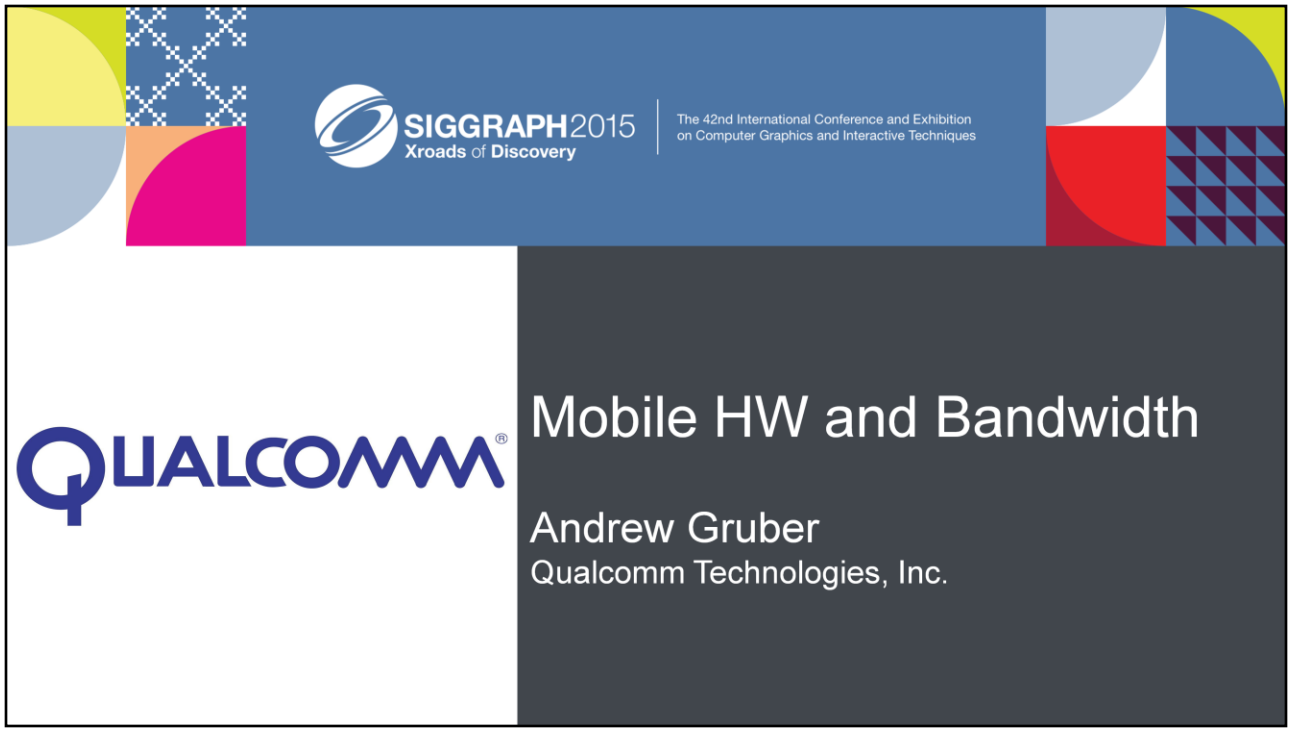

SIGGRAPH2015
Xroads of Discovery


SIGGRAPH2015
Xroads of Discovery
The 42nd International Conference and Exhibition
on Computer Graphics and Interactive Techniques
QUALCOMM
Mobile HW and Bandwidth
Andrew Gruber
Qualcomm Technologies, Inc.

Agenda and Goals

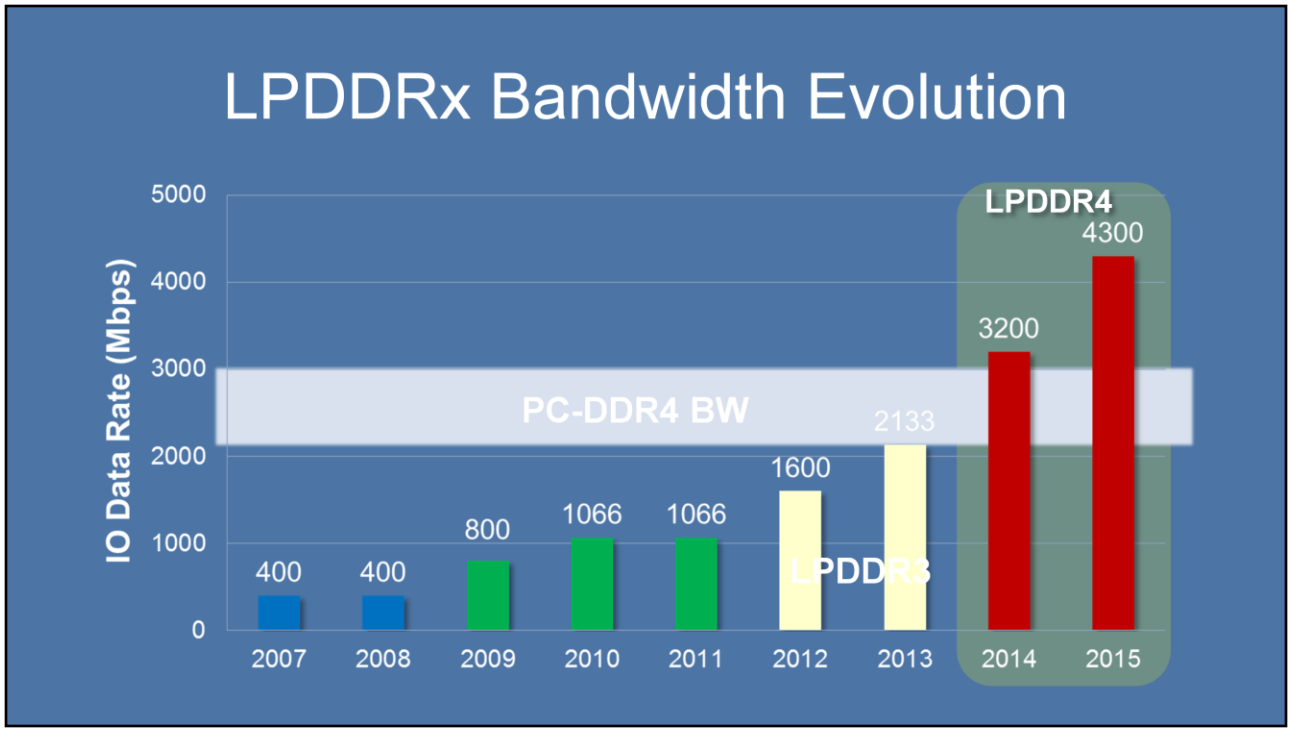
- Describe the Power and Bandwidth challenges facing Mobile Graphics
- Describe some of the Power Saving approaches used in Mobile Graphics to deal with these challenges
  - By no means a systematic account, just a sample of techniques used by major providers
- Provide hints and best practices to allow SW to take advantage of the above approaches.

The Mobile space has a lot of the demands of desktop – but doesn't have the same memory bandwidth or power and heat envelope.

After all – it goes in your pocket and has no cord. This has lead to a variety of innovative techniques showing up in the Mobile space.

Some might eventually migrate to desktop and console as well, as the power/heat envelope issues can start to be limiting in that

Space as well.



Bandwidth of mobile memory chips is approaching those used for PC main memory. However:

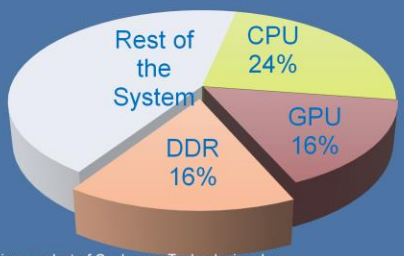
The bus width used in mobile is typically only 64 bits – rather than the 128 bit wide commonly used for PCs

Mobile Graphics has no separate 'VRAM' buffer that holds an entire frame buffer for rendering as high end discrete PC graphics systems do.

Mobile Power Consumption for a Game

- Game running at 1080p / 60 fps
- DDR consumption is ~16% for this gaming use case on Qualcomm ® Snapdragon™ processors.
  - Comparable to internal GPU power and CPU power consumption

Mobile device - Power Breakdown



Snapdragon is a product of Qualcomm Technologies, Inc.

Typical OpenGL ES 3.1 based game

DDR bandwidth consumption is a major component of power usage during game scenarios. In UI type scenarios and battery days-of-use considerations – it is even more dominant.

A key to improving mobile graphics is keeping DDR bandwidth consumption down.

Resolutions

- Mobile Resolutions are high and increasing

Resolutions are increasing – even faster than available memory bandwidth. So the issues are getting worse.

Binning/Tiling Architecture to Save Memory Bandwidth

- GPU has a dedicated fast tile buffer.
- The rendering surface is split up into “bins”.
- Rendering pass draws only pixels that are visible (saving GPU cycles and power)
- The GPU efficiently burst writes all blended pixels from the tile buffer as a single layer to the frame buffer in system memory – referred to as a “Resolve”
- Note that typically there is no Depth traffic to system memory and any MSAA color traffic is first down-filtered before being written to System Memory.

	Opaque	Alpha Blended	Alpha Blended
Tiled	Texture Read	Texture Read	Texture Read, Resolve (Write)
Direct	Texture Read, FB Write	FB Read, Texture Read, FB Write	FB Read, Texture Read, FB Write

GPU has a dedicated fast memory buffer called GMEM.

The rendering surface is split up into “bins”. Bin size is determined by format (including Z) and render target resolution divided into the total amount of GMEM. The number of bins varies based on both the hardware version, the target resolution size, and the target display format.

One triangle “Visibility Stream” is created per bin (we show this in detail on the next slide)

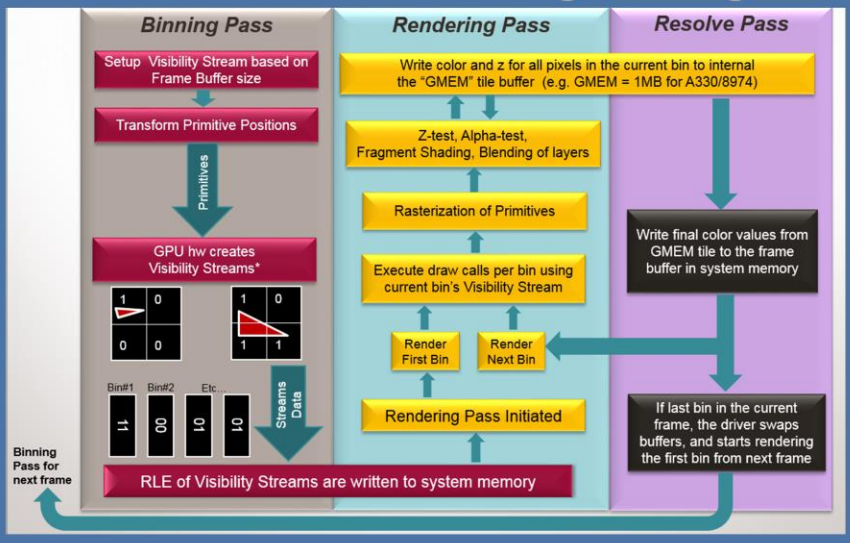
Rendering pass uses visibility streams to draw only pixels that are visible in that bin (saving GPU cycles and power)

All pixels for each bin are drawn into the GMEM, with a very high bandwidth that matches the capabilities of the GPU

The GPU efficiently burst writes all blended pixels from GMEM as a single layer to the frame buffer in system memory – referred to as a “Resolve”

Avoiding resolves is an important optimization strategy (more on this later)

Qualcomm Technologies' Approach to Binning/Tiling



- The only output of Binning Pass is a compressed 1 bit/primitive 'visibility stream'
- Avoids write bandwidth for transformed positions and overflow/
- management of transformed VBO.

The binning/tiling architecture uses 3 passes for rendering.

The first pass creates the visibility streams

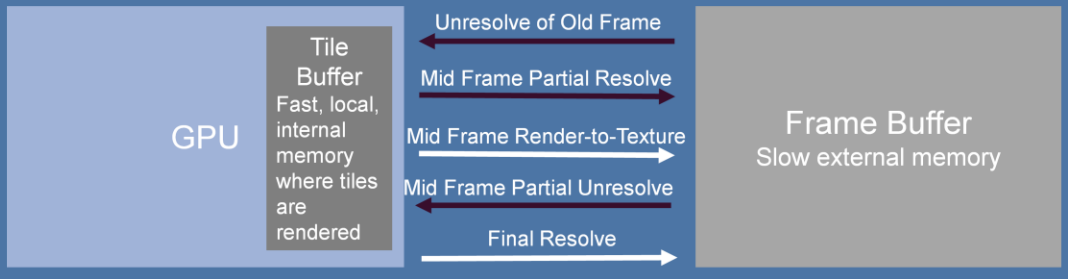
The second pass renders each bin by processing the primitives/triangles in the bin and compositing into GMEM.

In the third pass the GMEM values are written to the frame memory in system memory.

When all bins are complete, the driver swaps buffers and starts the binning pass again for the next frame

Tiling Gotcha's

- Use 'Frame Buffer Discard' to indicate that old data will be overwritten
  - Otherwise we need to copy ('unresolve') the old frame buffer back into the Tile Buffer
  - 'Frame Buffer Clear' works as well – but means we have to waste time clearing the Tile Buffer
- Mid frame calls of glTexImage2D, glBufferData, glReadPixels, glCopyTexImage2D, etc., force the driver to Save/Restore



The Black arrows indicate 'extra' bandwidth that could be avoided.

"Unresolve Old Frame' can be avoided by clears or discards

Mid Frame Partial Resolve/Mid Frame Partial unresolved could be avoided by a complete render of the Texture prior to the start of the Frame Buffer Render.

Bandwidth Saving Technique – Deferred Rending

- 'Hidden Surface Removal' or 'Deferred Rendering' can avoid rendering occluded pixels – even for non-sorted command streams.
- Works by deferring color render until Z value is known – similar to a Z pre-render.
- Doesn't work for all cases -- Alpha Blending or Pixel Kill are problematic
- Still Depth sort when possible. Allows 'Early Z' to remove a lot of processing

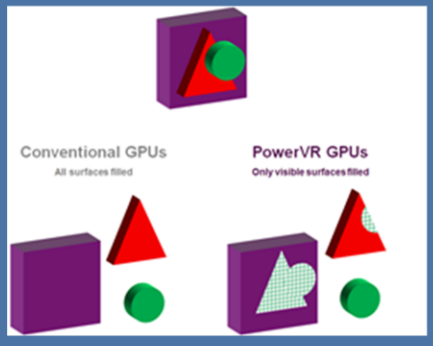


Image used with permission of Imagination Technologies

In the above image – it is assumed that rendering proceeds back-to-front –i.e. Purple Square, then Red Triangle, then Green Circle

A conventional renderer will render all of the Purple Square – it has no knowledge that some of the Purple Square will be later obscured.

But the above technique of rendering a complete Z prior to any color provides such knowledge

A tiled rendering system reduces the buffering cost of this approach.

Bandwidth Saving Technique – Forward Pixel Kill

- Use 'Early Z' test to 'reach into' processing pipeline and kill pixels that are still being processed.
- Many of the benefits of 'Deferred Rendering'
  - Kills even with back-to-front Ordering
  - Can kill pixels even if the tile has Alpha Blending.
- Effectiveness drops as tile size or tile complexity increases

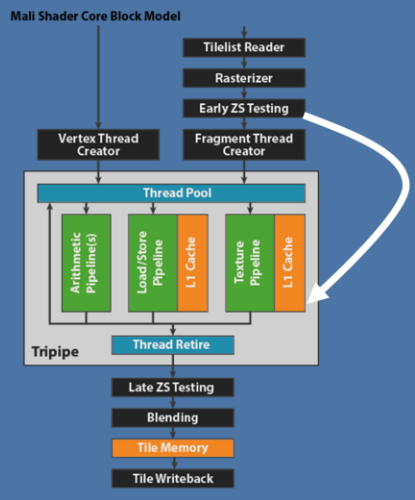


Image used with permission of ARM

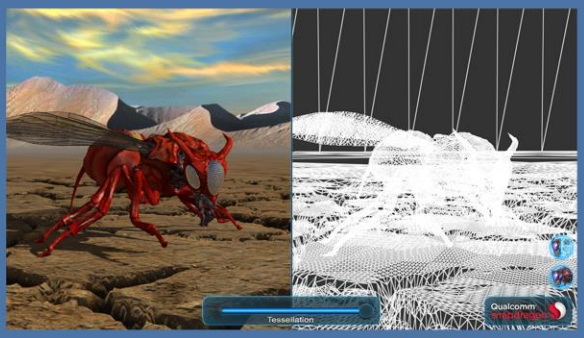
Rather than doing a complete render of just the Z – this approaches uses the existing pipeline delay to abort processing of pixels that previously passed the Z test –

But are discovered to be obscured while still in the pipeline.

Bandwidth Saving Technique – Tessellation

- Supported in AEP/OpenGL ES 3.1
- Provides the ability to render very smooth and high detailed scenes

Tessellation OFF



Tessellation ON

Tessellation is of particular importance with the new 3.1/AEP APIs because of the large performance benefits that can be achieved.

Tessellation Shaders

GPU Hardware creation of additional detail from low-resolution models

Saves memory bandwidth and footprint because the GPU is creating additional geometry without the need to send anything through the bus to the GPU.

Provides ability to render very smooth and high detailed scenes.

Bandwidth Saving Technique - Tessellation

3 Different Possibilities and Bounding Box Extension

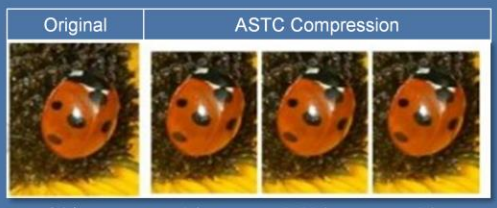
1. Tessellate on Binning Pass
   I. Works – but requires extra bandwidth to Save/Read tessellated Vertices
2. Tessellate on Render Pass
   I. Saves bandwidth - but requires tessellated primitive to appear in all bins
   II. Extra tessellation cycles in each Render Pass
3. Tessellate on Both Passes
   I. Limits 'extra' Render Pass work for non-visible Bins
   II. But pay extra tessellation cycles during Binning
4. Implementing a 'Bounding Box' extension can help limit Render pass work in (2) or Binning Pass work in (3).

Bandwidth savings associated with Tesselation is not universal for tiling architectures. The most bandwidth savings requires a trade-off with processor/tessellation cycles – i.e. duplicate tessellation to avoid extra vertex bandwidth.

Bandwidth Saving Technique - Texture Compression

- Texture compression is a critical tool for reducing bandwidth - which improves performance and lowers power consumption
- All textures containing color data should be compressed using one of the recommended formats below.

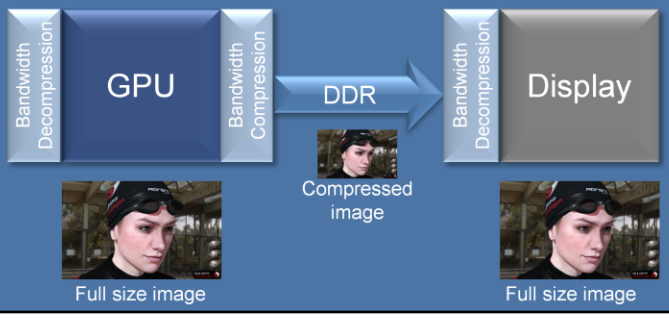
Target API	Format
Open GL ES 2.0	ATC
Open GL ES 3.0	ETC2
Open GL ES 3.1	ETC2 or ASTC



Texture compression techniques have gotten better. Most mobile HW is optimized for compressed textures – so no reason not to use them whenever possible.

Bandwidth Saving Technique – Render Target Compression

- Lossless – means that can be invoked anytime the CPU doesn't need access. Doesn't save space -- only memory bandwidth.
- End-to-End (understood by both Texture Unit and Display) means displayable Targets can be compressed – and saves bandwidth on the Scan-out as well.



This is a compression technique that is used 'under-the-covers'. Pure Bandwidth (not memory space) compression.

By enabling the Display to understand this format as well -- bandwidth is saved both on write and display of the frame buffer.

Summary

- Bandwidth is becoming the dominant performance and power consumption consideration for mobile graphics
- Mobile chips use ‘tricks’ to try minimize bandwidth as much as possible.
- Please cooperate with us by using mobile ‘friendly’ algorithms and using compressed surfaces.