A Comparison of GPU Box-Plane Intersection Algorithms for Direct Volume Rendering

Chair of Computer Science Prof. Lang
University of Cologne, Germany

Stefan Zellmann (zellmans@uni-koeln.de)
Ulrich Lang (lang@uni-koeln.de)
Problem - Overview

- Direct Volume Rendering (DVR) on GPUs
- Cartesian grids
- Subdivision into Boxes
  - Sort-Last Parallelization
  - Empty-Space Skipping
  - Spatial Search-Structures (e.g. Octrees)
Problem – Empty-Space Skipping
Problem – Box-Plane Intersections
Problem – Box-Plane Intersections

- Many DVR Systems calculate box-plane intersection on the CPU (single threaded)
- E.g. by 12x calculating plane-edge intersection and then sorting intersection points
- Then send polygons over PCIe to the GPU
  ➞ Bandwidth becomes an issue if #boxes increases
Approach - Related Work 2005

- Approach from Rezk-Salama & Kolb 2005 [1]
- Move box-plane intersection to the GPU
- Use vertex shader
- Can process 6 intersections in parallel
- At that time: vertex- and fragment shader on separate chips ➔ load balancing
Approach – Present

- Additional programmable pipeline stages (geometry stage, tessellation stage, ...)

- All shaders execute on the same chip (“Unified Shader Architecture”) ➔ load balancing obsolete

- But can make use of additional stages anyway
6 vertices per box per plane

Vertex Stage

# for 6 vertices
for (edges on path)
{
    isect_edge_plane()
    if (isect)
    {
        ...
    }
}

Fragment Stage

# for each fragment
post_classification()
{
    tex_lookup(3D, trans_func)
}
Approach – Present

1-3 vertices per box per plane

### Vertex Stage

```plaintext
# for 6 vertices
for (edges on path)
{
    isect_edge_plane()
    if (isect)
    {
        ...
    }
}
```

### Geometry Stage

### Fragment Stage

```plaintext
# for each fragment
post_classification()
{
    tex_lookup(3D, trans_func)
}
```

Image
Approach – Present

1-3 vertices per box per plane

Reduce bandwidth

Vertex Stage
Geometry Stage
Fragment Stage

# for 6 vertices
for (edges on path)
{
    isect_edge_plane()
    if (isect)
    {
        ...
    }
}

# for each fragment
post_classification()
{
    tex_lookup(3D, trans_func)
}

Split calculations ➔ finer granularity better for Unified Shader scheduler?
Intersection Algorithm

- Based on Rezk-Salama & Kolb 2005 [1]

- 6 paths from $V_0$ to $V_7$
- Vertex shader processes one path
- For each pair (colors): at least 1 intersection, maybe 2$^{nd}$ with the dotted edge
- If only 1 intersection: Draw overlapping
Intersection Algorithm

- Compare 3 Implementations
  - Vertex Shader only (Rezk-Salama & Kolb)
  - Geometry Shader only
  - Split Shader (Vertex & Geometry Shader)
Geometry Shader Only

- Send only 1 vertex per box per plane ➔ reduce bandwidth
- Emit only actual intersections
- Must process 6 paths in a loop ➔ reduced parallelism
Geometry Shader Only

No GL_POLYGON input type

Must use GL_TRIANGLE_STRIP

Implicit Sorting Problem
Split Shader (Vertex & Geometry)

- Send only 3 vertices per box per plane ➔ still reduced bandwidth over vertex only
- Split the computations ➔ better hints for the scheduler (task parallelism)
- Branching in geometry shader ➔ bad on GPUs
Split Shader (Vertex & Geometry)

- **Vertex shader**: process 3 paths (known intersections)
- **Geometry shader**: process the “dotted edges”
Split Shader (Vertex & Geometry)

Geometry Shader must fill in at right places:

```cpp
if (intersect(P_1)) emit P_1
emit P_0
emit P_2
if (intersect(P_5))
{
    emit P_5
    if (intersect(P_3)) emit P_3
    emit P_4
}
else
{
    emit P_4
    if (intersect(P_3)) emit P_3
}
```
**Split Shader (Vertex & Geometry)**

Geometry Shader must fill in at right places:

```c
if (intersect(P_1)) emit P_1
emit P_0
emit P_2
if (intersect(P_5))
{
    emit P_5
    if (intersect(P_3)) emit P_3
    emit P_4
}
else
{
    emit P_4
    if (intersect(P_3)) emit P_3
}
```

Pass-through from vertex shader
Split Shader (Vertex & Geometry)

Geometry Shader must fill in at right places:

```c
if (intersect(P_1))  emit P_1
emit P_0
emit P_2
if (intersect(P_5))
{
    emit P_5
    if (intersect(P_3))  emit P_3
    emit P_4
}
else
{
    emit P_4
    if (intersect(P_3))  emit P_3
}
```

Minimum amount of branches
Results

- Three test scenarios:
  - Notebook: Apple MBP, AMD Radeon 6490M
  - Desktop: NVIDIA GeForce GTX480
  - Workstation: NVIDIA Quadro FX5800

- Three algorithms:
  - Vertex shader only (Rezk-Salama & Kolb 2005)
  - Geometry shader only
  - Split Shader

- Evaluate with DVR and edge outlines only (no pressure on fragment stage)
Results – Direct Volume Rendering

<table>
<thead>
<tr>
<th>Device</th>
<th>Render Times</th>
</tr>
</thead>
<tbody>
<tr>
<td>Notebook (Radeon 6490M)</td>
<td>10</td>
</tr>
<tr>
<td>Desktop (GeForce GTX 480)</td>
<td>1</td>
</tr>
<tr>
<td>Workstation (Quadro FX 5800)</td>
<td>0.1</td>
</tr>
</tbody>
</table>

- **Vertex**
- **Geometry**
- **Split**

13/01/16 Stefan Zellmann, Chair of Computer Science Prof. Lang, University of Cologne
Results – Only Edges of Polygons

- Render Times

**Notebook (Radeon 6490M)**

- **Vertex**
- **Geometry**
- **Split**

**Desktop (GeForce GTX 480)**

**Workstation (Quadro FX 5800)**

13/01/16  Stefan Zellmann, Chair of Computer Science Prof. Lang, University of Cologne
Conclusions

- We presented a way to reduce PCIe bandwidth consumption
- Results not as positive as expected, on some platforms geometry shaders poorly implemented (Apple MBP)
- Slight performance gain on other platforms (Quadro FX5800)
- Maybe split scheme interesting for future hardware platforms anyway
Related Work

Questions?