PowerVR
A Master Class in Graphics Technology and Optimization

January 14th 2012
Company Overview

- **Leading silicon, software & cloud IP supplier**
  - Graphics, video, comms, processor, cloud
  - Licensing and royalty business model

- **Licensed to many top 20 semis & OEMs**
  - Servicing high volume, high growth markets

- **Shipped by most major consumer brands**
  - Smartphones, media players, tablets/netbooks, TVs/STBs, gaming devices, radios, connected devices, dashboards/navigation

- **Strategic product division: PURE**
  - Digital radio, internet connected audio (today)
  - IP business pathfinder, market maker

- **Established technology powerhouse**
  - Founded 1985; London FTSE 250 (IMG.L)
  - Employees: 1,000+
  - UK HQ; operations world-wide
  - Global customer base
A Crash Course in Graphics Architectures
Immediate Mode Renderer (IMR)

- Each triangle is processed to completion in submission order
  - This can result in 'overdraw'
- ‘Overdraw’ wastes processing time, bandwidth, and power
- Buffers kept in system memory, wasting even more bandwidth and power
- ‘Early-Z’ techniques help
  - Only if you sort your geometry front to back
  - Only as good as the granularity of your sorting
• All geometry is processed, then rasterization is done per-tile
  - Tiling allows on-chip buffers to be used

• Rasterization is done in triangle submission order per-tile
  - This can result in ‘overdraw’

• ‘Overdraw’ wastes processing time, bandwidth, and power

• ‘Early-Z’ techniques help
  - Only if you sort your geometry front to back
  - Only as good as the granularity of your sorting
• All geometry is processed, then rasterization is done per-tile
• Tiling allows on-chip buffers to be used
• Hidden Surface Removal (HSR) means only visible fragments are processed
  - Unlike ‘Early-Z’ techniques, its pixel perfect, and submission order independent
  - Only position data is retrieved to perform the next step, saving bandwidth (*1)
  - Normal data, texture data, et al are only retrieved for what's visible (*2)
What is Tile Based Deferred Rendering?
Tiling

• All scene geometry is processed and binned into tiles

• Tiles represent a set area of the framebuffer
  - 32x32 pixels for example, varying by GPU

• Processed geometry data is stored in the parameter buffer

• Fragment processing is done a tile at a time, entirely on-chip
Deferred Rendering

• Two stage process
  - Step One: Hidden Surface Removal (HSR)
  - Step Two: Shading

• HSR is submission order independent
  - No need for applications to submit geometry front to back

• HSR is pixel perfect
  - More effective than ‘Early Z’ techniques

• HSR significantly reduces ‘overdraw’
  - Only fragments that contribute to the final render are processed
  - Saves bandwidth & compute, thereby saving power
Optimized Bandwidth Usage

• Bandwidth usage is the biggest contributor to GPU power consumption

• Saving bandwidth means staying ‘on chip’ as much as possible

• It also means reducing ‘overdraw’
PowerVR Hardware Overview
The Graphics Pipeline

Geometry Processing

Application

Submit Render

Submit Geometry

Driver

USSE

Submit Geometry

Run Vertex Shaders

TA

Tile

Cull

Project

Clip

Parameter Buffer

Primitive List

Vertex Data
The Graphics Pipeline

Fragment Processing

Parameter Buffer

- Primitive List
- Vertex Data

On-Chip Depth/Stencil Buffer

HSR

Tag Buffer

ISP

Run Fragment Shaders

Textured Fetch

Texture Memory

USSE

On-Chip Colour Buffer

PBE

Iteration

Framebuffer

*1

*2
• **Unified Processing Architecture**
  - Processes Vertex, Fragment and GP-GPU tasks
  - SIMD style execution

• **Fed data by the Coarse Grain Scheduler (CGS)**
  - Hardware scheduler tasked with keeping the USSE busy and converting jobs into tasks the USSE can process
Universal Scalable Shader Engine (USSE): Unified Architecture

Dedicated Shader Processing Modules

Processing power wasted

Stall

Fragment

Unified Shader Processing Modules

Higher performance, or lower power consumption

Unified

Latency hidden
Universal Scalable Shader Engine (USSE): Thread Scheduler

- Each USSE has its own scheduler & thread queue
  - Fed tasks by the CGS

- 16 threads per queue
  - 4 threads active simultaneously
  - 4 clock instruction latency

- Zero cycle overhead when swapping threads

- Stalled threads are suspended, waiting threads swapped in
  - Suspended threads can resolve stalls
    - i.e. they can still receive data
Tile Accelerator (TA)

- Clips, projects, and culls geometry output by the USSE

- Vertex data of remaining geometry is written to the Parameter Buffer (PB)
  - This data is also called a ‘Primitive Block’

- Geometry is binned into tiles
  - If an object exists within a tile it is referenced by that tile’s primitive list (also called a ‘Display List’)

- Each tile represents a set area of the framebuffer
  - Tile size varies by hardware
Parameter Buffer (PB)

- **Buffer of data stored in system memory**

  - **Essential for deferring process**
    - Allows geometry and fragment processing to be separated
  
  - **Contains ‘Vertex Data’**
    - Transformed ‘Vertex Data’ for each vertex passed from the TA
    - Shader state for each stored vertex
    - Location data and other data separated to allow max. burst efficiency to ISP and TSP
    - Also known as ‘Primitive Blocks’

  - **Contains ‘Primitive Lists’**
    - Lists of which primitives belong to which tile
    - Also known as ‘Display Lists’
Parameter Buffer Management

• What happens when the PB is full?
  - A render is flushed

• What impact does this have?
  - Flushed renders benefit from HSR performed up to that point
  - Flushed data must be retrieved from the frame buffer as successive tile renders are performed

• How likely is filling the Parameter Buffer?
  - Highly unlikely, default PB size is optimal
    - Big enough you should never hit it
    - Small enough it doesn’t use too much memory

• PB memory can be increased/decreased by the developer on some platforms
• Performs HSR
• Performs Depth/Stencil Operations
  - Very fast Read-Modify-Write
  - Developers can choose not to flush Depth/Stencil buffers
• Contains the ‘Tag Buffer’
  - Buffer used to track visible fragments
• Submits fragments to the TSP, grouped by primitive for optimal cache efficiency
Texture & Shading Processor (TSP)

- Prepares fragments to be processed by the USSE
- Performs iterations
  - i.e. Varyings in shaders
- Pre-fetches independent texture reads
  - i.e. texture co-ords in varyings
- Submits fragment processing jobs to the USSE via the CGS
Micro Kernel

- Specialised control program running on the USSE
- Allows the GPU and CPU to operate with minimal synchronisation
  - Lowers CPU load
  - Improves performance

- Handles interrupts on GPU

- Competing solutions handle interrupts on CPU
  - Increases CPU load
  - Reduces parallelism
  - Can hinder performance
Multi-cores

• Almost linear performance scaling
  - 95%+ efficiency in typical performance conditions
  - Small fixed overhead in memory footprint
  - Increase <1% overall memory bandwidth per frame

• Geometry processing load-balanced across cores

• Each additional core allows another tile to be processed in parallel

• All multi-core logic is handled by the hardware
  - Completely transparent to the developer
Other Considerations
Alpha Blending

• Blending is done entirely on-chip

• Only takes a single pass through the ISP and TSP per primitive
  - Very fast Read-Modify-Write operations

• Unlike IMRs, blending does not waste system memory bandwidth

• Benefits from HSR
  - i.e. Occluded fragments won’t be processed
Multisampling Anti-Aliasing

- **4x MSAA performed completely on-chip, per-tile**
  - Using a sub-sampling method
  - Minimal impact on memory bandwidth

- **ISP performs HSR for each sub-sample**

- **Subsamples are combined by the Pixel Back End (PBE) and sent to the frame buffer as a single fragment**
  - Minimizing impact on memory bandwidth even further
Internal True Colour

- On other GPUs a 16 bit buffer can cause loss of precision
  - Multiple read-modify-writes can result in significant banding & dithering artefacts

- On-chip colour buffer always performs 32 bits precision blending
  - …regardless of the frame buffer precision

- High precision blending reduces banding

- Dithering is only applied once (when the colour buffer is flushed to the frame buffer)
Application Optimization
Bottlenecks

- Identifying which stage of the pipeline is limiting performance is crucial
- Understand the GPU architecture and how it relates to the pipeline
- Use the right tools to discover the bottlenecks
Optimizing blindly wastes time for no gain
The Right Tools

- **PVRTrace: API Tracer**
  - Analysis API calls
    - Are there redundant calls?
    - Is the render state changing?
    - What was submitted to each draw call?

- **PVRTune: GPU Profiler**
  - TA/3D view to easily spot big issues
  - Specific counters for detailed analysis

- **PVRScope: Analysis Library**
  - Retrieve hardware counter information
  - Send custom marks and counters to PVRTune
  - Analyse performance in your own tool chain.
Past Observation

Most Likely

- CPU
- Texture Cache
- Texture Fetches
- Fragment Shader Instructions
- Memory Bus
- Parameter Buffer

Least Likely

- Vertex Processing FIFO
- Vertex Shader Instructions
- Z-Load/Store
- Clipping and Culling
- Tiling
- ISP Setup
The Golden Rules
Golden Rule 1: Understand Your Target Device

- No two devices are identical
- Even when they look the same, they might not be
- Different SoCs might have different bottlenecks
  - Benchmark under a variety of conditions, including different SoCs
- Know the hardware and how it works
  - If you don’t understand the hardware, you don’t understand the bottlenecks
  - If you don’t understand the bottlenecks your optimizations will be poorly targeted
Golden Rule 2: Don’t Waste GPU Time for Little Gain

• a.k.a The Principle of ‘Good Enough’

• Don't waste polygons on un-needed detail

• Textures should never be much larger than their size on screen
  - Why waste time loading a 1024x1024 texture if it’s never going to appear bigger than 128x128?

• If the user won't notice it, don’t waste time drawing it

© 2012 Imagination Technologies Ltd.
Golden Rule 3: Avoid Accessing Render Targets

- Accessing render targets from the CPU is very bad for performance
  - The CPU blocks till the GPU flushes the render
  - Then the GPU blocks till the CPU finishes

- Serialising a massive parallel processor is bad – find a better way

- If you have to do it, check out the ‘GL_OES_EGL_image_external’ and ‘EGL_KHR_fence_sync’ extensions
Golden Rule 4: Promote Calculation up The Chain

- The fewer calculations performed, the better the performance
- Don’t do a calculation you don’t need to do
- If you can do it once per scene, do it once per scene
  - If you can’t, try and do it per vertex
    - There are generally fewer vertices in a scene than fragments.
- If you can, pre-bake
  - E.g. lighting
- Remember, ‘Good Enough’
Golden Rule 5: Use VBOs and Indexed Geometry

• **VBOs benefit from driver level optimisations**
  - They also save memory
  - Vertex Array Objects (VAOs) may be even better

• **Index your geometry**
  - It makes your data smaller
  - It also benefits from driver level optimisations
Golden Rule 6: Batch, Batch, Batch!

- Group static objects, and draw once
  - Fewer render state changes means lower driver overhead and better performance

- Branching in shaders can allow better batching
  - Just don’t branch to ‘discard’

- Use texture atlases

- Sort objects by render state

<table>
<thead>
<tr>
<th>Setup</th>
<th>Draw Call</th>
<th>Setup</th>
<th>Draw Call</th>
<th>Setup</th>
<th>Draw Call</th>
<th>Setup</th>
<th>Draw Call</th>
</tr>
</thead>
</table>

**Versus**

| Setup | Batch | Your | Draw | Calls |
Golden Rule 7: Compress Your Textures

• The lower the bitrate the better the performance

• Currently, the smallest formats are PVRTC & PVRTC2, at 2 & 4bpp RGB/RGBA

• Don’t confuse this with PNG or JPG which are decompressed in memory
  - Usually to 24bpp or 32bpp

• PVRTC is read directly from the compressed form
  - It stays in memory at 2bpp or 4bpp

• Use MIP-mapping
  - Better cache utilisation
  - Improved image quality when textures suffer minification
Golden Rule 8: Avoid Alpha Test/Discard

- Removes advantages of ‘Early-Z’ techniques and HSR
  - Fragment visibility isn’t known until fragment shader is run
- Prefer Alpha Blending
Golden Rule 9: Opaque, then Alpha Test, then Alpha Blend

- Render opaque objects first

- Render Alpha Test/Discard objects second

- Render Alpha Blended objects last

- Rendering in this order will make optimal use of HSR
Golden Rule 10: Use ‘Clear’ and ‘DiscardFrameBufferEXT’

• Calling ‘Clear’ ensures the previous render isn’t uploaded to the GPU

• By default, the depth/stencil buffers are written to system memory at the end of a render

• Calling DiscardFrameBufferExt(…) ensures the these buffers aren’t written to system memory
  - Look for the ‘GL_EXT_discard_framebuffer’ extension

Do both if you can!
Golden Rule 11: Perform Rough Culling

- The fastest geometry to process is the geometry that isn’t submitted.

- HSR removes a lot of fragment work, but not submitting redundant geometry is even better.
Golden Rule 12:
Target a Sensible Framerate

• Steady frame rates give smoother experiences…

• …but don’t update if you don’t have to.
  - There is no need for an idle UI with no animation to run at 30FPS

• If your framerate is unstable, target the lower rate
  - If your game fluctuates between 30 and 60, targeting 30 gives a smoother experience and lowers power consumption

• Do you need 60fps?
  - Forcing a lower rate will allow the GPU to go to sleep, saving power
  - Or reduced rate gives more GPU processing time per-frame
Finally....
Rogue
• 20x higher performance than previous generation in equivalent products

• New API features
  - DirectX 10.1
  - OpenCL 1.x
  - OpenGL 3.x/4.x

• New, advanced bandwidth saving mechanisms
  - Rogue will use the same bandwidth, or less, for the same scene, as Series 5

• PVRTC1 & PVRTC2 texture compression support

• Micro Kernel on a dedicated housekeeping processor

• More threads, improved scheduling, better latency hiding

• What to expect?
  - Designed to deliver 100GFLOPS+
  - Capable of scaling to TFLOPS range

• Announced Rogue SoC from STMicro (Nova A9600) targeting in excess of 210 GFLOPS
  - “…More than 13 gigapixels per second effective fill rate”
PowerVR Insider Program

- [http://www.powervrinsider.com](http://www.powervrinsider.com)

- Free to join

- Benefits of being a PowerVR Insider
  - PowerVR Insider SDK downloads
  - Open Developer Forums
  - Direct email contact with engineers from PowerVR Developer Technology
    - devtech@imgtec.com
  - Documentation
  - FAQs
  - Training (web based and onsite)
PowerVR
A Master Class in Graphics Technology and Optimization

devtech@imgtec.com