position ***i***

Radiant energy emitted at position ***i***

Numerical Solution

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



$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \prod_{j=0}^{i-1} (1 - A_j)$$

can be computed recursively

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

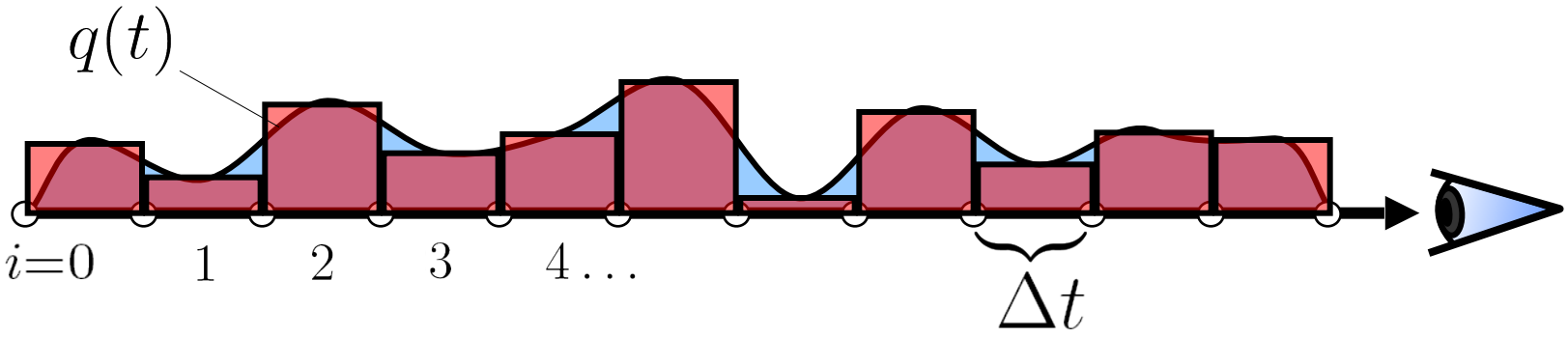
Radiant energy observed at position ***i***

Radiant energy emitted at position ***i***

Radiant energy observed at position ***i–1***

Numerical Solution

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



$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \prod_{j=0}^{i-1} (1 - A_j)$$

can be computed recursively

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

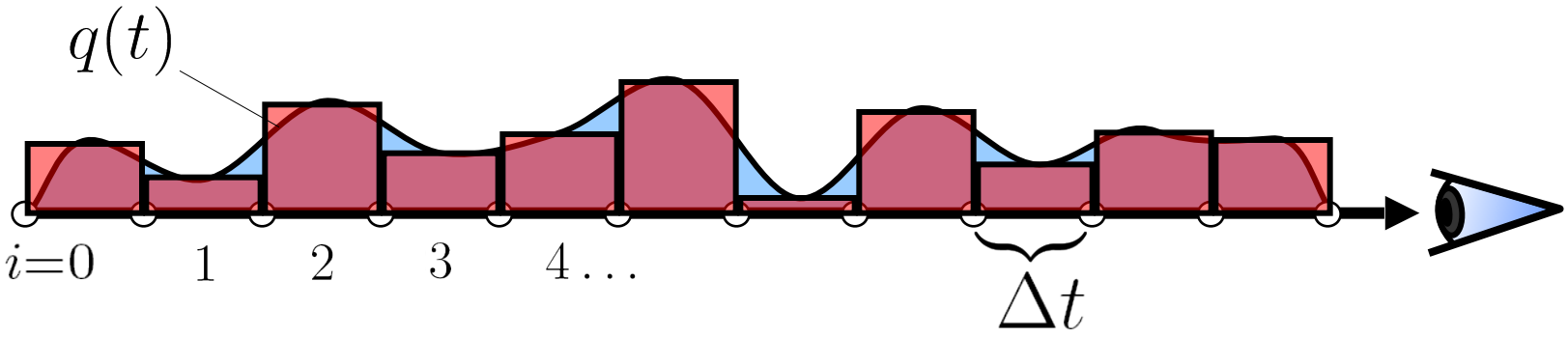
Radiant energy observed at position ***i***

Radiant energy emitted at position ***i***

Absorption at position ***i***

Radiant energy observed at position ***i–1***

Numerical Solution

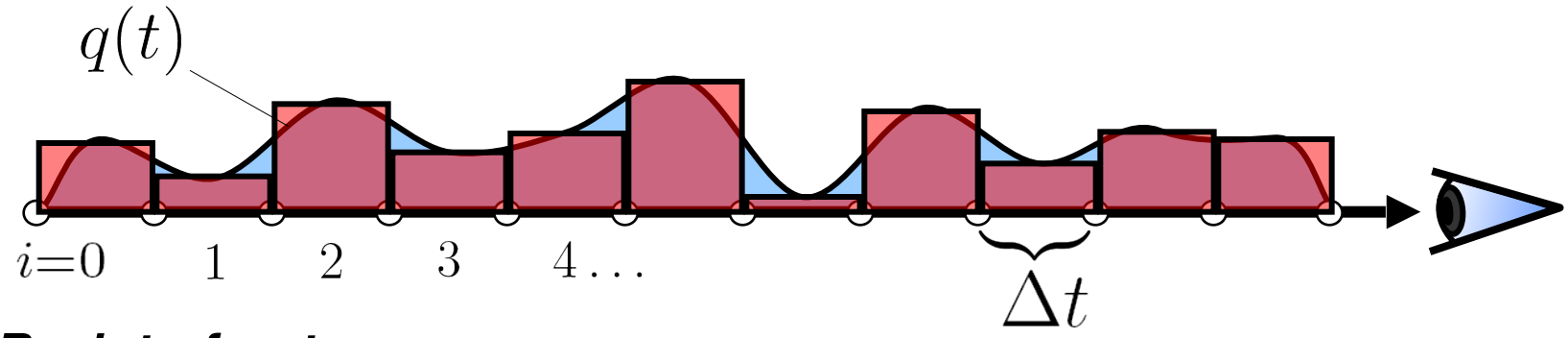


$$\tilde{C} = \sum_{i=0}^{\lfloor T/\Delta t \rfloor} C_i \prod_{j=0}^{i-1} (1 - A_j)$$

can be computed recursively

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

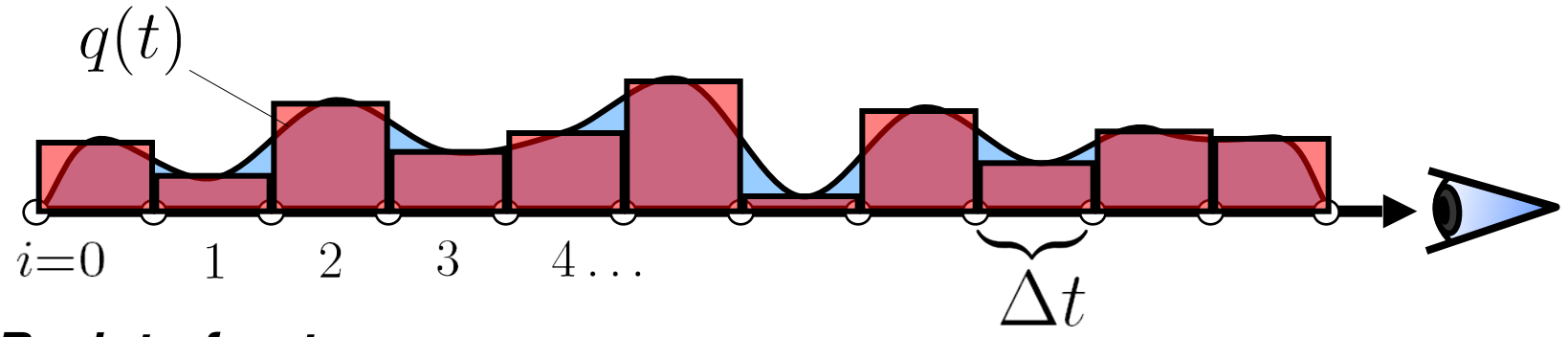
Numerical Solution



Back-to-front compositing

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

Numerical Solution



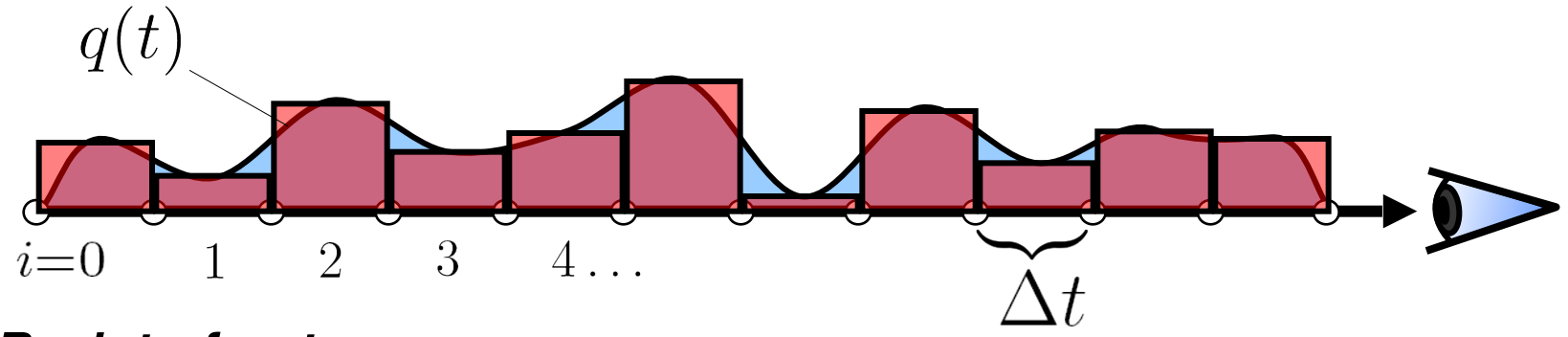
Back-to-front compositing

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

Front-to-back compositing

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

Numerical Solution



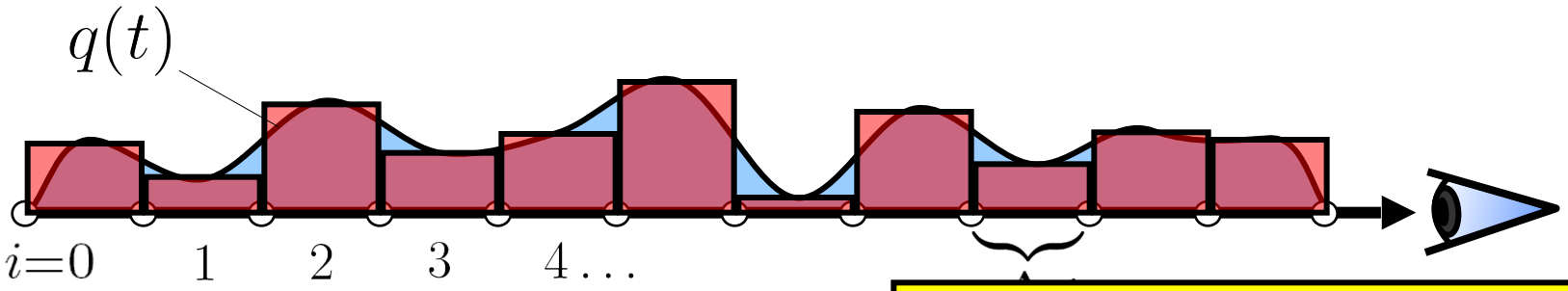
Back-to-front compositing

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

Front-to-back compositing

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

Numerical Solution



Back-to-front compositing

$$C'_i = C_i + (1 -$$

Early Ray Termination:

Stop the calculation when

$$A'_i \approx 1$$

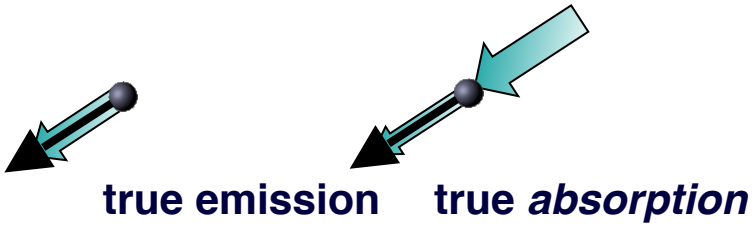
Front-to-back compositing

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

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

Summary

- Emission Absorption Model



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

- Numerical Solutions

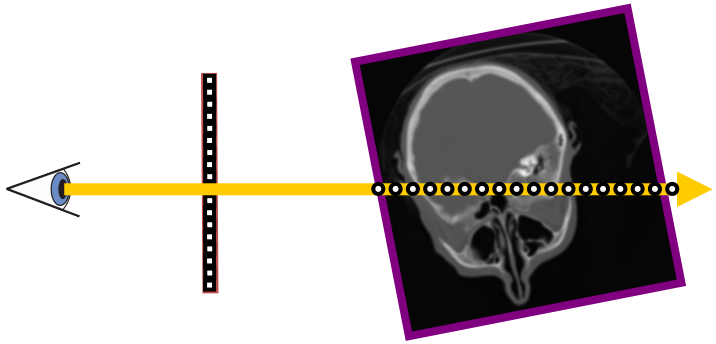
Back-to-front iteration

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

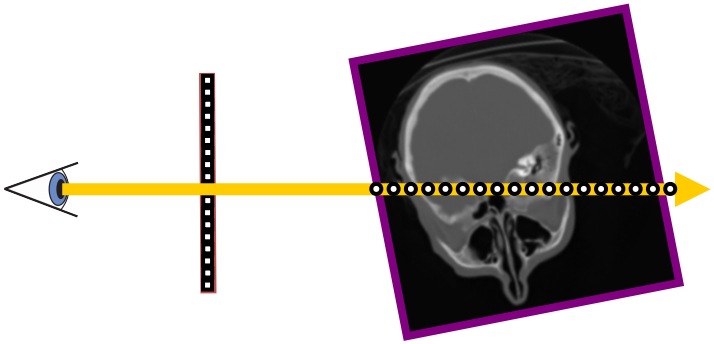
Front-to-back iteration

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

Direct Volume Rendering Pipeline

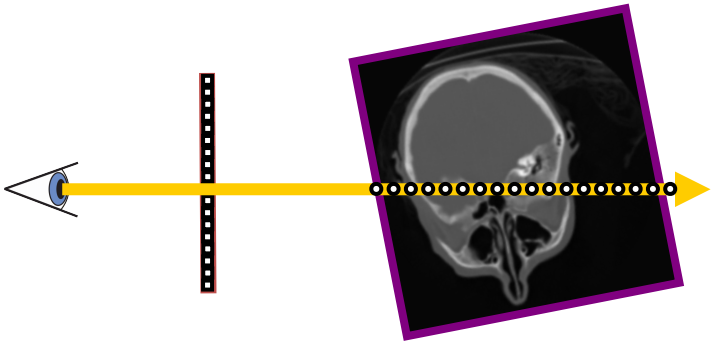


Direct Volume Rendering Pipeline



SAMPLING → CLASSIFICATION & ILLUMINATION → COMPOSITING

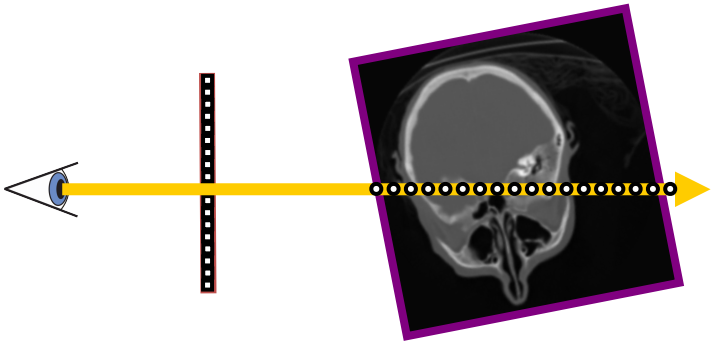
Direct Volume Rendering Pipeline



SAMPLING → CLASSIFICATION & ILLUMINATION → COMPOSITING

CLASSIFICATION & ILLUMINATION → SAMPLING

Direct Volume Rendering Pipeline

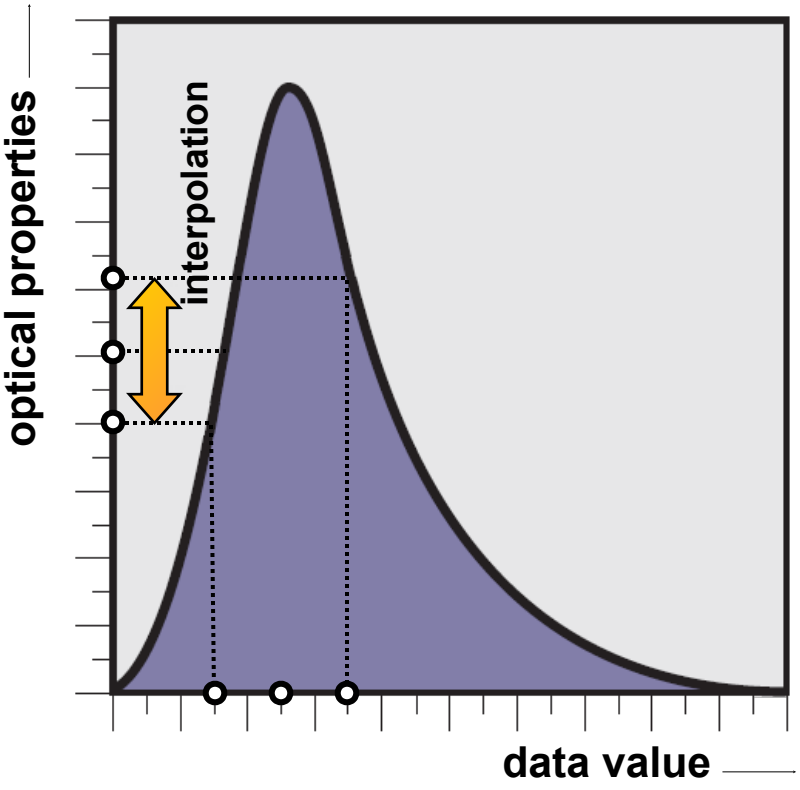


SAMPLING → CLASSIFICATION & ILLUMINATION → COMPOSITING

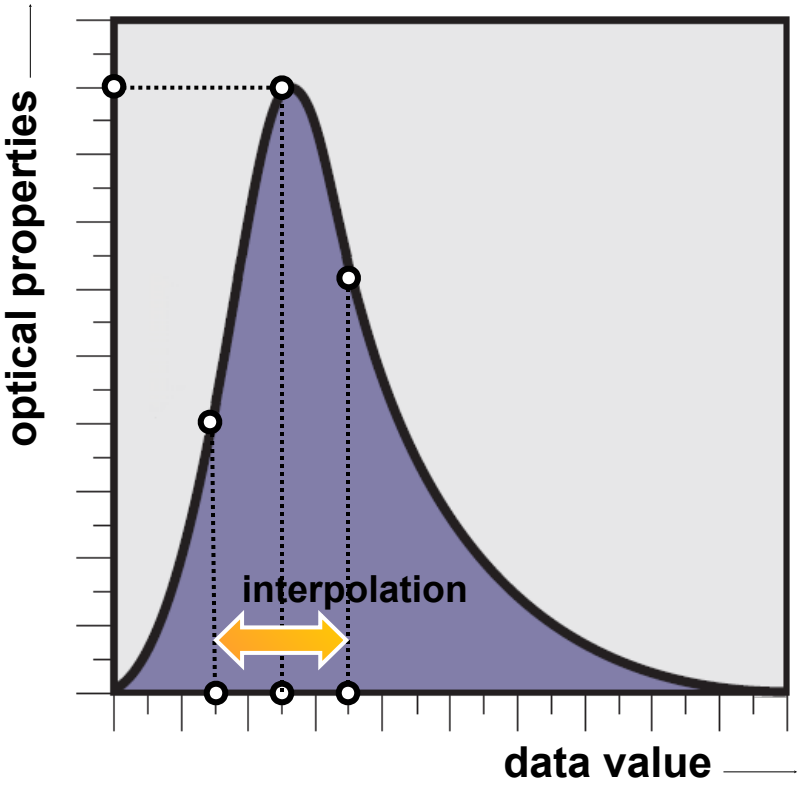
CLASSIFICATION & ILLUMINATION → SAMPLING → COMPOSITING

Pre- vs Post-Interpolative Classification

PRE-INTERPOLATIVE



POST-INTERPOLATIVE

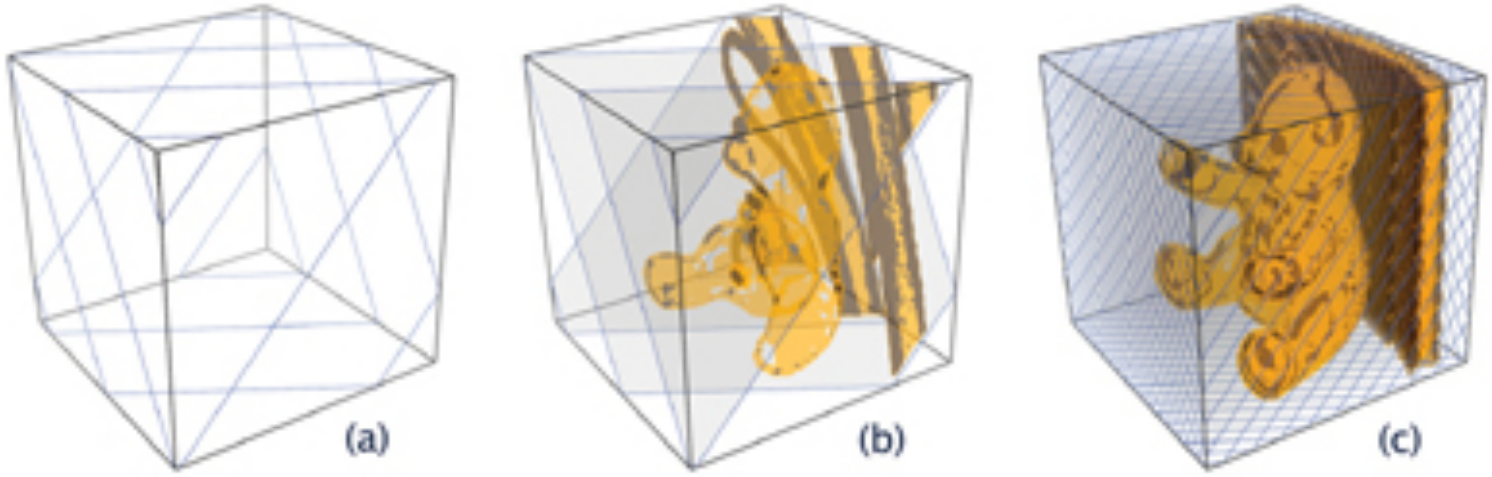


Algorithms for DVR

- OBJECT SPACE
  - Splatting, cell-projection (Westover'90, Chen'04)
    - project voxels to the image plane
    - spherical kernel
    - back-to-front
    - uniform regular grids
  - 2d- slicing (2D textures)
  - 3d- slicing (3D textures)
- IMAGE SPACE
  - Ray Casting
- FOURIER/WAVELET Domain

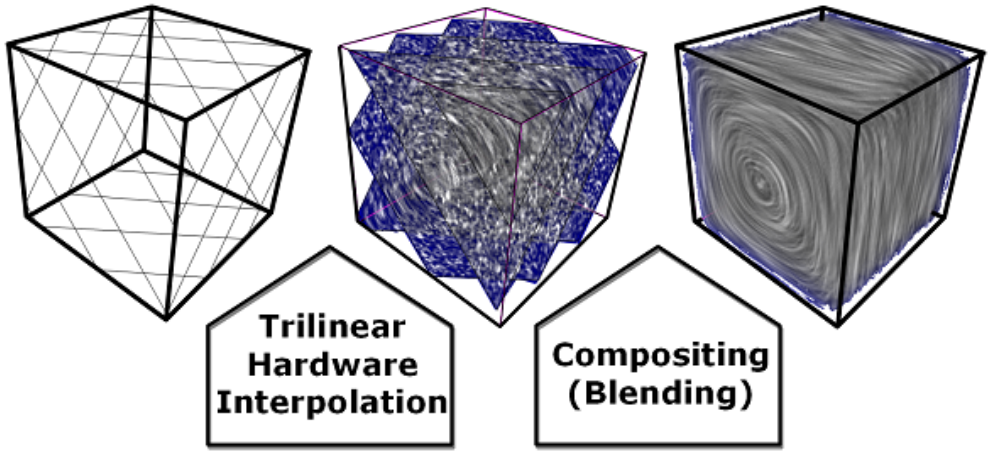
Slice-Based Volume Rendering (SBVR)

- No volumetric primitive in graphics API
- Proxy geometry - polygon primitives as slices through volume
- Texture polygons with volumetric data
- Draw slices in sorted order – back-to-front
- Use fragment shader to perform compositing (blending)



Volumetric Data

- Voxel data sent to GPU memory as
  - Stack of 2D textures
  - 3D texture
- Leverage graphics pipeline



Instructions for setting up 3D texture in OpenGL

- http://gpwiki.org/index.php/OpenGL_3D_Textures

Proxy Geometry

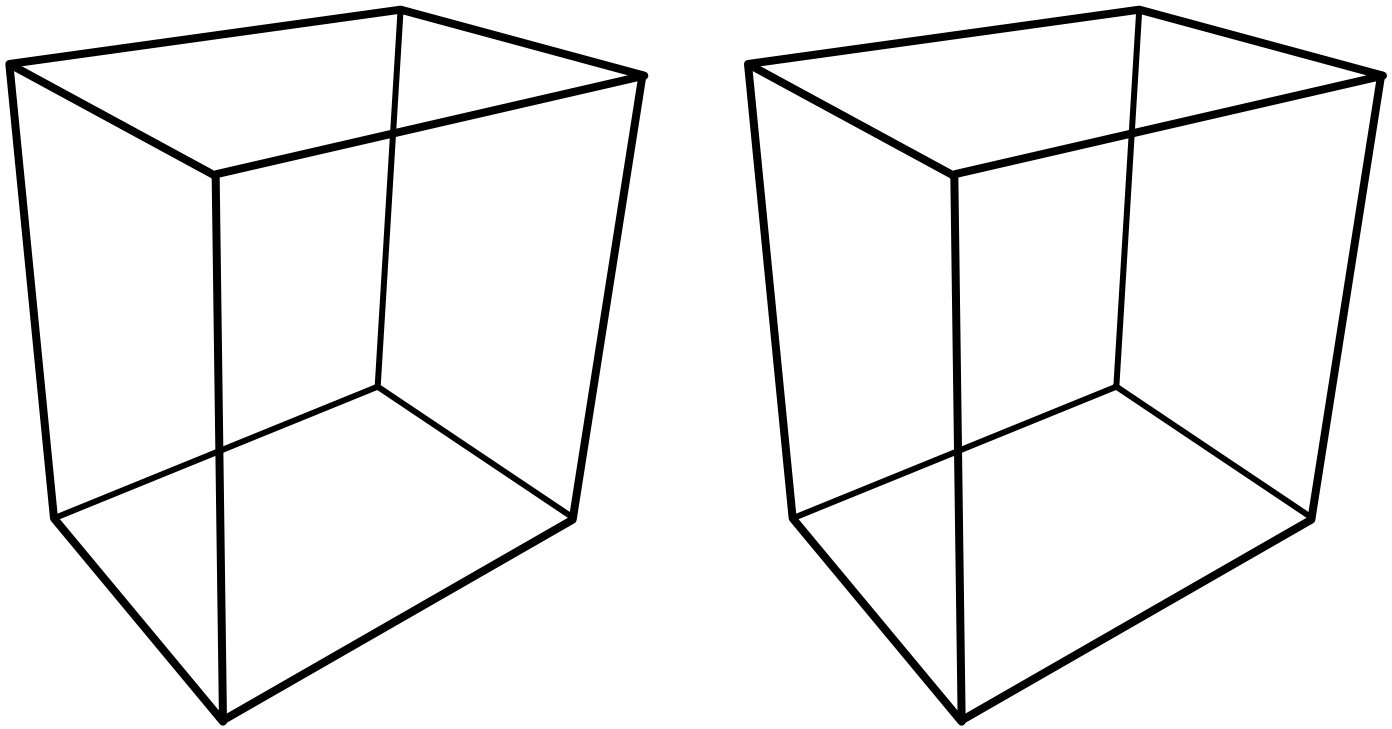
- Slices through 3D voxel data
- 3D voxel data = 3D texture on GPU
- Assign texture coordinate to every slice vertex
  - CPU or vertex shader

Texture-based Approaches

- No volumetric hardware-primitives!

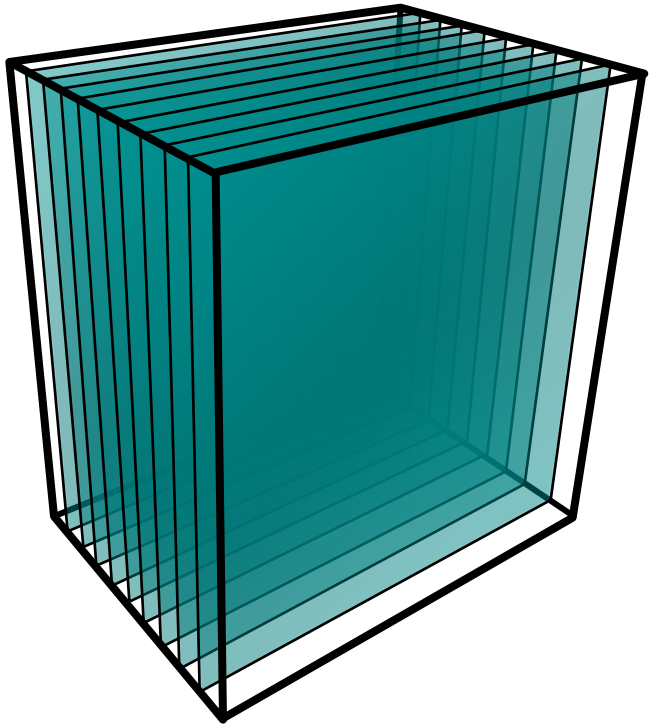
Texture-based Approaches

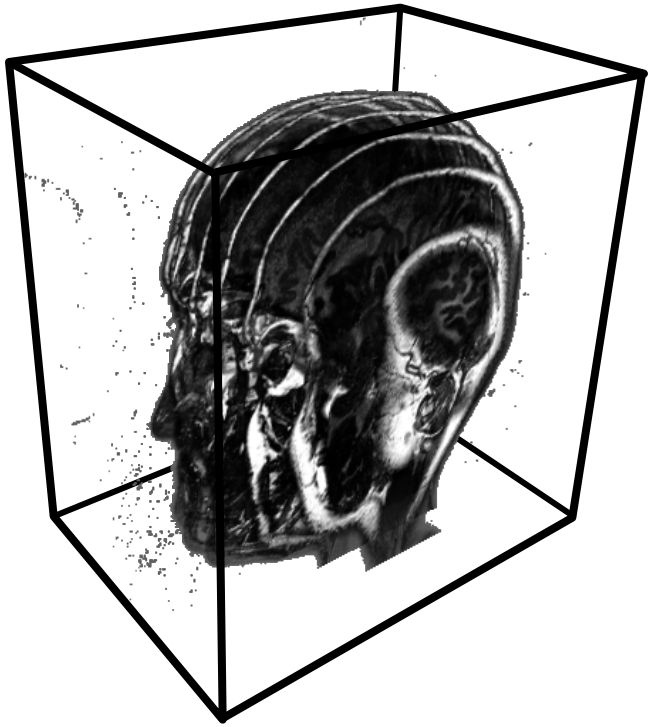
- No volumetric hardware-primitives!
- ⇨ Proxy geometry (Polygonal Slices)



Texture-based Approaches

- No volumetric hardware-primitives!
- ⇨ Proxy geometry (Polygonal Slices)





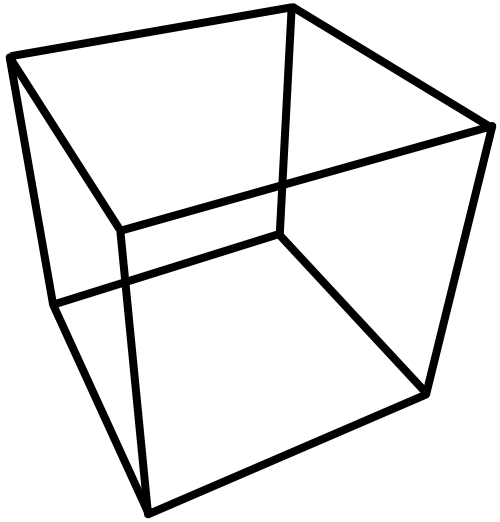
2D Textures

- Draw the volume as a stack of 2D textures
  Bilinear Interpolation in Hardware

2D Textures

- Draw the volume as a stack of 2D textures
  Bilinear Interpolation in Hardware

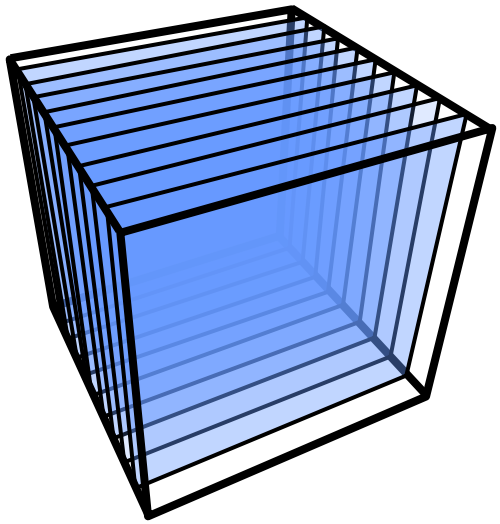
⇨ Decompostition into axis/object-aligned slices



2D Textures

- Draw the volume as a stack of 2D textures
  Bilinear Interpolation in Hardware

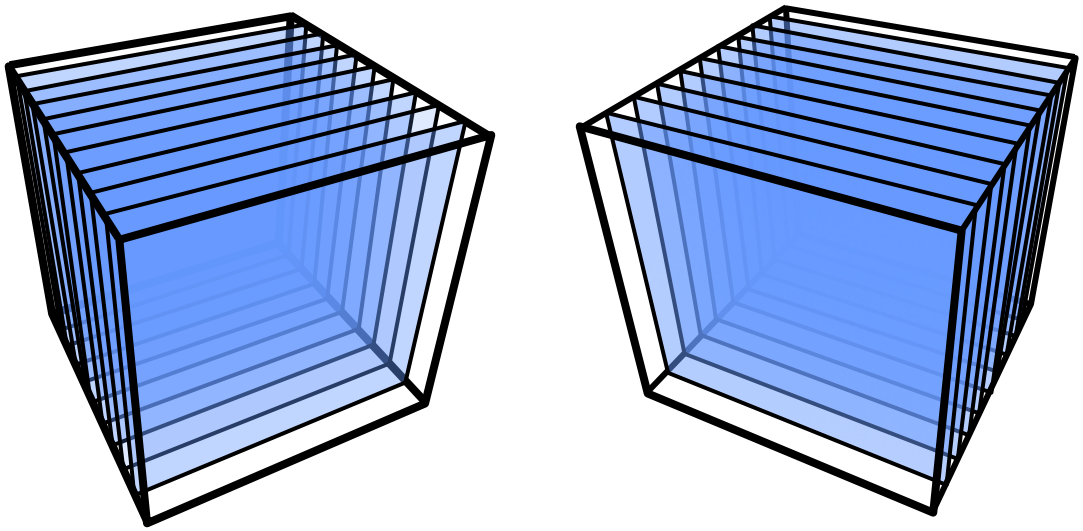
⇨ Decompostition into axis/object-aligned slices



2D Textures

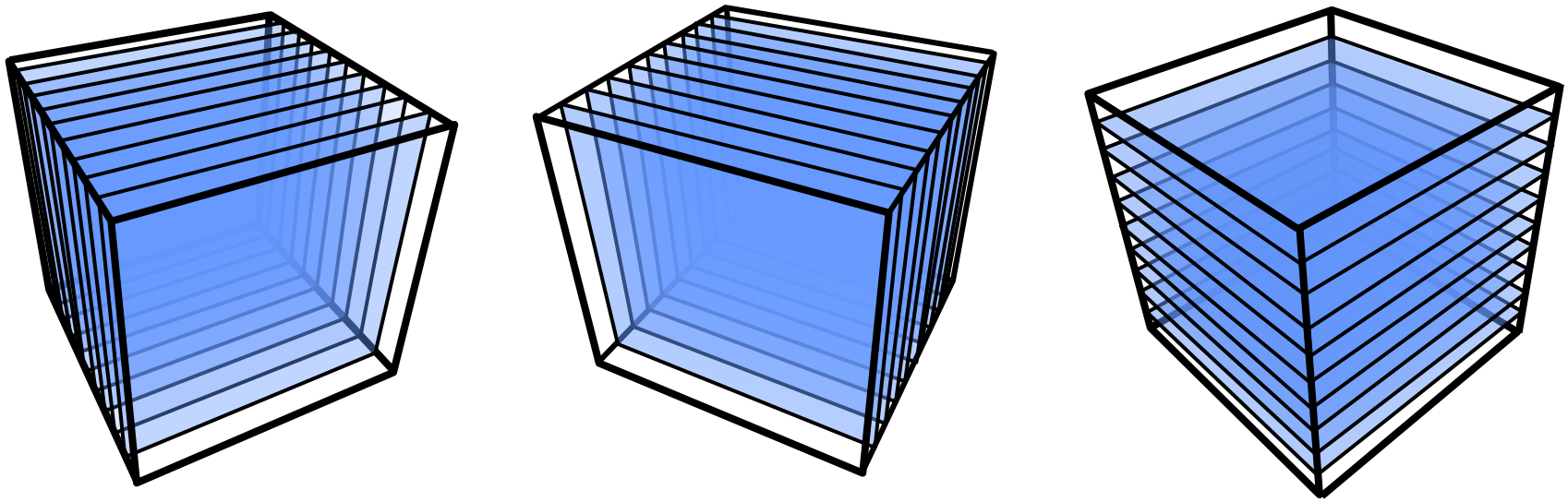
- Draw the volume as a stack of 2D textures
  Bilinear Interpolation in Hardware

⇨ Decompostition into axis/object-aligned slices



2D Textures

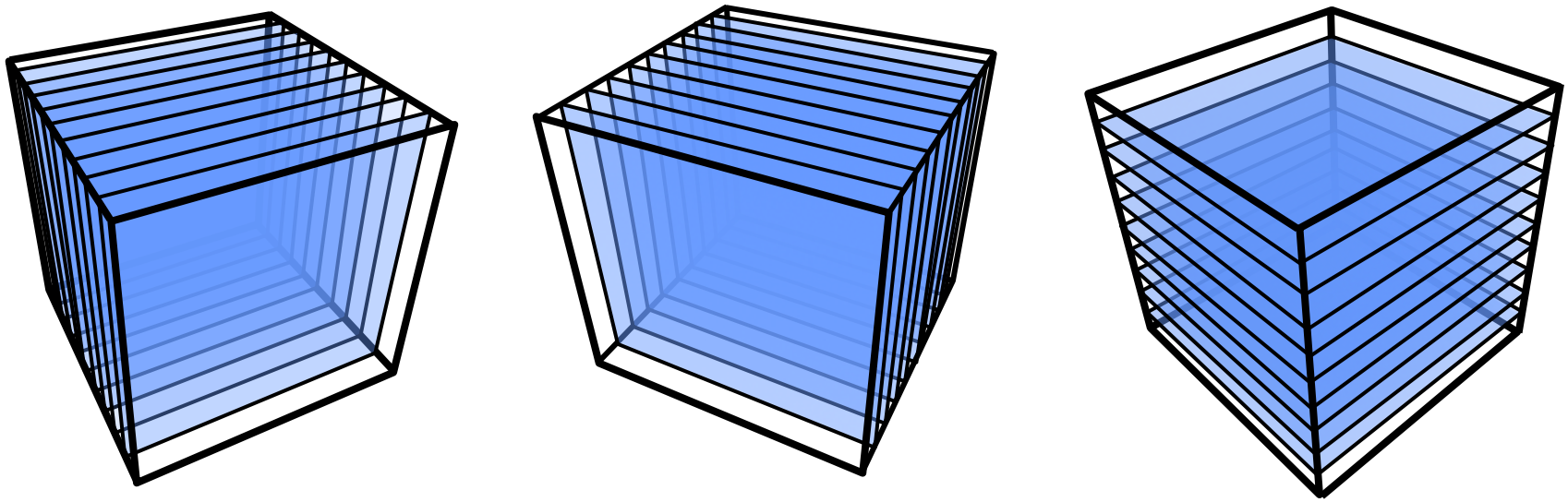
- Draw the volume as a stack of 2D textures
  Bilinear Interpolation in Hardware
  - ⇨ Decompostition into axis/object-aligned slices



2D Textures

- Draw the volume as a stack of 2D textures
  Bilinear Interpolation in Hardware

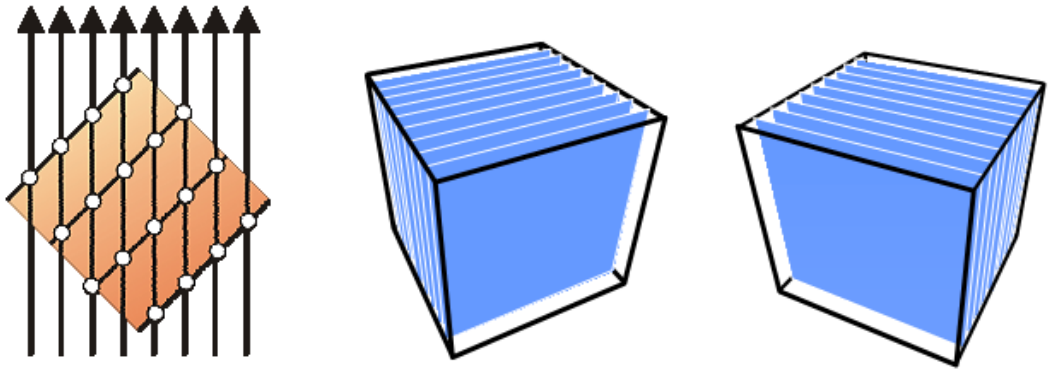
⇨ Decompostition into axis/object-aligned slices



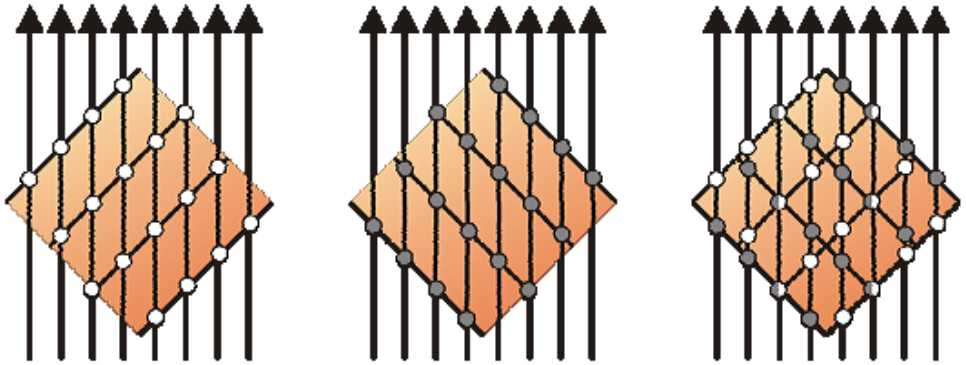
- 3 copies of the data set in memory

Proxy Geometry

- Object-Aligned Slices
  - Fast and simple
  - Three stacks of 2D textures – x, y, z principle directions

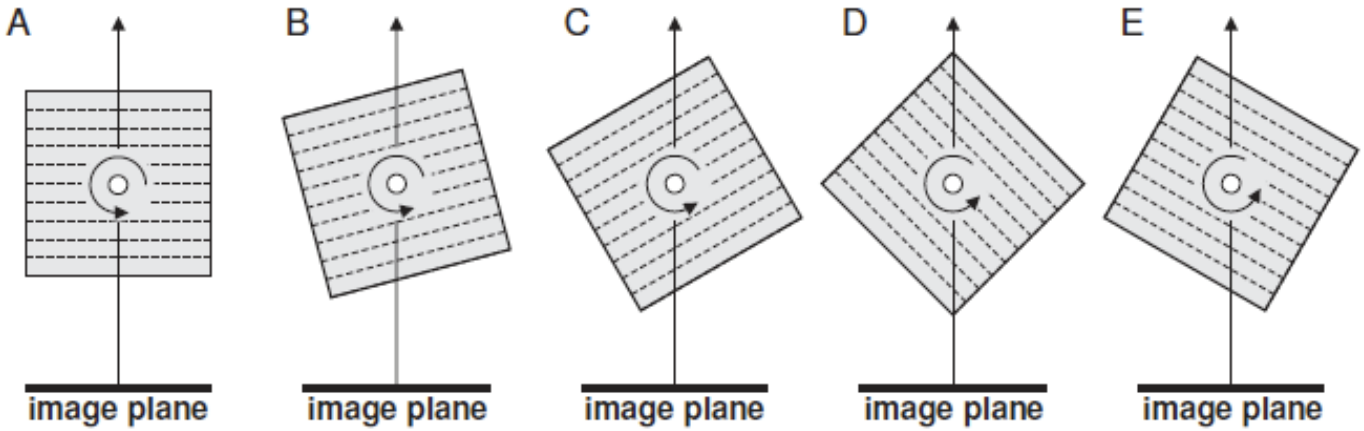


  - Texture stack swapped based on closest to viewpoint



Implementation

- Prepare the texture images for the 3 stacks of slices
- The slicing direction = minimizes the angle between the slice normal and the viewing direction
- Send to draw the slices/polygons with their correct bounded textures.
- Back-to-front composition:



2D Textures: Drawbacks. Flickering

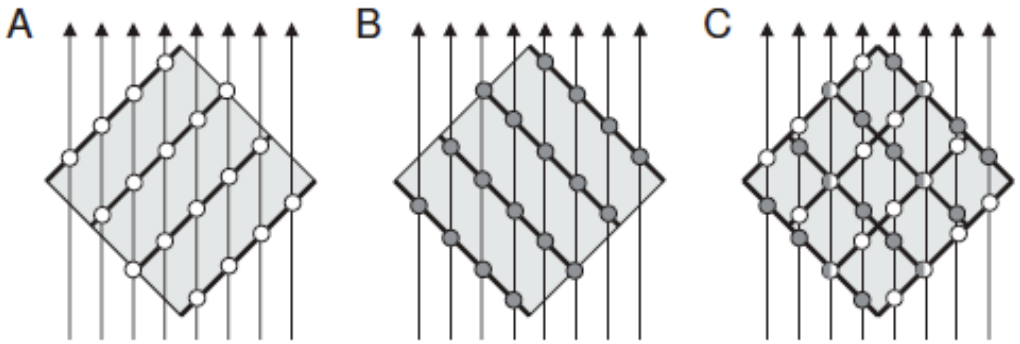


Figure 3.5: The location of sampling points changes abruptly (C), when switching from one slice stack (A), to the next (B).

Proxy Geometry

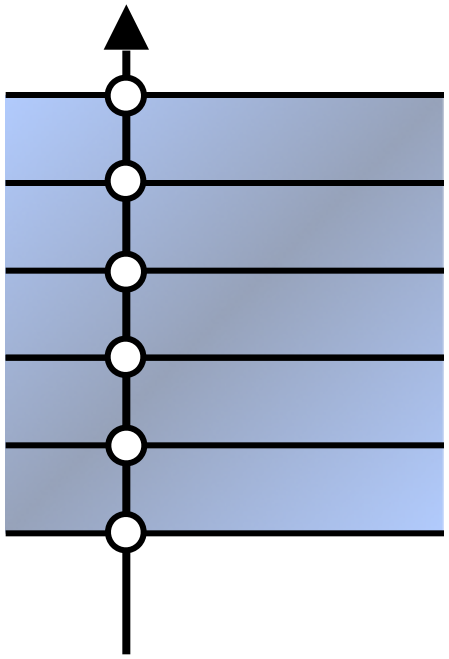
- Issues with Object-Aligned Slices
  - 3x memory consumption
    - Data replicated along 3 principle directions
  - Change in viewpoint results in stack swap
    - Image popping artifacts
    - Lag while downloading new textures
  - Sampling distance changes with viewpoint
    - Intensity variations as camera moves

2D Textures: Drawbacks. Sampling rate is inconsistent

- Sampling rate is inconsistent

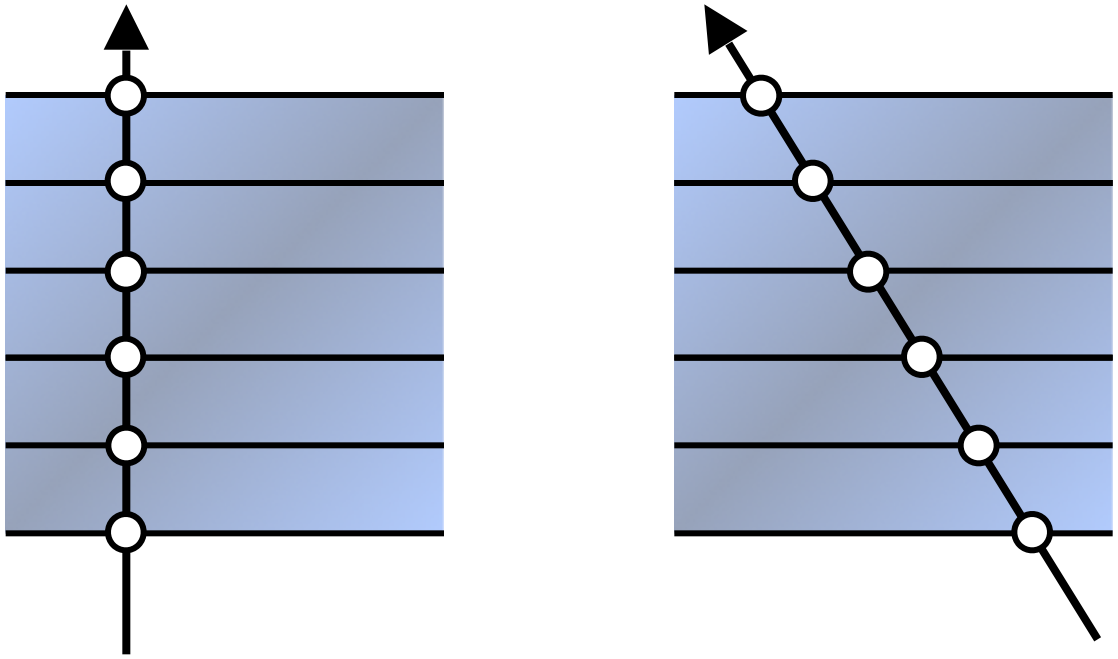
2D Textures: Drawbacks. Sampling rate is inconsistent

- Sampling rate is inconsistent



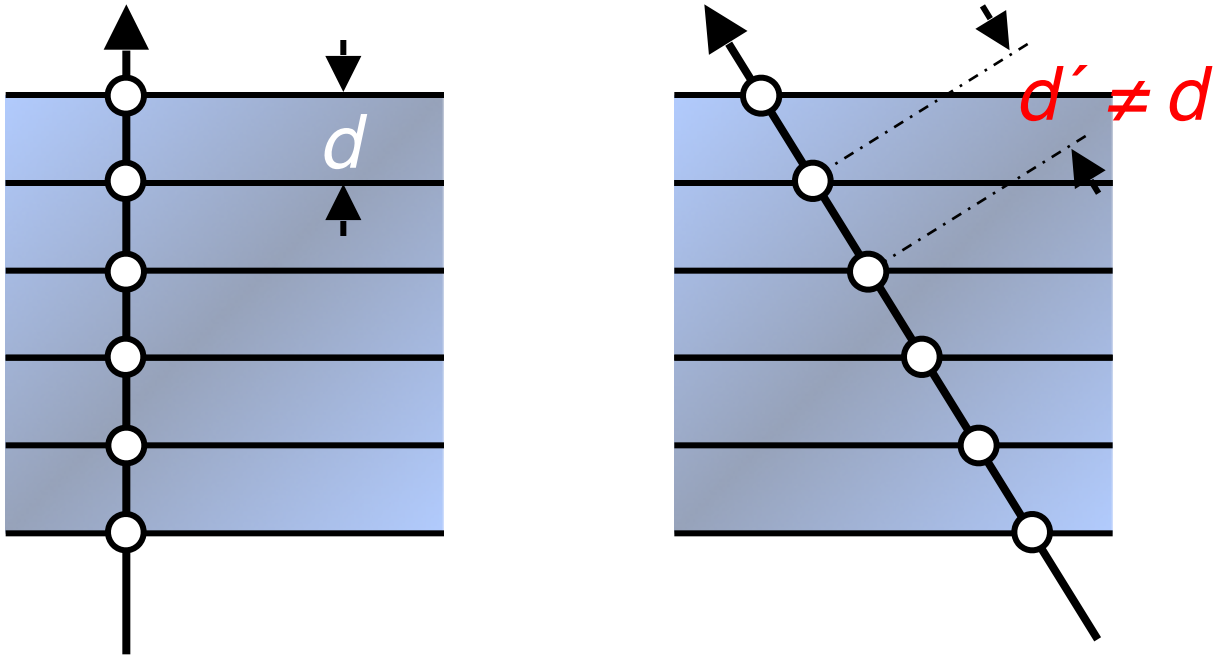
2D Textures: Drawbacks. Sampling rate is inconsistent

- Sampling rate is inconsistent



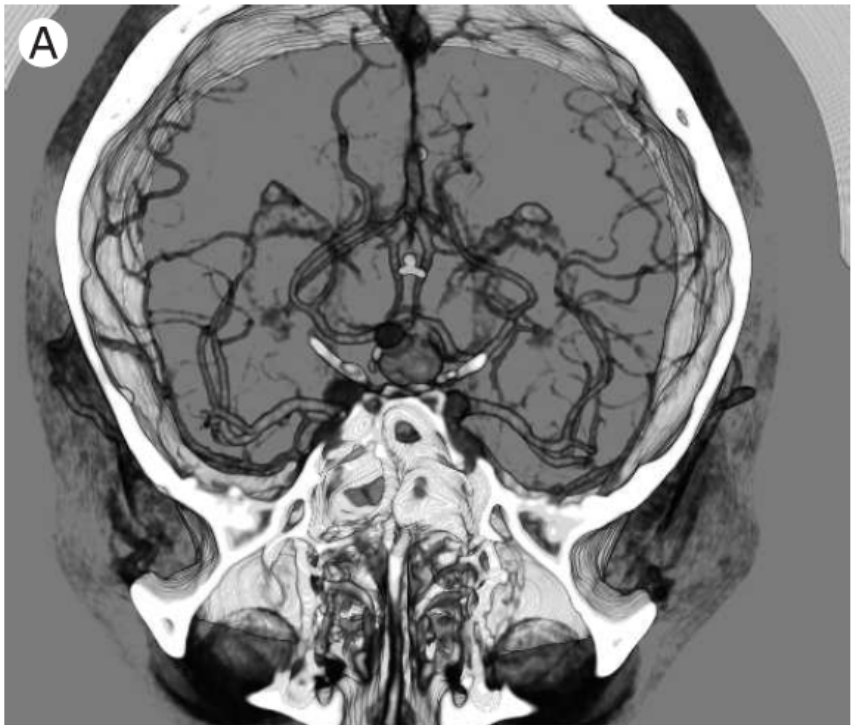
2D Textures: Drawbacks. Sampling rate is inconsistent

- Sampling rate is inconsistent

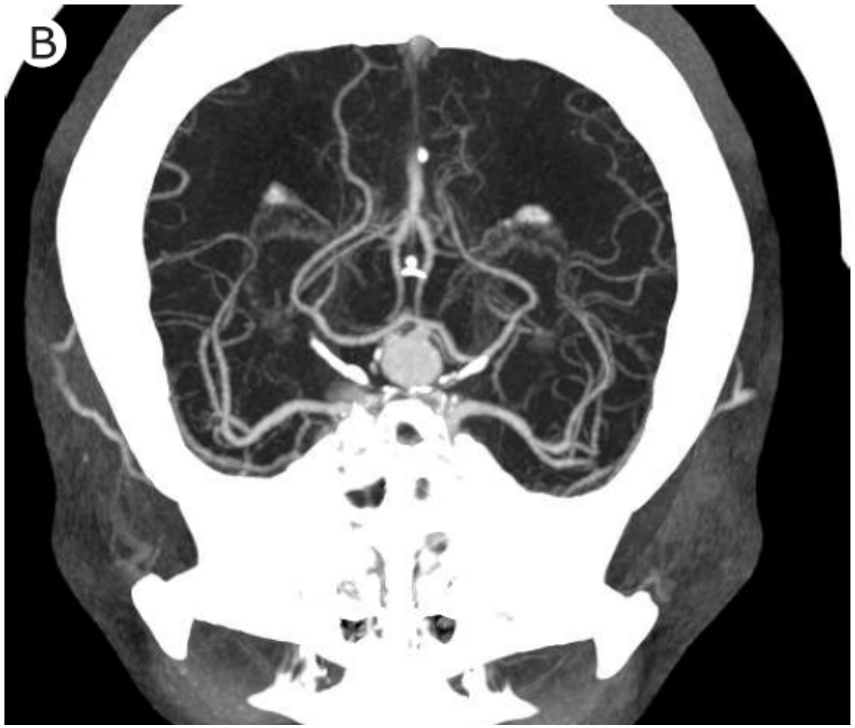


- Emission/absorption slightly incorrect
- ***Super-sampling on-the-fly impossible***

Compositing



Emission/Absorption



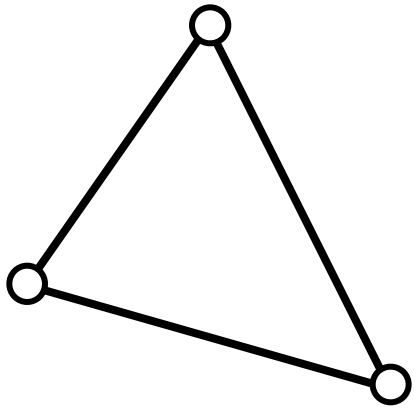
Maximum Intensity Projection

2D Textures: Drawbacks. Sampling rate is inconsistent

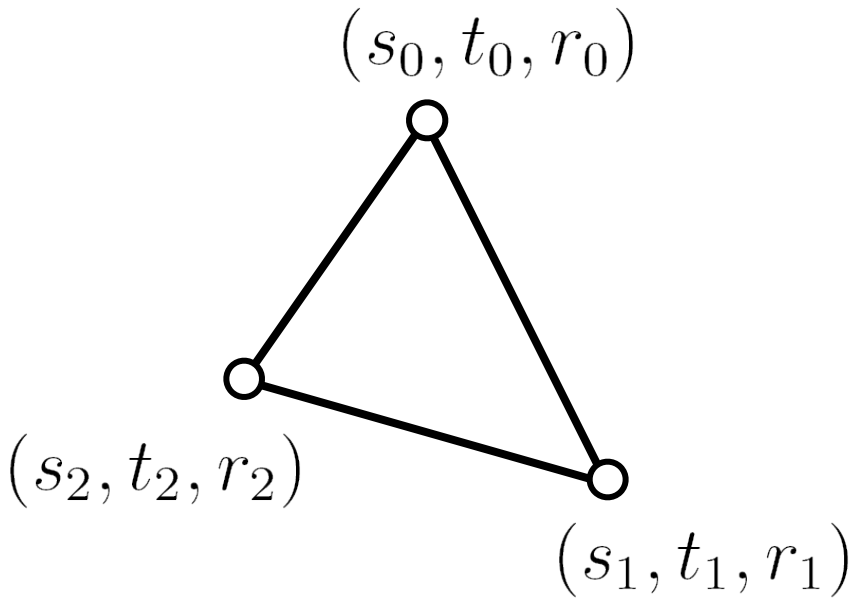
- Numerical Integration assumes constant length between samples.
- Distances between samples = distances between slices with respect to the view direction.
- Distance between slices is fixed, and color and opacity are computed for each texture assuming these fixed distances.

3D Textures

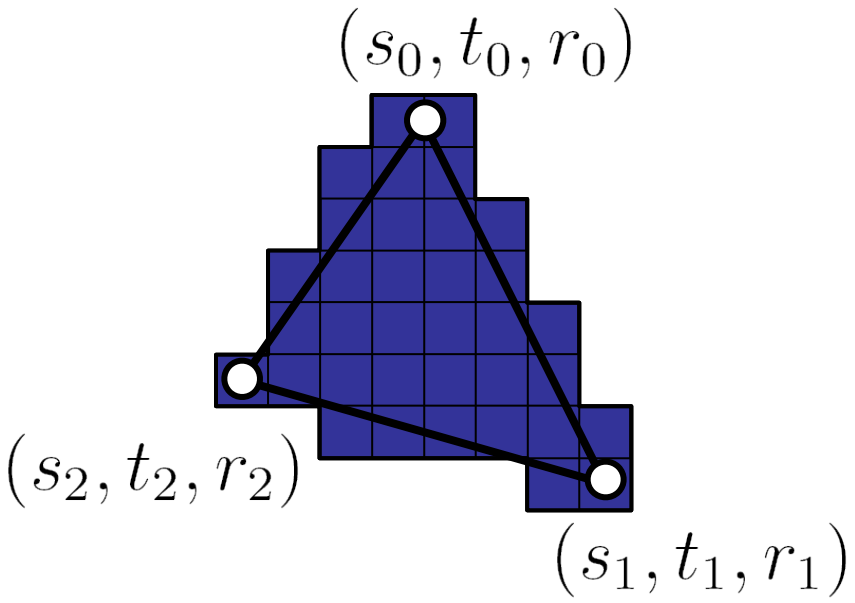
3D Textures



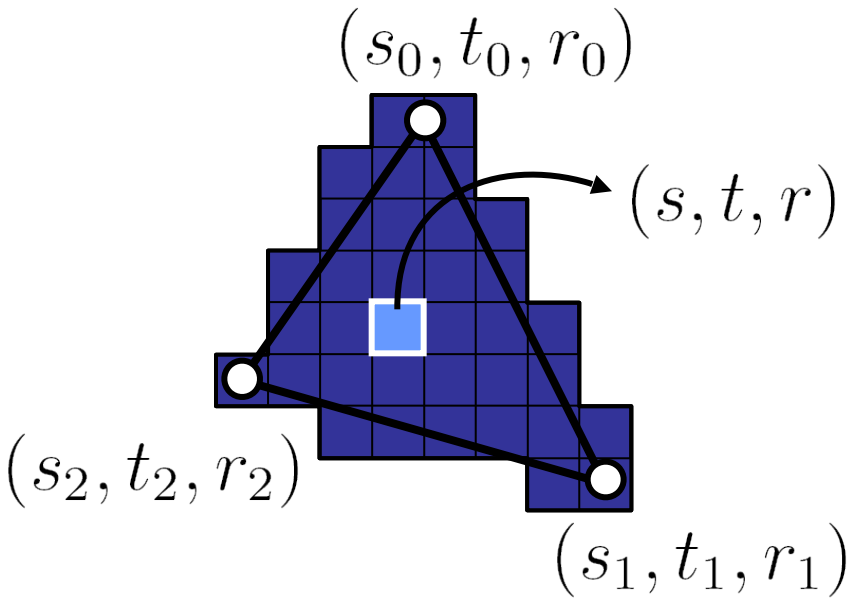
3D Textures

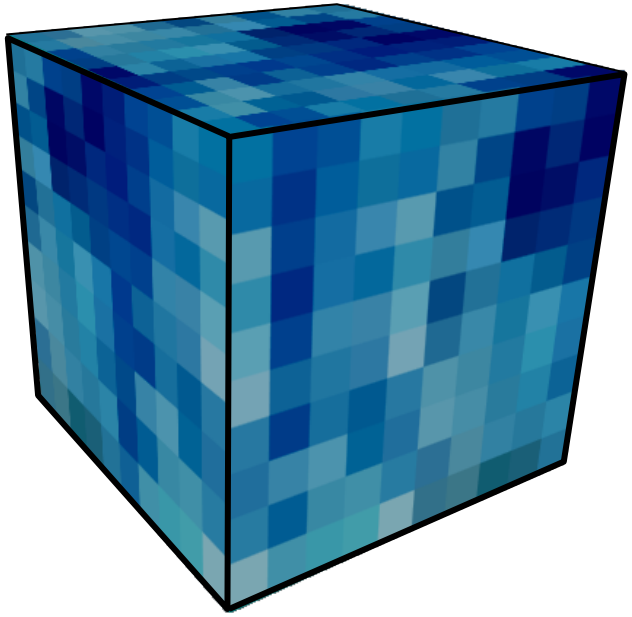


3D Textures

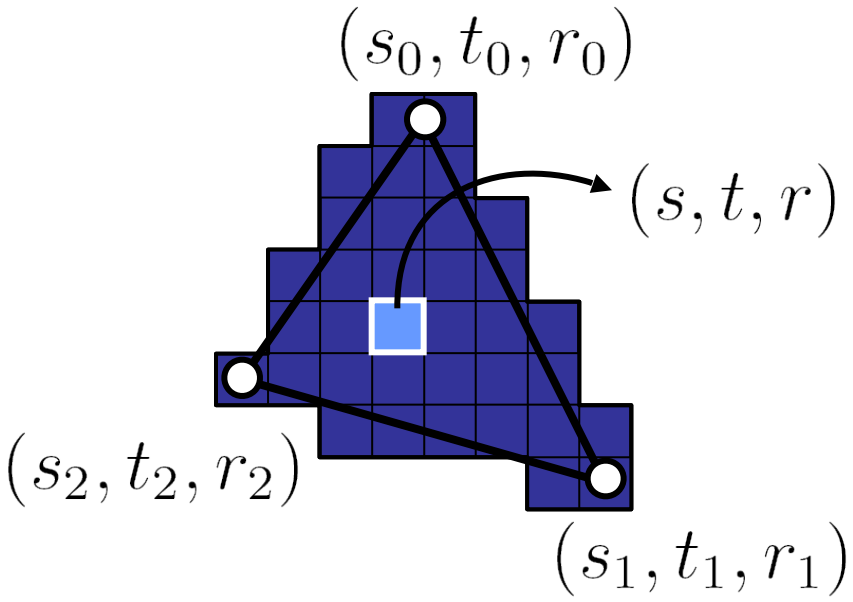


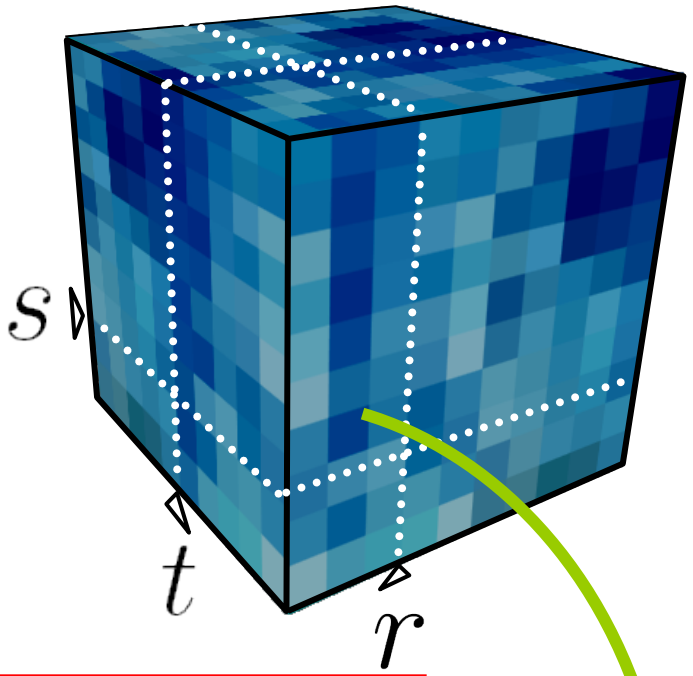
3D Textures





3D Textures





Don't be confused: *3D textures are not volumetric rendering primitives!*

Only ***planar polygons*** *are supported as rendering primitives.*

R G B A

3D Textures

3D Texture: Volumetric Texture Object

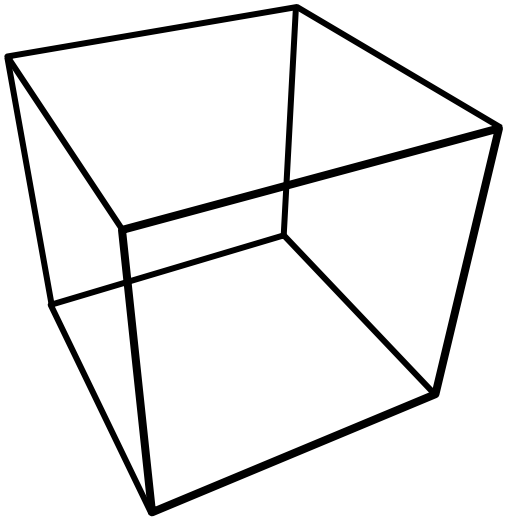
- Trilinear Interpolation in Hardware

3D Textures

3D Texture: Volumetric Texture Object

- Trilinear Interpolation in Hardware

⇨ Slices parallel to the image plane

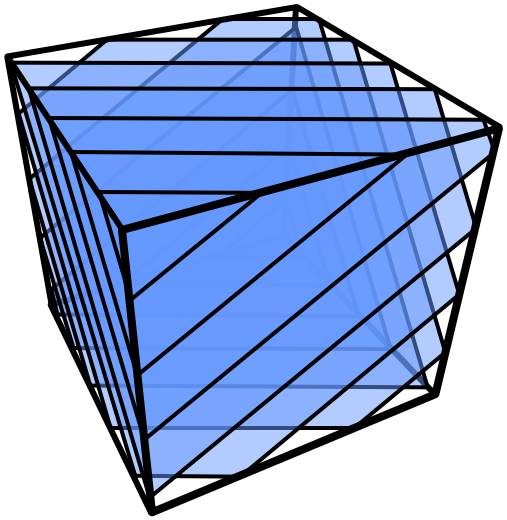


3D Textures

3D Texture: Volumetric Texture Object

- Trilinear Interpolation in Hardware

⇨ Slices parallel to the image plane

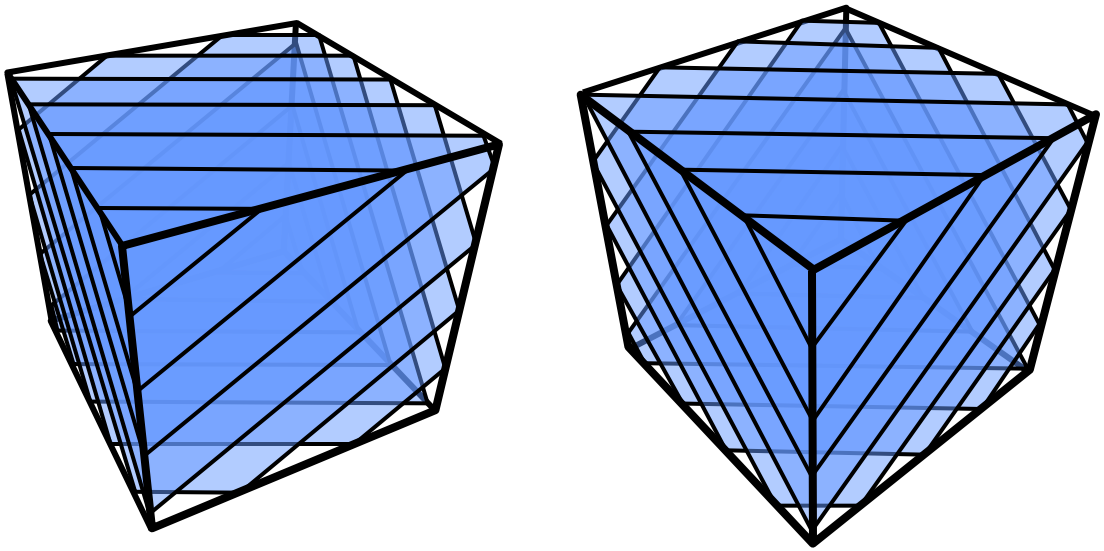


3D Textures

3D Texture: Volumetric Texture Object

- Trilinear Interpolation in Hardware

⇨ Slices parallel to the image plane

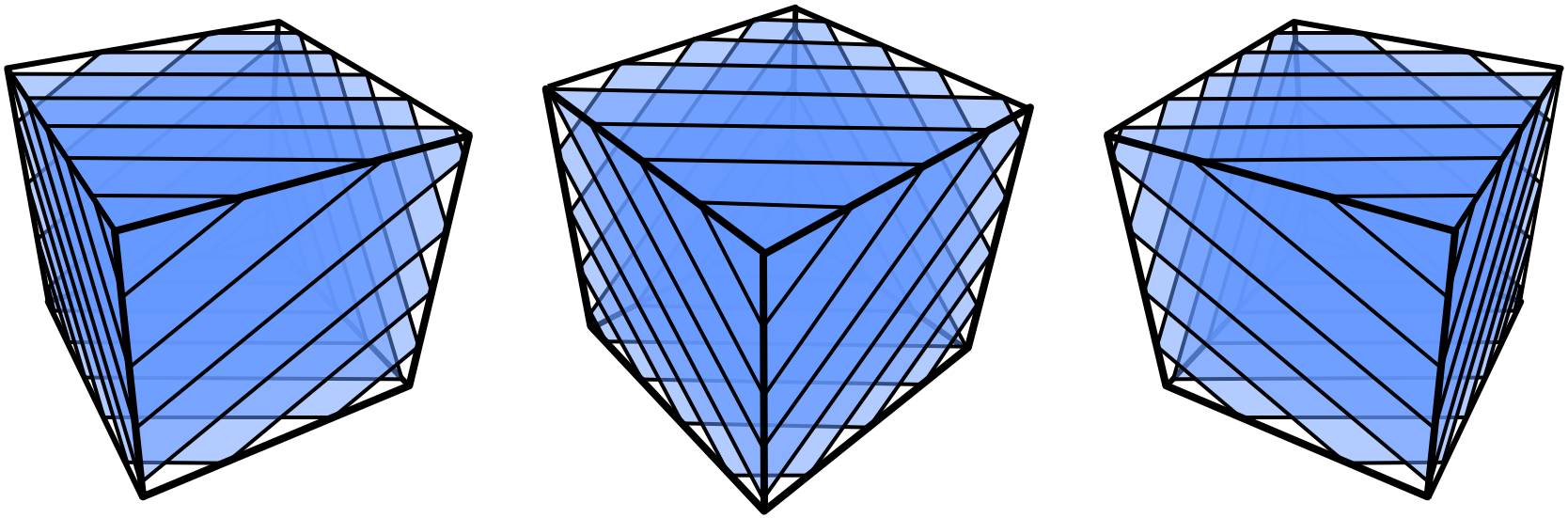


3D Textures

3D Texture: Volumetric Texture Object

- Trilinear Interpolation in Hardware

⇨ Slices parallel to the image plane

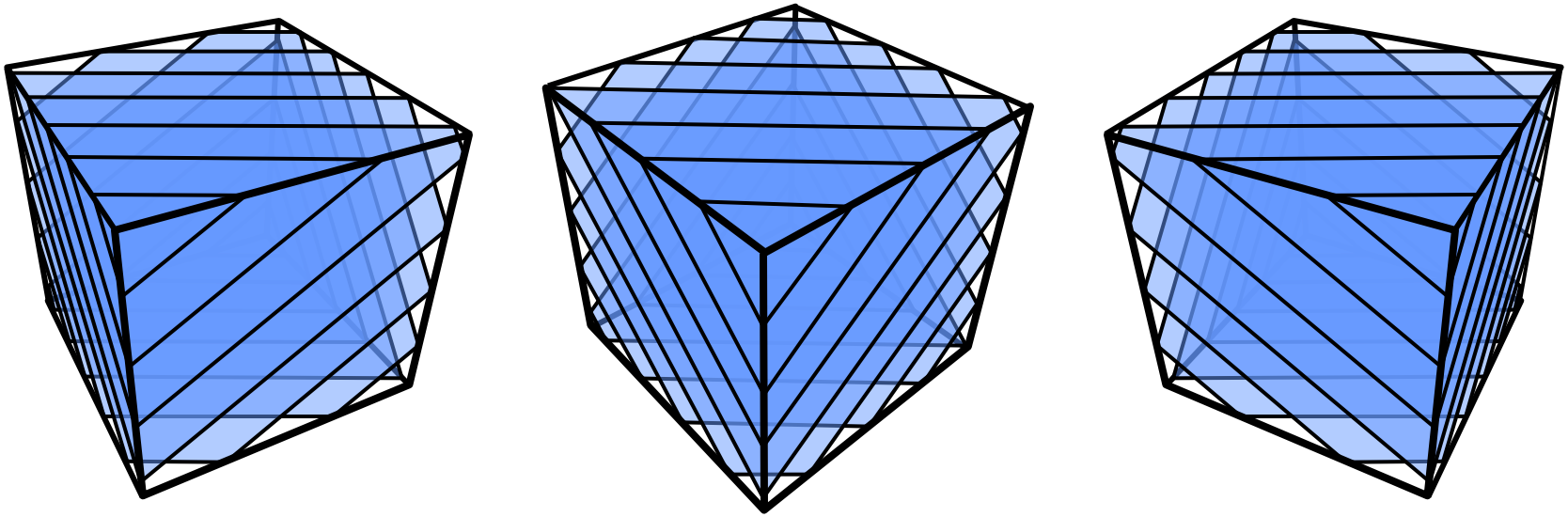


3D Textures

3D Texture: Volumetric Texture Object

- Trilinear Interpolation in Hardware

⇨ Slices parallel to the image plane



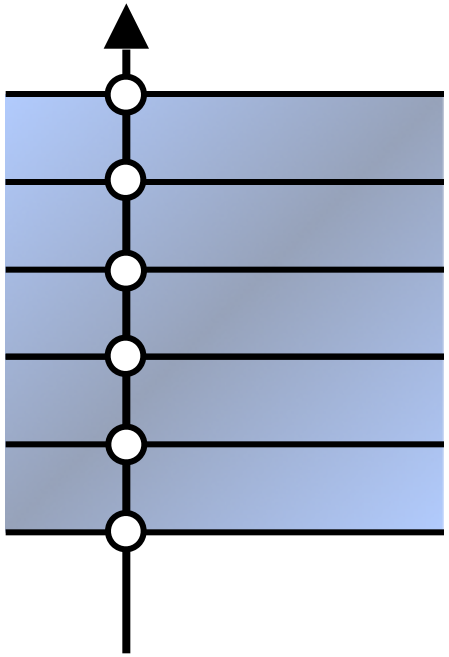
- One large texture block in memory

Resampling via 3D Textures

- ***Sampling rate is constant***

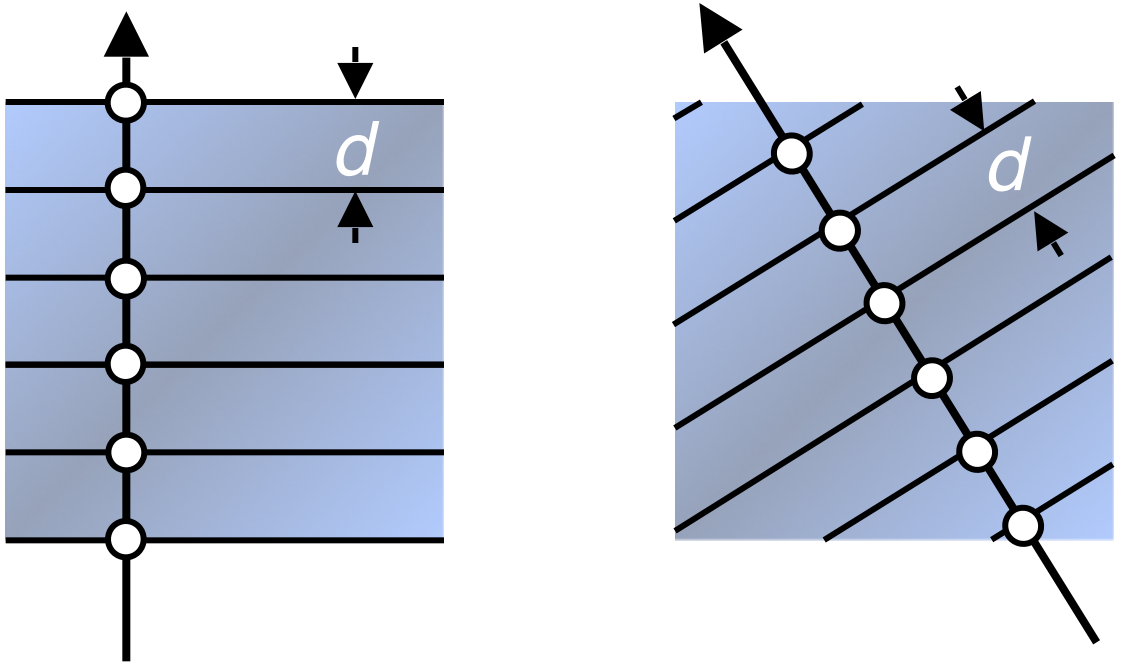
Resampling via 3D Textures

- ***Sampling rate is constant***



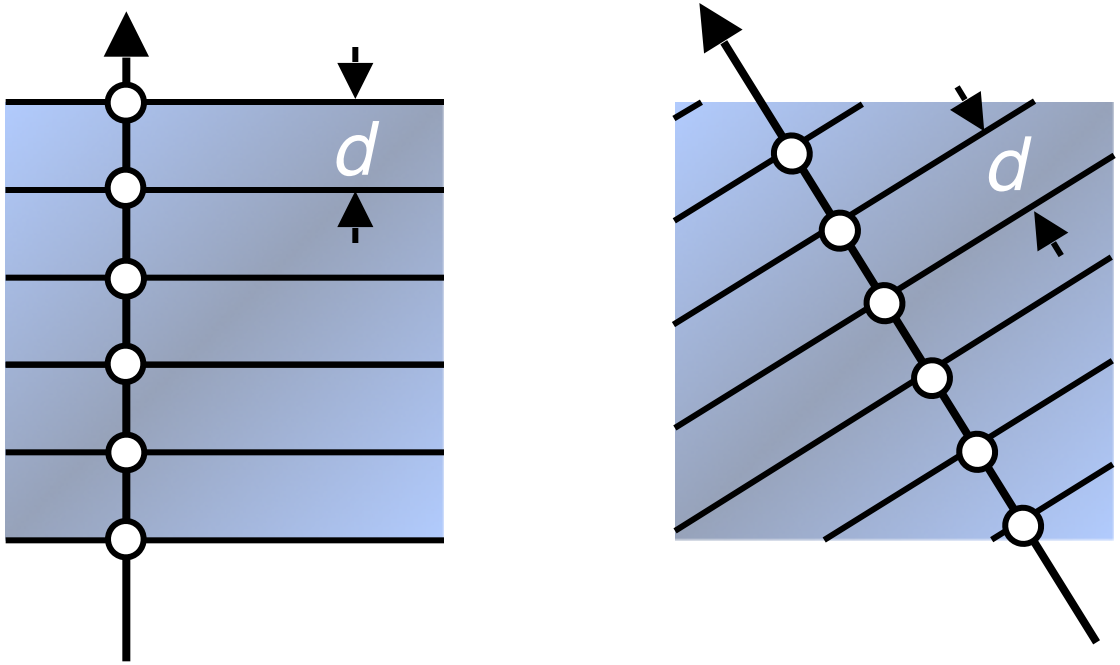
Resampling via 3D Textures

- ***Sampling rate is constant***



Resampling via 3D Textures

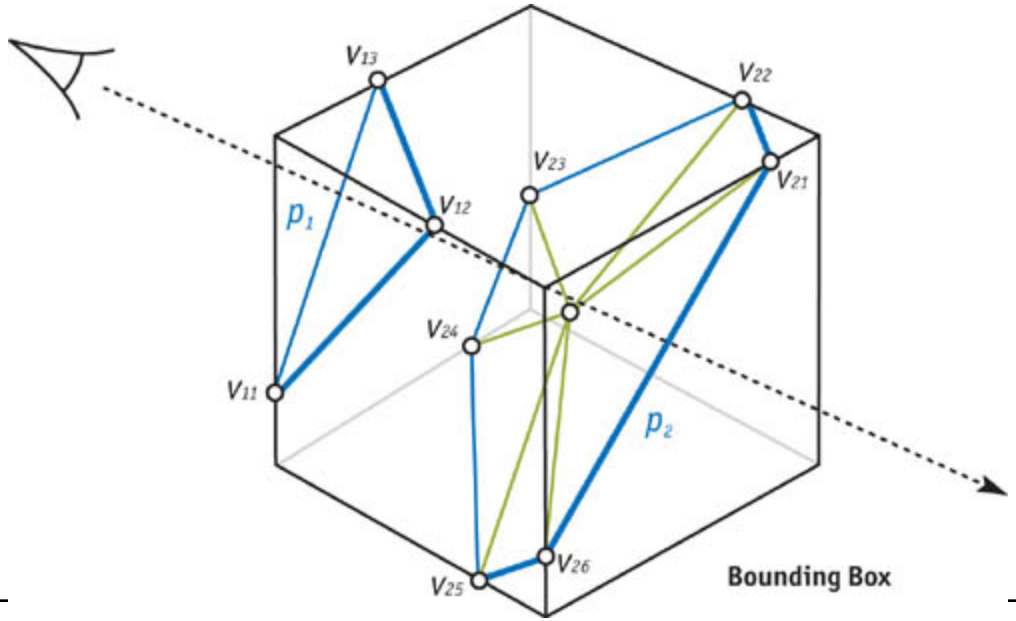
- ***Sampling rate is constant***



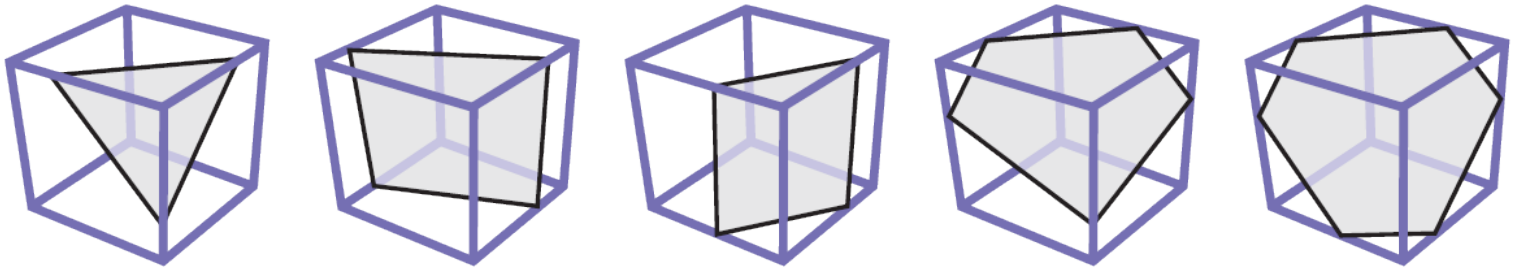
- Supersampling by increasing the number of slices

Proxy Geometry

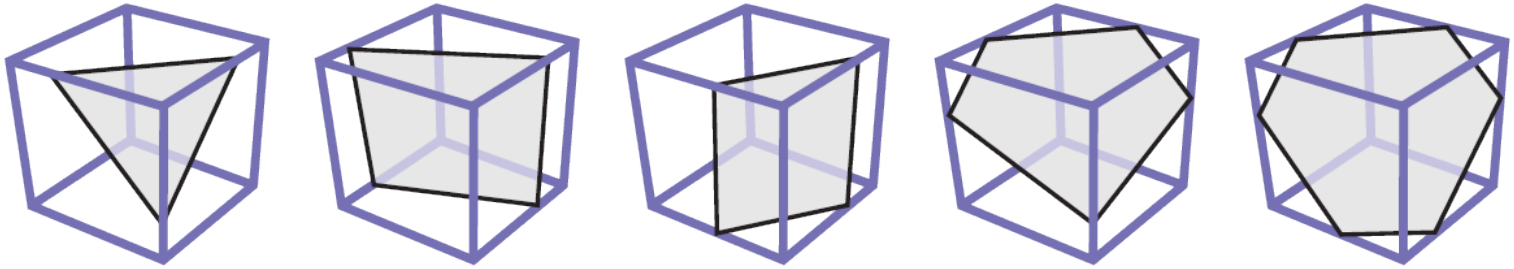
- View-Aligned Slices Algorithm
  - Intersect slicing planes with bounding box
  - Sort resulting vertices in (counter)clockwise order
  - Construct polygon primitive from centroid as triangle fan



Cube-Slice Intersection

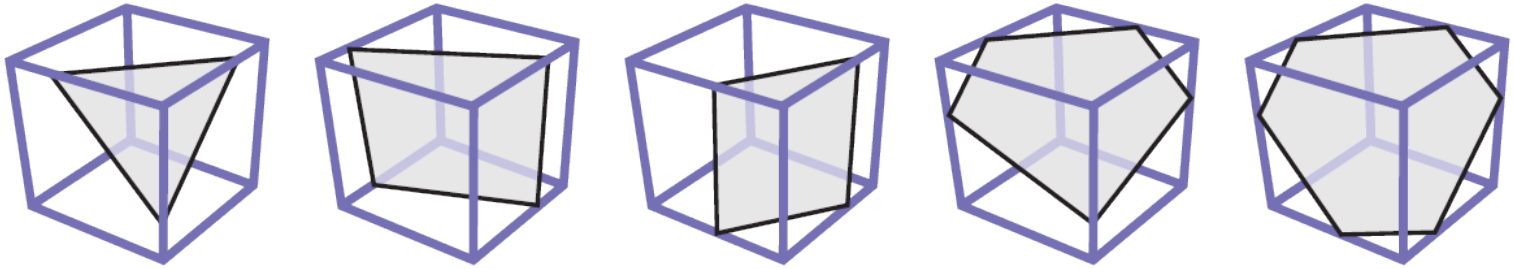


Cube-Slice Intersection



We can compute it in a vertex program

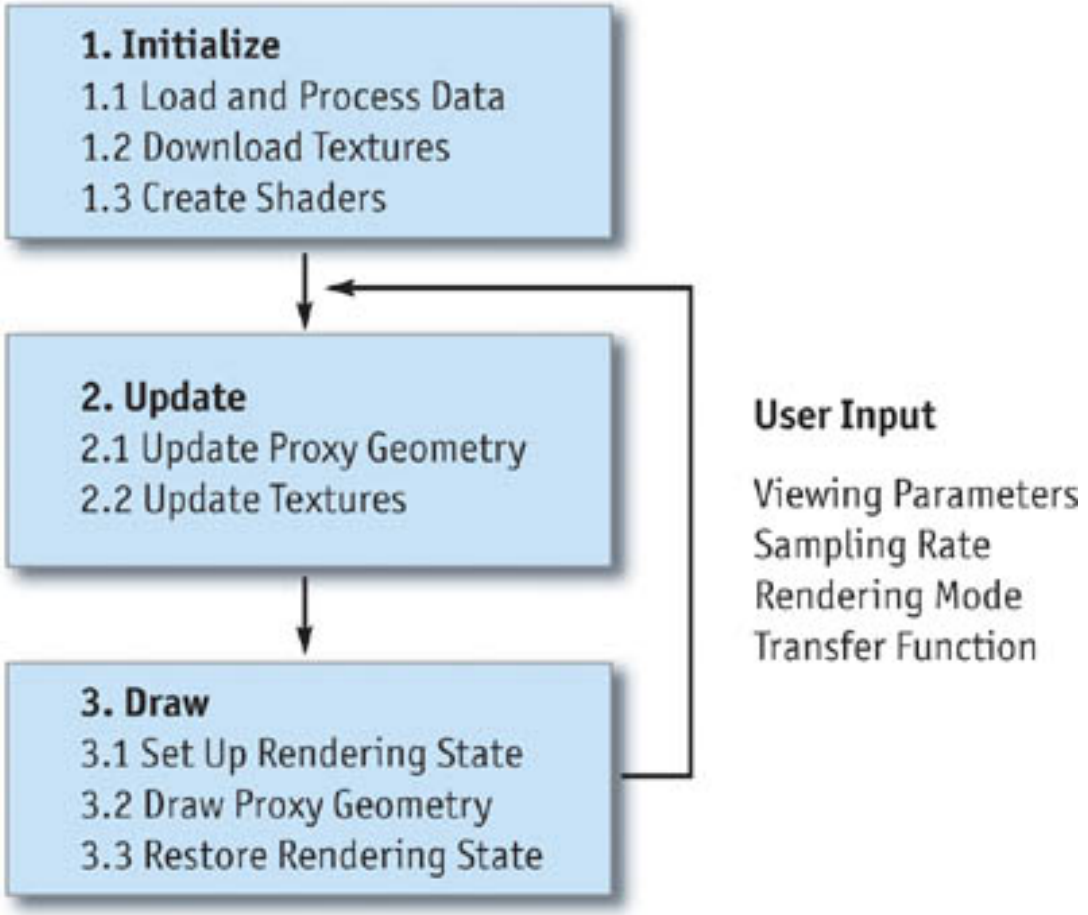
Cube-Slice Intersection



We can compute it in a vertex program

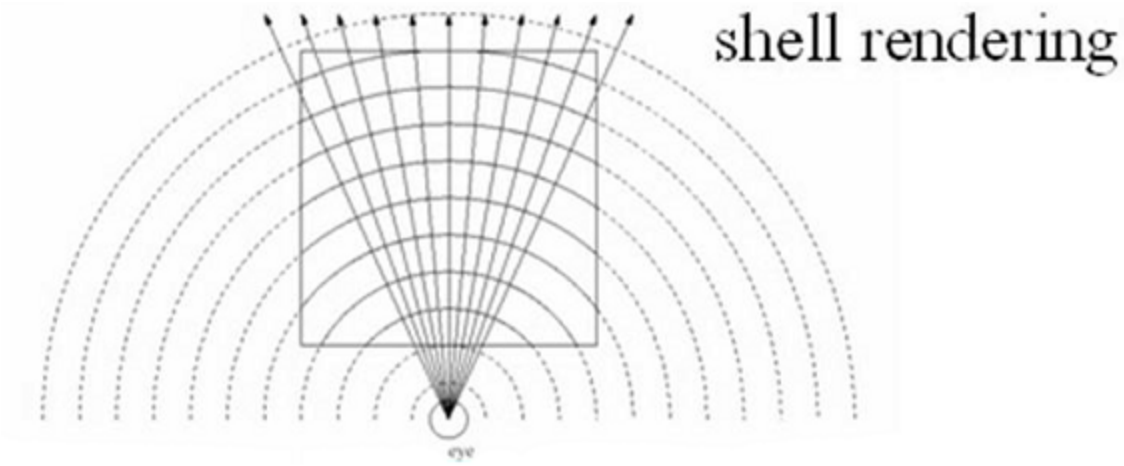
We can rotate the texture and having a fix set of object oriented planes.

Sliced-Based Volume Rendering Steps



Proxy Geometry

- Spherical Shells
  - Best replicates volume ray casting
  - Impractical – complex proxy geometry



Rendering Proxy Geometry

- Compositing
  - Over operator – back-to-front order

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

$$\hat{A}_i = A_i + (1 - A_i)\hat{A}_{i+1}$$

  - Under operator – front-to-back order

$$\hat{C}_i = (1 - \hat{A}_{i-1})C_i + \hat{C}_{i-1}$$

$$\hat{A}_i = (1 - \hat{A}_{i-1})A_i + \hat{A}_{i-1}$$

Rendering Proxy Geometry

- Compositing = Color and Alpha Accumulation Equations
- Easily implemented using hardware alpha blending
  - Over
    - Source = 1
    - Destination = 1 - Source Alpha
  - Under
    - Source = 1 - Destination Alpha
    - Destination = 1

Simple Volume Rendering Fragment Shader

```
void main(   uniform float3 emissiveColor,
             uniform sampler3D dataTex,
             float3 texCoord : TEXCOORD0,
             float4 color : COLOR)
{
   float a = tex3D(texCoord, dataTex); // Read 3D data
   texture color = a * emissiveColor; // Multiply by
   opac
}
```

Fragment Shader with Transfer Function

```
void main(   uniform sampler3D dataTex,
             uniform sampler1D tfTex,
             float3 texCoord : TEXCOORD0,
             float4 color : COLOR
         )
{
   float v = tex3d(texCoord, dataTex); // Read 3D data
   color = tex1d(v, tfTex); // transfer function
}
```

Local Illumination

- Blinn-Phong Shading Model

$$I = k_a + I_L k_d (\hat{l} \cdot \hat{n}) + I_L k_s (\hat{h} \cdot \hat{n})^N$$

Resulting = Ambient + Diffuse + Specular

Local Illumination

- Blinn-Phong Shading Model

$$I = k_a + I_L k_d (\hat{l} \cdot \hat{n}) + I_L k_s (\hat{h} \cdot \hat{n})^N$$

Resulting = Ambient + Diffuse + Specular

- Requires surface normal vector
  - Whats the normal vector of a voxel?

Local Illumination

- Blinn-Phong Shading Model

$$I = k_a + I_L k_d (\hat{l} \cdot \hat{n}) + I_L k_s (\hat{h} \cdot \hat{n})^N$$

Resulting = Ambient + Diffuse + Specular

- Requires surface normal vector
  - Whats the normal vector of a voxel? **Gradient**
  - Central differences between neighboring voxels

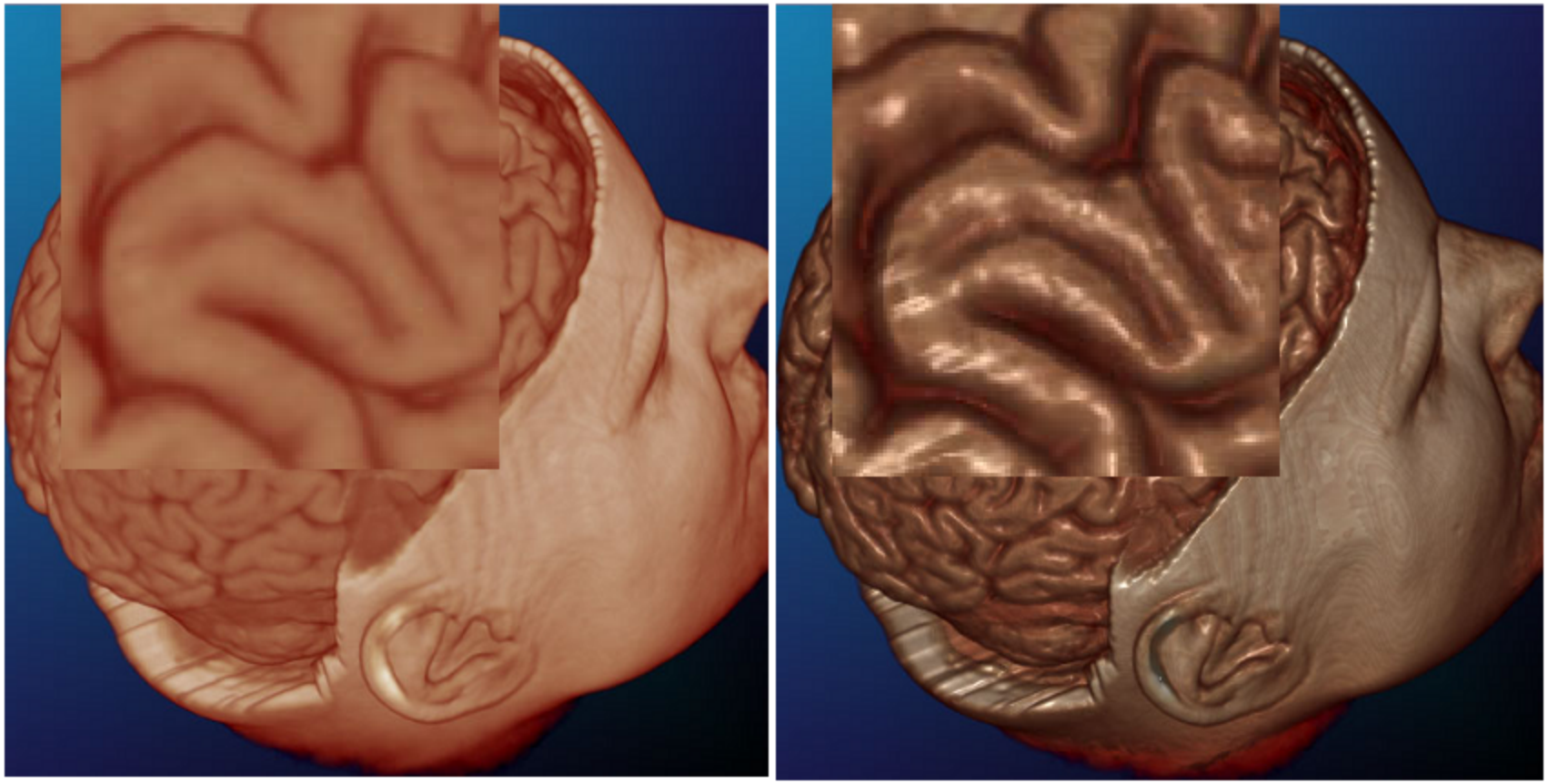
$$grad(I) = \nabla I = \frac{(right - left)}{2x}, \frac{(top - bottom)}{2x}, \frac{(front - back)}{2x}$$

Local Illumination

- Compute on-the-fly within fragment shader
  - Requires 6 texture fetches per calculation
- Precalculate on host and store in voxel data
  - Requires 4x texture memory
  - Pack into 3D RGBA texture to send to GPU

Local Illumination

- Improve perception of depth
- Amplify surface structure

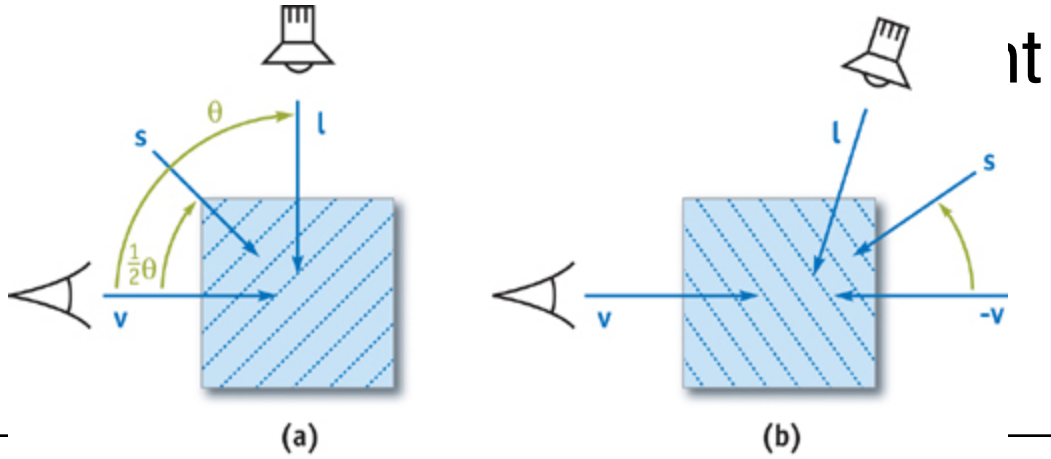


Volumetric Shadows on GPU

- Light attenuated from light's point of view
- CPU – Precomputed Light Transfer
  - Secondary raymarch from sample to light source
- GPU
  - Two-pass algorithm
  - Modify proxy geometry slicing
  - Render from both the eye and the light's POV
    - Two different frame buffers

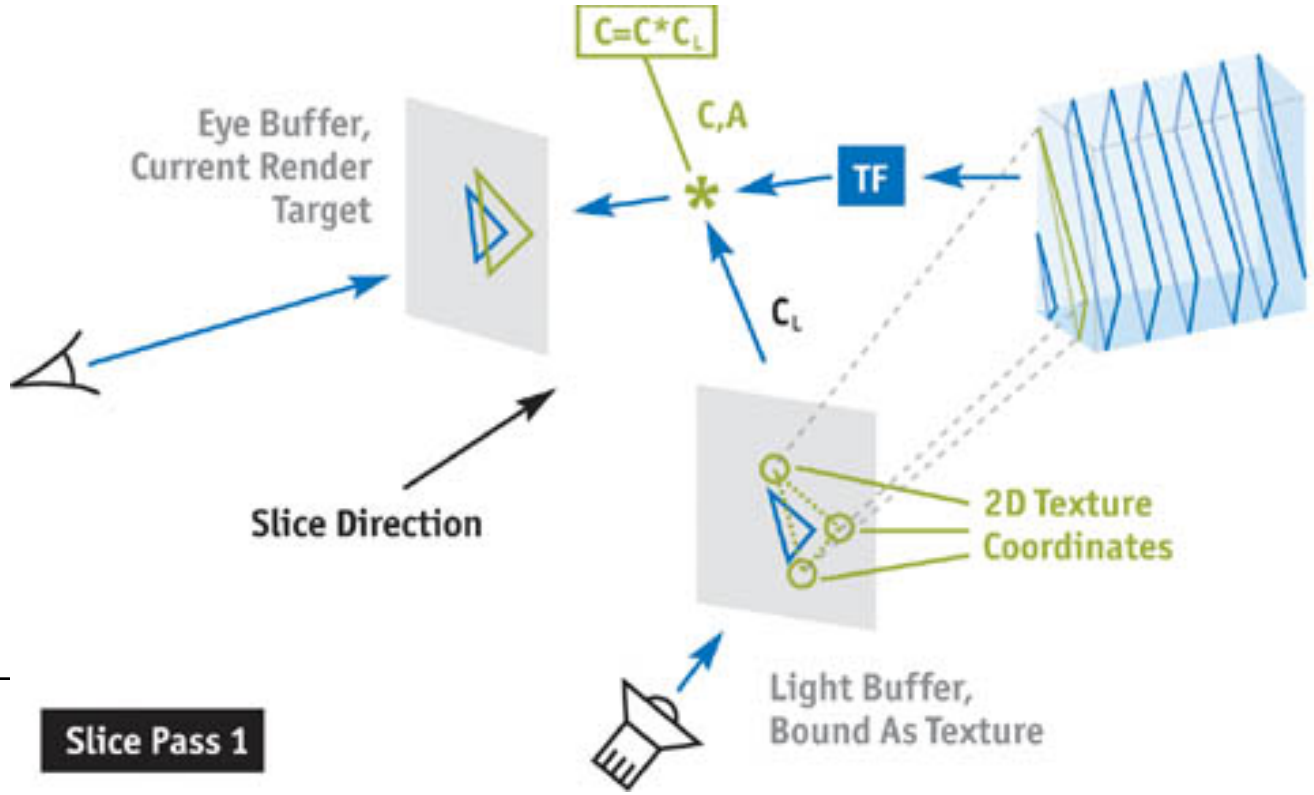
Two Pass Volume Rendering with Shadows

- Slice axis set half-way between view and light directions
  - Allows each slice to be rendered from eye and light POV
- Render order for light – front-to-back
- Render order for eye – (a) front-to-back



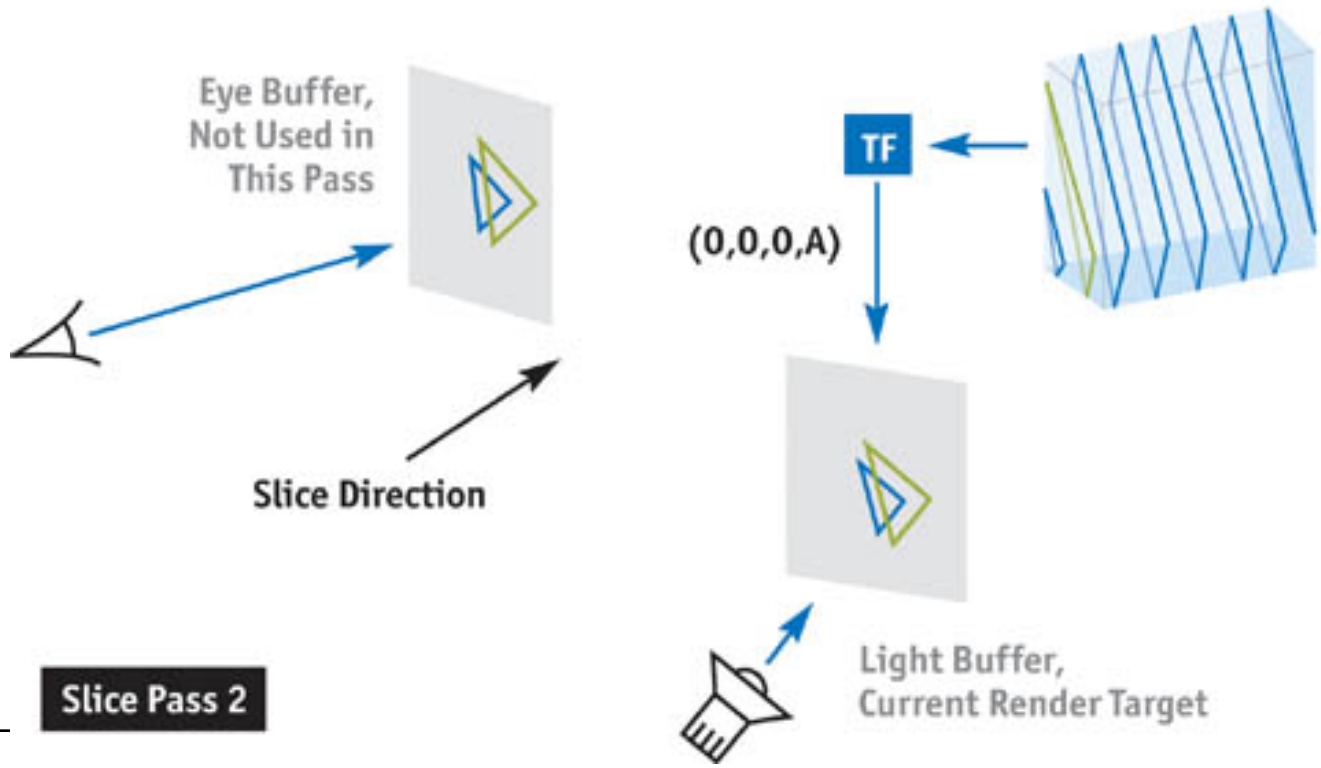
First Pass

- Render from eye
- Fragment shader
  - Look up light color from light buffer bound as texture
  - Multiply material color * light color

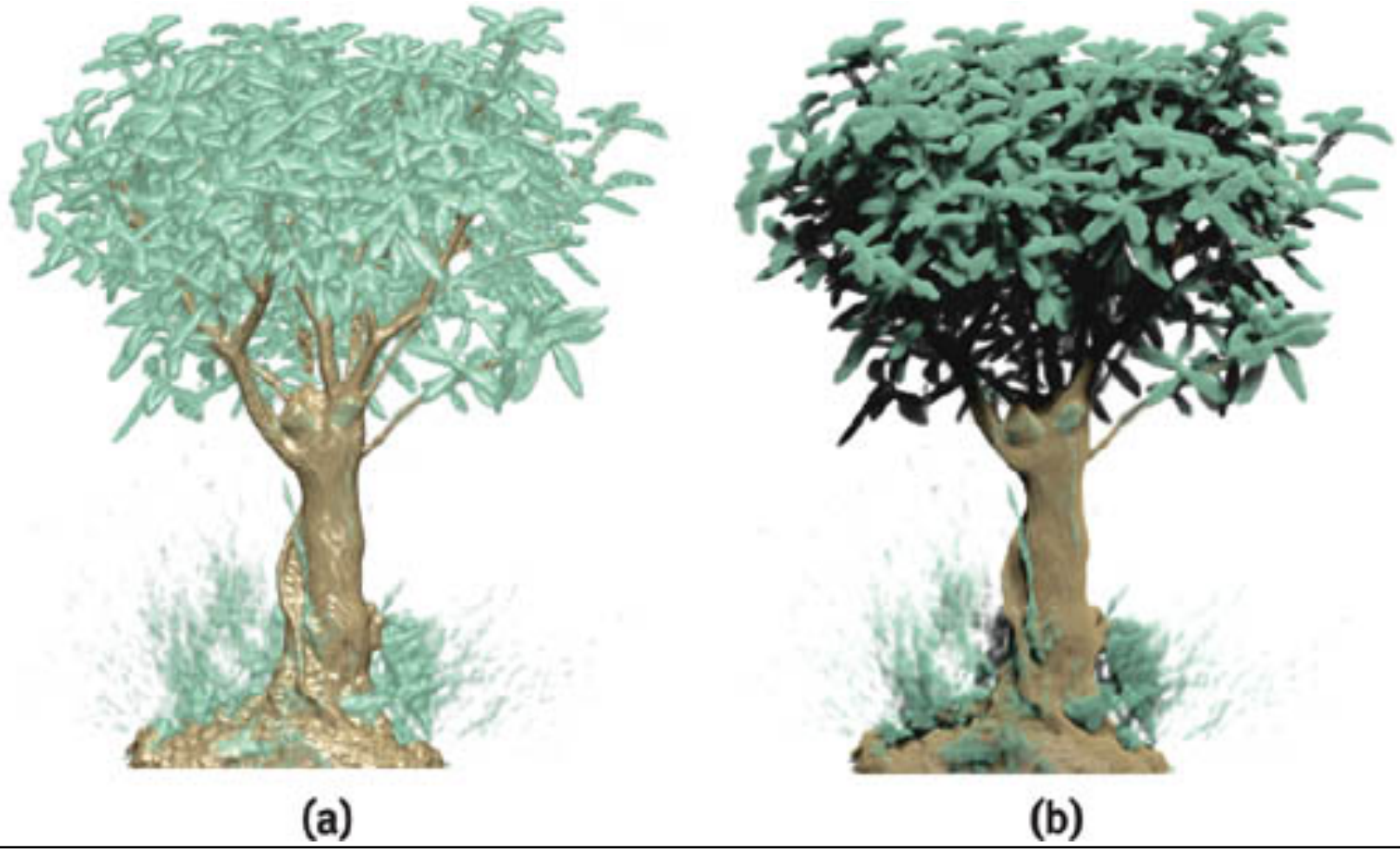


Second pass

- Render from light
- Fragment shader
  - Only blend alpha values – light transmissivity

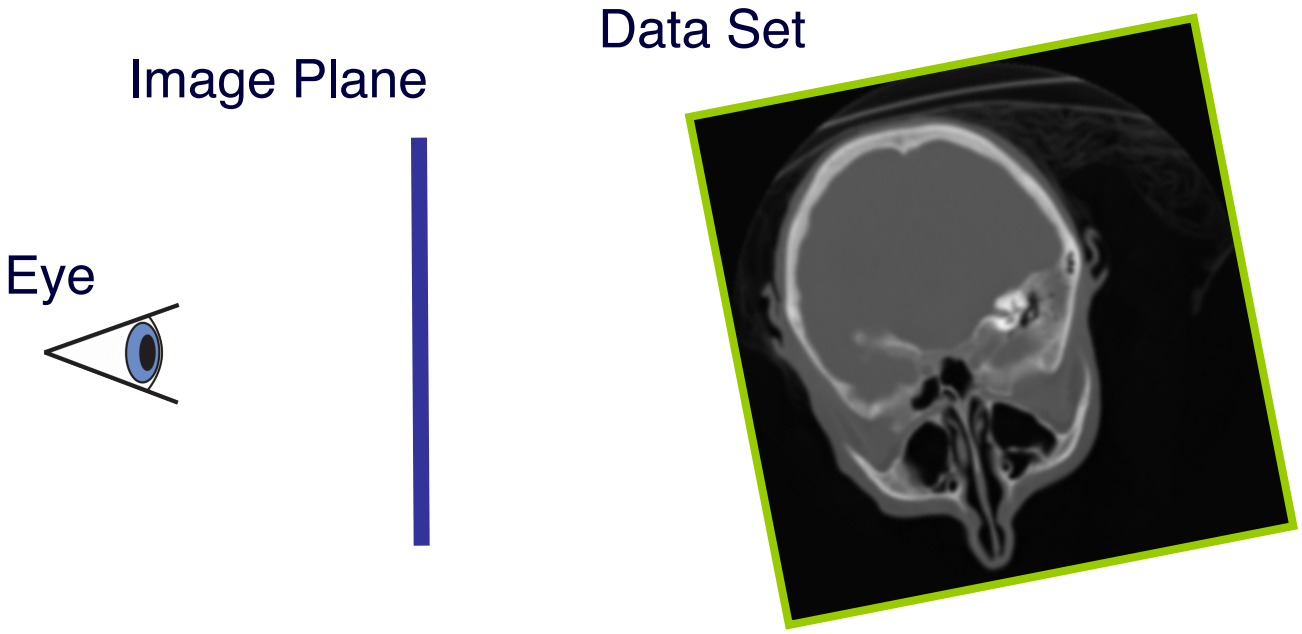


Volumetric Shadows

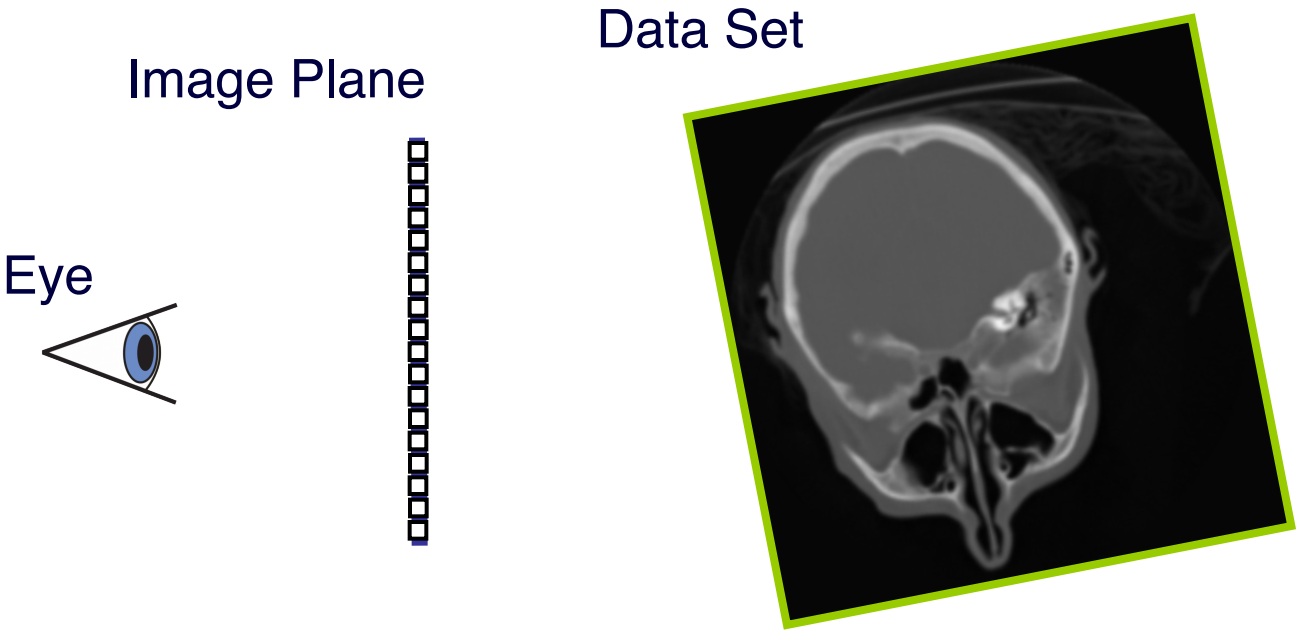


(a) (b)

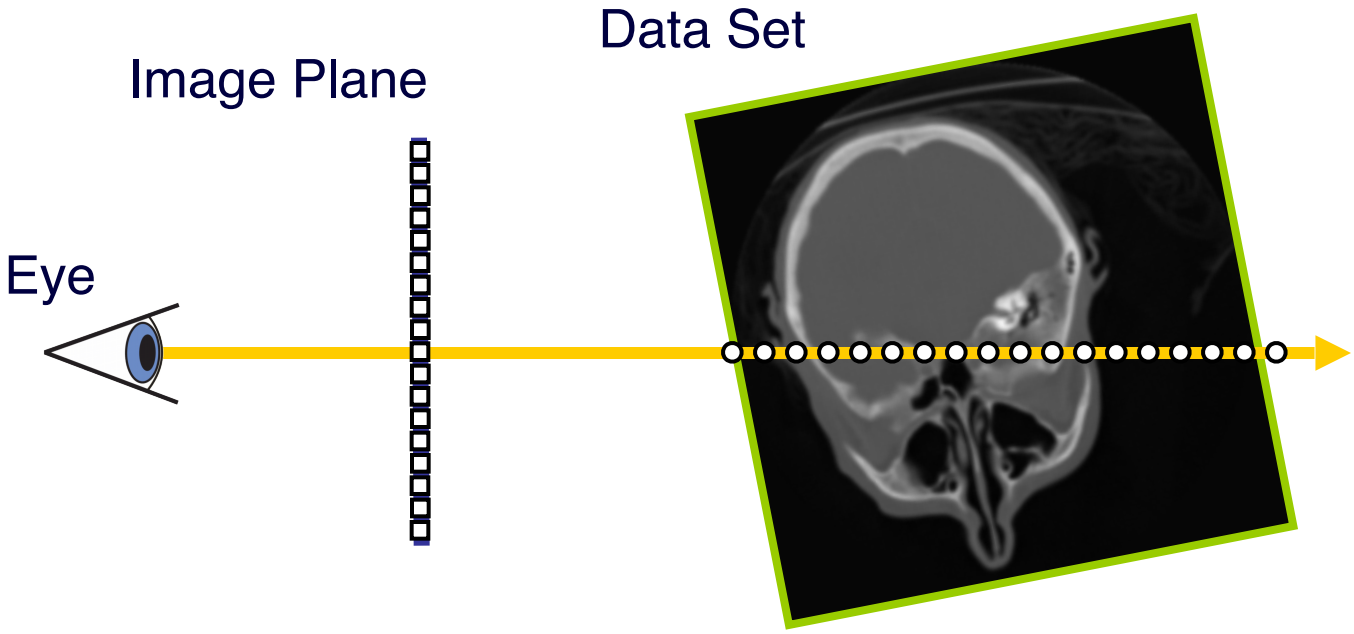
Ray-casting



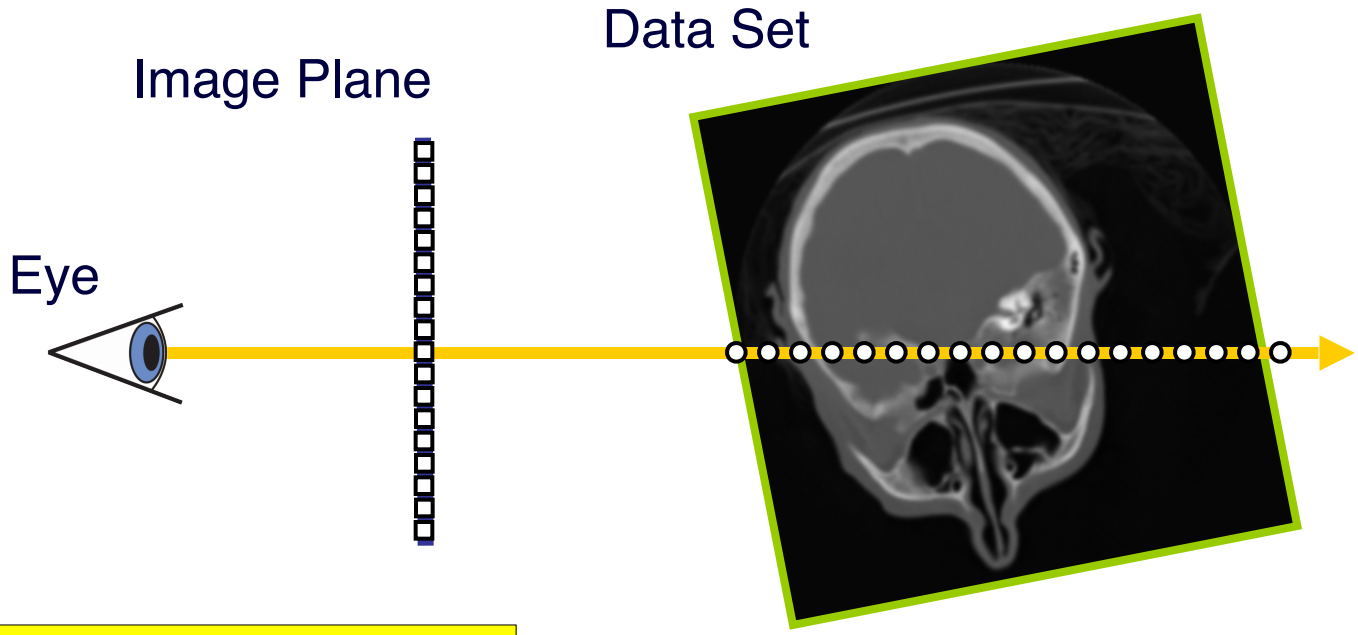
Ray-casting



Ray-casting



Ray-casting

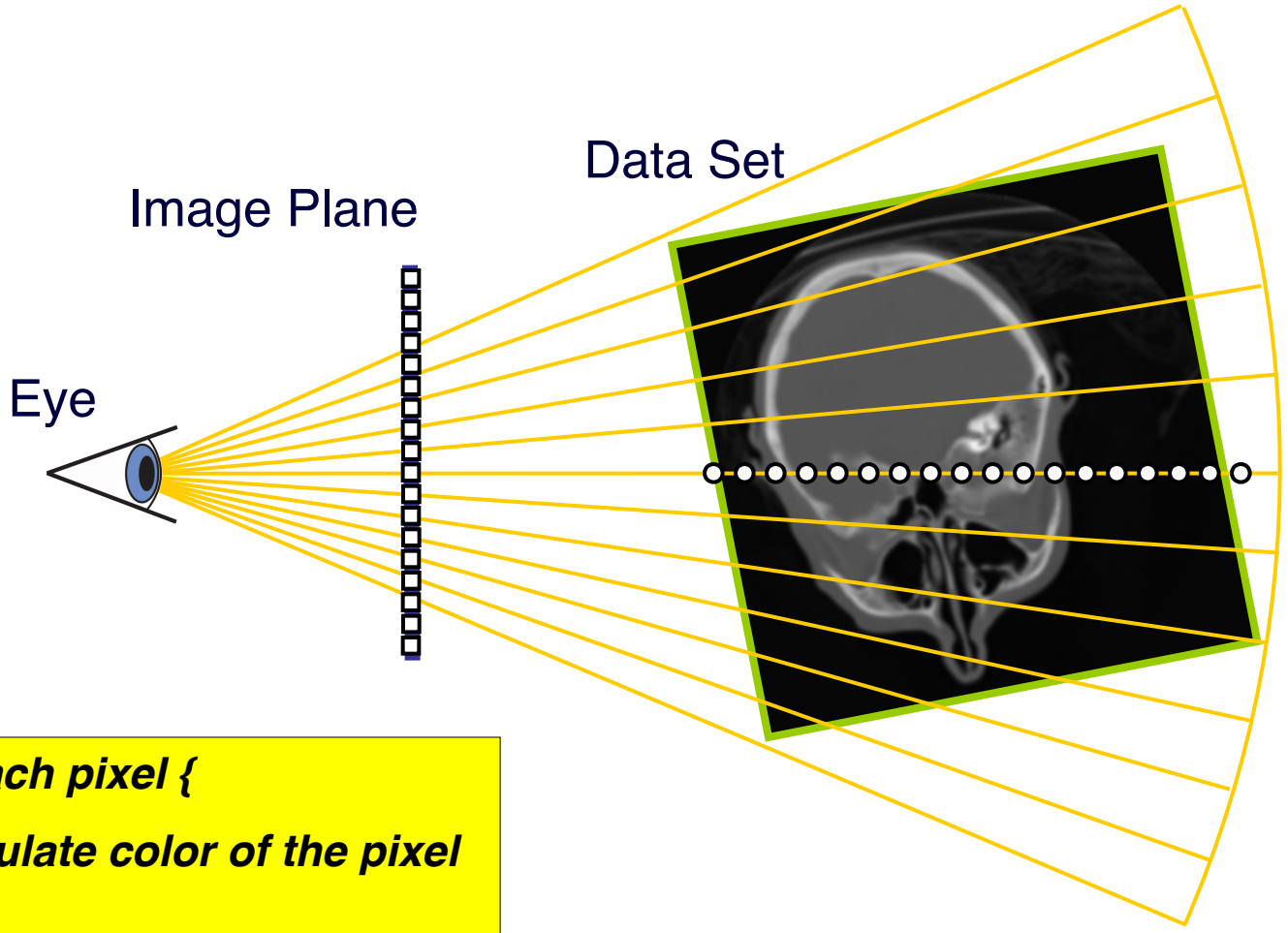


For each pixel {

calculate color of the pixel

}

Ray-casting



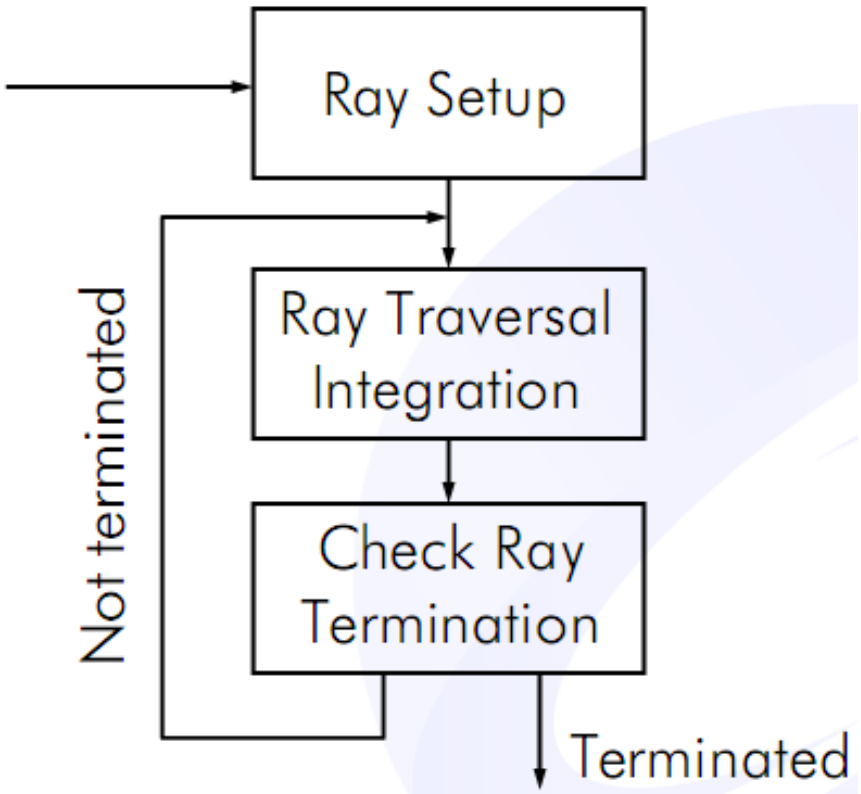
For each pixel {

calculate color of the pixel

}

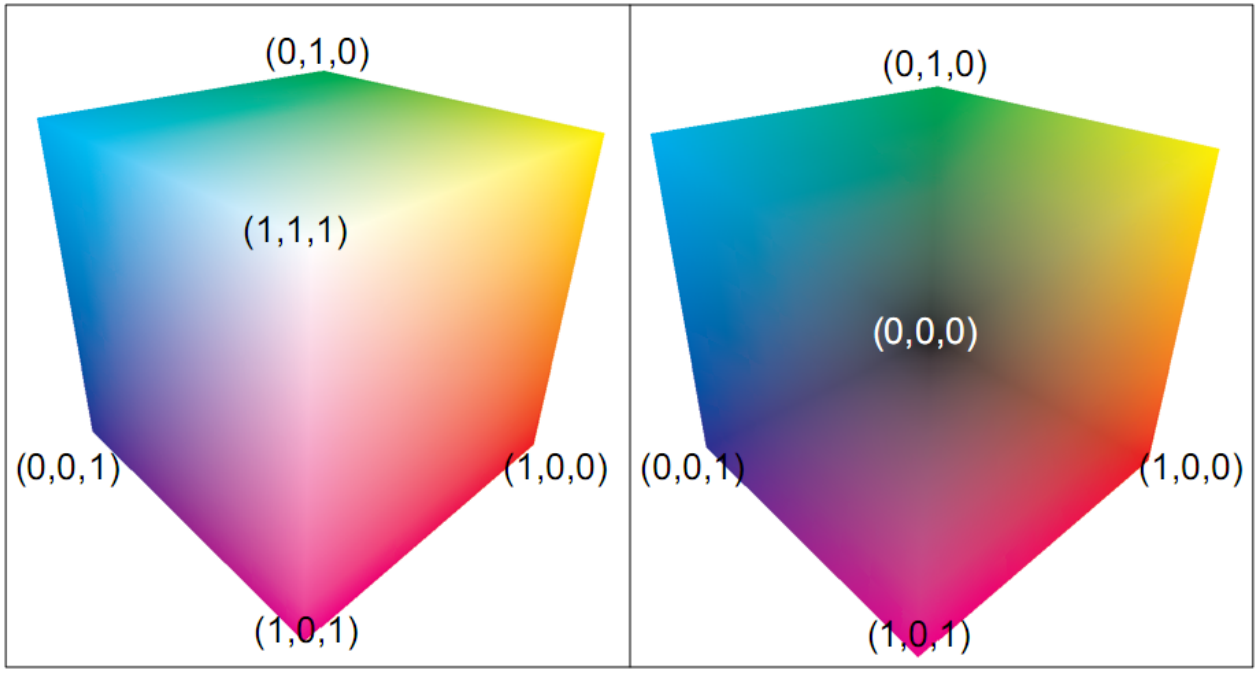
Volume Raycasting on GPU

- Multi-pass algorithm
- Initial passes
  - Precompute ray directions and lengths
- Additional passes
  - Perform raymarching in parallel for each pixel
  - Split up full raymarch to check for early termination



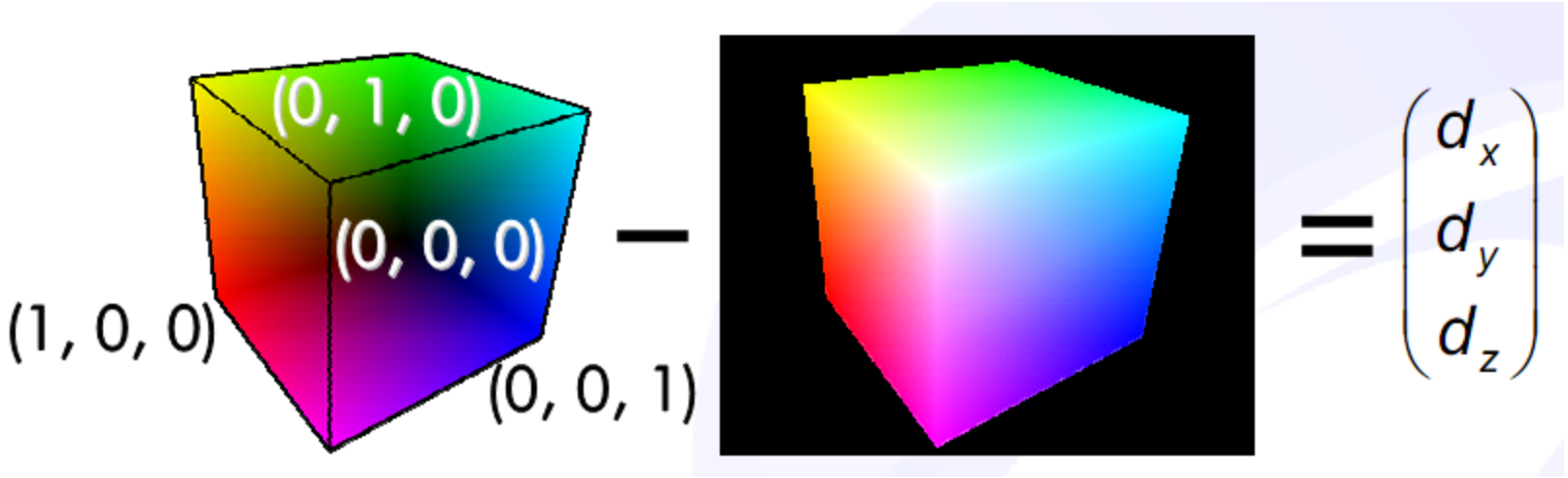
Step 1: Ray Direction Computation

- Ray direction computed for each pixel
- Stored in 2D texture for use in later steps
- Pass 1: Front faces of volume bounding box
- Pass 2: Back faces of volume bounding box
- Vertex color components encode object-space principle directions



Step 1: Ray Direction Computation

- Subtraction blend two textures
- Store normalized direction – RGB components
- Store length – Alpha component



Fragment Shader Ray Marching

- DIR[x][y] – ray direction texture
  - 2D RGBA values
- P – per-vertex float3 positions, front of volume bounding box
  - Interpolated for fragment shader by graphics pipeline
- s – constant step size
  - Float value
- d – total raymarched distance, s x #steps
  - Float value

Fragment Shader Raymarching

- DIR[x][y] – ray direction texture
  - 2D RGBA values
- P – per-vertex float3 positions, front of volume bounding box
  - Interpolated for fragment shader by graphics pipeline
- s – constant step size
  - Float value
- d – total raymarched distance, s x #steps
  - Float value

- Parametric Ray Equation

$$r = P + d \cdot DIR[x][y]$$

  - r – 3D texture coordinates used to sample voxel data

Fragment Shader Ray Marching

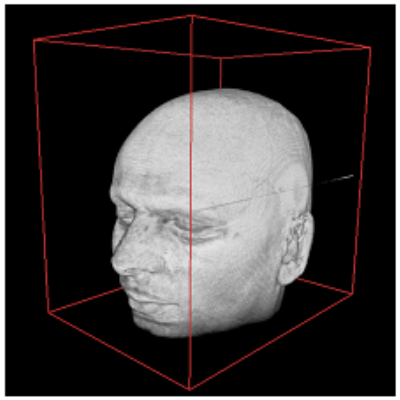
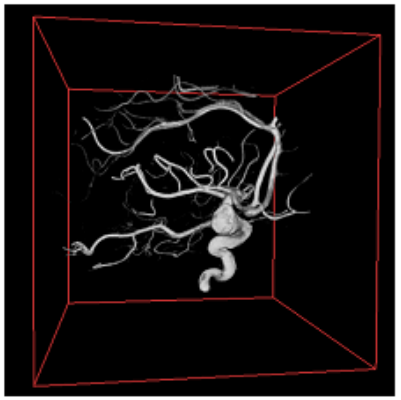
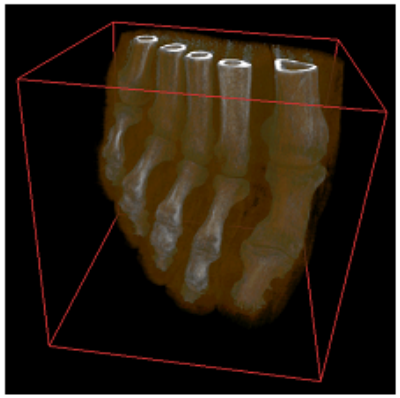
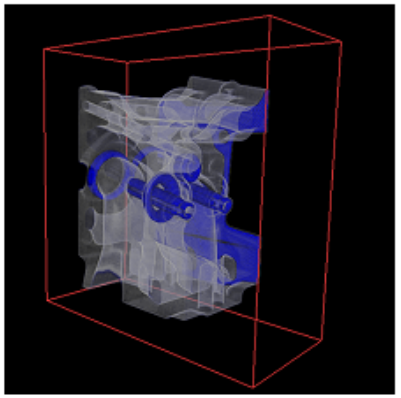
- Ray traversal procedure split into multiple passes
  - M steps along ray for each pass
  - Allows for early ray termination, optimization
- Optical properties accumulated along M steps
  - Simple compositing/blending operations
  - Color and alpha(opacity)
- Accumulation result for M steps blended into 2D result texture
  - Stores overall accumlated values between multiple passes
- Intermediate Pass – checks for early termination
  - Compare opacity to threshold
  - Check for ray leaving bounding volume

Optimizations

- Early Ray Termination
  - Compare accumulated opacity against threshold
- Empty Space Skipping
  - Additional data structure encoding empty space in volume
  - Oct-tree
  - Encode measure of empty within 3D texture read from fragment shader
  - Raymarching fragment shader can modulate sampling distance based on empty space value

Performance and Limitations

- More physically-based than slice-based volume rendering
  - Guarantees equal sampling distances
- Does not incorporate volumetric shadows
- Reduced number of fragment operations
  - Fragment programs made more complex
- Optimizations work best for non-opaque data sets
  - Early ray termination and empty space skipping can be applied

	SBVR	RC	RC-α	RC-β
Head	7.1	4.8	14.4	23.4
Aneurism	7.1	4.8	8.9	19.6
Foot	7.1	4.8	10.2	17.7
Engine	10.4	6.3	6.3	13.6

Isabel Navazo

Pere-Pau Vázquez

Scientific Visualization 2019 – 2020

FUNDAMENTAL BACKGROUND