CSE 167:  
Introduction to Computer Graphics  
Lecture #17: Volume Rendering  

Jürgen P. Schulze, Ph.D.  
University of California, San Diego  
Fall Quarter 2010
Announcements

- Second midterm grades on-line
  - 1st Midterm: min 28, max 85, average 59
  - 2nd Midterm: min 42, max 95, average 68
- Please check Gradesource for accuracy. Everything but the final project should be there.
- Final project to be presented on **Friday, Dec 3rd, between 2 and 4pm in room 4140**
  - No late submissions accepted
Lecture Overview

- Midterm Review
- Volume Rendering
Lecture Overview

- Midterm Review
- Volume Rendering
Rendering Methods

There are two categories of volume rendering algorithms:

1. Ray casting algorithms (Object Order)
   - Basic ray-casting
   - Using octrees

2. Plane Composing (Image Order)
   - Basic slicing with 2D textures
   - Shear-Warp factorization
   - Translucent textures with image-aligned 3D textures
Ray Casting

- Software Solution

- **Image Plane**
- **Data Set**

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

  **High Computational Load**
Plane Compositing

Proxy geometry (Polygonal Slices)
Compositing

- **Maximum Intensity Projection**
  - No emission/absorption
  - Simply compute maximum value along a ray
2D Textures

- Draw the volume as a stack of 2D textures

**Bilinear Interpolation in Hardware**

- Decomposition into axis-aligned slices

- 3 copies of the data set in memory
2D Textures: Drawbacks

- Sampling rate is inconsistent

![Diagrams showing inconsistency in sampling rate](image)

- Emission/absorption slightly incorrect

- **Super-sampling on-the-fly impossible**
3D Textures

For each fragment:
interpolate the texture coordinates
(barycentric)

Texture-Lookup:
interpolate the texture color
(trilinear)
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
Cube-Slice Intersection

**Question:** Can we compute this in a vertex program?

**Vertex program:**

*Input:* 6 Vertices  
*Output:* 6 Vertices

- **P0:** Intersection with red path
- **P1:** Intersection with dotted red edge or P0
- **P2:** Intersection with green path
- **P3:** Intersection with dotted green edge or P1
- **P4:** Intersection with blue path
- **P5:** Intersection with dotted blue edge or P2
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

Inefficient!
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!*
Videos

- Human head, rendered with 3D texture:
  http://www.youtube.com/watch?v=94_Zs_6AmQw&feature=related

- GigaVoxels:
  http://www.youtube.com/watch?v=HScYuRhgEJw&feature=related

- Future Gaming Technology:
  http://www.youtube.com/watch?v=mySER0p9F64&feature=related
Thank You

- Good luck with your final project!
- Happy Holidays!
- Happy New Year!

- See you tomorrow at 2pm