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

Jürgen P. Schulze, Ph.D.
University of California, San Diego
Fall Quarter 2012

Announcements

- Thursday, Dec 13: Final project presentations in EBU-3B room 1202, 3-6pm
- Midterms
  - Verify total score on front sheet is sum of individual scores
  - Cross-check total score with Ted
  - If exam kept past end of today's office hour, cannot dispute grade later

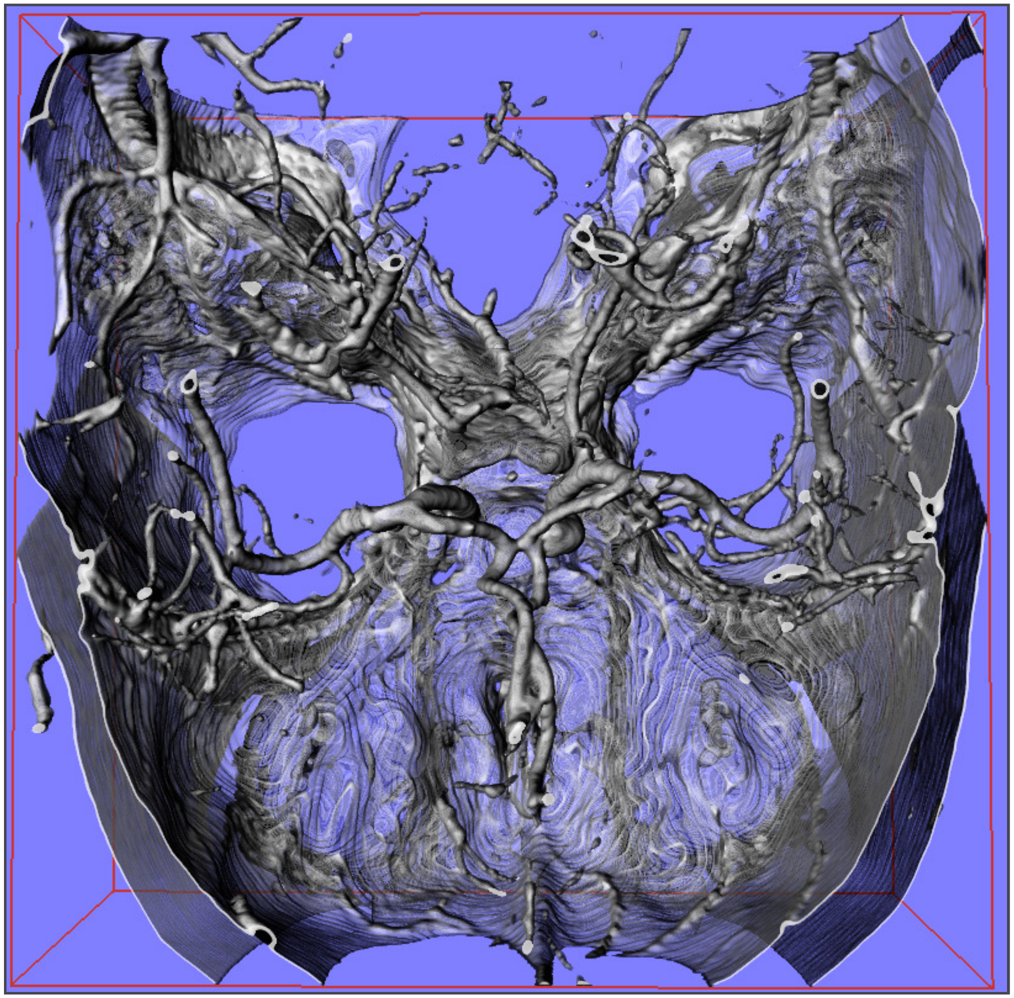
Midterm Statistics

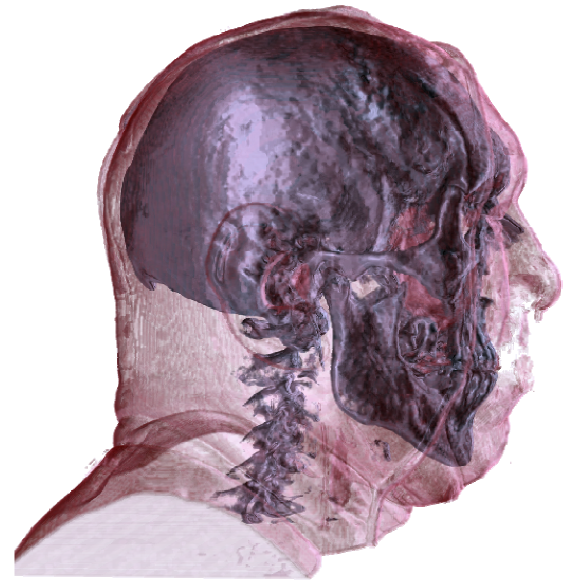
	Midterm 1	Midterm 2
# Submissions	53	49
Average score	70.5	69.9
Median score	72.5	69.5
Highest score	95	98
Lowest score	39.5	26
Standard Deviation	14.2	13.3

Lecture Overview

- Volume Rendering
- SSAO

Applications: Medicine





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

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

This and some of the following slides are from a Eurographics 2006 course by Dr. Christof Rezk Salama, Computer Graphics and Multimedia Group, University of Siegen, Germany

Translucent Objects

Source: GPU Gems

Methods of Representation

- Polygonal - Triangle Mesh
- Freeforms - parametric curves, patches…
- Solid Modelling - CSG (Constructive Solid Geometry)
- Direct Volume Rendering

Why Direct Volume Rendering?

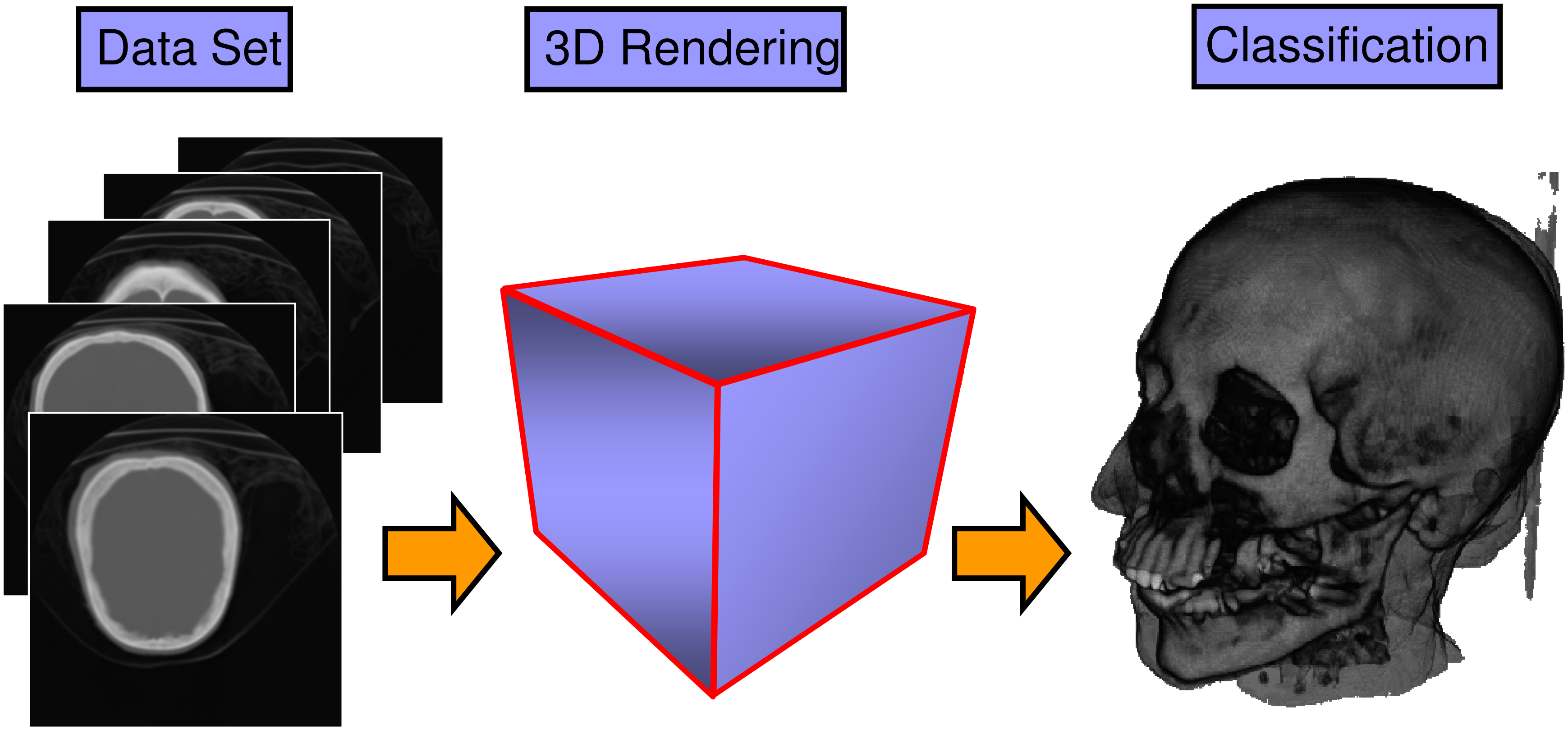
Pros

- Natural representation of CT/MRI images
- Transparency effects (Fire, Smoke…)
- High quality

Cons

- Huge data sets
- Computationally expensive
- Cannot be embedded easily into polygonal scene

Volume Rendering Outline



- in real-time on
  commodity graphics hardware

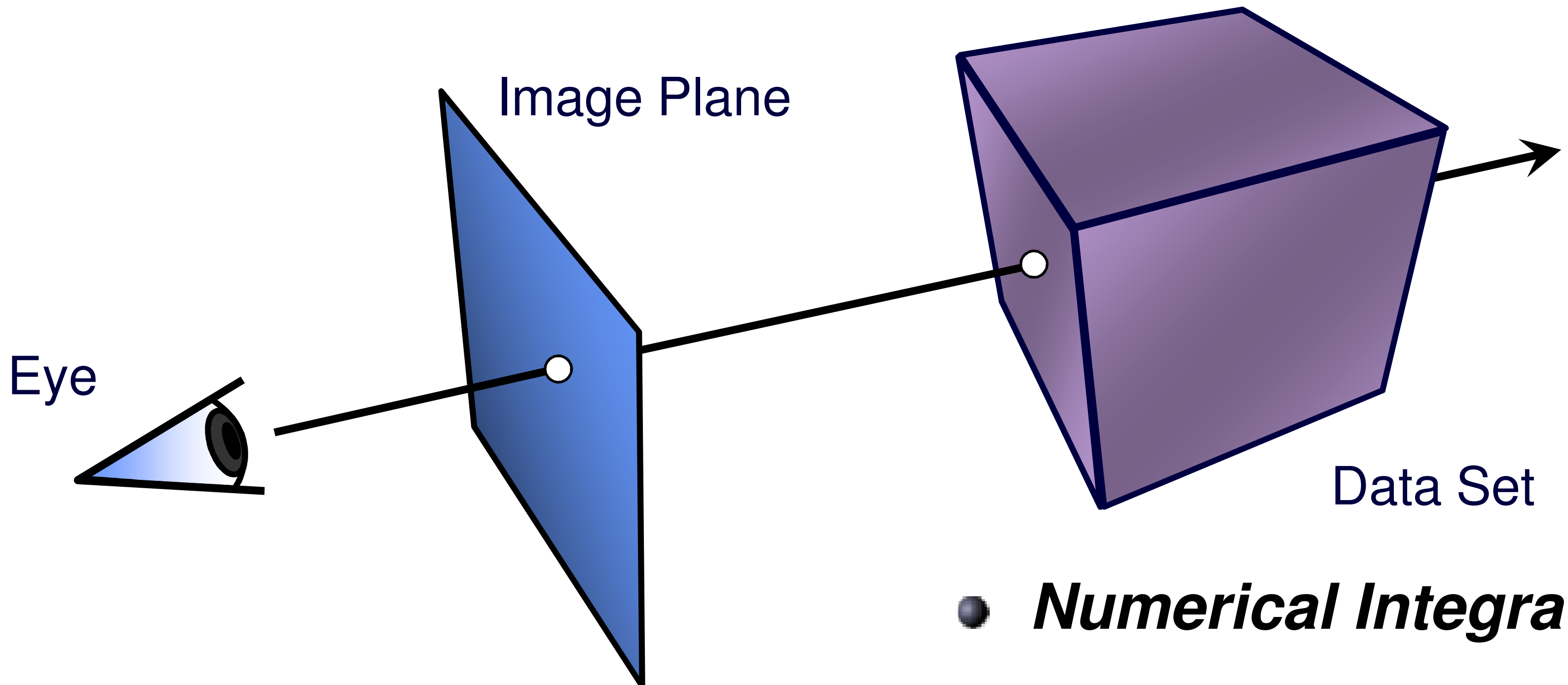
Rendering Methods

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



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

➡ ***High Computational Load***

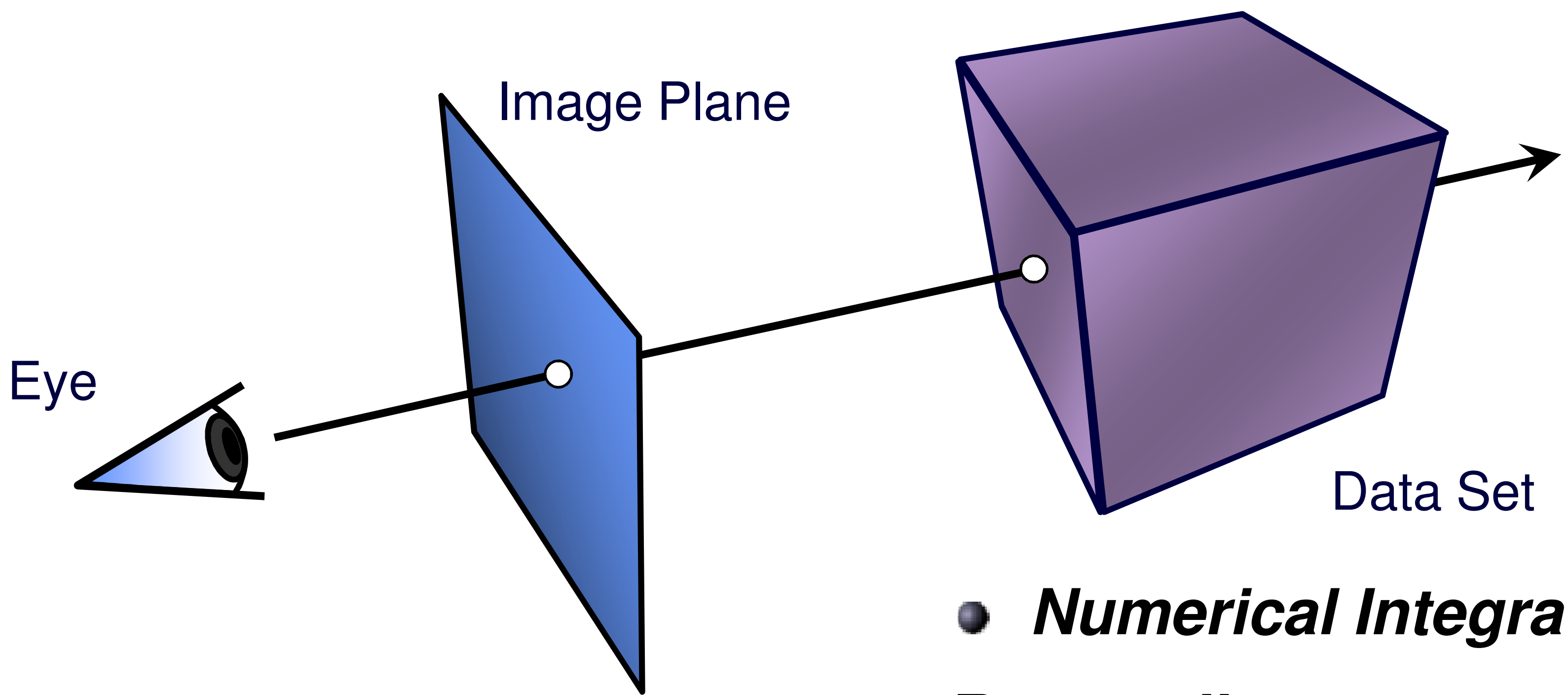
Rendering Methods

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

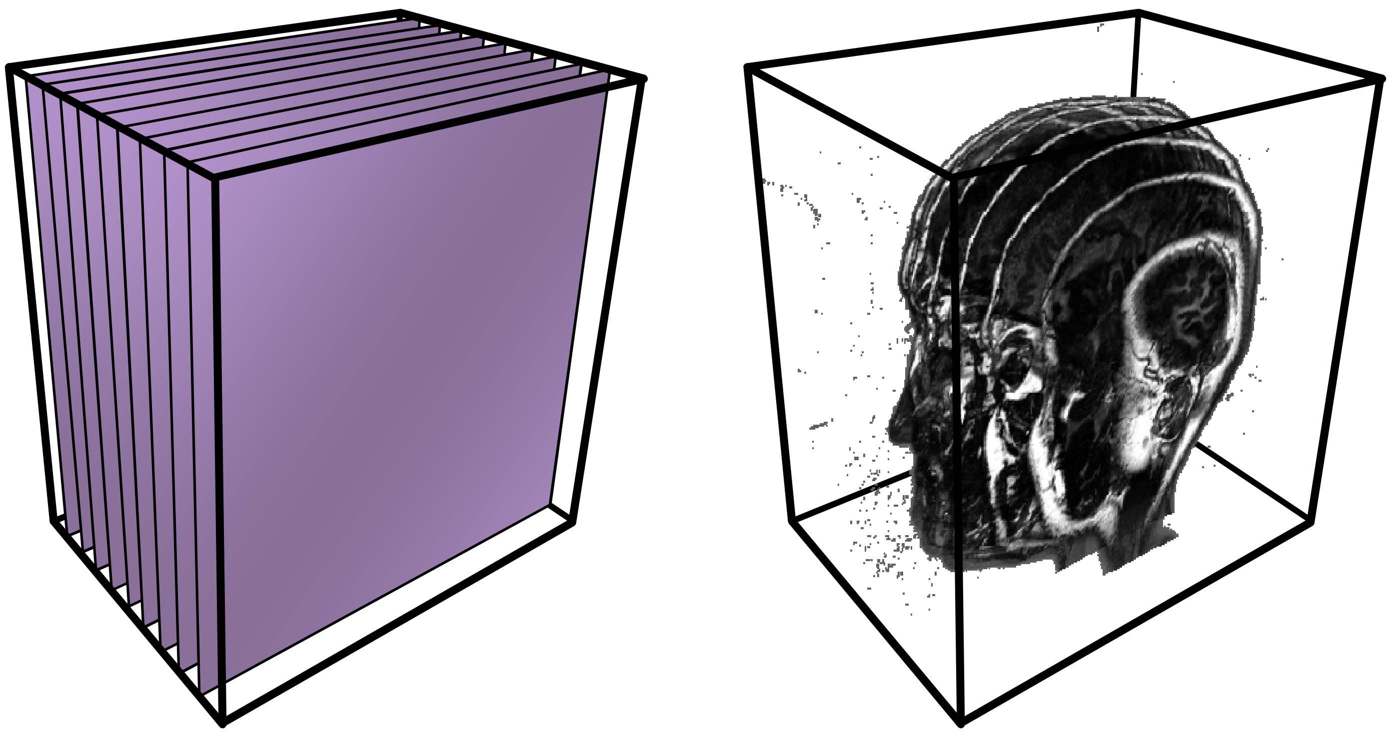


- ***Numerical Integration***
- ***Resampling***

➡ ***High Computational Load***

Plane Compositing

- Proxy geometry (Polygonal Slices)

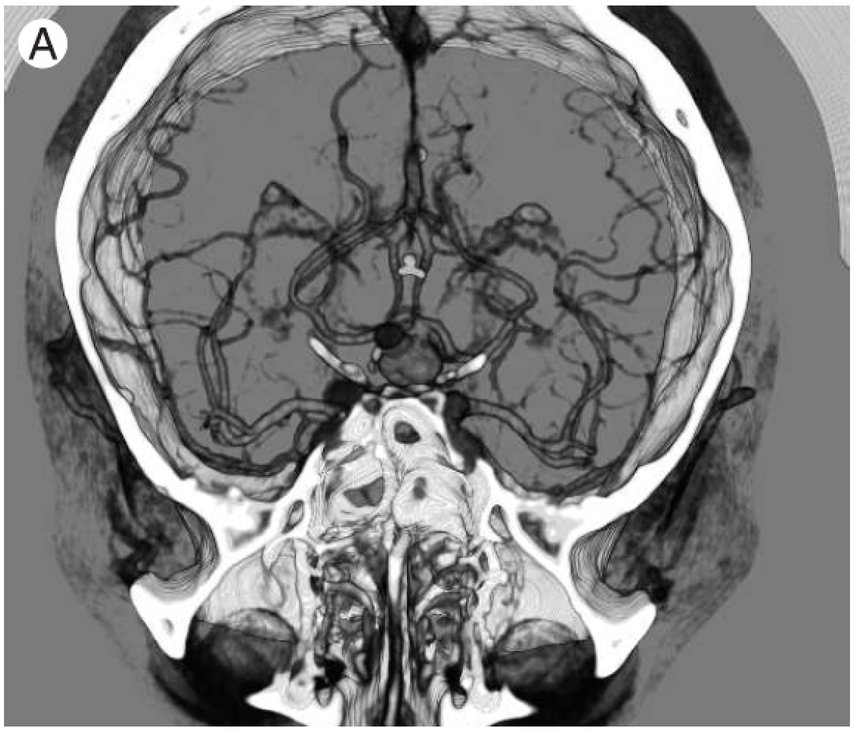


Compositing

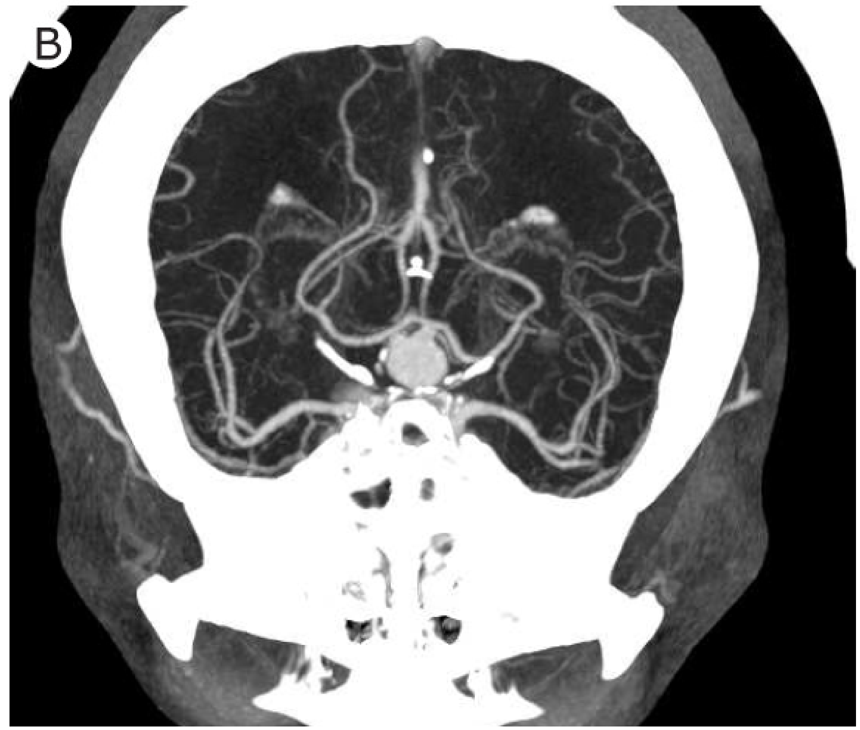
- ***Maximum Intensity Projection***

  No emission/absorption

  Simply compute maximum value along a ray



Emission/Absorption

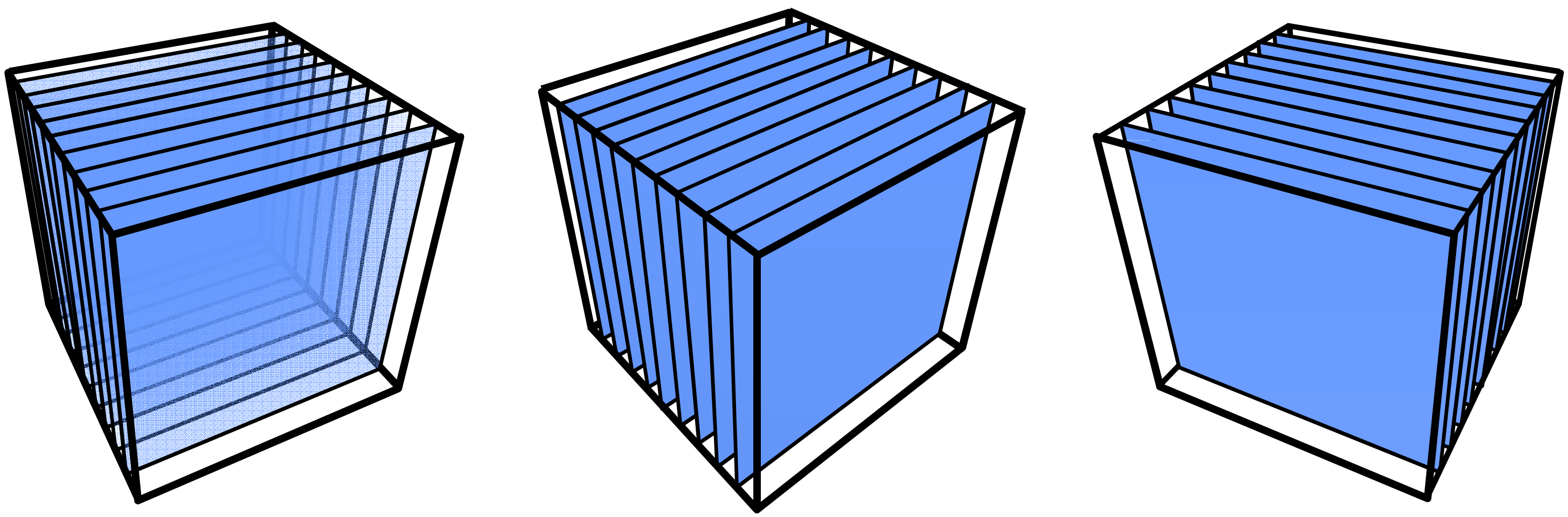


Maximum Intensity Projection

2D Textures

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

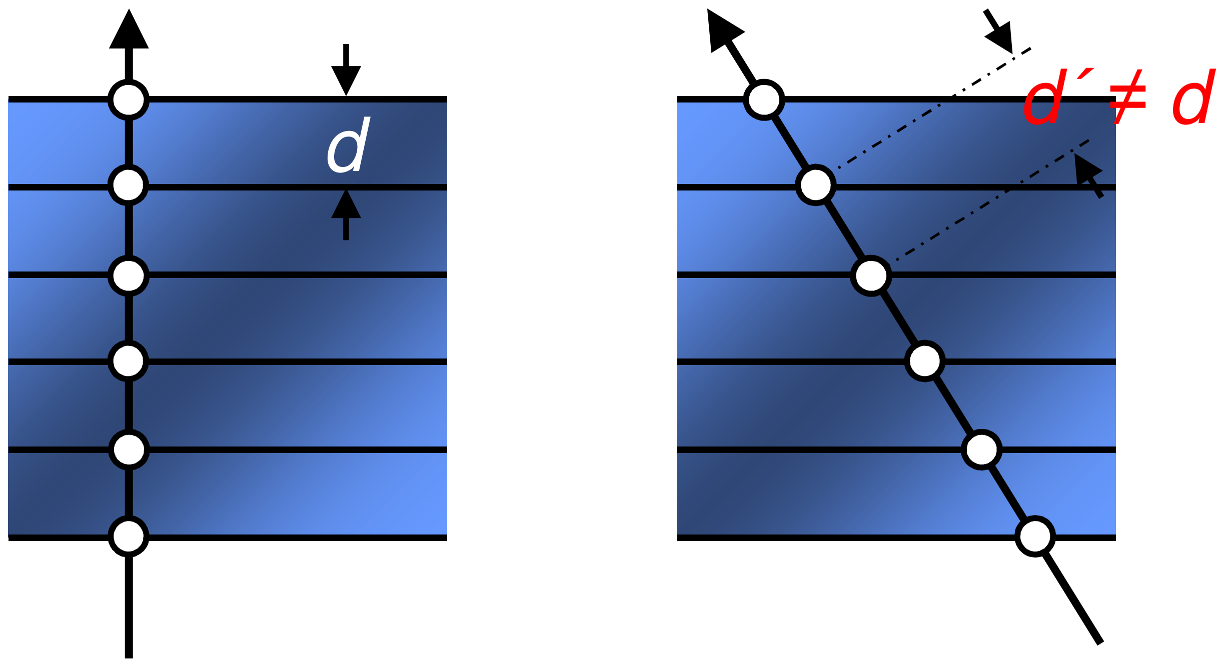
➡ Decompostition into axis-aligned slices



- 3 copies of the data set in memory

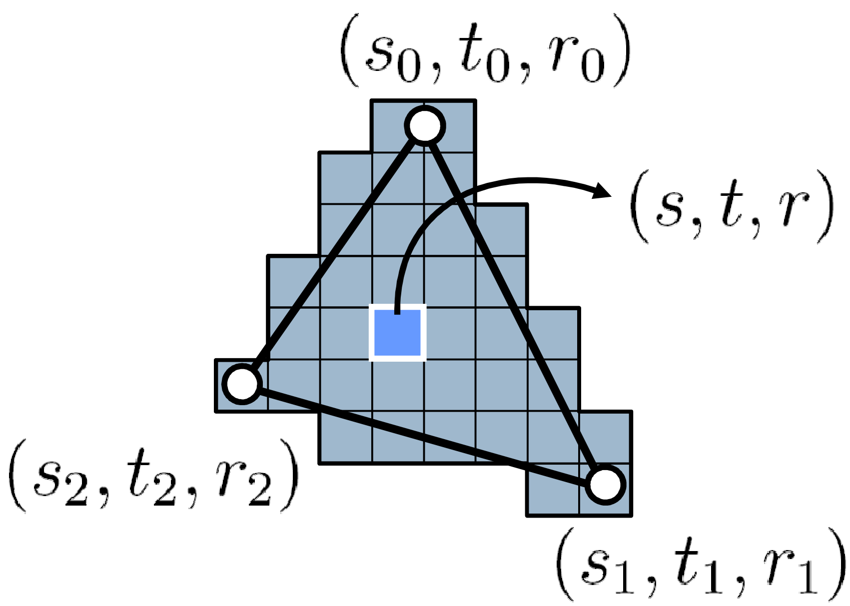
2D Textures: Drawbacks

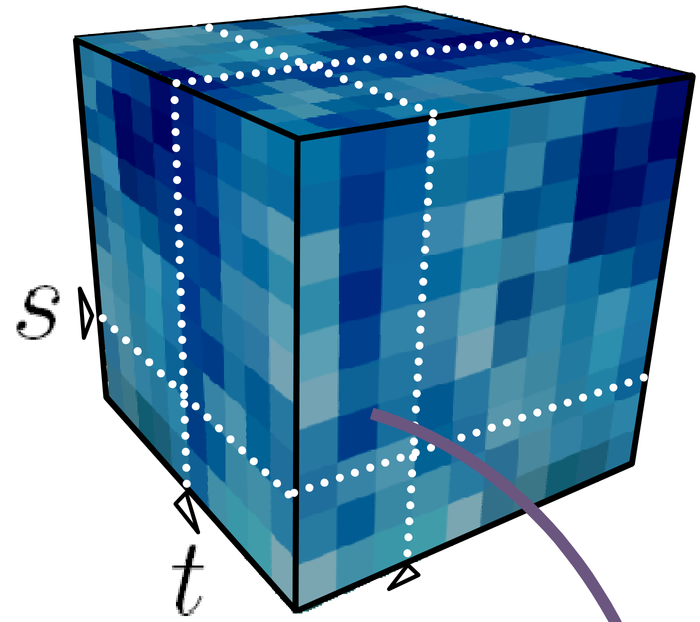
- Sampling rate is inconsistent



- 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)

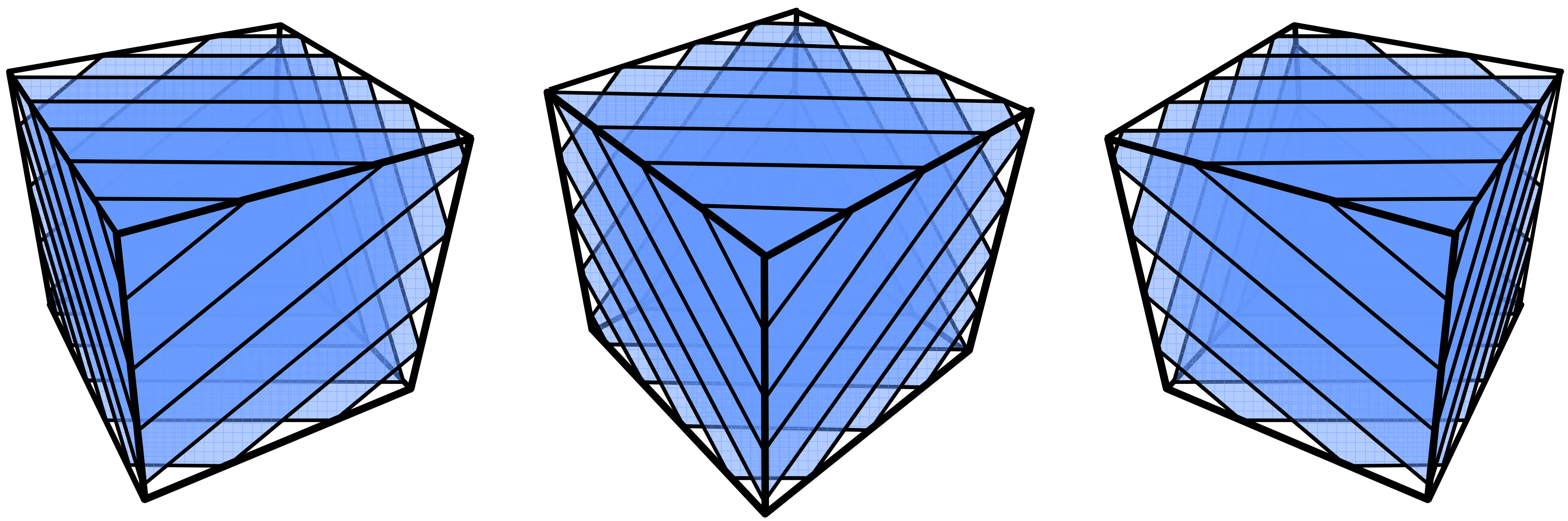
R G B A

3D Textures

3D Texture: Volumetric Texture Object

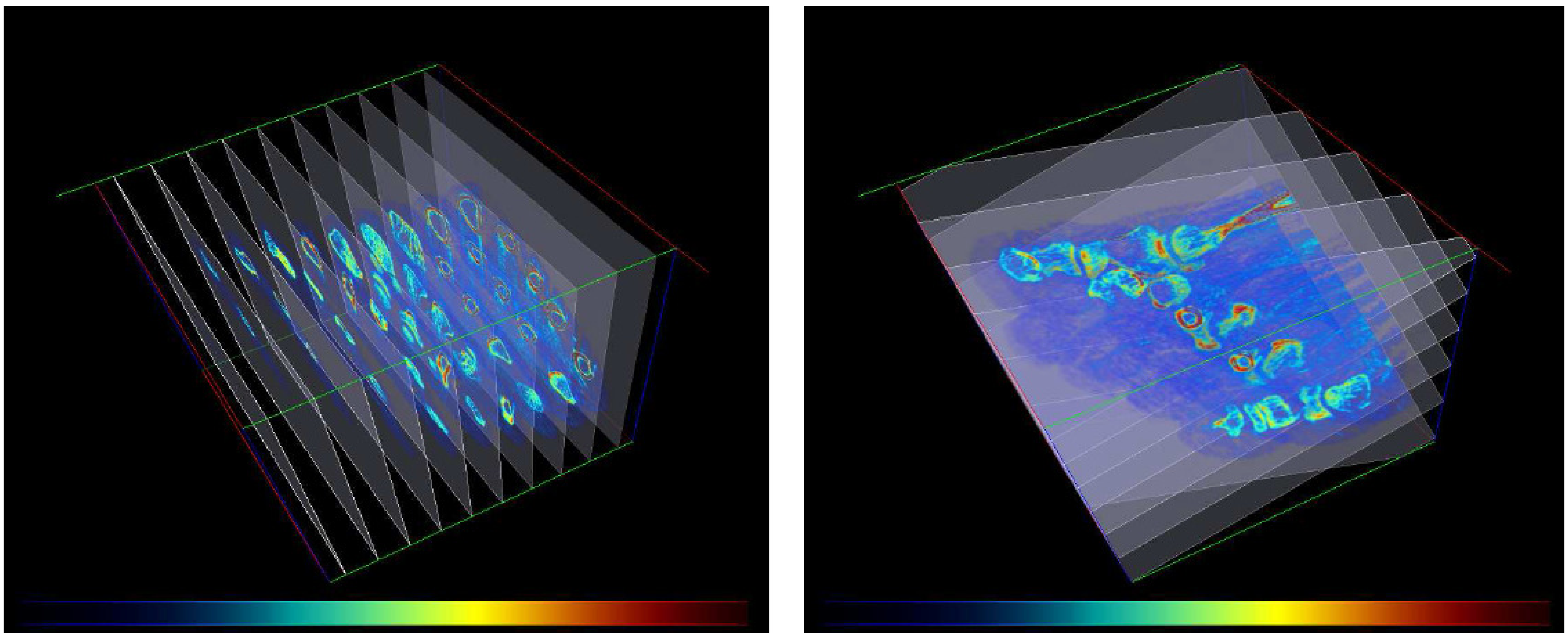
- Trilinear Interpolation in Hardware

➡ Slices parallel to the image plane



- One large texture block in memory

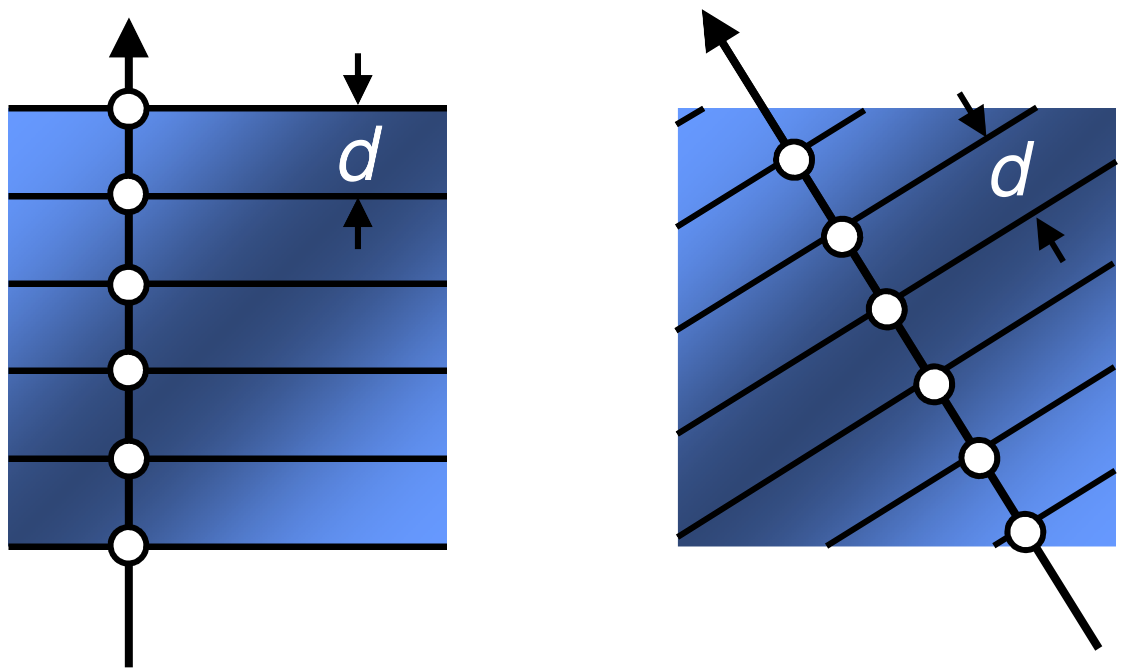
Comparison of 2D with 3D Texturing



Left: 2D textures, right: 3D textures
[Lewiner2006]

Resampling via 3D Textures

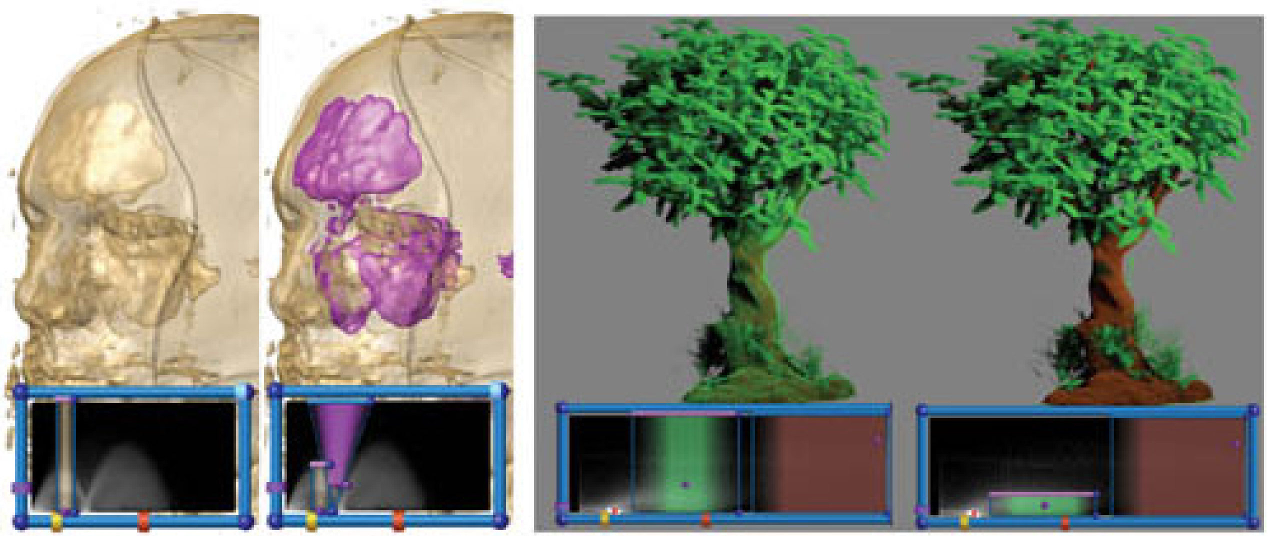
- ***Sampling rate is constant***



- Supersampling by increasing the number of slices

Transfer Functions

- 1D transfer function: maps RGBA to each data value (see a and c below).
- 2D transfer function: maps RGBA to each combination of data value and gradient magnitude (see b and d below).



(a) (b) (c) (d)

From GPU Gems

Shadows


Volume rendering with shadows
(from GPU Gems)

Implementation: Loading a 3D Texture

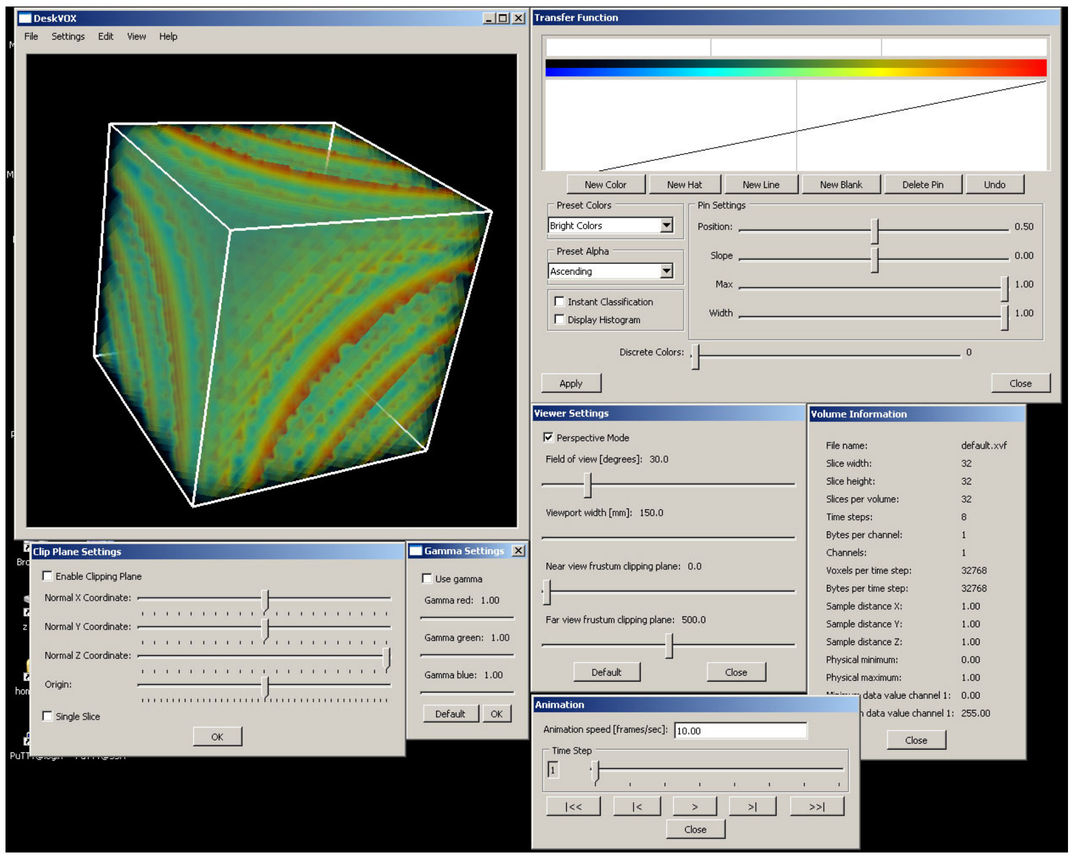
- // init the 3D texture
- glEnable(GL_TEXTURE_3D_EXT);
- glGenTextures(1, &tex_glid);
- glBindTexture(GL_TEXTURE_3D_EXT, tex_glid);
- // texture environment setup
- glTexParameteri(GL_TEXTURE_3D_EXT, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
- glTexParameteri(GL_TEXTURE_3D_EXT, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
- glTexParameteri(GL_TEXTURE_3D_EXT, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
- glTexParameteri(GL_TEXTURE_3D_EXT, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
- glTexParameteri(GL_TEXTURE_3D_EXT, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
- // load the texture image
- glTexImage3DEXT(GL_TEXTURE_3D_EXT, // target
- 0, // level
- GL_RGBA, // color storage
- (int) tex_ni(), // width
- (int) tex_nj(), // height
- (int) tex_nk(), // depth
- 0, // border
- GL_COLOR_INDEX, // format
- GL_FLOAT, // type
- _texture); // allocated texture buffer
- glPixelTransferi(GL_MAP_COLOR, GL_FALSE);

Videos

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

Demo: DeskVox

- Created at IVL/Calit2
  - http://ivl.calit2.net/wiki/index.php/VOX_and_Virvo



Lecture Overview

- Volume Rendering
- SSAO

Screen Space Ambient Occlusion

- Screen Space Ambient Occlusion = SSAO
- Rendering technique for approximating ambient occlusion in real time
- Developed by Vladimir Kajalin while working at Crytek
- First use in 2007 PC game Crysis

SSAO component

SSAO Demo

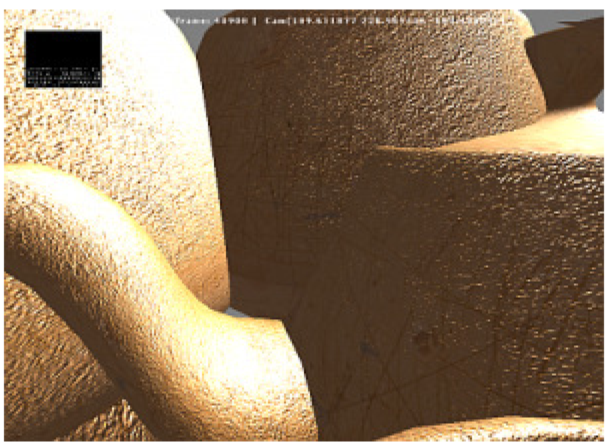
- Screen Space Ambient Occlusion (SSAO) in Crysis
  - http://www.youtube.com/watch?v=ifdAILHTcZk

Basic SSAO Algorithm

- Copy frame buffer to texture
- Pixel shader samples depth values around current pixel and tries to compute amount of occlusion
- Occlusion depends on depth difference between sampled point and current point
- Nvidia's documentation:
  - http://developer.download.nvidia.com/SDK/10.5/direct3d/Source/ScreenSpaceAO/doc/ScreenSpaceAO.pdf
- SSAO shader code from Crysis:
  - http://69.163.227.177/forum.php?mod=viewthread&tid=772
- Another implementation:
  - http://www.gamerendering.com/2009/01/14/ssao/

SSAO With Normals

- First pass: render depth information in a texture's alpha channel and scene normals in the RGB channels
- Use this information to render SSAO in a render target
- It uses the normals and pixel depth to compute the occlusion between current pixel and several samples around that pixel, chosen by sampling texels from depth map around it.



No SSAO

With SSAO

SSAO Discussion

- Advantages:
  - Independent from scene complexity.
  - No pre-processing, no memory allocation in RAM
  - Works with dynamic scenes
  - Works in the same way for every pixel
  - No CPU usage: executed completely on GPU
- Disadvantages:
  - Local and view-dependent (dependent on adjacent texel depths)
  - Hard to correctly smooth/blur out noise without interfering with depth discontinuities, such as object edges