Ray Casting on Programmable Graphics Hardware

Martin Kraus

PURPL group, Purdue University

PURPL
Purdue University Rendering & Perceptualization Lab

Overview

- Parallel volume rendering with a single GPU
- Implementing ray casting for a GPU
  - *Basics*
  - *Optimizations*
- Published systems
- Open questions

Parallel Volume Rendering with a single Graphics Processing Unit (GPU) (1)

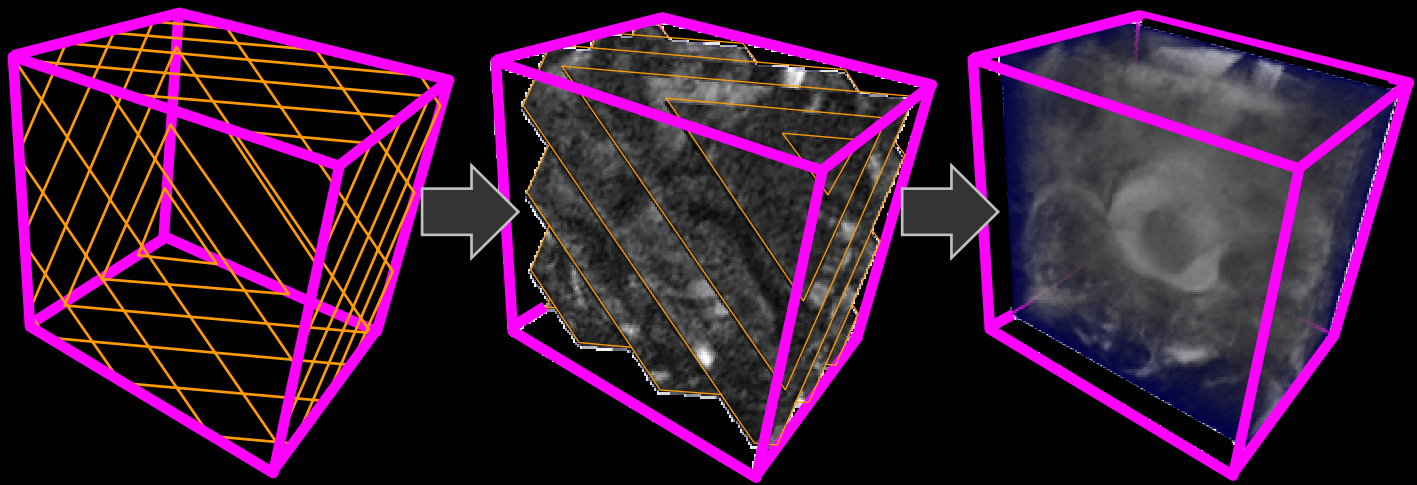
Even a single GPU is a parallel computer:

- Multiple vertex and pixel pipelines
  (e.g. “8 pixel pipelines” of QuadroFX and FireGL)
- APIs support this parallelism
  (independent processing of vertices and fragments)
- When implementing algorithms for a GPU, it’s crucial to exploit this parallelism

Parallel Volume Rendering with a single Graphics Processing Unit (GPU) (2)

Many volume rendering algorithms cannot fully exploit these parallel pixel pipelines:

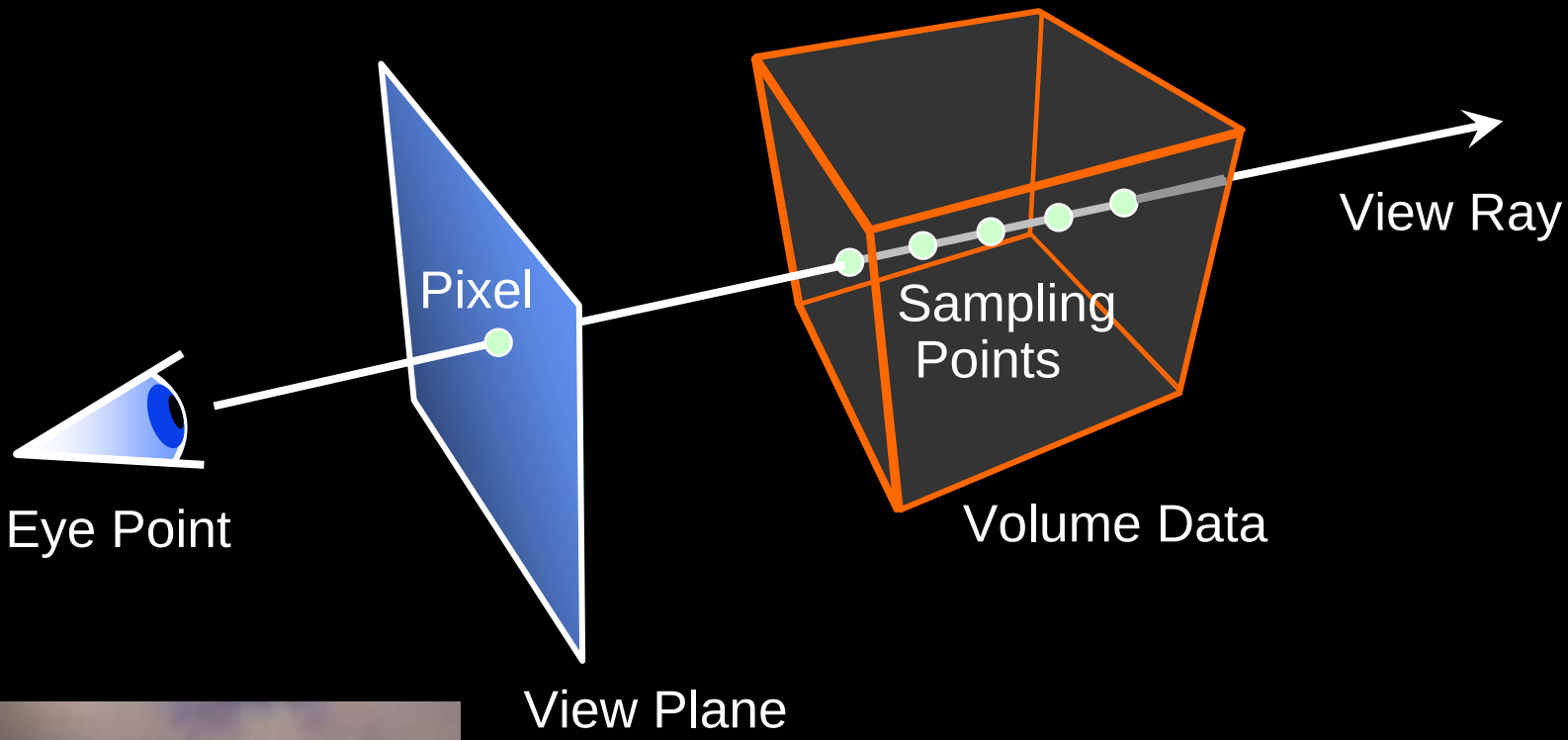
- Cell projection: requires visibility sorting of cells
- Textured slices: rasterize all fragments of all slices



Parallel Volume Rendering with a single Graphics Processing Unit (GPU) (3)

Ray casting can exploit parallel pixel pipes:

- Like ray tracing, ray casting is embarrassingly parallel



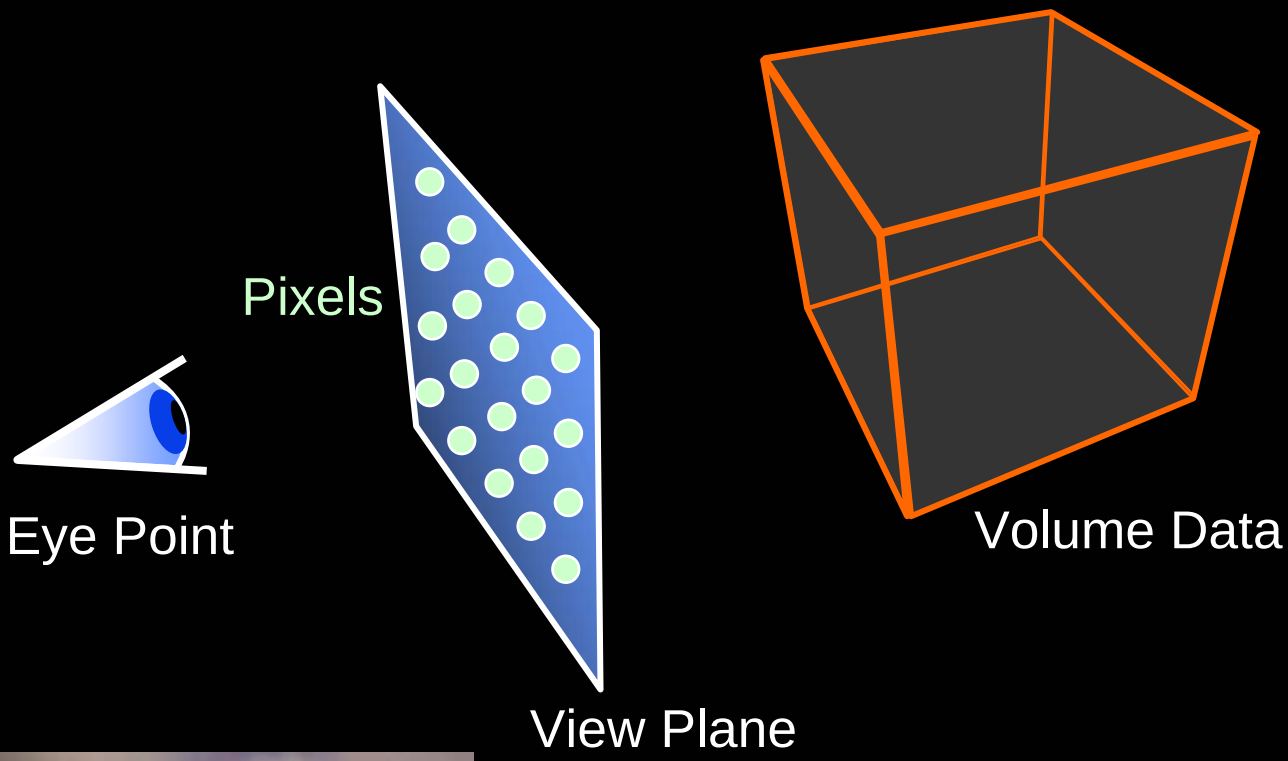
Parallel Volume Rendering with a single Graphics Processing Unit (GPU) (4)

More benefits of ray casting on a GPU:

- Applicable to uniform and tetrahedral meshes
- Important optimizations can be implemented:
  - *early ray termination*
  - *empty space leaping*
  - *adaptive sampling distance*

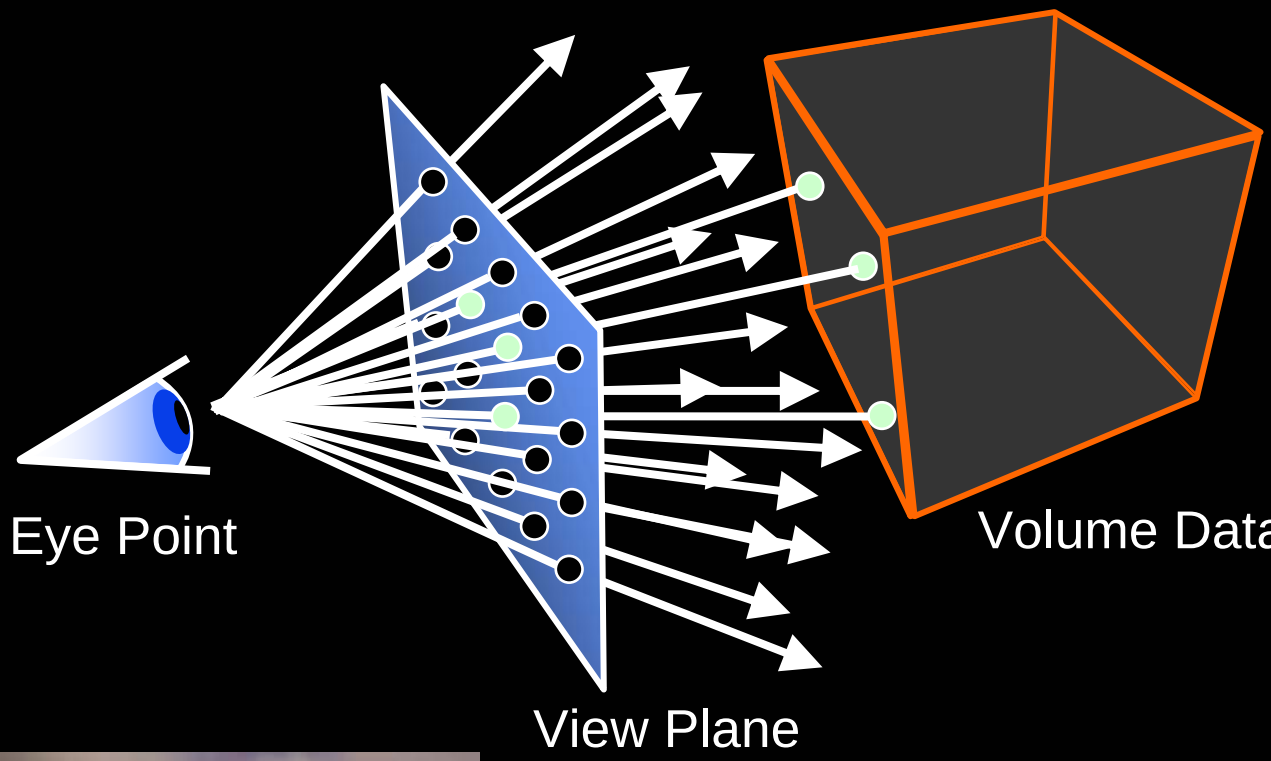
Implementing Ray Casting for a GPU: Basics (1)

Multi-pass approach: Render screen-filling rectangles to call a program for each pixel



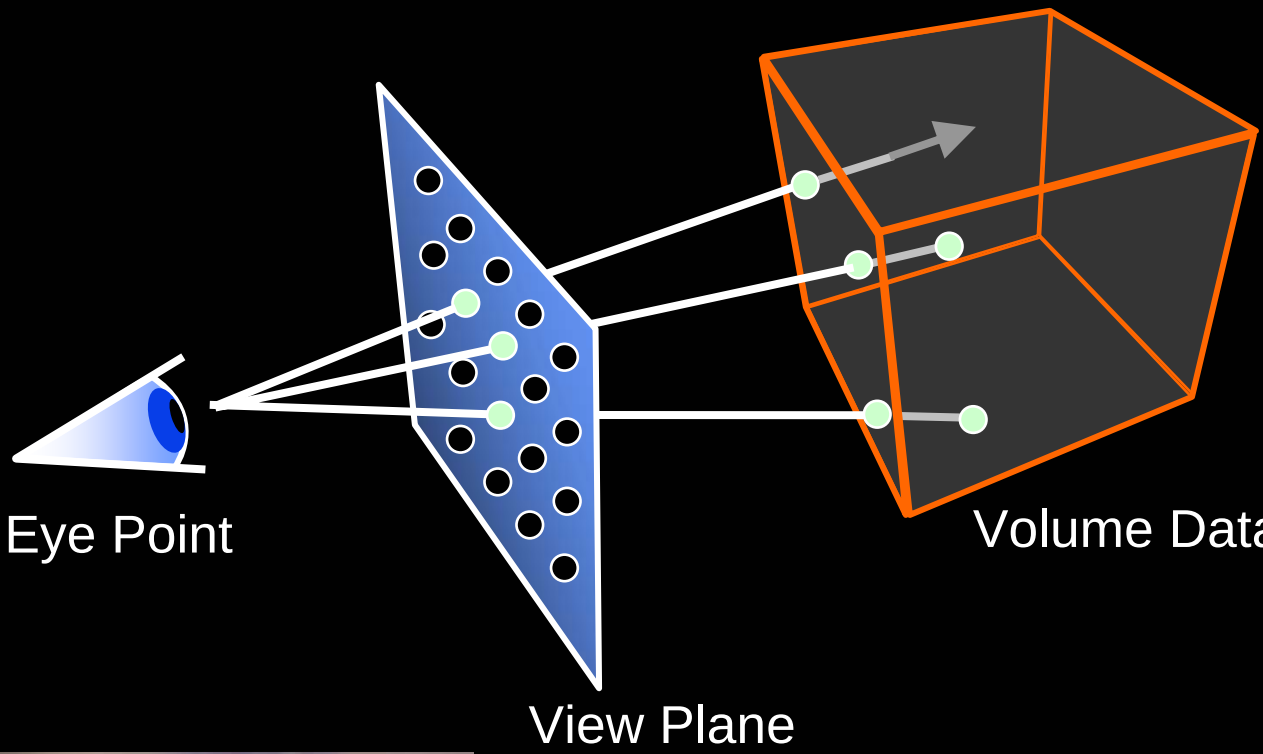
Implementing Ray Casting for a GPU: Basics (2)

Initialize pixels with first intersection of the rays with the volume data



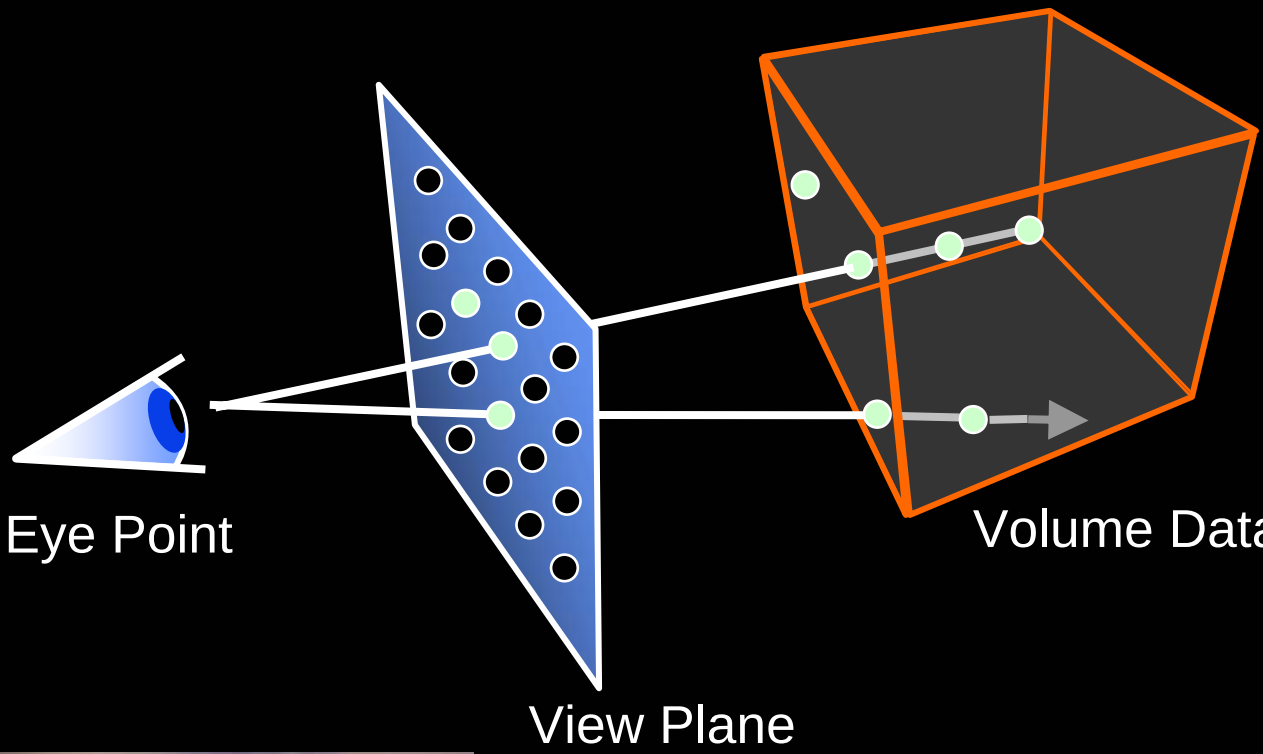
Implementing Ray Casting for a GPU: Basics (3a)

In each pass, propagate view rays and store accumulated color and sampling position



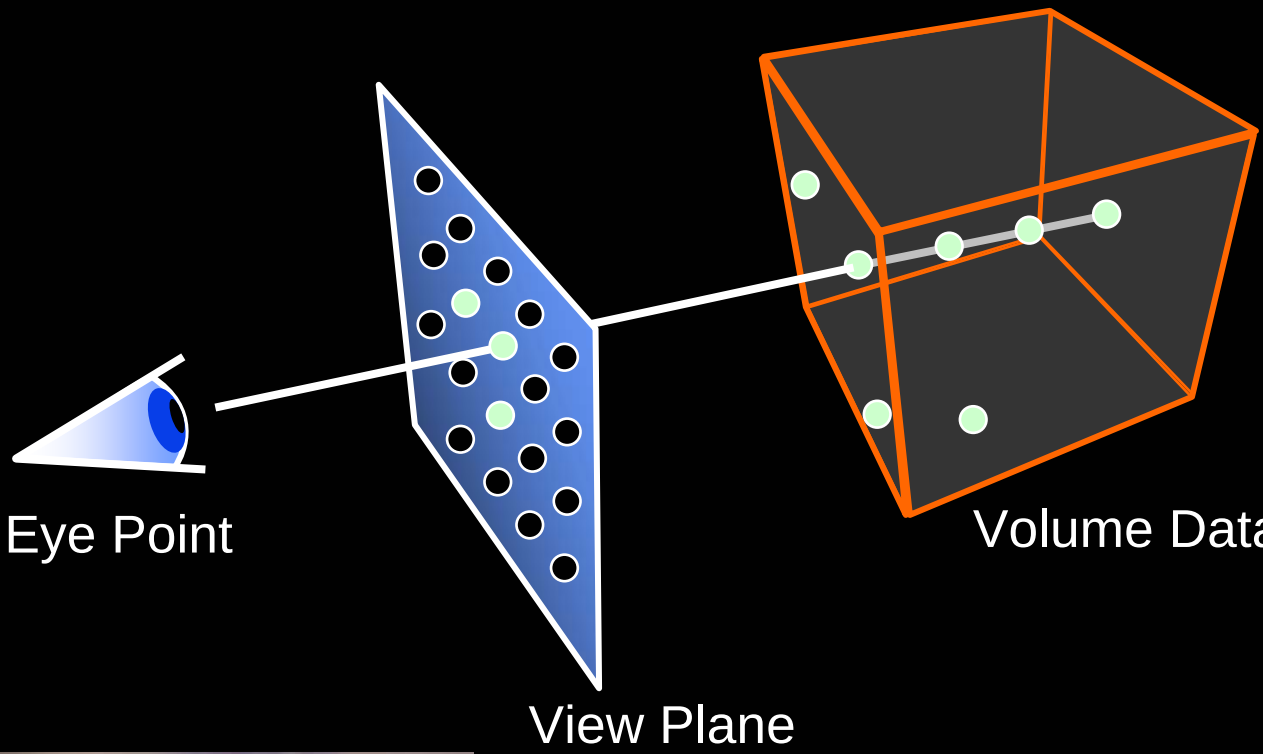
Implementing Ray Casting for a GPU: Basics (3b)

In each pass, propagate view rays and store accumulated color and sampling position



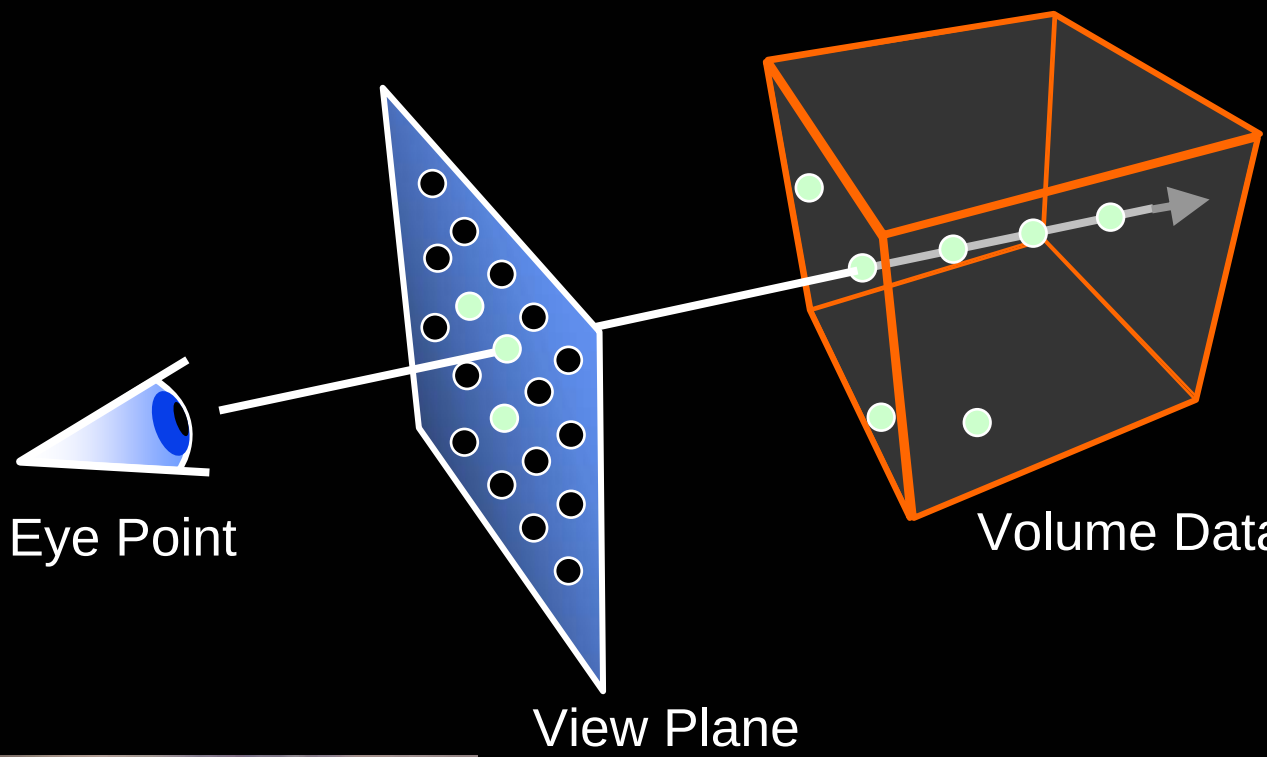
Implementing Ray Casting for a GPU: Basics (3c)

In each pass, propagate view rays and store accumulated color and sampling position



Implementing Ray Casting for a GPU: Basics (4)

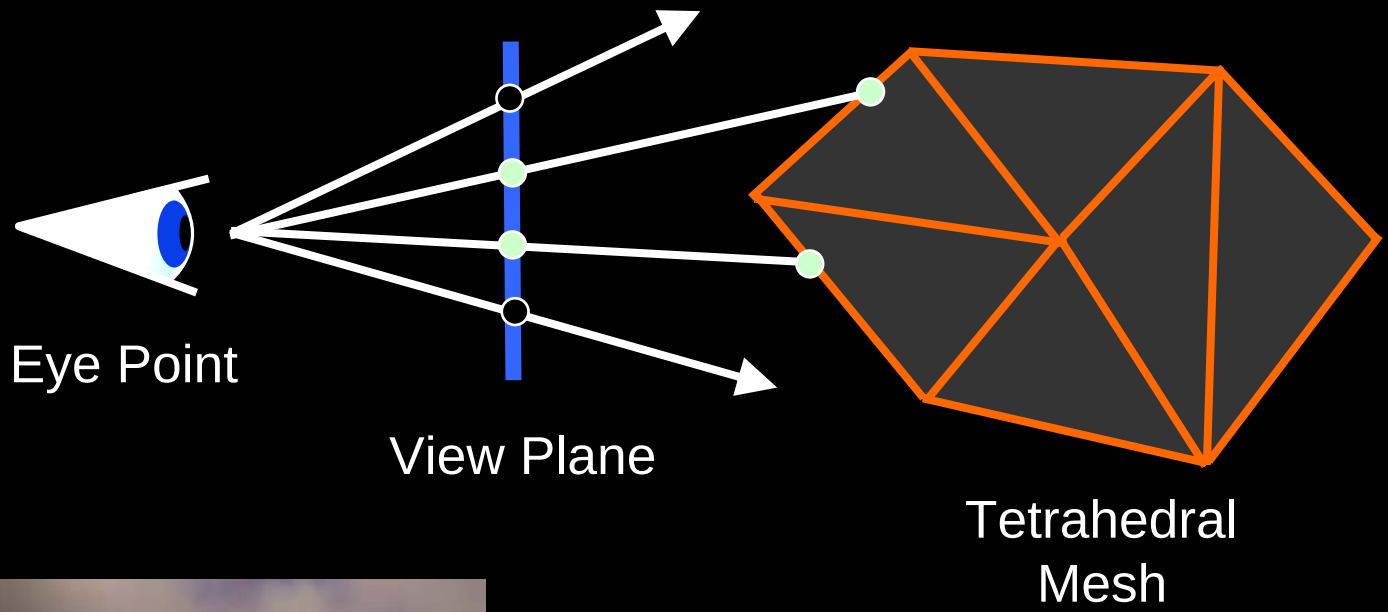
Stop when all rays have left the volume



Implementing Ray Casting for a GPU: Basics (5a)

Sampling of volume data:

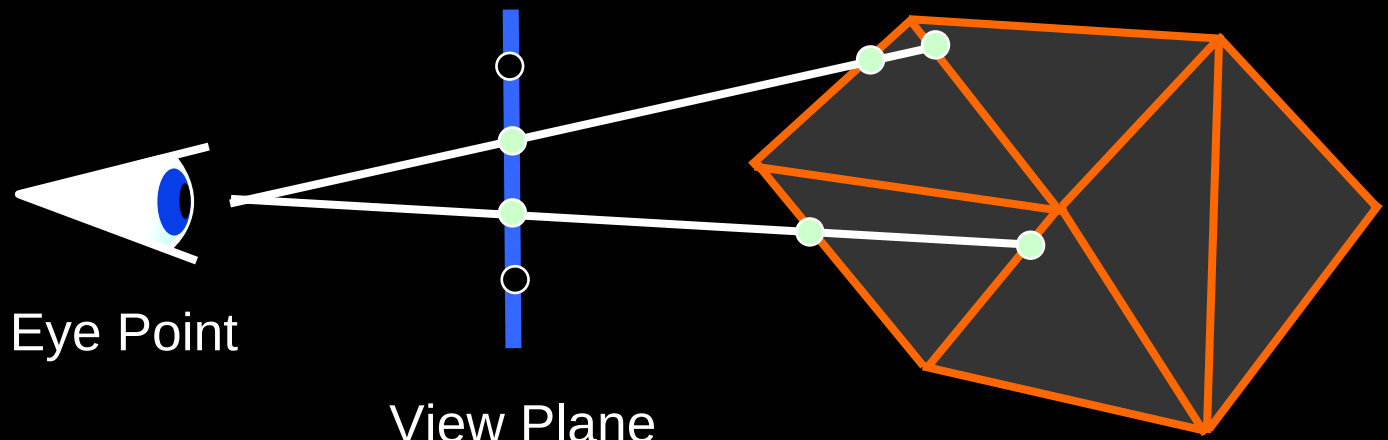
- Just a 3D-texture lookup for uniform data
- For tetrahedral cells: sample at cell boundaries



Implementing Ray Casting for a GPU: Basics (5b)

Sampling of volume data:

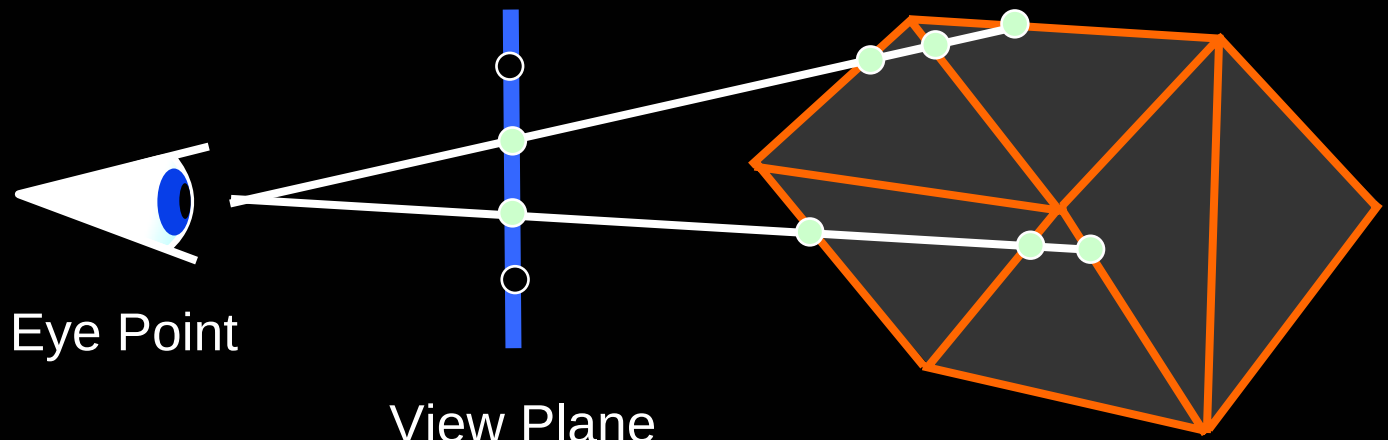
- Just a 3D-texture lookup for uniform data
- For tetrahedral cells: sample at cell boundaries



Implementing Ray Casting for a GPU: Basics (5c)

Sampling of volume data:

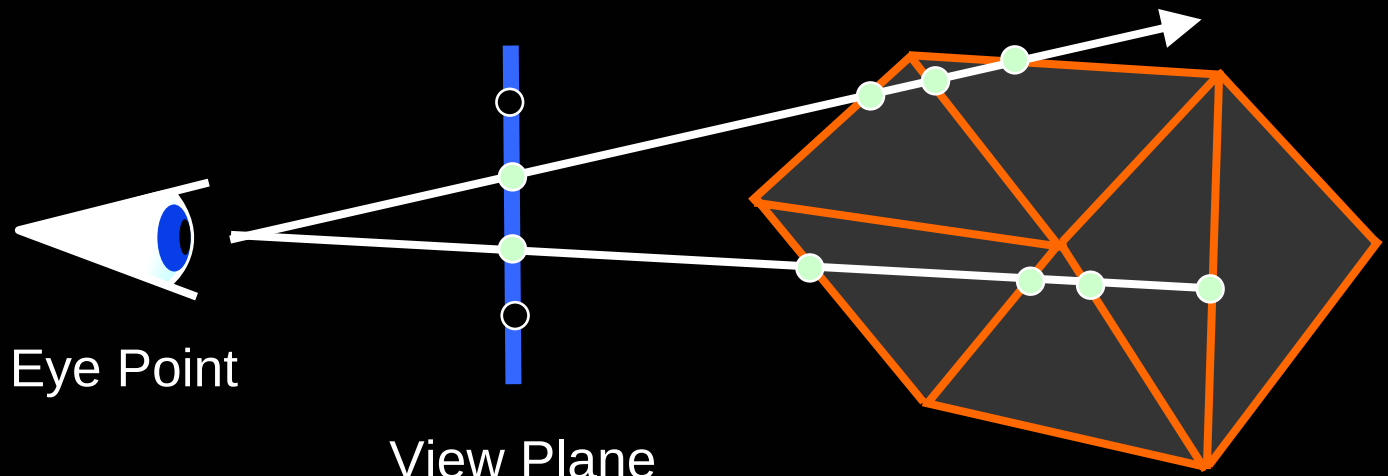
- Just a 3D-texture lookup for uniform data
- For tetrahedral cells: sample at cell boundaries



Implementing Ray Casting for a GPU: Basics (5d)

Sampling of volume data:

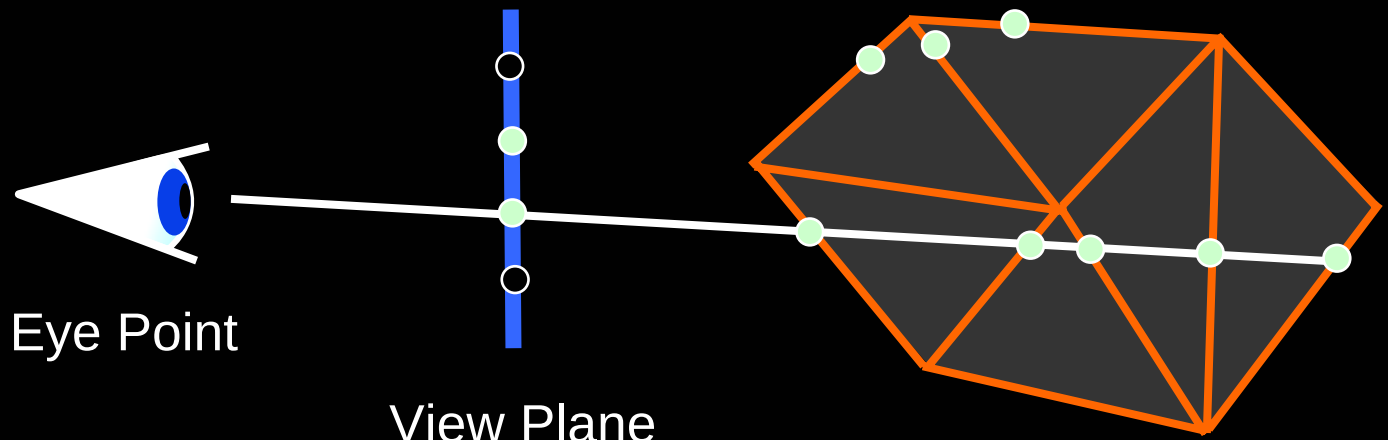
- Just a 3D-texture lookup for uniform data
- For tetrahedral cells: sample at cell boundaries



Implementing Ray Casting for a GPU: Basics (5e)

Sampling of volume data:

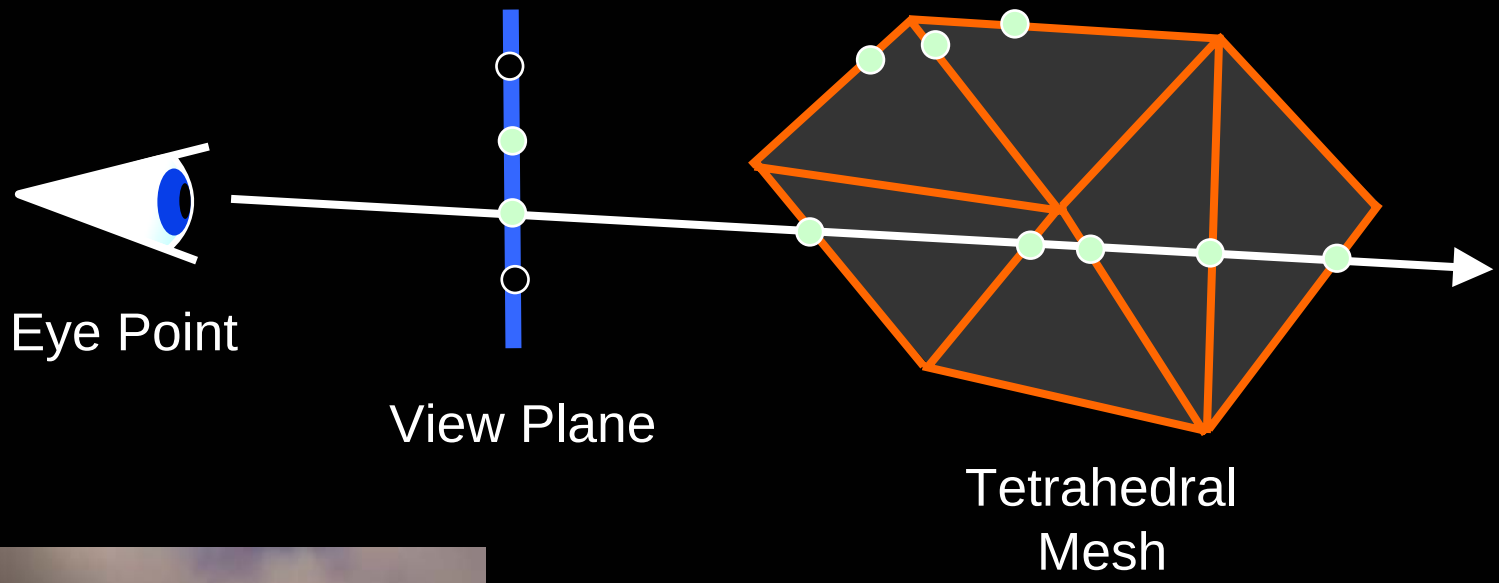
- Just a 3D-texture lookup for uniform data
- For tetrahedral cells: sample at cell boundaries



Implementing Ray Casting for a GPU: Basics (5f)

Sampling of volume data:

- Just a 3D-texture lookup for uniform meshes
- For tetrahedral cells: sample at cell boundaries



Implementing Ray Casting for a GPU: Optimizations (1)

Overview:

- Avoid work for rays that don't intersect the volume
- Avoid work for rays that have accumulated full opacity
- Quickly test whether all rays have left the volume
- Quickly go through empty regions
- Adapt sampling distance to data

Implementing Ray Casting for a GPU: Optimizations (2)

Avoid work for rays that don't intersect or have left the volume:

- Idea: avoid rasterization of corresponding pixels by depth tests
- Advantage: exploits early-depth tests to avoid execution of fragment program
- Requires updates of the depth-buffer (not after each pass but from time to time)

Implementing Ray Casting for a GPU: Optimizations (3)

Avoid work for rays that have accumulated full opacity (early ray termination):

- Idea: treat them as if they have left the volume
- Advantage: also exploits early-depth tests for these pixels

Implementing Ray Casting for a GPU: Optimizations (4)

Quickly test whether all rays have left the volume:

- Observation: once all rays have left the volume, all fragments fail the depth test
- Just use an “occlusion query” to determine whether all fragments have failed the depth test!

Implementing Ray Casting for a GPU: Optimizations (5)

Quickly go through empty regions (empty space leaping):

- Increase sampling distance in empty regions by lookup in a 3D-texture that encodes empty regions
- Works only for uniform meshes
- Generalization: Adapt sampling distance to data

Published Systems (1)

Purcell, Buck, Mark, and Hanrahan. Ray Tracing on Programmable Graphics Hardware. SIGGRAPH 2002.

- "Ray tracing" (not ray casting)
- Discusses concepts, not an actual implementation
- Features several similar techniques, e.g., for
  - *invoking fragment programs*
  - *ray termination*

Published Systems (2)

Röttger, Guthe, Weiskopf, and Ertl. Smart Hardware-Accelerated Volume Rendering. VisSym 2003.

- Ray casting on a GPU for uniform meshes
- Features:
  - *early ray termination*
  - *pre-integrated classification*
  - *adaptive sampling distance*

Published Systems (3)

Krüger and Westermann. Acceleration Techniques for GPU-based Volume Rendering. Visualization 2003 (Session P10).

- Ray casting on a GPU for uniform meshes
- Features:
  - *early ray termination*
  - *empty-space leaping*
  - *multiple sampling points per pass*

Published Systems (4)

Weiler, Kraus, Merz, and Ertl. Hardware-Based Ray Casting for Tetrahedral Meshes. Visualization 2003 (Session P11).

- Ray casting on a GPU for tetrahedral meshes
- Features:
  - *early ray termination*
  - *pre-integrated classification*
  - *imaginary tetrahedra for non-convex meshes*

Open Questions

Ray casting on GPUs is a rapidly developing method, thus there are many open question:

- What about worst-case performances?
- What about limited texture memory?
- What about hierarchical methods?
- What about multiple GPUs?

More Open Question?

Thanks!