A Bandwidth Effective Rendering Scheme for 3D Texture-based Volume Visualization on GPU

Won-Jong Lee, Tack-Don Han

Media System Laboratory (http://msl.yonsei.ac.kr)
Dept. of Computer Science, Yonsei University, Seoul, Korea
Contents

• Introduction
• Background & Related Work
• Proposed Octree-based Bandwidth Effective Volume Rendering
• Experimental Results
• Conclusion
Introduction
Introduction

• Why Volume Rendering?
  – Overcome limitation of polygon-based rendering
  – Volume rendering generate photorealistic and non-photorealistic scene
• Application#1 – Visualization

- material sciences
- geology
- medical visualization
• Application#2 - Entertainment

- atmospheric effects
- explosions, FX game
3D Texture-base Volume Rendering

- **Texture Based Volume-Rendering**
  - Low cost & high quality method

  - X-ray like shading (1996)
  - Shading with OpenGL extensions (1998)
  - Shading with Register Combiner & Multi-Texture Units (1999)
  - Classification with Multi-dimensional T/F (2001)

  - GPU ray-casting (2003)
  - Point-based VR (2004)
3D Texture-based Volume Rendering

• Typical GPU-based volume rendering flow
A Bandwidth Effective Rendering Scheme for 3D Texture-based Volume Visualization on GPU

3D Texture-base Volume Rendering

- Typical GPU-based volume rendering flow

Introduction

3D Texture VR Limitation

Related Work

Proposed Scheme

Experiment

Conclusion

3D Graphics Group

Media System Lab.

Introduction
A Bandwidth Effective Rendering Scheme for 3D Texture-based Volume Visualization on GPU

3D Texture-base Volume Rendering

• Typical GPU-based volume rendering flow

Introduction
3D Texture VR Limitation

Related Work

Proposed Scheme

Experiment

Conclusion

Polygons Slices

CPU

Main Memory

Disk

Gradient Computation

Vertex & Texture Coordinates Loading

Shaders and T/F Creation

3D Texture

Pre-TnL Cache

GPU

Programmable Vertex Shader

Post-TnL Cache

Triangle Setup And Rasterization

Programmable Pixel Shader

Raster Operation

Texture Cache

Pixel Cache

Graphic Memory

Original Volume Data Loading

Classification, 3D Texture Mapping, Lighting

Blending

YONSEI UNIVERSITY

Media System Lab.

Korea Computer Congress 2005
3D Texture-base Volume Rendering

• Typical GPU-based volume rendering flow

![Diagram of typical GPU-based volume rendering flow](image)
3D Texture-base Volume Rendering

- Typical GPU-based volume rendering flow

**Diagram:**
- CPU
- Disk
- Main Memory
- Graphic Memory
- GPU

**Key Components:**
- Original Volume Data Loading
- Gradient Computation Shaders and T/F Creation
- Pre-Tnl Cache
- Volume, Gradients, T/F, Loading
- Post-Tnl Cache
- Triangle Setup And Rasterization
- Texture Cache
- Pixel Cache
- Programable Vertex Shader
- Programable Pixel Shader
- Raster Operation

**Flow:**
- Polygon Slices
- 3D Texture
- PhalImage

**Sections:**
- Introduction
- Related Work
- 3D Texture VR Limitation
- Proposed Scheme
- Experiment
- Conclusion

**Keywords:**
- A Bandwidth Effective Rendering Scheme
- 3D Texture-based Volume Visualization on GPU
- Media System Lab.
- Korea Computer Congress 2005
3D Texture-base Volume Rendering

- Typical GPU-based volume rendering flow

**Introduction**
- 3D Texture VR
  - Limitation

**Related Work**
- Proposed Scheme
- Experiment
- Conclusion

**3D Texture-based Volume Rendering**

- Gradient Computation
- Shaders and T/F Creation
- Disk

**CPU**
- Main Memory
- Original Volume Data Loading
- Coordinates Transform
- Classification, 3D Texture Mapping, Lighting

**GPU**
- Programmable Vertex Shader
- Post-TnL Cache
- Triangle Setup And Rasterization
- Programmable Pixel Shader
- Texture Cache
- Pixel Cache
- Rasterization
- Blending

**Main Memory**
- Graphic Memory
- Volume, Gradients, T/F, Loading
- Vertex & Texture Coordinates Loading

**Disk**
- Polygon Slices
- 3D Texture
- Final Image

A Bandwidth Effective Rendering Scheme for 3D Texture-based Volume Visualization on GPU

Media System Lab.

Yonsei University

Korea Computer Congress 2005
• The size of volume data sets can be processed is still limited
  – usually, 64-256MB graphic memory, middle sized\((512^3)\) 8|16bit volume data, 256MB|512MB
Limitation of 3D Texture Mapping

- Bottleneck in graphic memory bus
  - lots of texture & pixel traffic (interpolation, $\alpha$-blending)
  - Lower localities in texture mapping & blending operation

Diagram:

- Disk
  - Original Volume Data Loading
  - Texture Cache

- Graphic Memory
  - Pre-TnL Cache
  - Post-TnL Cache
  - Triangle Setup And Rasterization
  - Programmable Vertex Shader
  - Programmable Pixel Shader
  - Raster Operation

- GPU

Legend:

1. request
2,3,4 Hit!
5,6,7,8 Miss!!

Threshing!
A Bandwidth Effective Rendering Scheme for 3D Texture-based Volume Visualization on GPU

Limitation of 3D Texture Mapping

- Bottleneck in graphic memory bus
  - lots of texture & pixel traffic (interpolation, \(\alpha\)-blending)
  - Lower localities in texture mapping & blending operation

Proposed Scheme

Conclusion

Experiment

Related Work

Introduction

3D Texture VR Limitation

Classification, 3D Texture Mapping, Lighting

Blending

Pre-TnL Cache

Post-TnL Cache

Triangle Setup And Rasterization

Programmable Pixel Shader

Raster Operation

Programmable Vertex Shader

Texture Cache

Pixel Cache

Graphic Memory

Disk

Main Memory

CPU

Graphic Memory

Blending Unit

Pixel Cache

Vertex & Texture Coordinates Loading

Data Loading

Hit!

Miss!!
Limitation of 3D Texture Mapping

- **Brute-forced approach**
  - classical optimization technique cannot be utilized (empty-space skipping, early-ray termination)

---

**Proposed Scheme**

1. **Volume, Gradients, T/F Loading**
2. **Pre-TnL Cache**
3. **Programmable Vertex Shader**
4. **Post-TnL Cache**
5. **Triangle Setup and Rasterization**
6. **Programmable Pixel Shader**
7. **Raster Operation**

---

**Related Work**

- Gradient Computation
- Shaders and T/F Creation

---

**Conclusion**

- Experiment
- Proposed Scheme
- Related Work
- Introduction
Contributions of This Research

- Propose an sub-division based rendering algorithm for GPU bandwidth-effectiveness
- Enabling empty space skipping for optimized rendering
- Utilizing modern GPU features for low-cost texture coordinate computation
- Build-up for effective H/W simulation environments for GPU based rendering
Related Work
Related Work

Related Work


partitioned by the growing boxes
converted into a BSP tree
Rendering with mixed boxes and texture
Related Work

Proposed Method
Proposed Octree-based V/R

Octree based Empty Space Skipping

- **standard method**
  - unable to skip empty space

- **proposed method**
  - enabling empty space skipping

Sub-division
Proposed Octree-based V/R

Cache Efficiency

standard method

proposed method

HIT for each sub-volume

only cold MISS on switching sub-volume
Proposed Octree-based V/R

- **Rendering Octree in GPU**
  - **Utilizing the property of regular sized sub-volumes**
  - **Easy to detect visibility order of each sub-volume**

- **Incremental texture coordinates computations in GPU**
Experimental Results
Benchmarks

- Lobster (255x255x56)
- Engine (255x255x110)
- Foot (255x255x255)
- CTdata (255x255x255)
A Bandwidth Effective Rendering Scheme for 3D Texture-based Volume Visualization on GPU

• CPU-based software simulation
  – For evaluating cache efficiency, memory bandwidth

  **Cg vertex & fragment shader**
  compile with Cg compiler (cgc) → GL function & shader assembly call

  **NV_vertex_program**
  **NV_fragment_program**

  **C++ OpenGL application**

  scene

  **Mesa 6.0**
  (fully compatible with OpenGL 1.5)

  **Dinero III**

  cache simulation results

  – Make comparison with standard and proposed method
  – Under perfectly same cache configuration
    – fully associative cache, 32Bytes block, 1~128KBytes cache
    – 512x512 screen size, 10 frame rendered
A Bandwidth Effective Rendering Scheme for 3D Texture-based Volume Visualization on GPU

- **GPU-based hardware experiment**
  - For evaluating rendering speedup

  - Compile with Cg compiler (cgc) -
  - GL function call
  - NV_vertex_program
  - NV_fragment_program
  - C++ OpenGL application

  - Scene & frame rates result
  - Load Shader assembly to GPU

- **Conclusion**
  - Make comparison with standard and proposed method
  - Under perfectly same condition
    - Pentium IV 2.80GHz PC with 512MB RDRAM & AGP8x bus
    - Nvidia GeForce FX 5900 with 256MB graphic memory
Trade-Off Analysis

Average # of Vertices per Frame

Non-Empty Region Ratio

A Bandwidth Effective Rendering Scheme for 3D Texture-based Volume Visualization on GPU

3D Graphics Group
YONSEI UNIVERSITY
Media System Lab.

Korea Computer Congress 2005
A Bandwidth Effective Rendering Scheme for 3D Texture-based Volume Visualization on GPU

Miss Rates – Texture Cache

<table>
<thead>
<tr>
<th>Texture Cache Size (KBytes)</th>
<th>Lobster (255x255x56)</th>
<th>Engine (255x255x110)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSD</td>
<td>OSD(64^3)</td>
<td>OSD(32^3)</td>
</tr>
<tr>
<td>NSD</td>
<td>OSD(64^3)</td>
<td>OSD(32^3)</td>
</tr>
</tbody>
</table>

Foot (255x255x255) | CTdata (255x255x255)

NSD : Non-Subdivided
OSD : Octree Subdivided
(number) : size of sub-volume
Miss Rates – Pixel Cache (Color)

![Graph showing miss rates for different cache sizes and volumes.](image)

- NSD: Non-Subdivided
- OSD: Octree Subdivided

- Lobster (255x255x56)
- Engine (255x255x110)
- Foot (255x255x255)
- CTdata (255x255x255)
A Bandwidth Effective Rendering Scheme
for 3D Texture-based Volume Visualization on GPU

Miss Rates – Pixel Cache (Depth)

Introduction
Related Work
Proposed Scheme
Experiment
Environment
Analysis
Texture Cache
Pixel Cache
Total Bandwidth
Frame Rates
Conclusion

NSD: Non-Subdivided
OSD: Octree Subdivided
(number): size of sub-volume
## Total Bandwidth Comparison #1 for 10 frames

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Rendering Method</th>
<th>Number of Vertices</th>
<th>CPU-GPU Bandwidth by Vertices(Bytes)</th>
<th>Cache Size KBytes</th>
<th>Read from TexMem by Texture Cache (Bytes)</th>
<th>Read/Write from Primary Buffer (Bytes)</th>
<th>Total Internal GPU Bandwidth (GBytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NSD</td>
<td>Non Sub-Divided standard method</td>
<td>11,169</td>
<td>357,408</td>
<td>16</td>
<td>2,600,184,288</td>
<td>105,712,448</td>
<td>195,004,928</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>2,012,665,728</td>
<td>104,112,000</td>
<td>194,826,464</td>
<td>2.15</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>2,006,103,296</td>
<td>97,560,896</td>
<td>192,344,928</td>
<td>2.14</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>1,986,313,536</td>
<td>51,613,760</td>
<td>184,948,760</td>
<td>2.07</td>
<td></td>
</tr>
<tr>
<td>Lobster</td>
<td>Octree-based Sub-Divided method</td>
<td>83,220</td>
<td>1,331,520</td>
<td>16</td>
<td>254,303,584</td>
<td>4,777,888</td>
<td>7,078,400</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>167,828,960</td>
<td>4,270,688</td>
<td>6,572,640</td>
<td>0.17</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>125,515,840</td>
<td>2,630,048</td>
<td>5,170,336</td>
<td>0.12</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>124,341,216</td>
<td>757,696</td>
<td>3,187,248</td>
<td>0.12</td>
<td></td>
</tr>
<tr>
<td>OSD(16³)</td>
<td>Non Sub-Divided standard method</td>
<td>169,569</td>
<td>2,713,104</td>
<td>16</td>
<td>94,798,688</td>
<td>4,246,336</td>
<td>6,894,976</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>90,461,824</td>
<td>3,614,336</td>
<td>6,354,464</td>
<td>0.09</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>78,578,944</td>
<td>2,200,288</td>
<td>4,848,480</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>70,120,800</td>
<td>693,632</td>
<td>2,562,030</td>
<td>0.07</td>
<td></td>
</tr>
<tr>
<td>Engine</td>
<td>Octree-based Sub-Divided method</td>
<td>203,889</td>
<td>3,262,224</td>
<td>16</td>
<td>563,959,232</td>
<td>6,307,072</td>
<td>12,603,032</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>265,399,392</td>
<td>5,759,936</td>
<td>10,045,024</td>
<td>0.26</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>264,792,368</td>
<td>3,987,552</td>
<td>8,949,856</td>
<td>0.26</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>261,793,120</td>
<td>1,672,064</td>
<td>5,464,384</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>Engine</td>
<td>Octree-based Sub-Divided method</td>
<td>554,542</td>
<td>8,847,552</td>
<td>16</td>
<td>239,474,752</td>
<td>5,806,688</td>
<td>13,193,216</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
<td>222,423,544</td>
<td>5,183,968</td>
<td>9,941,792</td>
<td>0.22</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>64</td>
<td>201,845,760</td>
<td>3,042,752</td>
<td>8,708,096</td>
<td>0.20</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>128</td>
<td>186,871,328</td>
<td>809,568</td>
<td>4,365,376</td>
<td>0.18</td>
<td></td>
</tr>
</tbody>
</table>

### Conclusion
- **At best case,** 29x reduced!
- **Even worst case,** 8Mbytes increased!
### Total Bandwidth Comparison #2 for 10 frames

<table>
<thead>
<tr>
<th>Datasets</th>
<th>Rendering Method</th>
<th>Number of Vertices</th>
<th>CPU-GPU Bandwidth by Vertices (Bytes)</th>
<th>Cache Size (KBytes)</th>
<th>Read from TexMem by Texture Cache (Bytes)</th>
<th>Read/Write from Color Cache</th>
<th>Read/Write from Depth Cache</th>
<th>Total Internal GPU Bandwidth (GBytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Foot</td>
<td>NSD</td>
<td>11,169</td>
<td>357,408</td>
<td>16</td>
<td>2,660,184,288</td>
<td>105,712,448</td>
<td>155,004,928</td>
<td>2.76</td>
</tr>
<tr>
<td></td>
<td>OS(32^3)</td>
<td>214,982</td>
<td>3,439,760</td>
<td>16</td>
<td>663,009,632</td>
<td>7,019,488</td>
<td>3,684,112</td>
<td>0.63</td>
</tr>
<tr>
<td></td>
<td>OS(16^3)</td>
<td>487,596</td>
<td>7,801,536</td>
<td>16</td>
<td>207,982,752</td>
<td>5,876,048</td>
<td>13,278,112</td>
<td>0.21</td>
</tr>
<tr>
<td></td>
<td>NSD</td>
<td>11,169</td>
<td>357,408</td>
<td>16</td>
<td>2,660,184,288</td>
<td>105,712,448</td>
<td>155,004,928</td>
<td>2.76</td>
</tr>
<tr>
<td>CTdata</td>
<td>OS(32^3)</td>
<td>217,759</td>
<td>3,484,144</td>
<td>16</td>
<td>673,297,024</td>
<td>6,838,304</td>
<td>13,083,168</td>
<td>0.68</td>
</tr>
<tr>
<td></td>
<td>OS(16^3)</td>
<td>457,632</td>
<td>7,322,112</td>
<td>16</td>
<td>252,801,216</td>
<td>6,173,088</td>
<td>11,939,872</td>
<td>0.25</td>
</tr>
</tbody>
</table>

#### Conclusion

- Proposed Scheme
- Pixel Cache
- Total Cache
- Frame Bandwidth

---

**A Bandwidth Effective Rendering Scheme for 3D Texture-based Volume Visualization on GPU**

---

**3D Graphics Group Media System Lab.**

**Korea Computer Congress 2005**
## Rendering Performance

<table>
<thead>
<tr>
<th>Data Set</th>
<th>Size</th>
<th>NSD</th>
<th>OSD(32^3)</th>
<th>Speedup</th>
<th>OSD(16^3)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lobster</td>
<td>256x256x56</td>
<td>4.4</td>
<td>25.7</td>
<td>5.9</td>
<td>29.1</td>
<td>6.7</td>
</tr>
<tr>
<td>Engine</td>
<td>256x256x110</td>
<td>4.6</td>
<td>11.1</td>
<td>2.4</td>
<td>12.2</td>
<td>2.6</td>
</tr>
<tr>
<td>Foot</td>
<td>256x256x256</td>
<td>4.5</td>
<td>10.3</td>
<td>2.3</td>
<td>11.7</td>
<td>2.6</td>
</tr>
<tr>
<td>CTdata</td>
<td>256x256x256</td>
<td>4.7</td>
<td>10.4</td>
<td>2.2</td>
<td>12.5</td>
<td>2.7</td>
</tr>
<tr>
<td><strong>Average</strong></td>
<td><strong>256x256x256</strong></td>
<td><strong>4.7</strong></td>
<td><strong>10.4</strong></td>
<td><strong>2.2</strong></td>
<td><strong>12.5</strong></td>
<td><strong>2.7</strong></td>
</tr>
</tbody>
</table>

NSD: Non Sub-Divided standard method, OSD: Octree-based Sub-Divided method
• Octree based Bandwidth-Effective Volume Rendering
  – Maximize the temporal & spacial locality of memory access
  – Enable the empty space skipping
  \[\Rightarrow 2\sim29x \text{ bandwidth reduction} \]
  \[\Rightarrow 2\sim6x \text{ faster rendering} \]
• Application – Volumetric Effects