SIGGRAPH2004

Real-Time Volume Graphics

Klaus Engel
Siemens Corporate Research
Princeton

Markus Hadwiger
VR VIS Research Center
Vienna, Austria

Joe M. Kniss
Scientific Computing and
Imaging Insitute,
University of Utah

Aaron E. Lefohn
Institute for Data Analysis
and Visualization.
University of California, Davis

Christof Rezk Salama
Computer Graphics and
Multimedia Group
University of Siegen, Germany

Daniel Weiskopf
Visualization and Interactive
Systems Group,
University of Stuttgart, Germany

Welcome and Speaker Introduction

Klaus Engel
Siemens Corporate Research
Princeton

Markus Hadwiger
VR VIS Research Center
Vienna, Austria

Joe M. Kniss
Scientific Computing and
Imaging Insitute,
University of Utah

Aaron E. Lefohn
Institute for Data Analysis
and Visualization,
University of California, Davis

Christof Rezk Salama
Computer Graphics and
Multimedia Group
University of Siegen, Germany

Daniel Weiskopf
Visualization and Interactive
Systems Group,
University of Stuttgart, Germany

Real-Time Volume Graphics

What to expect?

- Direct Volume Rendering
- Hardware-Acceleration
- From the basics to the state-of-the-art.
- Interaction Techniques and Usability Aspects

Scientific Visualization

- High Precision, Image Quality for Engineering and Medicine

Visual Arts and Entertainment

- Translucency and Scattering
- Visual Effects, Volumetric Models
- Procedural Textures and Animation

Real-Time Volume Graphics

Prerequisites:

- *Working Knowledge in Computer Graphics*
- *Familiarity with Graphics Hardware Programming and APIs (OpenGL or DirectX)*

Courses Evaluation

At the end of this course:

- Evaluate the course online at
  www.siggraph.org/courses_evaluation
  or follow the link on the course page

Course 28 -Morning

8:40 – 9:40	*Introduction to GPU-Based Volume Rendering*
9:40 –10:15	*GPU-Based Ray Casting*
10:15 – 10:30	*BREAK*
10:30 – 10:55	*Local Illumination for Volumes*
10:55 – 11:20	*Transfer Function Design: Classification*
10:20 – 10:45	*Transfer Function Design: Optical Properties*
11:45 – 12:15	*Pre-Integration and High-Quality Filtering*
12:15 – 1:45	*LUNCH BREAK*

Course 28 - Afternoon

1:45 – 2:30	*Atmospheric Effects, Participating Media*
2:30 – 3:00	*High-Quality Volume Clipping*
3:00 – 3:30	*NPR and Segmented Volumes*
3:30 – 3:45	*BREAK*
3:45 – 4:15	*Volume Deformation & Animation*
4:15 – 4:45	*Dealing with Large Volumes*
4:45 – 5:15	*Rendering from Difficult Data Formats*
5:15 – 5:30	*Q & A*

GPU-based Volume Rendering

Klaus Engel
Siemens Corporate Research
Princeton

Markus Hadwiger
VR VIS Research Center
Vienna, Austria

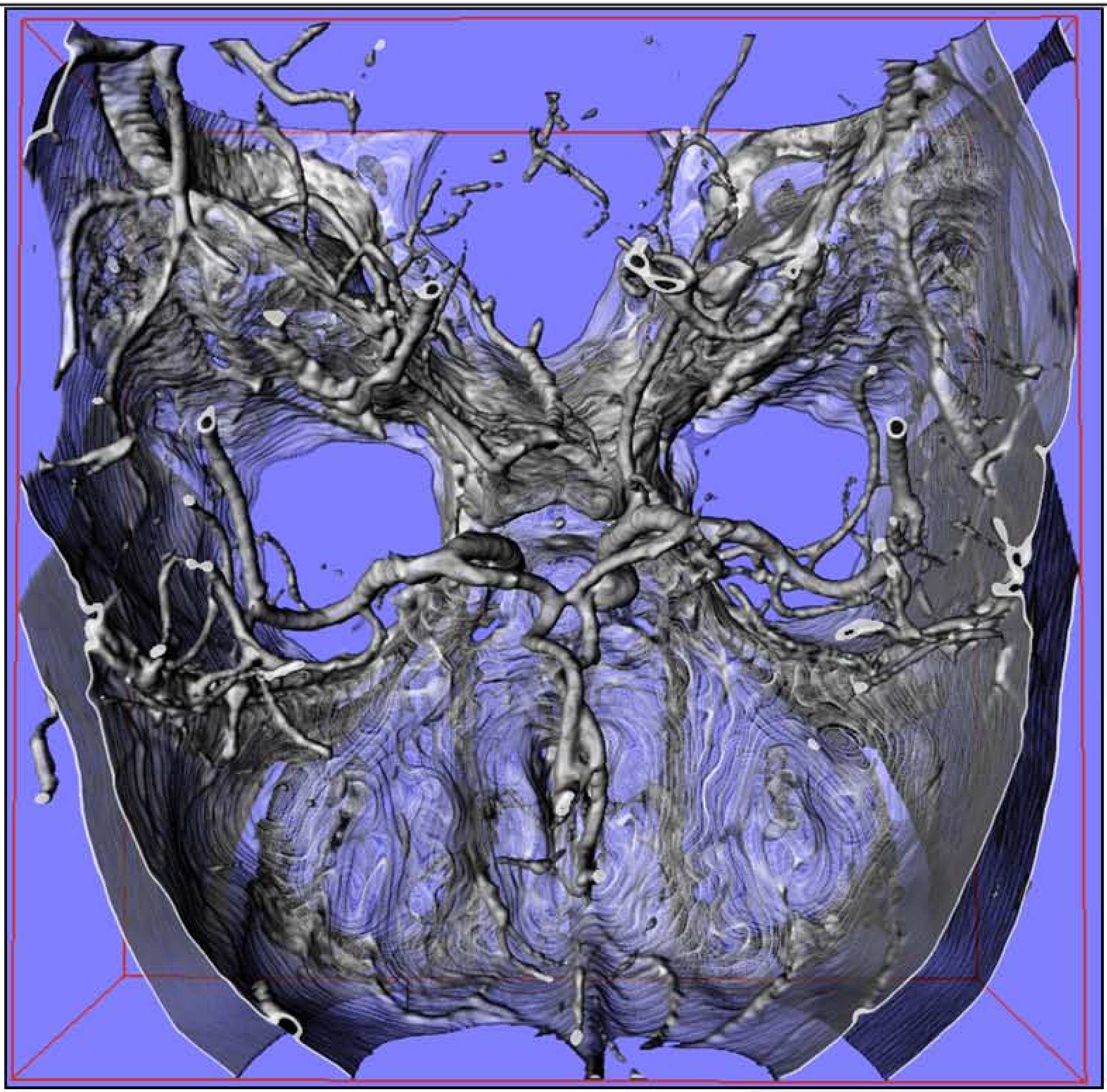
Joe M. Kniss
Scientific Computing and
Imaging Insitute,
University of Utah

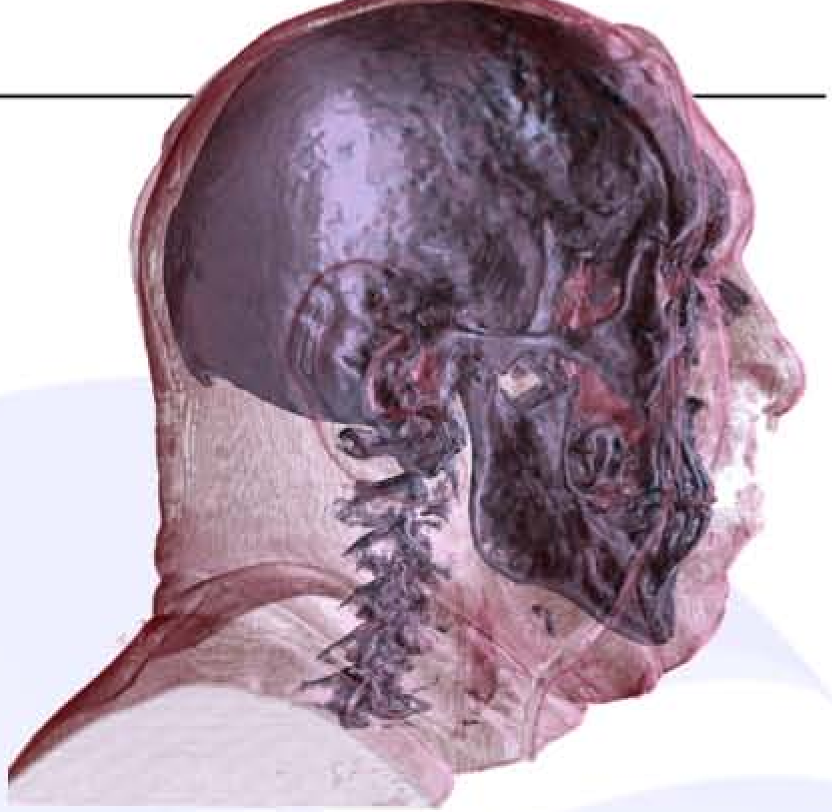
Aaron E. Lefohn
Institute for Data Analysis
and Visualization.
University of California, Davis

Christof Rezk Salama
Computer Graphics and
Multimedia Group
University of Siegen, Germany

Daniel Weiskopf
Visualization and Interactive
Systems Group,
University of Stuttgart, Germany

Appliations: Medicine



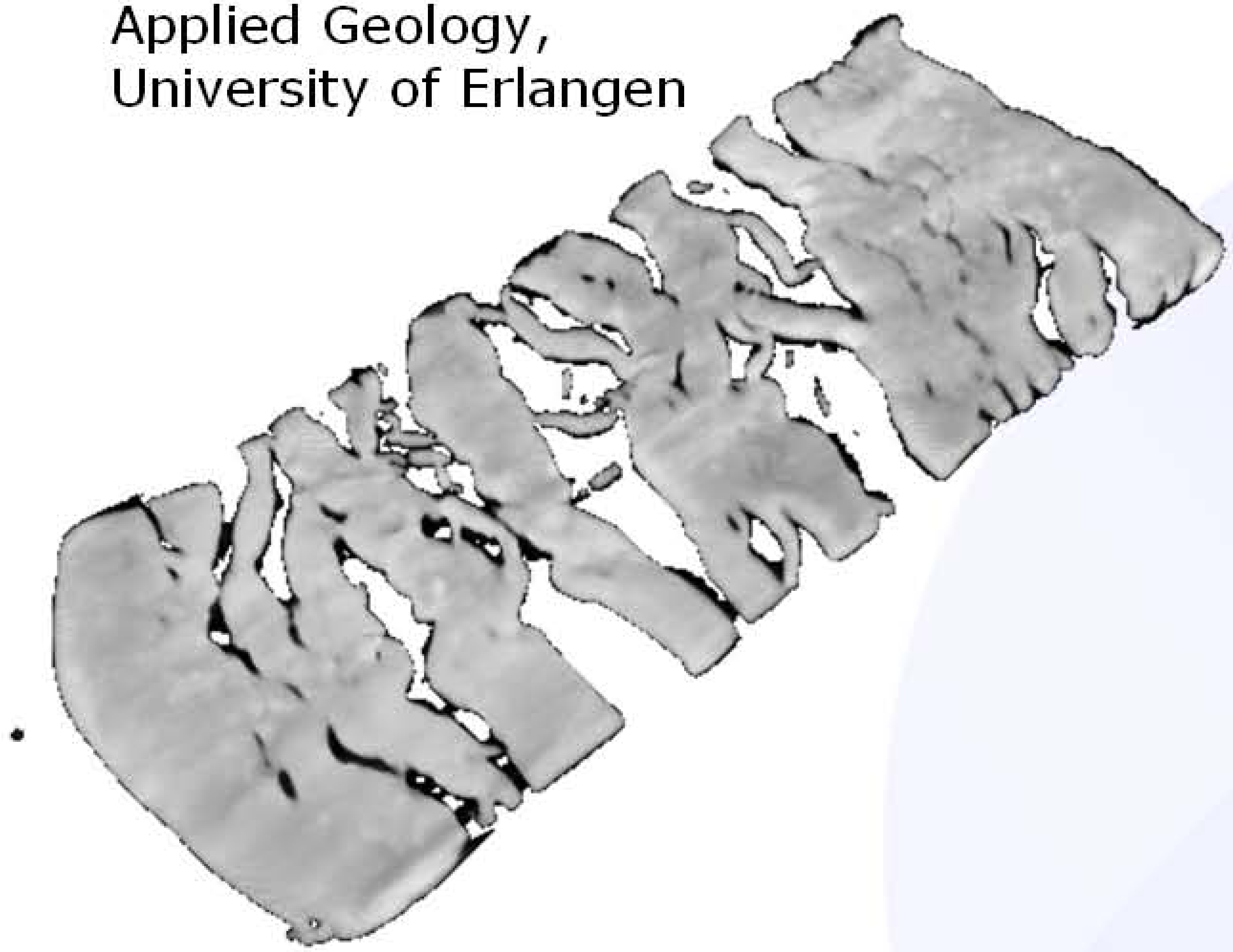


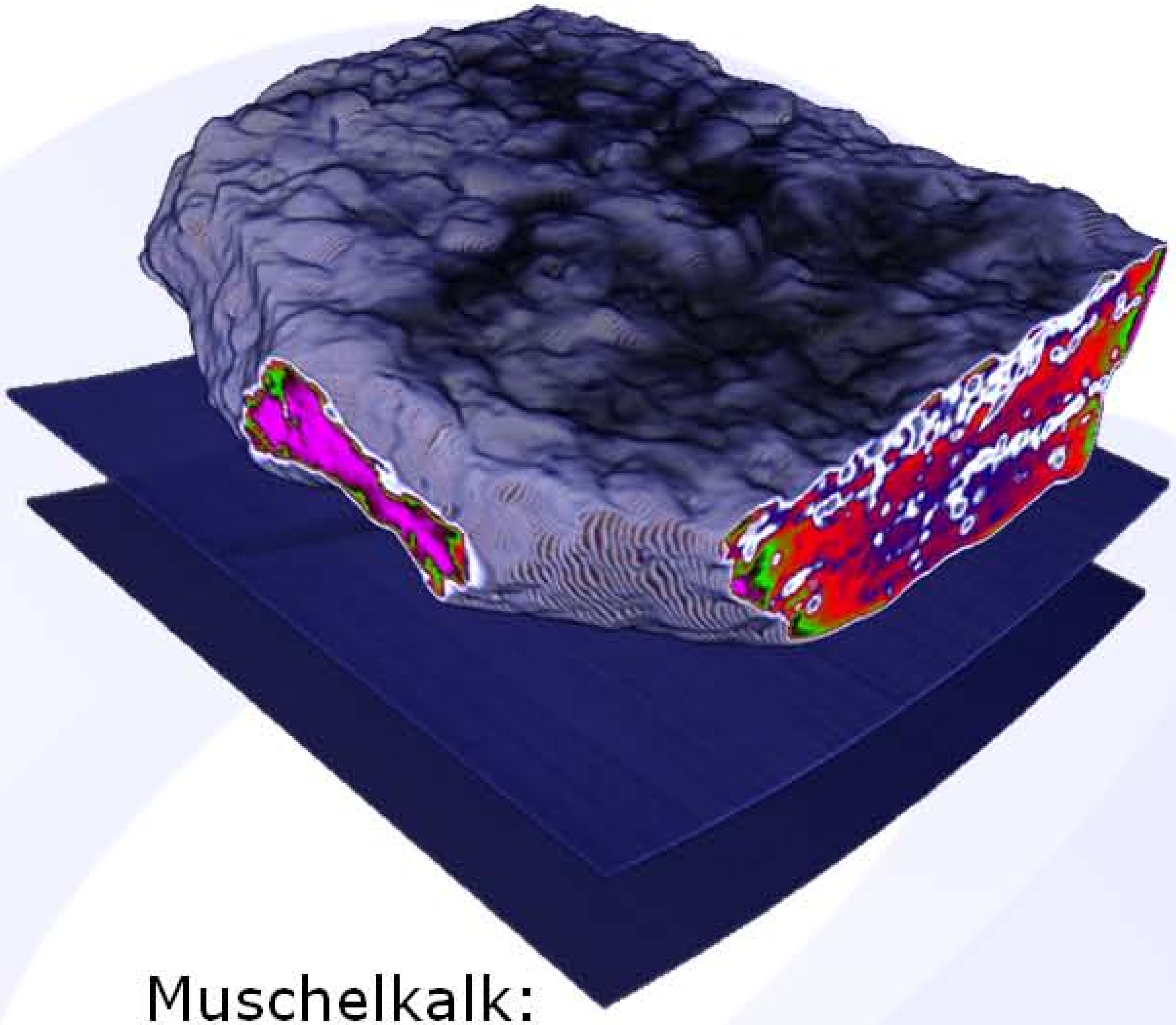
CT Human Head:
Visible Human Project, US National Library of Medicine, Maryland, USA

CT Angiography:
Dept. of Neuroradiology University of Erlangen, Germany

Applications: Geology

Deformed Plasticine Model,
Applied Geology,
University of Erlangen





Muschelkalk:
Paläontologie,
Virtual Reality Group,
University of Erlangen

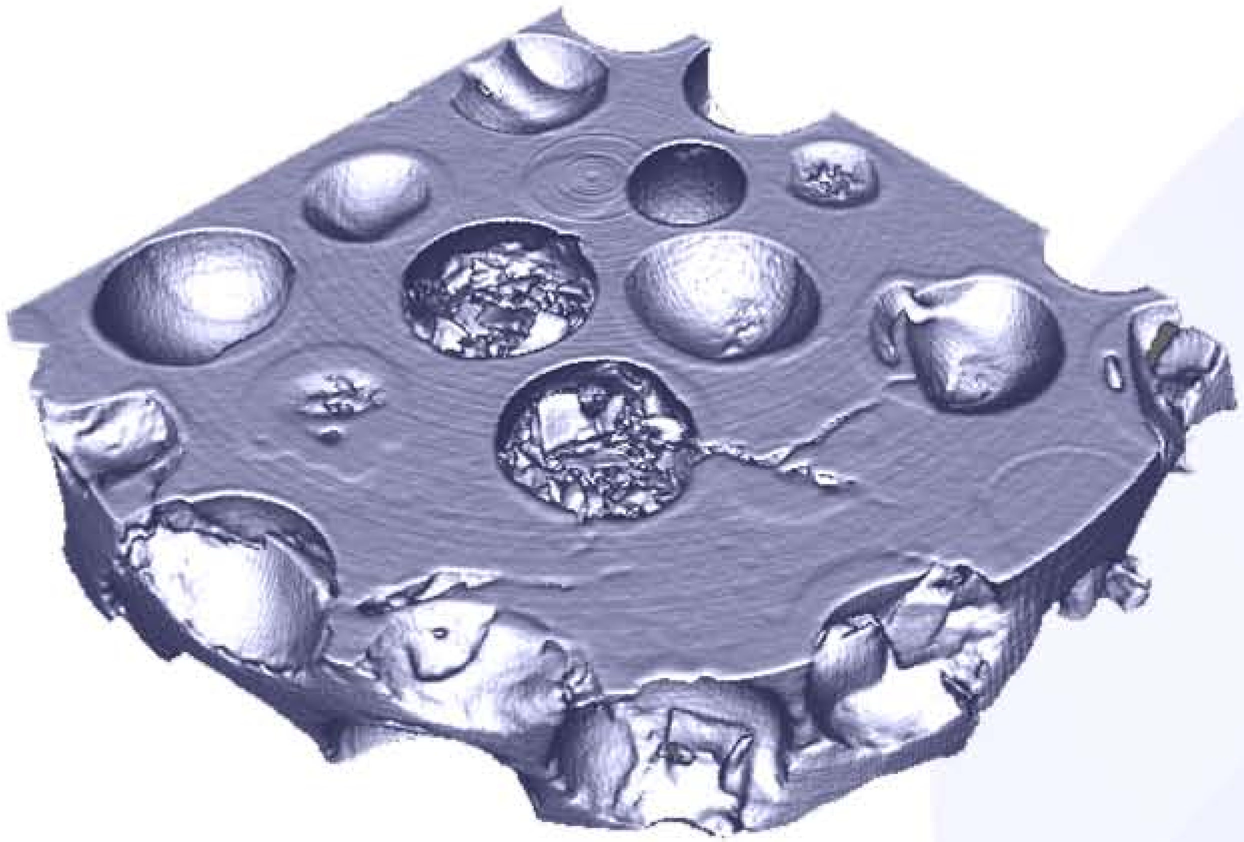
Applications: Archeology

Hellenic Statue of Isis
3rd century B.C.
ARTIS, University of Erlangen-Nuremberg, Germany

Sotades Pygmaios Statue,
5th century B.C
ARTIS, University of Erlangen-Nuremberg, Germany

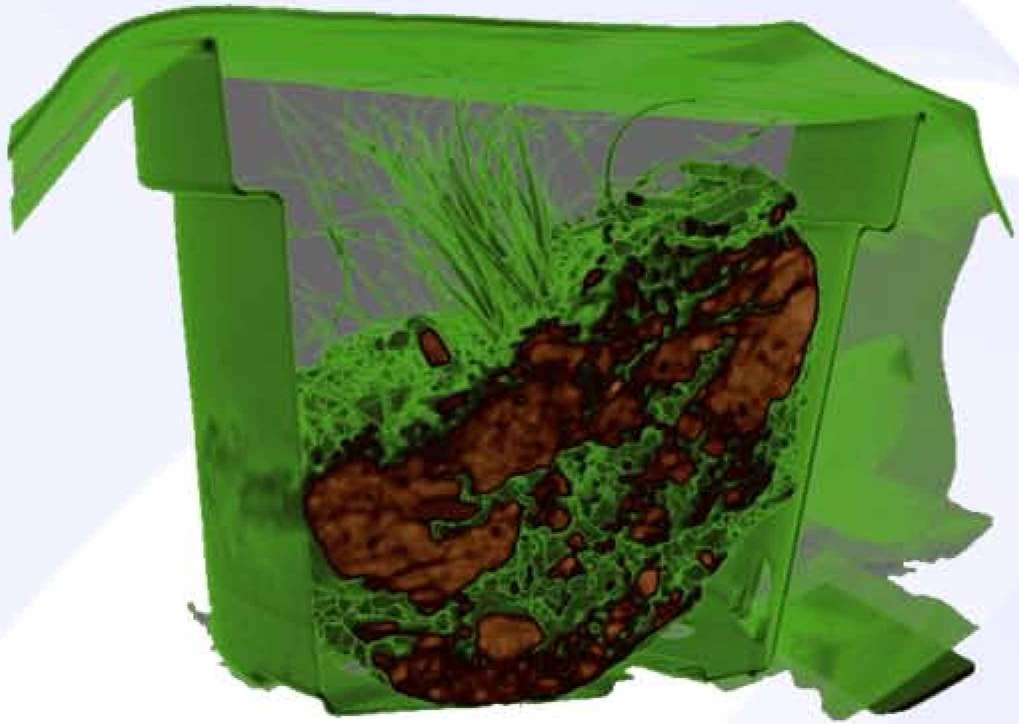
Applications:

Material Science,
Quality Control



Micro CT, Compound Material,
Material Science Department, University of Erlangen

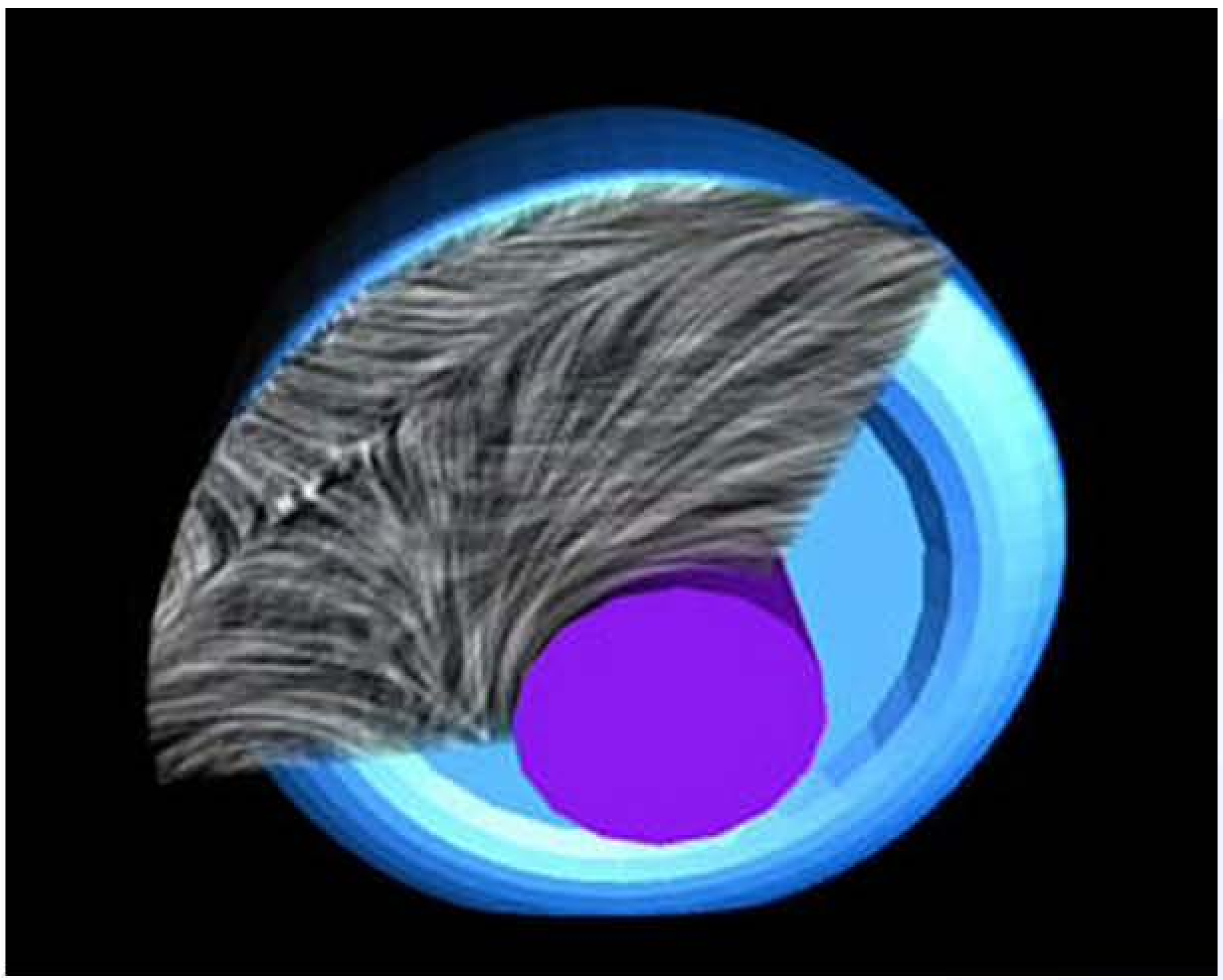
Biology

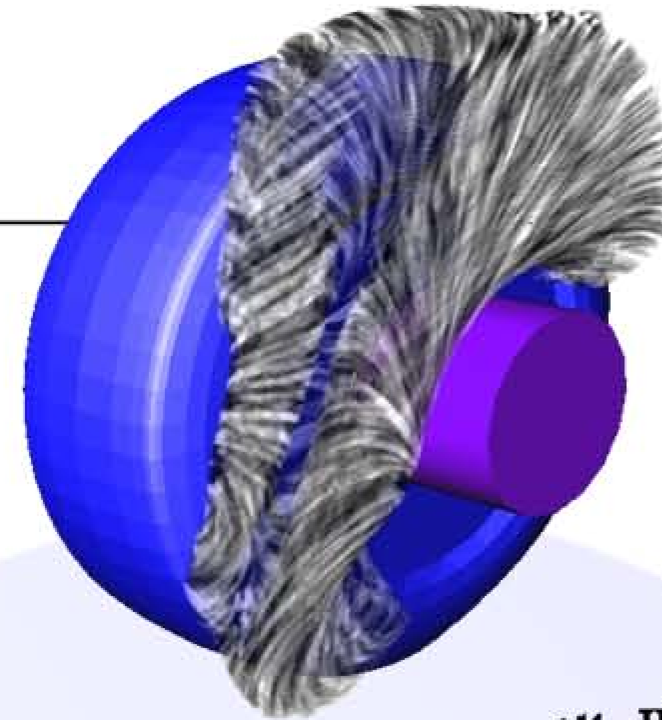


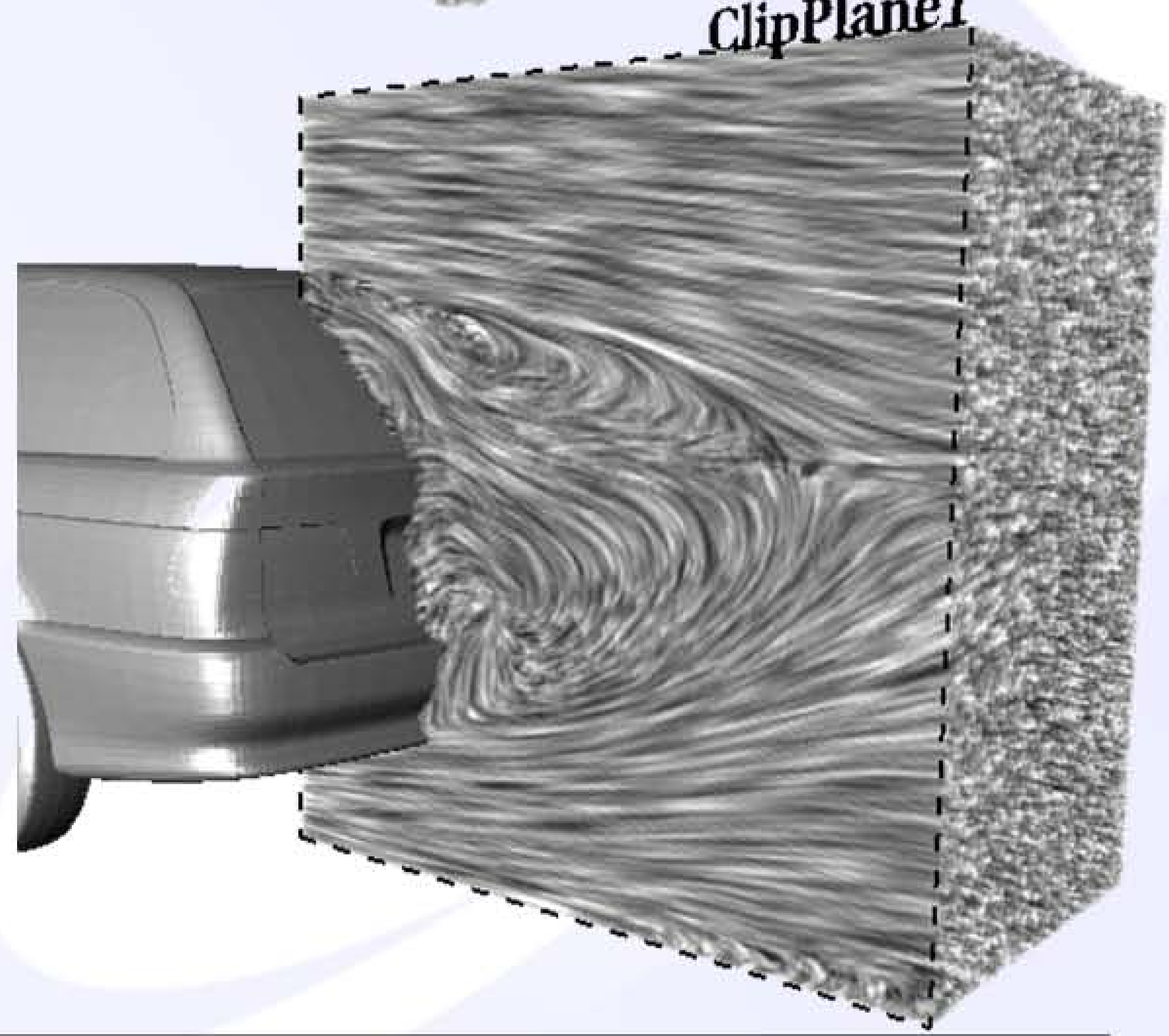
biological sample of the soil, CT,
Virtual Reality Group, University if Erlangen

Applications

Computational Science and Engineering

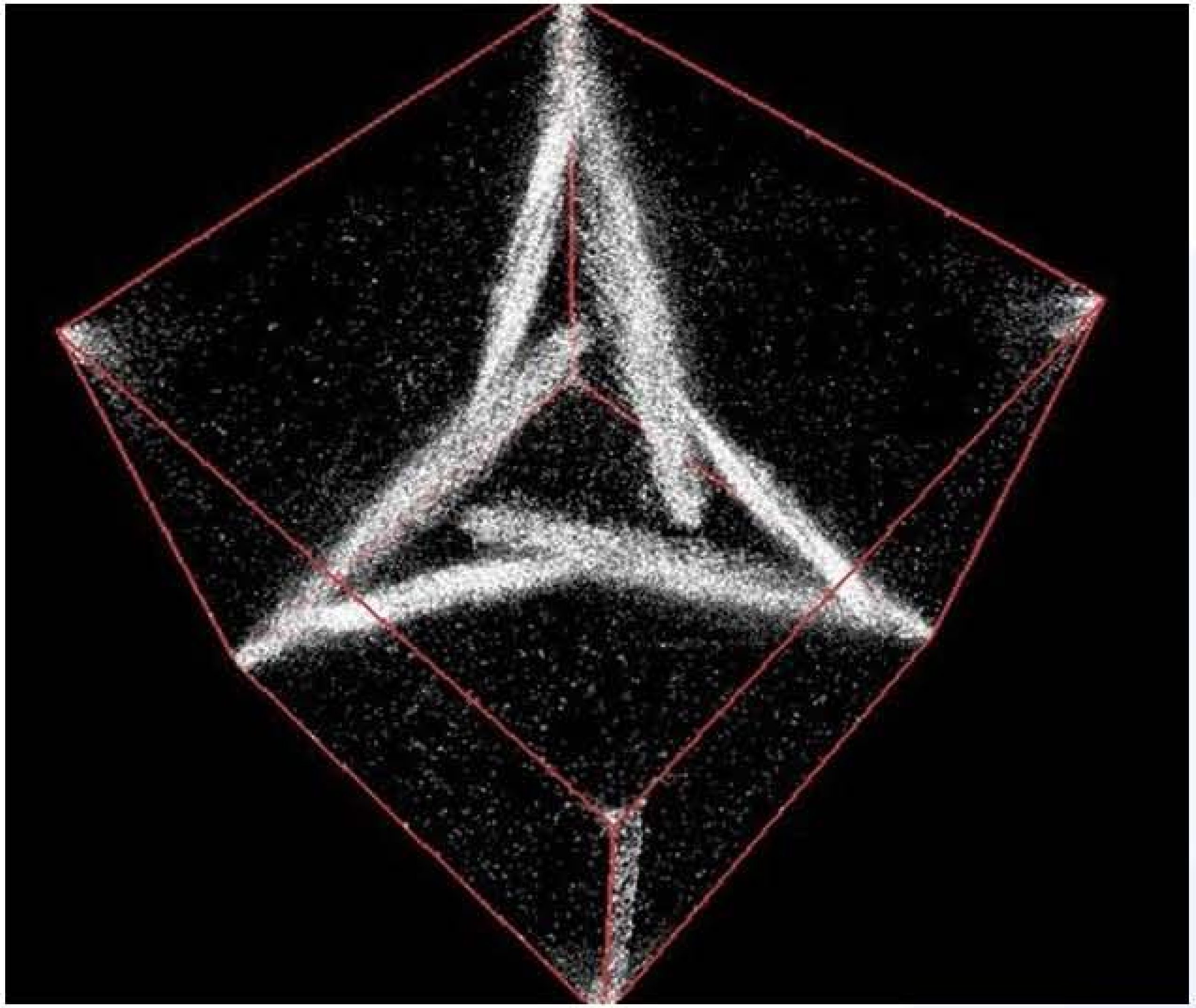


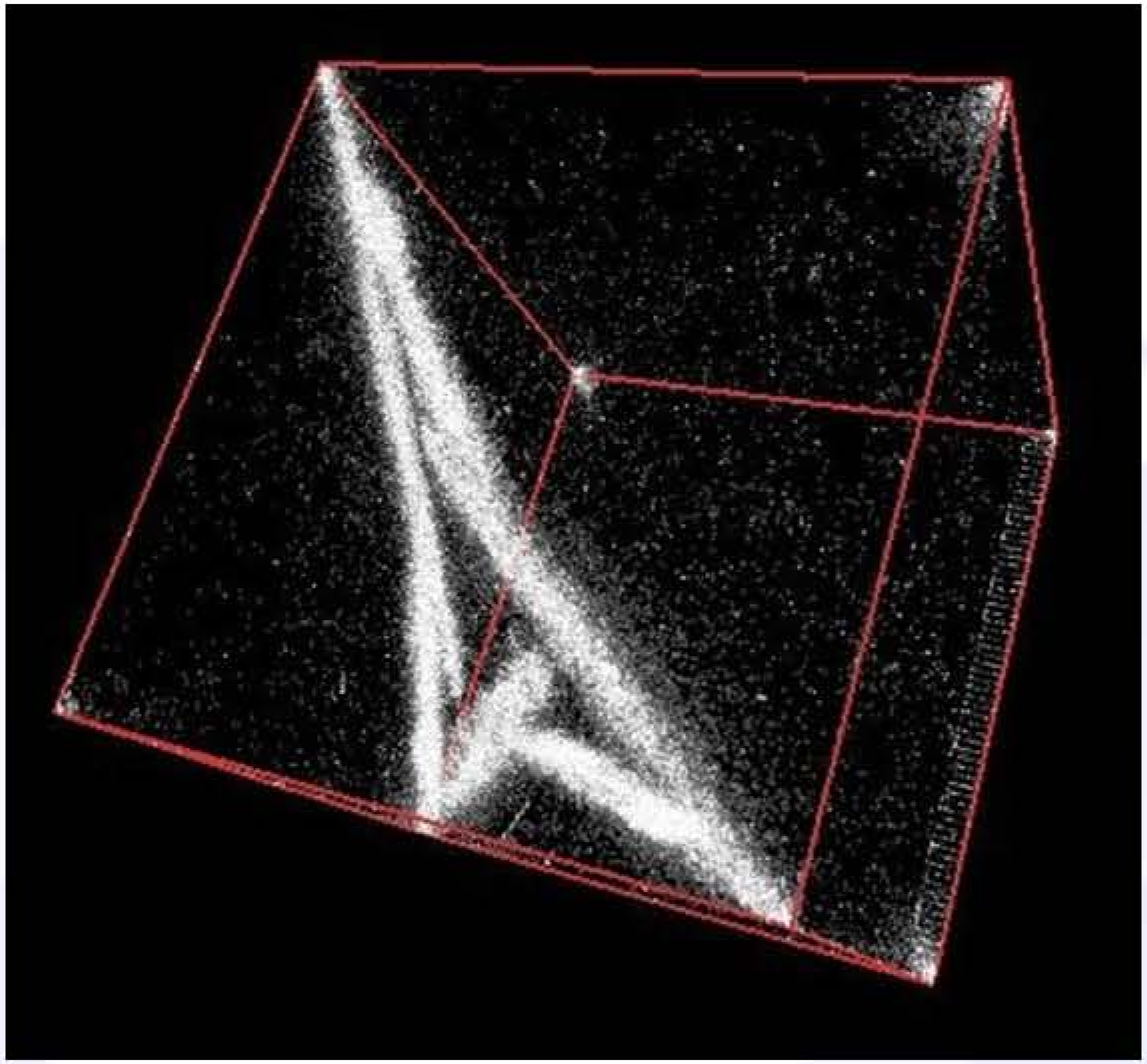




Applications: Computer Science

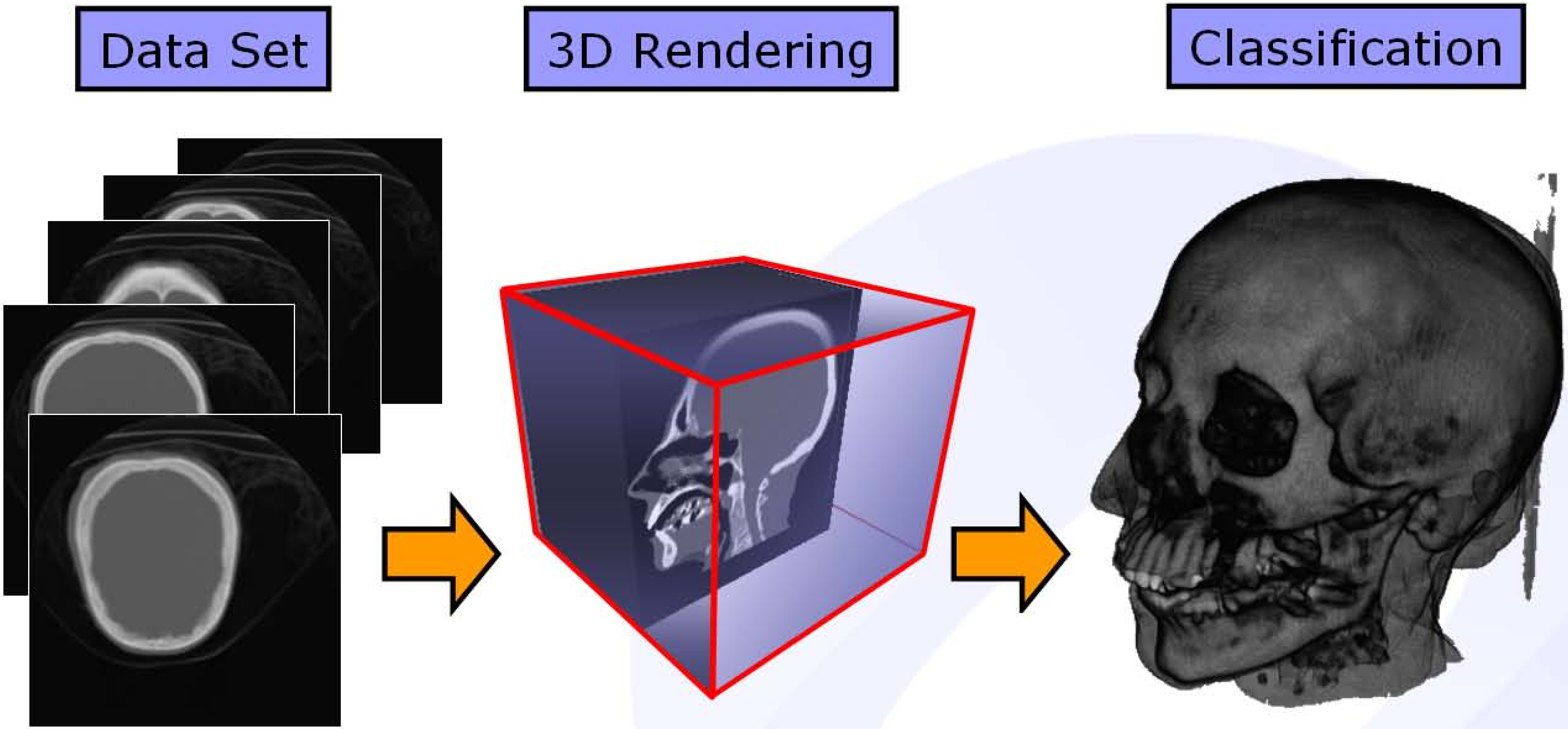
- Visualization of Pseudo Random Numbers





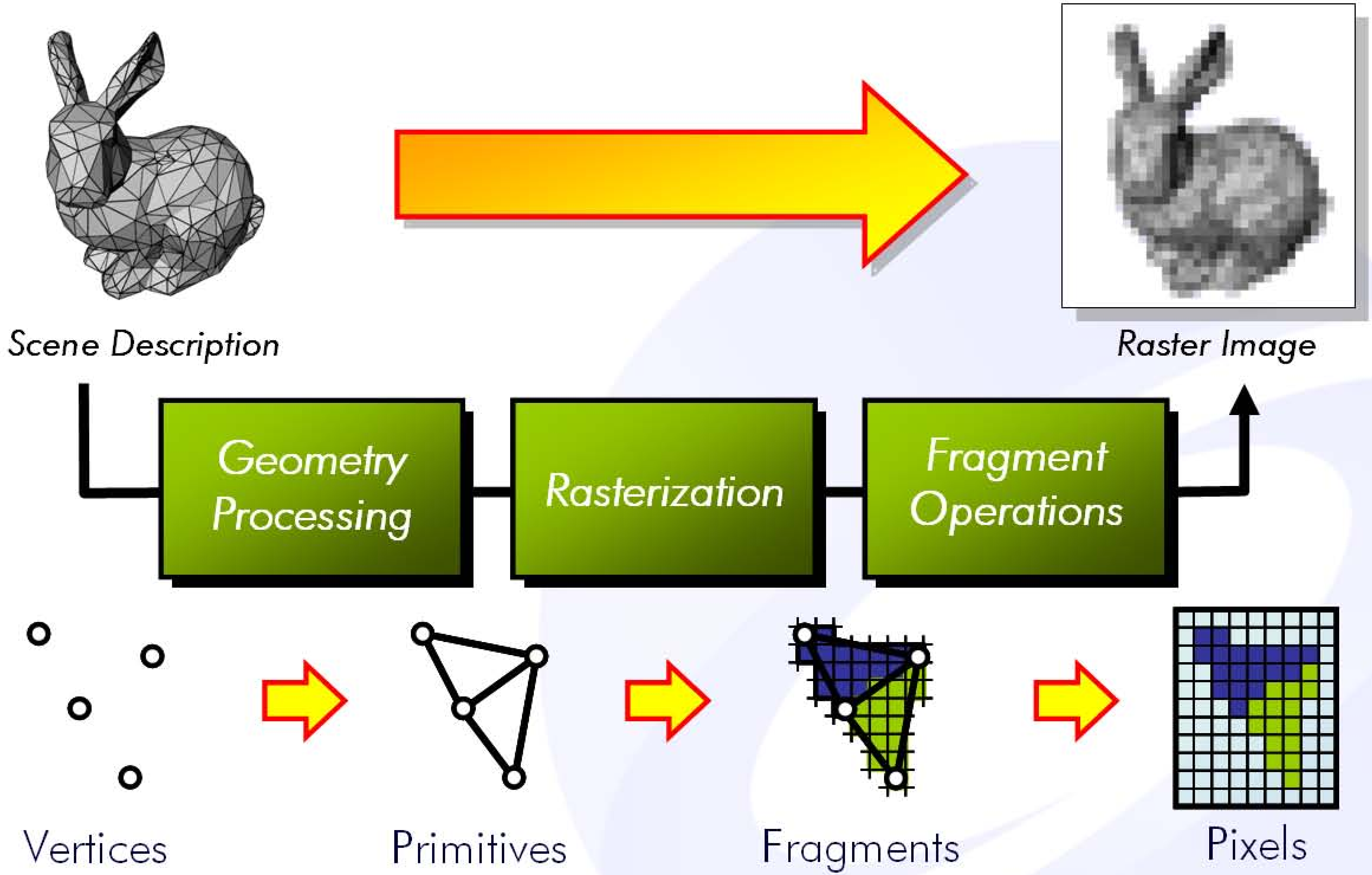
Entropy of Pseudo Random Numbers,
Dan Kaminsky, Doxpara Research, USA,
www.doxpara.com

Outline

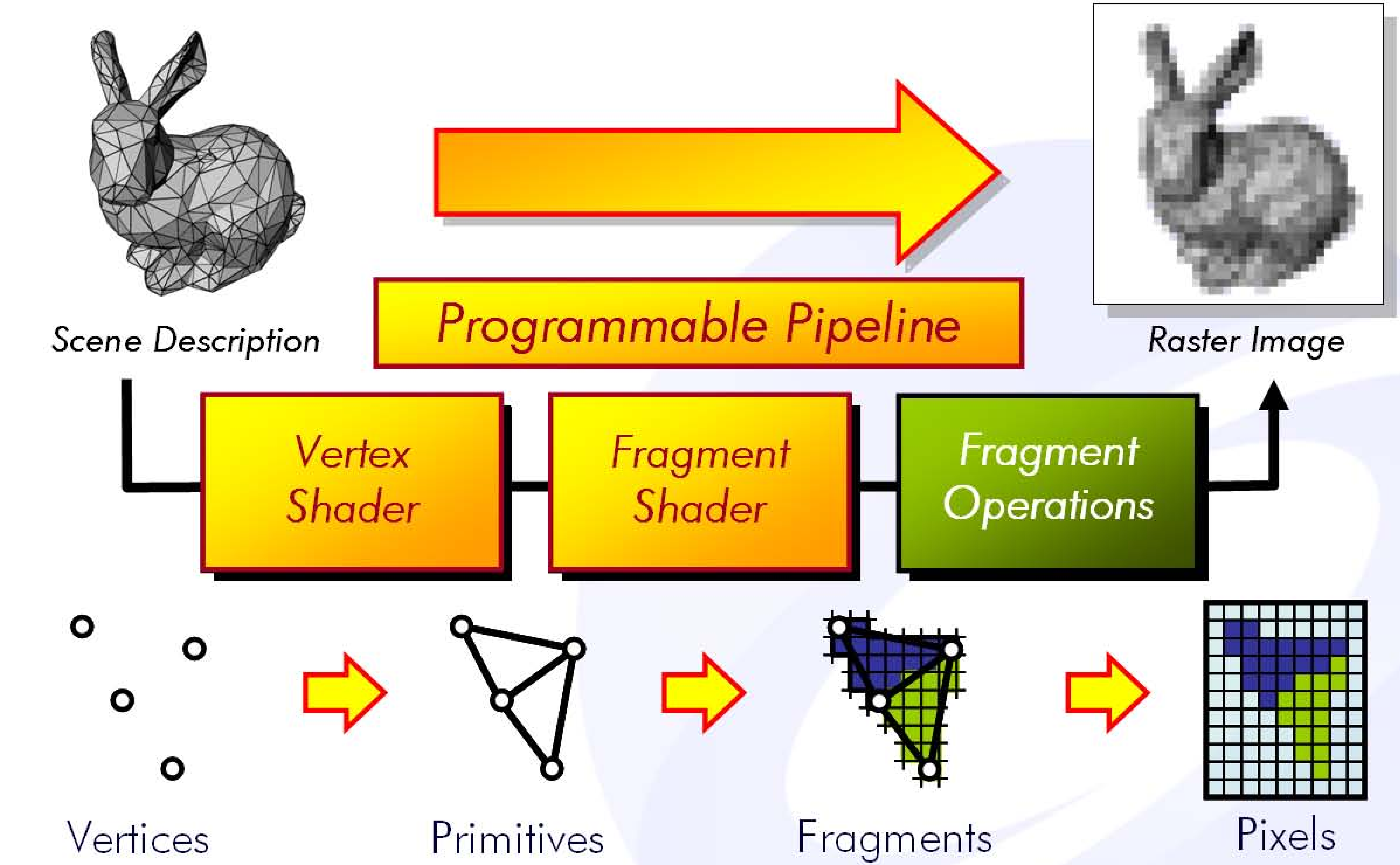


- in real-time on commodity graphics hardware

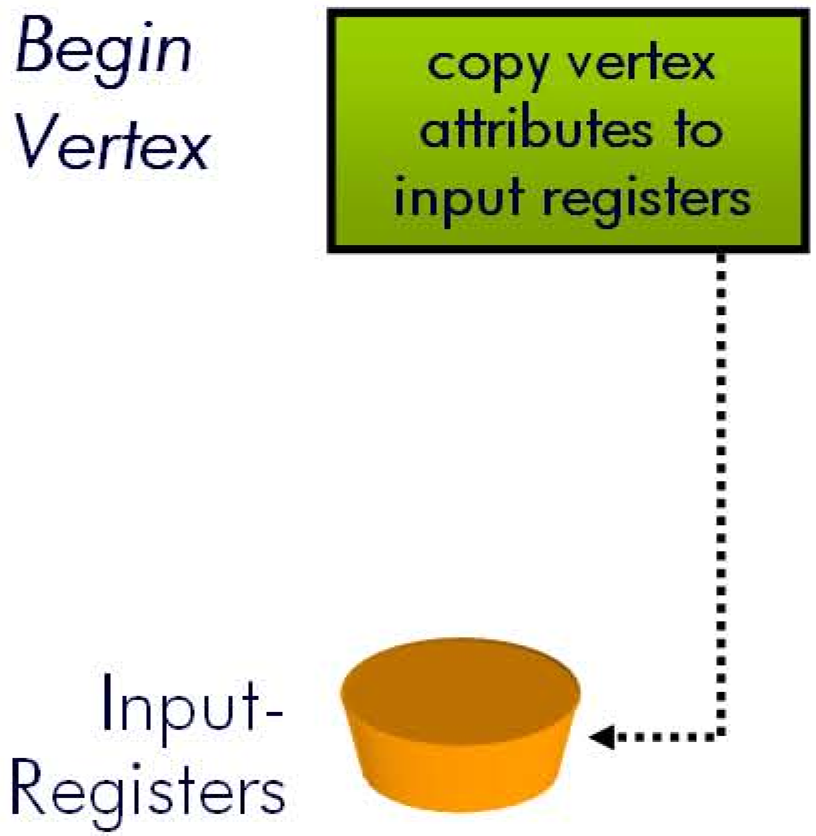
Graphics Hardware



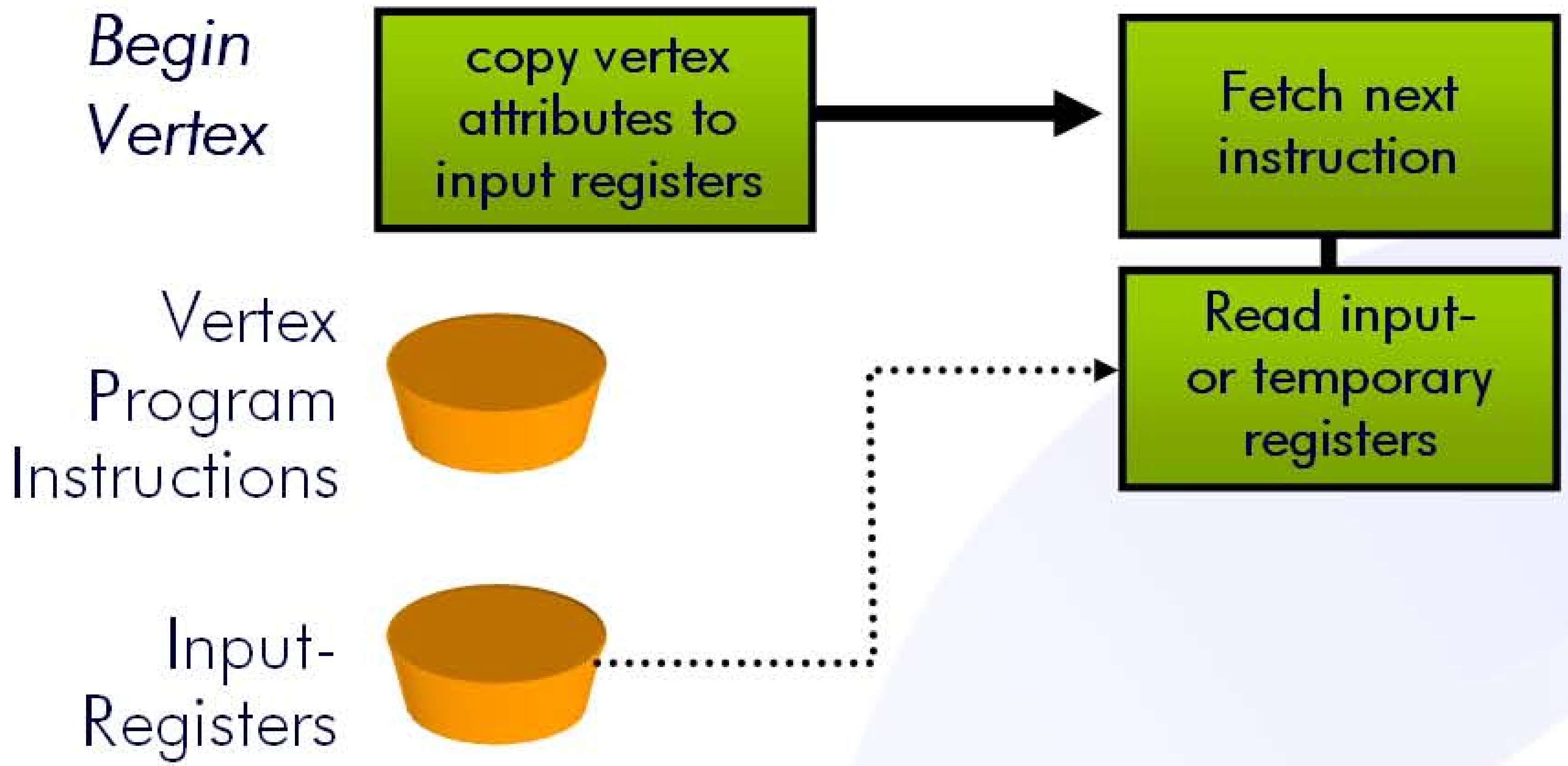
Graphics Hardware



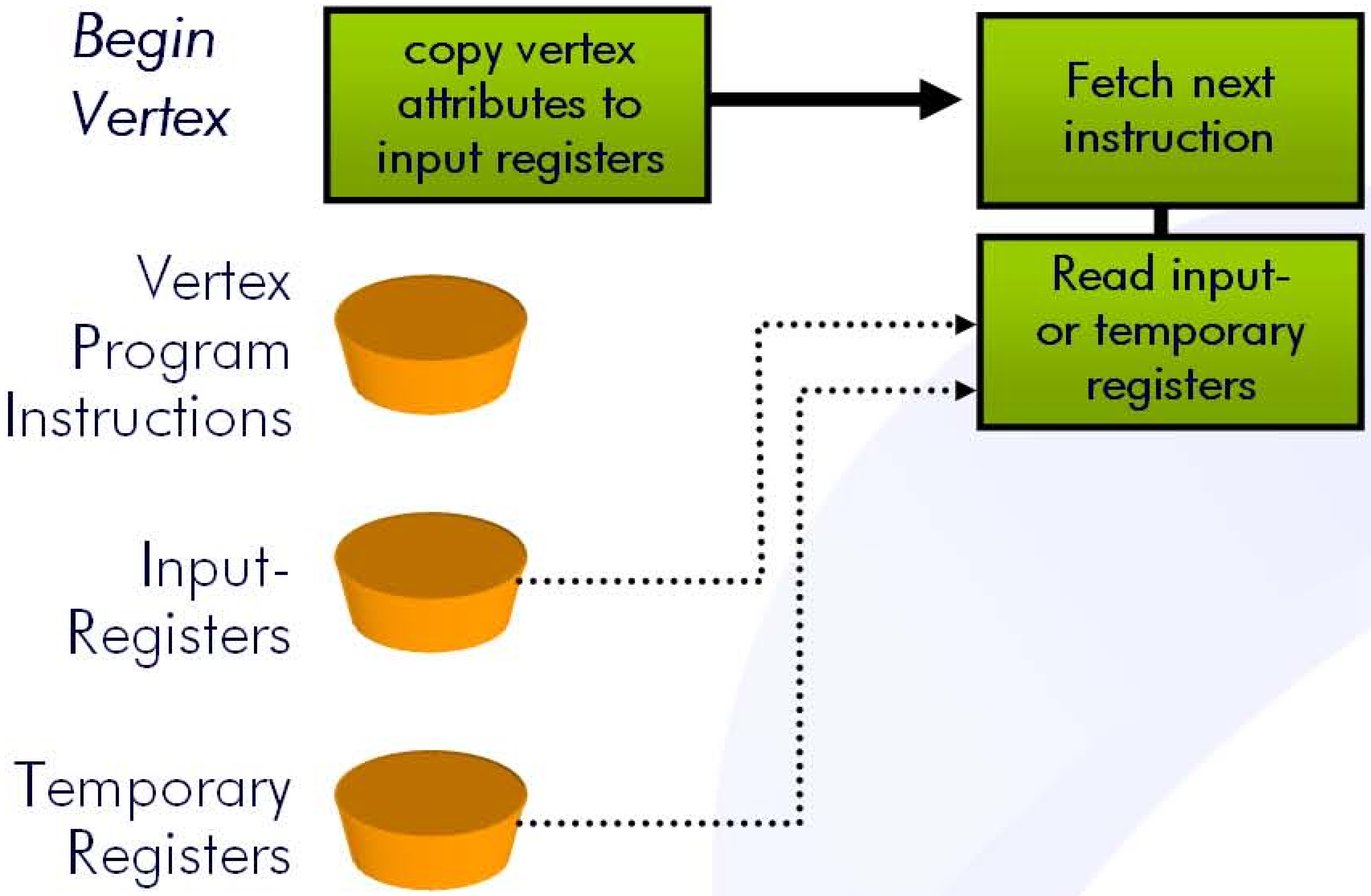
Programmable Vertex Processor



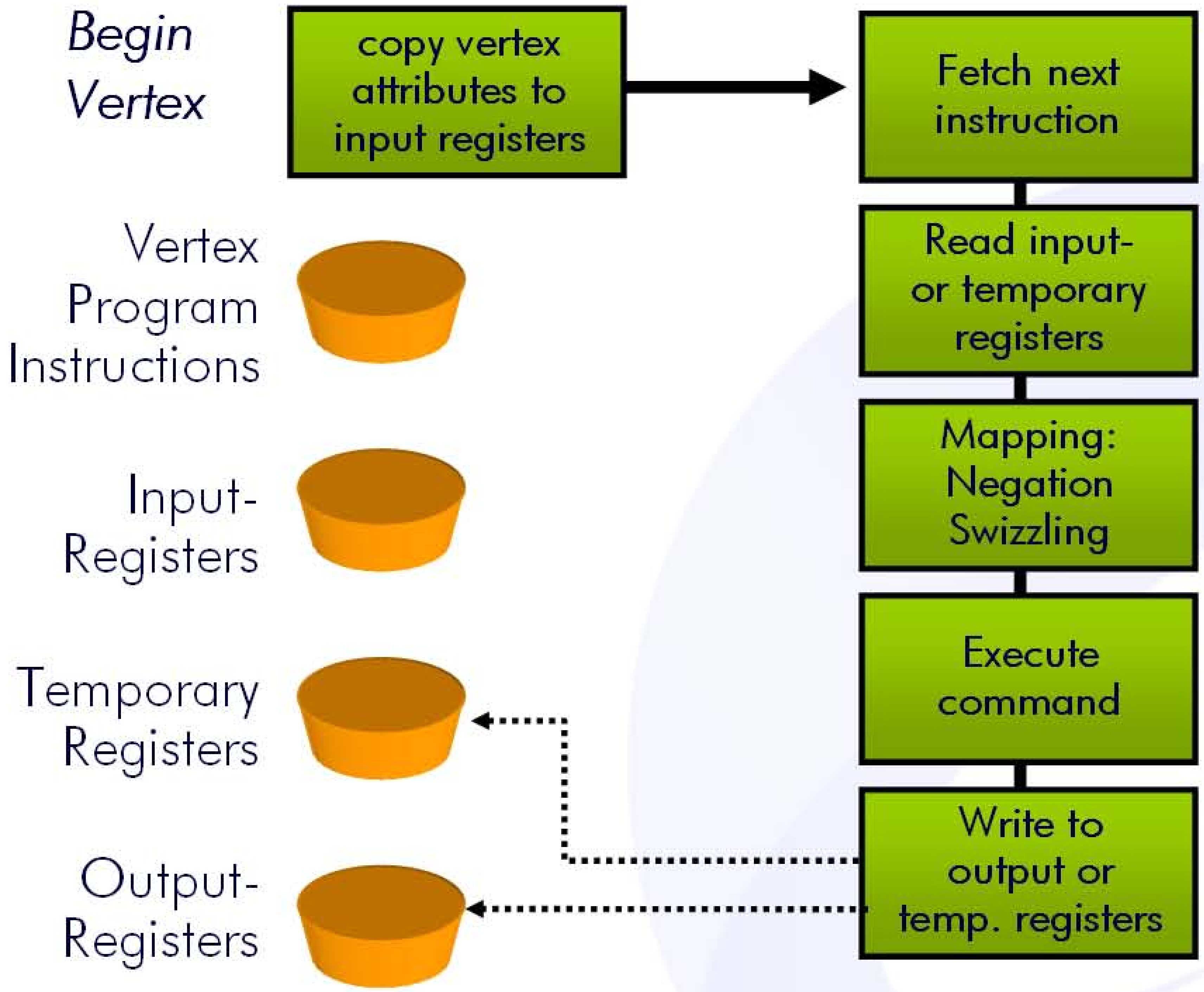
Programmable Vertex Processor



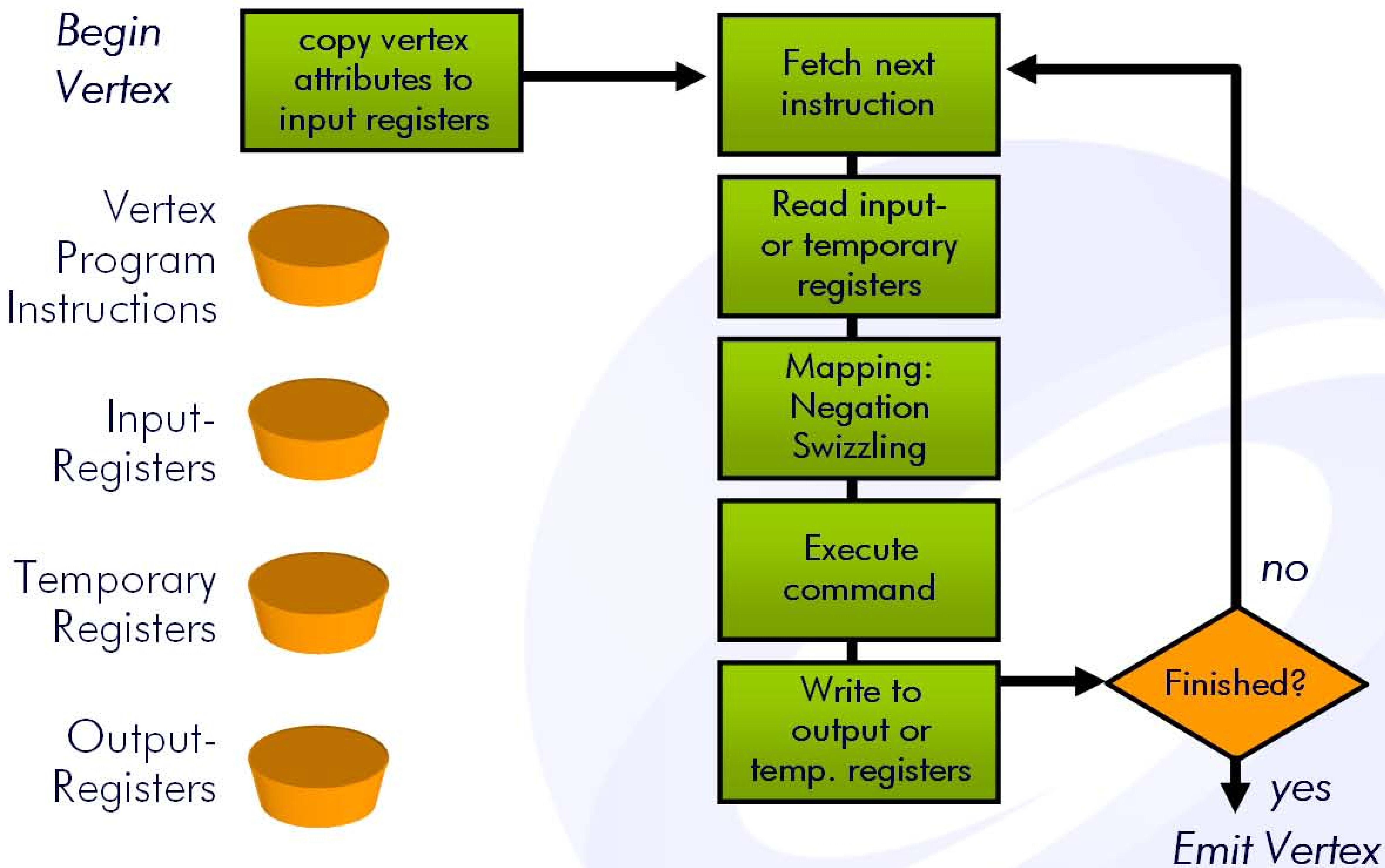
Programmable Vertex Processor



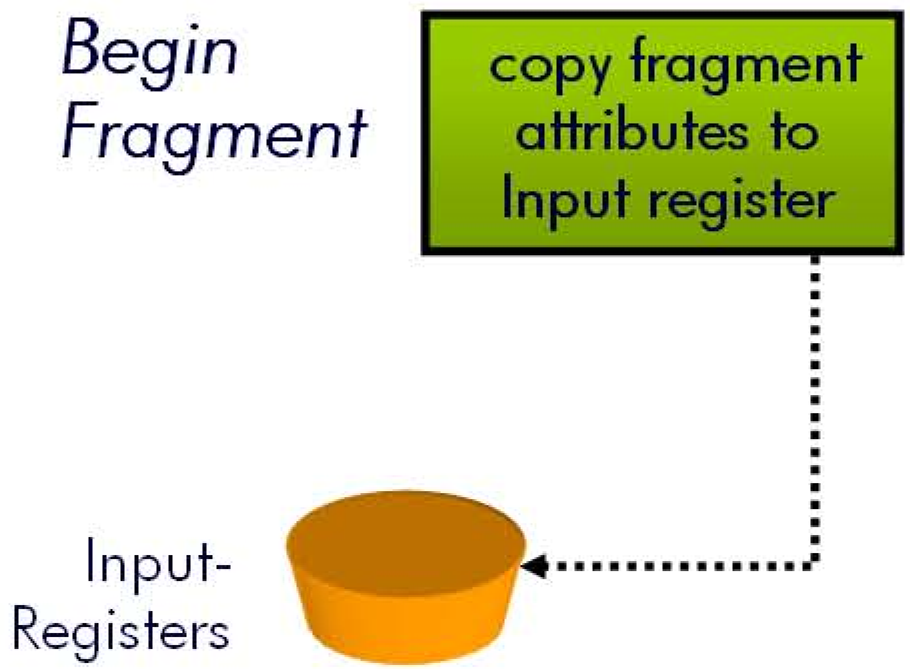
Programmable Vertex Processor



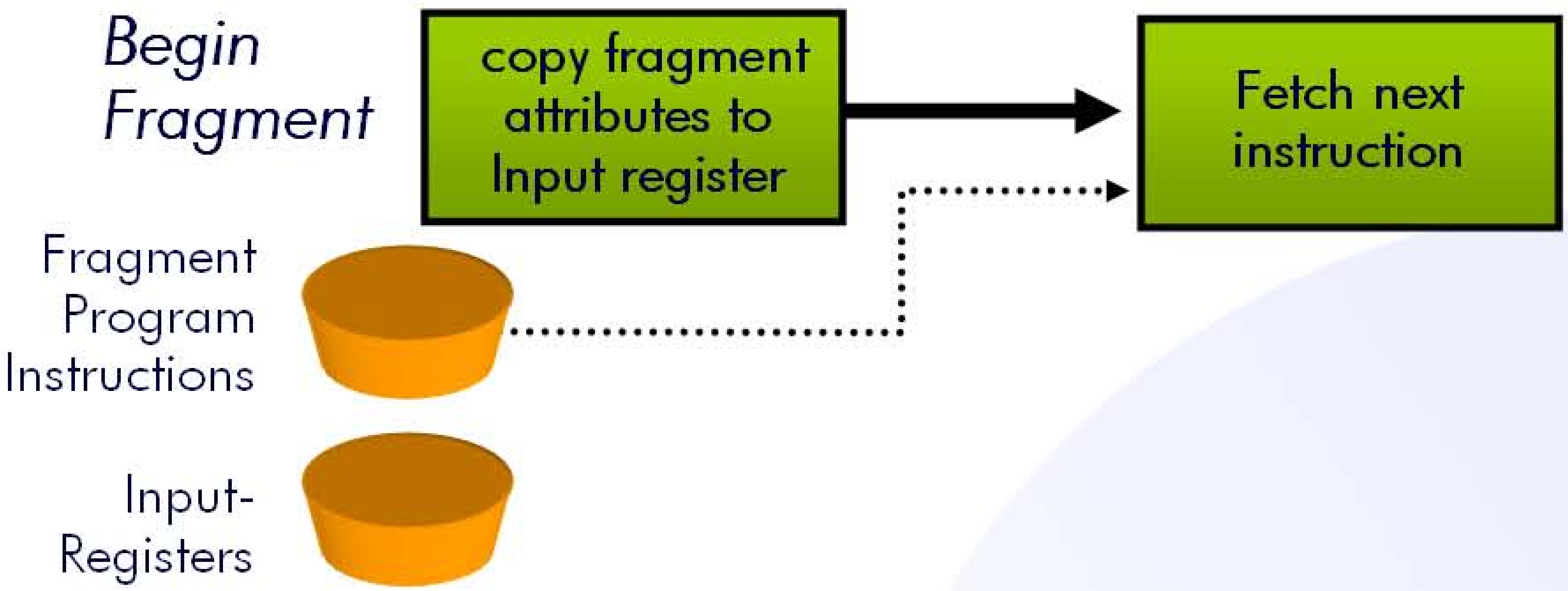
Programmable Vertex Processor



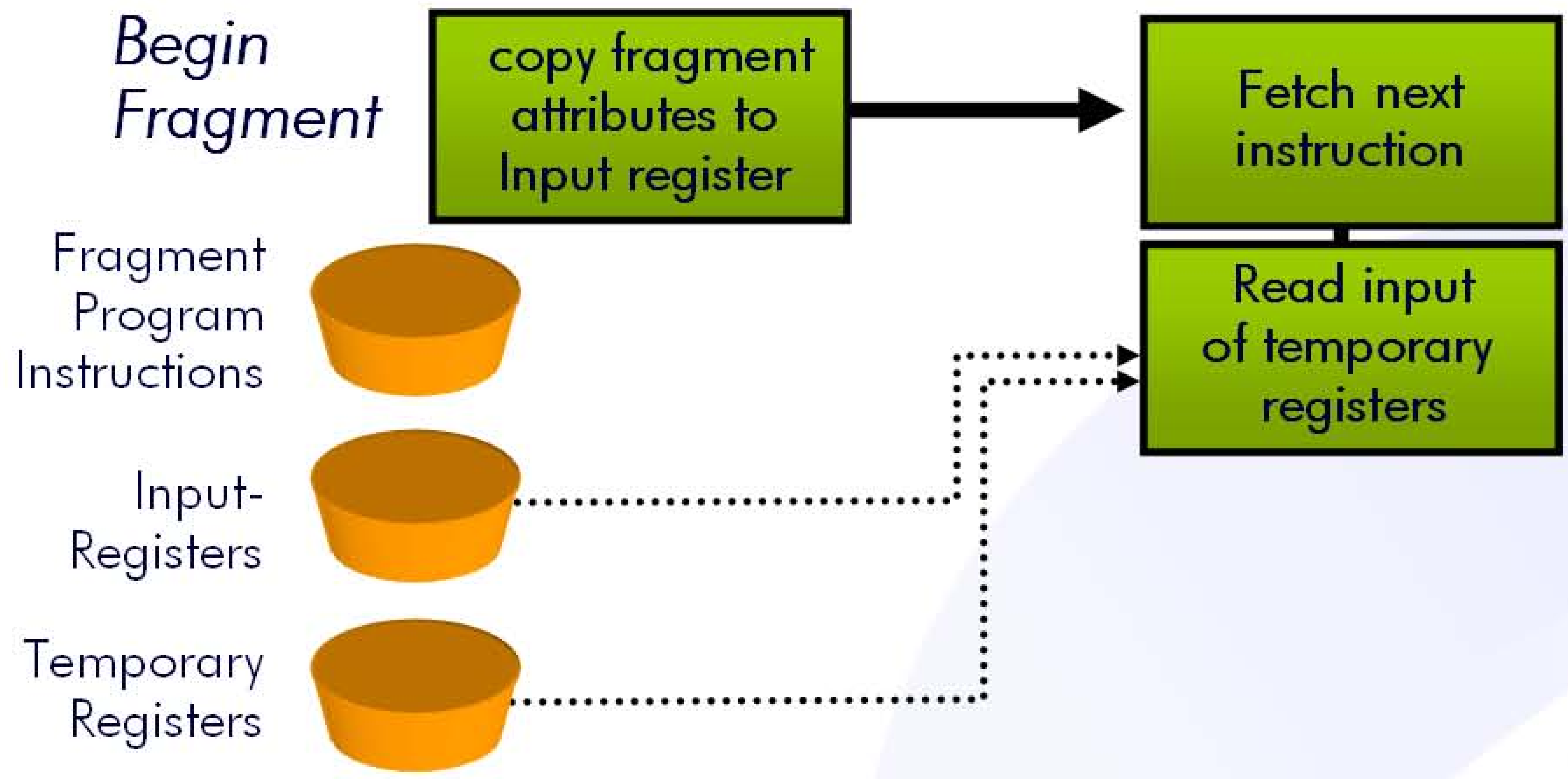
Fragment Processor



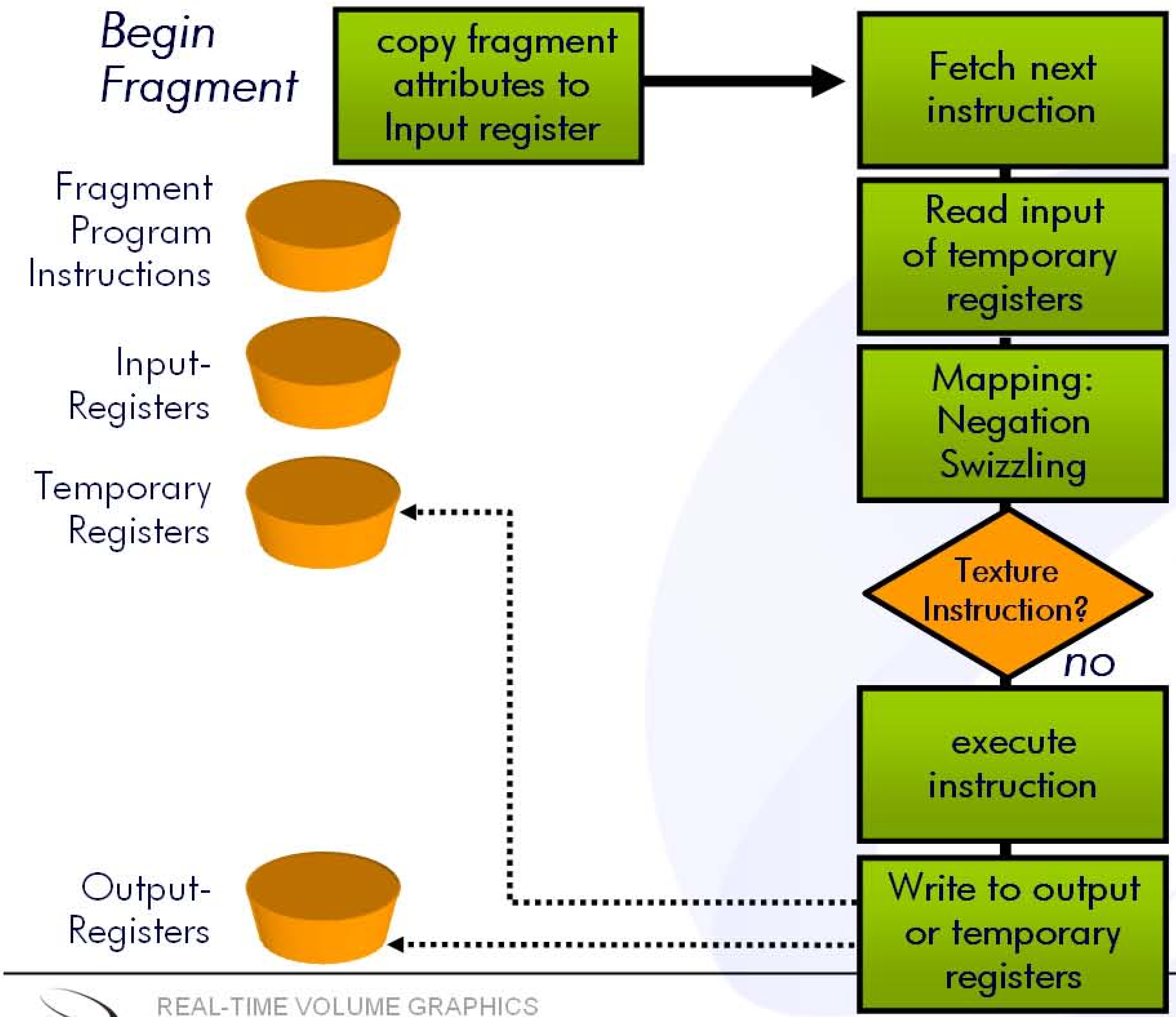
Fragment Processor



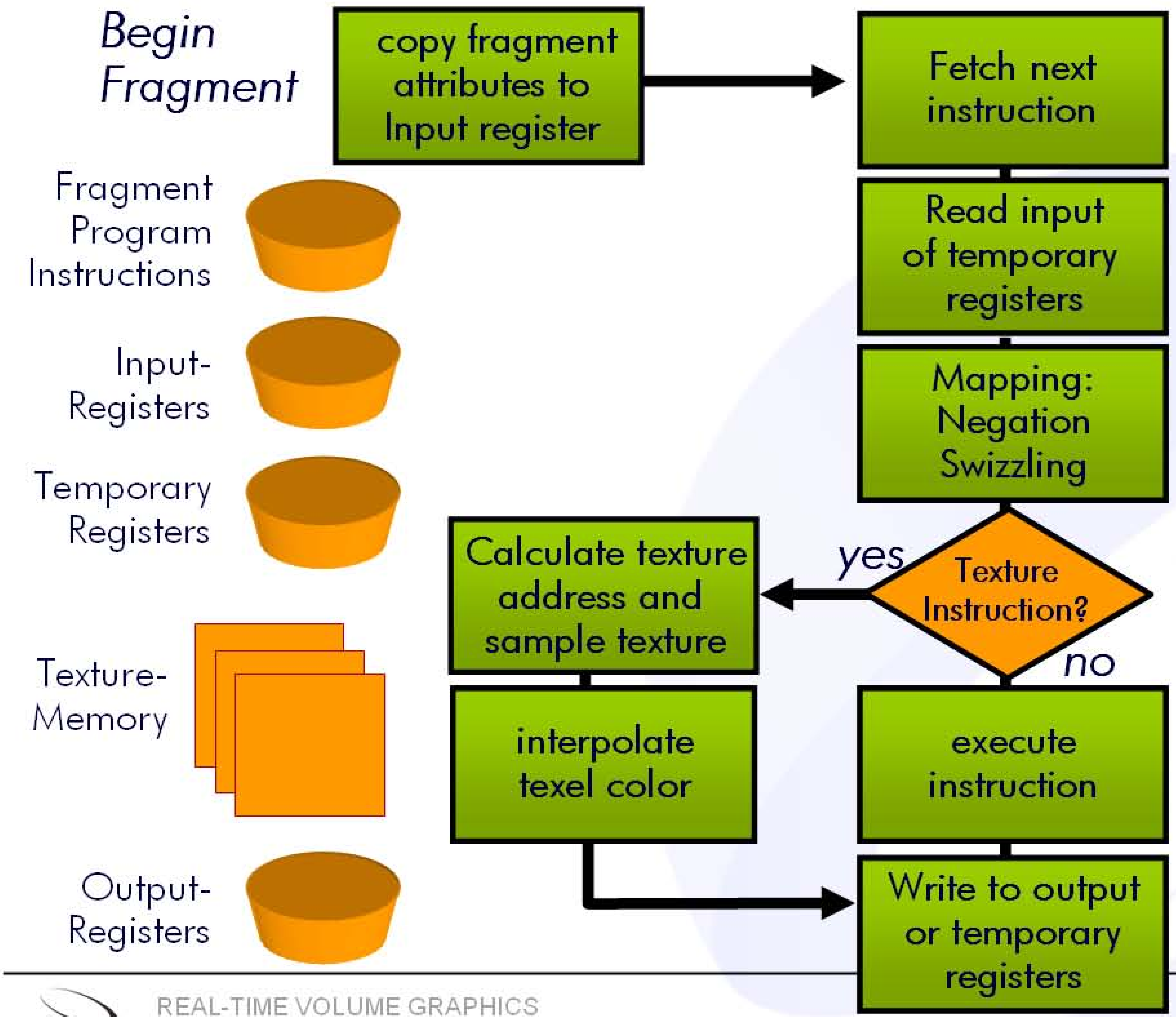
Fragment Processor



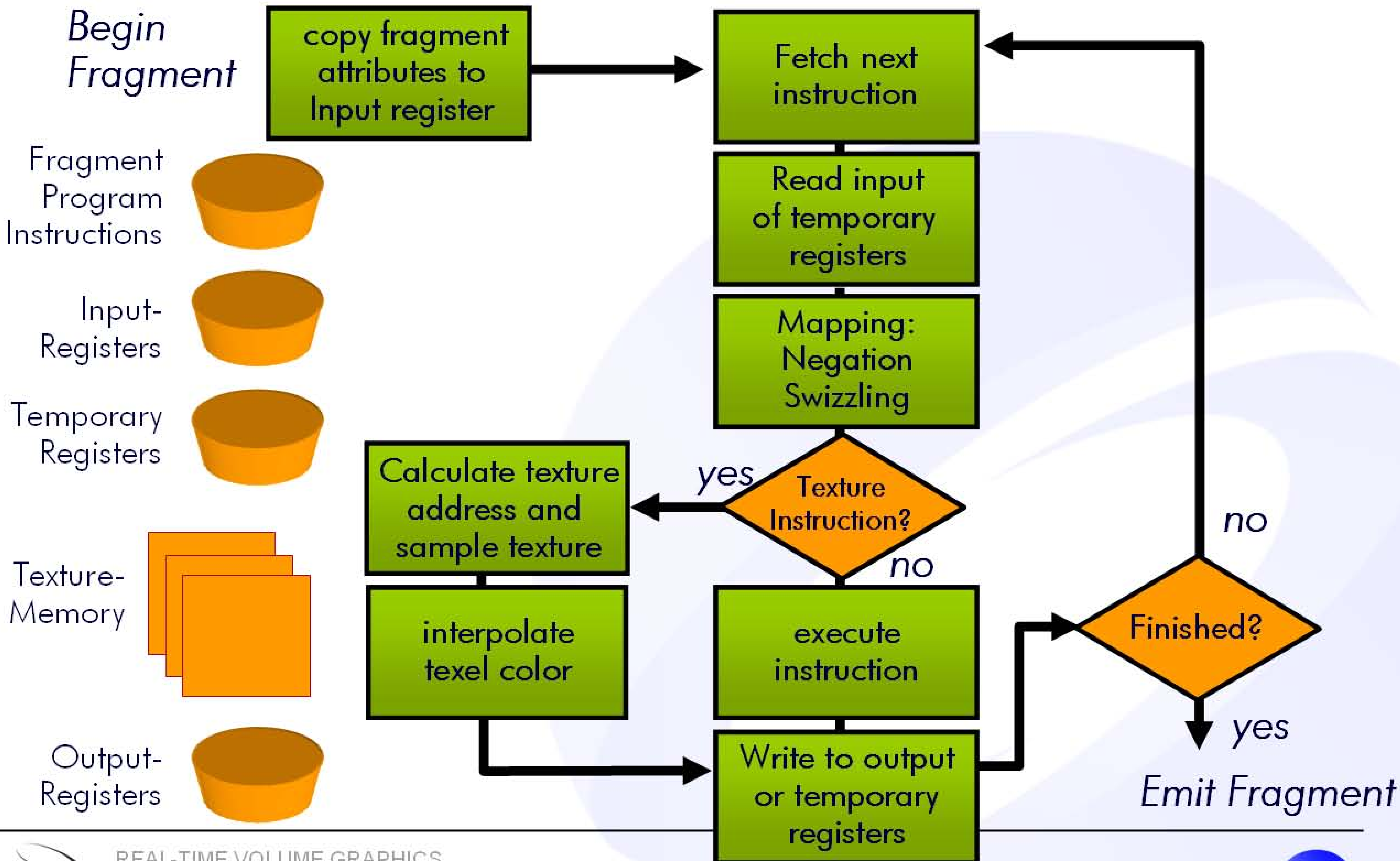
Fragment Processor



Fragment Processor



Fragment Processor



Phong Shading

- *Per-Pixel Lighting:* Local illumination in a fragement shader

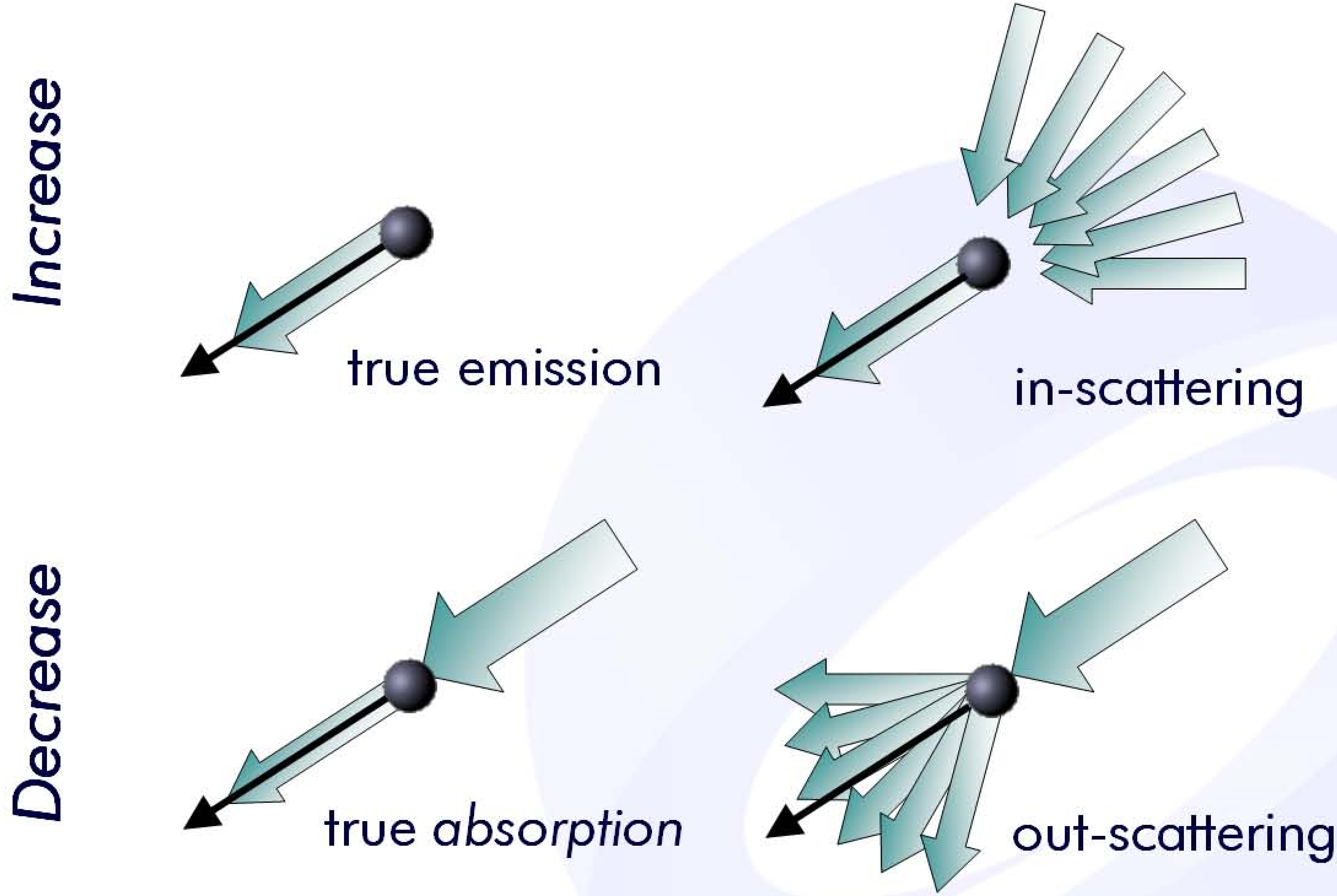
```
void main(float4 position  : TEXCOORD0,
          float3 normal    : TEXCOORD1,

      out float4 oColor      : COLOR,

    uniform float3 ambientCol,
    uniform float3 lightCol,
    uniform float3 lightPos,
    uniform float3 eyePos,
    uniform float3 Ka,
    uniform float3 Kd,
    uniform float3 Ks,
    uniform float  shiny)
{
```

Phong Shading

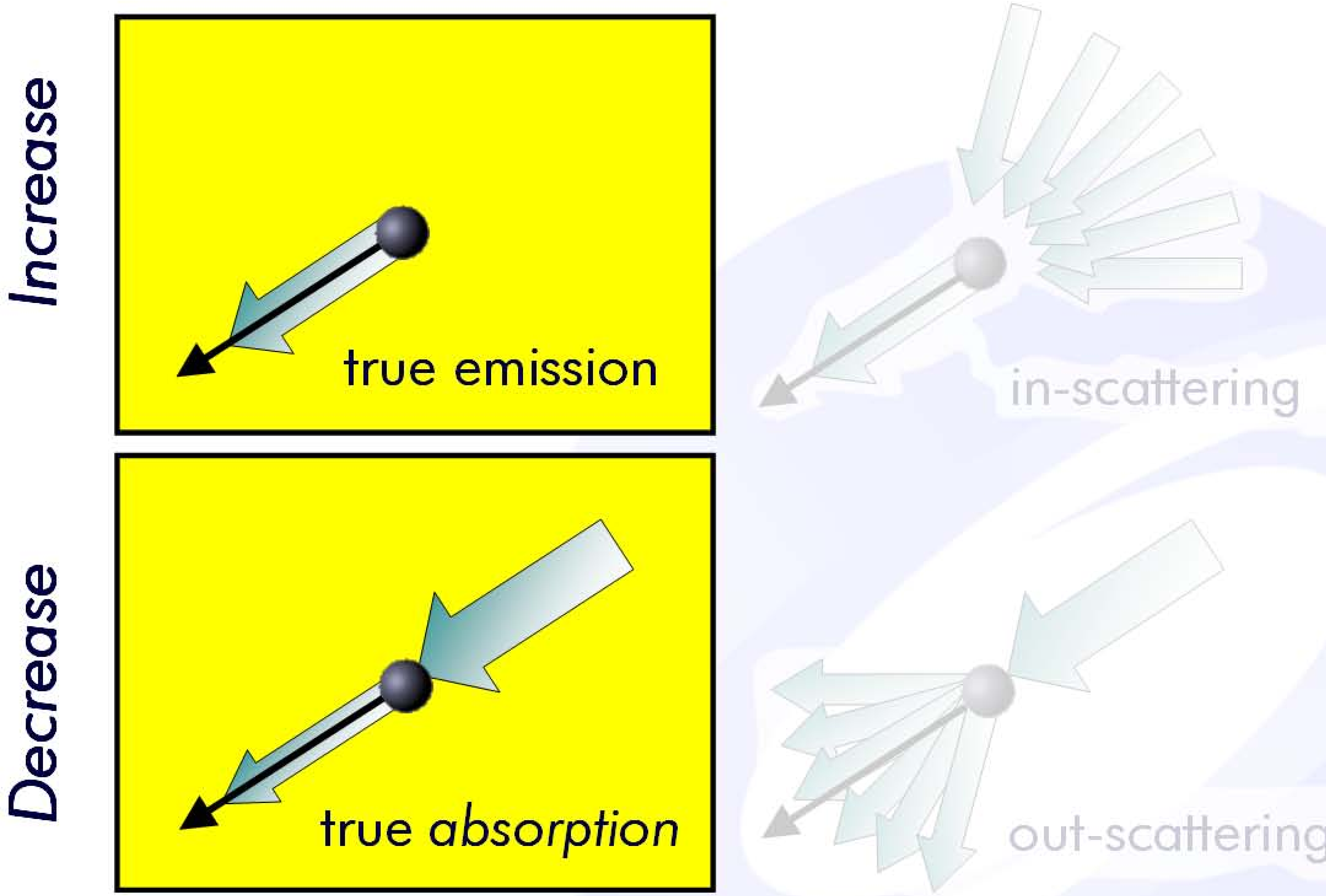
- *Per-Pixel Lighting:* Local illumination in a fragement shader

```
  float3 P = position.xyz;
  float3 N = normal;
  float3 V = normalize(eyePosition - P);
  float3 H = normalize(L + V);

  float3 ambient = Ka * ambientCol;

  float3 L         = normalize(lightPos - P);
  float  diffLight = max(dot(L, N), 0);
  float3 diffuse   = Kd * lightCol * diffLight;

  float specLight = pow(max(dot(H, N), 0), shiny);
  float3 specular = Ks * lightCol * specLight;

  oColor.xyz = ambient + diffuse + specular;
  oColor.w = 1;
}
```

Physical Model of Radiative Transfer



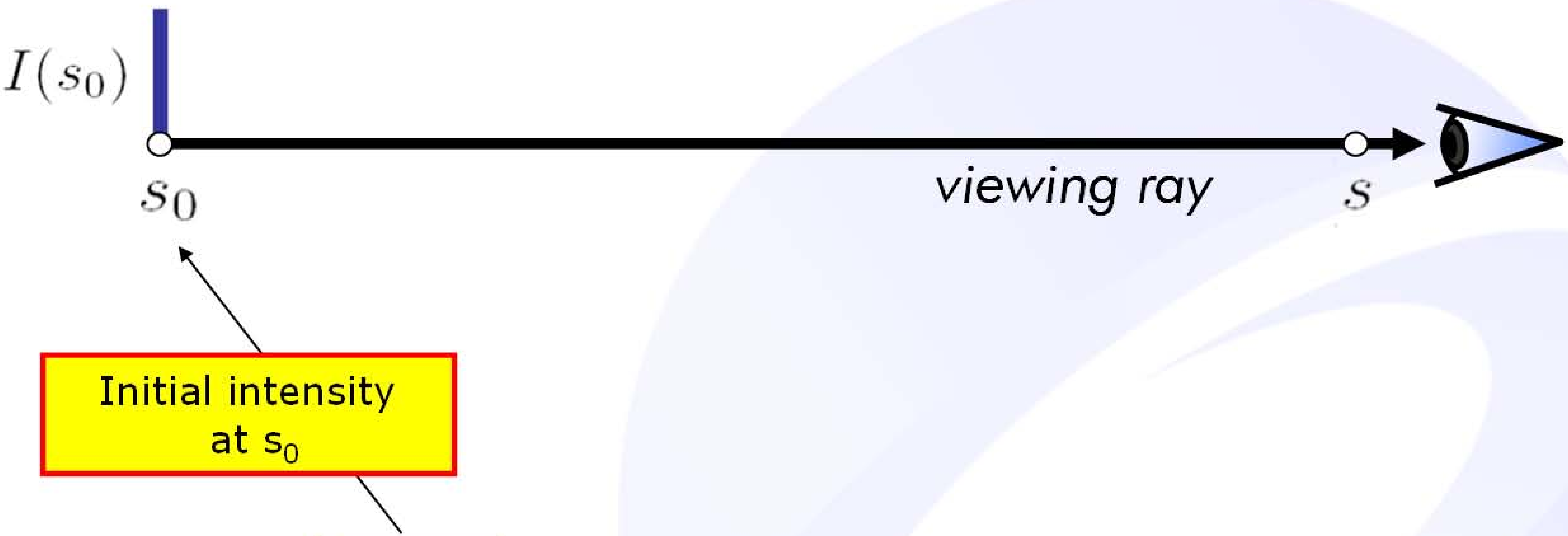
Physical Model of Radiative Transfer



Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



$$I(s) = \boxed{I(s_0)}$$

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering

Initial intensity at s_0

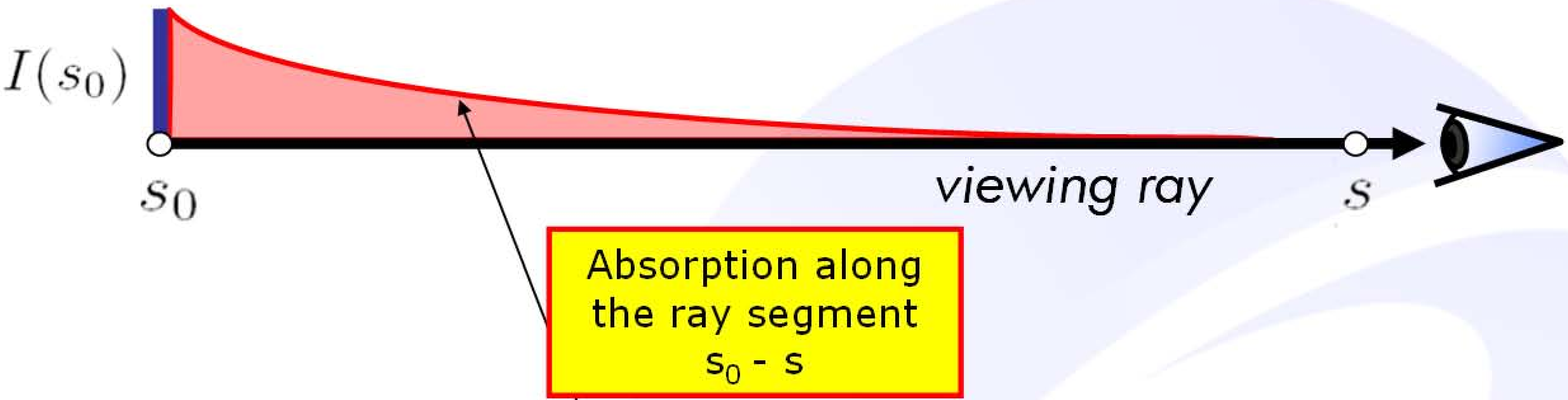
$$I(s) = \boxed{I(s_0)}$$

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

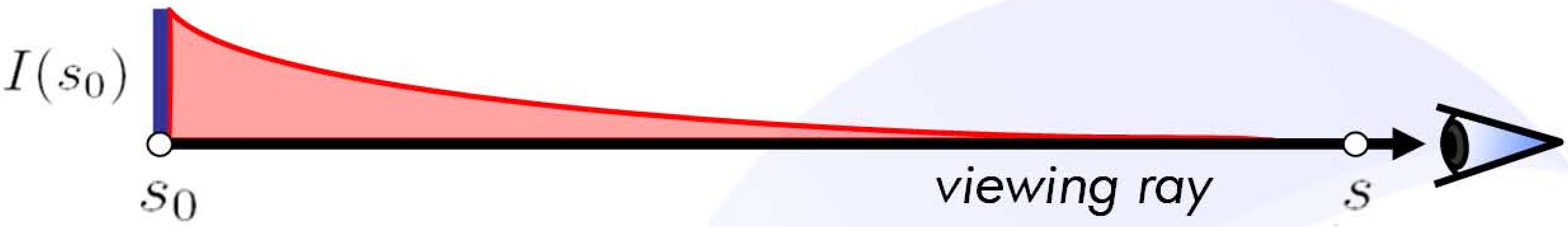


$$I(s) = I(s_0)\, e^{-\tau(s_0, s)}$$

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



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

Extinction τ

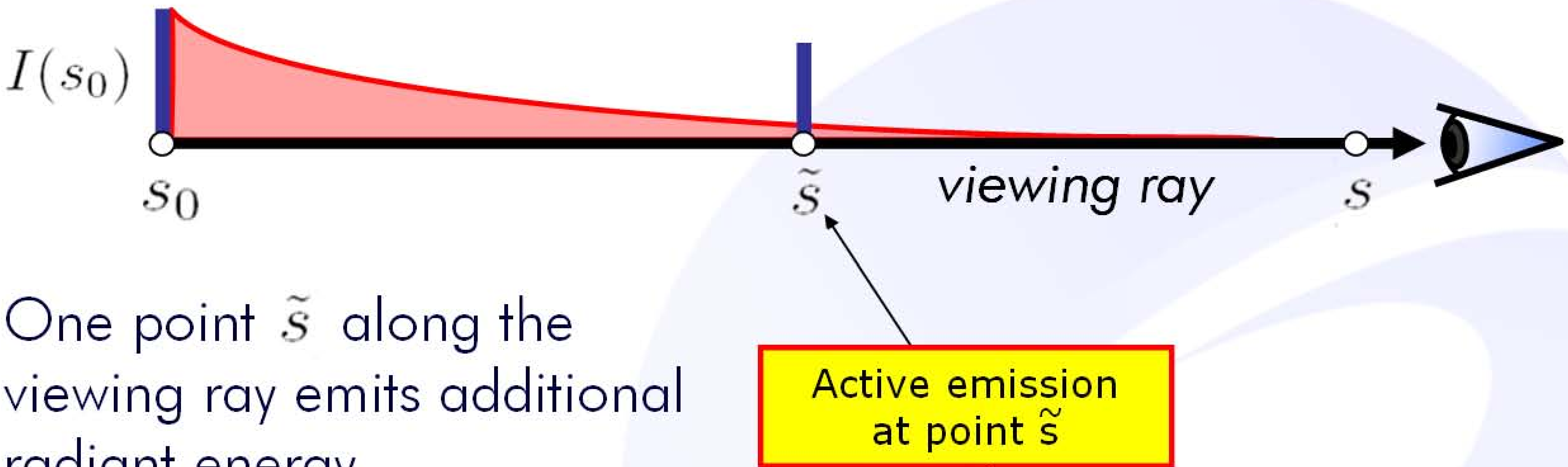
Absorption κ

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

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



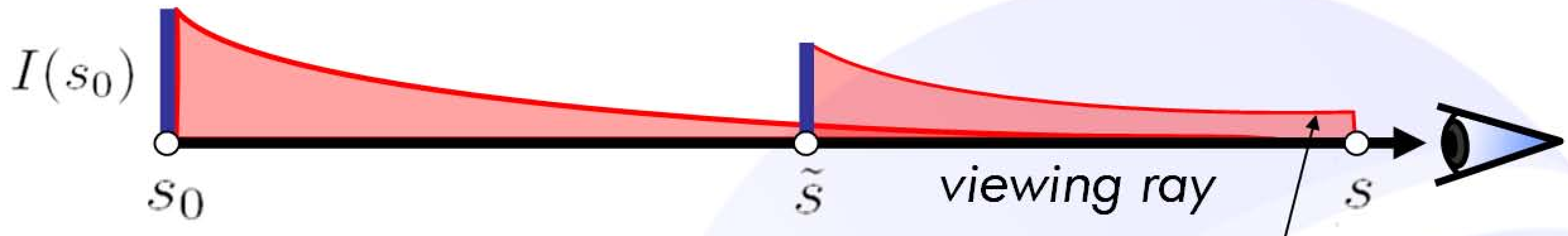
One point $\tilde{s}$ along the viewing ray emits additional radiant energy.

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

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering



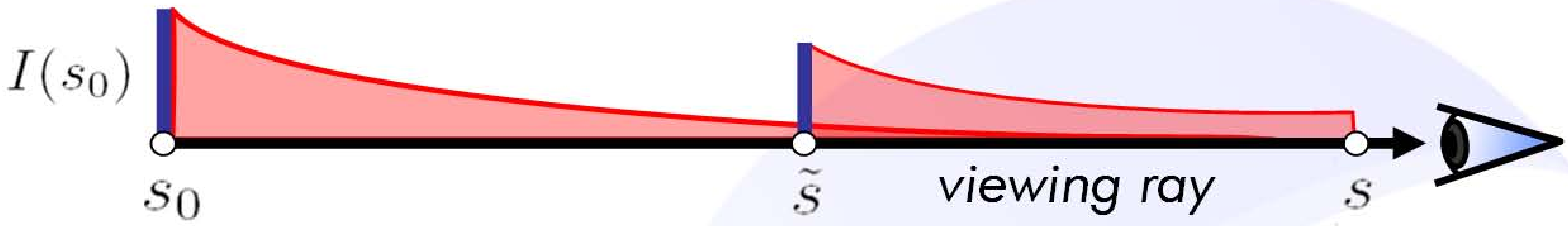
One point $\tilde{s}$ along the viewing ray emits additional radiant energy.

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

Ray Integration

How do we determine the radiant energy along the ray?

Physical model: emission and absorption, no scattering

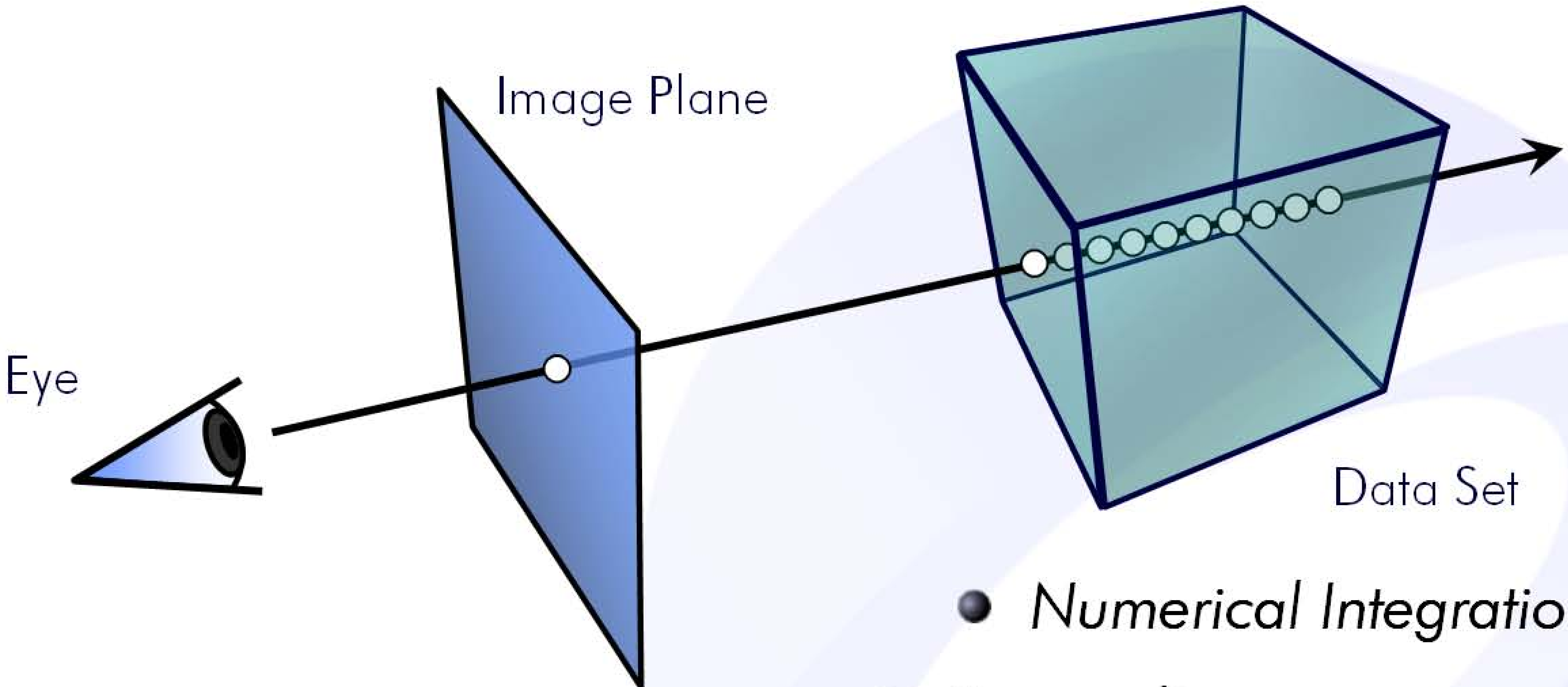


Every point $\tilde{s}$ along the viewing ray emits additional radiant energy

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

Ray Casting

Software Solution:

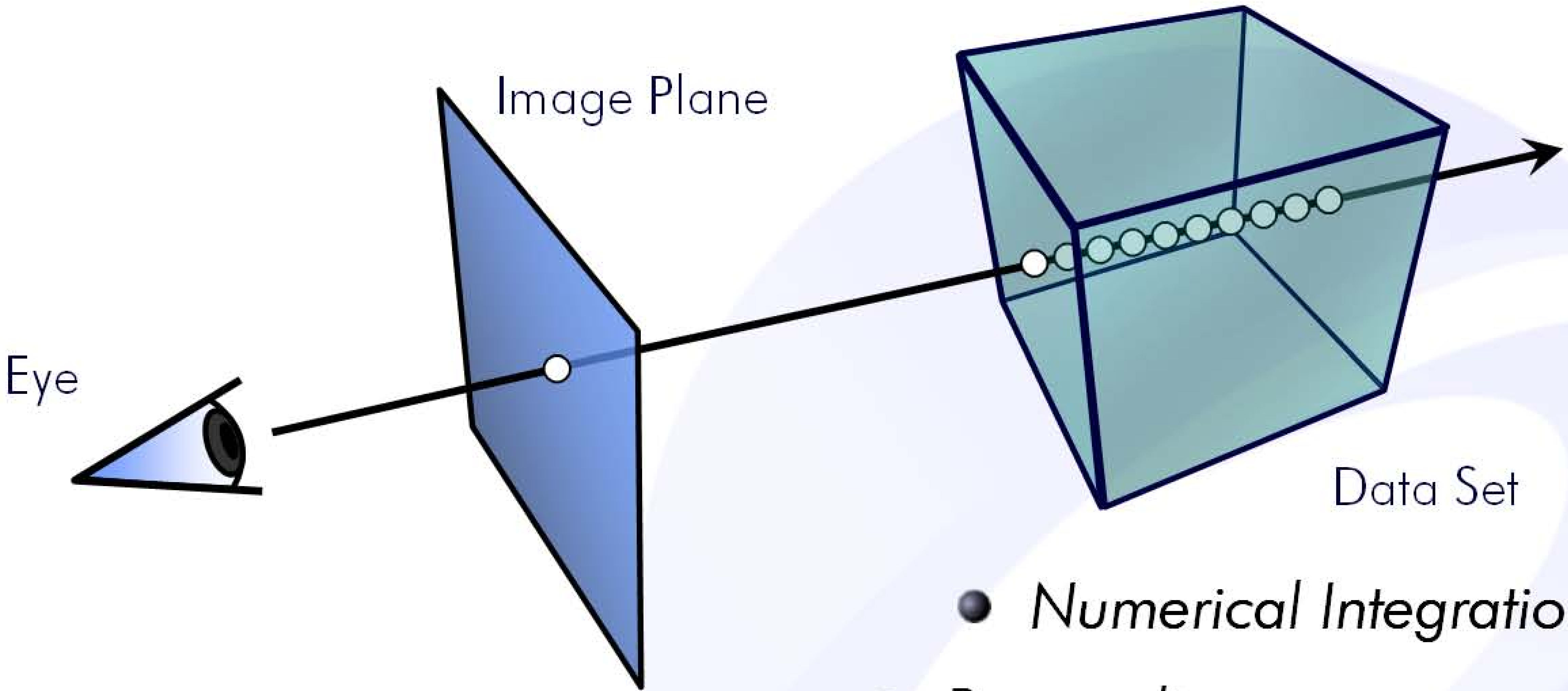


- *Numerical Integration*
- *Resampling*

➡ *High Computational Load*

Ray Casting

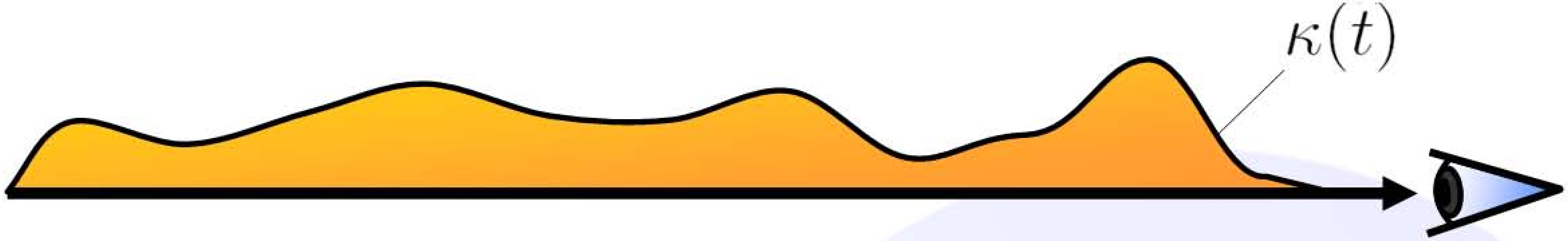
Software Solution:



- *Numerical Integration*
- *Resampling*

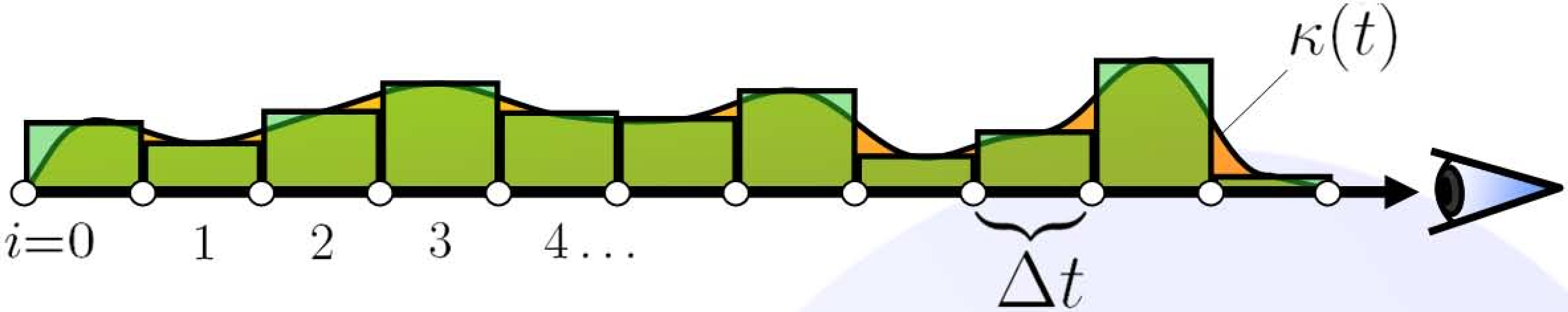
➡ *High Computational Load*

Numerical Solution



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

Numerical Solution

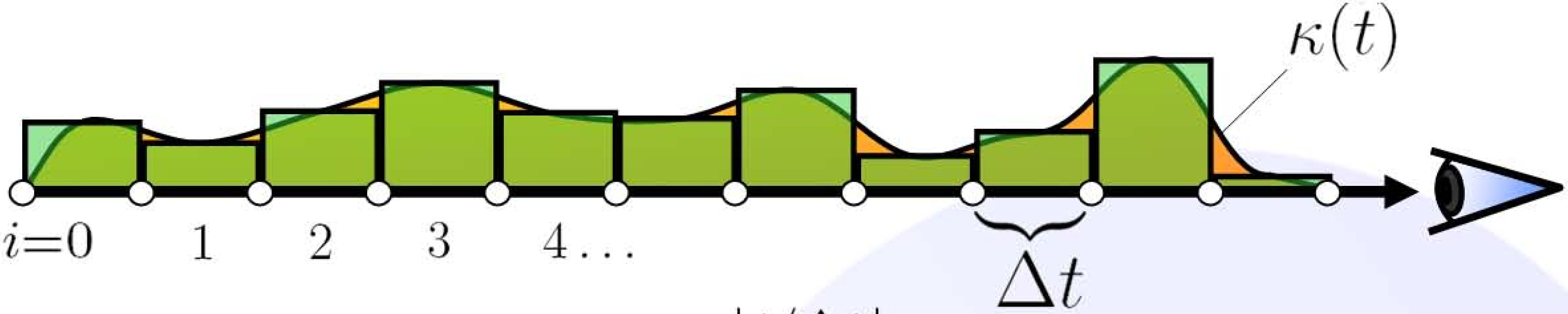


Extinction: $$\tau(0,t) = \int_0^t \kappa(\hat{t})\, d\hat{t}$$

Approximate Integral by Riemann sum:

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

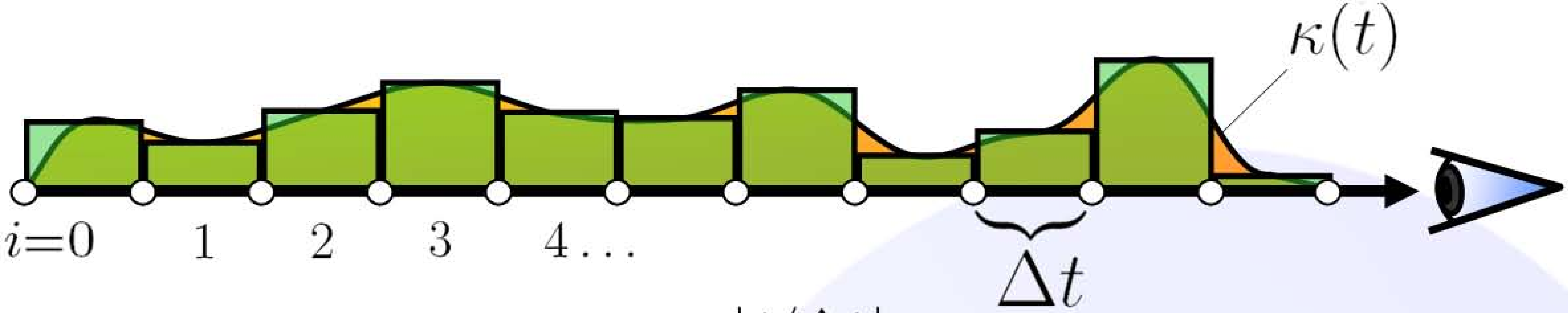
Numerical Solution



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

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

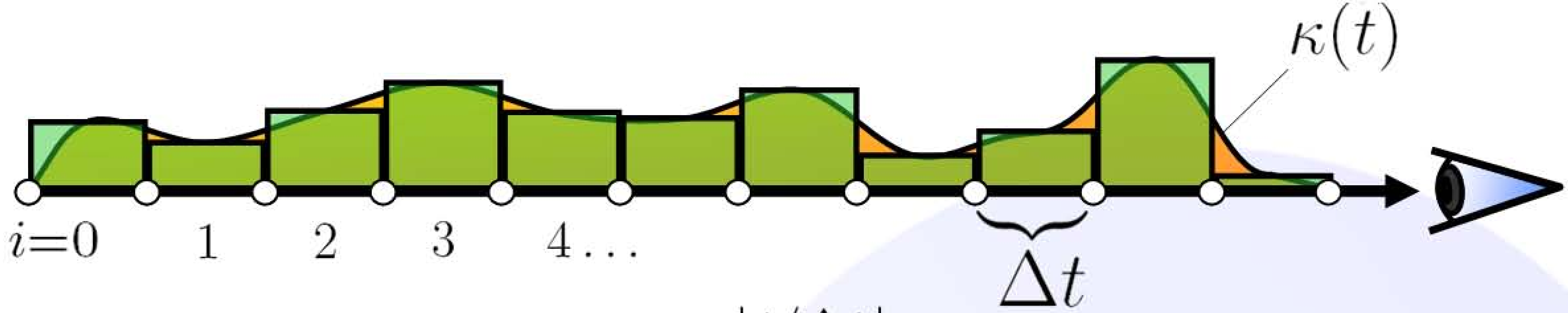
Numerical Solution



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

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

Numerical Solution



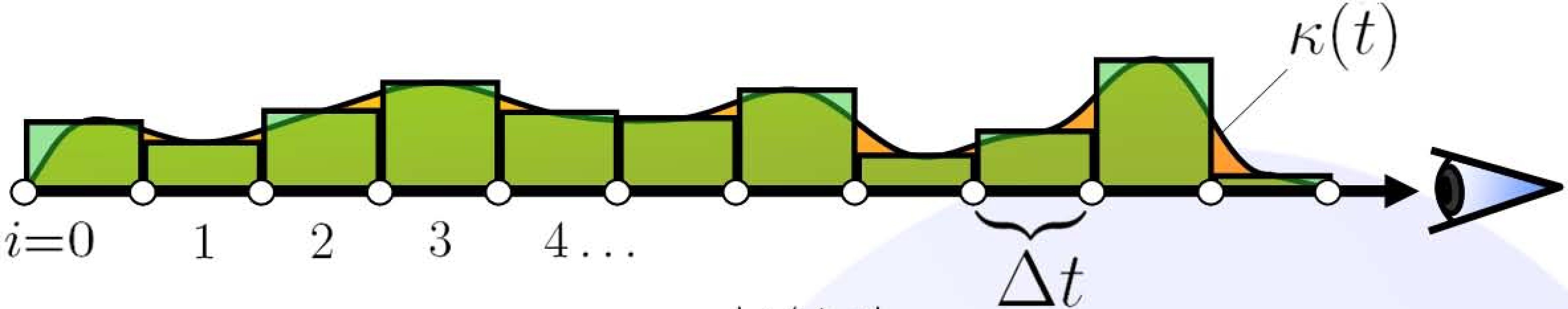
$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

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

Now we introduce opacity:

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

Numerical Solution



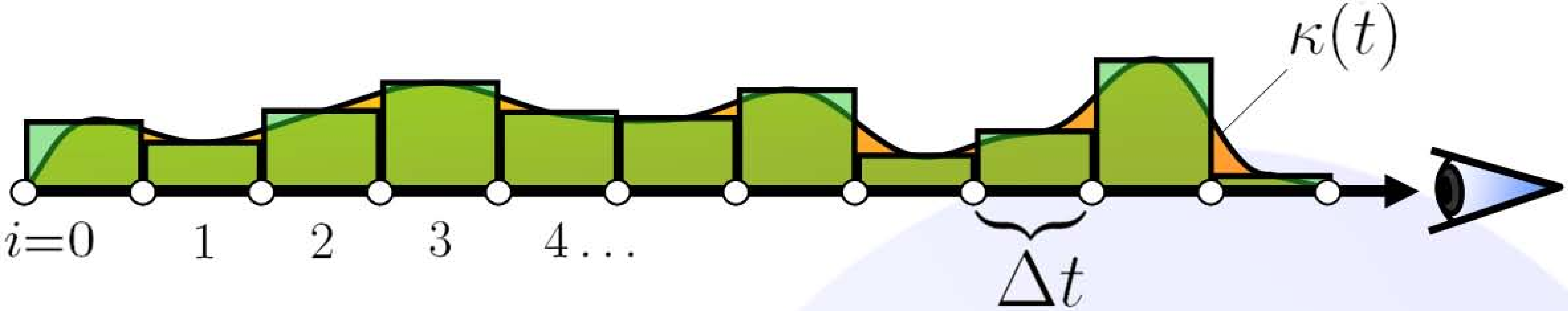
$$\tau(0,t) \approx \tilde{\tau}(0,t) = \sum_{i=0}^{\lfloor t/\Delta t \rfloor} \kappa(i \cdot \Delta t)\, \Delta t$$

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

Now we introduce opacity:

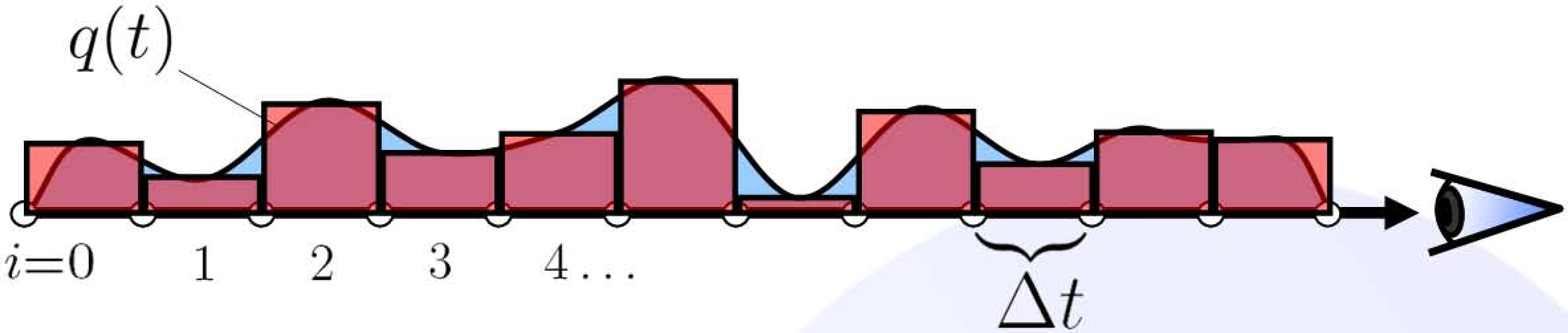
$$1 - A_i = e^{-\kappa(i \cdot \Delta t)\, \Delta t}$$

Numerical Solution



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

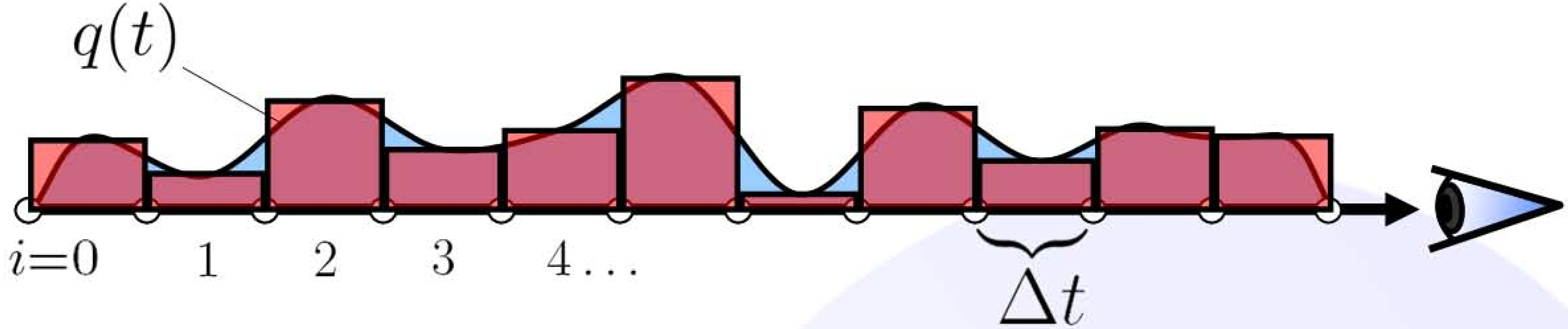
Numerical Solution



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

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

Numerical Solution

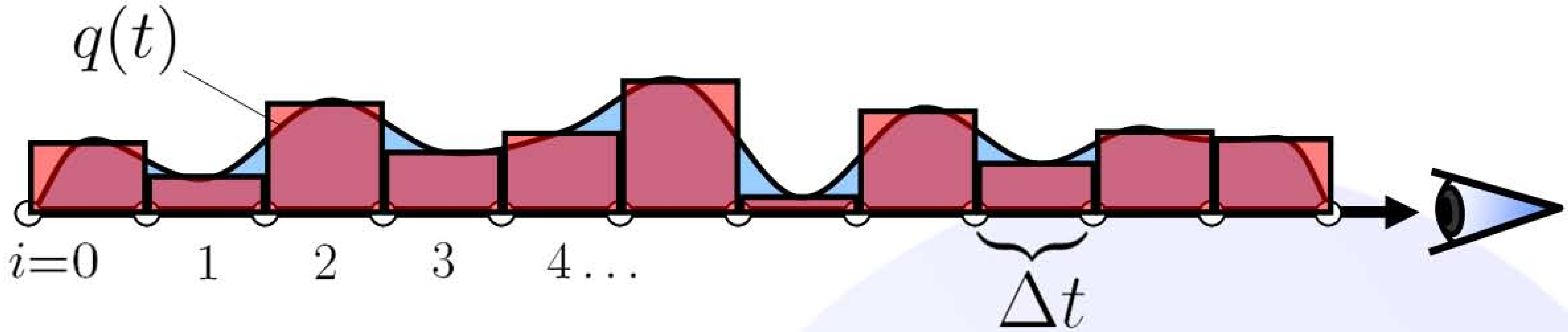


$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

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

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

Numerical Solution

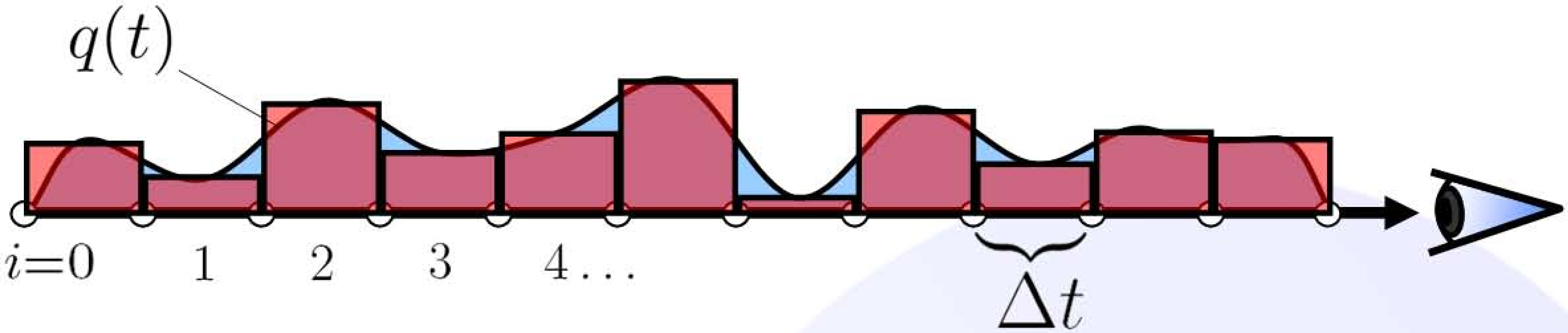


$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

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

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

Numerical Solution

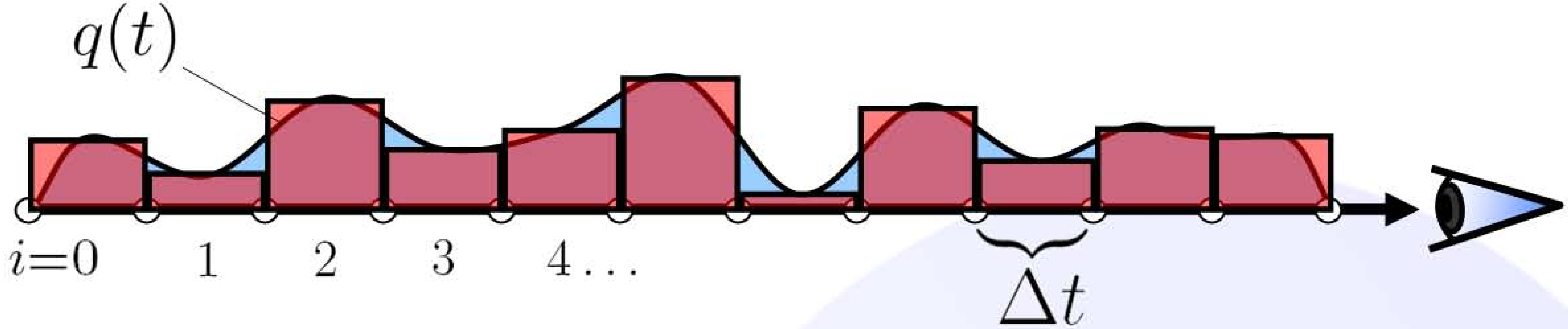


$$e^{-\tilde{\tau}(0,t)} = \prod_{i=0}^{\lfloor t/\Delta t \rfloor} (1 - A_i)$$

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

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

Numerical Solution



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

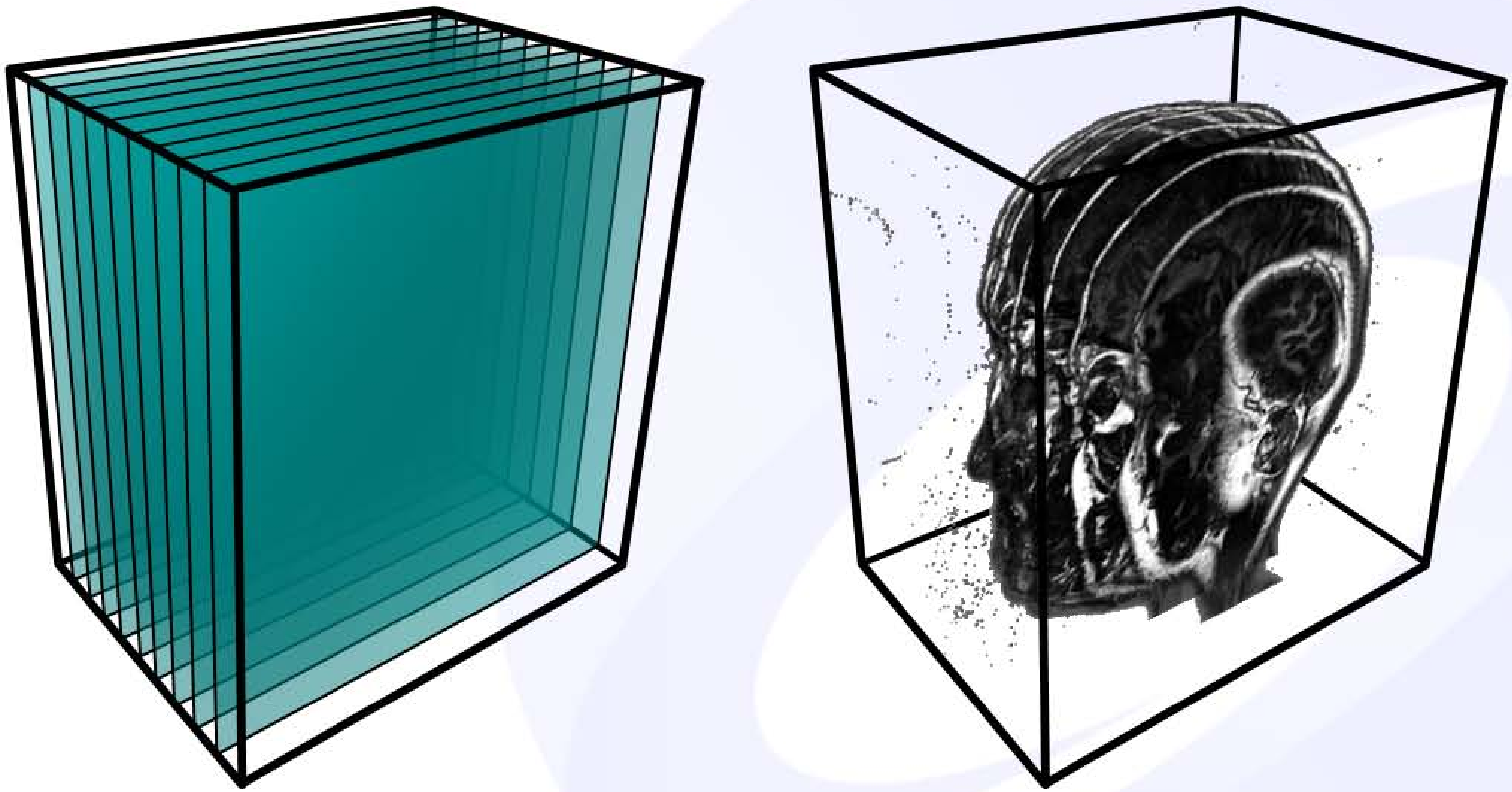
can be computed recursively

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

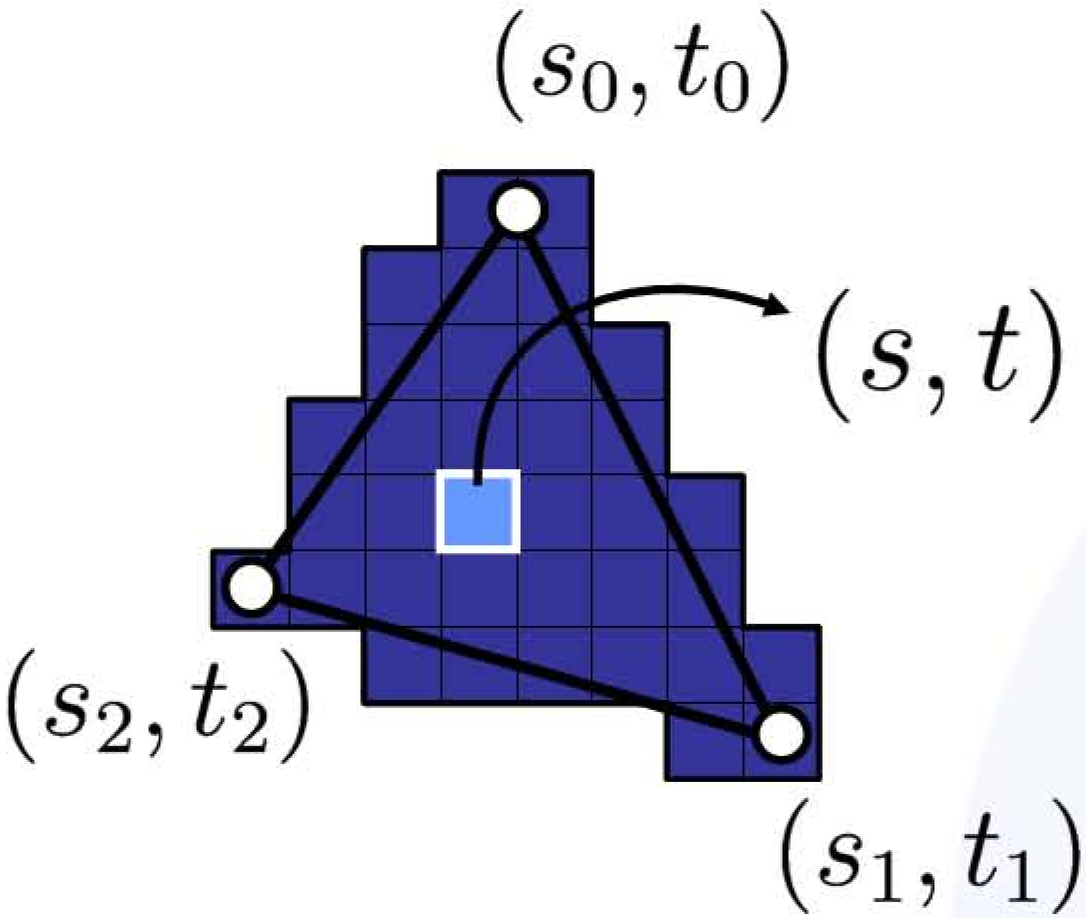
- C'_i: Radiant energy observed at position *i*
- C_i: Radiant energy emitted at position *i*
- A_i: Absorption at position *i*
- C'_{i-1}: Radiant energy observed at position *i*–1

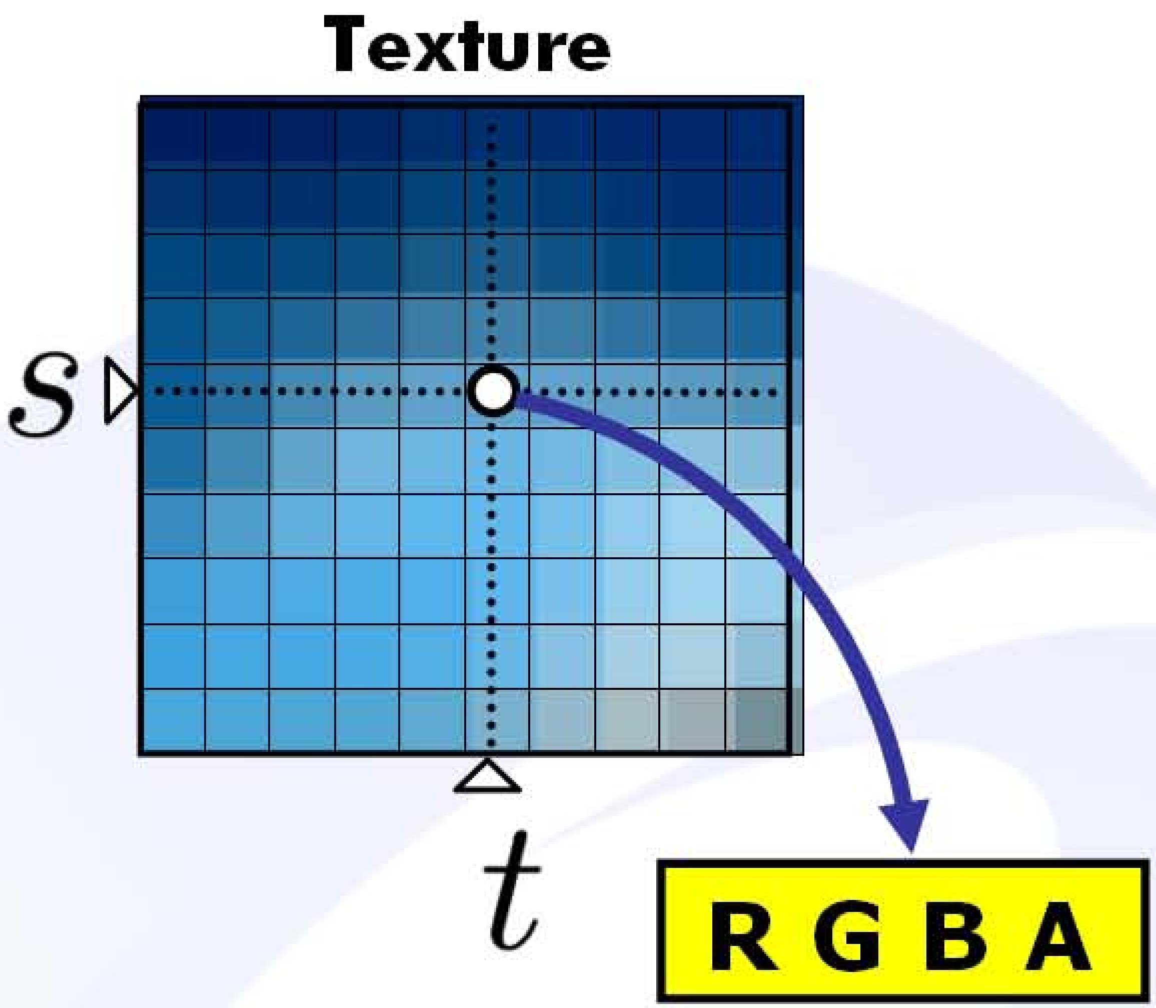
Texture-based Approaches

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



How does a texture work?





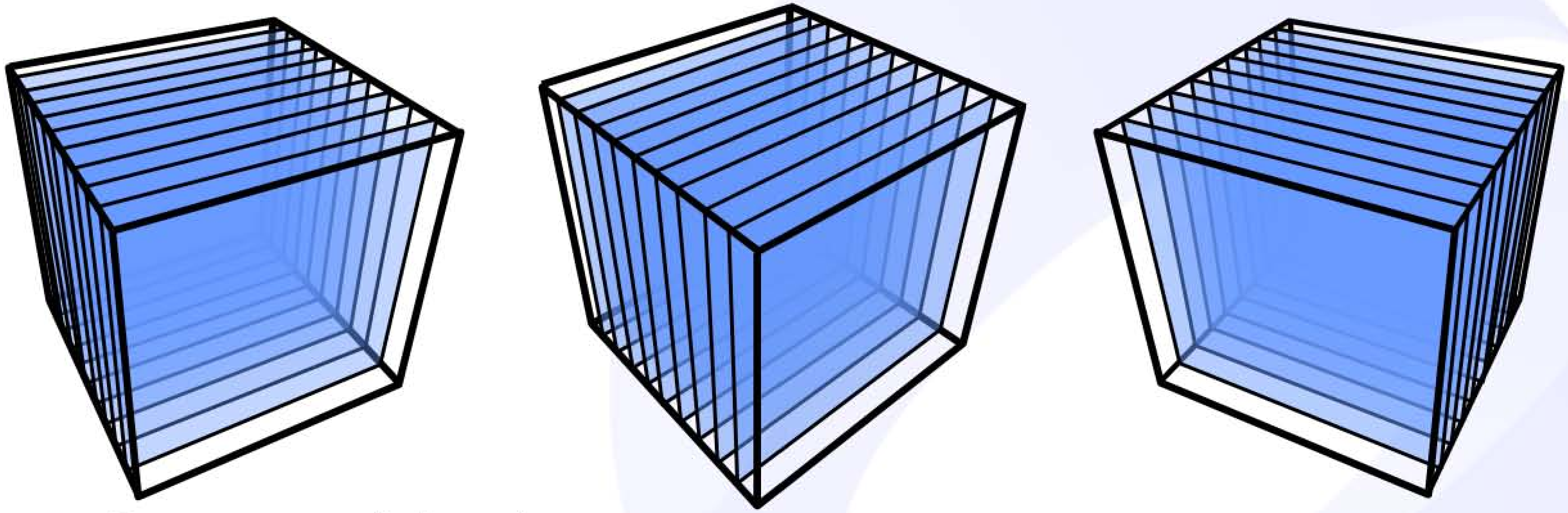
For each fragment:
interpolate the
texture coordinates
(barycentric)

Texture-Lookup:
interpolate the
texture color
(bilinear)

2D Textures

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

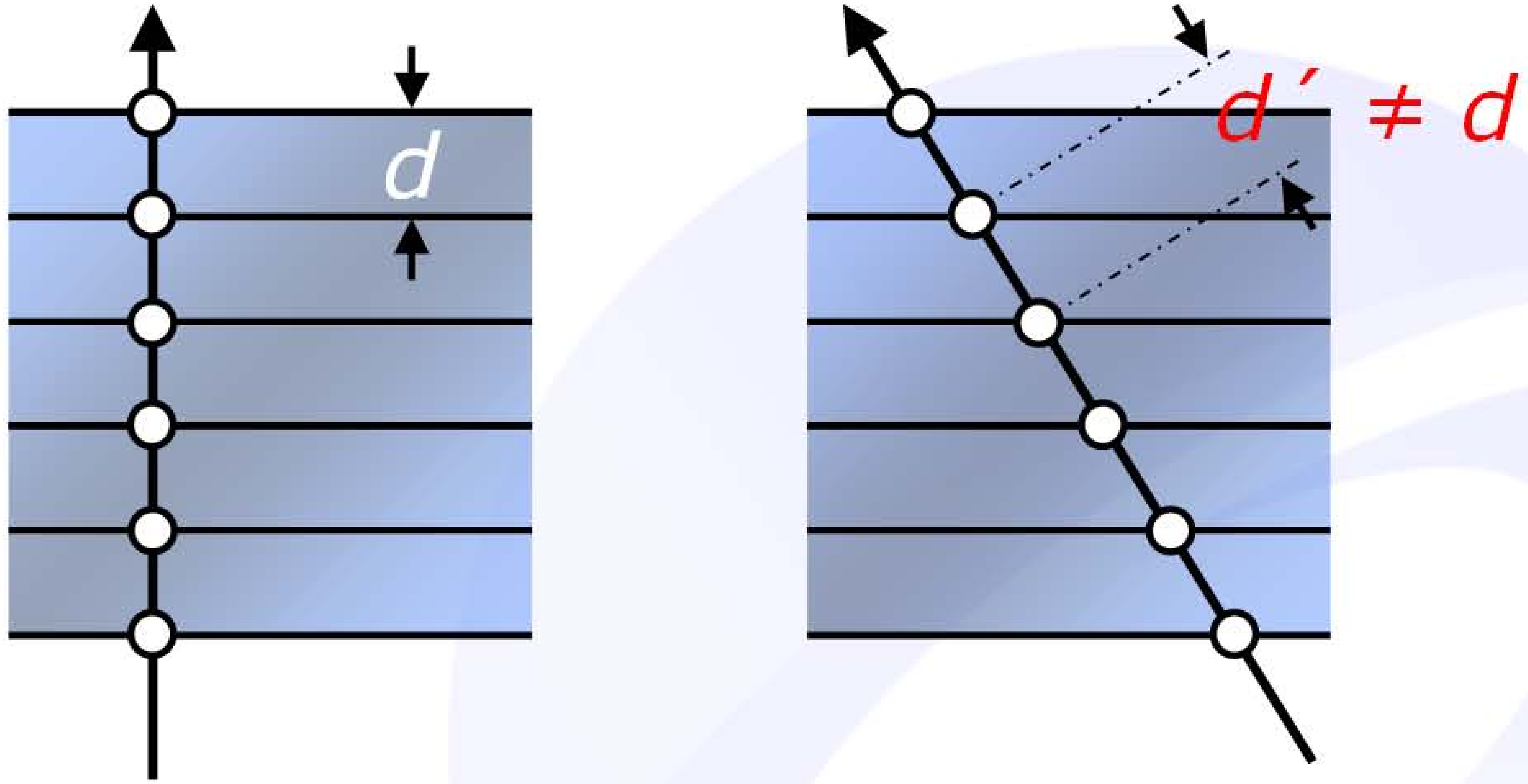
➡ Decompostition into axis-aligned slices



- 3 copies of the data set in memory

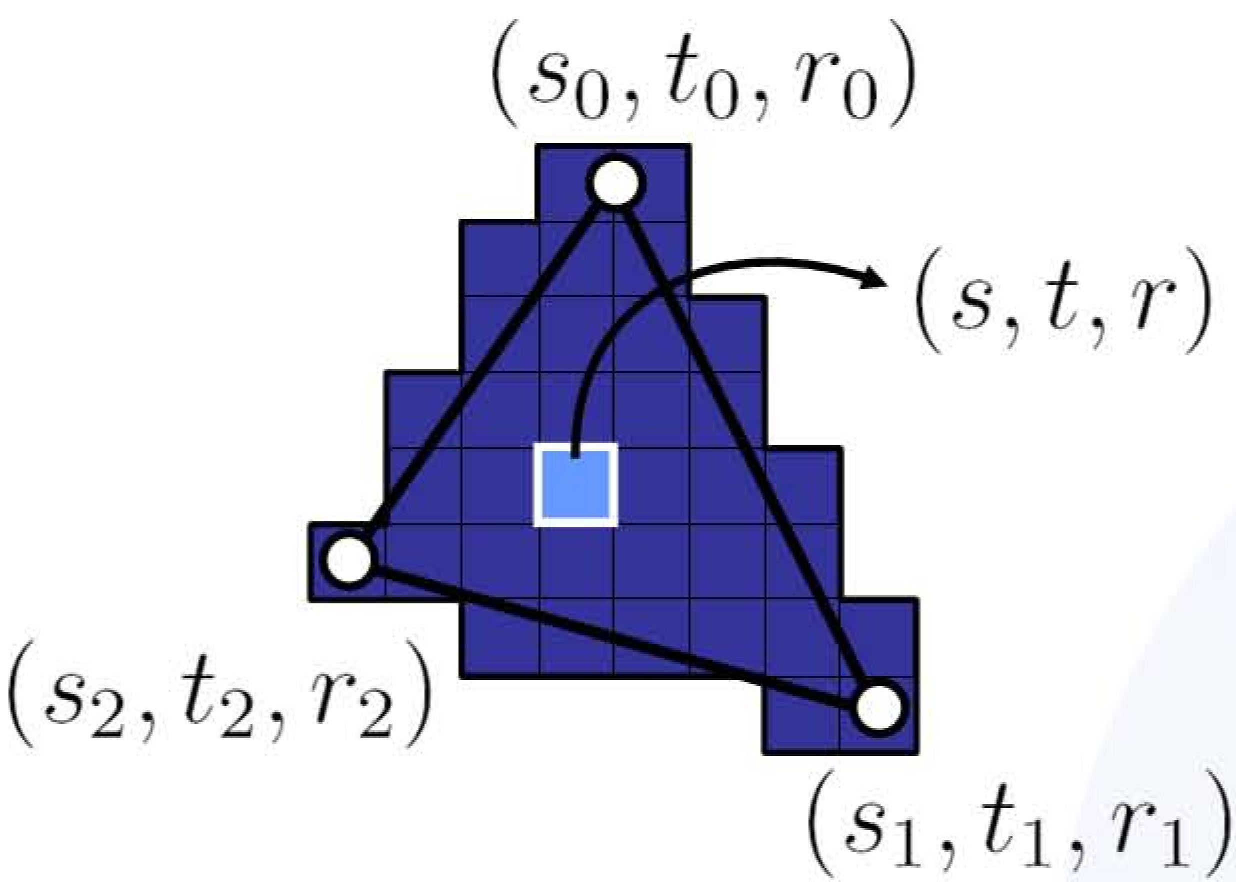
2D Textures

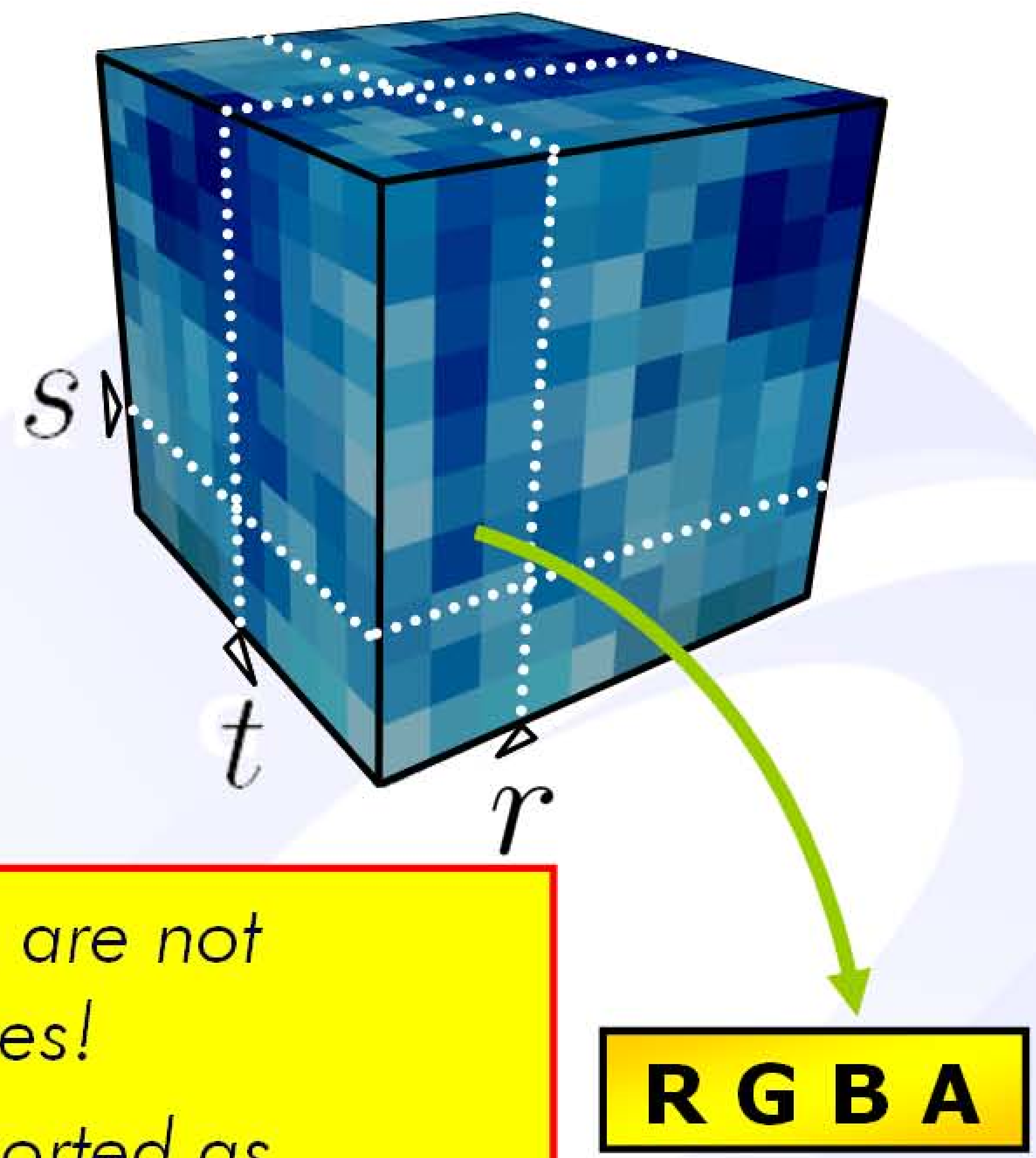
- Sampling rate is inconsistent



- emission/absorption slightly incorrect
- Super-sampling on-the-fly impossible

3D Textures





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

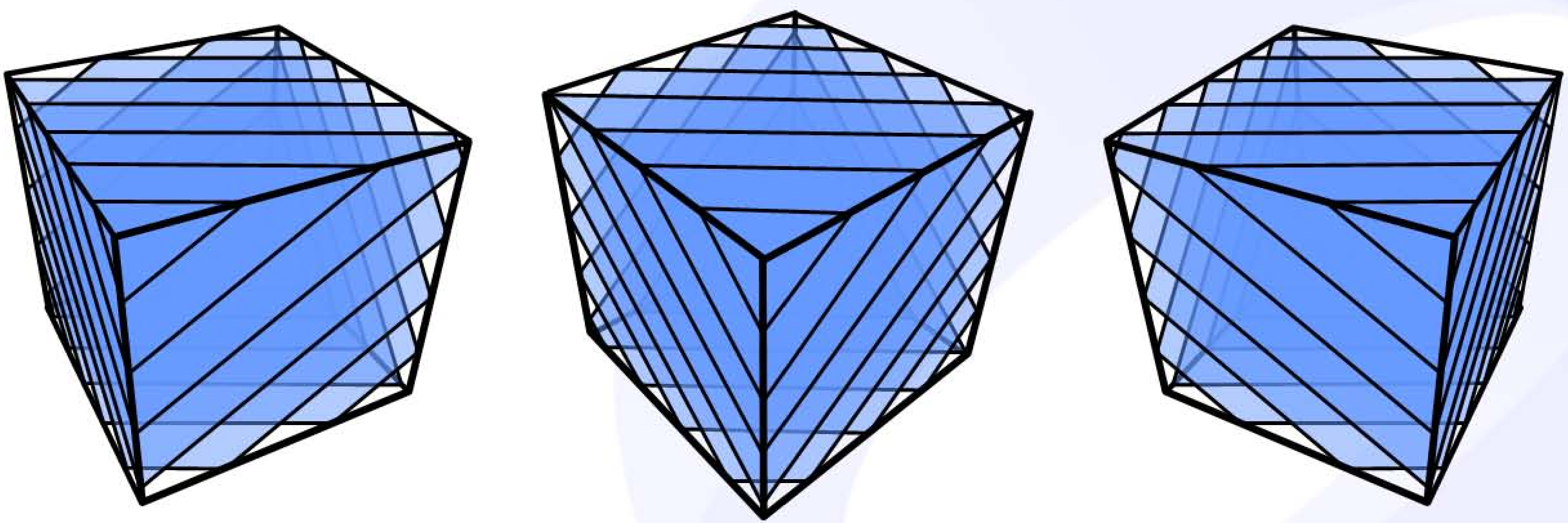
*Only **planar polygons** are supported as rendering primitives.*

3D Textures

3D Texture: Volumetric Texture Object

- Trilinear Interpolation in Hardware

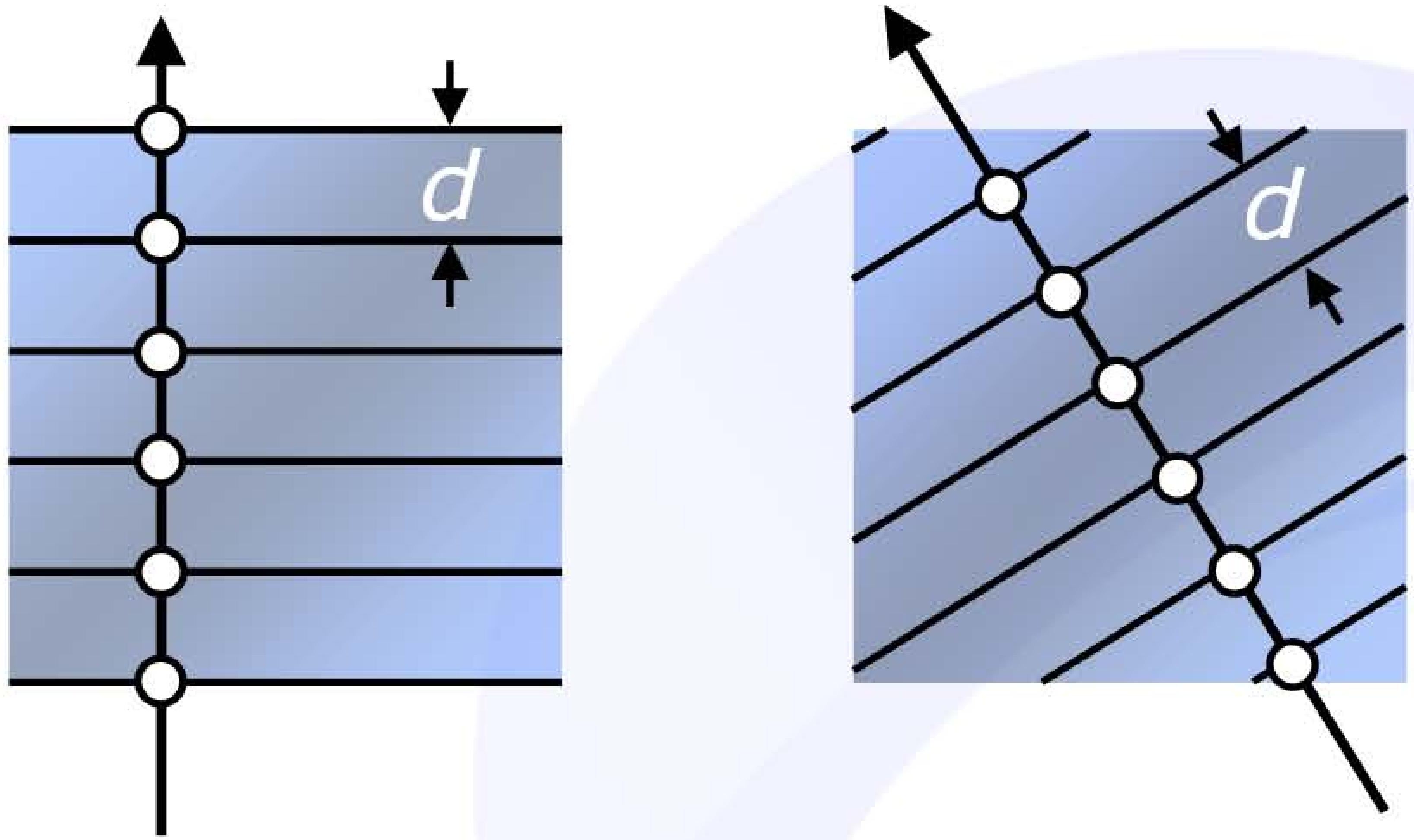
➡ Slices parallel to the image plane



- One large texture block in memory

Resampling via 3D Textures

- *Sampling rate is constant*

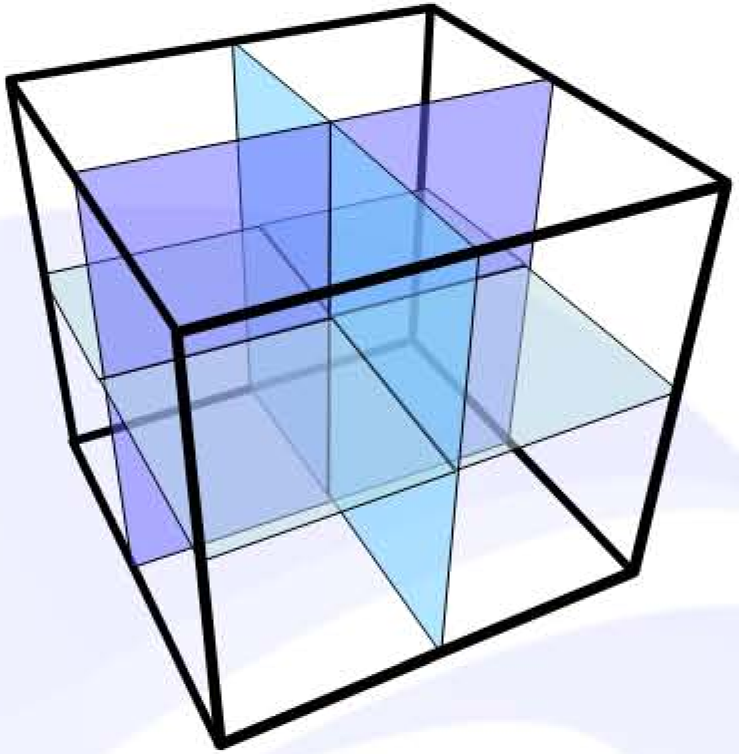


- Supersampling by increasing the number of slices

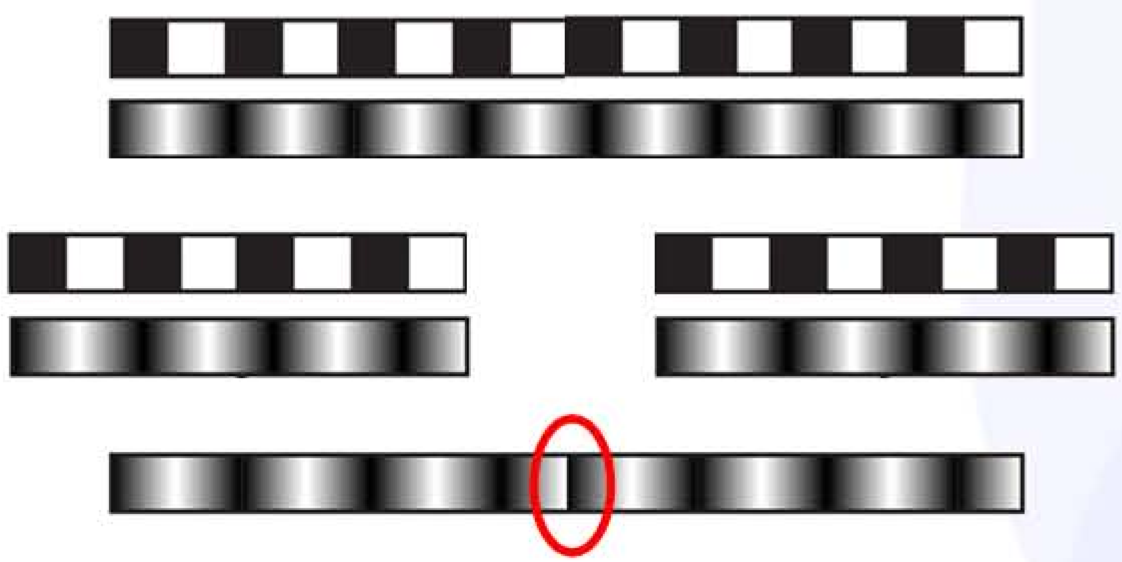
Bricking

- What happens if data set is too large to fit into local video memory?

➔ Divide the data set into smaller chunks (bricks)



One plane of voxels must be duplicated to enable correct interpolation across brick boundaries

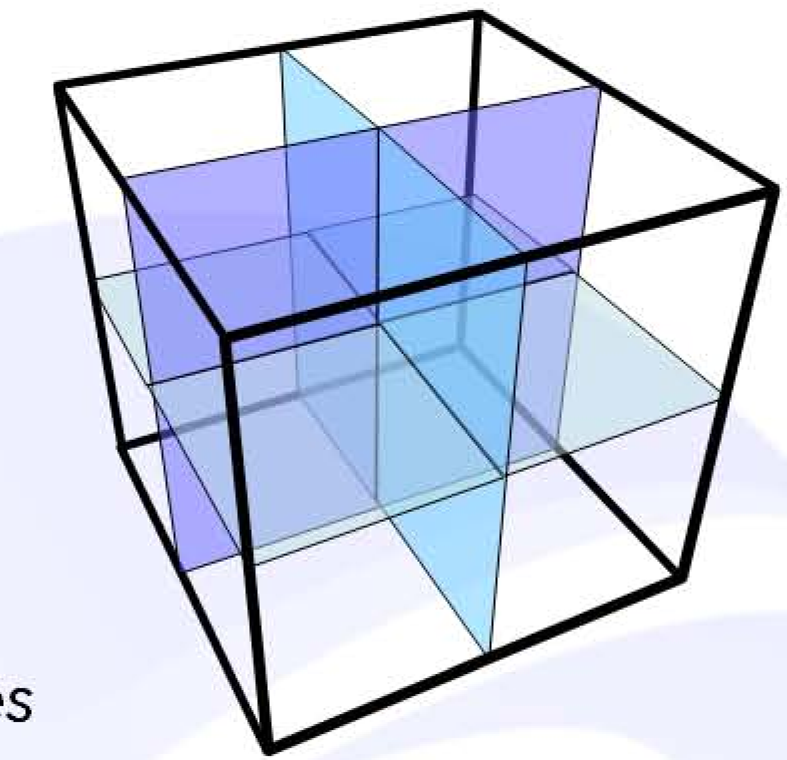


incorrect interpolation!

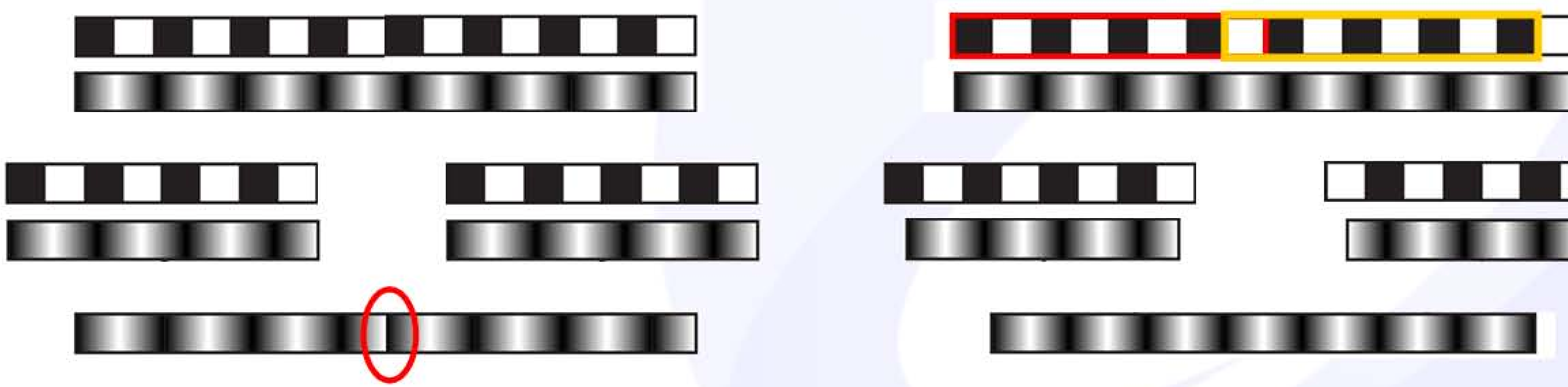
Christof Rezk Salama
Computer Graphics and Multimedia Group, University of Siegen, Germany

Bricking

- What happens if data set is too large to fit into local video memory?
- ➔ Divide the data set into smaller chunks (bricks)



One plane of voxels must be duplicated to enable correct interpolation across brick boundaries



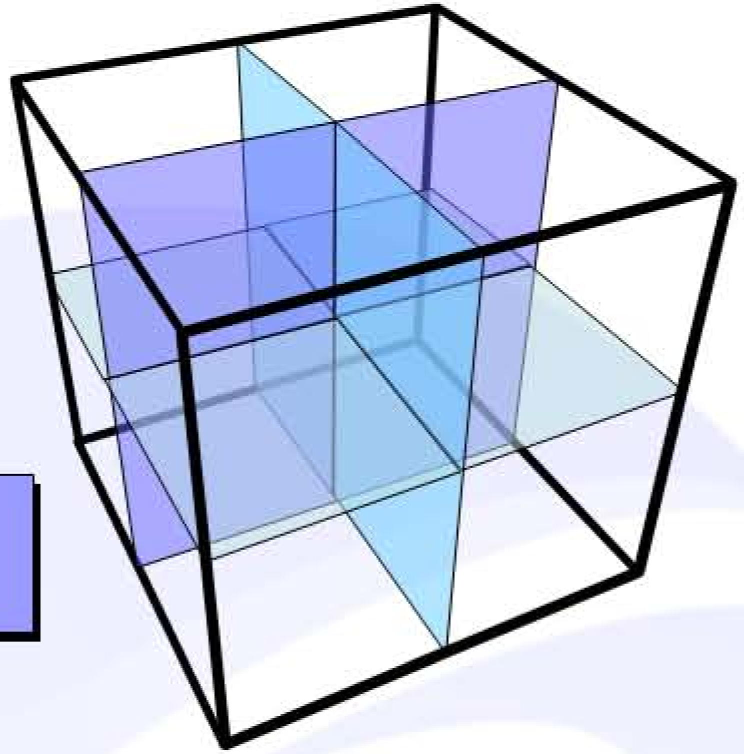
incorrect interpolation!

Bricking

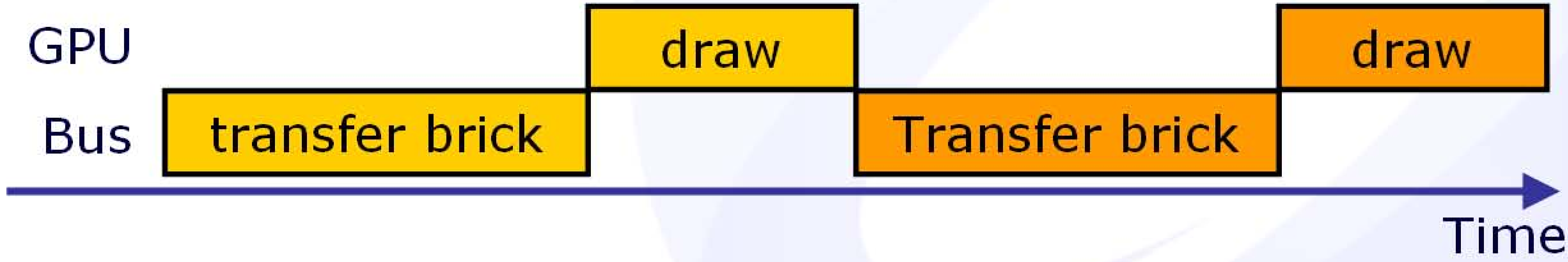
- What happens if data set is too large to fit into local video memory?

➡ Divide the data set into smaller chunks (bricks)

Problem: Bus-Bandwidth



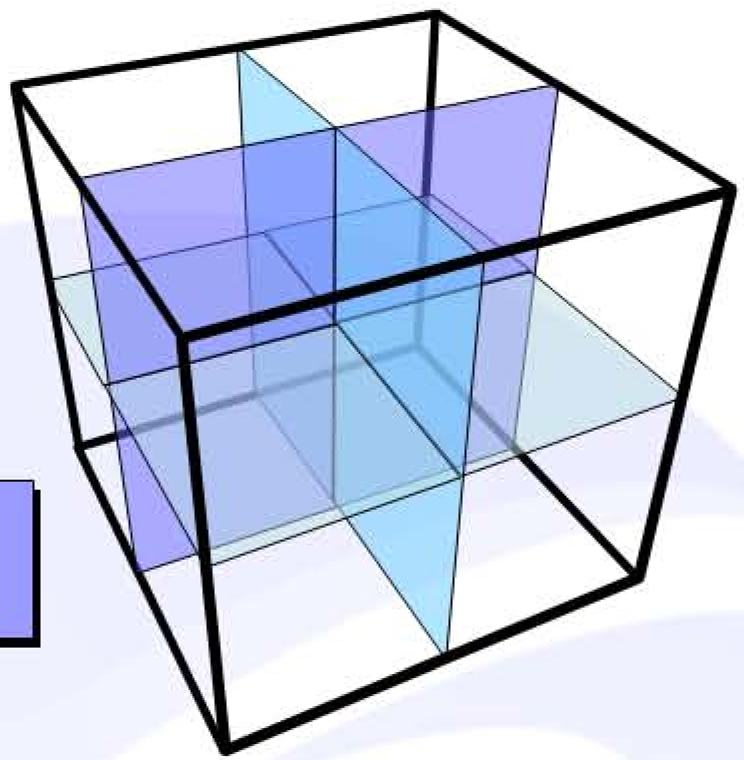
- Unbalanced Load for GPU und Memory Bus



Bricking

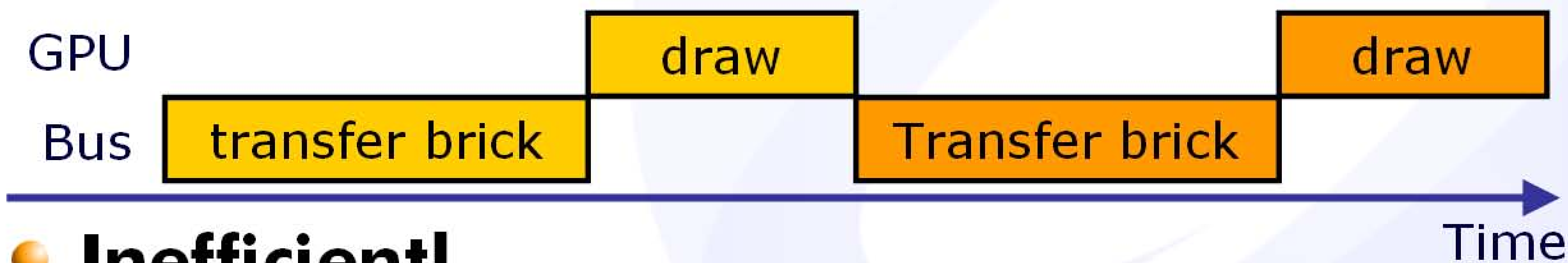
- What happens if data set is too large to fit into local video memory?

➡ Divide the data set into smaller chunks (bricks)



Problem: **Bus-Bandwidth**

- Unbalanced Load for GPU und Memory Bus



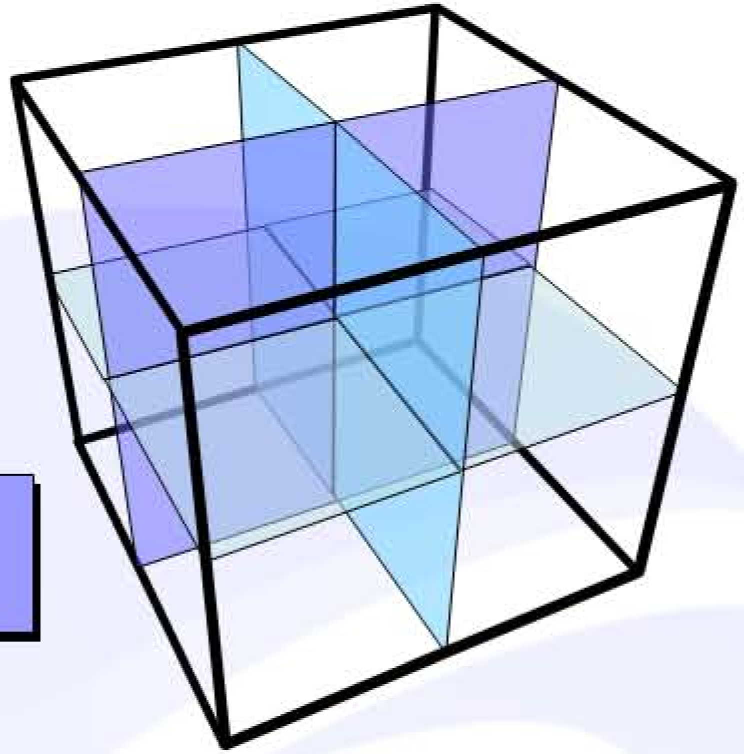
- **Inefficient!**

REAL-TIME VOLUME GRAPHICS
Christof Rezk Salama
Computer Graphics and Multimedia Group, University of Siegen, Germany

Bricking

- What happens if data set is too large to fit into local video memory?

➡ Divide the data set into smaller chunks (bricks)

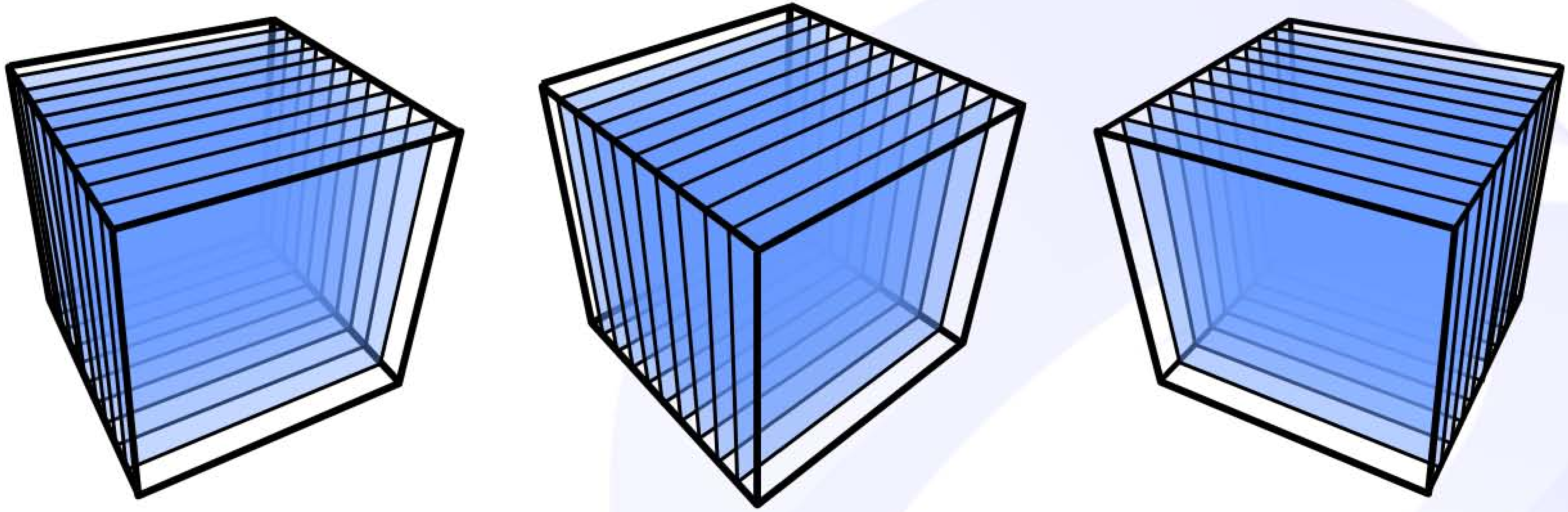


Problem: **Bus-Bandwidth**

- Keep the bricks small enough!
  More than one brick must fit into video memory !
  - Transfer and Rendering can be performed in parallel
  - Increased CPU load for intersection calculation!
  - *Effective load balancing still very difficult!*

Back to 2D Textures

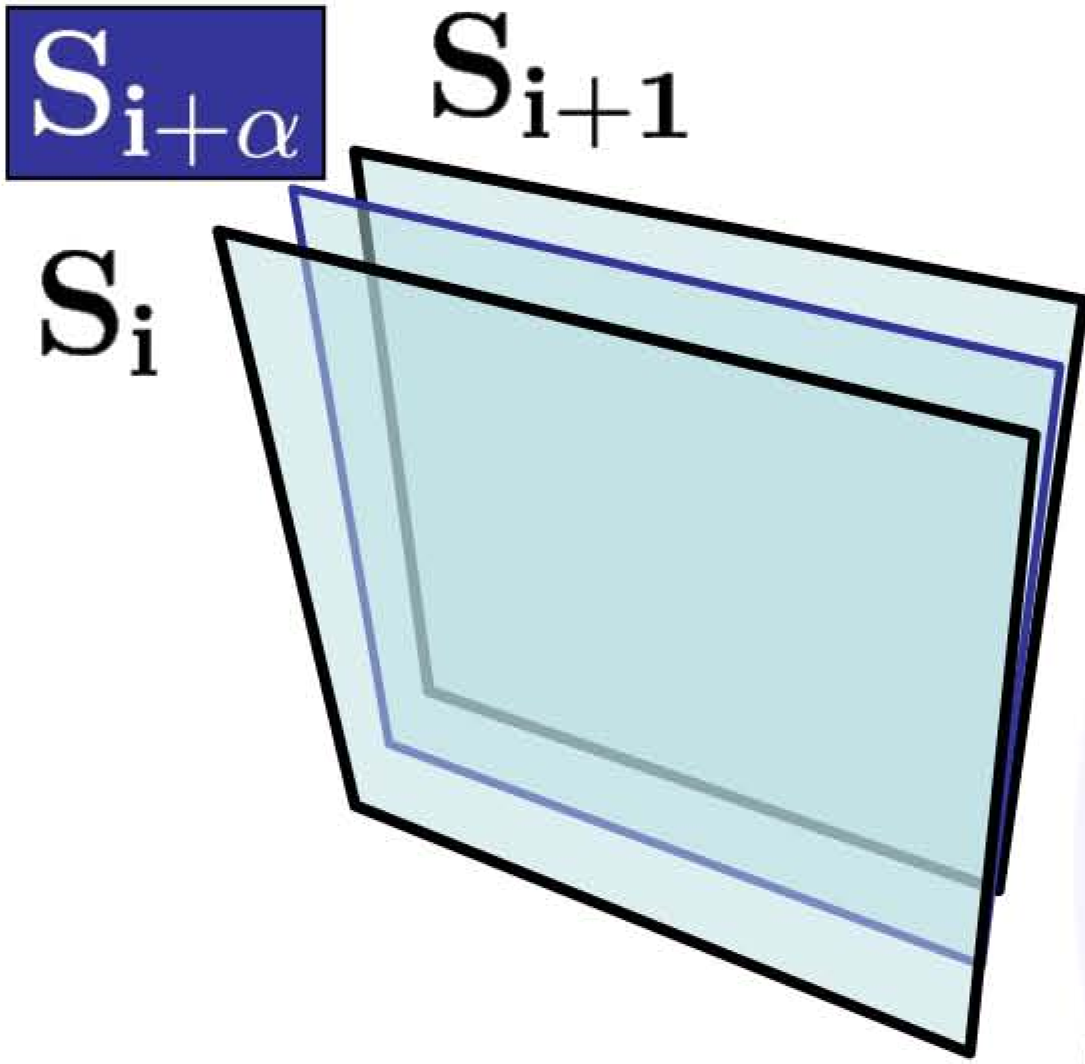
- ~~fixed~~ number of object aligned slices
- visual artifacts due to bilinear interpolation



- Utilize Multi-Textures (2 textures per polygon) to implement trilinear interpolation!

2D Multi-Textures

Axis-Aligned Slices



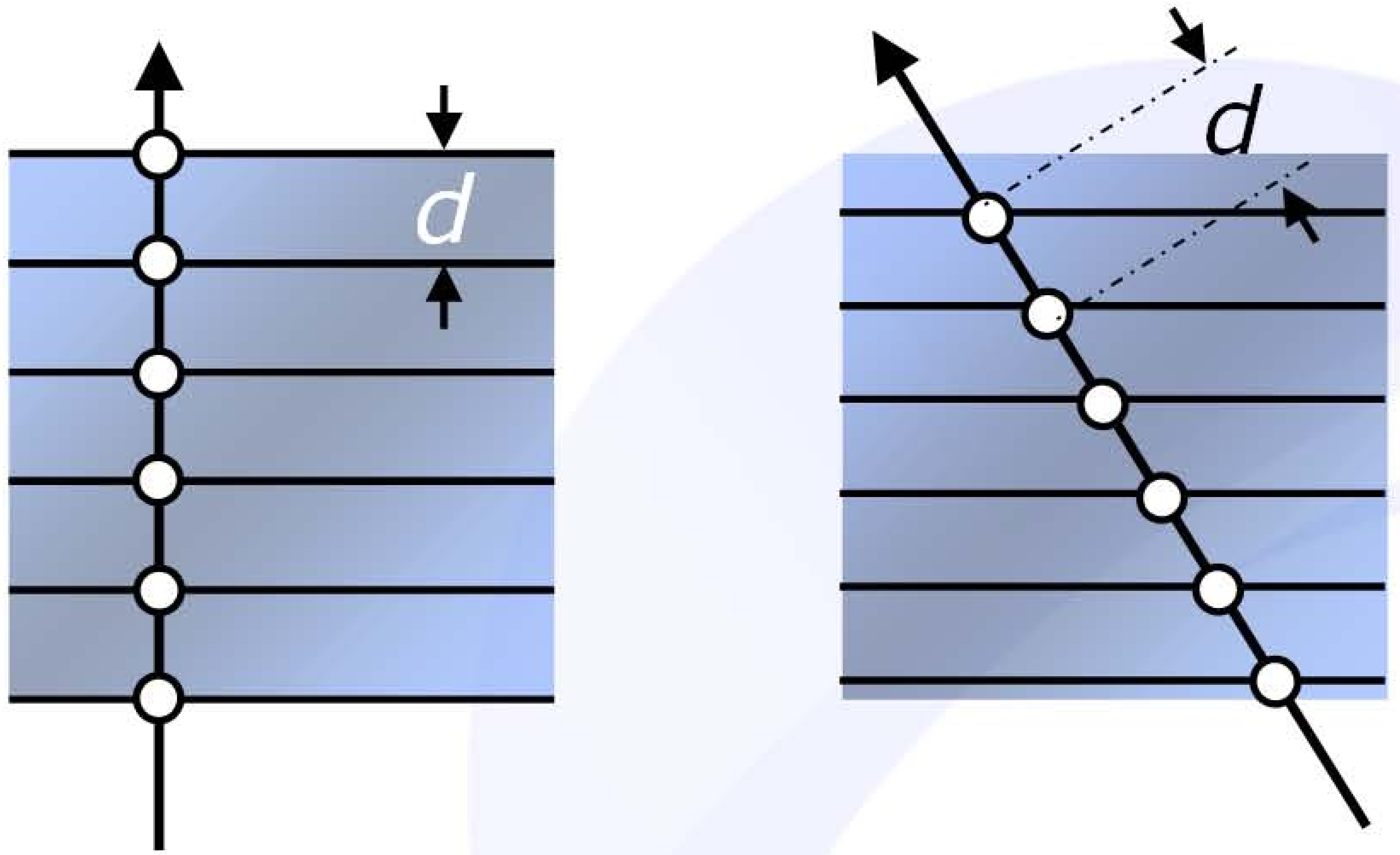
- Bilinear Interpolation by 2D Texture Unit
- Blending of two adjacent slice images

$$\mathbf{S_{i+\alpha}} = (\mathbf{1} - \alpha)\mathbf{S_i} + \alpha \cdot \mathbf{S_{i+1}}$$

- Trilinear Interpolation

2D Multi-Textures

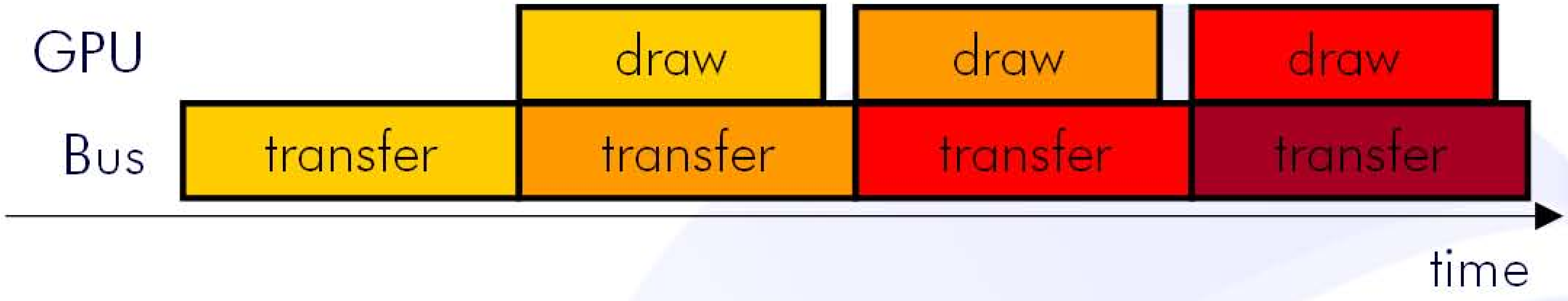
- Sampling rate is constant



- Supersampling by increasing the number of slices

Advantages

- More efficient load balancing



  - Exploit the GPU and the available memory bandwidth in parallel
  - Transfer the smallest amount of information required to draw the slice image!
- **Significanly higher performance**, although 3 copies of the data set in main memory

Summary

Rasterization Approaches for Direct Volume Rendering

- *2D Texture Based Approaches*
  - 3 fixed stacks of object aligned slices
  - Visual artifacts due to bilinear interpolation only
  - No supersampling
- *3D Texture Based Approaches*
  - Viewport aligned slices
  - Supersampling with trilinear interpolation
  - Bricking: Bus transfer inefficient for large volumes
- *2D Multi-Texture Based Approaches*
  - 3 variable stacks of object aligned slices
  - Supersampling with Trilinear interpolation
  - Higher performance for larger volumes

Thanks

Special thanks to *Mark Segal* from *ATI* for providing the *Radeon X800 XT* demo machine