A brief introduction

• Joe Davis
  • Lead Developer Support Engineer, PowerVR Graphics
  • With Imagination’s PowerVR Developer Technology team for 6 years

• PowerVR Developer Technology
  • SDK, tools, documentation and developer support/relations (e.g. this session 😊)
PowerVR Rogue Hardware
PowerVR Rogue

Recap

- **Tile-based deferred renderer**
  - Building on technology proven over 5 previous generations
- **Formally announced at CES 2012**
- **USC - Universal Shading Cluster**
  - New scalar SIMD shader core
  - General purpose compute is a first class citizen in the core …
  - … while not forgetting what makes a shader core great for graphics
### TBDR

*Tile-based*

- **Tile-based**
  - Split each render up into small tiles (32x32 for the most part)
  - Bin geometry after vertex shading into those tiles
  - Tile-based rasterisation and pixel shading
  - Keep all data access for pixel shading on chip
- Deferred rasterisation
  - Don’t actually get the GPU to do any pixel shading straight away
  - HW support for fully deferred rasterisation and then pixel shading
  - Rasterisation is pixel accurate
TBDR

Bandwidth savings

- Bandwidth savings across all phases of rendering
  - Only fetch the geometry needed for the tile
  - Only process the visible pixels in the tile

- Efficient processing
  - Maximize available computational resources
  - Do the best the hardware can with bandwidth
TBDR

Power savings

- Maximizing core efficiency
  - Lighting up the USC less often is always going to be a saving

- Minimizing bandwidth
  - Texturing less is a fantastic way to save power
  - Geometry fetch and binning is often more than 10% of per-frame bandwidth
  - Saves bandwidth for other parts of your render
Rogue USC
Architectural Building Block

- **Unified Shading Cluster**
  - Basic building block of the Rogue architecture
  - Laid out in pairs, with a shared TPU

- **1, 0.5 and 0.25 USC designs are special**
  - Different balance in the design
  - Tend to find their way into non-gaming applications
Rogue USC

Shader Architecture

- 16-wide in hardware
- 32-wide branch granularity
  - We run half a task/warp per clock
- Scalar SIMD
- Optimized ALU pipeline
  - Mix of F32, F16, integer, floating point specials, logic ops
Rogue USC

Pipeline datapaths

- **Configurable in the IP core**
  - F16 paths were sometimes optional, thankfully not any more
  - F16 paths performance increased significantly after the first generation

- **Performance in your shader**
  - F32 paths are dual FMAD
  - F16 paths can do different things per cycle depending on shader
  - ISA is available for you to interrogate though, with disassembling compilers
Rogue USC

Scalar

- Scalar ALUs
  - Hard to understate what a benefit this is
  - Seems obvious to do, right?
  - Vector architectures are just hard to program well
  - Scalar isn’t a free lunch
  - Makes performance a lot more predictable for you
Rogue USC

Programmable output registers

- The pixel output registers in the ISA are read/write
- One per pixel
- Width depends on IP core
- We expose it programmatically with Pixel Local Storage
  - Worked closely with ARM
Evolution

Health Warning: Really Bad Diagrams™
Rogue Evolution

- Architecture has changed quite a bit over time
- Rogue in 2010 still mostly looks like a Rogue today
- Significant evolutionary changes across the architecture
- Lots of it driven by developers before the IP is baked
- Lots of it driven by also analysing your stuff anyway
PowerVR Series6XT Rogue

- Host CPU Interface
- Vertex Data Master
- Pixel Data Master
- Compute Data Master
- System Memory Interface
- Core Mgmt Unit
- Unified Shading Cluster Array
  - USC0
  - USC1
  - USCn
- Texture Unit
- Control and Register Bus
- System Memory Bus
- Coarse Grain Scheduler
- Multi-level Memory Cache Unit (MCU)
- Tiling Co-Processor
- Pixel Co-Processor
- 2D Core (TLA)

Extra low power GFLOPS
Supports both LDR and HDR ASTC formats
PowerVR Series6XT Unified Shading Cluster Array

PowerVR Series6XT USC

ALU core (FP32)
FLOP
FLOP
ALU core (FP32)
FLOP
FLOP

ALU core (FP16)
FLOP
FLOP
ALU core (FP16)
FLOP
FLOP

ALU core (FP16)
FLOP
FLOP
ALU core (FP16)
FLOP
FLOP

Special function
FLOP

16 pipelines

8 clusters
Series6 to Series6XT

Lots of lessons learned

- Improved scheduler
- Streamlined ISA
- Improved compute task efficiency
- Completely new F16 datapath
- Improved front-end for sustained geometry performance
- ASTC
PowerVR Series7XT Unified Shading Cluster Array

PowerVR Series7XT USC

- ALU core (FP32)
  - FLOP
  - FLOP

- ALU core (FP16)
  - FLOP
  - FLOP

- ALU core (FP64)
  - FLOP
  - FLOP

Special function FLOP

16 pipelines

2-16 clusters

USC

Optional
Series6XT to Series7XT

Adding features and smoothing off rough edges

- Changed how the architecture scales
- Improved USC
- Streamlined ISA
- Features
  - Hardware tessellation
  - DX11-compliant USC (precision mainly)
  - FP64
Into the future

- Exciting changes being worked on across the architecture
  - USC
  - Front-end
  - Scaling
  - Stuff you want!

- You can help
  - We love feedback about the architecture and how it could best fit what you’re doing
  - Don’t be shy
PowerVR Wizard
Ray Tracing Update
What is Ray Tracing?

Ray tracing is the ability for the shader program for one object to be aware of the geometry of other objects.
PowerVR Architecture

PowerVR Series 6XT

- Host CPU Interface
  - Vertex Data Master
  - Pixel Data Master
  - Compute Data Master
  - Coarse Grain Scheduler
  - Unified Shading Cluster Array
    - USC
    - Shared Texture Unit
  - Tiling Coprocessor
  - Pixel Coprocessor

Control and Register Bus

System Memory Interface
- Core Management Unit
- Multi-level Memory Cache Unit (MCU)
- 2D Core (TLA)
3 Unique Features of Wizard

- Fixed-function Ray-Box and Ray-Triangle testers
- Coherence-Driven Task-Forming and Scheduling
- Streaming Scene Hierarchy Generator
Fixed-Function Ray-Box and Ray-Triangle Testers

44x Less Area for Box Testing
The Coherency Engine lets us process all these rays at the same time.
Streaming Scene Hierarchy Generator
What is Ray Tracing?

Ray tracing is the ability for the shader program for one object to be aware of the geometry of other objects.
Just a few use cases

- Hybrid Shadows, Reflections, etc.
- Augmented Reality
- Production-Quality Renders
- Order-Independent Transparency
- Ambient Occlusion
- Asset creation / compression

- Global Illumination
- Physics & Collision Detection
- Virtual Reality
- Lens correction, Ultra-low latency rendering, Lenticular Displays
- A.I. & Line of Sight Calculations
- Rapid photo-quality output
Ray Tracing Requirements

Sustained Ray Throughput at 1080p, 60fps

Technique vs Ray throughput

<table>
<thead>
<tr>
<th>Technique</th>
<th>Ray throughput (GRays/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Physics / AI / etc.</td>
<td>0</td>
</tr>
<tr>
<td>In-Engine Lightmap baking</td>
<td>0.15</td>
</tr>
<tr>
<td>Hybrid, Reflections</td>
<td>0.5</td>
</tr>
<tr>
<td>Hybrid, Soft Shadows, 1 light</td>
<td>1</td>
</tr>
<tr>
<td>Dynamic AO</td>
<td>1.5</td>
</tr>
<tr>
<td>Interactive GI, (Light Probes)</td>
<td>2.5</td>
</tr>
<tr>
<td>GI, Lens Effects, etc.</td>
<td>3</td>
</tr>
<tr>
<td>Fully ray traced game</td>
<td>3.5</td>
</tr>
</tbody>
</table>

Note: The chart shows the sustained ray throughput for different techniques at 1080p, 60fps.
Session – 11:00-12:00
Enhancing Traditional Rasterization Graphics with Ray Tracing

- James Rumble
  - Developer Technology Engineer, Imagination Technologies

- Session includes:
  - Ray tracing fundamentals
  - Ray tracing pipeline & API key concepts
  - Applications for PowerVR raytracing, e.g. efficient soft-shadows in deferred lighting renderers
PowerVR developer tools
PowerVR Tools

*Release schedule*

• **PowerVR Tools release process**
  • Minor revision roughly every 6 months

• **Recent/upcoming releases**
  • 3.5 SDK (April 2015)
  • 4.0 SDK (due October/November 2015)
PVRTrace

What is PVRTrace?

OpenGL ES API tracer
- OpenGL ES 1.x, 2.0 and 3.x recording libraries
- GUI for analysis

Features
- Inspect, analyse and playback captured data
# New shader analysis

## Shader Analysis

### Selected Pixel (488, 729)

<table>
<thead>
<tr>
<th>UID</th>
<th>Call</th>
<th>Writes</th>
<th>Program</th>
<th>Fragment</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000</td>
<td>glClear</td>
<td>1</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>2024</td>
<td>glDrawElements</td>
<td>52</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>2347</td>
<td>glDrawElements</td>
<td>55</td>
<td>57</td>
<td></td>
</tr>
</tbody>
</table>

### Frame Summary

- Total Vertices: 52,823
- Total Vertex Cost: 4,535,852
- Total Shaded Fragments: 4,782,532
- Total Fragment Cost: 27,113,399
- Average Overdraw: 1.62168

### Fragment Cost

- **Program Costs**: Program 1: 23,648 cycles, 70,944 cost, 1 Tex Reads, 23,648 total reads
- **Shader Costs**: Shader 46: 6,583 cycles, 98,745 cost, 1 Tex Reads, 6,583 total reads
- **Fragment Costs**: Fragment 52: 4,007,496 cycles, 24,044,976 cost, 1 Tex Reads, 4,007,496 total reads
- **Fragment 55**: 203,111 cycles, 1,218,666 cost, 1 Tex Reads, 203,111 total reads

## Draw Calls

<table>
<thead>
<tr>
<th>UID</th>
<th>Call</th>
<th>Vertices</th>
<th>Vertex Cost</th>
<th>Fragments</th>
<th>Fragment Cost</th>
<th>Total Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>2921</td>
<td>glDrawElements</td>
<td>456</td>
<td>61,104</td>
<td>0</td>
<td>61,104</td>
<td></td>
</tr>
<tr>
<td>2940</td>
<td>glDrawElements</td>
<td>342</td>
<td>45,828</td>
<td>0</td>
<td>45,828</td>
<td></td>
</tr>
<tr>
<td>2972</td>
<td>glDrawElements</td>
<td>666</td>
<td>78,588</td>
<td>846</td>
<td>83,664</td>
<td></td>
</tr>
<tr>
<td>2990</td>
<td>glDrawElements</td>
<td>1,275</td>
<td>150,450</td>
<td>3,076</td>
<td>18,456</td>
<td>168,906</td>
</tr>
<tr>
<td>3014</td>
<td>glDrawElements</td>
<td>666</td>
<td>89,244</td>
<td>376</td>
<td>1,880</td>
<td>91,124</td>
</tr>
<tr>
<td>3033</td>
<td>glDrawElements</td>
<td>1,275</td>
<td>170,850</td>
<td>1,043</td>
<td>5,215</td>
<td>176,065</td>
</tr>
<tr>
<td>3064</td>
<td>glDrawElements</td>
<td>666</td>
<td>78,588</td>
<td>8,305</td>
<td>45,830</td>
<td>128,418</td>
</tr>
<tr>
<td>3082</td>
<td>glDrawElements</td>
<td>1,275</td>
<td>150,450</td>
<td>8,292</td>
<td>49,752</td>
<td>202,202</td>
</tr>
<tr>
<td>3106</td>
<td>glDrawElements</td>
<td>666</td>
<td>89,244</td>
<td>1,272</td>
<td>6,360</td>
<td>95,604</td>
</tr>
<tr>
<td>3125</td>
<td>glDrawElements</td>
<td>1,275</td>
<td>170,850</td>
<td>955</td>
<td>4,775</td>
<td>173,625</td>
</tr>
<tr>
<td>3149</td>
<td>glDrawElements</td>
<td>24</td>
<td>384</td>
<td>2,849</td>
<td>8,547</td>
<td>9,931</td>
</tr>
<tr>
<td>3158</td>
<td>glDrawElements</td>
<td>6</td>
<td>96</td>
<td>35,421</td>
<td>106,283</td>
<td>106,359</td>
</tr>
<tr>
<td>3167</td>
<td>glDrawElements</td>
<td>12</td>
<td>192</td>
<td>9,609</td>
<td>28,827</td>
<td>29,019</td>
</tr>
<tr>
<td>3179</td>
<td>glDrawElements</td>
<td>6</td>
<td>96</td>
<td>0</td>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>3188</td>
<td>glDrawElements</td>
<td>6</td>
<td>96</td>
<td>0</td>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>3200</td>
<td>glDrawElements</td>
<td>6</td>
<td>96</td>
<td>0</td>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>3212</td>
<td>glDrawElements</td>
<td>6</td>
<td>96</td>
<td>0</td>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>3221</td>
<td>glDrawElements</td>
<td>6</td>
<td>96</td>
<td>0</td>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>3233</td>
<td>glDrawElements</td>
<td>6</td>
<td>288</td>
<td>7,889</td>
<td>23,667</td>
<td>23,955</td>
</tr>
<tr>
<td>3245</td>
<td>glDrawElements</td>
<td>6</td>
<td>96</td>
<td>0</td>
<td>0</td>
<td>96</td>
</tr>
<tr>
<td>3257</td>
<td>glDrawElements</td>
<td>6</td>
<td>96</td>
<td>15,489</td>
<td>46,467</td>
<td>46,563</td>
</tr>
<tr>
<td>3269</td>
<td>glDrawElements</td>
<td>48</td>
<td>768</td>
<td>314,562</td>
<td>943,686</td>
<td>944,454</td>
</tr>
<tr>
<td>3277</td>
<td>glDrawArrays</td>
<td>4</td>
<td>64</td>
<td>2,595</td>
<td>7,785</td>
<td>7,849</td>
</tr>
<tr>
<td>3282</td>
<td>glDrawArrays</td>
<td>4</td>
<td>64</td>
<td>2,595</td>
<td>7,785</td>
<td>7,849</td>
</tr>
<tr>
<td>3287</td>
<td>glDrawArrays</td>
<td>4</td>
<td>64</td>
<td>2,595</td>
<td>7,785</td>
<td>7,849</td>
</tr>
<tr>
<td>3292</td>
<td>glDrawArrays</td>
<td>4</td>
<td>64</td>
<td>2,595</td>
<td>7,785</td>
<td>7,849</td>
</tr>
<tr>
<td>3297</td>
<td>glDrawArrays</td>
<td>4</td>
<td>64</td>
<td>2,595</td>
<td>7,785</td>
<td>7,849</td>
</tr>
<tr>
<td>3302</td>
<td>glDrawArrays</td>
<td>4</td>
<td>64</td>
<td>2,595</td>
<td>7,785</td>
<td>7,849</td>
</tr>
<tr>
<td>3310</td>
<td>glDrawElements</td>
<td>18</td>
<td>288</td>
<td>48,786</td>
<td>146,358</td>
<td>146,646</td>
</tr>
<tr>
<td>3318</td>
<td>glDrawArrays</td>
<td>4</td>
<td>64</td>
<td>2,436</td>
<td>7,308</td>
<td>7,372</td>
</tr>
</tbody>
</table>
PVRShaderEditor

What is PVRShaderEditor?

Shader profiler & editor

- OpenGL ES shader & OpenCL kernel support

Features

- Syntax highlighting
- As you type performance stats
- Shader disassembly
Session: 14:55-15:40

Optimizing Games for PowerVR

- Paul Sobek
  - Developer Support – Android & Web Browsers

- Session includes:
  - Overview of the new features in the 4.0 SDK developer tools
  - Live demos!
PowerVR SDK & Framework
PowerVR SDK

Release schedule

• PowerVR SDK release process
  • Minor revision roughly every 6 months

• Recent/upcoming releases
  • 3.5 SDK (April 2015)
  • 4.0 SDK (due October/November 2015)
PowerVR SDK

What’s new?

• SDK 4.0: Framework
  • New, modular framework written from the ground up
  • Designed for explicit APIs (e.g. Vulkan). Heavily optimized for OpenGL ES too

• SDK 4.0: Examples
  • New artwork
  • New examples – more to come in the 4.1 release!
Session: 13:30-14:15
Introducing the new PowerVR SDK Framework

■ Gerry Raptis
  ■ Leading Developer Technology Engineer, Imagination Technologies

■ Session includes:
  ■ Intro to the new framework and why we have created it
  ■ In-depth overview of the features and how we have abstracted explicit APIs
  ■ Source code examples
Rogue graphics driver
Rogue graphics driver

_Release schedule_

- **DDK (Driver Development Kit) release process**
  - Reference driver source code released to PowerVR IP licensees
  - Minor revision roughly every 6 months
  - Top-tier customers engage early. Drivers in products shortly after official DDK release
Rogue graphics driver

1.4 DDK

- **Release date**
  - Q4 2014 (release 1)
  - Q1 2015 (release 2)

- **OpenGL ES: Key features (release 1)**
  - OpenGL ES 3.1
    - Compute shaders, shader storage buffer objects, draw indirect and more

- **OpenGL ES: Key features (release 2)**
  - Android Lollipop support
Rogue graphics driver

1.5 DDK

• Release date
  • Q2/Q3 2015

• OpenGL ES: Key features
  • Android Extension Pack (AEP)
    • ASTC, blend equation advanced, GPU shader model 5 and more
  • sRGB PVRTC
  • Pixel local storage
    • 128/256 bits per-pixel on-chip
Rogue graphics driver

1.6 DDK

• **Release date**
  • Q4 2015

• **OpenGL ES: Key features**
  • Bicubic texture filtering
  • Shader group vote
  • Polygon offset clamp
  • Pixel local storage 2
    • Simultaneously write to pixel local storage and a framebuffer attachment
Session: 13:00-13:00
The Golden Rules of Mobile Graphics Performance

- Paul Ly
  - Developer Support Engineer, Imagination Technologies

- Session includes:
  - Our “Golden Rules” for using OpenGL ES efficiently
Session: 15:50-16:20
Optimizing OpenGL ES Games for Android (Google)

- Shanee Nishry
  - Developer Advocate, Google

**Session includes:**
- OpenGL ES performance tips for Android
- OpenGL ES vs. Vulkan
- Source code examples!
Vulkan

About

• **What is Vulkan?**
  • New open standard API developed by the Khronos group
  • Designed for high-efficiency access to graphics and compute on modern GPUs

• **Key features**
  • Minimizes driver overhead and enables multi-threaded GPU command preparation
  • Designed for mobile, desktop, console and embedded platforms
  • Designed for all GPUs - tile based GPUs are first-class citizens!
  • SPIR-V – binary intermediate language for shaders
Vulkan

PowerVR driver status

- **PowerVR Vulkan driver**
  - Driver development on-going
  - Working with key partners on initial content bring up
  - Gnome Horde SIGGRAPH demo
Vulkan

Gnome Horde
Session: 14:25-14:55
Great Looking Graphics on Modern PowerVR GPUs

- Ashley Smith
  - Leading Applications Engineer, Demo Engineering

- Session includes:
  - Overview of the latest PowerVR marketing demos
  - Introduction to the rendering techniques used and how they were optimized for PowerVR
  - Initial thoughts on the Vulkan API after writing the Gnome Horde demo
PowerVR Graphics

*Future roadmaps*

- What drives our roadmaps?
  - Market analysis
  - Customer feedback
  - Developer feedback
Questions?
www.imgtec.com/idc