GPU-Based Volume Rendering of Unstructured Grids

Module 3: Isosurface Techniques

João L. D. Comba

UFRGS
Isosurfaces

Isosurface: surface with the same scalar field

[Lorensen 95] Marching Through the Visible Man
Marching Cubes

- Isosurface extraction from voxels [Lorensen 87]
  - Assumes Linear Interpolation between data
  - Corners are marked as inside/outside surface

- Mesh extraction
  - Connect surface intersections

- Render Mesh using traditional techniques
Marching Cubes

- Create table with all configurations
  - Number of triangles
  - Connectivity
- For each voxel:
  - Identify configuration using table
  - Compute triangles and connect with adjacent voxels triangles
Marching Cubes

- Create table with all configurations
  - Number of triangles
  - Connectivity
- For each voxel:
  - Identify configuration using table
  - Compute triangles and connect with adjacent voxels triangles
Marching Tetrahedra

• [Doi an Koide 91]
HW Accelerated Isosurface based on Cell Projection

• [Röttger et al] Vis 00
HW Accelerate Marching Cells Algorithm

- Computing Isosurfaces using XOR
  [Westerman 98]

Back faces
Front faces
Projected Tetra with flat-shaded isosurfaces
Smoothly Shaded and Multiple Isosurfaces
Results

Multiple Isosurfaces

Smoothly-Shaded Isosurfaces
Isosurface Computation using Vertex Programs

Isosurface Computation using Vertex Programs

- Possible Isosurfaces by Linear Interpolation

<table>
<thead>
<tr>
<th>isosurface iso-value</th>
<th>geometric primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f = 0$</td>
<td>empty</td>
</tr>
<tr>
<td>$f = 1.8$</td>
<td>triangle</td>
</tr>
<tr>
<td>$f = 2.5$</td>
<td>quad</td>
</tr>
<tr>
<td>$f = 3.7$</td>
<td>triangle</td>
</tr>
</tbody>
</table>
Isosurface Computation using Vertex Programs

• Rendering Step:

```c
set_global_parameters();
set_isovalue();
glBegin(GL_QUADS);  // Start drawing quads
for i=0 to num_tets do:
    set_tet_parameters(i);  // Store vertices in registers
    glVertex2b(0,0);       // Run program four times
    glVertex2b(1,1);       // with v[OPOS].x set
    glVertex2b(2,2);       // successively to 0,1,2,3.
    glVertex2b(3,3);
glEnd();              // Stop drawing quads
```
Isosurface Computation using Vertex Programs

- **Vertex Program Registers:**

<table>
<thead>
<tr>
<th>Const. Reg.</th>
<th>Edge Selection</th>
<th>V0</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>Vertex Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>E0 start</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>V0</td>
</tr>
<tr>
<td>71</td>
<td>E0 end</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>V1</td>
</tr>
<tr>
<td>72</td>
<td>E1 start</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>V2</td>
</tr>
<tr>
<td>73</td>
<td>E1 end</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>V3</td>
</tr>
<tr>
<td>74</td>
<td>E2 start</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>E2 end</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>E3 start</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>E3 end</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>E4 start</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>E4 end</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>E5 start</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>E5 end</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Isosurface Intersection Table:**

<table>
<thead>
<tr>
<th>Const. Reg.</th>
<th>Edge</th>
<th>Interp. case</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>41</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>43</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>44</td>
<td>100</td>
<td>100</td>
</tr>
<tr>
<td>45</td>
<td>101</td>
<td>101</td>
</tr>
<tr>
<td>46</td>
<td>110</td>
<td>110</td>
</tr>
<tr>
<td>47</td>
<td>111</td>
<td>111</td>
</tr>
<tr>
<td>48</td>
<td>1000</td>
<td>1000</td>
</tr>
<tr>
<td>49</td>
<td>1001</td>
<td>1001</td>
</tr>
<tr>
<td>50</td>
<td>1010</td>
<td>1010</td>
</tr>
<tr>
<td>51</td>
<td>1011</td>
<td>1011</td>
</tr>
<tr>
<td>52</td>
<td>1100</td>
<td>1100</td>
</tr>
<tr>
<td>53</td>
<td>1101</td>
<td>1101</td>
</tr>
<tr>
<td>54</td>
<td>1110</td>
<td>1110</td>
</tr>
<tr>
<td>55</td>
<td>1111</td>
<td>1111</td>
</tr>
</tbody>
</table>
Isosurface Computation using Vertex Programs

- **Vertex Program Registers:**

<table>
<thead>
<tr>
<th>Const. Reg.</th>
<th>Edge Selection</th>
<th>V0</th>
<th>V1</th>
<th>V2</th>
<th>V3</th>
<th>Vertex Selection</th>
</tr>
</thead>
<tbody>
<tr>
<td>70</td>
<td>E0 start</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>V0</td>
</tr>
<tr>
<td>71</td>
<td>E0 end</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>V1</td>
</tr>
<tr>
<td>72</td>
<td>E1 start</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>V2</td>
</tr>
<tr>
<td>73</td>
<td>E1 end</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>V3</td>
</tr>
<tr>
<td>74</td>
<td>E2 start</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>E2 end</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>E3 start</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>77</td>
<td>E3 end</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>E4 start</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>E4 end</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>E5 start</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>81</td>
<td>E5 end</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

**Isosurface Intersection Table:**

<table>
<thead>
<tr>
<th>Const. Reg.</th>
<th>Edge</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>Interp. case</th>
</tr>
</thead>
<tbody>
<tr>
<td>40</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>0</td>
</tr>
<tr>
<td>41</td>
<td>74</td>
<td>80</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>1</td>
</tr>
<tr>
<td>42</td>
<td>80</td>
<td>76</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>10</td>
</tr>
<tr>
<td>43</td>
<td>74</td>
<td>80</td>
<td>76</td>
<td>78</td>
<td>78</td>
<td>11</td>
</tr>
<tr>
<td>44</td>
<td>70</td>
<td>76</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>100</td>
</tr>
<tr>
<td>45</td>
<td>70</td>
<td>76</td>
<td>80</td>
<td>74</td>
<td>74</td>
<td>101</td>
</tr>
<tr>
<td>46</td>
<td>70</td>
<td>80</td>
<td>74</td>
<td>74</td>
<td>74</td>
<td>110</td>
</tr>
<tr>
<td>47</td>
<td>70</td>
<td>80</td>
<td>74</td>
<td>74</td>
<td>74</td>
<td>111</td>
</tr>
<tr>
<td>48</td>
<td>80</td>
<td>70</td>
<td>74</td>
<td>74</td>
<td>74</td>
<td>1000</td>
</tr>
<tr>
<td>49</td>
<td>70</td>
<td>78</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>1001</td>
</tr>
<tr>
<td>50</td>
<td>70</td>
<td>74</td>
<td>80</td>
<td>76</td>
<td>76</td>
<td>1010</td>
</tr>
<tr>
<td>51</td>
<td>76</td>
<td>70</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>1011</td>
</tr>
<tr>
<td>52</td>
<td>74</td>
<td>78</td>
<td>76</td>
<td>80</td>
<td>80</td>
<td>1100</td>
</tr>
<tr>
<td>53</td>
<td>76</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>80</td>
<td>1101</td>
</tr>
<tr>
<td>54</td>
<td>80</td>
<td>74</td>
<td>78</td>
<td>78</td>
<td>78</td>
<td>1110</td>
</tr>
<tr>
<td>55</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>70</td>
<td>1111</td>
</tr>
</tbody>
</table>
Results
Results
Results

• View-dependent refinement
Isosurface Computation using Fragment Programs

Isosurface Computation using Fragment Programs

- System Overview

![Diagram showing isosurface computation](https://example.com/diagram.png)

- Tetrahedron Index $\tau$
- Isovalue $c$
- Edge Index $\pi$
- Scalars
  - Texture: 32-bit RGBA 2D
- Bit Mask $\beta$
- Scalars
- Edge Table: 4x8 8-bit RGBA 2D
- Local Vertex Indices: 8-bit RGBA 2D
- Index Texture: 8-bit RGBA 2D
- Vertex Coordinates Texture: 32-bit RGBA 3D
- lerp

GPU-Based Volume Rendering of Unstructured Grids
Isosurface Computation using Fragment Programs

- Tetrahedron Classification

![Diagram showing tetrahedron classification with isovalue and bitmask]

- Isovalue = 1.5
  - $1 < 1.5$ → $1 R$
  - $3 < 1.5$ → $0 G$
  - $2 < 1.5$ → $0 B$
  - $1 < 1.5$ → $1 A$
Isosurface Computation using Fragment Programs

- Edge Table

<table>
<thead>
<tr>
<th>$\beta^T$</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0,0,0,0)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>(0,0,0,1)</td>
<td>3</td>
<td>0</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>(0,0,1,0)</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>(0,1,0,0)</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>(1,0,0,0)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>(0,0,1,1)</td>
<td>0</td>
<td>2</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>(0,1,0,1)</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>(0,1,1,0)</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

Intersected Edges
- 0 — 1
- 0 — 3
- 2 — 3
- 2 — 1
Isosurface Computation using Fragment Programs
Implicit Occluders

• [Pesco et al] VolVis 2004
Motivation:
• Exploration of large data sets using isosurfaces

Challenges:
• Isosurfaces can be larger than the original volume
• Often too large to keep in memory

Goal:
• Speed up the computation and rendering of opaque isosurfaces by performing visibility computations

PPM Data Set: 1536 x 1536 x 512
Isovalue = 127
Total number of faces = 89 Million
• **f** is a continuous scalar field defined on a convex domain **D**.
• region **A**: \( f(x) \) assume only positive value.
• region **B**: \( f(x) \) assume only negative value.
• a viewpoint and central projection \( A^* \) of region **A** onto the boundary of **B**
• Any segment $r$ connecting the current viewpoint with $A^*$ must also intersect the isosurface $f^{-1}(0)$.
• Therefore, region $R$ behind $A^*$ is completely occluded and can be used as an occluder in place of $f^{-1}(0)$. 

Implicit Occluder
• Exploits the continuity of $f$ to determine conservative visibility bounds implicitly, i.e. without computing the isosurface $f^{-1}(0)$.
• Implicit Occluders are generated based on the change in sign of $f$. 

Implicit Occluder
Implicit Occluders

- Use an octree with per node min-max information
- Find the closest occurrence of sign change
Implicit Occluders

Algorithm can be divided into three main parts:

1. Build screen-space per-pixel occluders
2. For the remaining (visible) nodes, compute and render the isosurface.
3. Use hardware occlusion-culling queries to prune invisible parts of the octree.
Step 1: Build screen-space per-pixel occluders

Finding regions of screen-space overlap between nodes that are above and below the isosurface value.
Step 1: Build screen-space per-pixel occluders

Finding regions of screen-space overlap between nodes that are above and below the isosurface value.

Two-pass strategy:
- all nodes with negative function value are drawn in the first pass
- the nodes with positive function value are drawn in the second pass
Step 1: Build screen-space per-pixel occluders

glClearDepth(1);  Clear Buffers  glClearStencil(0);
Step 1: Build screen-space per-pixel occluders

First render pass: Nodes with negative function value are drawn.

Depth Buffer

Unsigned distance

Negative nodes

Stencil Buffer

Front to Back

∞
Step 1: Build screen-space per-pixel occluders

First render pass: Nodes with negative function value are drawn.

Depth Buffer

Stencil Buffer

Front to Back

∞
Step 1: Build screen-space per-pixel occluders

Second render pass: Nodes with positive function value are drawn, set Depth Buffer to GL_GREATER

Depth Buffer

Stencil Buffer
Step 1: Build screen-space per-pixel occluders

Second render pass: The second pass is performed front-to-back. Sets correct depth for the complete I. Occluders.

Depth Buffer

Stencil Buffer

Front to Back

∞
The stencil buffer is used to determine which pixels have been covered in the first pass.

Step 1: Build screen-space per-pixel occluders
Step 1: Build screen-space per-pixel occluders
Step 1: Build screen-space per-pixel occluders
Step 1: Build screen-space per-pixel occluders
Step 1: Build screen-space per-pixel occluders
Step 1: Build screen-space per-pixel occluders
Step 1: Build screen-space per-pixel occluders
Step 1: Build screen-space per-pixel occluders
Step 1: Build screen-space per-pixel occluders

Depth Buffer

Stencil Buffer

Front to Back
Step 1: Build screen-space per-pixel occluders

The depth of the Implicit Occluders that remain incomplete after the second pass is finally set back to infinity.
Step 1: Build screen-space per-pixel occluders
Step 1: Build screen-space per-pixel occluders

- The yellow region corresponds to the Implicit Occluder.

Depth Buffer

Front to Back
To test if a given node of the octree is visible or not, check whether the bounding box of the node is visible.
If the node is determined to be visible, then the isosurface contained in the node is computed and rendered.
Implicit Occluder Results

Example

Visible Woman

Implicit Occluder

Occluded faces in this example.
Implicit Occluder Results

Example

Engine

Implicit Occluder

Occluded faces in this example.