GPU-Based Volume Rendering of Unstructured Grids

Module 4: Hardware-Assisted Visibility Sorting

Steven P. Callahan
University of Utah
Overview

• Recent advances in GPU programmability
  • $k$-Buffer: A fragment stream sorter
  • Hardware-Assisted Visibility Sorting
  • Dynamic Level-of-Detail
GPU: Recent Features

Render to texture

• Why?
  – Better performance

• Applications
  – Dynamic textures
  – Multi-pass algorithms
  – Image processing
Multiple Render Targets (MRTs)

- Write into multiple textures simultaneously
OpenGL Pixel Buffers (PBuffers)

- Enables off-screen rendering
- Contains its own depth, stencil, and aux buffers
- MRT support by rendering into Front and up to 3 AUX buffers
Disadvantages of PBuffers

- Each has its own OpenGL context
- Switching between PBuffers is expensive
- Cannot share buffers between PBuffers
- Pixel format selection
- Extensions only available on Windows
OpenGL Framebuffer Objects (FBOs)

- A collection of attachable textures
  - Color, depth, stencil, etc.
- Attached textures are source and destination for fragment shaders
- MRTs are available using multiple color attachments
Advantages of FBOs

- A single context
- Pixel format determined by texture format
- Share buffers between FBOs
- Easier to use than PBuffers
- Works on multiple platforms
GPU: Recent Features

Code

• PBuffers
  – RenderTexture 2.0 (Mark Harris)
• FBOs
  – Framebuffer Object Class (Aaron Lefohn)

www.gpgpu.org/developer
Overview

• Recent advances in GPU programmability
• $k$-Buffer: A fragment stream sorter
• Hardware-Assisted Visibility Sorting
• Dynamic Level-of-Detail
Sorting

Application

Object-Space Sorting

Rasterization

Image Space

Display

GPU-Based Volume Rendering of Unstructured Grids
GPU-Based Volume Rendering of Unstructured Grids

Image-Space Sorting

A-Buffer

[Image of A-Buffer diagram]

[Carpenter]
Sorting

Application
Object-Space Sorting
Rasterization
Image Space
Display

GPU-Based Volume Rendering of Unstructured Grids
$k$-Buffer

- Fixed-size A-Buffer
- As a new pixel is inserted, another is removed
- Can efficiently sort a $k$-Nearly Sorted Sequence ($k$-NSS)
GPU-Based Volume Rendering of Unstructured Grids

**k-Buffer**

<table>
<thead>
<tr>
<th>input</th>
<th>1</th>
<th>3</th>
<th>2</th>
<th>4</th>
<th>6</th>
<th>5</th>
</tr>
</thead>
</table>

**k-buffer**

<table>
<thead>
<tr>
<th>output</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>

GPU-Based Volume Rendering of Unstructured Grids
GPU-Based Volume Rendering of Unstructured Grids
GPU-Based Volume Rendering of Unstructured Grids

*k*-Buffer

input

1 3 2 4 6 5

*k*-buffer

3 2

output

1
GPU-Based Volume Rendering of Unstructured Grids

**k-Buffer**

**input**

1 3 2 4 6 5

**k-buffer**

3 4

**output**

1 2
GPU-Based Volume Rendering of Unstructured Grids

k-Buffer

input: 1 3 2 4 6 5

k-buffer: 4 6

output: 1 2 3
GPU-Based Volume Rendering of Unstructured Grids

SIBGRAPI 2005

k-Buffer

input

1 3 2 4 6 5

k-buffer

6 5

output

1 2 3 4
GPU-Based Volume Rendering of Unstructured Grids
GPU-Based Volume Rendering of Unstructured Grids

$k$-Buffer

input: 1 3 2 4 6 5

$k$-buffer

output: 1 2 3 4 5 6
Overview

- Recent advances in GPU programmability
- $k$-Buffer: A fragment stream sorter
- Hardware-Assisted Visibility Sorting
- Dynamic Level-of-Detail
Object-Space Sorting

- Performed on CPU
- Sort faces by center
- Least Significant Digit Radix Sort
- Handles floating-point numbers

```cpp
inline unsigned int float2fint (unsigned int f)
{
    return f ^ (((-f) >> 31) | 0x80000000);
}
```

- Results: 15 million faces/sec
Image-Space Sorting

- Performed on GPU
- Uses $k$-Buffer as a fragment stream sorter
- Keeps $k$ entries per pixel, each entry contains a fragment’s scalar value and distance from the viewpoint ($v,d$)
- An incoming fragment replaces the entry that is closest to the eye (front-to-back compositing)
Hardware-Assisted Visibility Sorting

Sort in image-space and object-space
1. Approximate sort in object-space
2. Complete sort in image-space

- Partially Sort Faces By Centroid
- Rasterize Faces
- Completely Sort Fragments With $k$-buffer
- Composite Final Image
- Use MRTs
- Attach 4 32-bit floating-point RGBA textures to FBO as color attachments

<table>
<thead>
<tr>
<th>Texture 1</th>
<th>Texture 2</th>
<th>Texture 3</th>
<th>Texture 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>( r_{comp} )</td>
<td>( v_1 )</td>
<td>( v_3 )</td>
<td>( v_5 )</td>
</tr>
<tr>
<td>( g_{comp} )</td>
<td>( d_1 )</td>
<td>( d_3 )</td>
<td>( d_5 )</td>
</tr>
<tr>
<td>( b_{comp} )</td>
<td>( v_2 )</td>
<td>( v_4 )</td>
<td>( v_6 )</td>
</tr>
<tr>
<td>( a_{comp} )</td>
<td>( d_2 )</td>
<td>( d_4 )</td>
<td>( d_6 )</td>
</tr>
</tbody>
</table>
$k$-Buffer in Hardware

Incoming

\[ f_{\text{new}} \]

1

1

Lookup Table

Framebuffer

Texture

\[ f_1 \]

3

2

Texture

\[ f_2 \]

\[ f_3 \]

4

GPU-Based Volume Rendering of Unstructured Grids
Details

• Fix incorrect texture coordinates caused by perspective-correct interpolation

Perspective Correct

Projecting vertices to find tex coords

Projecting tex coords in shader
• Simultaneously reading and writing to a texture is undefined when fragments are rasterized in parallel
Details

- Initialization and Termination
- Non-convex objects
Experiments

Environment

- 3.2 GHz Pentium 4
- 2048 MB RAM
- Windows XP
- ATI Radeon 9800 Pro

Results

- $k$-Buffer analysis
- Performance results
# k-Buffer Analysis

## Accuracy Analysis
- $k$ depth required to render datasets
- Max values from 14 fixed viewpoints

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Max A</th>
<th>Max $k$</th>
<th>$k &gt; 2$</th>
<th>$k &gt; 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spx2</td>
<td>476</td>
<td>22</td>
<td>10,262</td>
<td>512</td>
</tr>
<tr>
<td>Torso</td>
<td>649</td>
<td>15</td>
<td>43,317</td>
<td>1,683</td>
</tr>
<tr>
<td>Fighter</td>
<td>904</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
$k$-Buffer Analysis

Distribution Analysis

- Shows the actual pixels that require large $k$ depths to render correctly

$k \leq 2$: green
$2 < k \leq 6$: yellow
$k > 6$: red
### Results

#### Performance
- $512^2$ viewport with a $128^3$ pre-integrated lookup table

#### GPU Sorting:

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Cells</th>
<th>$k = 2$ fps</th>
<th>$k = 2$ tets/s</th>
<th>$k = 6$ fps</th>
<th>$k = 6$ tets/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spx2</td>
<td>0.8 M</td>
<td>2.07</td>
<td>1712 K</td>
<td>1.7</td>
<td>1407 K</td>
</tr>
<tr>
<td>Torso</td>
<td>1.1 M</td>
<td>3.13</td>
<td>3390 K</td>
<td>1.86</td>
<td>1977 K</td>
</tr>
<tr>
<td>Fighter</td>
<td>1.4 M</td>
<td>2.41</td>
<td>3387 K</td>
<td>1.56</td>
<td>2190 K</td>
</tr>
</tbody>
</table>
### Results

**Performance**
- CPU sorting + GPU sorting and compositing
- Pipeline optimization = max(CPU, GPU)

**Total Time:**

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Cells</th>
<th>CPU</th>
<th>GPU</th>
<th>Total</th>
<th>Tets/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spx2</td>
<td>0.8 M</td>
<td>160 ms</td>
<td>368 ms</td>
<td>528 ms</td>
<td>1568 K</td>
</tr>
<tr>
<td>Torso</td>
<td>1.1 M</td>
<td>210 ms</td>
<td>390 ms</td>
<td>600 ms</td>
<td>1805 K</td>
</tr>
<tr>
<td>Fighter</td>
<td>1.4 M</td>
<td>268 ms</td>
<td>505 ms</td>
<td>773 ms</td>
<td>1816 K</td>
</tr>
</tbody>
</table>
Movie

Spx2
828K Tetrahedra
GPU-Based Volume Rendering of Unstructured Grids

Movie

Fighter
1.40M Tetrahedra
Conclusion

- Introduced the $k$-buffer and an efficient GPU implementation
- Fastest volume renderer for unstructured data
- Handles arbitrary non-convex meshes
- Requires minimal pre-processing of data
- Maximum data size is bounded by main memory
- Code is short and simple
- Can easily be extended
Overview

- Recent advances in GPU programmability
- $k$-Buffer: A fragment stream sorter
- Hardware-Assisted Visibility Sorting
- Dynamic Level-of-Detail
Dynamic Level-of-Detail

100%
25%
5%
LOD Background

Geometric Approach

[Cignoni et al. 04]
LOD Background

Texture Approach

[Leven et al. 02]
Definitions

Given a scalar field

\[ f : D \subseteq \mathbb{R}^3 \rightarrow \mathbb{R} \]

An approximation can be made such that

\[ |\bar{f} - f| \leq \varepsilon \quad \text{and} \quad |\bar{D}| < |D| \]

A ray passing through the domain forms a continuous function

\[ g(t) = f(r_0 + tr_d) \]
Domain-Based Simplification

GPU-Based Volume Rendering of Unstructured Grids
Sample-Based Simplification

GPU-Based Volume Rendering of Unstructured Grids
Domain vs. Sample

Domain-based simplification computes the exact volume integral over the approximate geometry.

Sample-based simplification computes an approximate volume integral over the original geometry.
Dynamic Level-of-Detail

Face sub-sampling
- Draw a subset of the original faces
- Base case: boundary faces
- Sample the internal faces

\[ |I| = \frac{|I_{prev}| \times \text{TargetTime}}{\text{RenderTime}} \]
Dynamic Level-of-Detail

1. Partially Sort Faces by Centroid
2. Rasterize Faces
3. Completely Sort Fragments With k-buffer
4. Composite Final Image

- Determine Subset of Faces
- Sample Faces
Sampling Strategies

Topology: target continuity
Sampling Strategies

View: target screen-space coverage
Sampling Strategies

Field: target histogram
Area: target faces that cause greater error
Results: Time

![Graph showing the relationship between sampling percentage and render time. The x-axis represents the sampling percentage ranging from 0 to 100, while the y-axis represents the render time ranging from 0 to 0.8 seconds. The graph includes three lines, each representing a different dataset or condition, with the black line showing the highest render time, followed by the red and blue lines.]
## Results: Preprocessing

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Tets</th>
<th>Topology</th>
<th>View</th>
<th>Field</th>
<th>Area</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spx2</td>
<td>0.8 M</td>
<td>17.8</td>
<td>5.3</td>
<td>4.5</td>
<td>13.9</td>
</tr>
<tr>
<td>Torso</td>
<td>1.0 M</td>
<td>87.2</td>
<td>11.6</td>
<td>10.5</td>
<td>11.2</td>
</tr>
<tr>
<td>Fighter</td>
<td>1.4 M</td>
<td>75.6</td>
<td>15.3</td>
<td>13.9</td>
<td>15.3</td>
</tr>
</tbody>
</table>
Results: Comparison

![Graphs showing RMSE comparison for Fighter and Spx2](image)

**Fighter**
- Field: blue line
- Topology: red line
- View: green line
- Area: orange line

**Spx2**
- Field: blue line
- Topology: red line
- View: green line
- Area: orange line

GPU-Based Volume Rendering of Unstructured Grids
Results: Comparison

- Full Quality
- Topology
- View
- Field
- Area

SIBGRAPI 2005

GPU-Based Volume Rendering of Unstructured Grids
Results

100%
1.3 fps

15%
4.5 fps

5%
10.0 fps

GPU-Based Volume Rendering of Unstructured Grids
Comparison

Full Quality
100% @ 20 fps

Sample
50% @ 30 fps

Domain
50% @ 23 fps
GPU-Based Volume Rendering of Unstructured Grids

Comparison

- Full Quality: 100% @ 20 fps
- Sample: 50% @ 30 fps
- Domain: 25% @ 30 fps
Movie

Torso
1.08M Tetrahedra
Movie

Fighter
1.40M Tetrahedra
Conclusion

• New sampling approach which simplifies LOD
• Well-suited for a GPU implementation
• Allows dynamic changes to LOD without keeping hierarchical information
Open Research

- Develop techniques to refine datasets to respect a given $k$
- Parallel techniques
- Other $k$-Buffer uses: Isosurfaces, rendering effects, etc.
- Better sampling strategies
- Handle even larger data
Acknowledgements

• Cláudio Silva, João Comba, Milan Ikits, Peter Shirley