Introduction to PowerVR Ray Tracing
Tuesday 18th March, 2014 @ GDC

James A. McCombe
What are we launching today?
What are the applications for this?
Augmented Reality
Virtual Reality a.k.a Multi-persective rendering

Lens distortion and aberration correction

Ultra-low latency rendering

Lenticular Displays
Hybrid rendering in games

- Shadows
- Reflections
- Transparency
- Better scaling with multiple dynamic lights
- Real-time light map updates for the rasteriser
- Easy to bolt on
Fully ray traced graphics

- Brute force path tracing
- Produces photo realism easily
- Pretty much requires all the 3D content to be ray traced
- Possible in today's technology in console / desktop using Wizard
- Probably not practical for fully real-time use in mobile for a couple more generations. Fine for sub-realtime use.
Non-graphics

- Better in-game AI
- Collision Detection
- 3D spatial search
Overview of PowerVR
Power and Bandwidth

- Graphics Watts
- Graphics Bandwidth

- Workstation
- Mobile Workstation
- Laptop
- Light Laptop
- Tablet

- 0W
- 75W
- 150W
- 225W
- 300W

- 0Gb/s
- 30Gb/s
- 60Gb/s
- 90Gb/s
- 120Gb/s
- 150Gb/s
- 180Gb/s
- 210Gb/s
- 240Gb/s
- 270Gb/s
- 300Gb/s
Capability of our mobile graphics

- Almost 5x compute capability
- 4x vec4 MAD → 16x Scalar 2xMAD
- 1.5 → 3.6 Gtexels/s
- OpenGLES 3.0 Support
• An instance is a vertex, pixel or OpenCL thread

• All instances in the task share Program, Uniforms, Parameters, etc.
Shader Uniforms and Constants

Common Store Registers (Shared across instances in the task)

Per Pixel / Vertex data and shader temps

Unified Store Registers (Private registers per instance)
ABS, NEG, etc. ⩽ 2 Flops

Integer Math

ABS, NEG, etc. Format Conversions

Comparison

Format Conversions

12 Cycles of Latency

(not exact)
varying vec2 texCoord;

void main() {
    if (texCoord.x > 0.5) {
        ...
        ...
    } else {
        ...
        ...
    }
Overall capability

- Series6 gives you over 100Gflops for shading on latest mobile devices
- OpenGL ES 3.0 support
- Massive capability gain allowing math heavy shaders
- Mobile graphics is where the actual innovation is happening
- Try it!
A Technical introduction to Wizard
Source 0
ABS, NEG, etc.

Source 1
ABS, NEG, etc.

Source 2
ABS, NEG, etc.

Source 3
ABS, NEG, etc.

Source 4
ABS, NEG, etc.

Source 5
ABS, NEG, etc.

ABS, NEG, etc.

= 2 Flops

Integer Math
Format Conversions

Comparison

Format Conversions

Output 0
Ray RAM

Output 1
Ray RAM

12 Cycles of Latency

(not exact)
New shader type

- Ray Shaders are invoked when a ray intersects a triangle
- Shaders types can emit any number of rays
- OpenRL has a frame shader to emit primary rays
- Existing fragment / pixel shaders can emit rays also!

- GLSL Programming model is the same
- Built-in functions, parameters, etc.
Primitive Objects

- Encapsulate the rendering state for one mesh
- Includes VBOs, uniforms, shader programs, texture bindings, etc.
- Ray tracing unit sorts rays into tasks with common primitive objects

- Persistent between frames
- Mutable objects by the client
Constraints / Limitations

- Shaders cannot wait for results of individual ray trace operations
- Shaders must provide a worst case estimate on number of child rays
- Per-ray user data payload must be carefully managed
Parallelism is on rays NOT pixels

(In this example, ALU width is 4 to make the diagram smaller)
Parallelism is on rays NOT pixels
Relief: Quick ray emission demo
Intersection Processor Array
or
“Who needs Fixed Function?”
Ray:AABB Test

- “Fast Ray-Axis Aligned Bounding Box Overlap Tests with Plucker Coordinates.” – Jeffrey Mahovsky and Brian Wyvill

- 6 lines form the silhouette of the AABB
- 6 planes from the ray origin and each edge vector
- Dot product of plane normal and ray direction vector
- 6 signs must match and be negative.
## USC instruction group packing

<table>
<thead>
<tr>
<th>LD</th>
<th>MUL</th>
<th>ADD</th>
<th>MUL</th>
<th>OR</th>
</tr>
</thead>
<tbody>
<tr>
<td>LD</td>
<td>MUL</td>
<td>ADD</td>
<td>MUL</td>
<td>–</td>
</tr>
<tr>
<td>LD</td>
<td>MUL</td>
<td>ADD</td>
<td>MUL</td>
<td>OR</td>
</tr>
<tr>
<td>LD</td>
<td>SUB</td>
<td>ADD</td>
<td>MUL</td>
<td>–</td>
</tr>
<tr>
<td>LD</td>
<td>SUB</td>
<td>ADD</td>
<td>MUL</td>
<td>OR</td>
</tr>
<tr>
<td>LD</td>
<td>SUB</td>
<td>ADD</td>
<td>MUL</td>
<td>–</td>
</tr>
<tr>
<td>LD</td>
<td>SUB</td>
<td>–</td>
<td>MUL</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MUL</td>
<td>OR</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MUL</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>MUL</td>
<td>–</td>
</tr>
</tbody>
</table>

26 Instructions
(32 if using compressed data formats)
44x less area for this function

<table>
<thead>
<tr>
<th>USC</th>
<th>USC</th>
<th>USC</th>
<th>USC</th>
<th>USC</th>
<th>USC</th>
<th>USC</th>
<th>USC</th>
</tr>
</thead>
<tbody>
<tr>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
</tr>
<tr>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
</tr>
<tr>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
</tr>
<tr>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
</tr>
<tr>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
</tr>
<tr>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
</tr>
<tr>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
</tr>
<tr>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
</tr>
<tr>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
<td>USC</td>
</tr>
</tbody>
</table>

Diagram showing the reduction in area for a function.
The Ray Tracing Unit
and
Coherence Engine
Ray Data Master → Ray Tracing Unit

Course Grain Scheduler → System Memory Bus

Unified Shading Cluster Array

USC0 → Texture Unit → USC1

USCn-1 → Texture Unit → USCn

Ray Tracing Unit

Intersection Processor Array → Coherency Engine

System Memory Bus

Rays grouped by PrimitiveObject (Shader + Resources)
Rays in arbitrary order

Intersection Processor Array → System Memory Interface
AABB Blocks  
Vertex Blocks  
Varyings  

Ascending Virtual Memory Addresses

( around 100Mbytes for 1M triangles including varyings )
(coherence queues)

PrimitiveObject 0

Primitive Object 1

Primitive Object 2
Automatically finds coherence paths
The Scene Hierarchy Generator
Unified Shading Cluster Array

- USC0
- Texture Unit
- USC1
- USCn-1
- Texture Unit
- USCn

Scene Hierarchy Generator

World-space Triangles and associated varyings

System Memory Bus

System Memory Interface
Limitations

- Scene is represented by triangles – same as today
- BVH is in a defined format optimised for construction and traversal
- Triangle order must generally follow a spatially coherent flow
- An approximate scene scale estimation is needed
- Geometry shaders are not inline with the ray tracing pipeline
Strengths

- Shading cluster workload is no higher than a vertex shader
- Only needs to process geometry that actually moved in world space
- Unique algorithm constrains working set to internal registers only
- Single pass operation: in-line with vertex shader execution
- Handles the “long skinny triangles” problem well
- Streaming writes to external memory
- Losslessly compressed output formats due to build algorithm
- Compact logic
Built on sparse log2 oct-tree scaffolding

Level = 1
Level = 2
Level = 3
For each triangle

Select LOD

Leaf VoxelCache

ProcessTriangle()

AssembleParents()

Select LOD

Foreach VBNode in LOD

Compute Parent Voxel

On VoxelCache HIT

Tree VoxelCache

GenerateVBNode()

On eviction

Add VBNode

Per LOD VBNode linked list head pointers

Output Scene Acceleration Structure

Voxel Cache

VBBounds

VBNODE Pool

On eviction

For each triangle

VBBounds
After some triangles...

LOD10
Leaf Count = 3
Tree Count = 0

LOD14
Leaf Count = 0
Tree Count = 0

LOD15
Leaf Count = 3
Tree Count = 0

LOD16
Leaf Count = 8
Tree Count = 0
Assembling parents...

- LOD10
  - LeafCount = 3
  - TreeCount = 0

- LOD14
  - LeafCount = 3
  - TreeCount = 0

- LOD15
  - LeafCount = 3
  - TreeCount = 0

- LOD16
  - LeafCount = 8
  - TreeCount = 0

**Tree Voxel Cache**

**Generate VBNode()**

**Add VBNode to linked list of VBNodes for its LOD**

**On eviction:**

**Flush Leaf Nodes**
After parents for one level have been assembled...

LOD10
LeafCount=3
TreeCount=0

LOD14
LeafCount=0
TreeCount=0

LOD15
LeafCount=3
TreeCount=0

LOD16
LeafCount=8
TreeCount=0

Leaf Head
Tree Head

A0   A1   B0   C0   C1   B1   B2   C2

Leaf Count=8
Tree Count=0
Performance Expectations
Get early access to the programming concepts at:
http://tinyurl.com/9ddlsv2

Talk immediately after this on Hybrid rendering

See the demos at Imagination Booth #402 South Hall
Thank you