CSC 544
Data Visualization

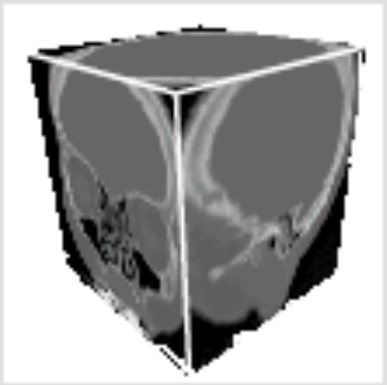
Joshua Levine
josh@arizona.edu

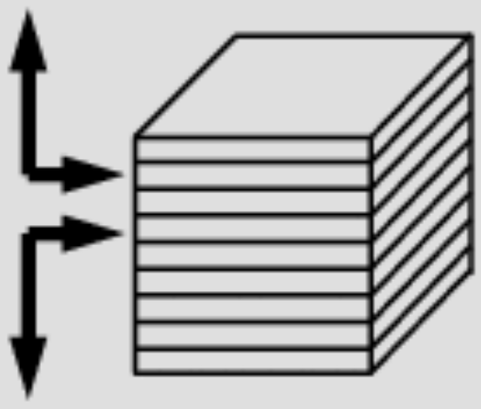
Lecture 19
Volume Rendering

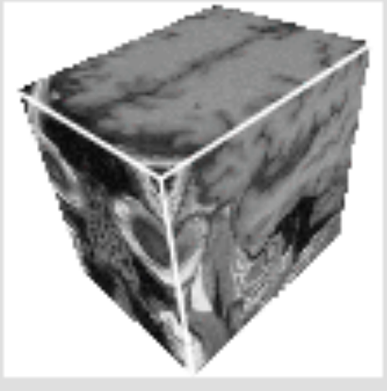
March 27, 2023

Today's Agenda

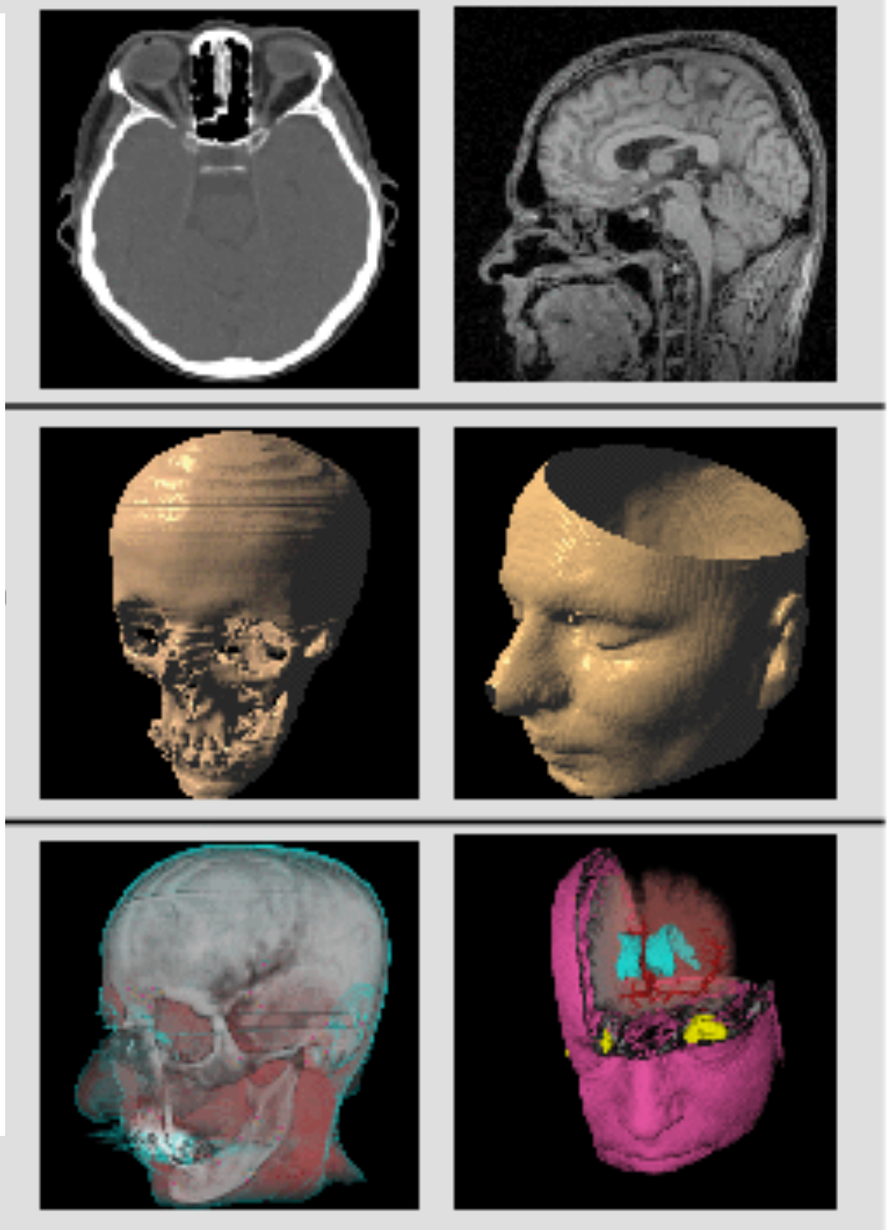
- Reminders:
  - A04 DUE! A05 posted (due Apr. 10)
  - P02 questions (due Mar. 29)?
- Goals for today: Discuss visualizing scalar data using volume rendering

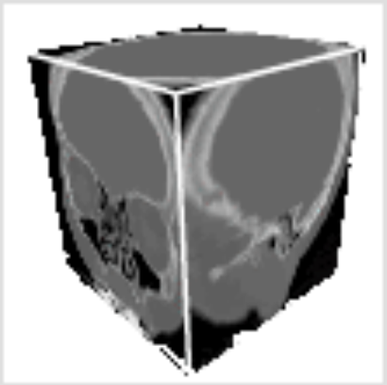


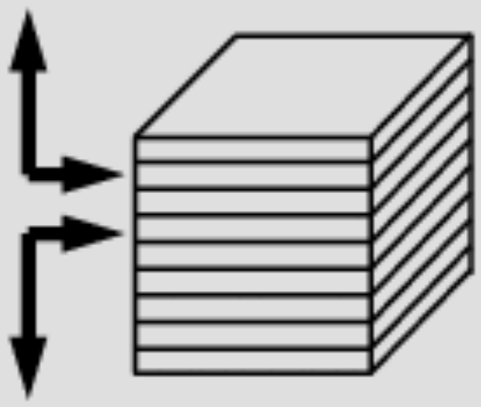


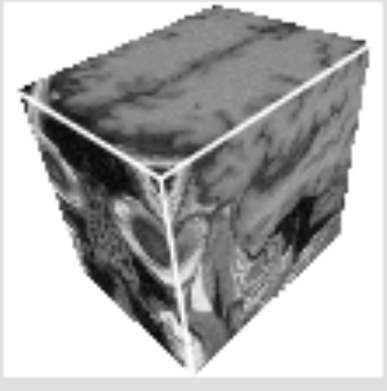


- 2D visualization
  slice images
  (or multi-planar
  reformating MPR)
- *Indirect*
  3D visualization
  isosurfaces
  (or surface-shaded
  display SSD)
- *Direct*
  3D visualization
  (direct volume
  rendering DVR)

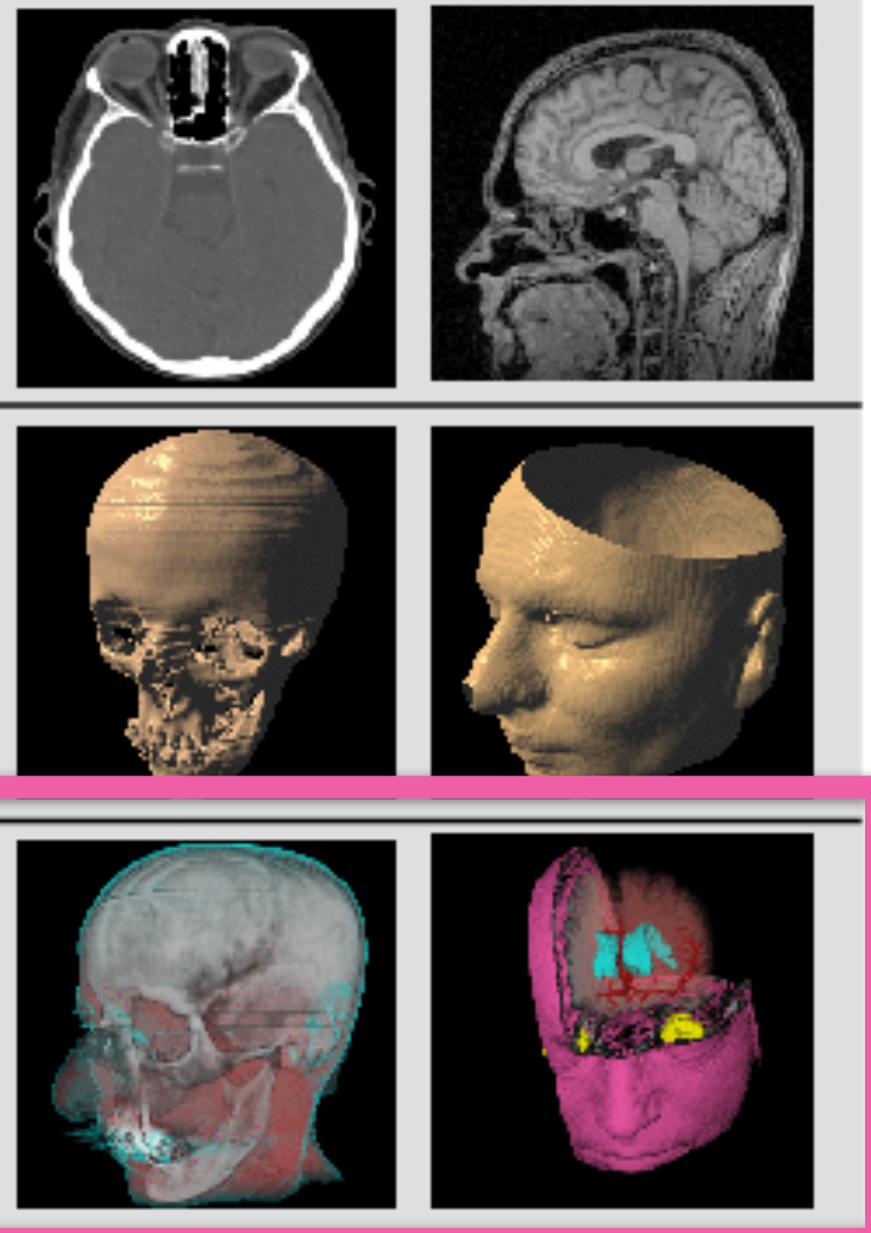








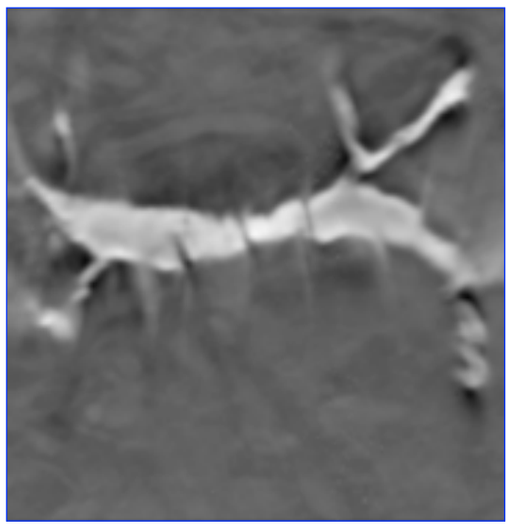
- 2D visualization
  slice images
  (or multi-planar
  reformating MPR)
- *Indirect*
  3D visualization
  isosurfaces
  (or surface-shaded
  display SSD)
- *Direct*
  3D visualization
  (direct volume
  rendering DVR)



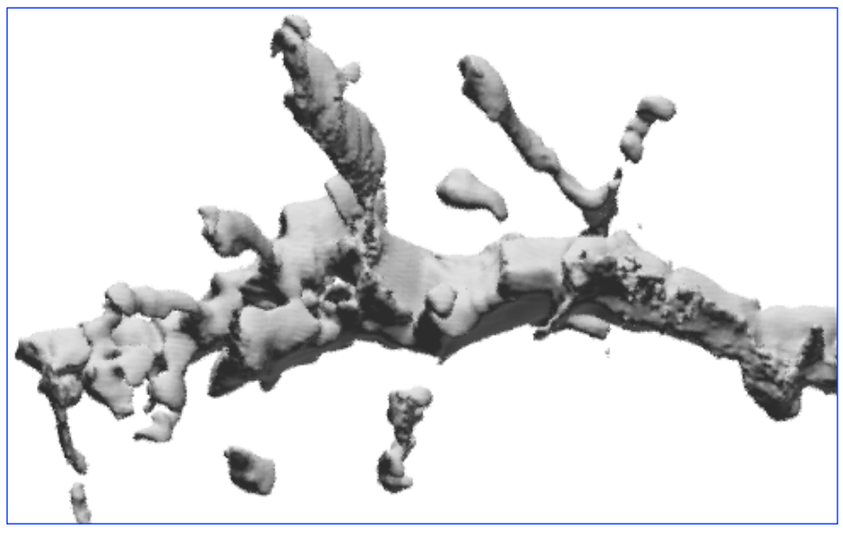
Limitations of Isosurfaces

- Isosurfacing is "binary"
  - What about points inside isosurface?
  - How does each voxel contributes to image?

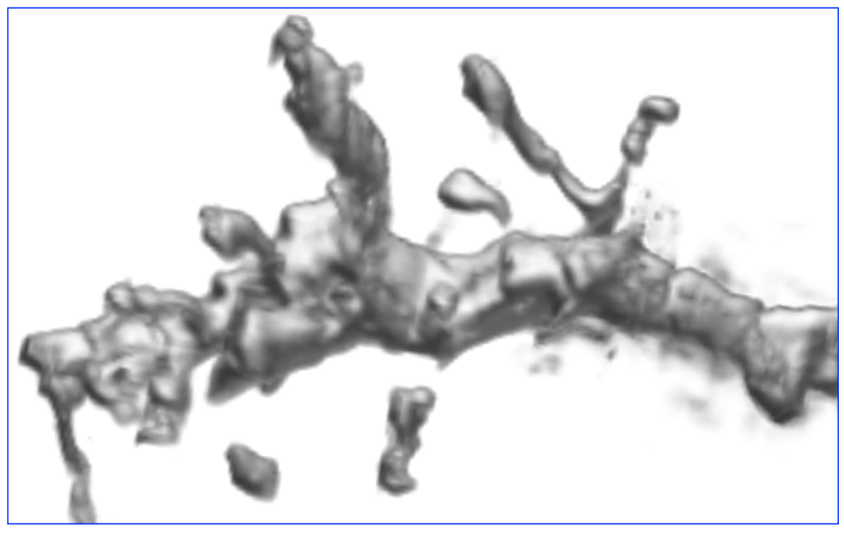
- Is a hard, distinct boundary necessarily appropriate for the visualization task?



Slice

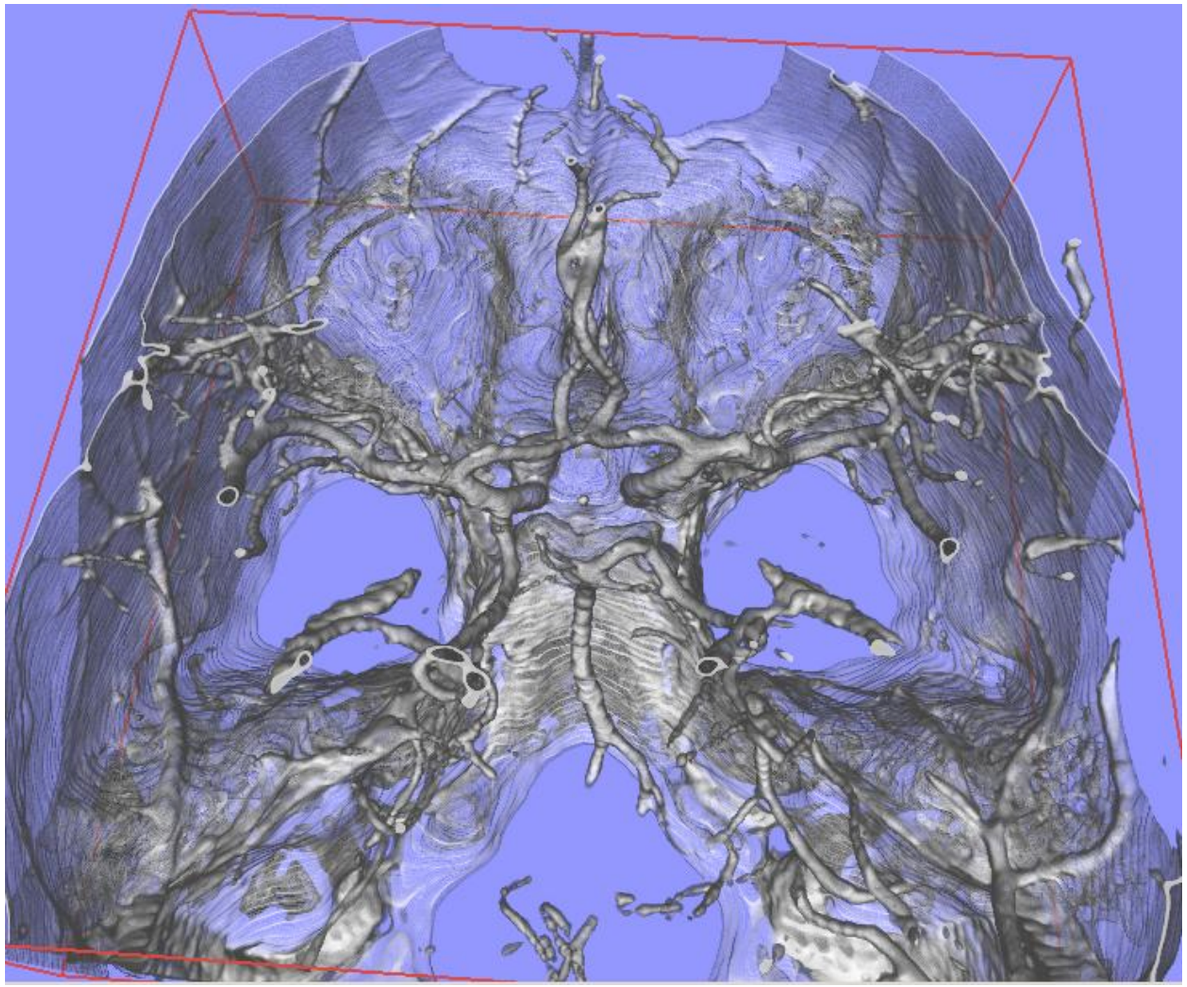


Isosurface

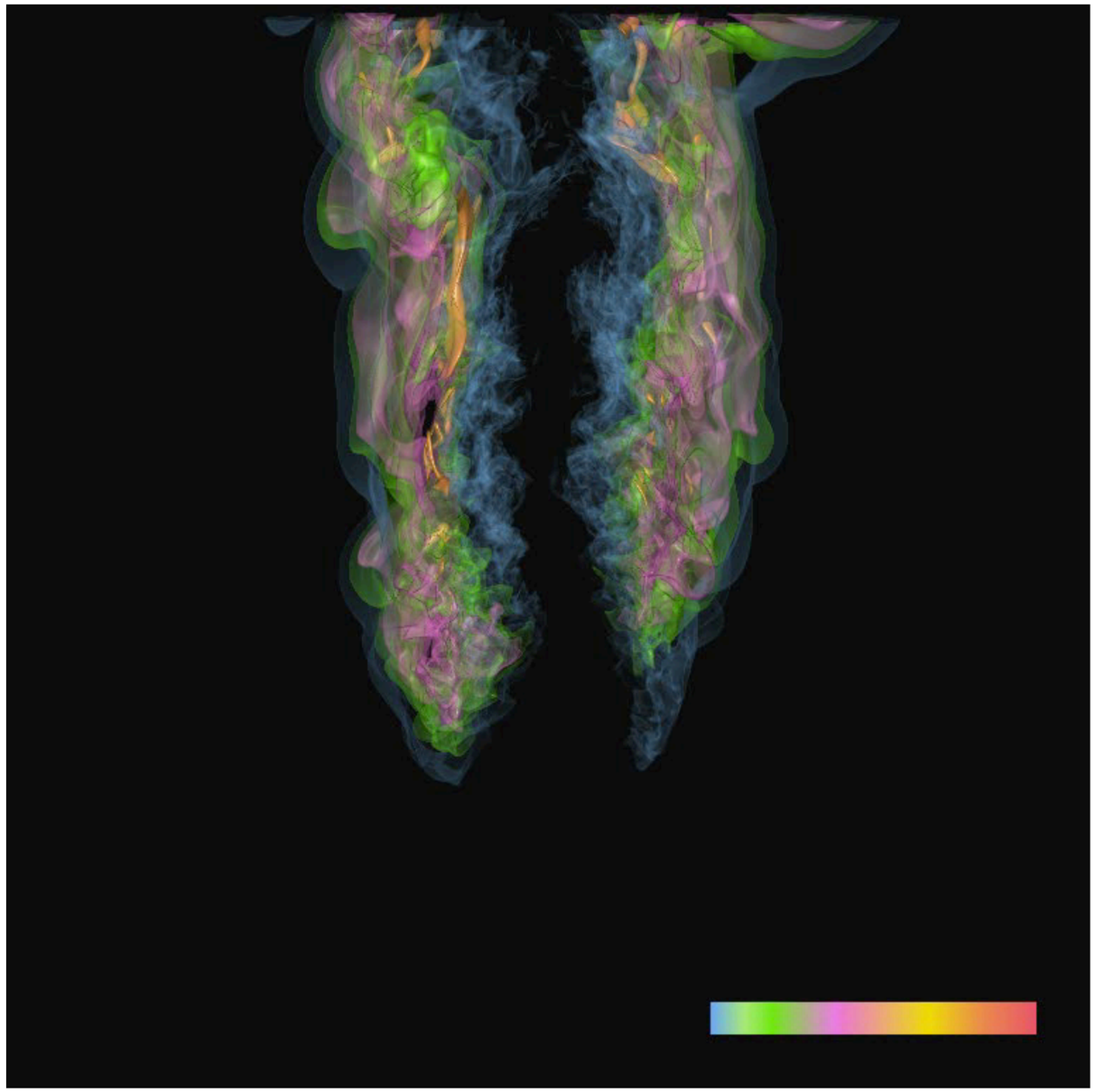


Volume Rendering

- Isosurfacing is poor for ...
  - Measured, "real-world" (noisy) data
  - Amorphous, "soft" objects

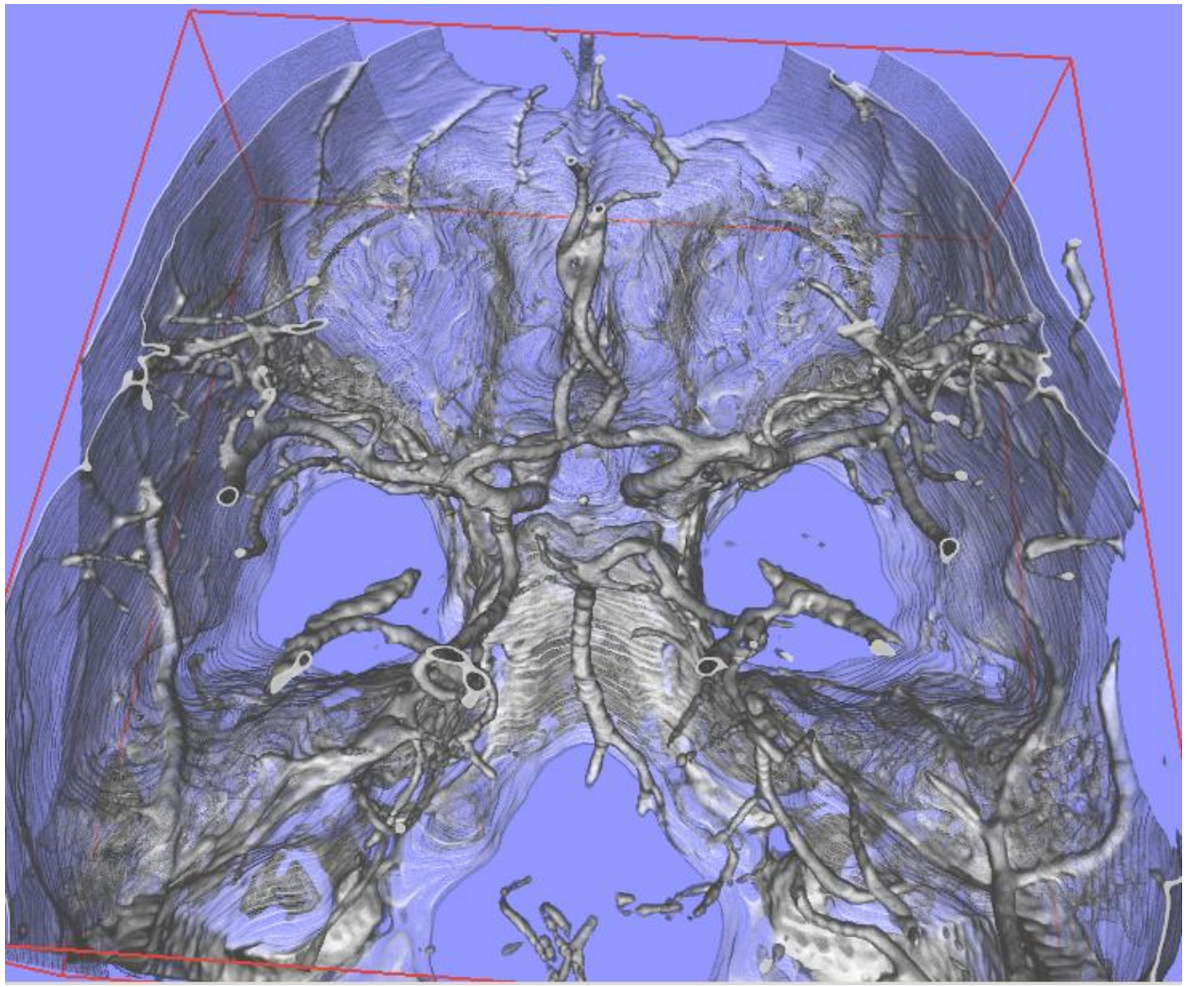


http://www.cg.informatik.uni-siegen.de/

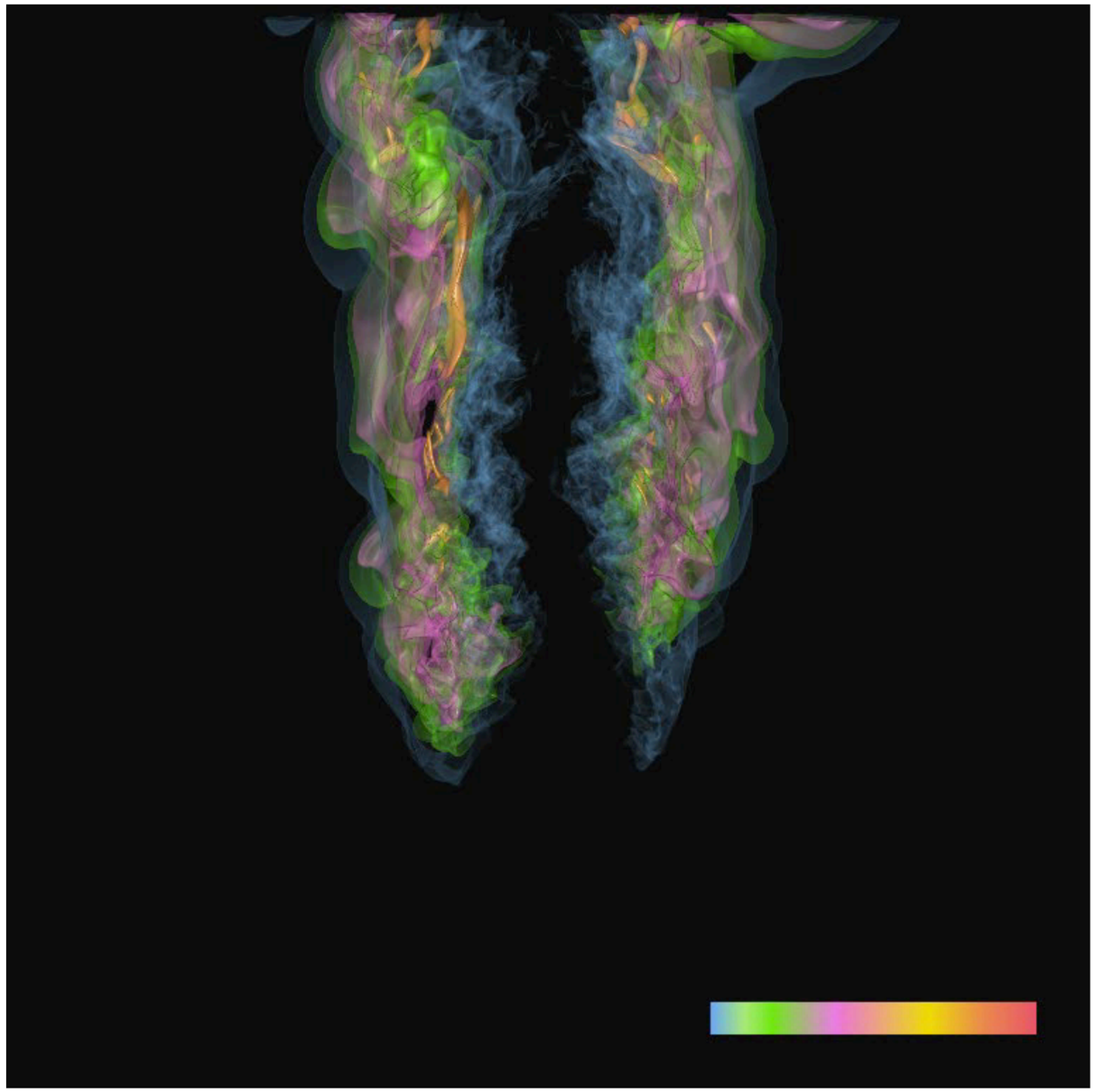


http://vis.cs.ucdavis.edu/gallery/Yu/combustion/

- Isosurfacing is poor for ...
  - Measured, "real-world" (noisy) data
  - Amorphous, "soft" objects



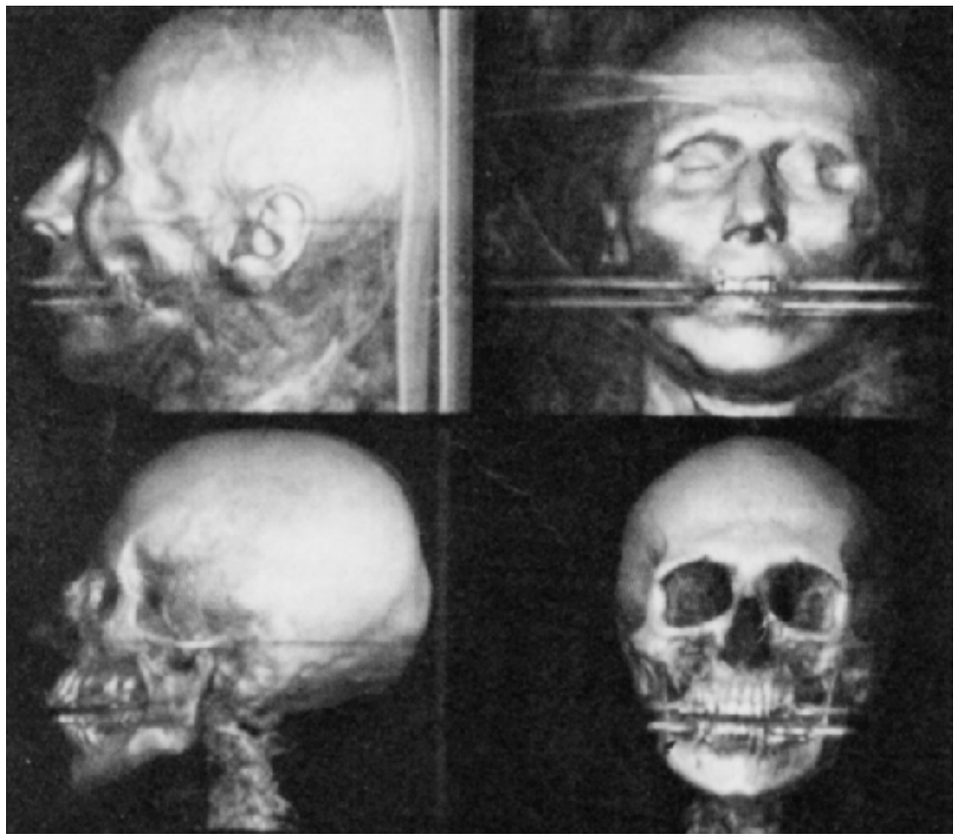
http://www.cg.informatik.uni-siegen.de/



http://vis.cs.ucdavis.edu/gallery/Yu/combustion/

Why Volume Rendering?

- Allows every voxel to contribute to image
- Provides greater flexibility



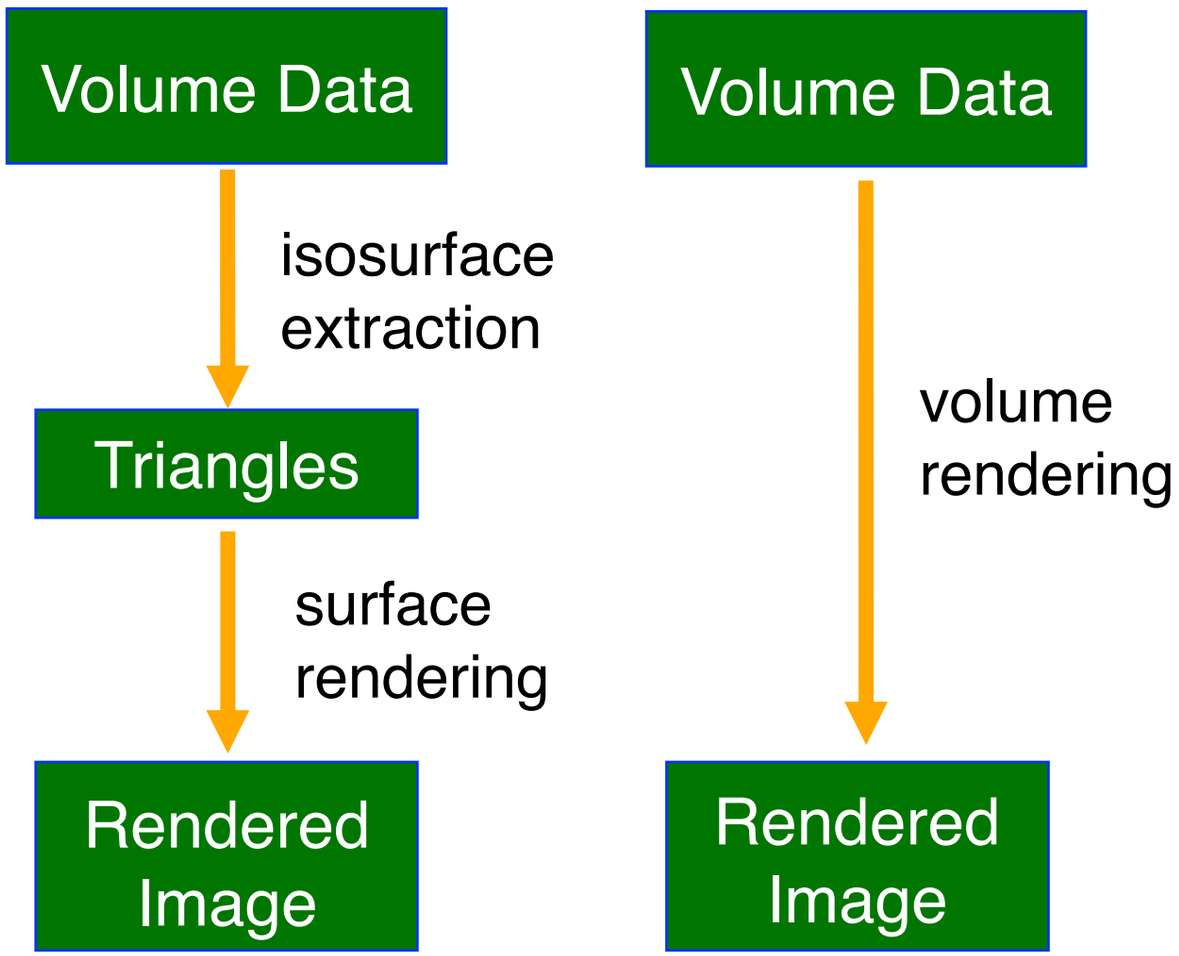
Marc Levoy, Display of Surfaces from Volume Data, 1988

Volume Rendering

- The key idea for today:
    - Generally: Visualization transforms data into **primitives** that are displayed.

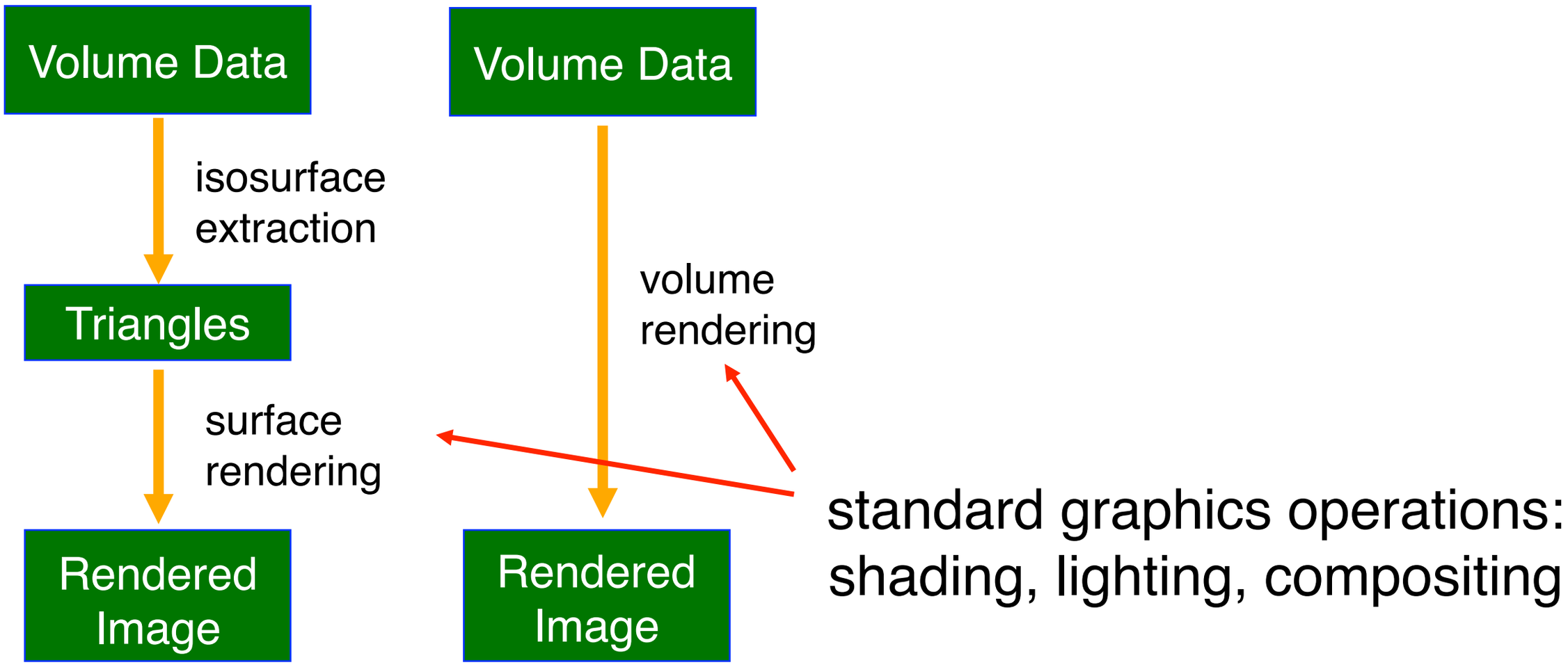
- The key idea for today:
    - Generally: Visualization transforms data into **primitives** that are displayed.
    - Instead: Visualization transforms data into **primitives** that are **rendered** using (three-dimensional) **computer graphics** techniques

Pipelines: Isosurfacing vs. Volume Rendering



Pipelines: Isosurfacing vs. Volume Rendering

the standard pipeline:
no intermediate geometric structures



Course Notes 28

Real-Time Volume Graphics

Klaus Engel
Siemens Corporate Research, Princeton, USA

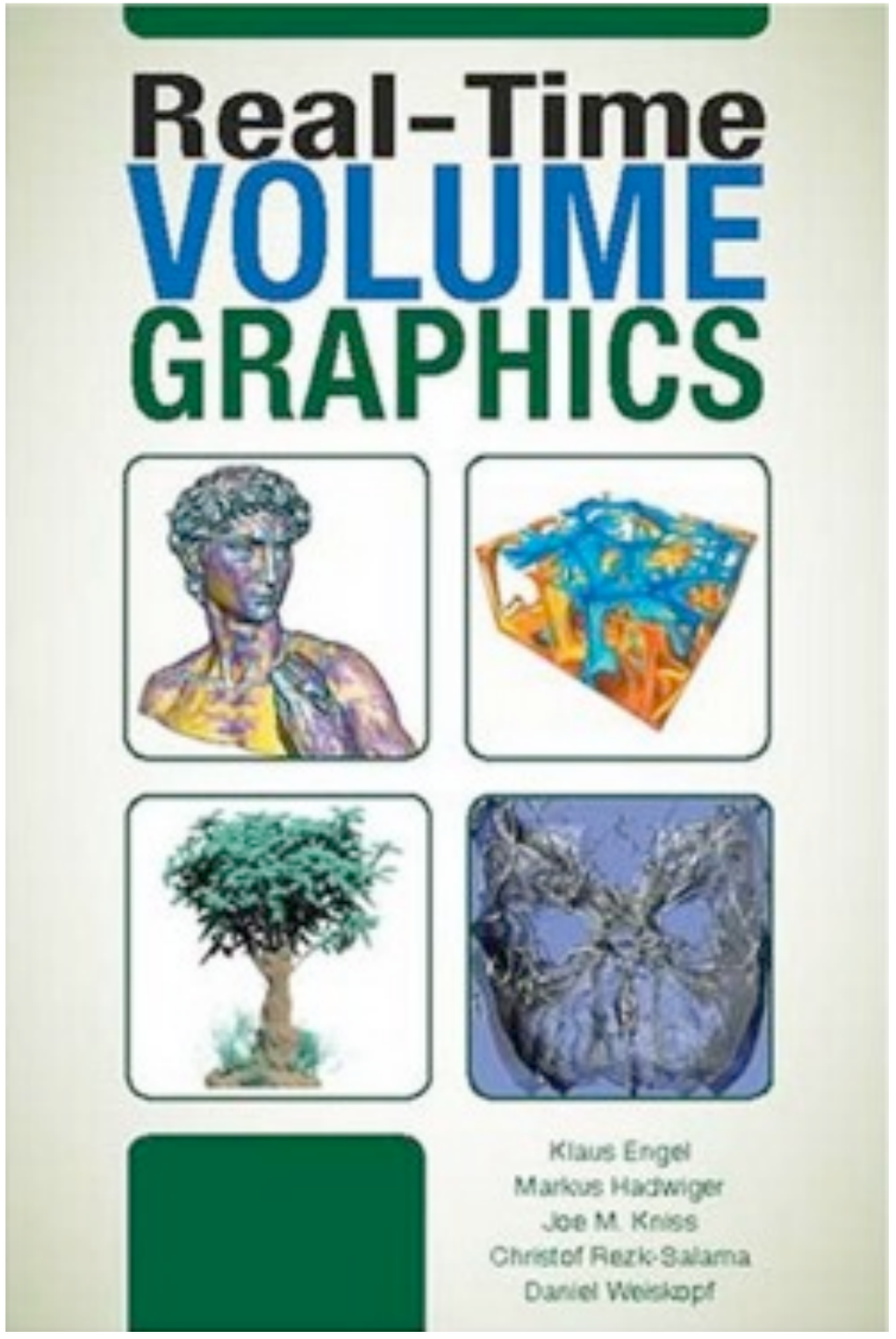
Markus Hadwiger
VR VIS Research Center, Vienna, Austria

Joe M. Kniss
SCI, University of Utah, USA

Aaron E. Lefohn
University of California, Davis, USA

Christof Rezk Salama
University of Siegen, Germany

Daniel Weiskopf
University of Stuttgart, Germany



SIGGRAPH2004

http://www.real-time-volume-graphics.org/

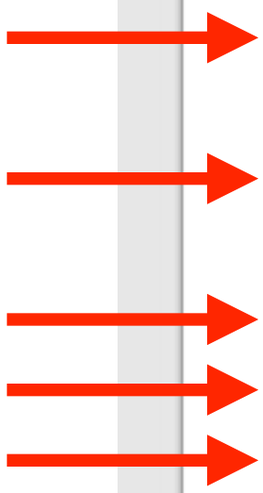
Real-Time Volume Graphics

GLU-based volume rendering for scientific visualization and visual arts

Tutorials

Real-Time Volume Graphics Tutorial

Here are the slides form the original course, the book is based upon. The latest revision are the powerpoint slides of the Tutorial held at the Eurographics 2006 conference in Vienna, Austria, September/5/2006. The same course was also held at SIGGRAPH



» Part 01: Introduction and Theory (ppt) (pdf)

» Part 02: GPU Programming (ppt) (pdf)

» Part 03: Texture-Based Volume Rendering (ppt) (pdf)

» Part 04: GPU-Based Ray Casting (ppt) (pdf)

» Part 05: Transfer Functions (ppt) (pdf)

» Part 06: Local Illumination (ppt) (pdf)

» Part 07: Global Illumination (ppt) (pdf)

» Part 08: Improving Performance (ppt) (pdf)

» Part 09: Improving Quality (ppt) (pdf)

» Part 10: Transfer Functions Reloaded (ppt) (pdf)

» Part 11: Game Developers Guide to Volume Graphics (ppt) (pdf)

» Part 12: Volume Modeling, Deformation and Animation (ppt) (pdf)

Recent Comments

» Priscilla on Voreen Framework
» is quibids legit on Voreen Framework
» Prweb.Com on Voreen Framework
» musculo on Voreen Framework
» Антон Павлович on Voreen

What is Volume Rendering?

- Any rendering process which maps from volume data to an image (without introducing binary distinctions or intermediate geometry)

Core question (for the visualizer):

- How do you make the data visible?
- Answer: color and opacity

Direct 3D Representation of Volume Data

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

- DVR: Render volume **without extracting any surfaces**
- Map scalar values to **optical properties** (color, opacity)
- Needs an **optical model**
- Solve **volume rendering integral** for viewing rays into the volume

ParaView Demo

Volume Ray Casting

Image order approach

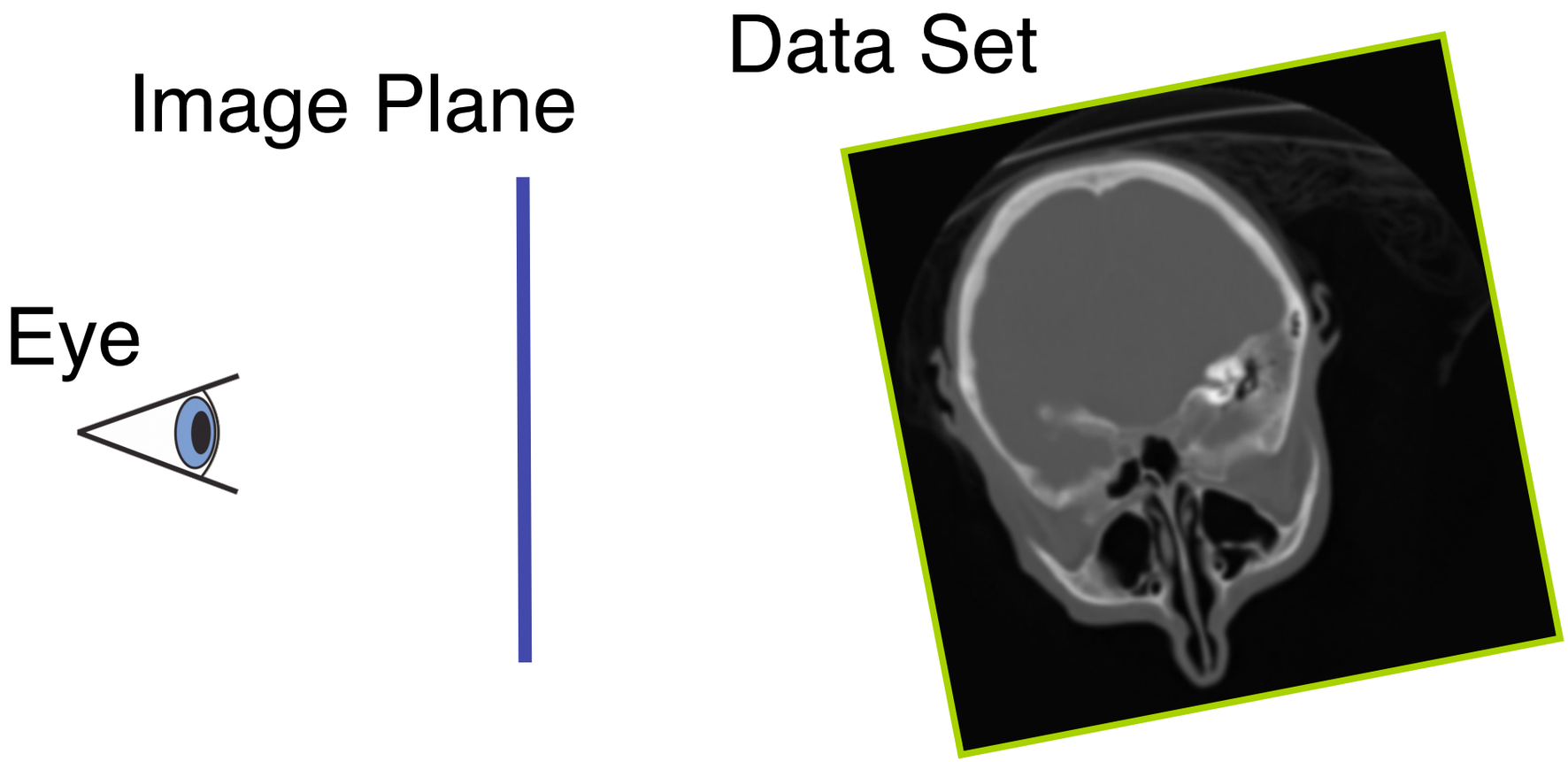


Image order approach

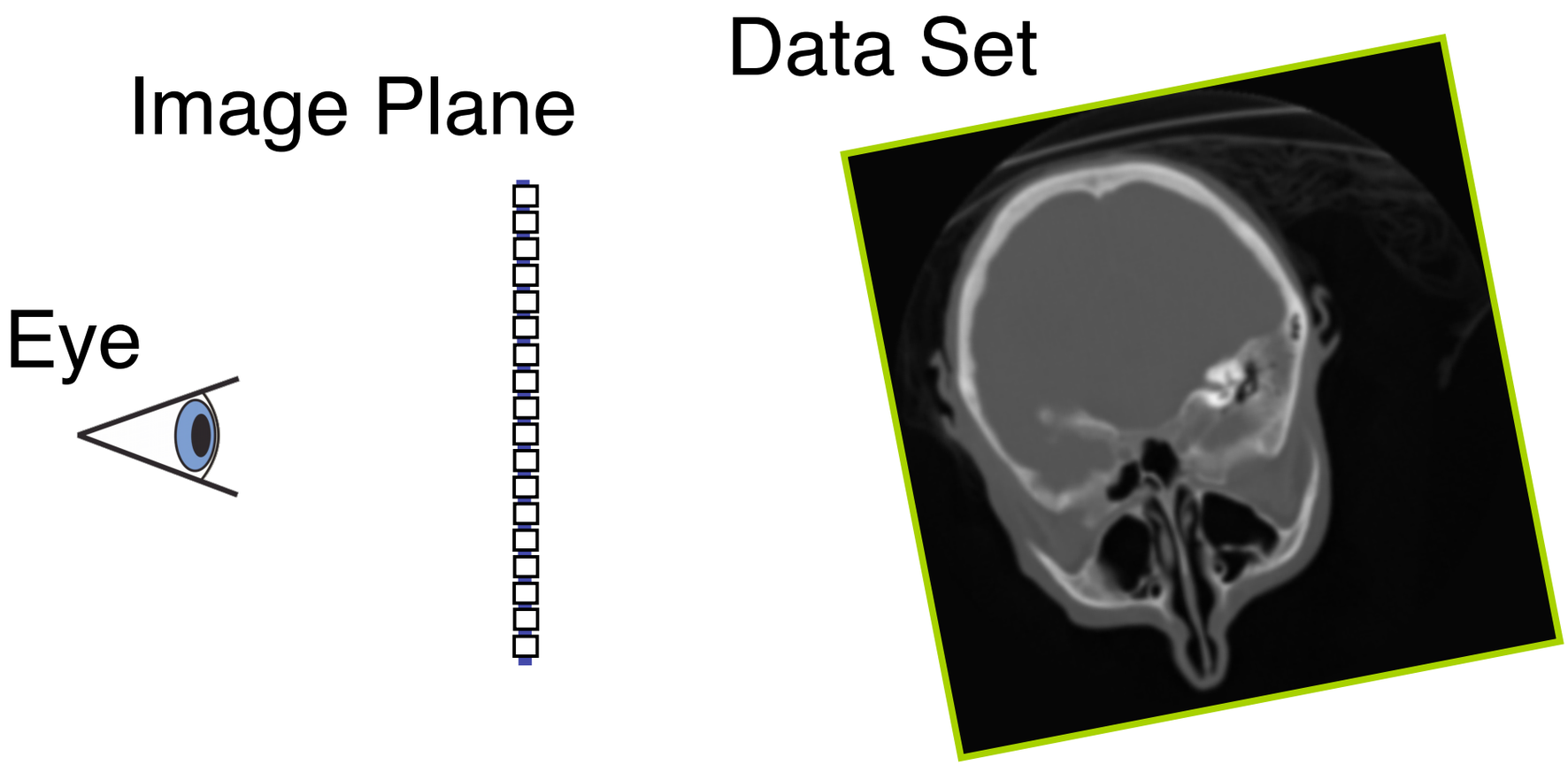


Image order approach

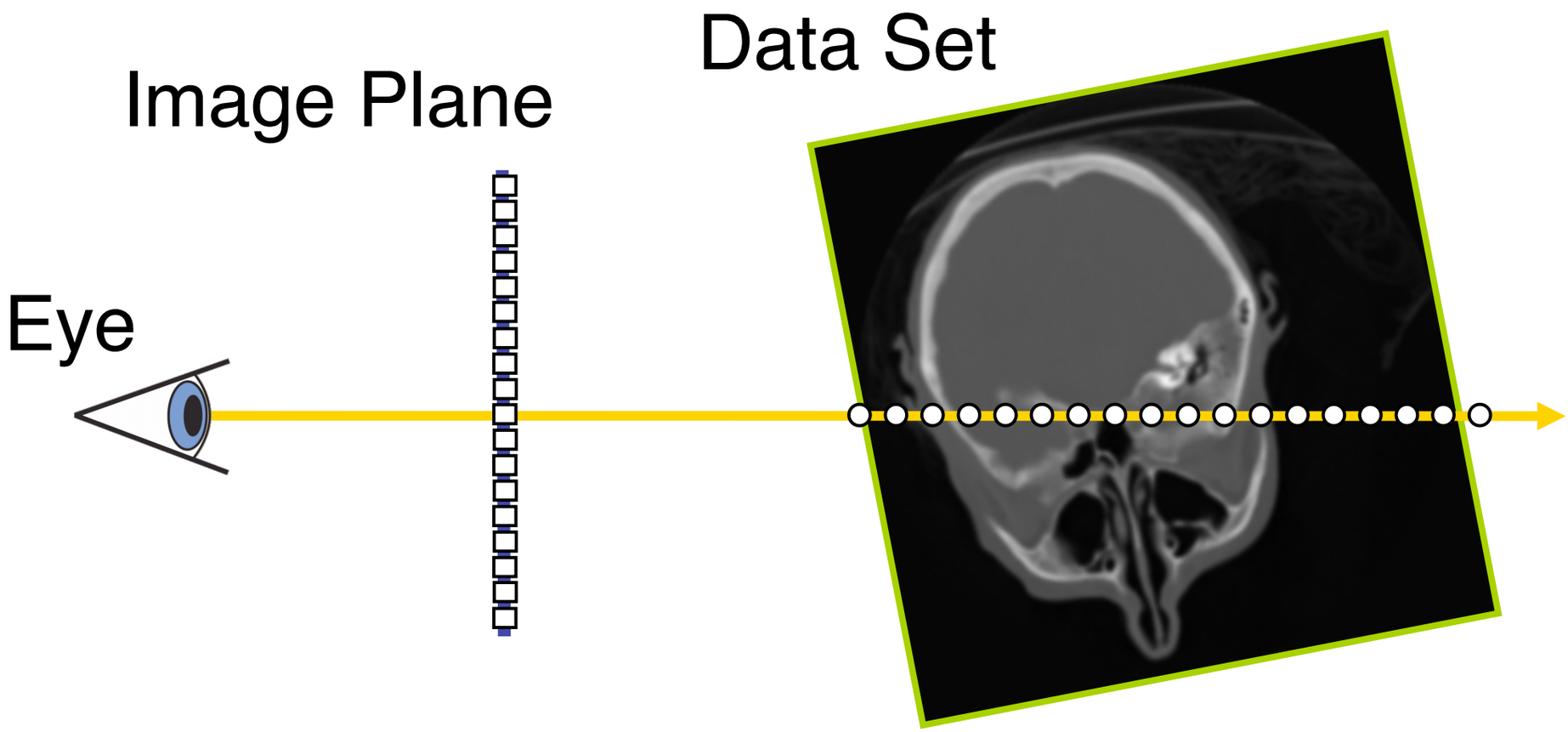
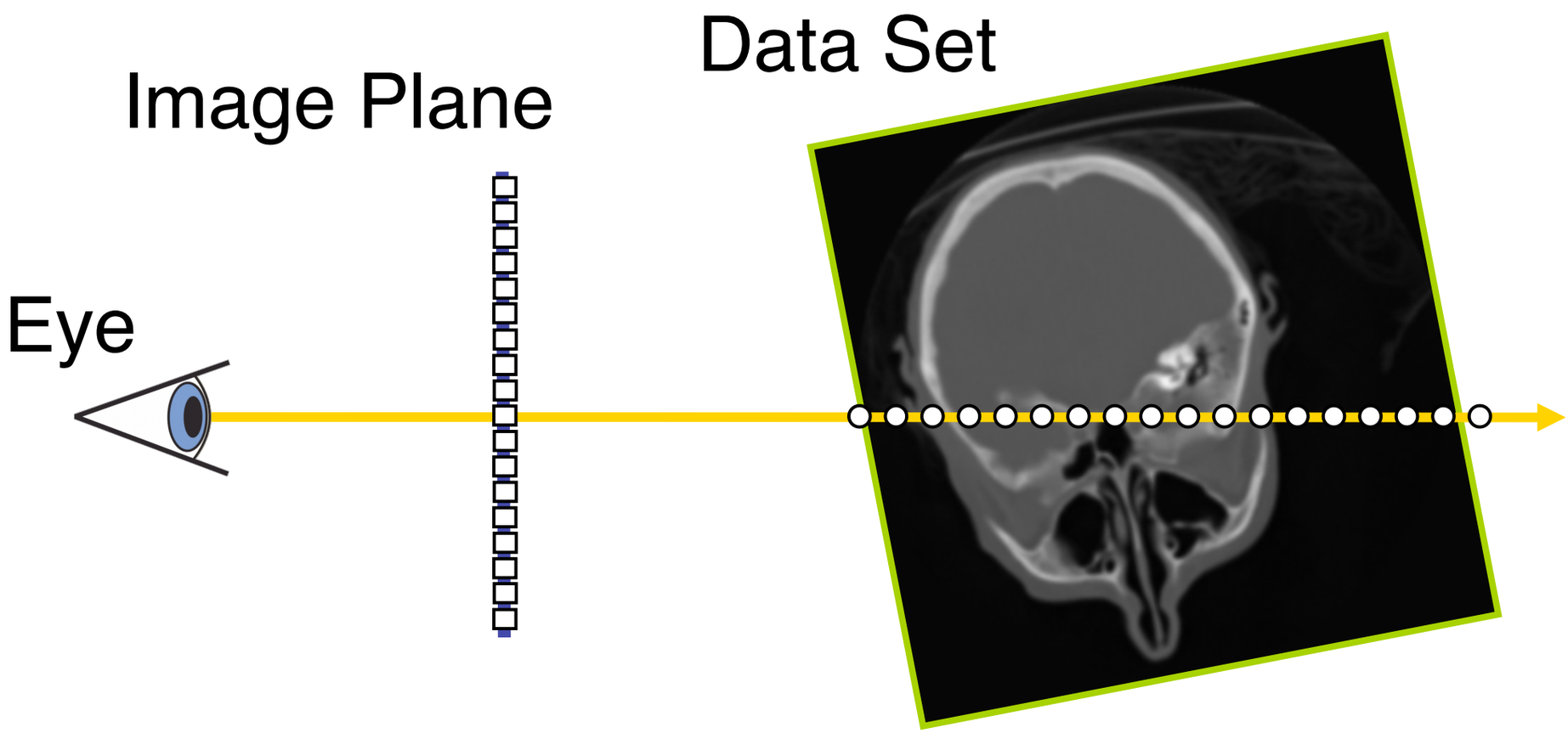
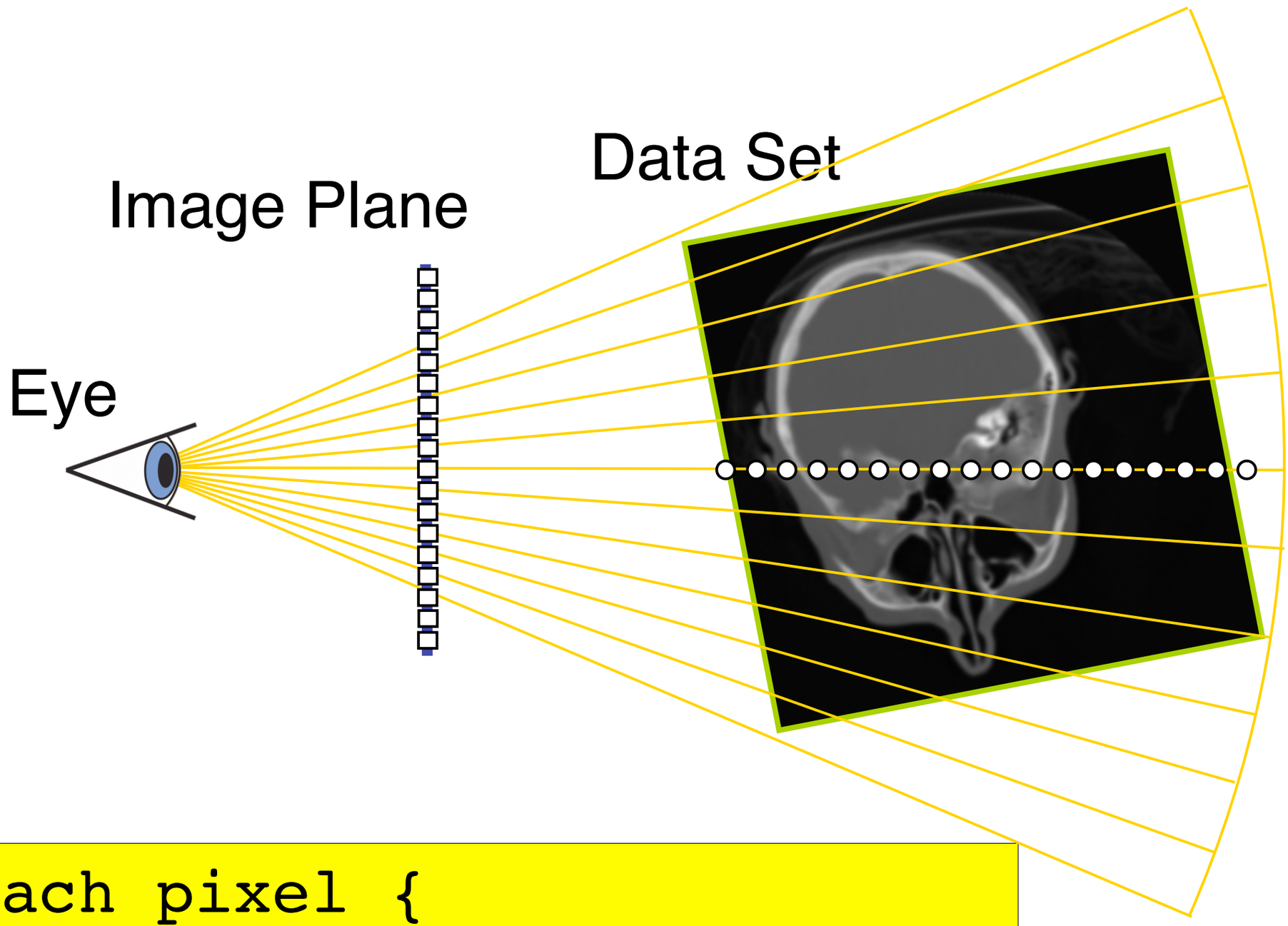


Image order approach




```
For each pixel {
    calculate color of the pixel
}
```

Image order approach



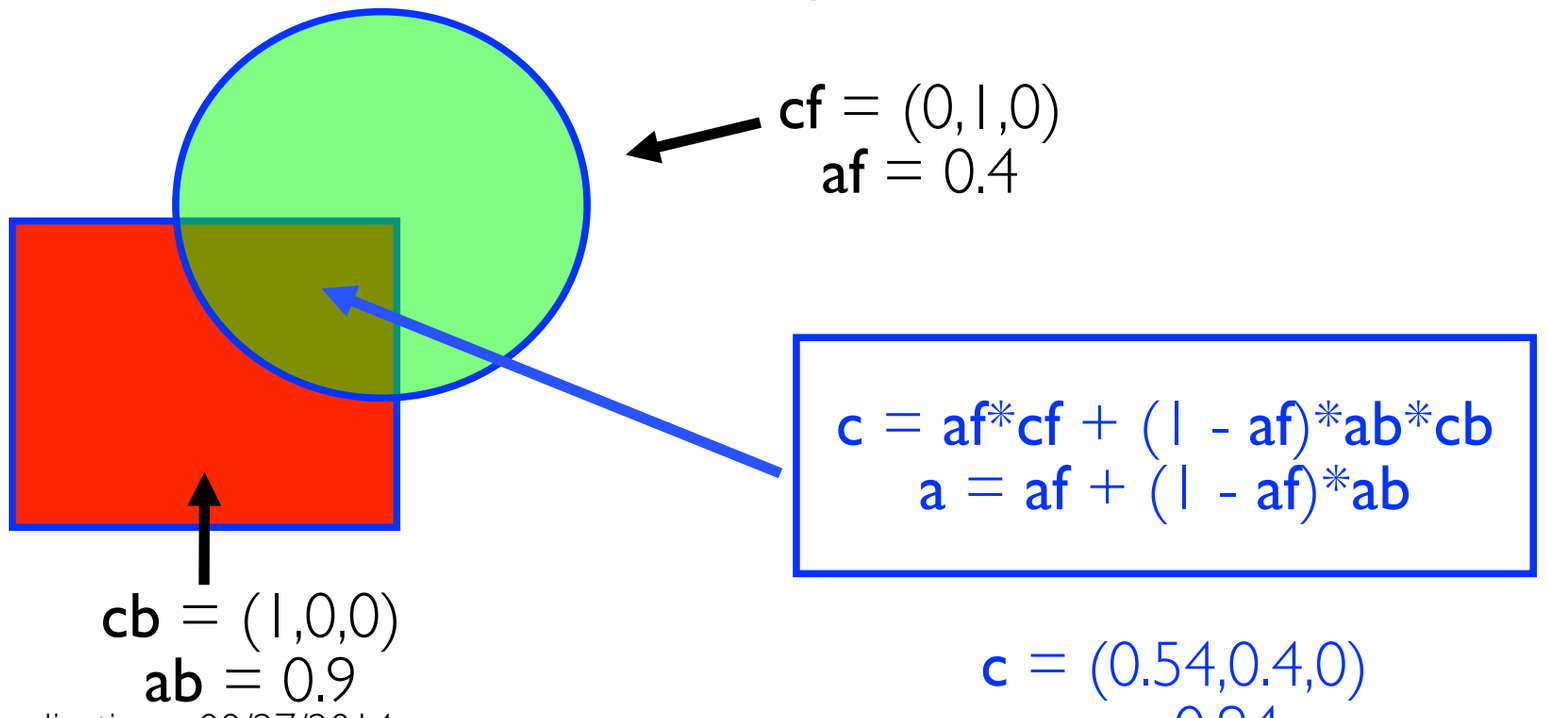

```
For each pixel {
    calculate color of the pixel
}
```

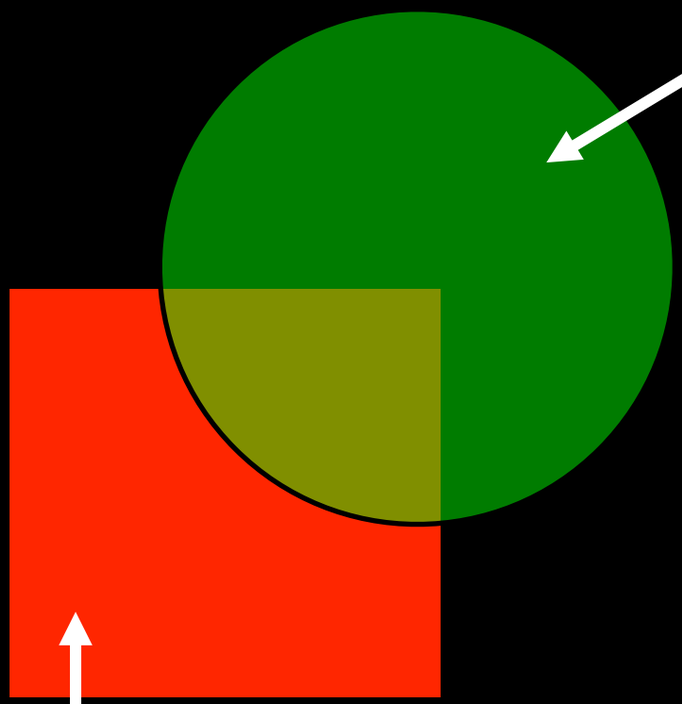
Compositing

Fundamental Algorithms

Alpha blending / compositing

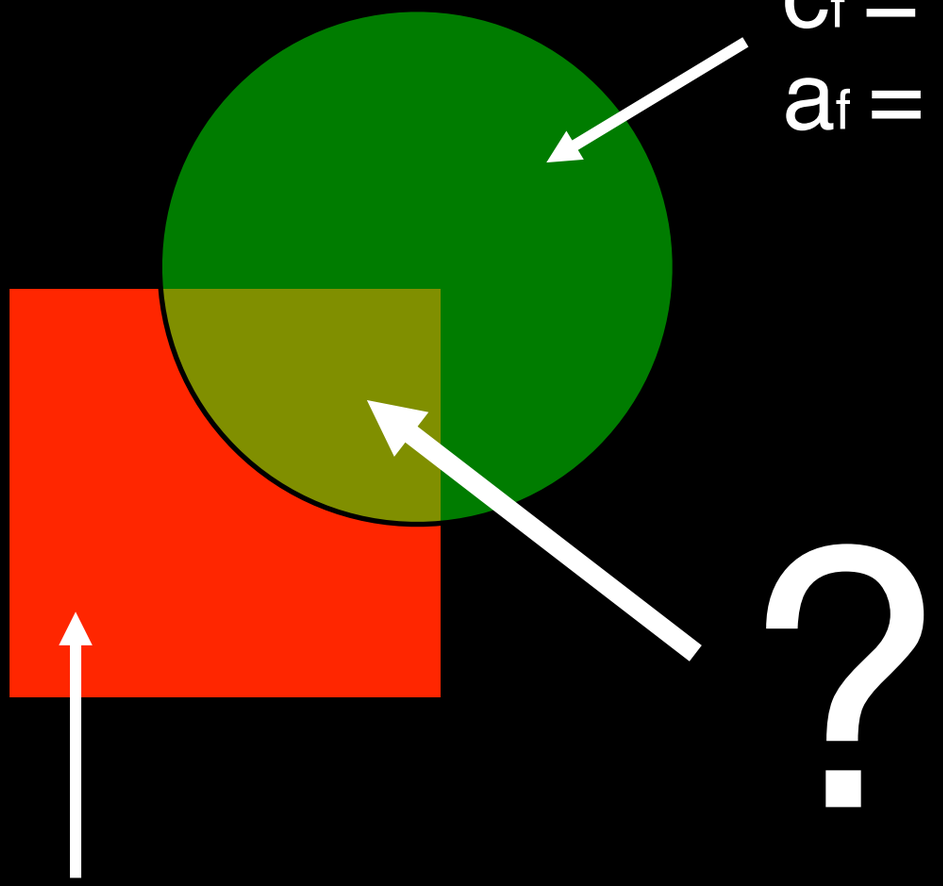
- Approximate visual appearance of semi-transparent object in front of another object
- Implemented with OVER operator

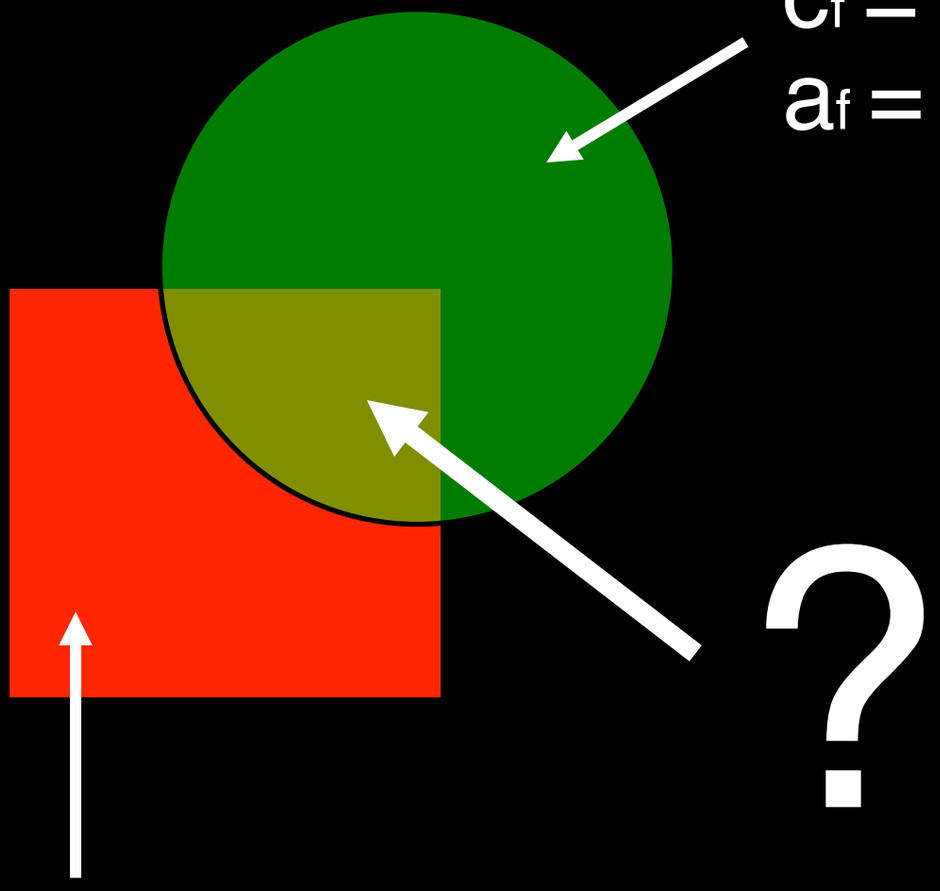




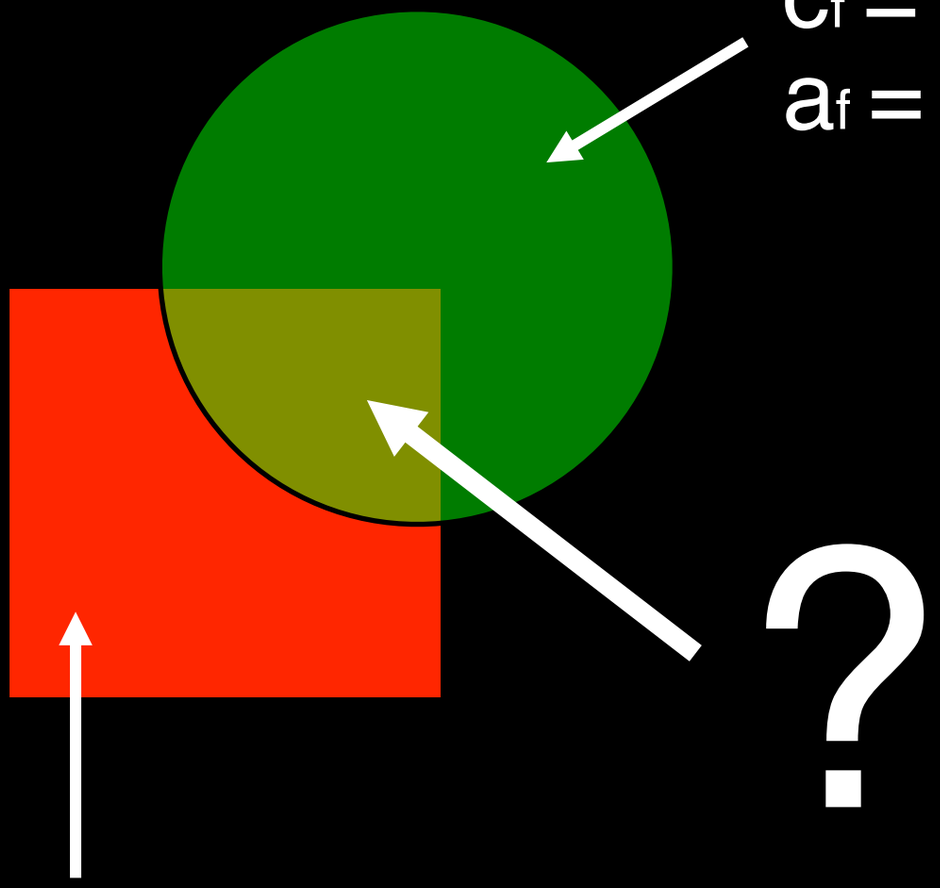
$c_f = (0,1,0)$
$a_f = 0.4$

$c_b = (1,0,0)$
$a_b = 0.9$


$c_f = (0,1,0)$
$a_f = 0.4$
?
$c_b = (1,0,0)$
$a_b = 0.9$

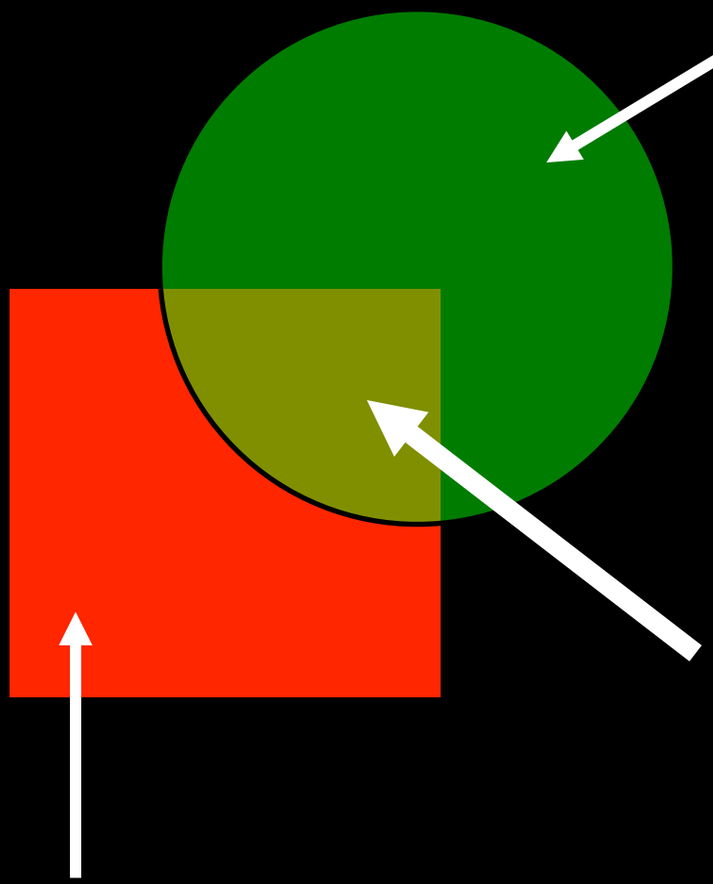


over operator



over operator

$$c = a_f * c_f + (1 - a_f) * a_b * c_b$$
$$a = a_f + (1 - a_f) * a_b$$



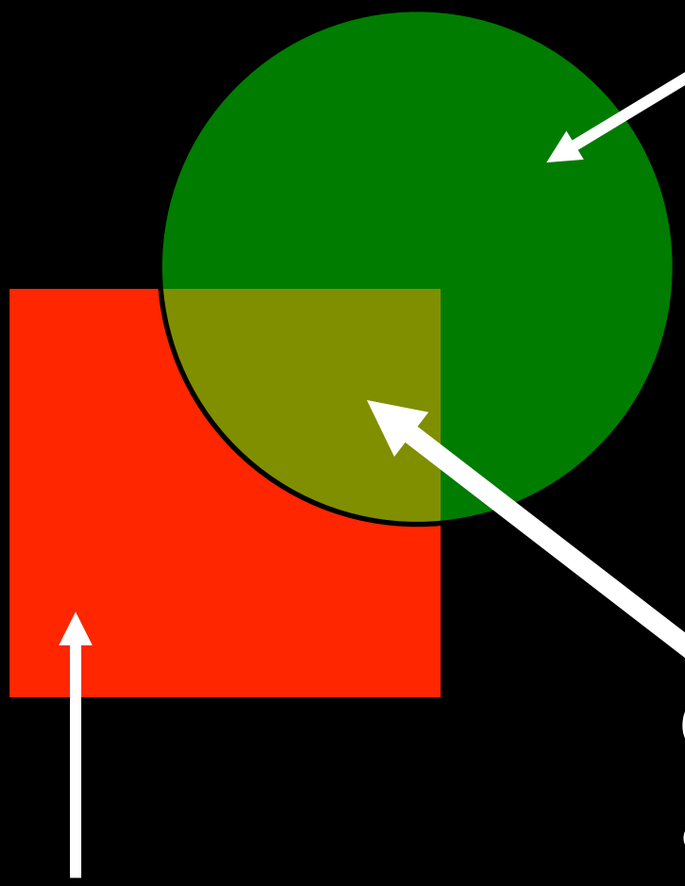
$c_f = (0,1,0)$
$a_f = 0.4$

$c_{red} = 0.4*0 + (1-0.4)*0.9*1 = 0.6*0.9 = 0.54$
$c_{green} = 0.4*1 + (1-0.4)*0.9*0 = 0.4$
$c_{blue} = 0.4*0 + (1-0.4)*0.9*0 = 0$
$a = 0.4 + (1 - 0.4)*(0.9) = 0.4 + 0.6*0.9)$

$c_b = (1,0,0)$
$a_b = 0.9$

over operator

$$c = a_f * c_f + (1 - a_f) * a_b * c_b$$
$$a = a_f + (1 - a_f) * a_b$$



$c_f = (0,1,0)$
$a_f = 0.4$

$c_{red} = 0.4*0 + (1-0.4)*0.9*1 = 0.6*0.9 = 0.54$
$c_{green} = 0.4*1 + (1-0.4)*0.9*0 = 0.4$
$c_{blue} = 0.4*0 + (1-0.4)*0.9*0 = 0$
$a = 0.4 + (1 - 0.4)*(0.9) = 0.4 + 0.6*0.9)$

$c = (0.54,0.4,0)$
$a = 0.94$

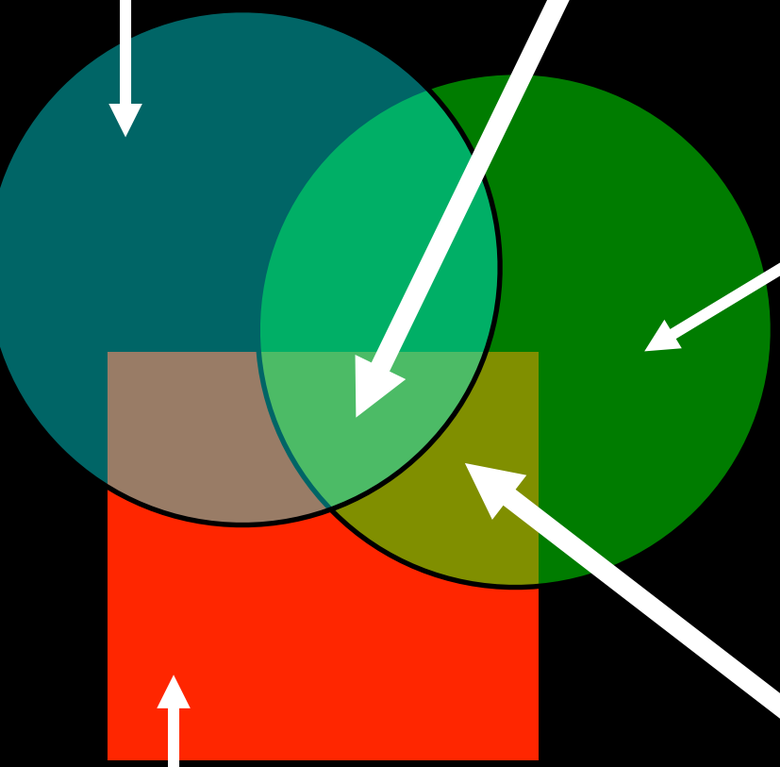
$c_b = (1,0,0)$
$a_b = 0.9$

over operator

$$c = a_f * c_f + (1 - a_f) * a_b * c_b$$
$$a = a_f + (1 - a_f) * a_b$$

$c_f = (0,1,1)$
$a_f = 0.4$

?



$c_1 = (0,1,0)$
$a_1 = 0.4$

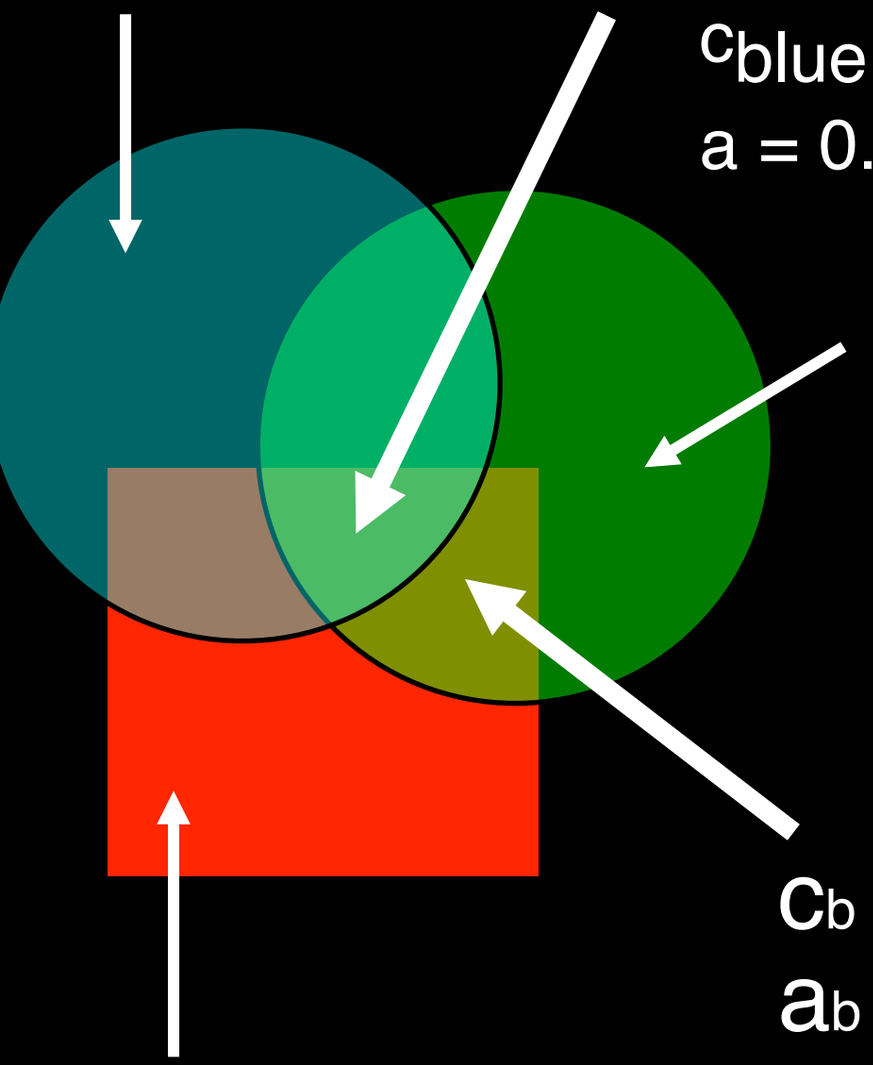
$c_b = (0.54,0.4,0)$
$a_b = 0.94$

$c_2 = (1,0,0)$
$a_2 = 0.9$

over operator

$$c = a_f * c_f + (1 - a_f) * a_b * c_b$$
$$a = a_f + (1 - a_f) * a_b$$

$c_f = (0,1,1)$
$a_f = 0.4$



$c_1 = (0,1,0)$
$a_1 = 0.4$

$c_2 = (1,0,0)$
$a_2 = 0.9$

$c_b = (0.54,0.4,0)$
$a_b = 0.94$

$c_{red} = 0.4*0 + (1-0.4)*0.94*0.54 = 0.6*0.94*.54 = 0.30$

$c_{green} = 0.4*1 + (1-0.4)*0.94*0.4 = 0.6*0.94*.4 = 0.23$

$c_{blue} = 0.4*1 + (1-0.4)*0.94*0 = .4$

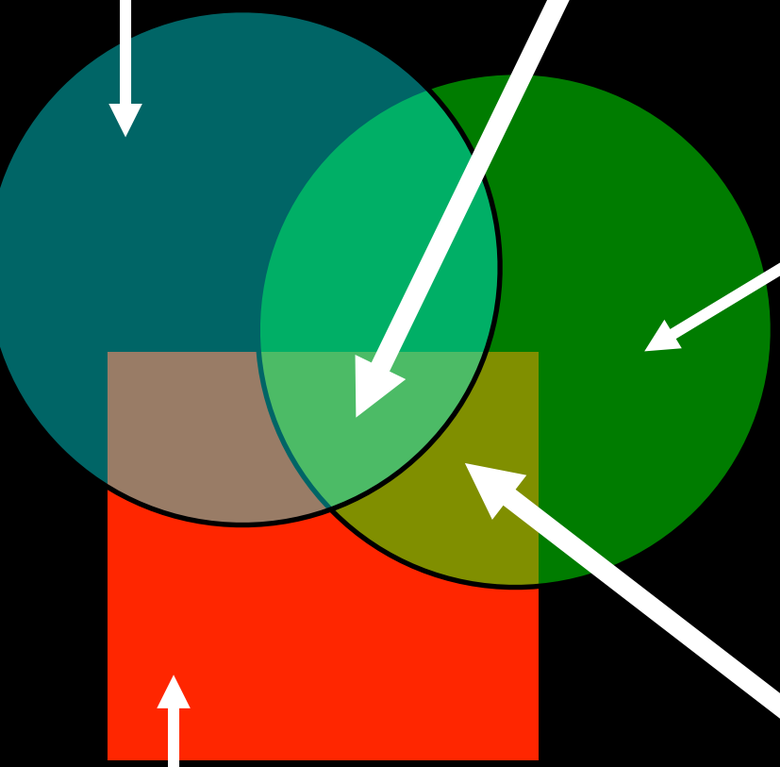
$a = 0.4 + (1 - 0.4)*(0.94) = 0.4 + 0.6*0.94) = .964$

over operator

$$c = a_f * c_f + (1 - a_f) * a_b * c_b$$
$$a = a_f + (1 - a_f) * a_b$$

$c_f = (0,1,1)$
$a_f = 0.4$

$c = (0.3,0.23,0.4)$
$a = 0.964$



$c_1 = (0,1,0)$
$a_1 = 0.4$

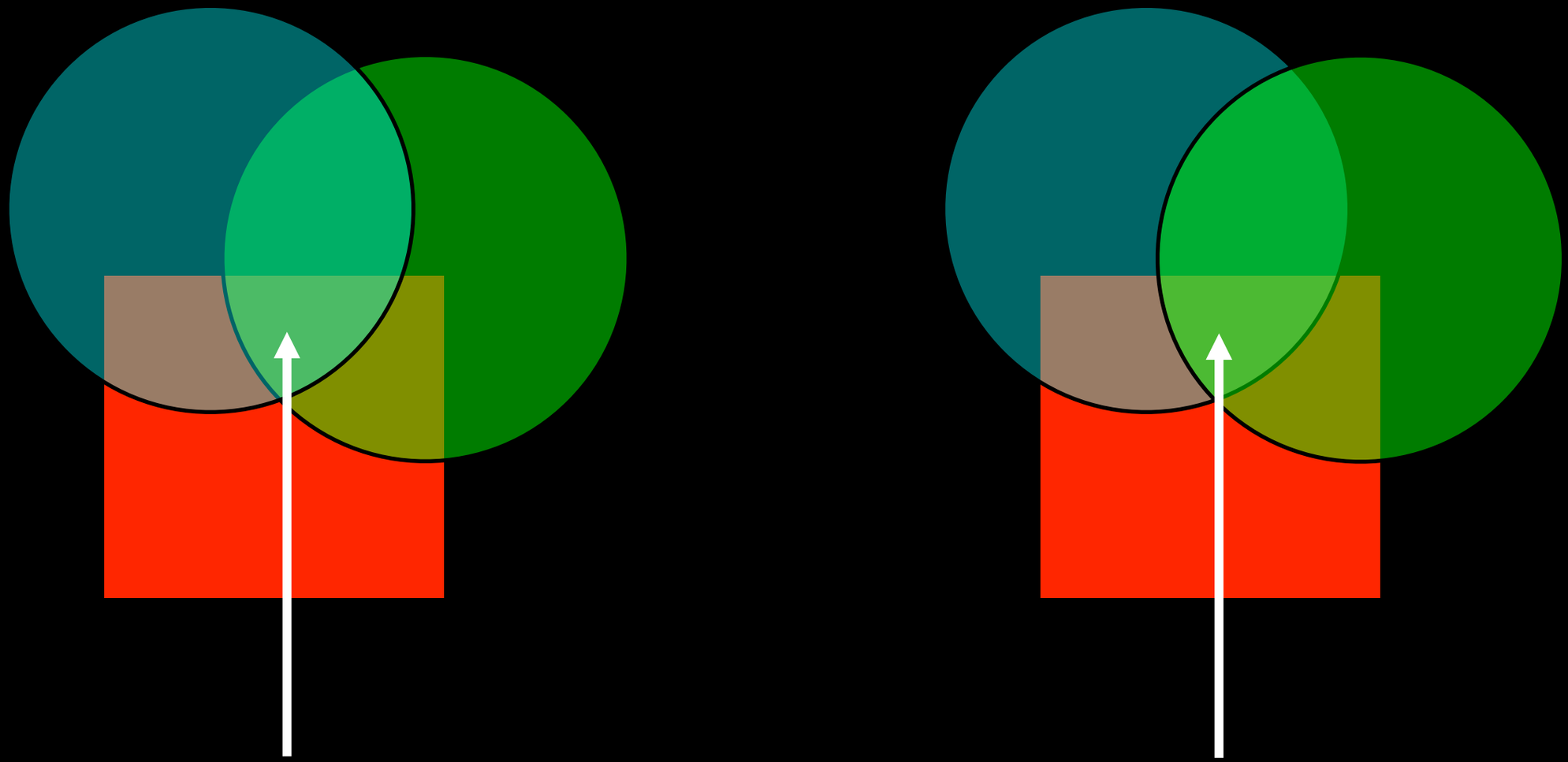
$c_b = (0.54,0.4,0)$
$a_b = 0.94$

$c_2 = (1,0,0)$
$a_2 = 0.9$

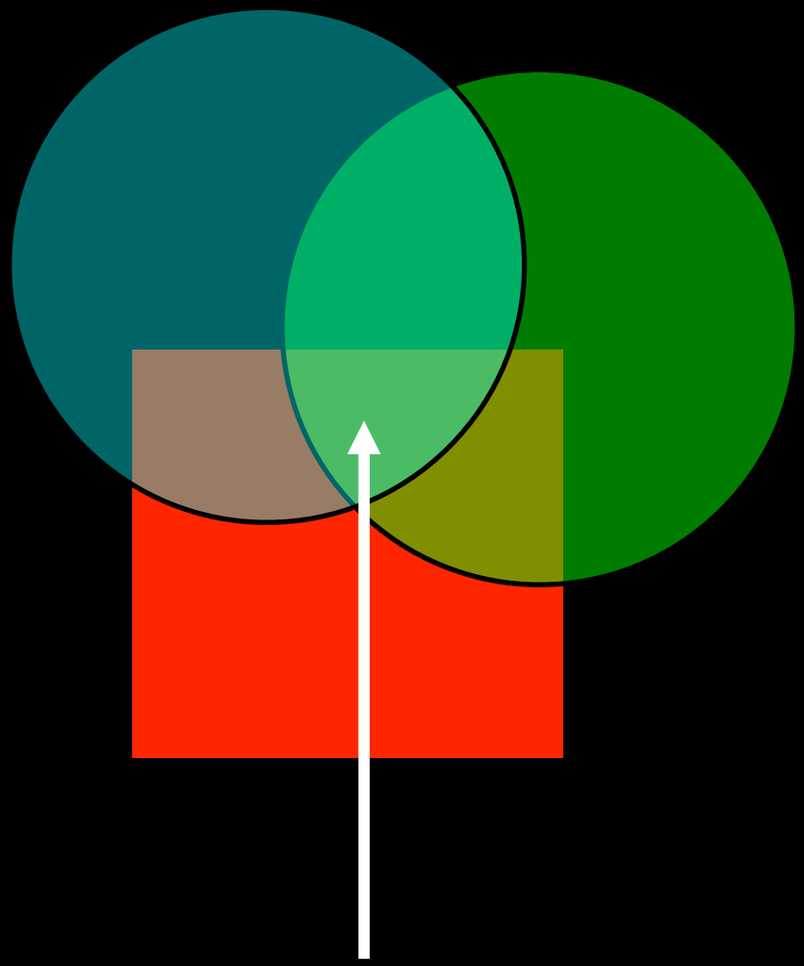
over operator

$$c = a_f * c_f + (1 - a_f) * a_b * c_b$$
$$a = a_f + (1 - a_f) * a_b$$

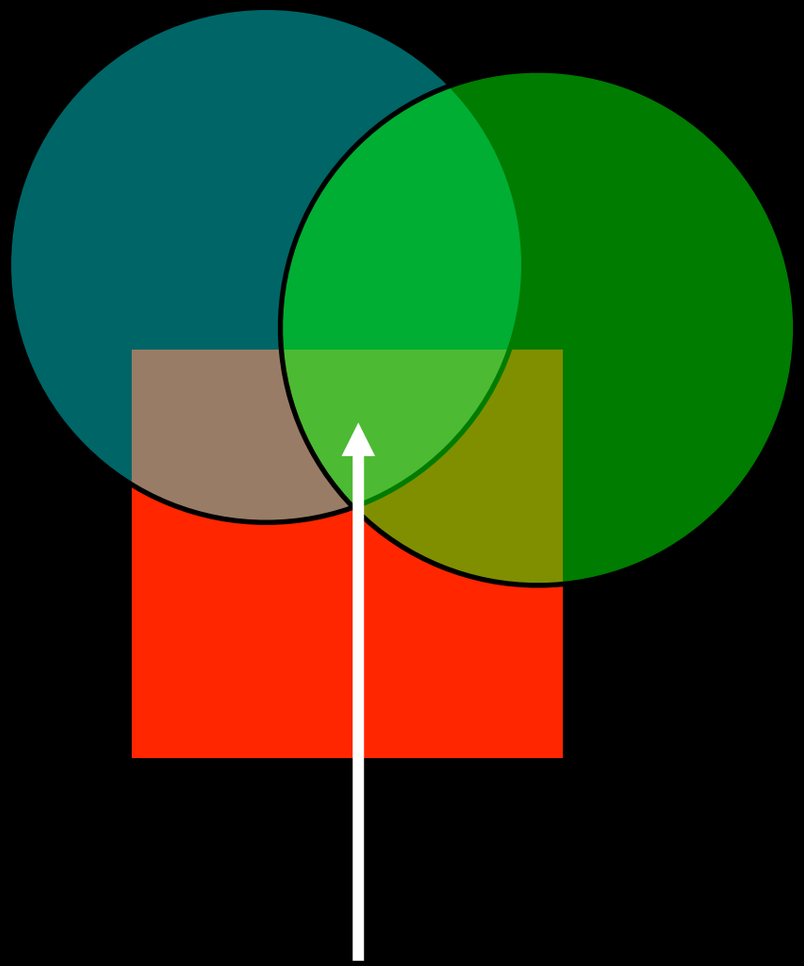
Order Matters!



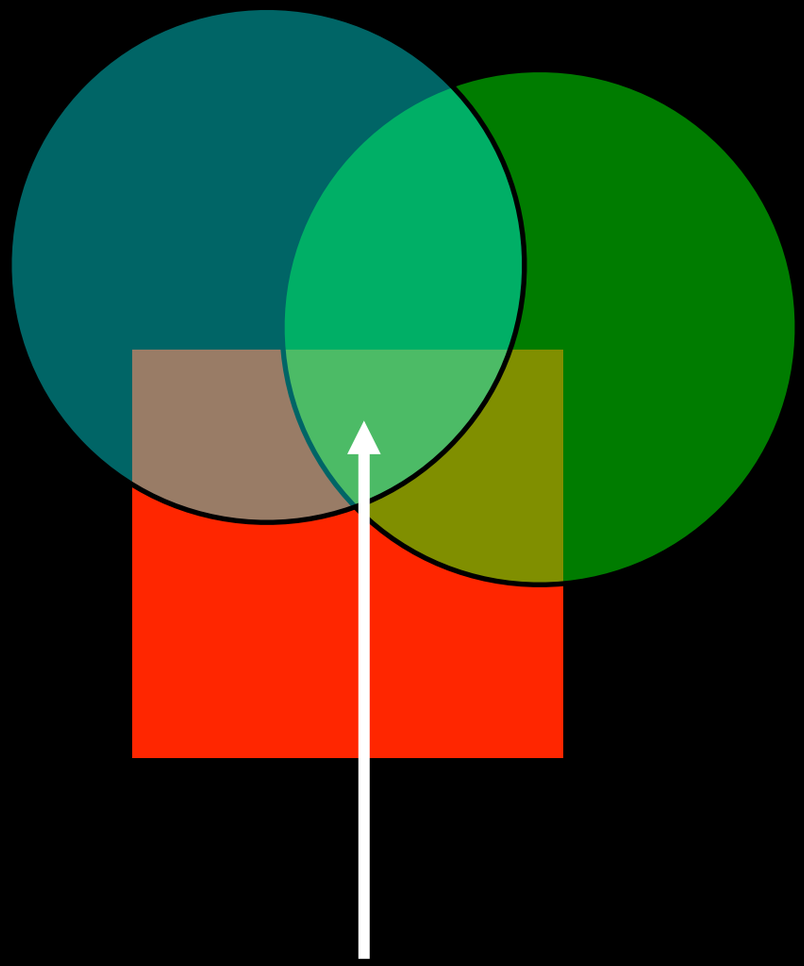
Order Matters!



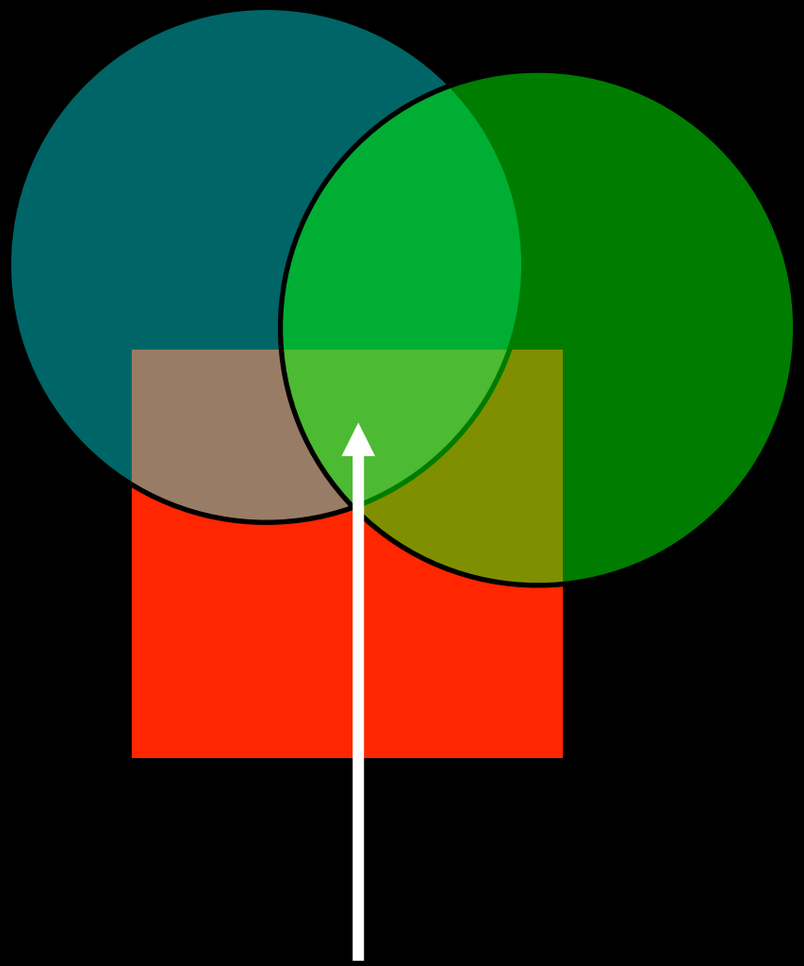
c = (0.3,0.23,**0.4**)
a = 0.964



Order Matters!

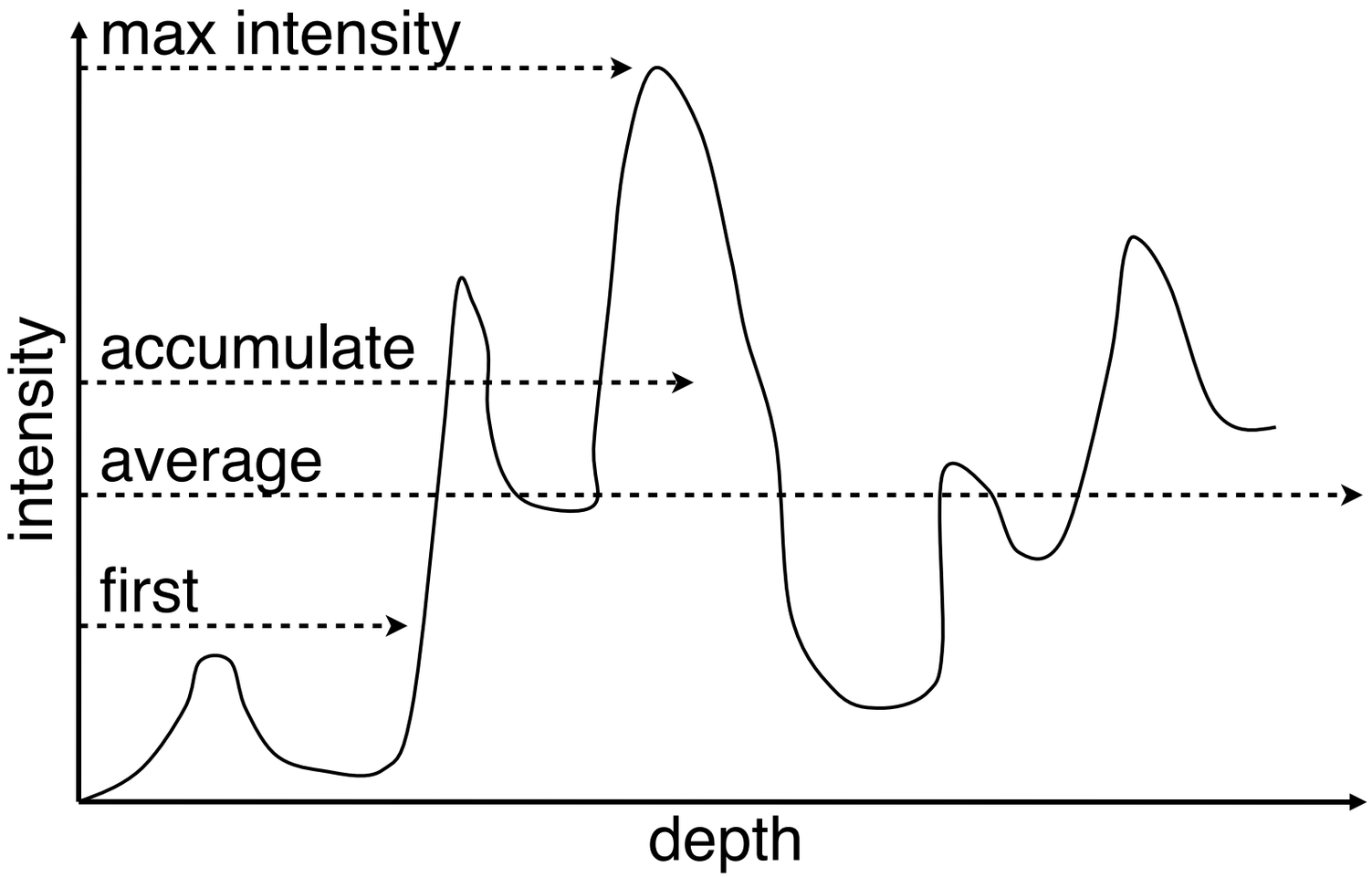


c = (0.3,0.23,**0.4**)
a = 0.964

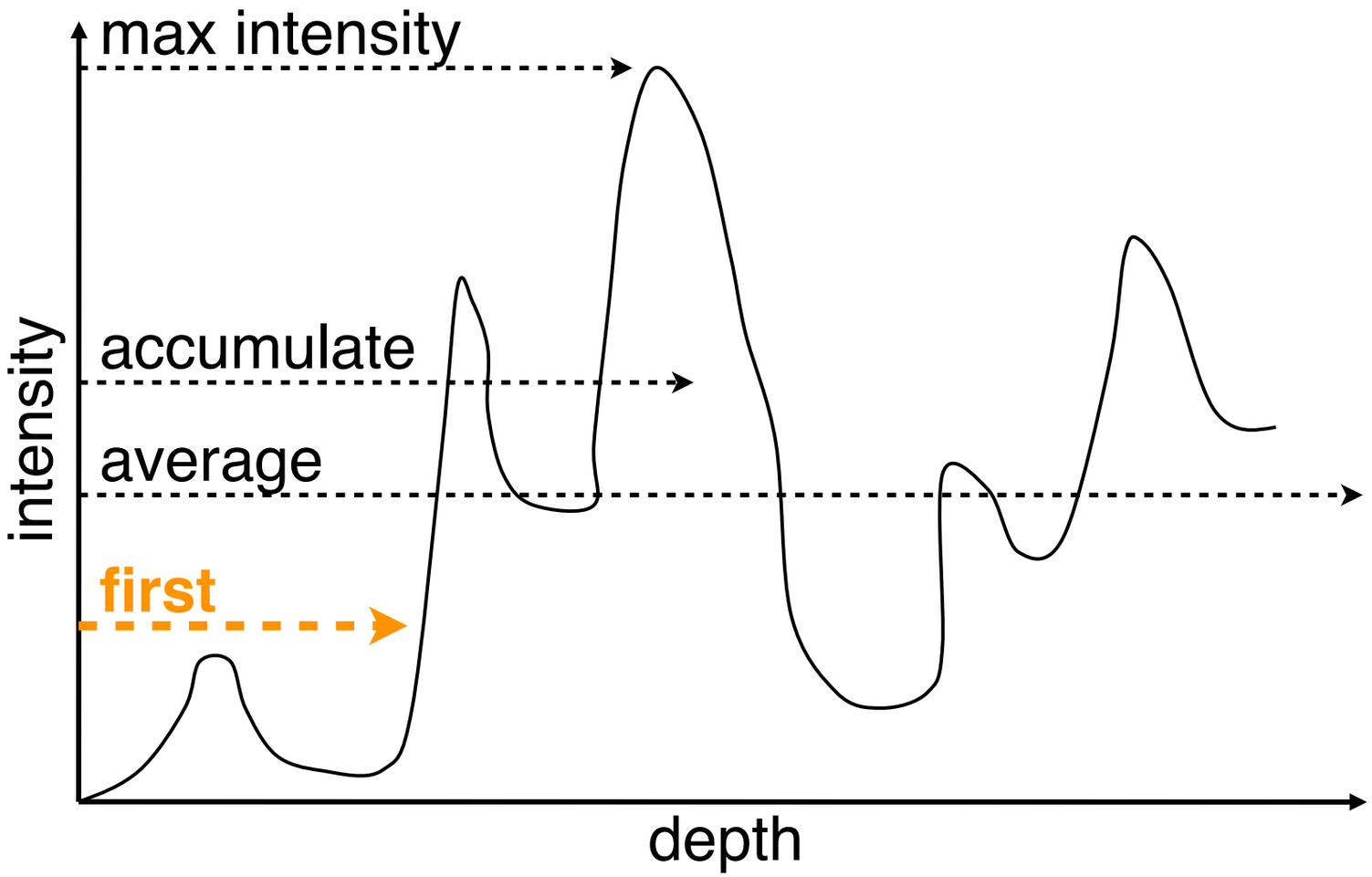


c = (0.3,0.23,**0.23**)
a = 0.964

Pixel Compositing Schemes

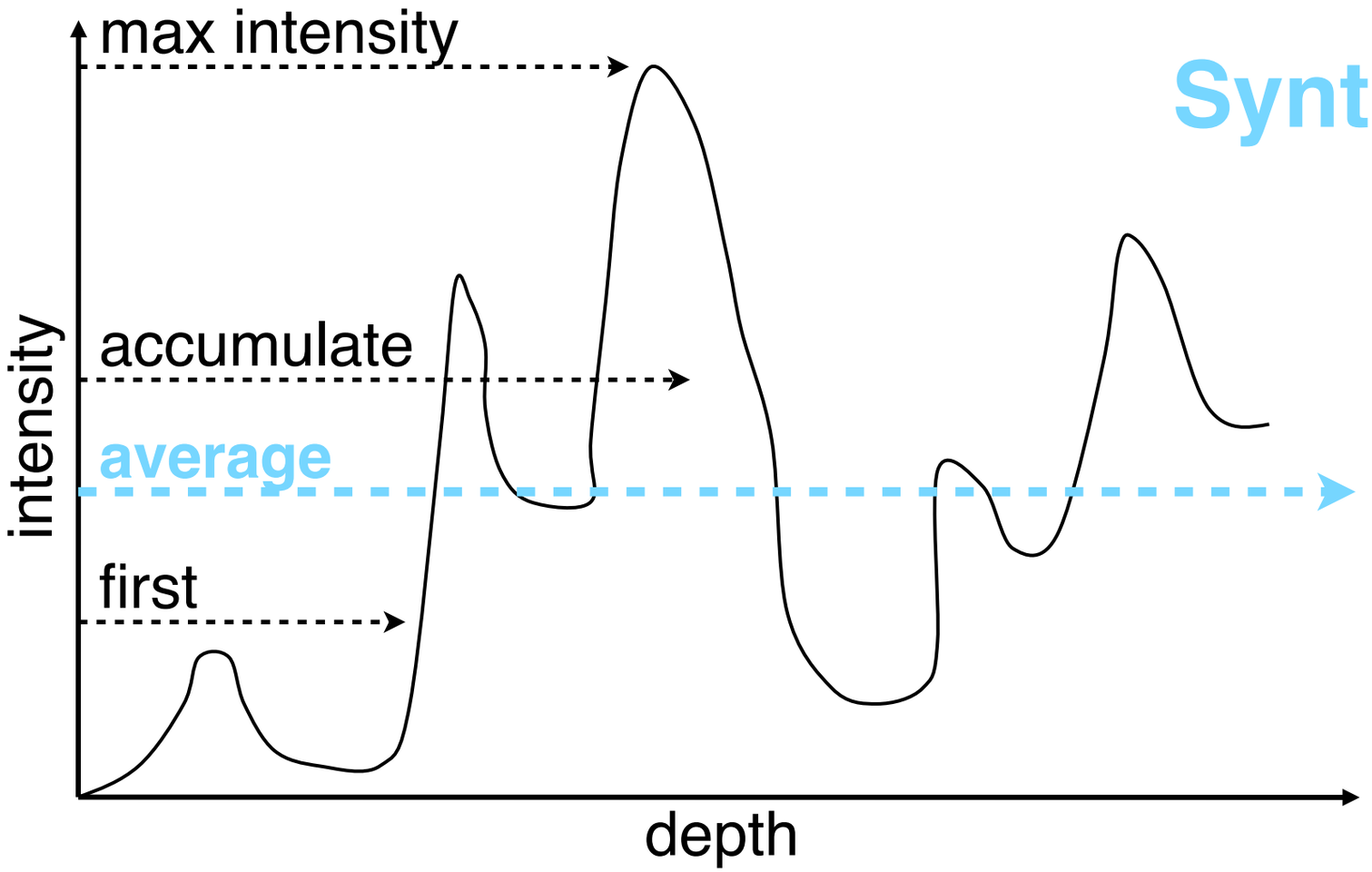


Pixel Compositing Schemes



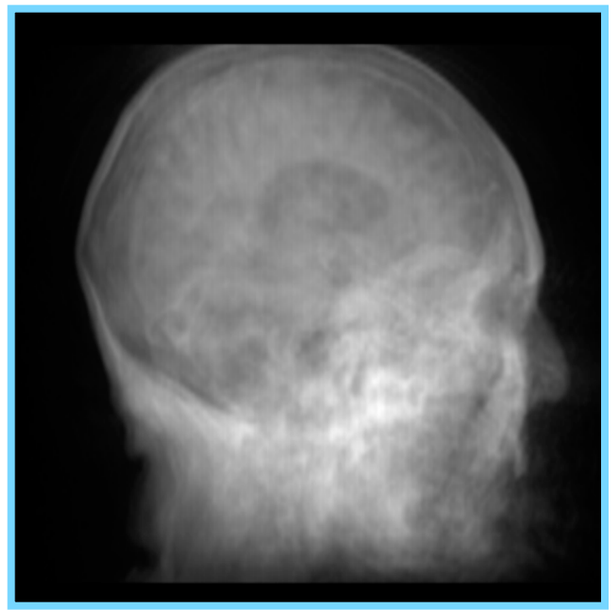
Exact Isosurface

Pixel Compositing Schemes



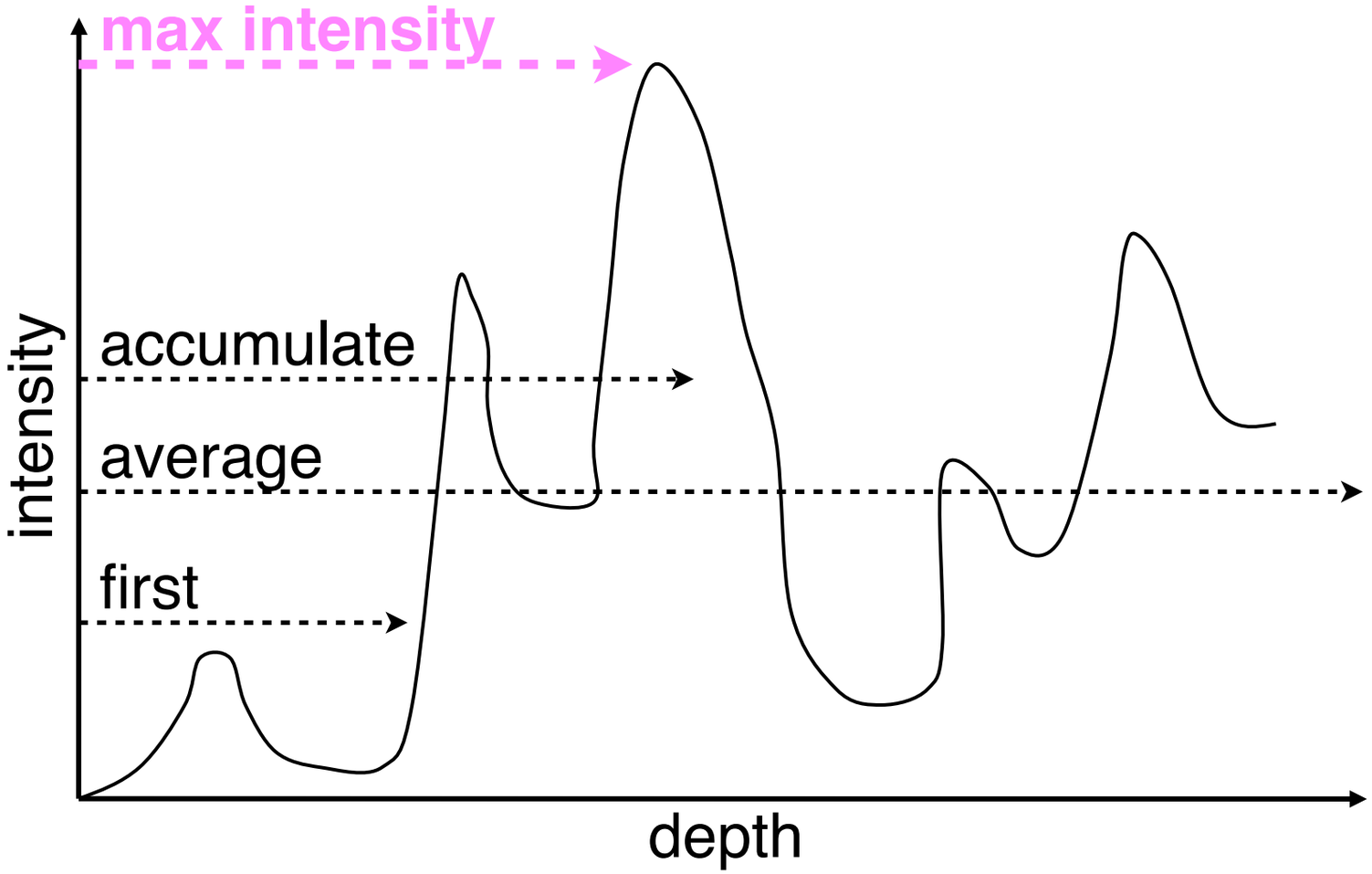
Similar to X-rays

Synthetic Reprojection

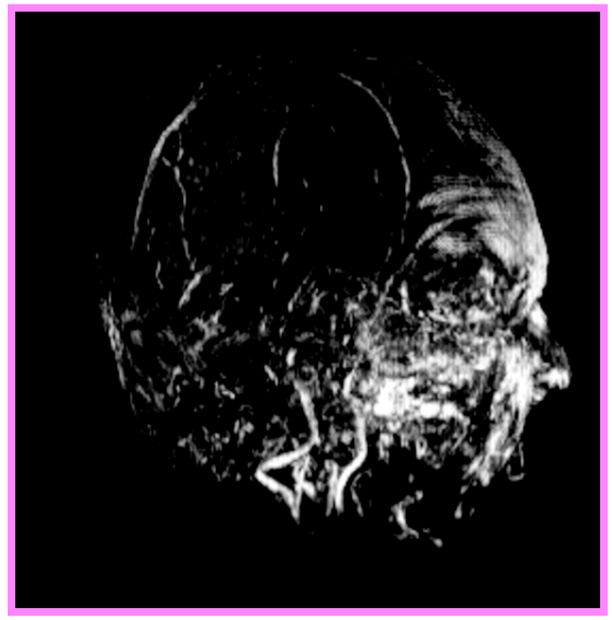


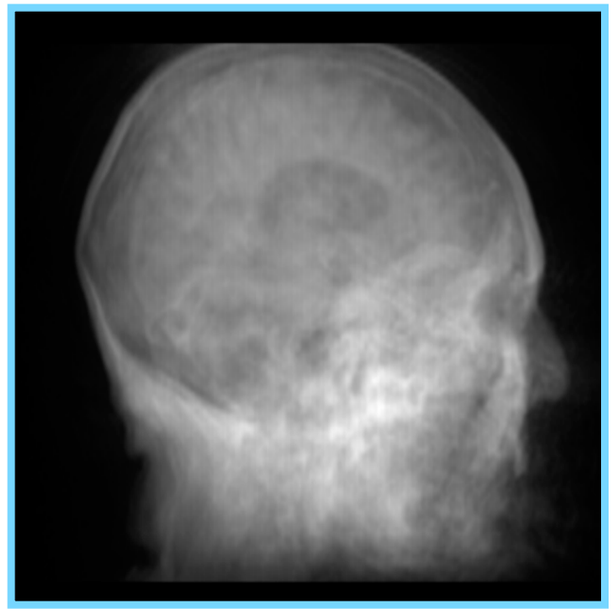
maximum intensity projection (MIP)

Pixel Compositing Schemes

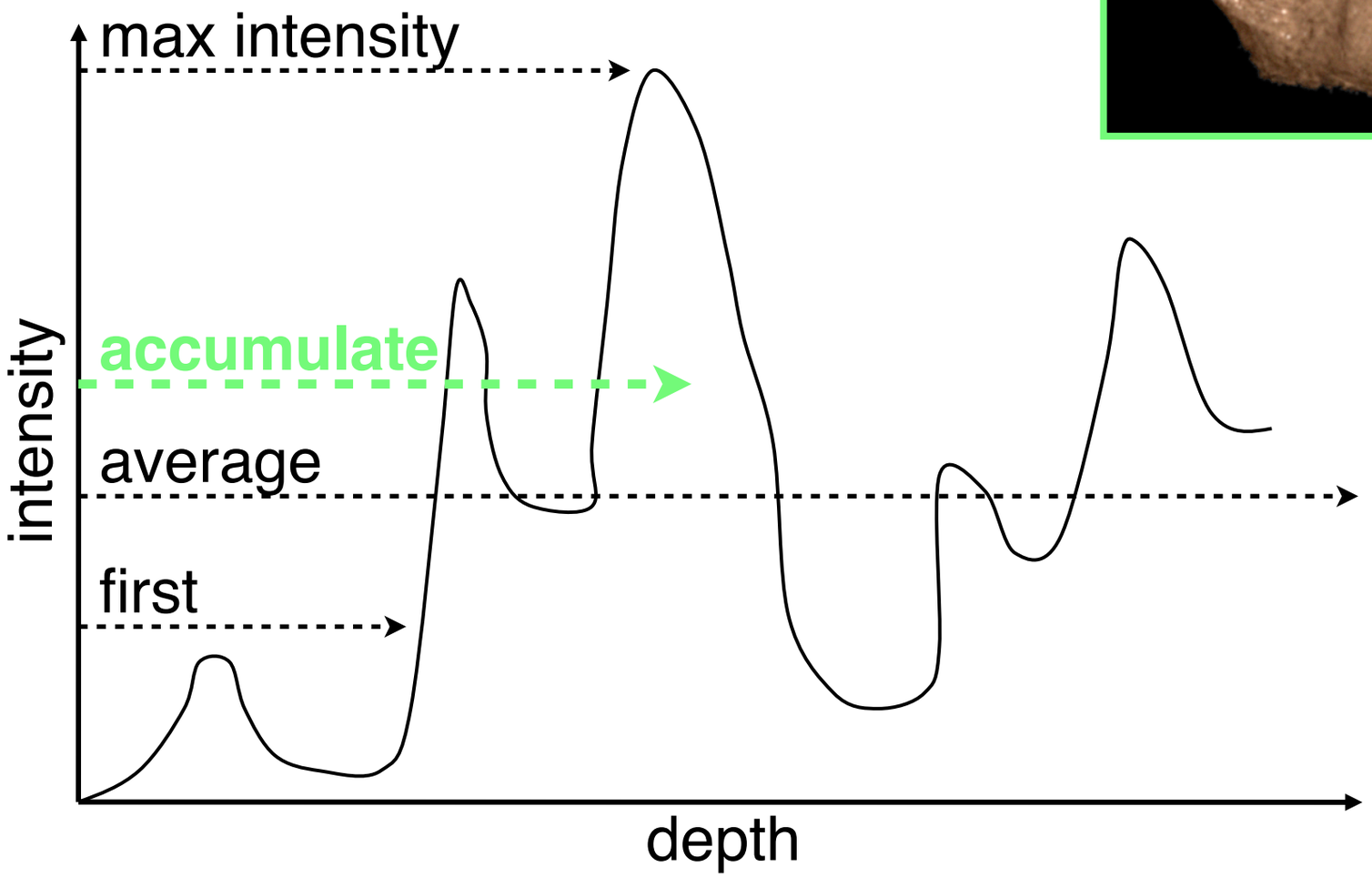


Used in PET and Magnetic Resonance Angiograms

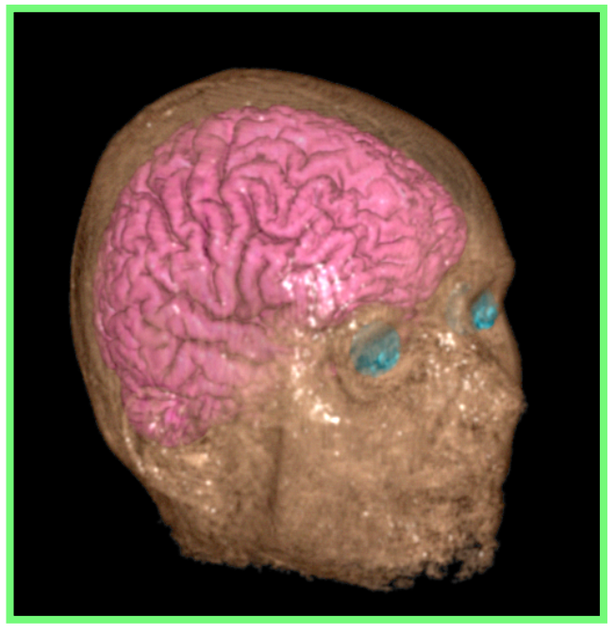


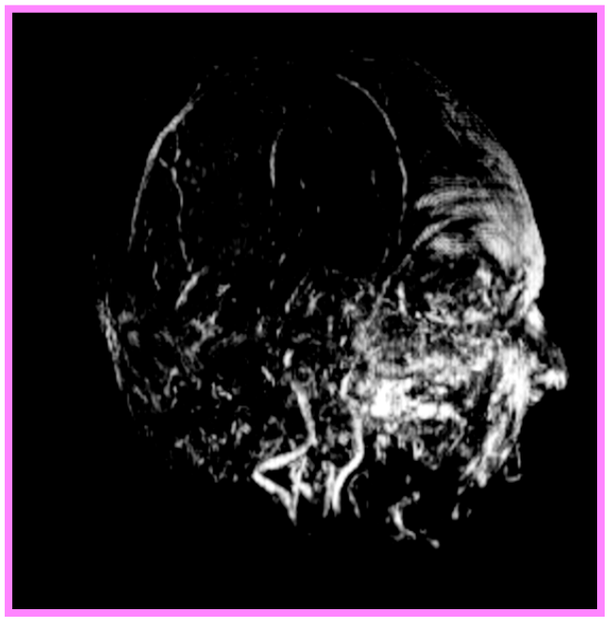


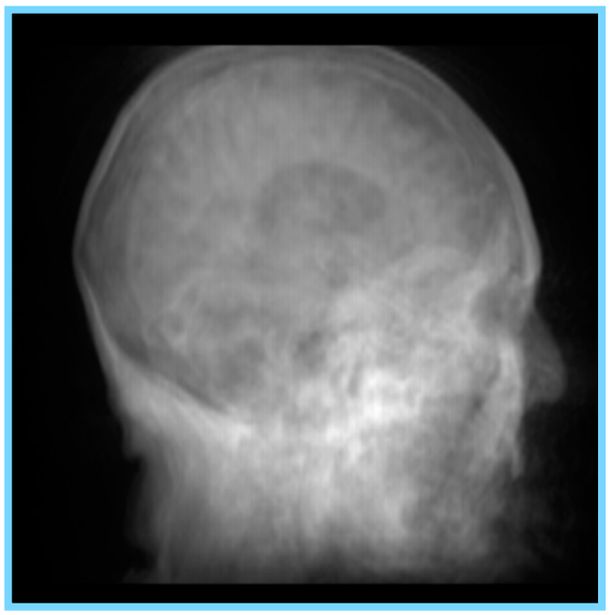
Pixel Compositing Schemes



color to distinguish structures
opacity to show inside



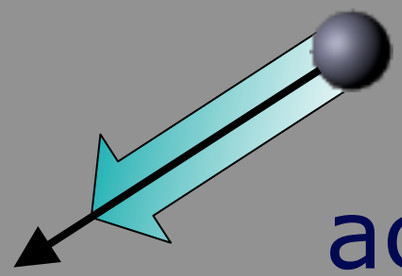




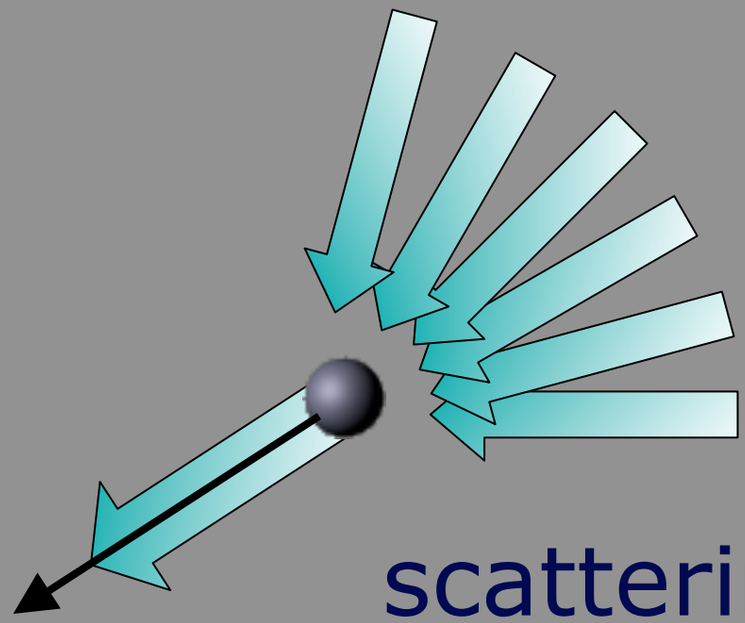
Modeling Light

Optical Models (1)

Emission



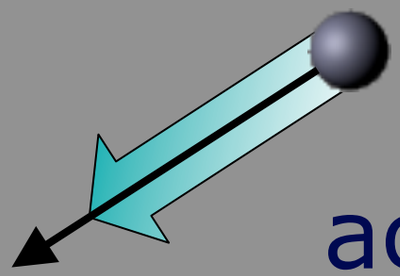
active



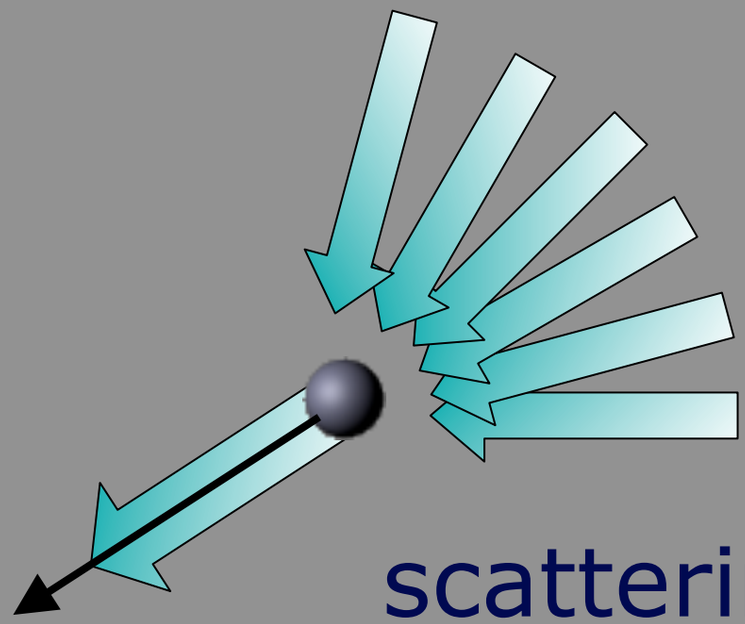
scattering

Optical Models (1)

Emission

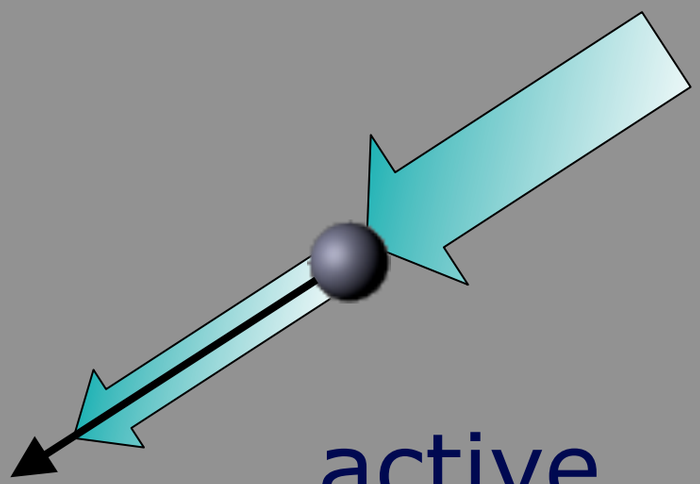


active

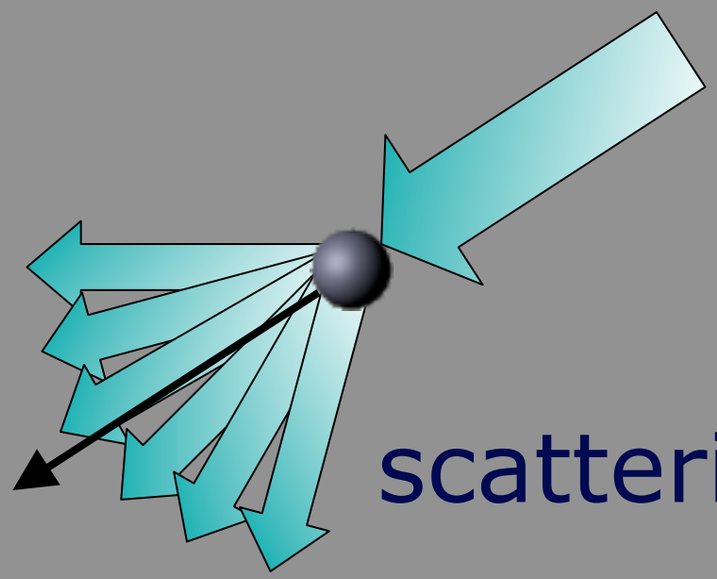


scattering

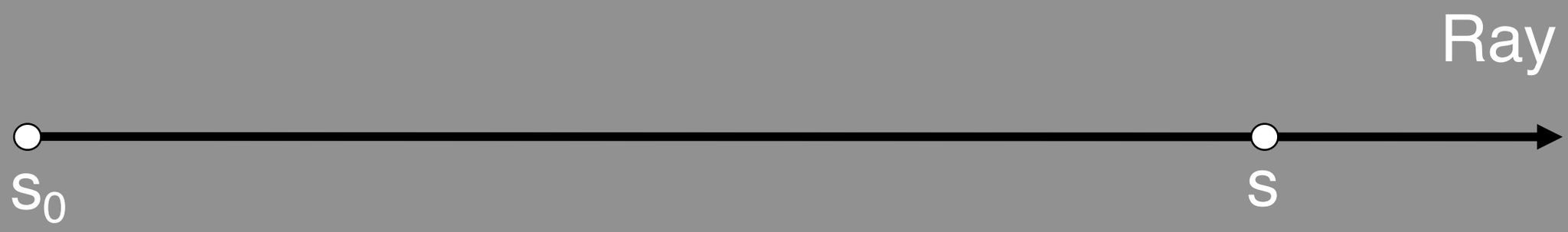
Absorption



active



scattering

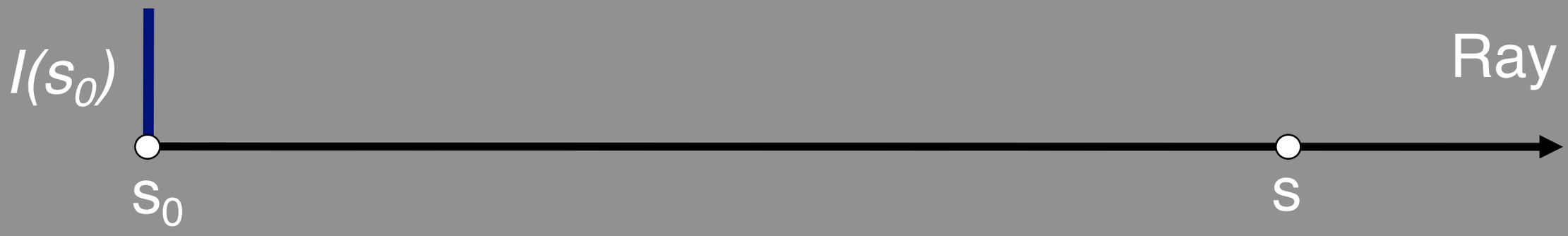


What must be integrated?

physically correct: emission and absorption of light

$$I(s) \;=$$

Sabella 1988



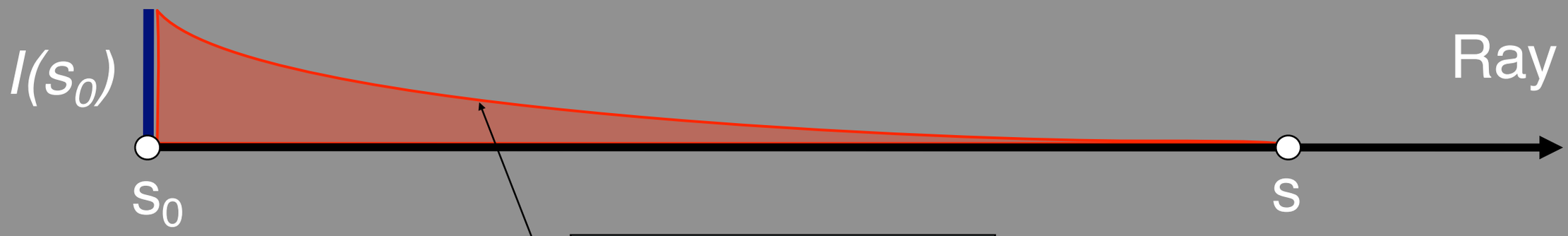
What must be integrated?

Initial intensity (Emission) at s_0

physically correct: emission and absorption of light

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

Sabella 1988



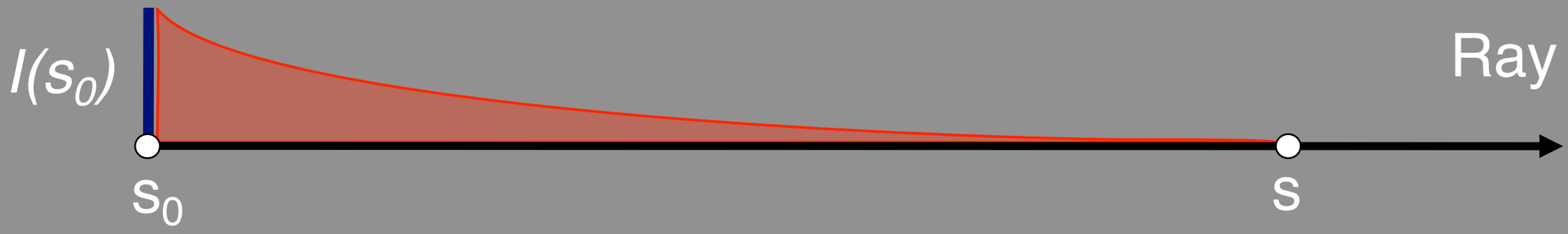
Absorption along the distance s_0 - s

What must be integrated?

physically correct: emission and absorption of light

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

Sabella 1988



What must be integrated?

physically correct: emission and abso

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

Optical Depth τ

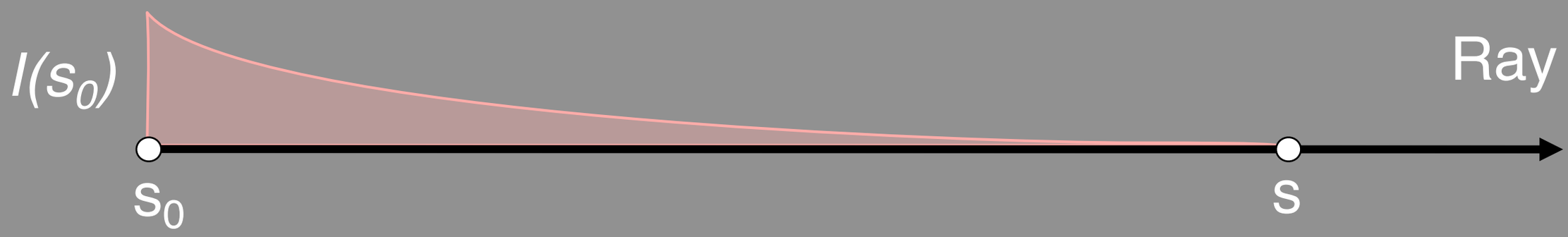
Absorption κ

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

Sabella 1988

What must be integrated?

physically correct: emission and absorption of light

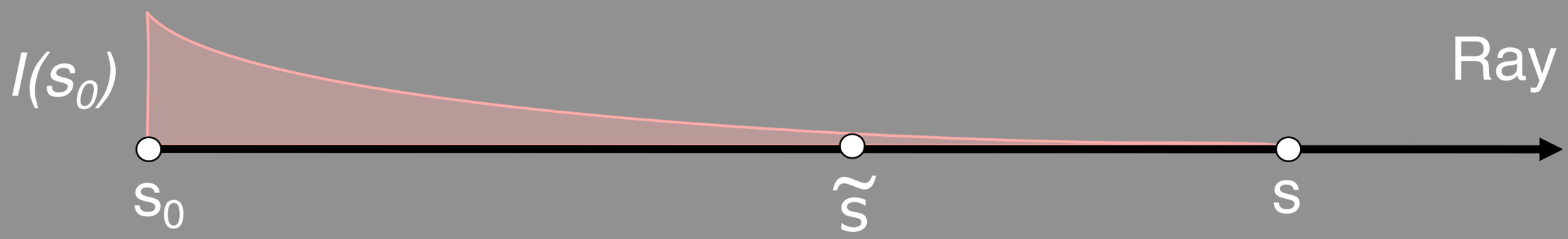


$$I(s) \;=\; I(s_0)\, e^{-\tau(s_0,s)}$$

Sabella 1988

What must be integrated?

physically correct: emission and absorption of light

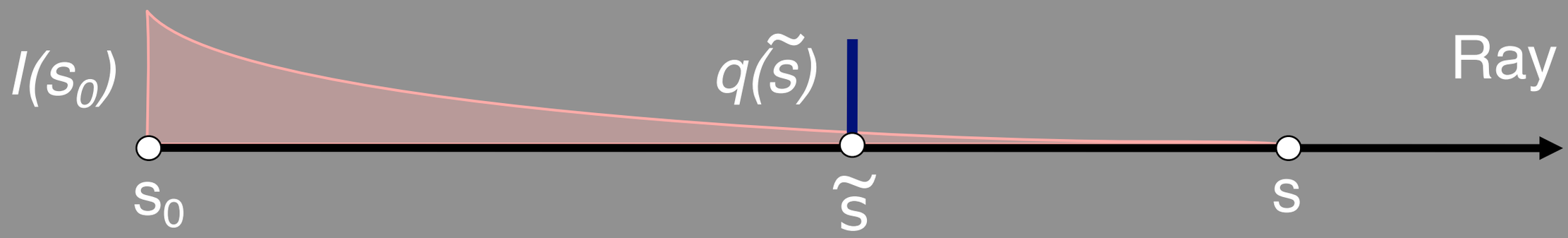


$$I(s) \;=\; I(s_0)\, e^{-\tau(s_0,s)}$$

Sabella 1988

What must be integrated?

physically correct: emission and absorption of light



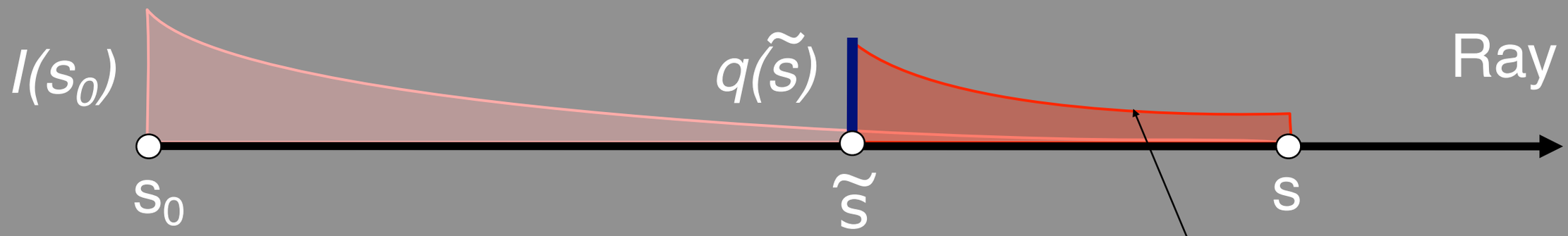
Active Emission
at s̃

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

Sabella 1988

What must be integrated?

physically correct: emission and absorption of light



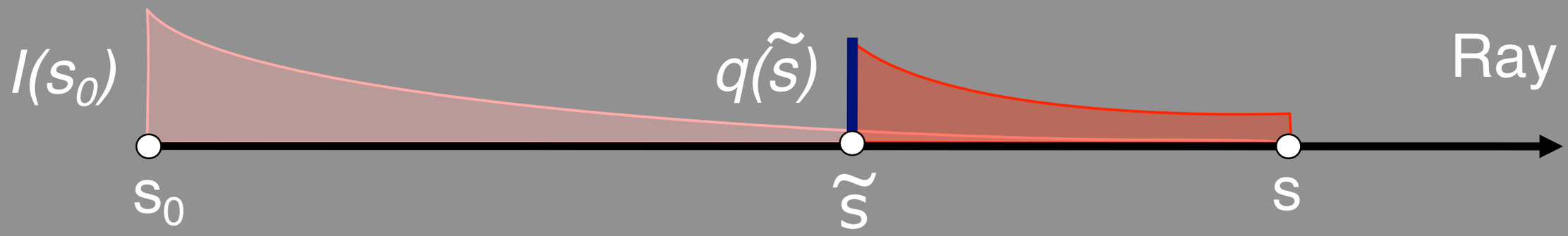
Absorption along the distance $\tilde{s}$ - s

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

Sabella 1988

What must be integrated?

physically correct: emission and absorption of light

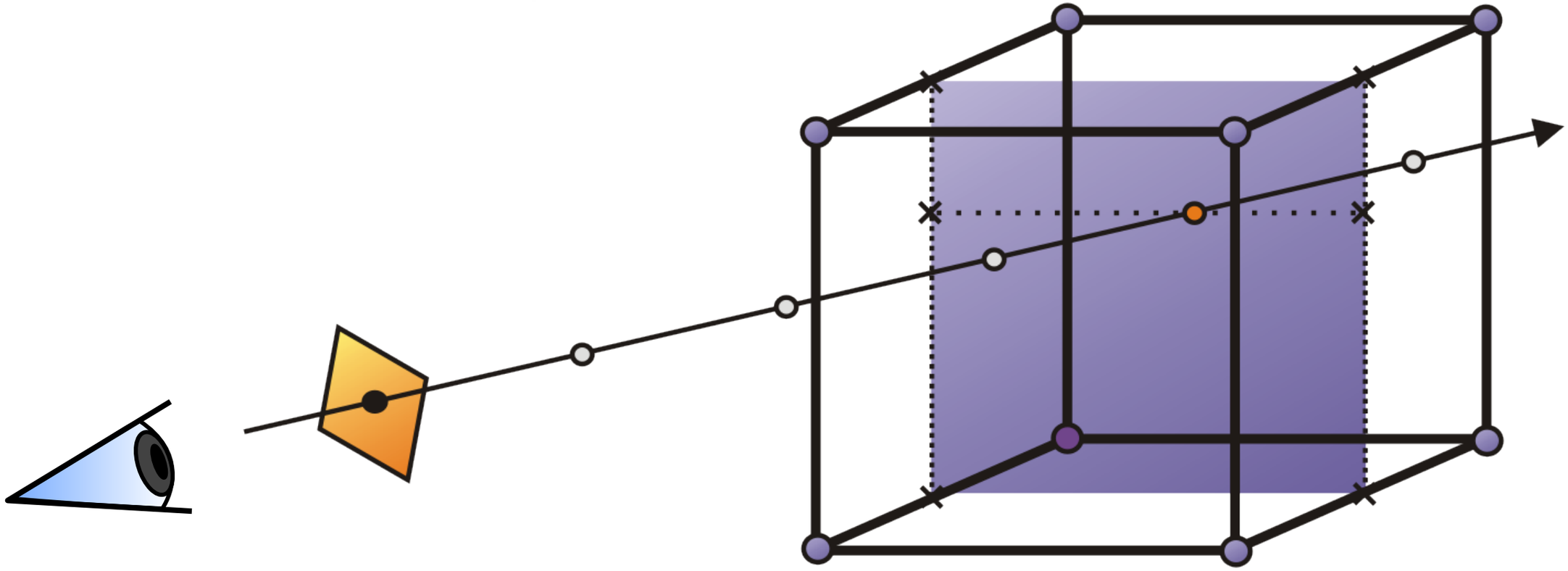


$$I(s) = I(s_0)\, e^{-\tau(s_0,s)} + \int_{s_0}^{s} q(\tilde{s})\, e^{-\tau(\tilde{s},s)}\, d\tilde{s}$$

Sabella 1988

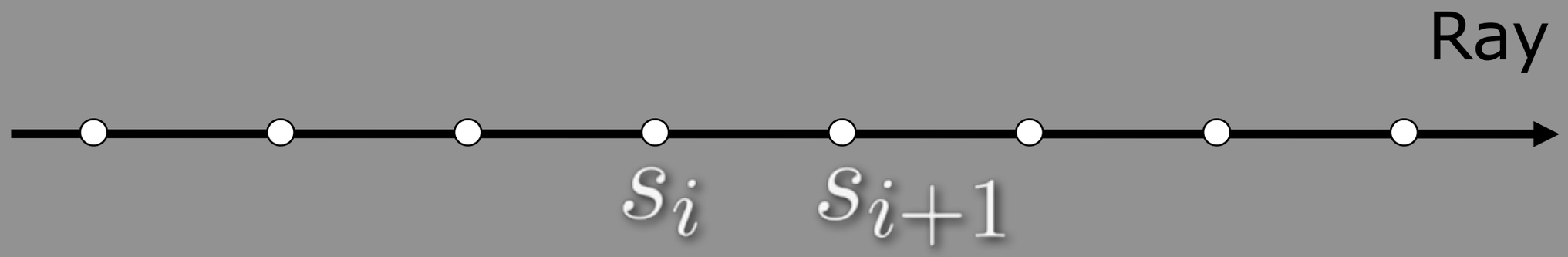
- We perform a **numerical approximation** of volume rendering integral

- Idea: resample volume at equispaced intervals along the ray
  - Use trilinear interpolation



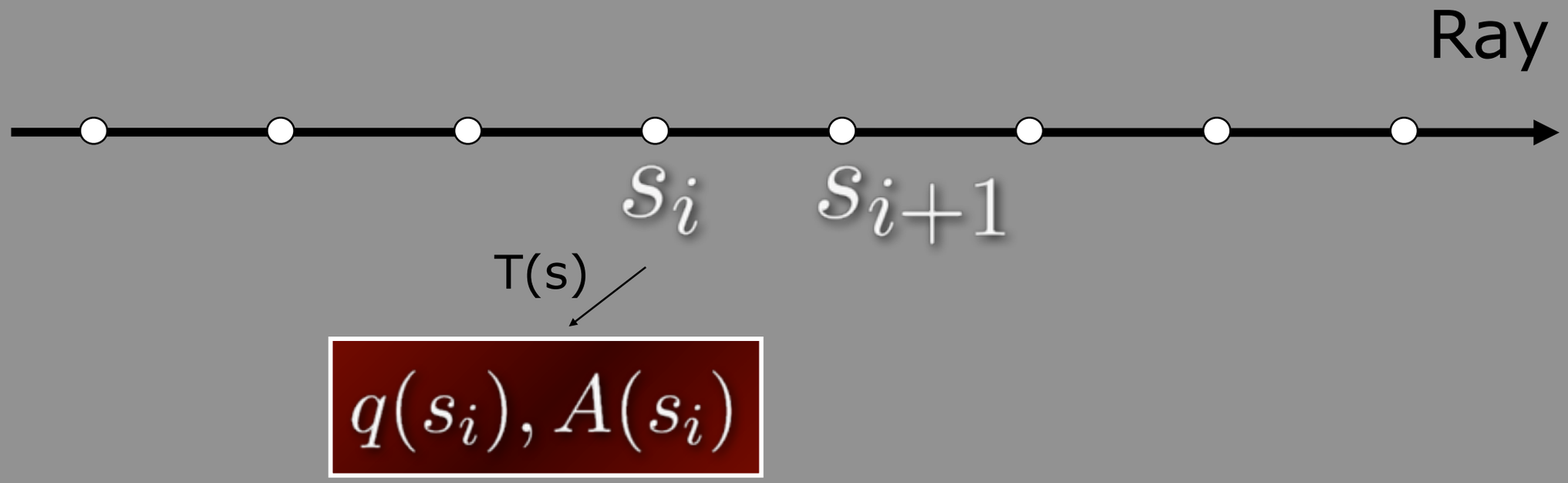
Discrete Solution

Resample the scalar field at discrete locations along the viewing ray:



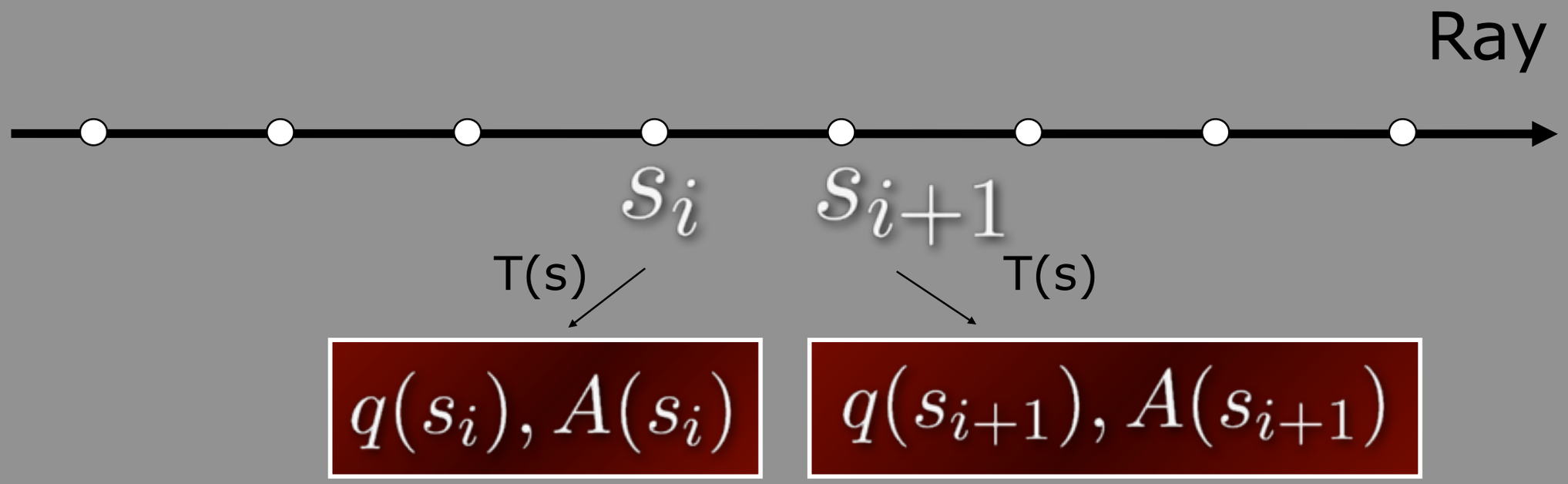
Discrete Solution

Resample the scalar field at discrete locations along the viewing ray:



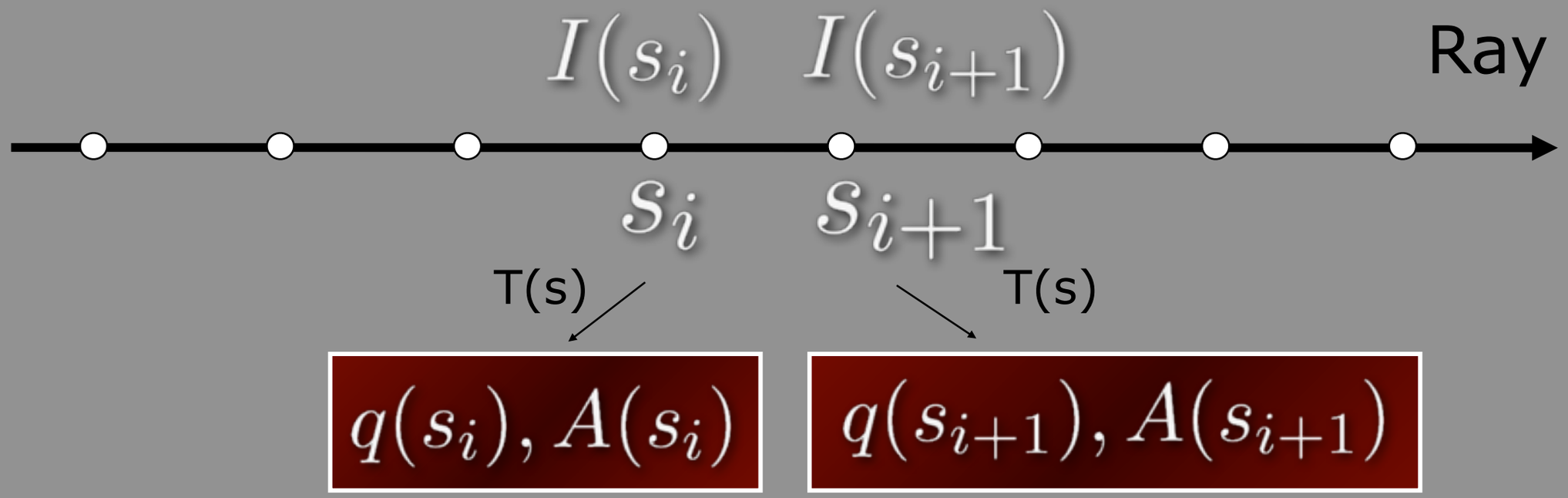
Discrete Solution

Resample the scalar field at discrete locations along the viewing ray:



Discrete Solution

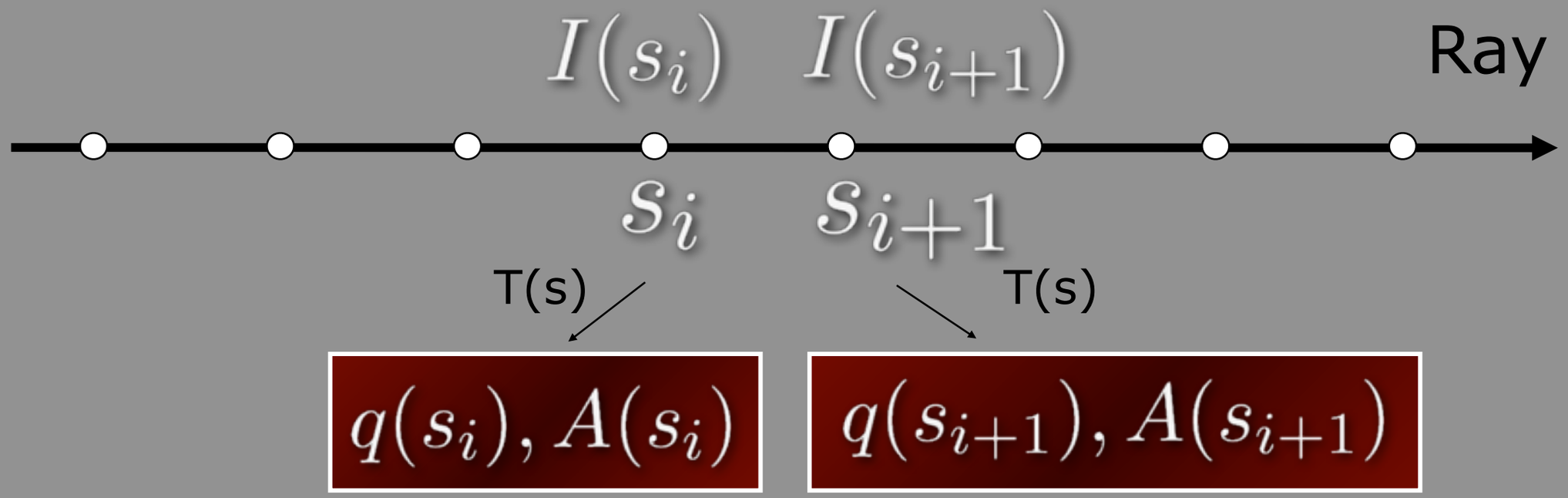
Resample the scalar field at discrete locations along the viewing ray:



$$I(s_{i+1}) \quad =$$

Discrete Solution

Resample the scalar field at discrete locations along the viewing ray:

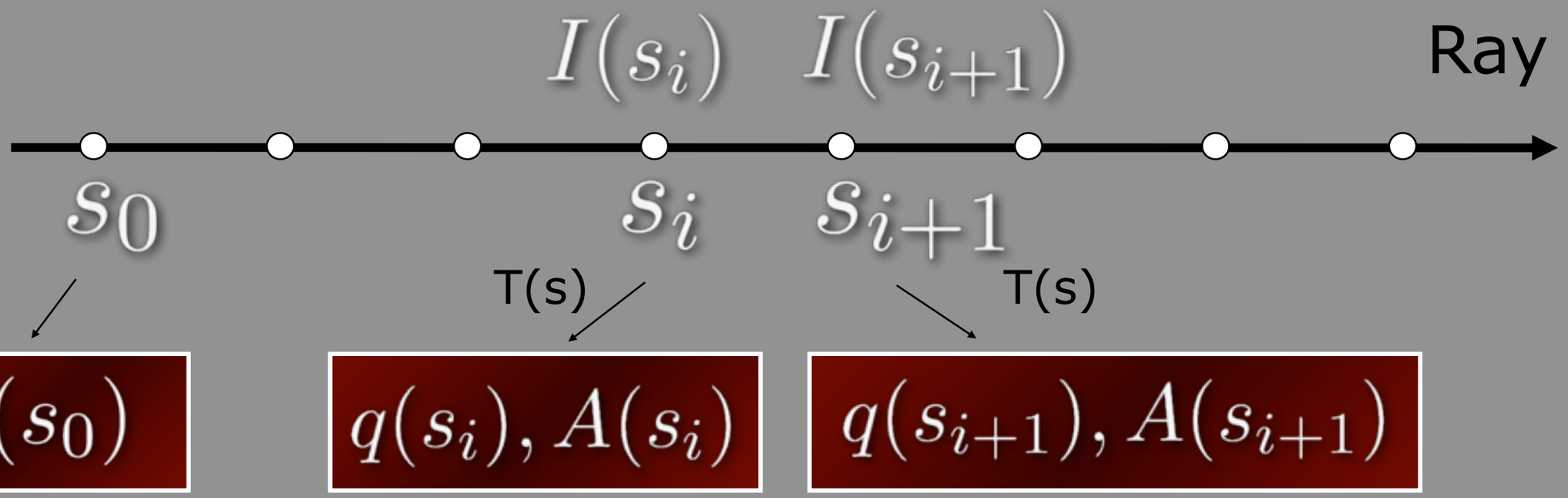


Back-to-front Compositing with $\alpha = A(s_i)$

$$I(s_{i+1}) = \alpha \cdot q(s_{i+1}) + (1 - \alpha) I(s_i)$$

Discrete Solution

Resample the scalar field at discrete locations along the viewing ray:



Back-to-front Compositing with $\alpha = A(s_i)$

$$I(s_{i+1}) = \alpha \cdot q(s_{i+1}) + (1 - \alpha) I(s_i)$$

Alpha Blending

- **Numerical approximation of volume rendering integral**
- **Back-to-front compositing**

$$\mathbf{C}'_{\mathbf{i}} = \mathbf{C}_{\mathbf{i}} + (1 - \mathbf{A}_{\mathbf{i}})\mathbf{C}'_{\mathbf{i+1}}$$

- **New composited color is color at current location blended with previous composited color**
- **Opacity-weighted colors!**

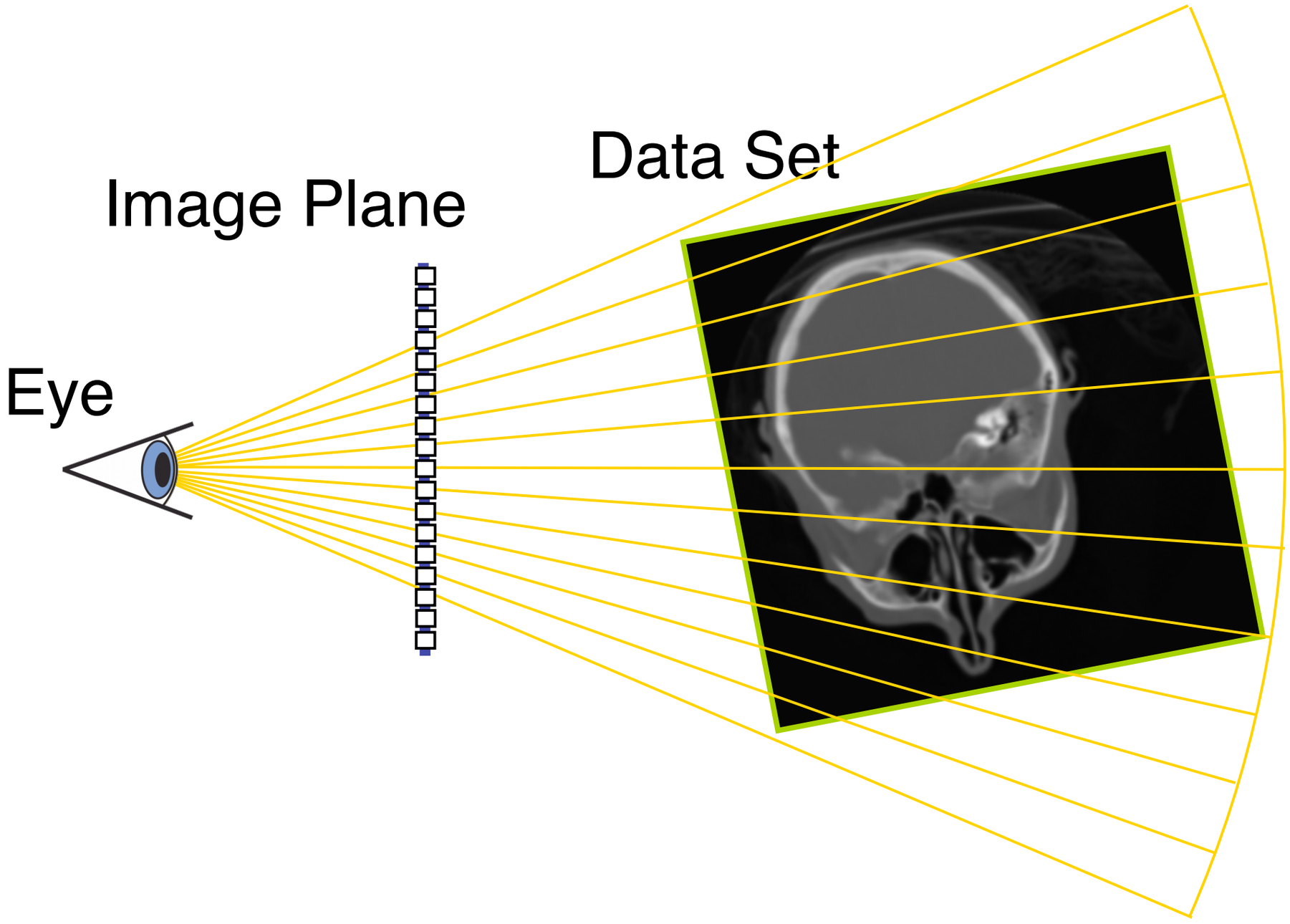
Ray-Casting Highlights

- Advantages
  - Simple algorithm
  - Inherently parallel
  - Can add features (like a ray-tracer)

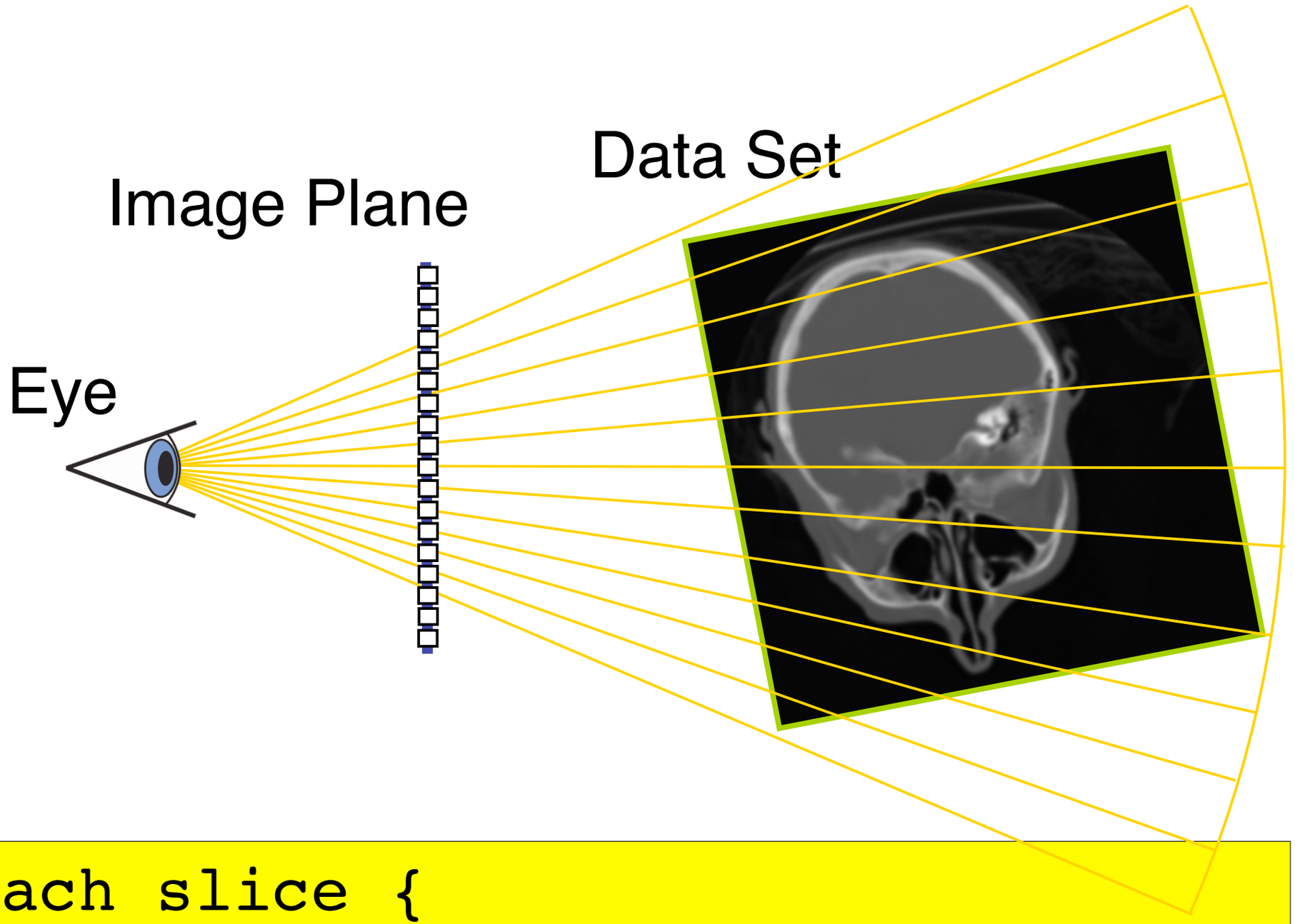
- Disadvantages
  - Slow (lots of rays, lots of samples)
  - Must sample densely
  - Requires data set in memory

Object-Ordered Approaches

Object order approach

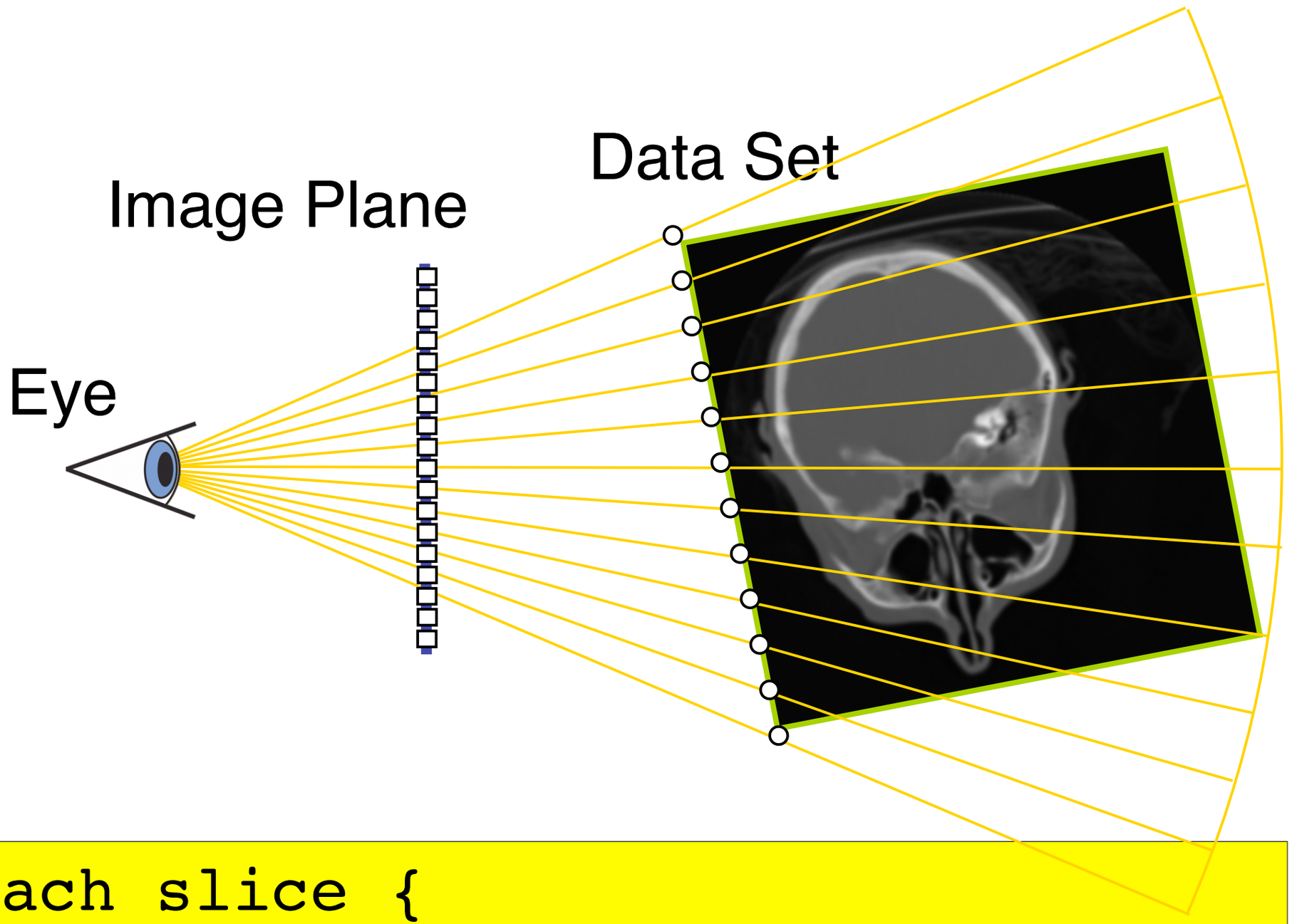


Object order approach



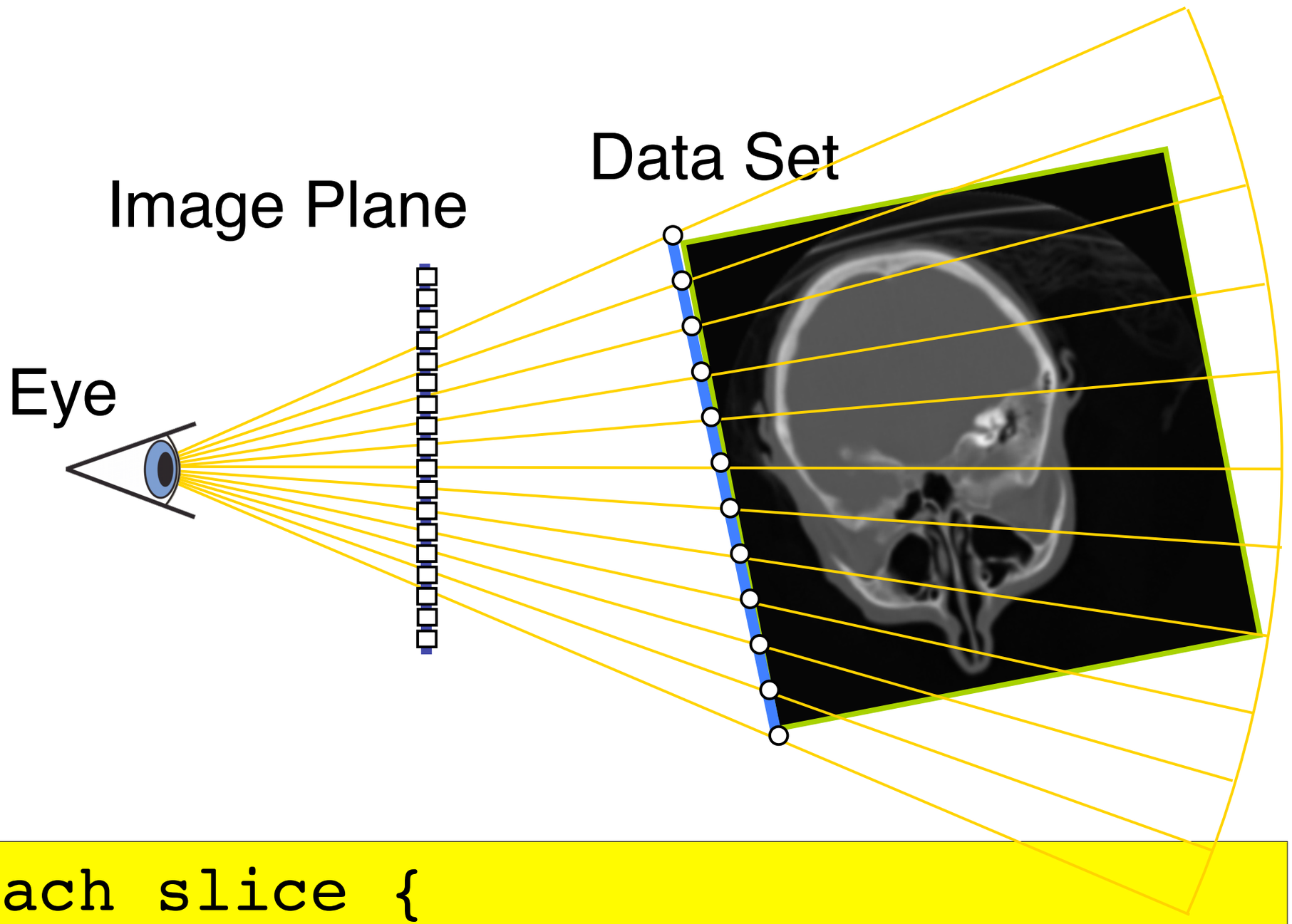

```
For each slice {
   calculate contribution to the image
}
```

Object order approach



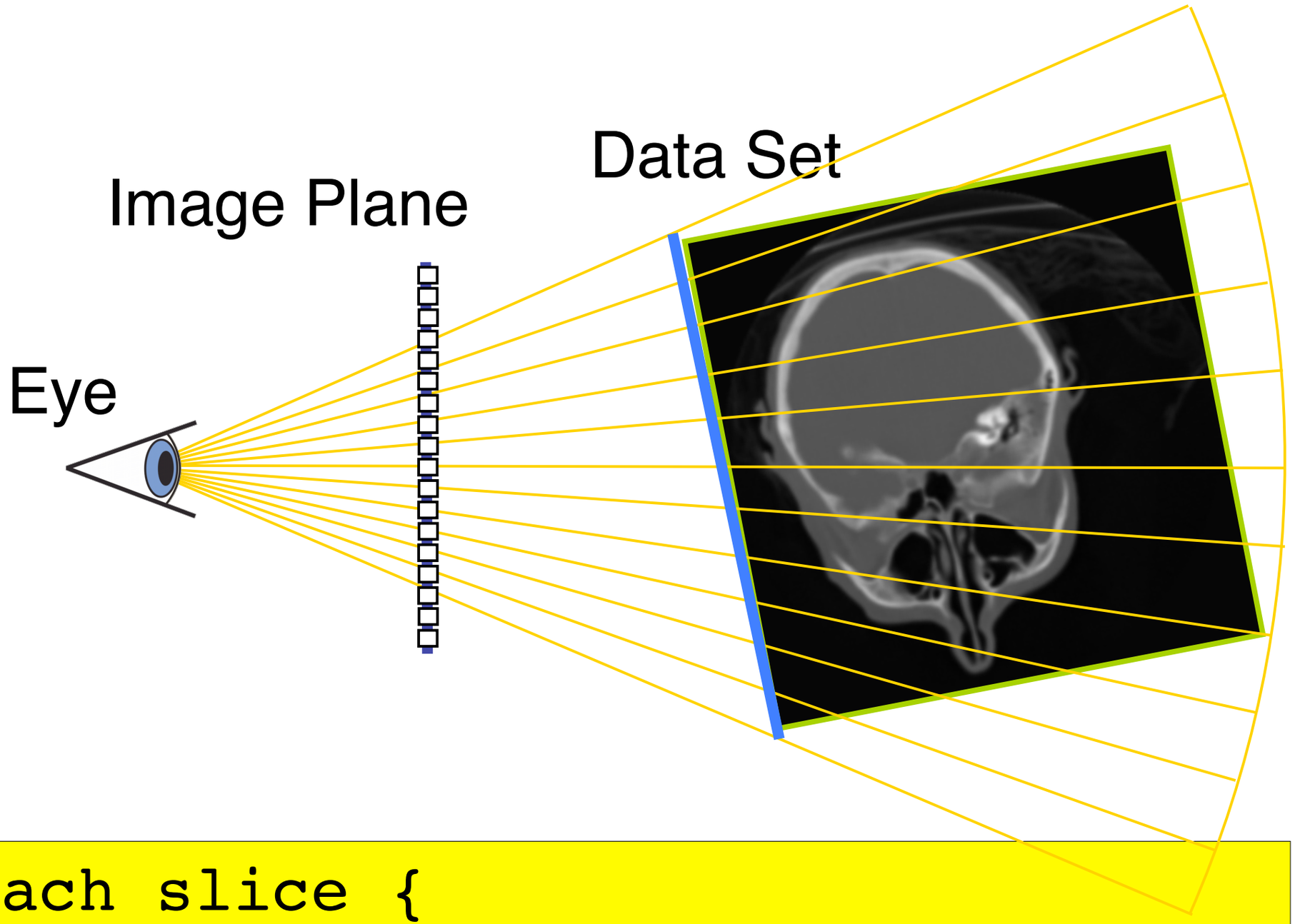

```
For each slice {
   calculate contribution to the image
}
```

Object order approach



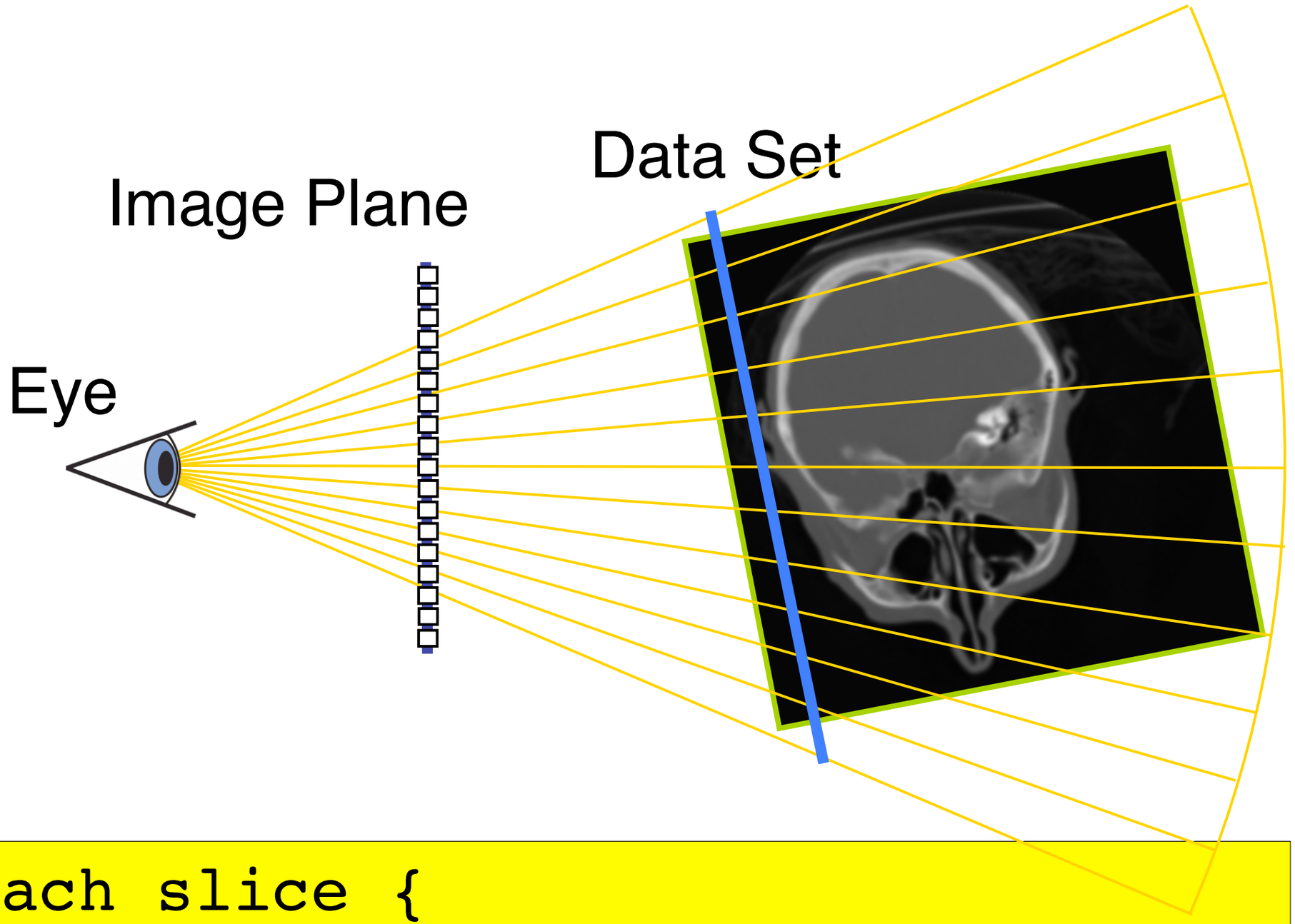

```
For each slice {
   calculate contribution to the image
}
```

Object order approach



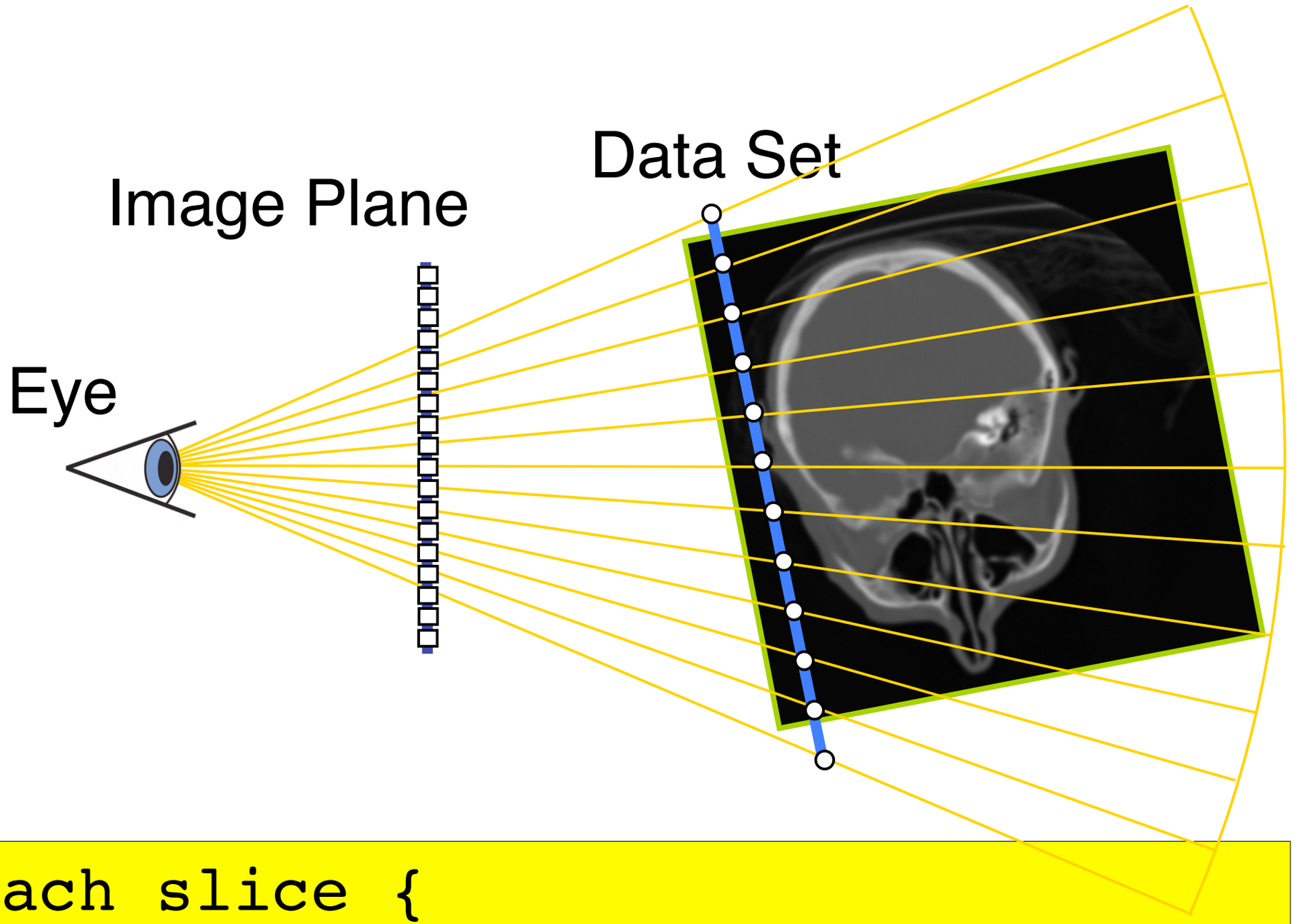

```
For each slice {
   calculate contribution to the image
}
```

Object order approach



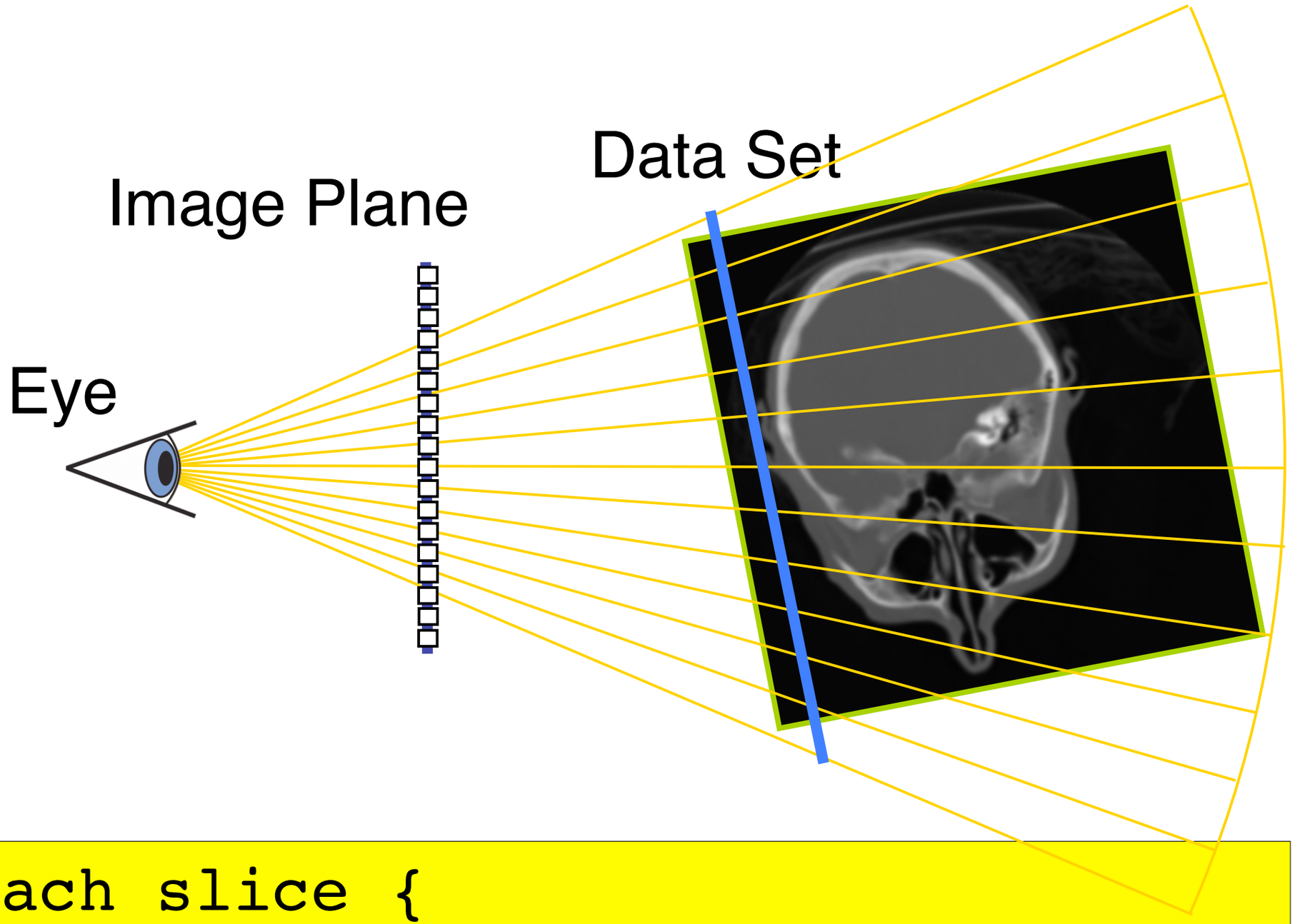

```
For each slice {
   calculate contribution to the image
}
```

Object order approach



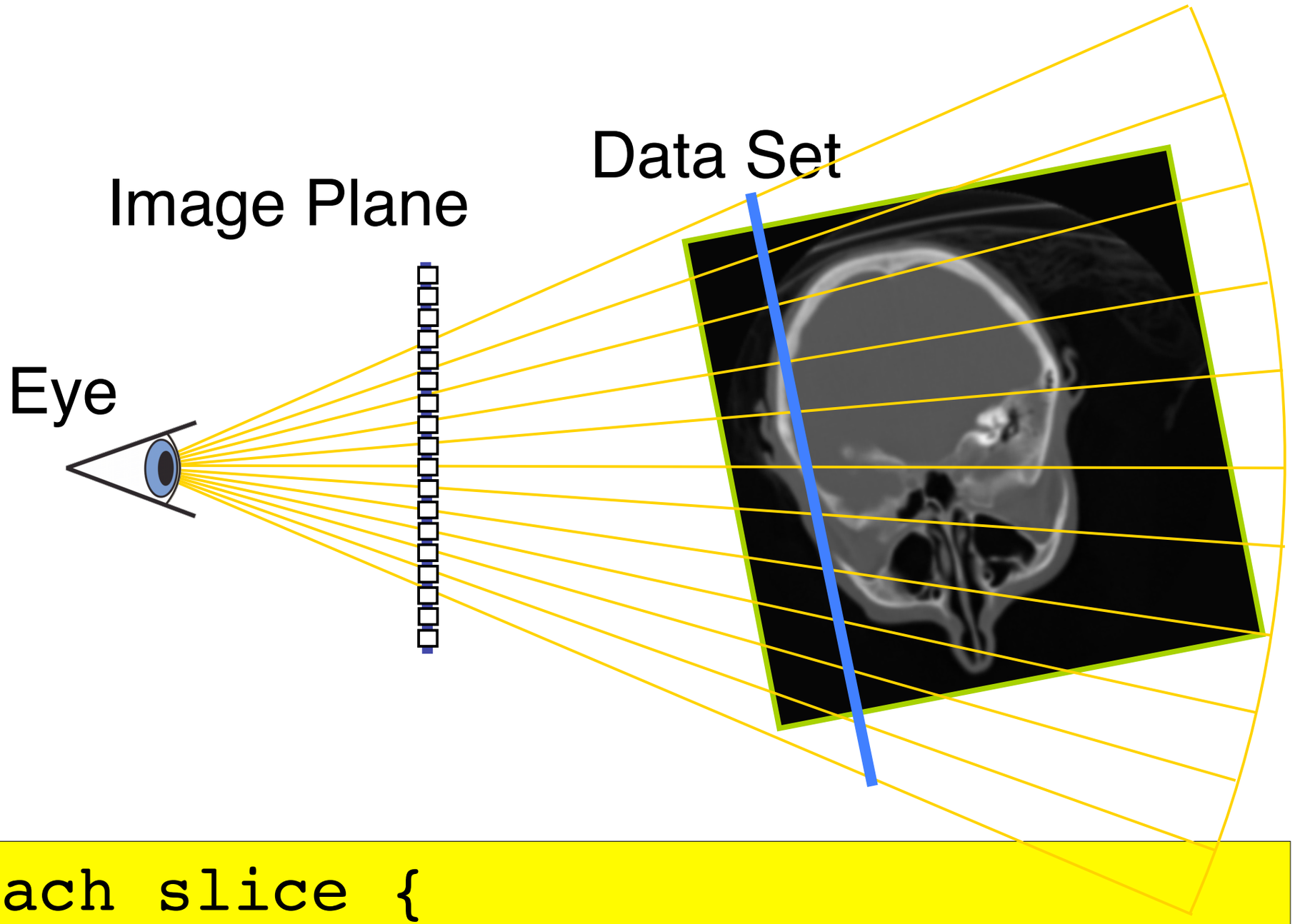

```
For each slice {
   calculate contribution to the image
}
```

Object order approach



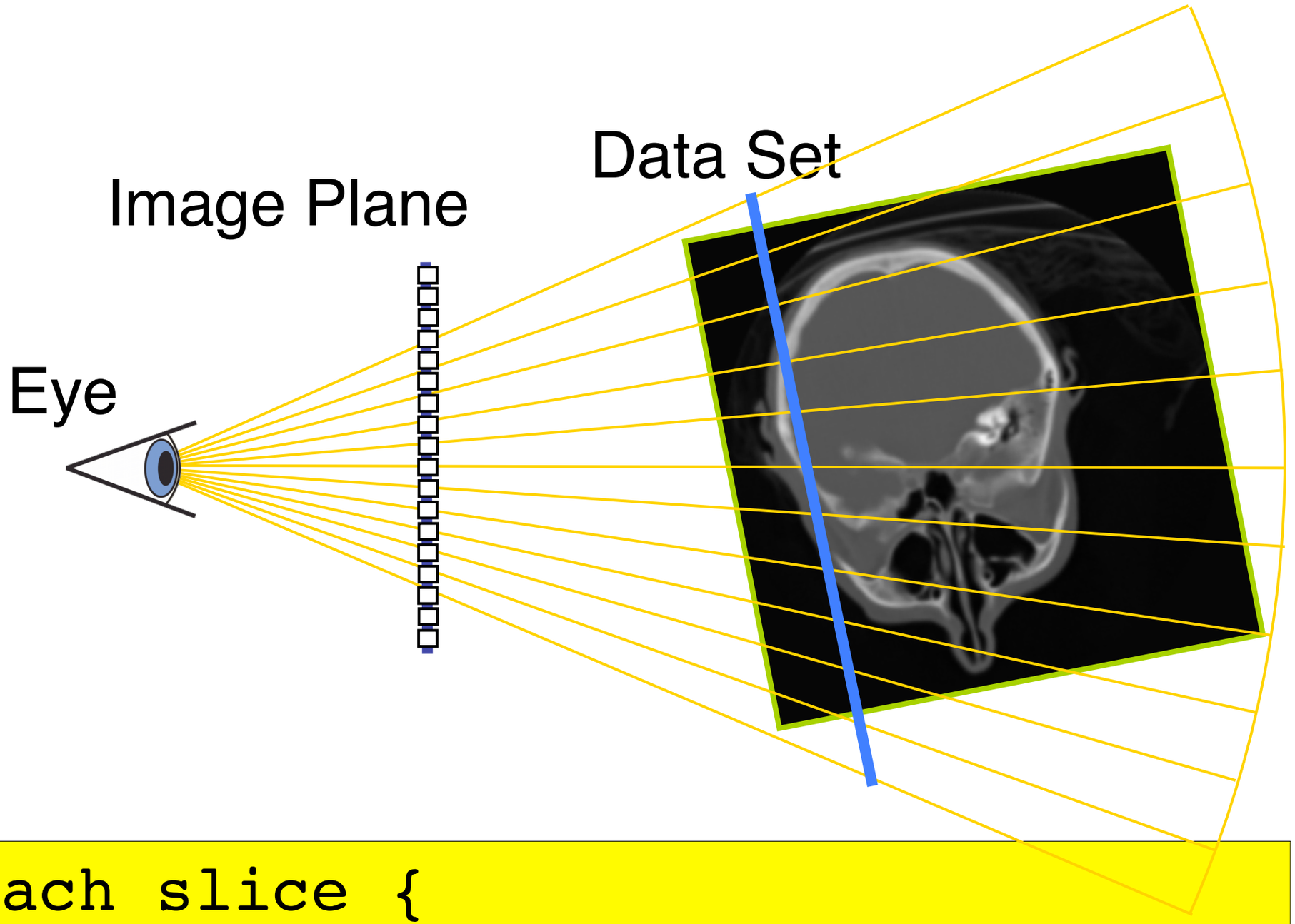

```
For each slice {
   calculate contribution to the image
}
```

Object order approach




```
For each slice {
   calculate contribution to the image
}
```

Object order approach




```
For each slice {
   calculate contribution to the image
}
```

Object Order: Splatting

- Main Idea: “Throw” voxels to the image
  - Front-To-Back or Back-To-Front
- Original method - fast, poor quality, many many improvements since then!
  - Crawfis’93: textured splats
  - Swan’96, Mueller’97: anti-aliasing
  - Mueller’98: image-aligned sheet-based splatting
  - Mueller’99: post-classified splatting
  - Huang’00: new splat primitive: FastSplats

Computer Graphics, Volume 24, Number 4, August 1990

Footprint Evaluation for Volume Rendering

Lee Westover
Numerical Design Limited
The University of North Carolina at Chapel Hill

ABSTRACT

This paper presents a forward mapping rendering algorithm to display regular volumetric grids that may not have the same spacings in the three grid directions. It takes advantage of the fact that convolution can be thought of as distributing energy from input samples into space. The renderer calculates an image plane footprint for each data sample and uses the footprint to spread the sample's energy onto the image plane. A result of the technique is that the forward mapping algorithm can support perspective without excessive cost, and support adaptive resampling of the three-dimensional data set during image generation.

KEYWORDS: 3D Image, Volume Rendering, Reconstruction, Algorithms.

INTRODUCTION

Volume rendering is the direct display of data sampled in three dimensions. There are two principle approaches to volume rendering: backward mapping algorithms that map the image plane onto the data by shooting rays from pixels into the data space, and forward mapping algorithms that map the data onto the image plane.

* Author's current address:
Sun Microsystems Inc
PO Box 13447
Research Triangle Park NC
27709

This distinction principly manifests itself in how and when reconstruction of the three-dimensional signal is done. Convolution can be thought of as either generating an output sample from many input samples or as spreading one input sample to many output samples. Backward mapping algorithms typically reconstruct the signal at a point in space by looking at that point's nearest data samples and performing some type of interpolation. Forward mapping algorithms differ in that they incrementally reconstruct the original signal by spreading each data sample's energy into space.

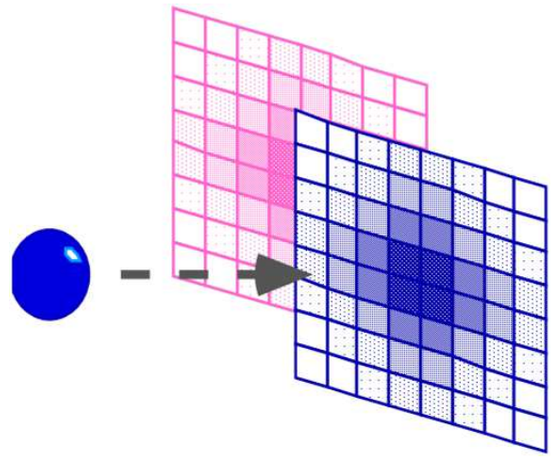
Forward mapping algorithms are important because they are easily made parallel. Since each data sample only needs to know about a small surrounding neighborhood of other samples, shading and transforming can be done in parallel for sub-sections of the data. With today's parallel machines having limited local memory, this data distribution gets around the backward mapping problem of having the entire data set at each node.

The reconstruction step is the most complicated part of the algorithm. The renderer must determine the screen space contribution of each sample point to the final image. A brute force method would perform a one-dimensional integration of the reconstruction kernel for every pixel for every input sample. If the renderer can calculate the screen space extent of the kernel, the number of integrations reduces to the number of samples times the number of pixels that fall within the extent. However, this is still an enormous number of integrations.

In an orthographic view, the footprint of the projected reconstruction kernel for any sample is a constant except

Splatting

- Instead of asking which data samples contribute to a pixel value, ask, to which pixel values does a data sample contribute?
  - Ray casting: pixel value computed from multiple data samples
  - Splatting: multiple pixel values (partially) computed from a single data sample
- Overview:
  - High-quality
  - Relatively costly ->relatively slow
- Idea: contribute every voxel to the image
  - Projection from voxel: splat
  - Composite in image space



Splatting - Footprint

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

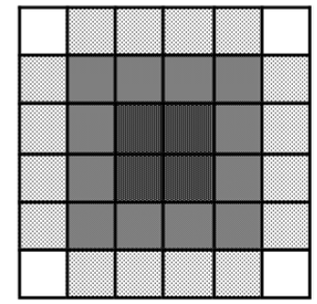
- Footprint geometry

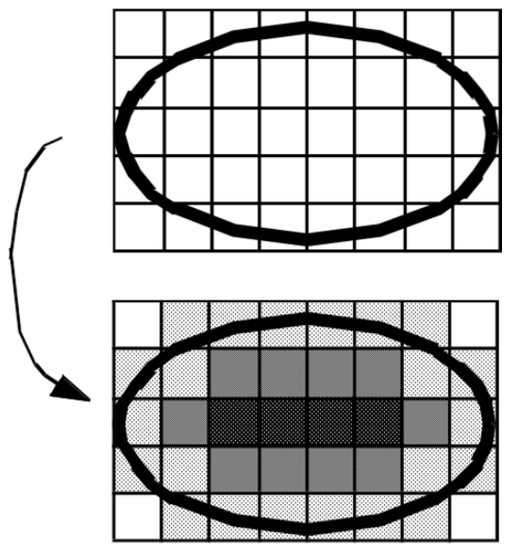
    - Orthographic projection: footprint is independent of the view point

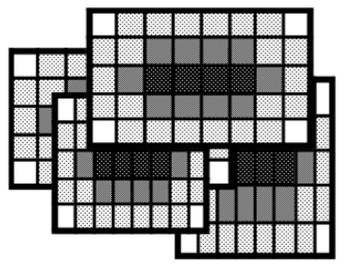
    - Perspective projection: footprint is elliptical

- Pre-integration of footprint

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

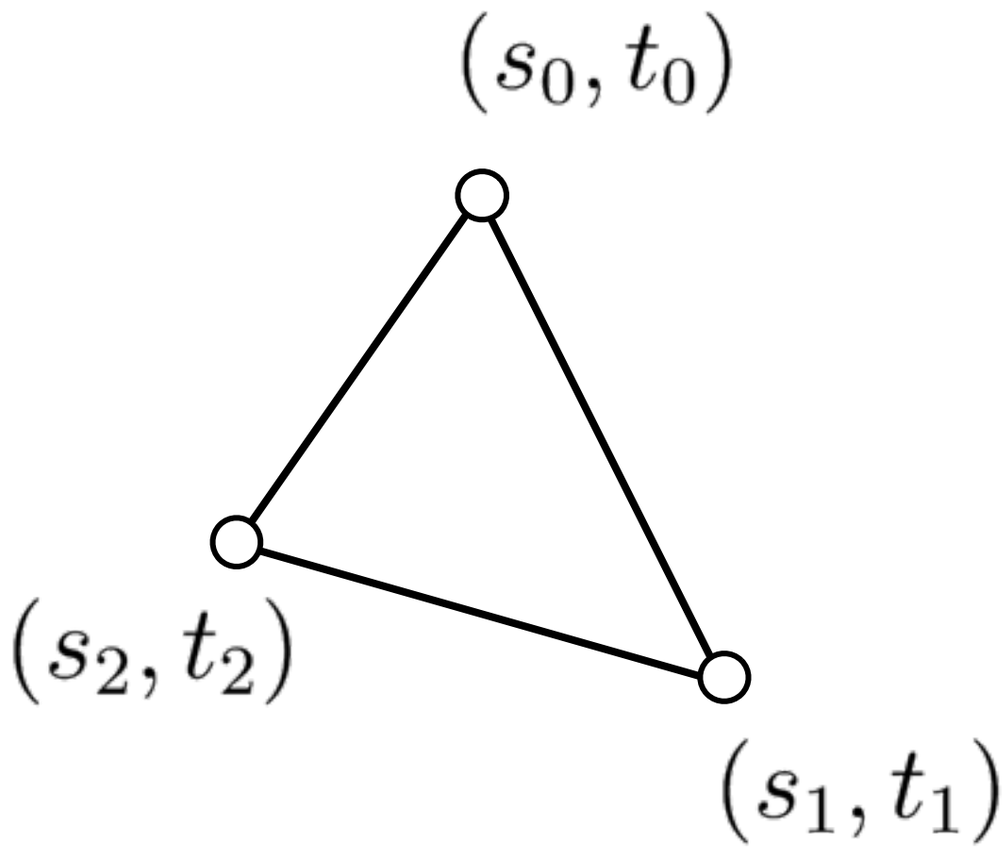




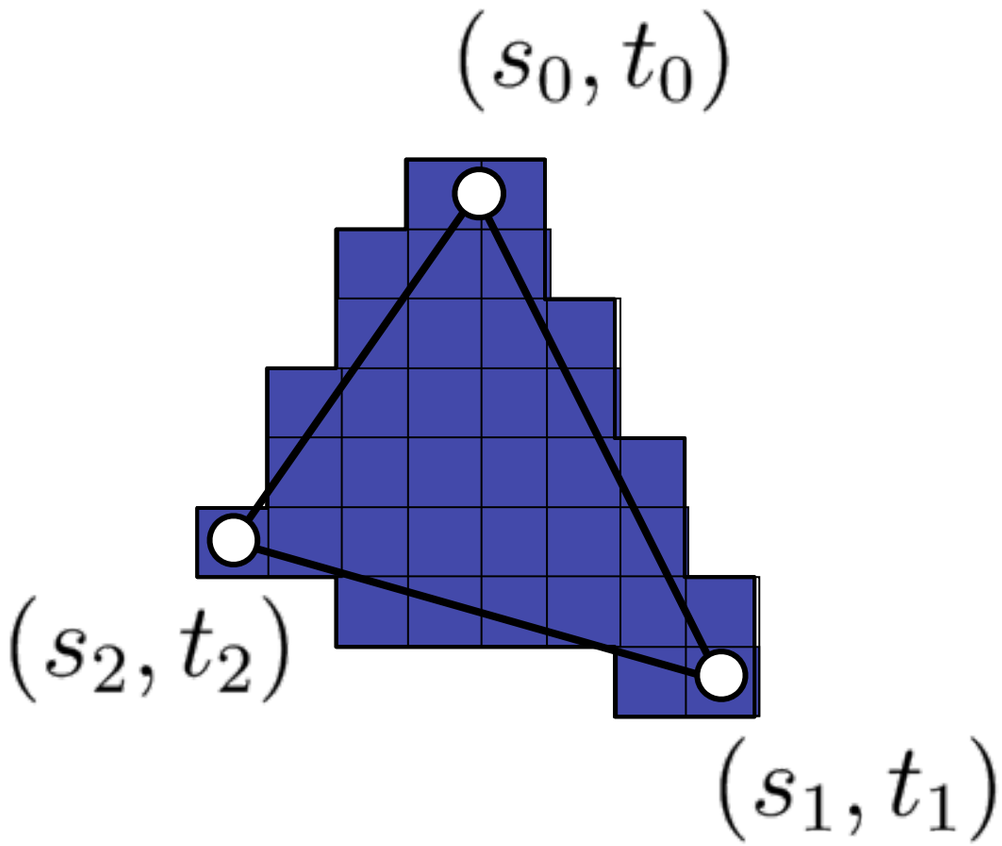


Texture-Mapping

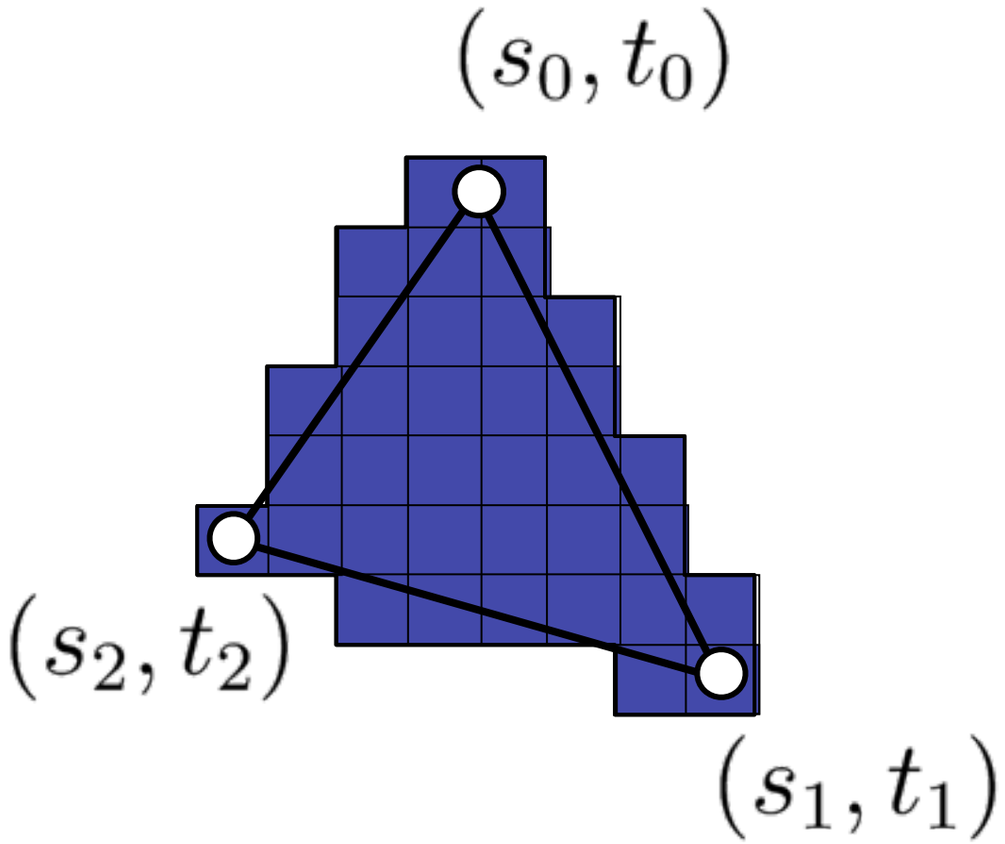
How does a texture work?

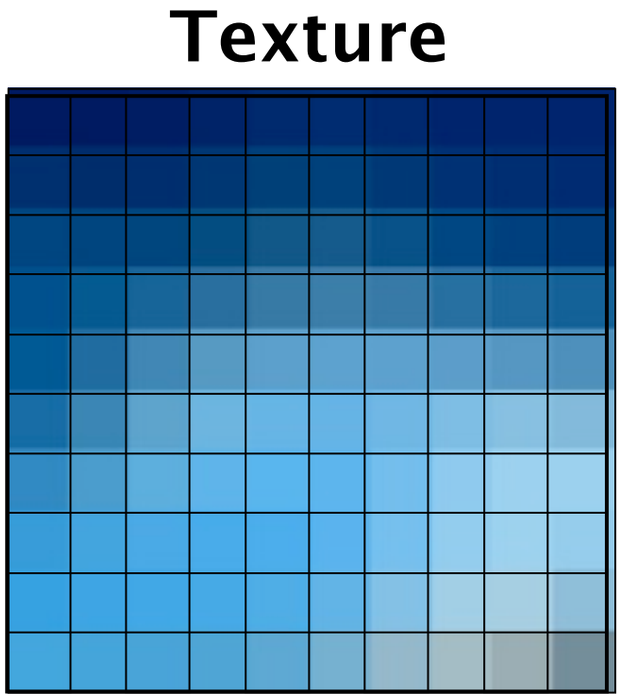


How does a texture work?

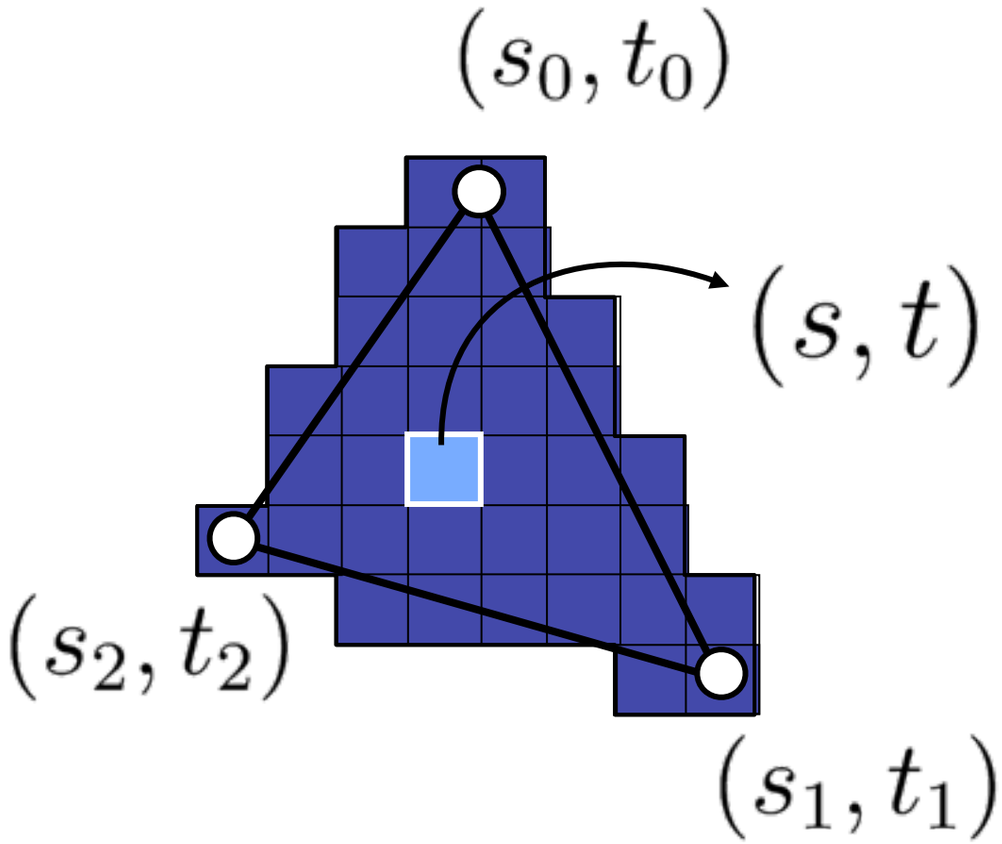


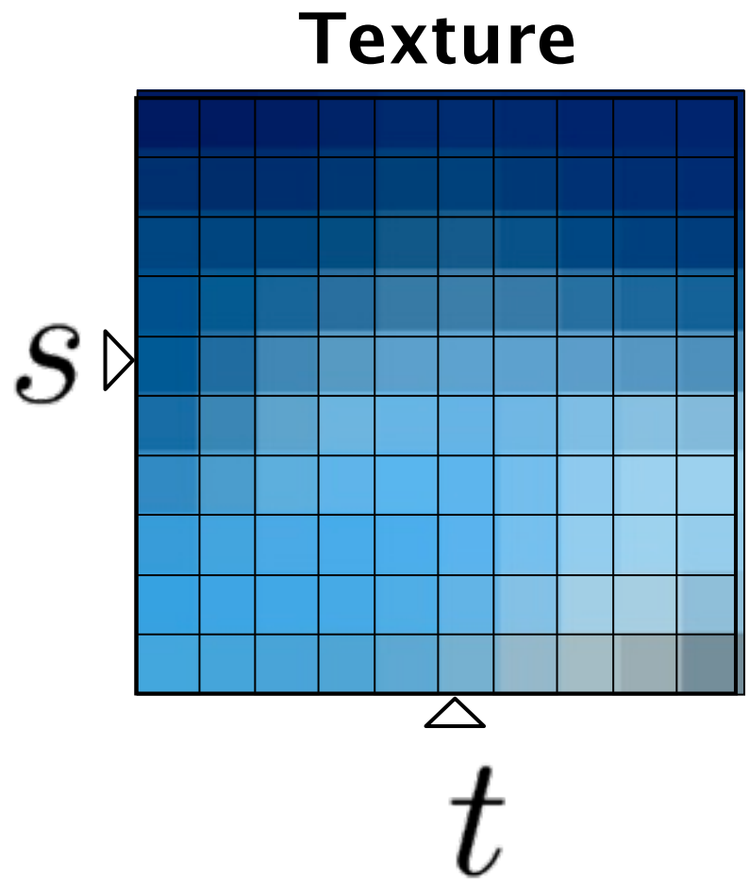
How does a texture work?





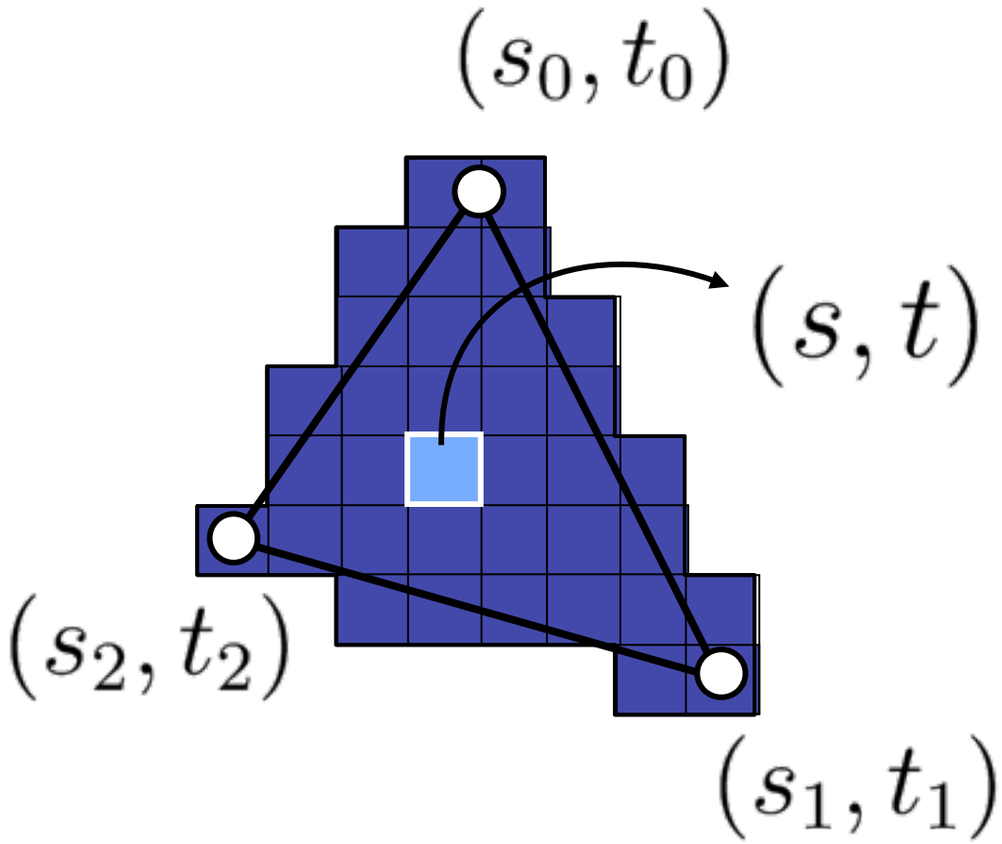
How does a texture work?

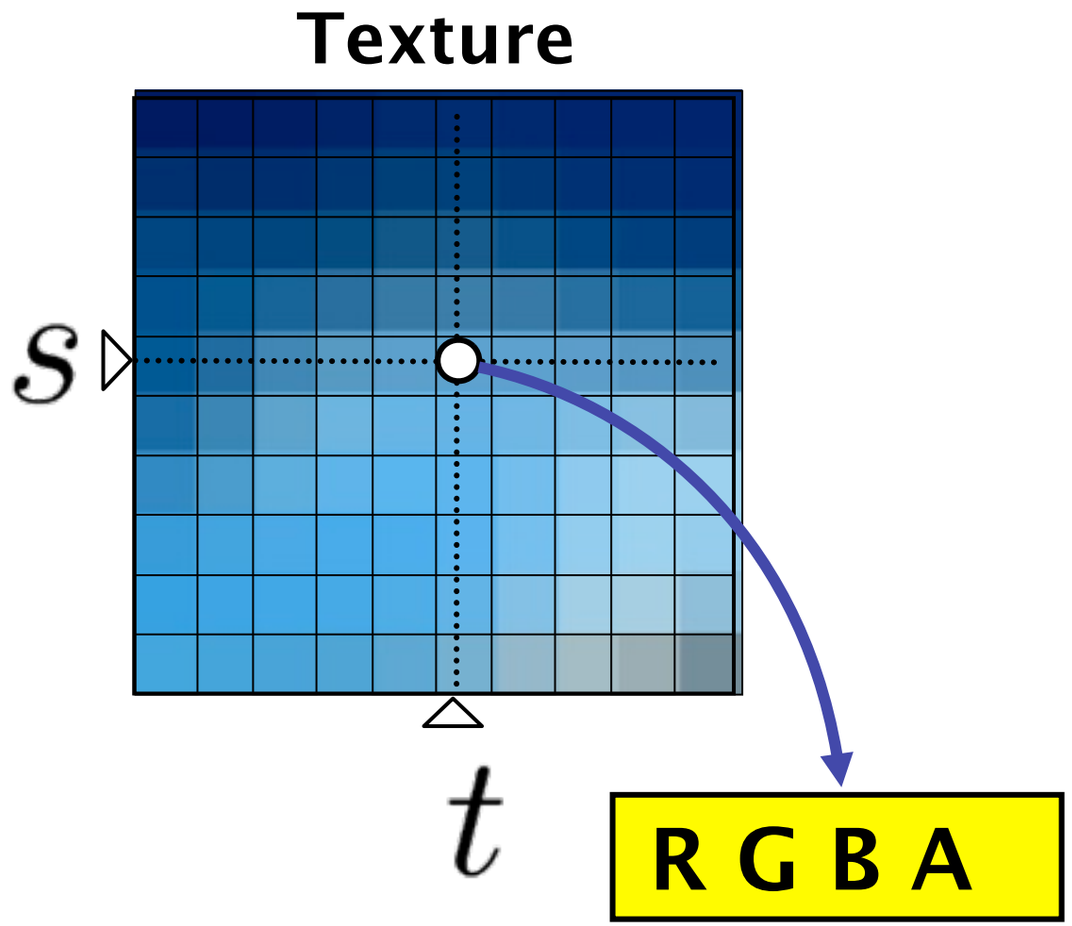




For each fragment:
interpolate the
texture coordinates
(barycentric)

How does a texture work?





For each fragment:
interpolate the
texture coordinates
(barycentric)

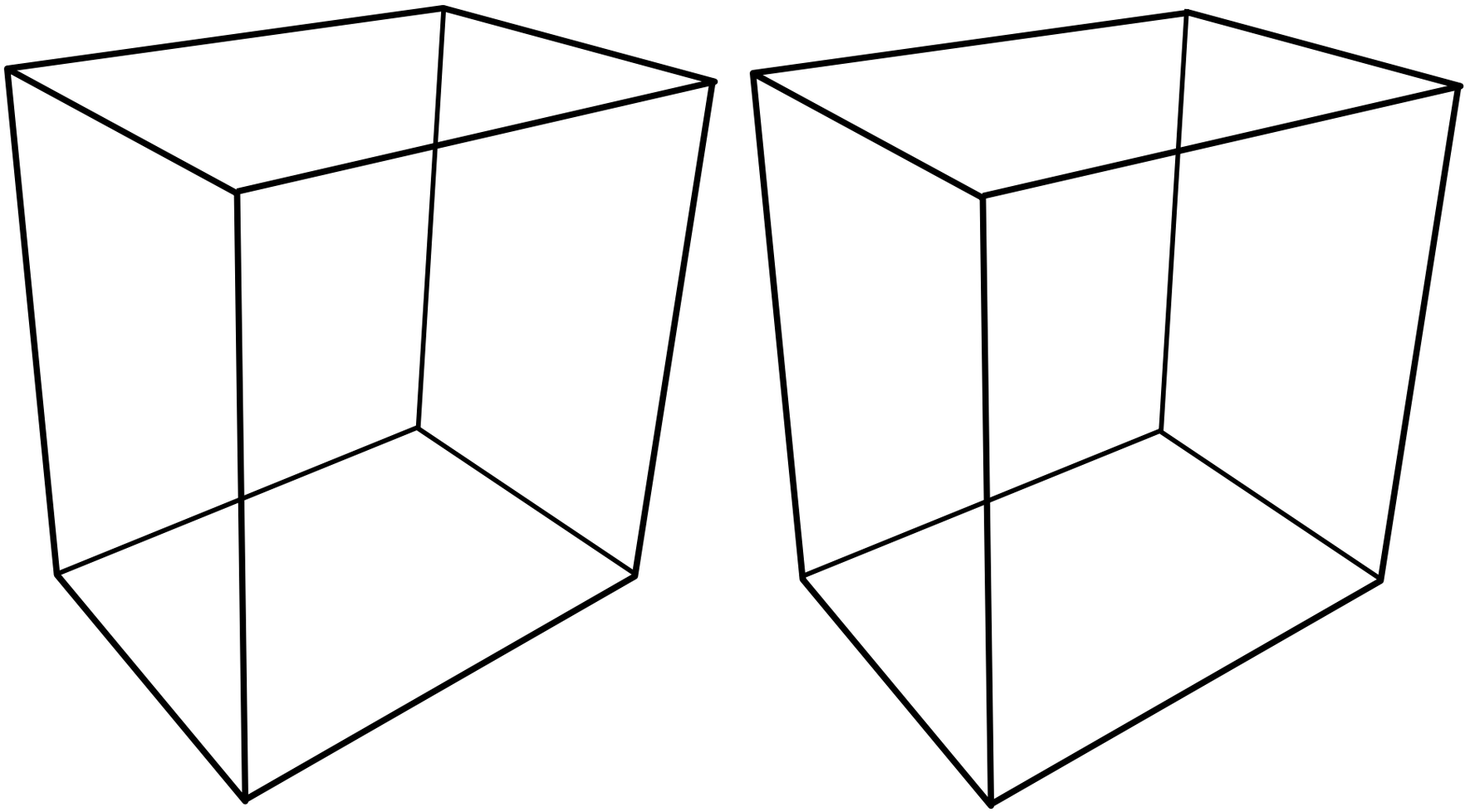
Texture-Lookup:
interpolate the
texture color
(bilinear)

Texture-based

- Volume rendering by 2D texture mapping
  - Use planes parallel to “base plane” (front face of volume which is most orthogonal to view ray)
  - Draw textured rectangles using bilinear interpolation
  - Render back-to-front using compositing

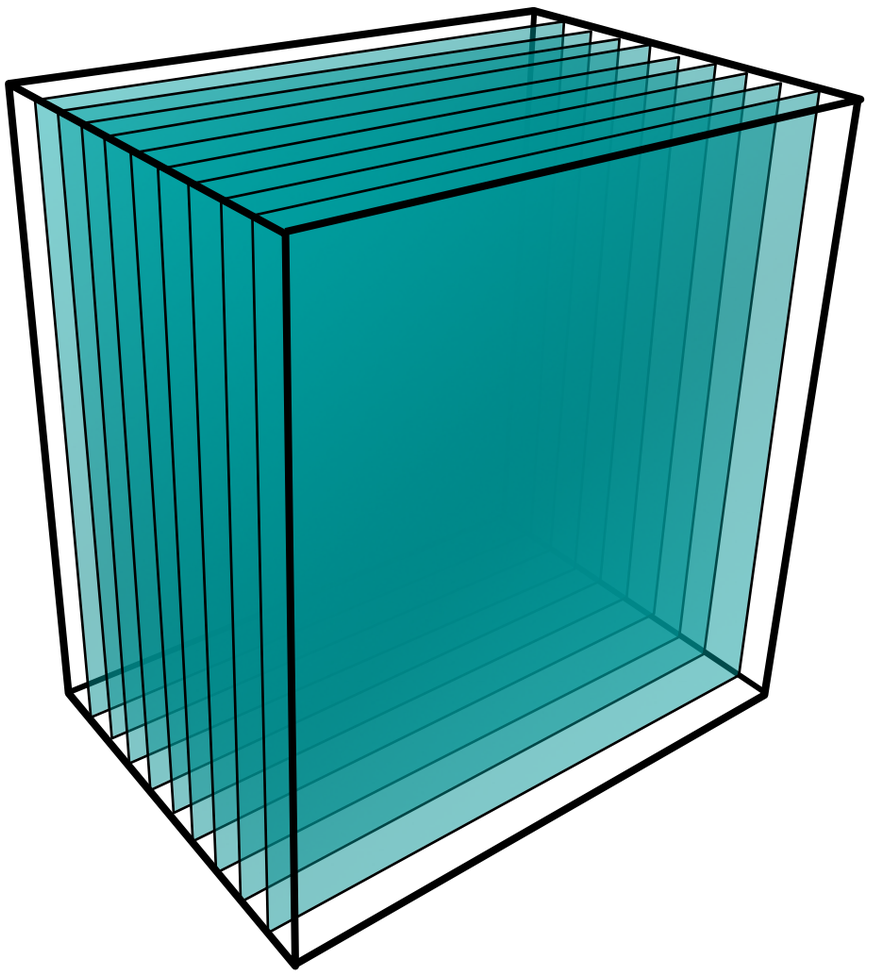
Texture-based

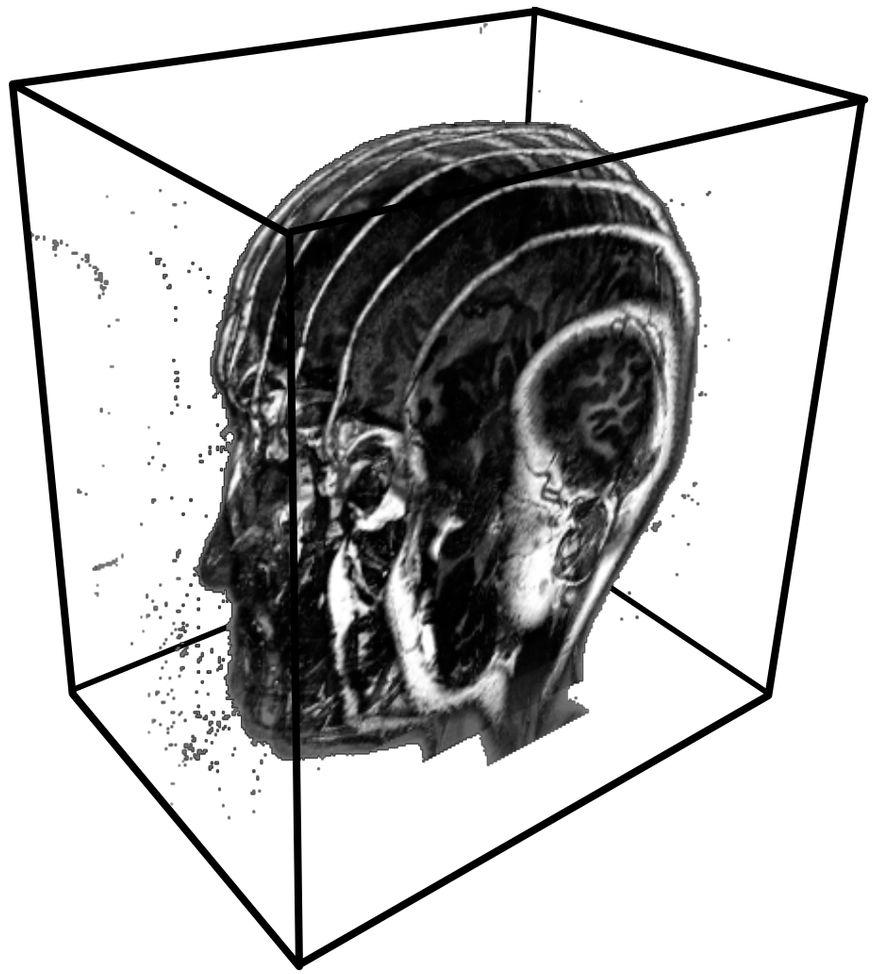
- Volume rendering by 2D texture mapping
  - Use planes parallel to “base plane” (front face of volume which is most orthogonal to view ray)
  - Draw textured rectangles using bilinear interpolation
  - Render back-to-front using compositing



Texture-based

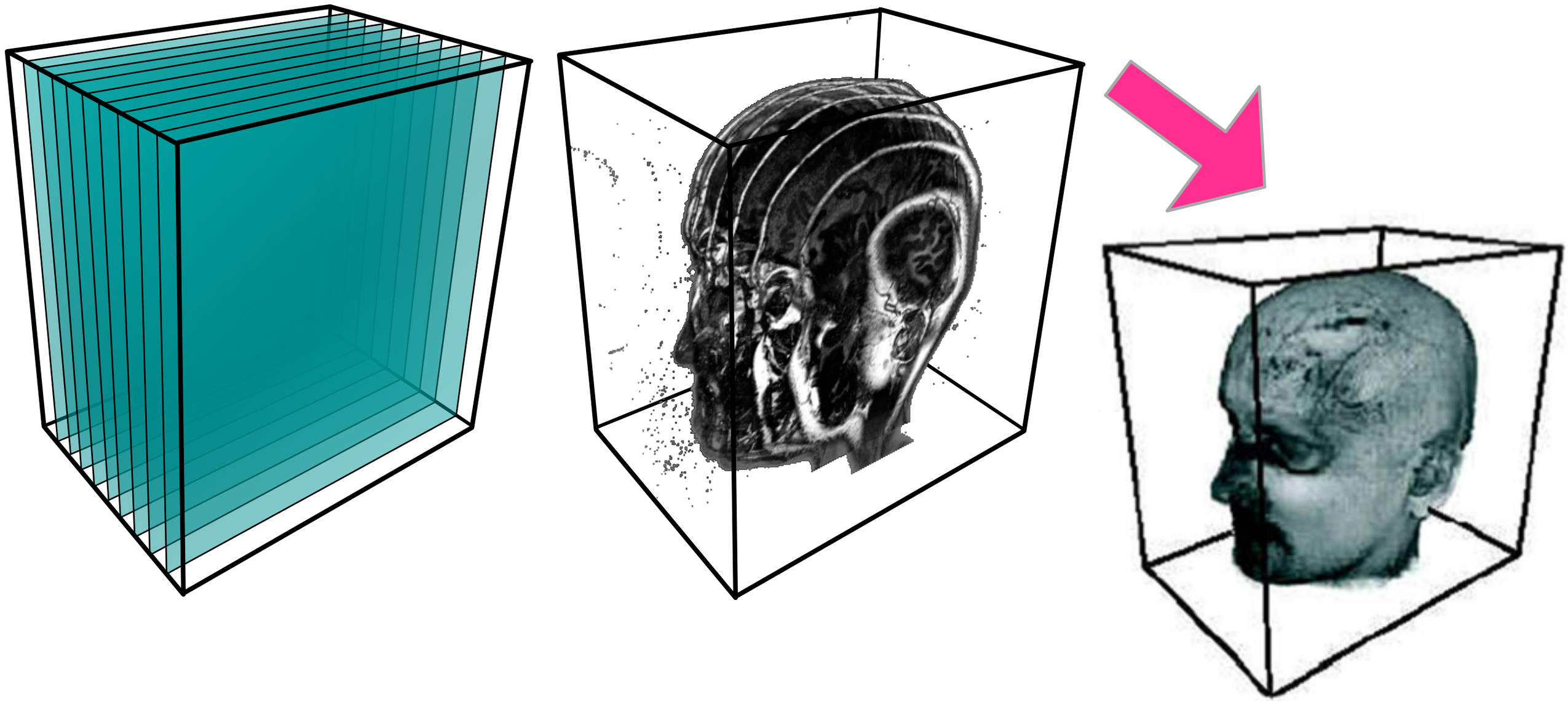
- Volume rendering by 2D texture mapping
  - Use planes parallel to “base plane” (front face of volume which is most orthogonal to view ray)
  - Draw textured rectangles using bilinear interpolation
  - Render back-to-front using compositing





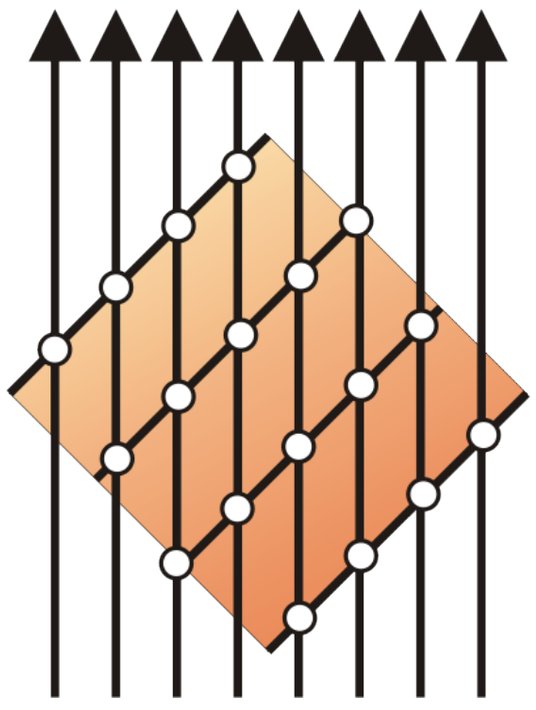
Texture-based

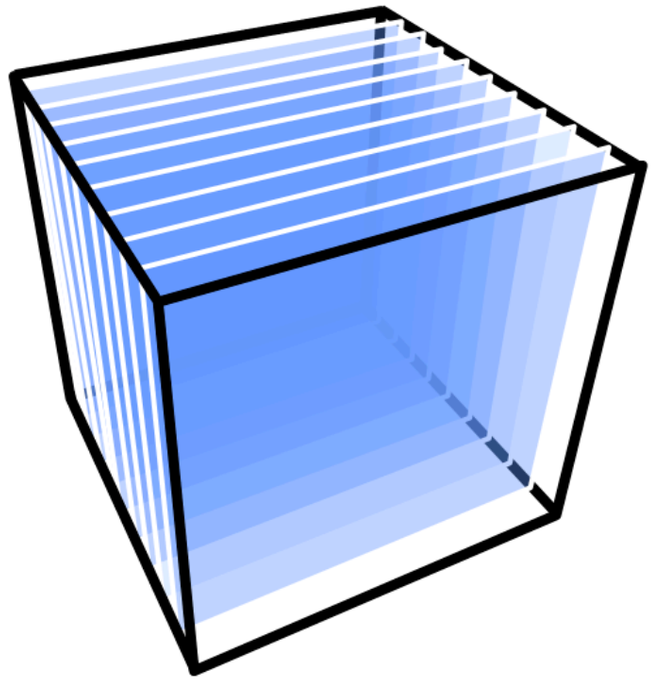
- Volume rendering by 2D texture mapping
  - Use planes parallel to “base plane” (front face of volume which is most orthogonal to view ray)
  - Draw textured rectangles using bilinear interpolation
  - Render back-to-front using compositing

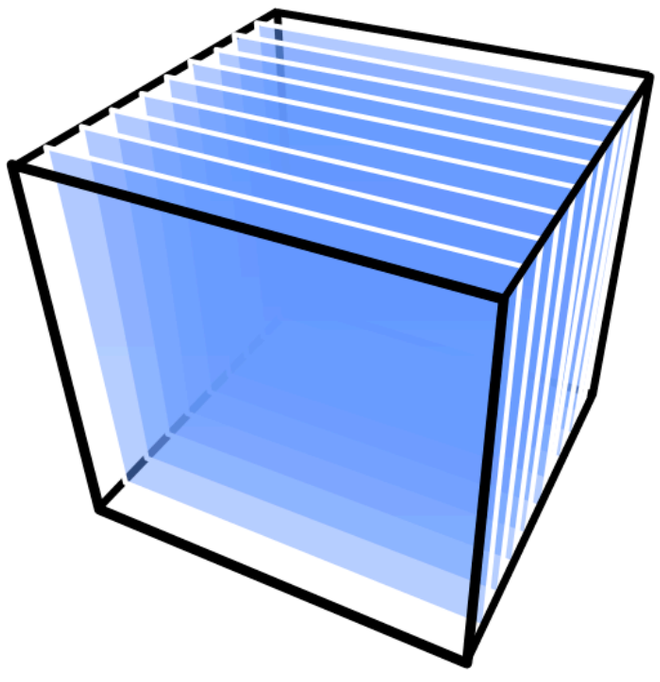


Texture-Based Volume Rendering

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

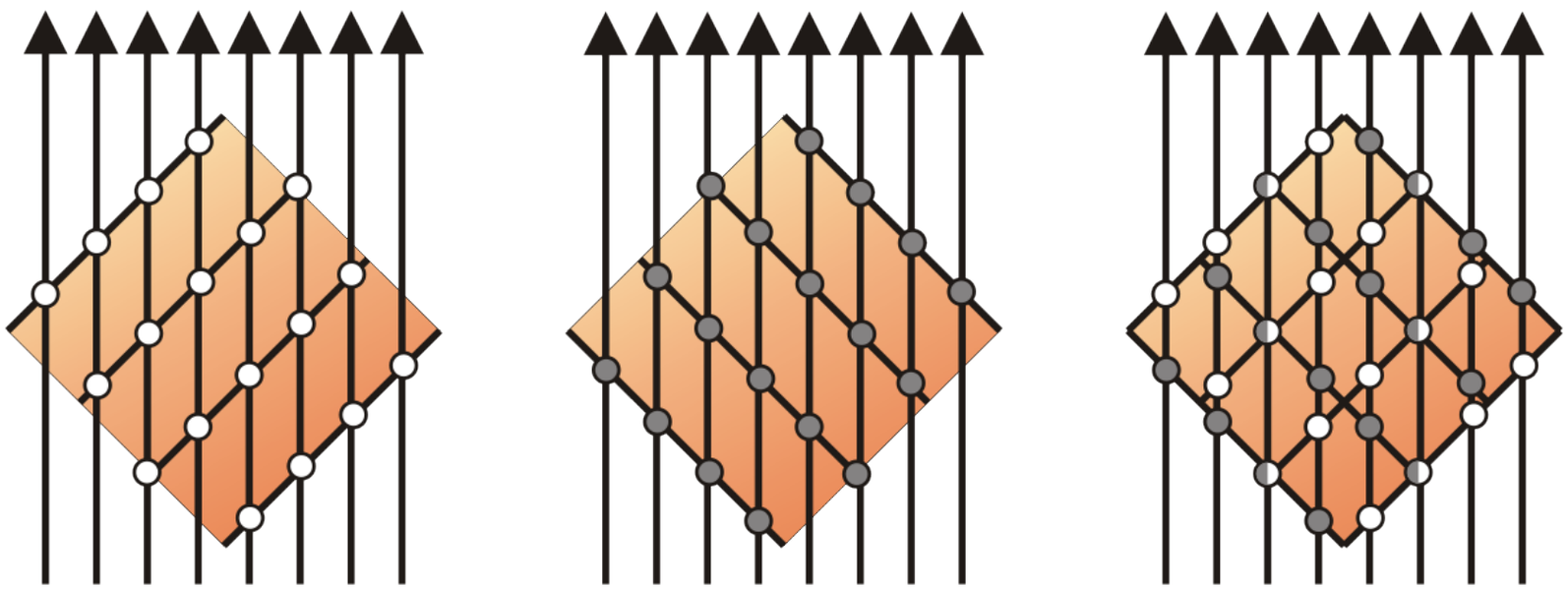






Texture-Based Volume Rendering

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

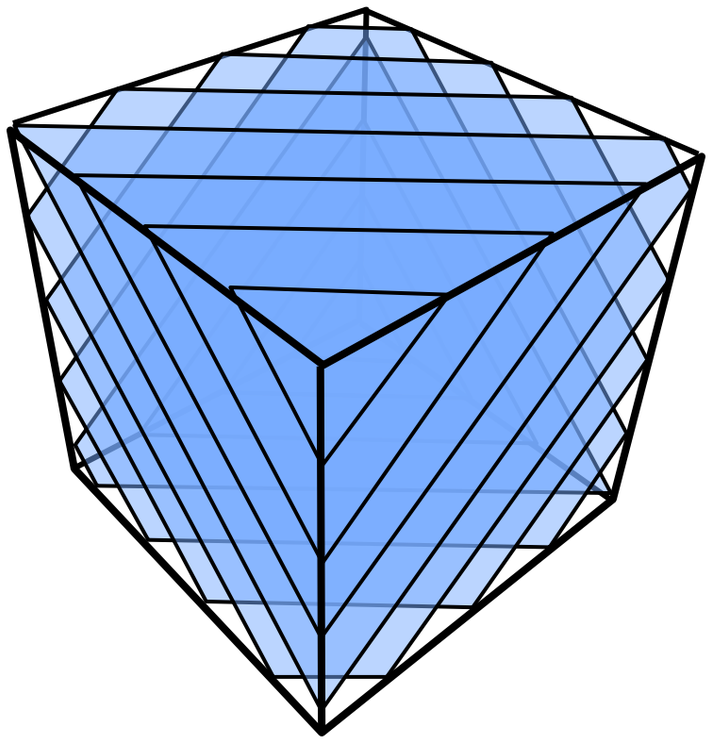


3D Textures

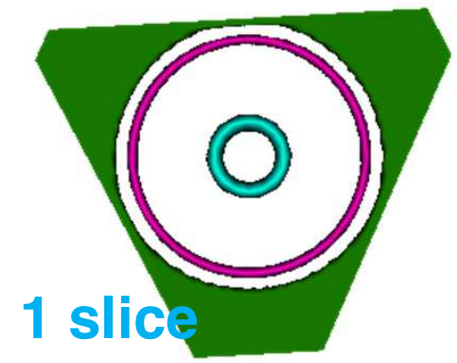
- Voxel data becomes a 3D Texture Object
  - Trilinear Interpolation in Hardware
  - Uses an arbitrary number of slices parallel to the image plane
  - Back-to-front compositing used just as in 2d texturing
  - Can be limited by the size of the texture memory

3D Textures

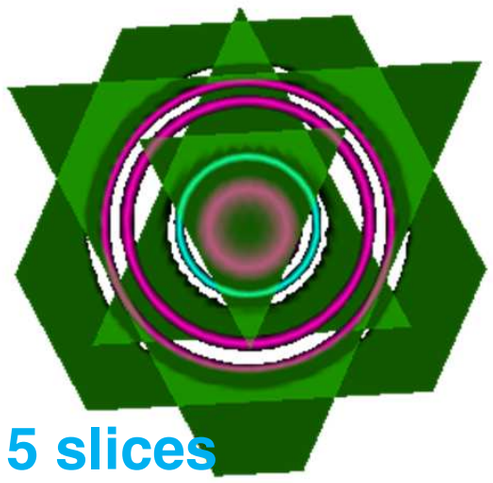
- Voxel data becomes a 3D Texture Object
  - Trilinear Interpolation in Hardware
  - Uses an arbitrary number of slices parallel to the image plane
  - Back-to-front compositing used just as in 2d texturing
  - Can be limited by the size of the texture memory



Effect of the Sample Rate

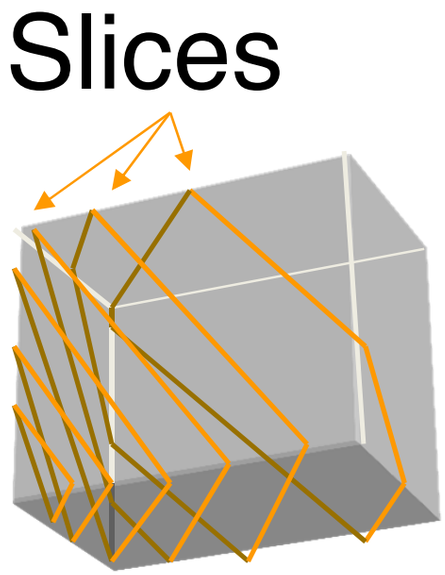


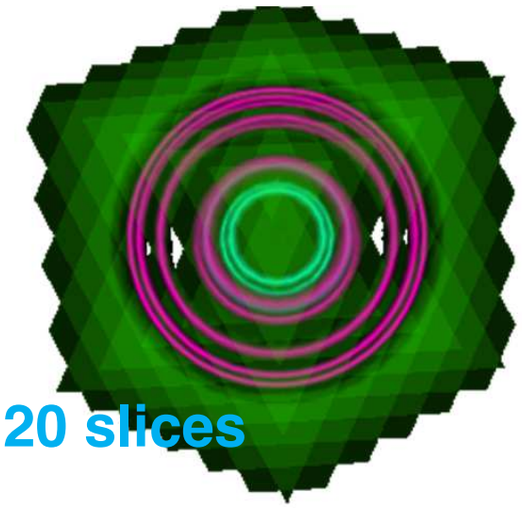
1 slice



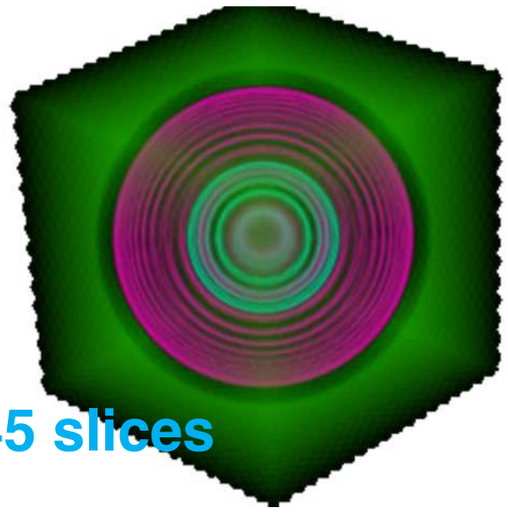
5 slices

View direction

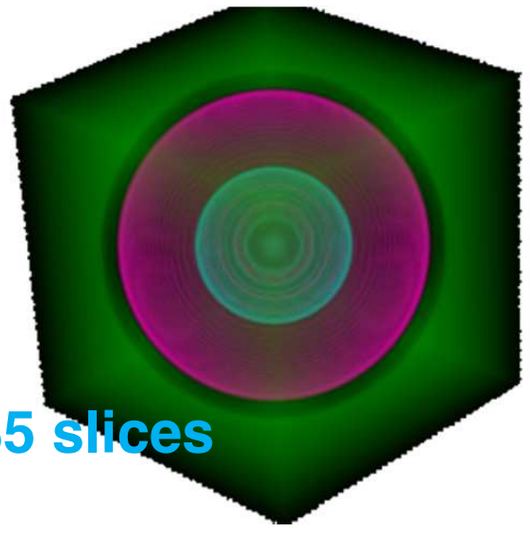




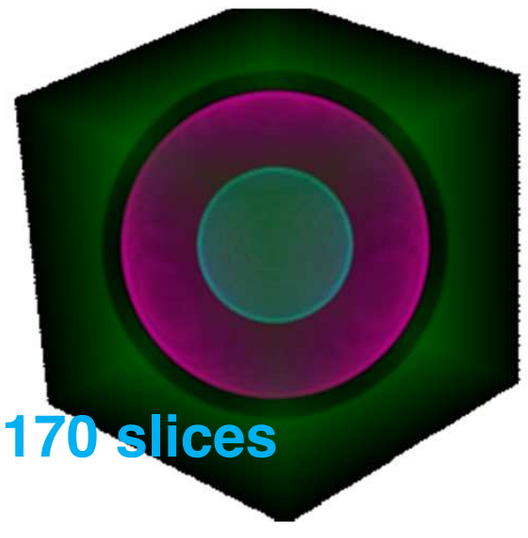
20 slices



45 slices



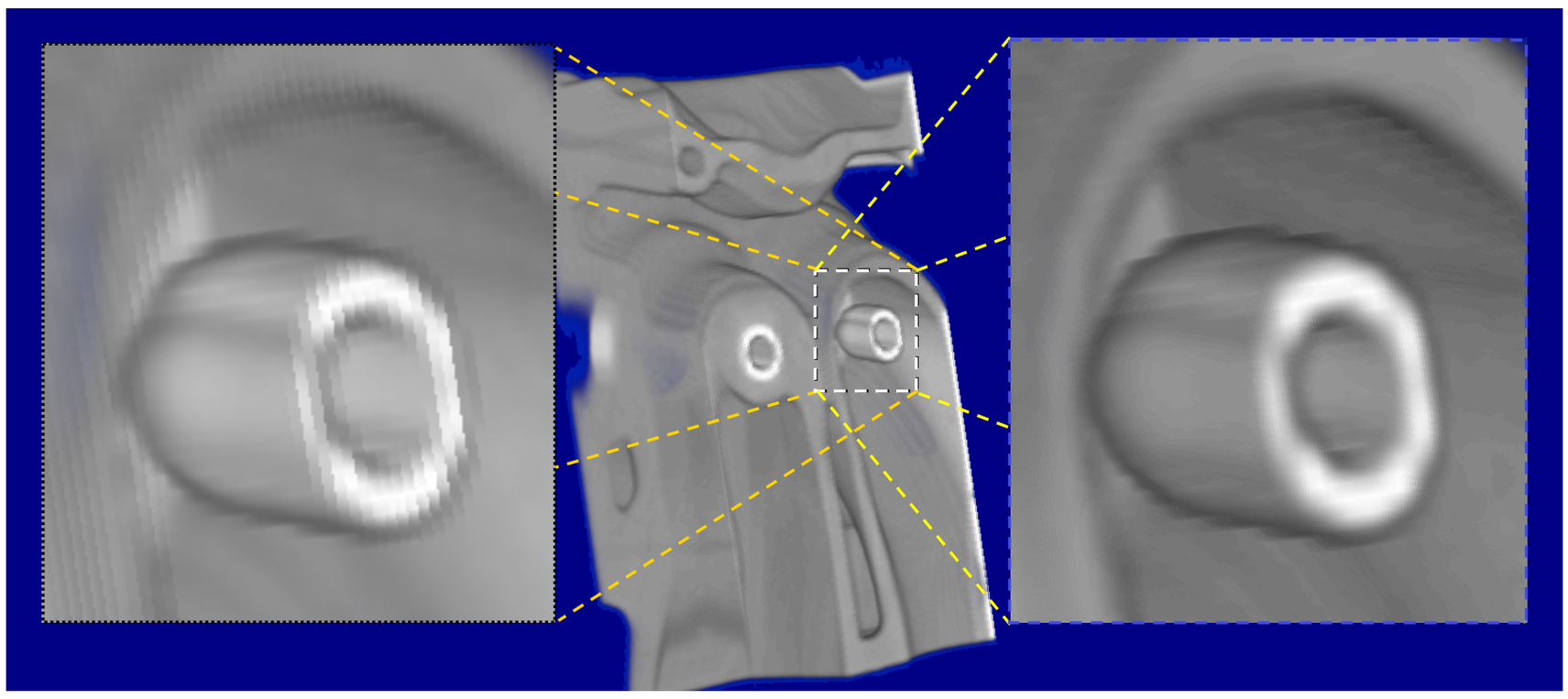
85 slices

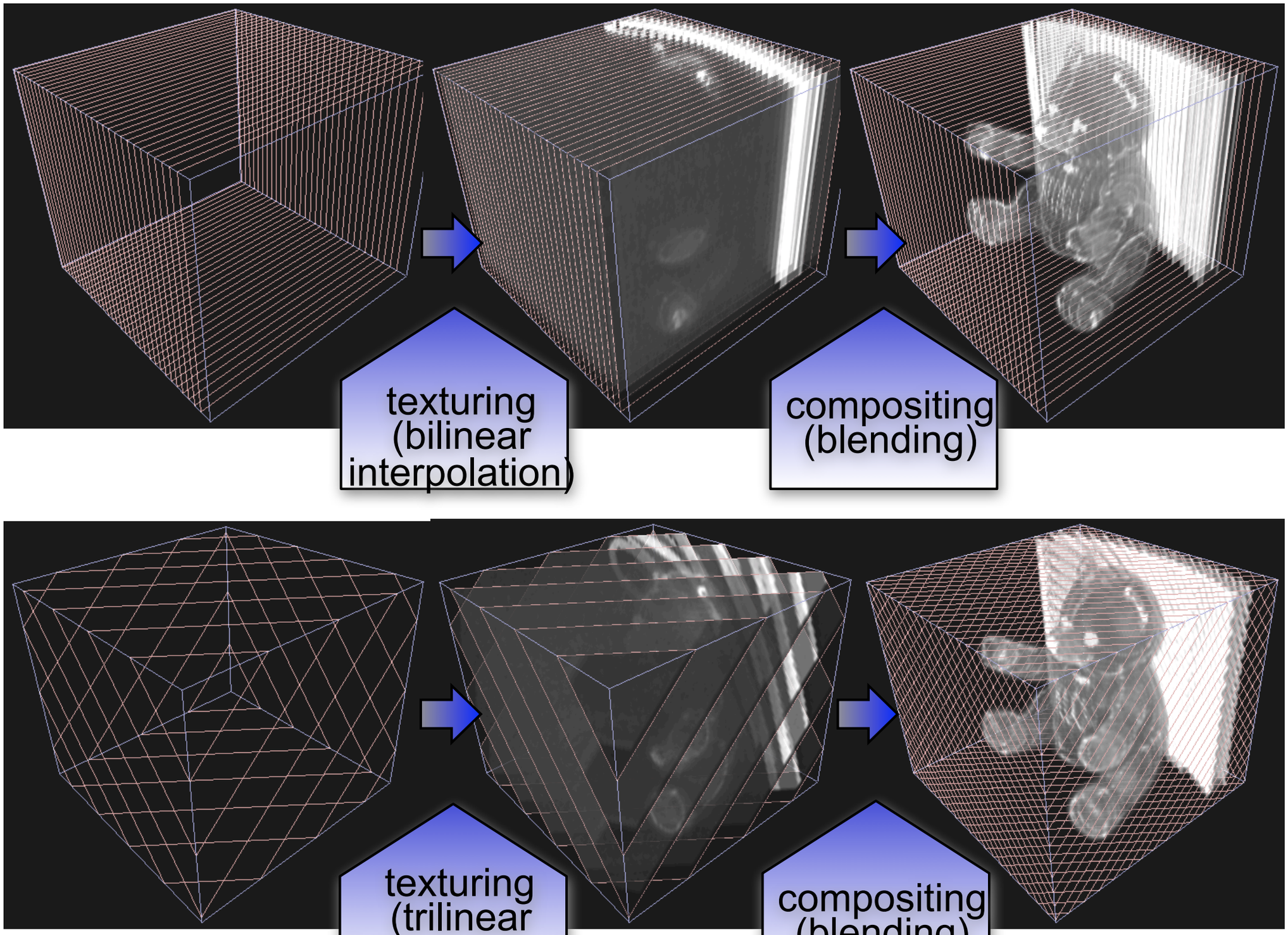


170 slices

Texture-Based Volume Rendering

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

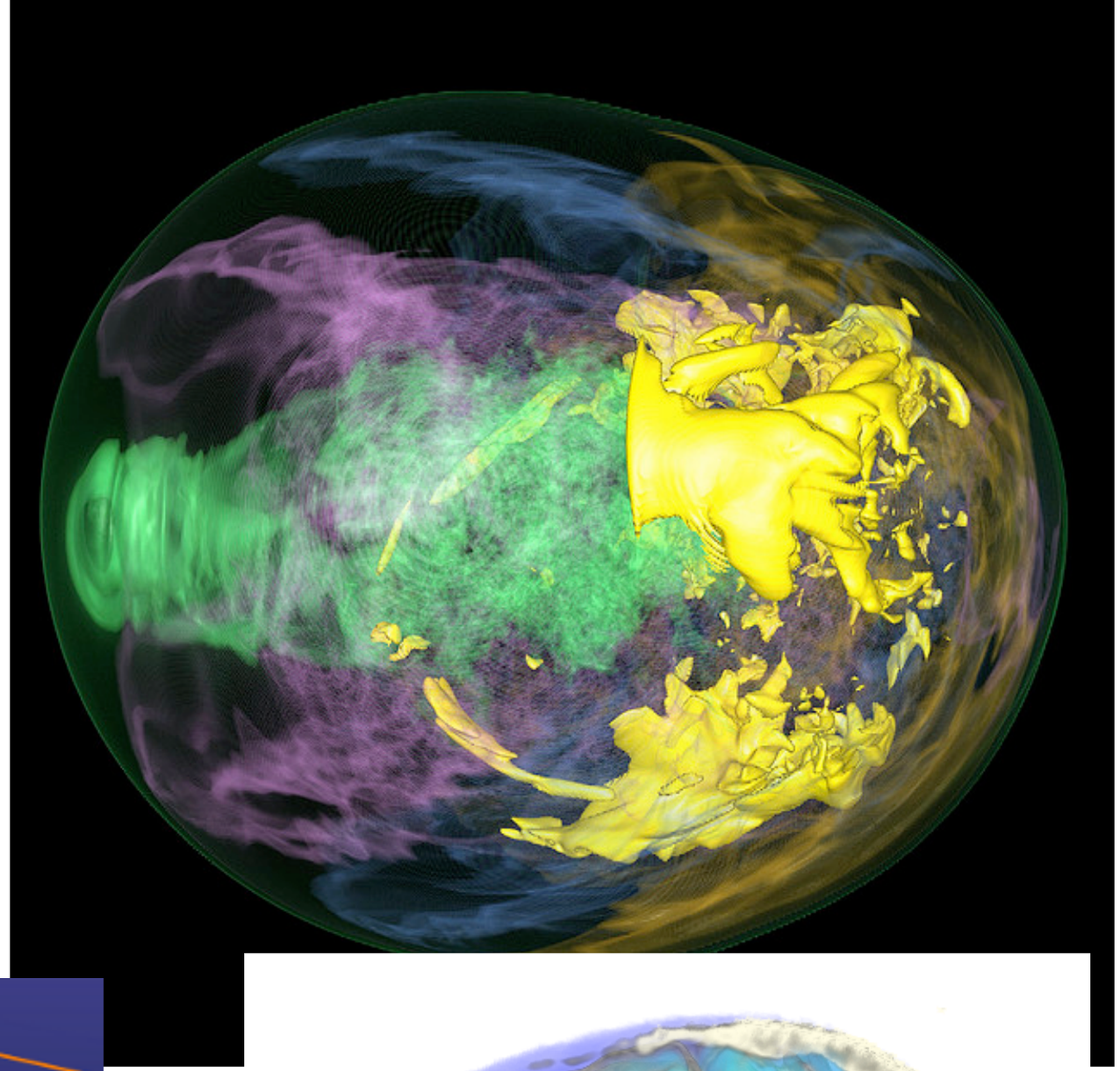


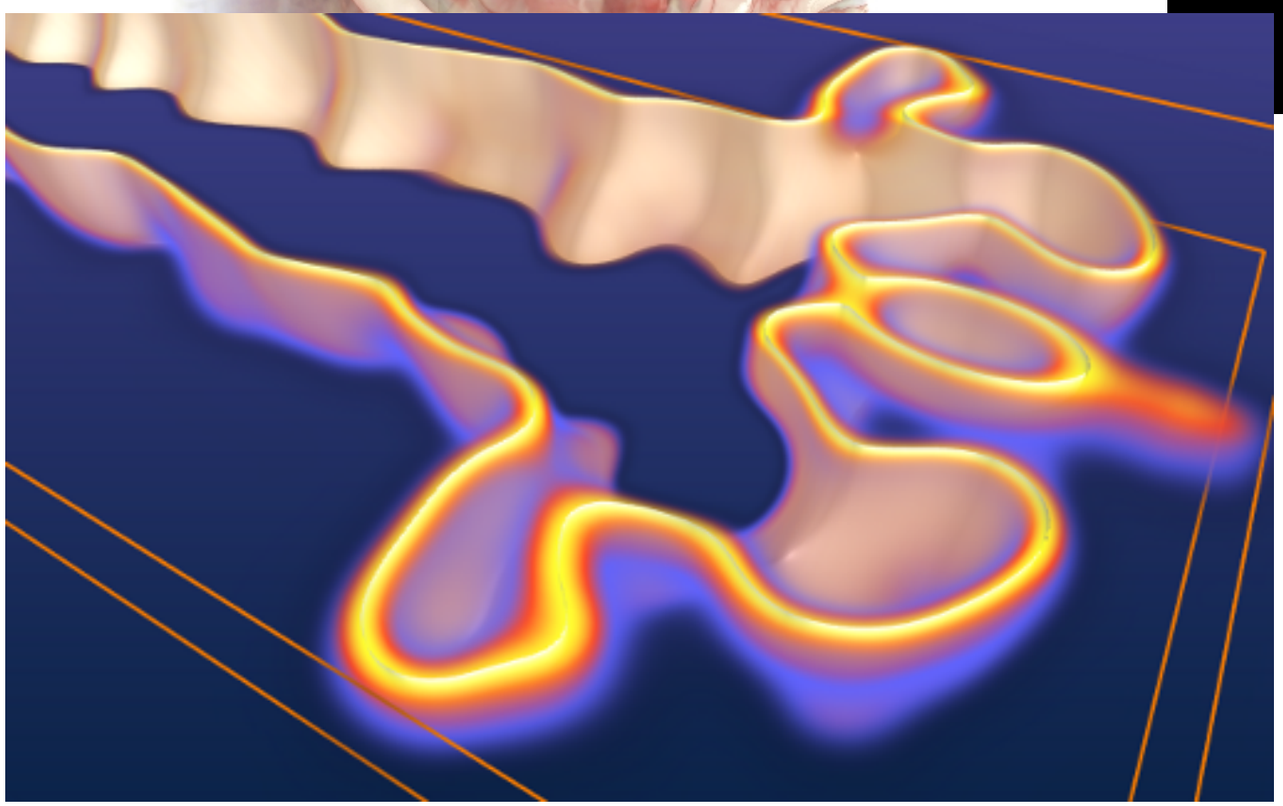


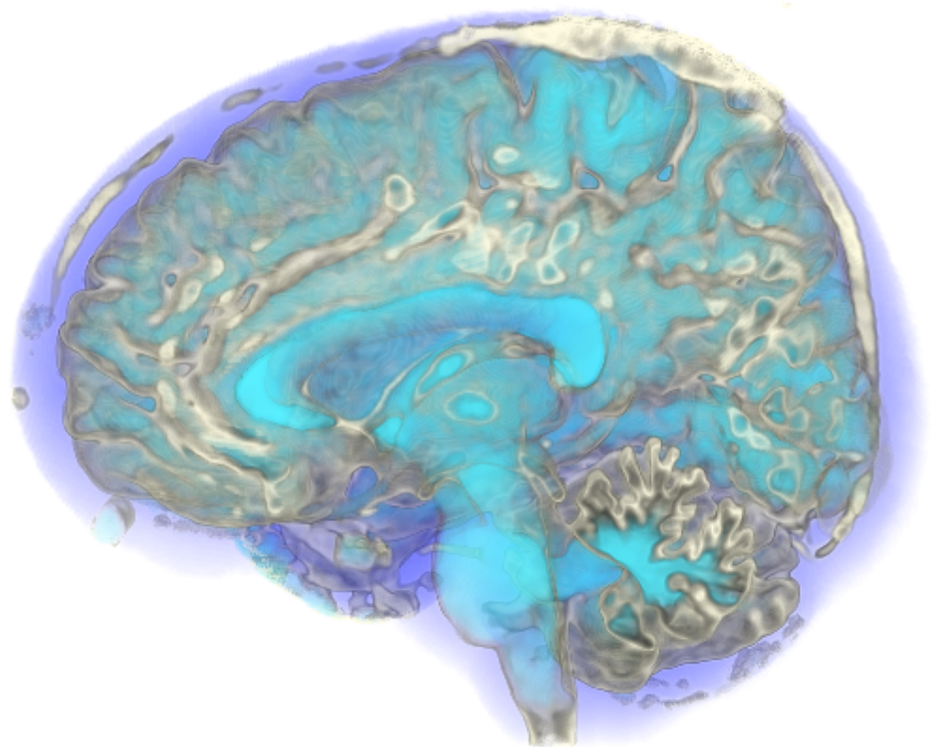
2D textures
axis-aligned

3D texture
view-aligned

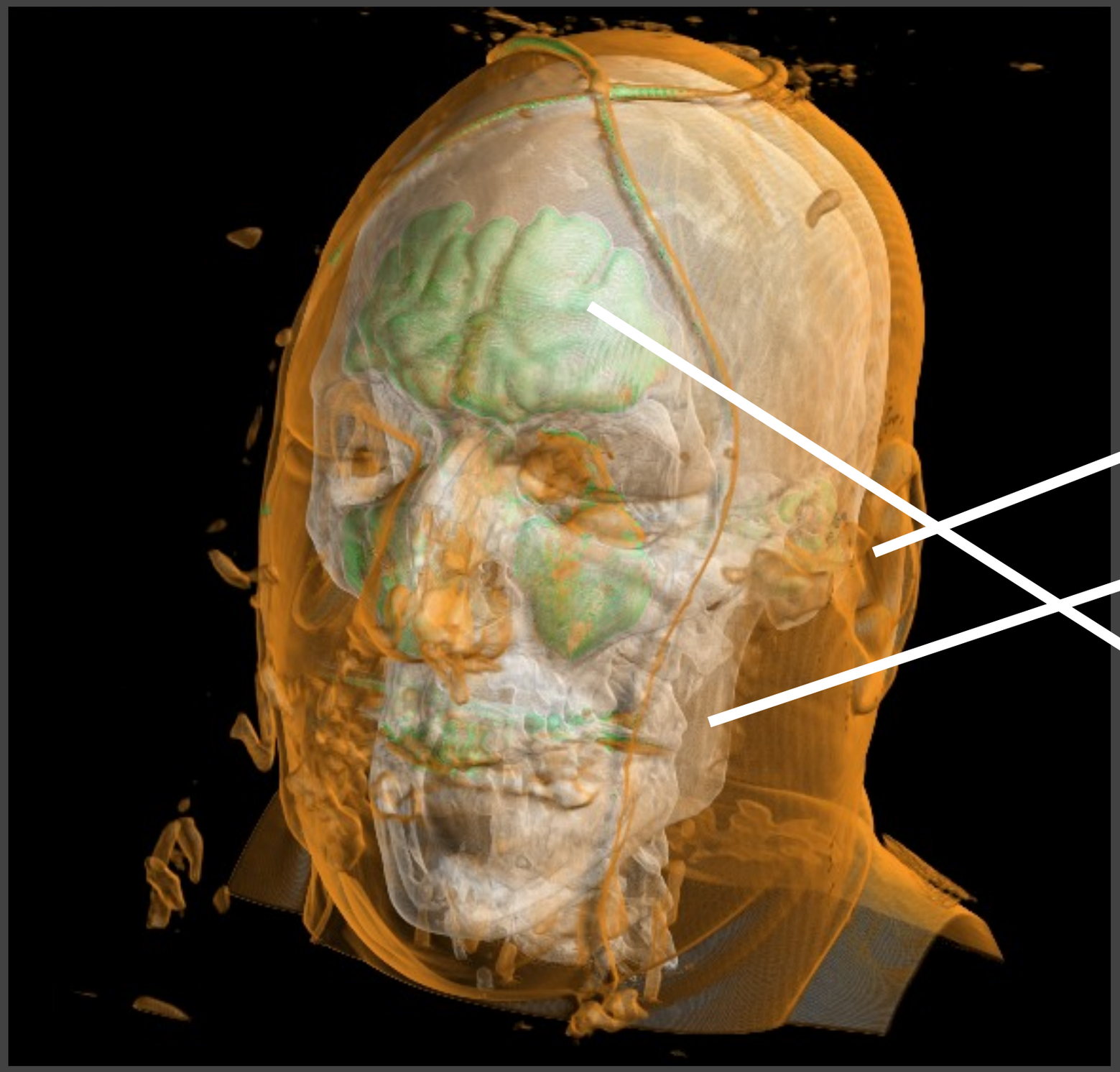
Sum Up







Volume Rendering: Interfaces



**Transfer function,
With shading**

Skin/Air

Bone/Soft tissue

Bone/Air

Computational Strategies

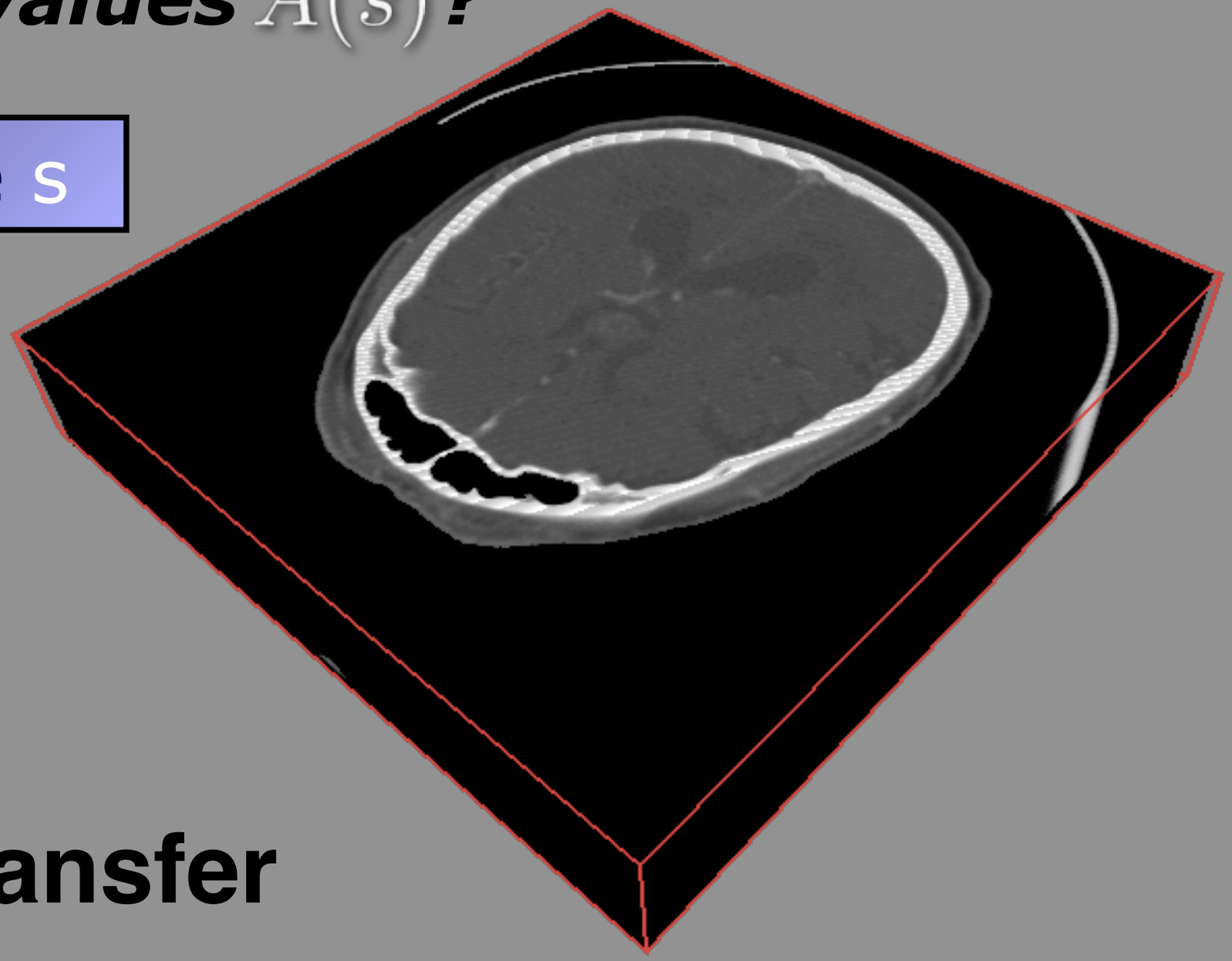
- Image Order:
    - Ray casting (many options)
- Object Order:
    - Splatting, Texture mapping

Next Time

Classification

How do I obtain the emission values $q(s)$ ***and Absorption values*** $A(s)$***?***

scalar value s



Answer: Transfer Functions

Lec20 Reading

- The Transfer Function Bake-Off. Hanspeter Pfister, William E. Lorensen, Chandrajit L. Bajaj, Gordon L. Kindlmann, William J. Schroeder, Lisa Sobierajski Avila, Ken Martin, Raghu Machiraju, Jinho Lee. IEEE Computer Graphics and Applications 21(3): 16-22 (2001).

- Multidimensional Transfer Functions for Interactive Volume Rendering. Joe Kniss, Gordon L. Kindlmann, Charles D. Hansen IEEE Trans. Vis. Comput. Graph. 8(3): 270-285 (2002).

Assignment 05

Assigned: Monday, March 27
Due: Monday, April 10, 4:59:59 pm

Reminder
Project Milestone 02

Assigned: Wednesday, February 22
Due: Wednesday, March 29, 4:59:59 pm