High-Quality Volume Graphics on Consumer PC Hardware

Introduction

Motivation (1)
Scientific visualization
- DVR, iso, shading, mixed mode

Motivation (2)

Applications
Medicine

Example: CT Angiography
Applications

- Engineering
- Computational Science

Volume Data
Continuous scalar field in 3D
\[ s = f(x, y, z); \quad x, y, z \in \mathbb{R} \]
- Discrete volume: voxels
- Sampling
- Reconstruction

Sampling & Reconstruction
- Rectilinear 3D grid; scalar values
- Convolution of samples with reconstruction filter (box, tent, …)

Surface extraction
- Extract Geometry, then render

Direct Volume Rendering
- Render volume without extracting any surfaces (DVR)
- Map scalar values to optical properties (color, opacity)
- Need optical model
- Solve volume rendering integral for viewing rays into the volume
**Optical Models (1)**

- Emission
  - active
  - scattering
- Absorption
  - active
  - scattering

**Optical Models (2)**

- Light interaction with density volume of particles
  - Absorption only
  - Emission only
  - Absorption + emission
  - Scattering + shading/shadowing
  - Multiple scattering

**Ray Casting**

- **Numerical approximation** of volume rendering integral
- Resample volume at equi-spaced intervals along the ray
- **Tri-linear** interpolation

**Integration**

- What must be integrated?
  - physically correct: emission and absorption of light

\[
I(s) = I(s_0) e^{-\tau(s)} + \int_{s_0}^{s} q(s') e^{-\tau(s')} ds'
\]

**Discrete Solution**

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

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

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

\[
I(s_{i+1}) \text{ OVER } I(s_i)
\]
**Alpha Blending**
- **Numerical approximation** of volume rendering integral
- **Back-to-front compositing**
  \[ C_f = C_i + (1 - A_i)C_{f+1} \]
- New composited color is color at current location blended with previous composited color
- **Opacity-weighted colors!**

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

**Alternative Compositing**
**Maximum Intensity Projection (MIP)**
- no **Emission** and **Absorption**
- pixel value is the maximum scalar value along the viewing ray

- **Advantage**: transfer function not required!
- **Drawback**: misleading depth information!

**Maximum Intensity Projection**
- Emission/Absorption
- Maximum Intensity Proj.

**Volume Graphics on GPUs**
**Texture-Based Methods**
- **Numerical approximation** of volume rendering integral
- Resample volume at equi-spaced intervals along the ray
- **Tri-linear interpolation**
**Introduction**

- Data Set
- Proxy Geometry
- Image

**Texture-Based Methods**

**Proxy Geometry**
- 2D textured slices
- 3D textured slices
- 2D slice interpolation

**Volume Renderer Components**
- Setup and texture download
- Fragment shader and blending setup
- Proxy geometry rendering

---

**Volume Rendering Pipeline**

- Slice Decomposition (Proxy Geometry)
- Rendering of textures slices
- Final Image

- Trilinear Hardware Interpolation
- Compositing (Blending)

**Geometry Processing**

- No volumetric hardware primitives!
- Proxy Geometry (Polygonal Slices)

---

**2D Textures**

- Bilinear Interpolation in Hardware
- Axis-aligned slices

- 3 copies of the data set in memory!

**2D Textures: Drawbacks**

- Bilinear instead of trilinear interpolation
2D Textures: Drawbacks

- Inconsistent sampling rate
- Emission/Absorption incorrect
- Supersampling not possible!

Better: 3D Textures

- Trilinear Interpolation in Hardware
- Viewport aligned slices
- Volume is one texture block in memory

Advantages:

- Consistent sampling rate (except for perspective projection)
- Supersampling by increasing the number of slices

Bricking

- Dataset is too large to fit into local video memory
- Subdivide the data into smaller chunks (bricks)

Bottleneck: Bus-Bandwidth

- Bad load balancing for GPU and Bus

Possible Solutions

- Keep the bricks small enough!
- More than one brick must fit into local video memory!
- Texture transfer and rendering must be performed in parallel.
- Higher CPU load for slice decomposition!
- Effective load balancing is still difficult!

Alternative: Use 2D multi-textures instead of 3D textures!
Compositing

- Fragment Operations:
  - Alpha Test
  - Stencil Test
  - Depth Test
  - Alpha Blending

Fractural Operations: Rasterization

Stencil

Alpha

Depth

Alpha Disables

High-Quality Volume Graphics on Consumer PC Hardware
Tutorial 2 Markus Hadwiger

Compositing

- Ray Integration

Ray Integration

$I(s_i) I(s_{i+1})$

Back-to-front Compositing

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

Alpha Blending

$gEnable(GL_BLEND)$;
$gBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA) ;$

Emission/ Absorption

Maximum Intensity Proj.

Compositing

- Maximum Intensity Projections

OpenGL Extension: GL EXT_blend_minmax

Alpha Blending: Maximum Operator

$gEnable(GL_BLEND);$ 
# ifdef GL_EXT_blend_minmax
$gBlendEquationEXT(GL_MAX_EXT);$ 
$gBlendFunc(GL_SRC_COLOR, GL_DST_COLOR);$ 
# endif // GL_EXT_blend_minmax

Opacity Correction

- Sampling distance depends on slice distance: adjust integration
- May depend on viewing direction (2D-textured slices)

Texture Definition

Volume textures

- Palette indices (pre-class.)
- Intensity texture (post-class.)

Transfer function tables

- Texture palette
- Transfer function texture (1D, ...)
- Pre-integration texture (2D)
Per-Volume Setup

**Fragment shader configuration**
- Transfer function application, shading, slice interpolation, ...
- Blending mode configuration
- Final conversion fragment → pixel

**Texture unit configuration (3D)**
- When view-aligned slices are used
- All slices use the same texture

---

**Fragment Shader Config (1)**

NVIDIA GeForce 3/4

```c
// configure texture shaders
GLuint shader_name = gNewFragmentShaderATI( 1 );
gEnable( GL.TEXTURE_SHADER, shader_name );
gEnable( GL.TEXTURE_SHADER_ARB, shader_name );
gEnable( GL.TEXTURE_SHADER_3D, shader_name );
... // enable texture shader
```

---

**Per-Slice Setup**

**Texture unit configuration (2D)**
- When object-aligned slices are used
- Each slice uses different texture

- Slice interpolation needs two adjacent slice textures
- Fractional position between slices for slice interpolation

---

**Generate Fragments**

- **Render proxy geometry**
- Submit slices in back-to-front order

**2D textured slices**
- Align slices with volume
- Specify texcoords for each slice

**3D textured slices**
- Align slices with viewport
- Use automatic texcoord generation

---