Next-gen Mobile Rendering

Niklas Smedberg
Senior Engine Programmer, Epic Games

Timothy Lottes
Senior Rendering Programmer, Epic Games
Introduction

• Niklas Smedberg, a.k.a. “Smedis”
  – 17 years in the game industry
  – Graphics programmer since the C64 demo scene

• Timothy Lottes
  – Industry background in games, GPU hardware, and movie special effects
  – Programming since days of the 80386 PCs
Content

• Hardware
  – How next-gen mobile hardware really works
  – Case study:
    • ImgTec Series 6 G6430 (“Rogue”)
    • Qualcomm Adreno 420 (Snapdragon 805)

• Software
  – Next-gen mobile rendering techniques in Unreal Engine 4
  – Bringing AAA PC graphics to mobile
Next Generation Mobile Hardware

• Big leap in features and performance

• Full-featured (e.g. OpenGL ES Next, DirectX 10/11)

• Peak performance now comparable to consoles (Xbox 360/PS3)
  – About 300+ GFLOPS and 26 GB/s

• New goal: Bring AAA PC graphics to mobile
GDC 2012 vs. GDC 2014

- Next-gen performance?

20x?
Performance Trends (FP16 GFLOPS)

2010 SGX 535
2011 SGX 543MP2
2012 SGX 543MP3
2013 G6430
2014 Adreno, K1, GX6650
Tile-based Mobile GPU

- Mobile GPUs are usually tile-based (next-gen too)
  - Tile-based: ImgTec, Qualcomm*, ARM
  - Direct: NVIDIA, Intel, Qualcomm*, Vivante

* Qualcomm Adreno can render either tile-based or direct to frame buffer
  - Extension: GL_QCOM_binning_control
Tile-Based Mobile GPU

Summary:
• Split the screen into tiles
  – E.g. 32x32 pixels (ImgTec) or 300x300 (Qualcomm)
• The whole tile fits within GPU, on chip
• Process all drawcalls for one tile
  – Write out final tile results to RAM
• Repeat for each tile to fill the image in RAM
ImgTec Tile-based Rendering Process

Per Tile:

- Hidden Surface Removal
- Pixel Processing (Top-most only)
- Tile Memory
- Frame Buffer (RAM)
ImgTec Series 6 G6430

- One GPU core
- Four shader units (USC)
  - 16-way scalar
- **FP16: Two SOP per clock**
  - \((a \times b + c \times d)\)
- **FP32: Two MADD per clock**
  - \((a \times b + c)\)
- 154 GFLOPS @ 400 MHz
  - 16-bit floating point
ImgTec Series 6 vs SGX 5xx

- Each shader unit (USC) operates on multiple tiles in parallel
  - Don’t need a whole separate GPU core for that anymore
  - Tiles are processed roughly left-to-right, top-to-bottom

- Throughput:
  - 48 FP16 scalars per clock
  - Compared to 4 floating point scalars per clock

- Parallelism:
  - 32 vertices/pixels per Wave
  - Compared to 4 (or 1) vertices/pixels per Thread
ImgTec Series 6: Improvements

- Supports OpenGL ES 3.0 and beyond
- Scalar, not vector
- No additional cost for dependent texture reads
  - Pixelshader math on texture coordinates
  - Texture coordinates can be in .zw (swizzle)
- Coherent dynamic flow control at full speed (not 1/4th speed)
- MRT support: 16 bytes per pixel in total
- FP16 is the minimum precision (no more lowp)
FP16 Is Faster Than FP32

- **ImgTec Series 6**: 50% faster
  - FP16 pipeline: Two SOP per clock
  - FP32 pipeline: Two MADD per clock

- **ImgTec Series 6XT**: 100% faster
  - FP16 pipeline: Four MADD per clock
  - FP32 pipeline: Two MADD per clock

- **Qualcomm Snapdragon**: 100% faster
Example: Scalar vs. Vector Performance

- GLSL shader source:
  
  \[
  \text{vec3 } V = A \times B + C \times D;
  \]

- Executed on SGX 543 GPU:  
  
  \[
  \text{vec4 } V' = A \times B; \\
  \text{vec4 } V = C \times D + V';
  \]

  (75% speed, .w component is wasted)

- Executed on G6430 GPU:  
  
  \[
  \text{half } V.x = A.x \times B.x + C.x \times D.x; \\
  \text{half } V.y = A.y \times B.y + C.y \times D.y; \\
  \text{half } V.z = A.z \times B.z + C.z \times D.z;
  \]

  (Full speed)
ImgTec Rendering Tips

• Hidden Surface Removal
  – For opaque only
  – Don’t keep alpha-test enabled all the time
  – Don’t keep “discard” keyword in shader source, even if it’s not used

• Group opaque drawcalls together

• Sort on state, not distance
Qualcomm Snapdragon Rendering Process

Game Developer Conference, March 17-21, 2014
Qualcomm Snapdragon Rendering Tips

- **Traditional handling of overdraw (via depth test)**
  - Cull as much as you can on CPU, to avoid both CPU and GPU cost
  - Sort on distance (front to back) to maximize early z-rejection

- **The Adreno SIMD is wide**
  - Check your ALU utilization in the Adreno Profiler and optimize
  - Minimize temp register usage
  - Use long shaders with a lot of ALU instructions
  - Avoid dependent texture fetches (or cover the latency with a lot of ALUs)
Framebuffer Resolve/Restore

- Expensive to switch Frame Buffer Object on Tile-based GPUs
  - Saves the current FBO to RAM
  - Reloads the new FBO from RAM
Framebuffer Resolve/Restore

• Clear ALL FBO attachments after new frame/rendertarget
  – Clear after eglSwapBuffers / glBindFramebuffer
  – Avoids reloading FBO from RAM
  – NOTE: Do NOT do unnecessary clears on non-tile-based GPUs (e.g. NVIDIA)

• Discard unused attachments before new frame/rendertarget
  – Discard before eglSwapBuffers / glBindFramebuffer
  – Avoids saving unused FBO attachments to RAM
  – glDiscardFramebufferEXT / glInvalidateFramebuffer
iOS Performance Profiling

- How we clear FBO at the beginning of every render pass
- Other important performance info
Programmable Blending

- GL_EXT_shader_framebuffer_fetch (gl_LastFragData)
- Reads current pixel background “for free”
- Potential uses:
  - Custom color blending
  - Blend by background depth value (depth in alpha)
    - E.g. Soft intersection against world geometry for particles
  - Deferred shading without resolving GBuffer
    - Stay on GPU and avoid expensive round-trip to RAM
Tips

• Think scalar! Avoid using unnecessary components
  – Avoid: \((\text{vec4} \times \text{float}) \times \text{float}\) \((8 \text{ MUL})\)
  – Use: \(\text{vec4} \times (\text{float} \times \text{float})\) \((5 \text{ MUL})\)

• Prefer 16-bit floating point operations (mediump)

• Leverage ALU to hide memory fetches
  – E.g. ALU can be faster than using lookup-textures
More Tips

• Don’t switch back and forth between mediump/highp
  – ImgTec: Requires shader instructions to convert each time
  – Qualcomm: Many conversions are free

• Branch spatially coherent for many pixels
  – Uses predication to ignore invalid path
Part II: Unreal Engine 4 Rendering Techniques
Core Optimization: Opaque Draw Ordering

• All platforms,
  – 1. Group draws by material (shader) to reduce state changes

• Then for all platforms except ImgTec,
  – 2. Skybox last: 5 ms/frame savings (vs drawing skybox first)
  – 3. Sort groups nearest first: extra 3 ms/frame savings
  – 4. Sort inside groups nearest first: extra 7 ms/frame savings

• Timing from a UE4 map on a Qualcomm based device
Core Optimization: Render Resolution

- Reducing render resolution to get perf/pixel
  - Example Phone = 4.7 tex/ms/pixel at native 0.7 Mpix display
  - Example Tablet = 0.9 tex/ms/pixel at native 3.1 Mpix display
  - Example Last Gen Console = 3.9 tex/ms/pixel

  • at HDTV native 2.1 Mpix display
Comparison: PC
Comparison: Mobile
Authoring Consistency

- Core motivating factor in design
  - Authoring consistency between PC and Mobile

- Authoring environment for both platforms
  - Physically based shading model
  - High dynamic range linear color space
  - High quality post processing
Consistency: Material Editor

- One material for many platforms
  - Artist authored feature levels to scale shader perf from PC to mobile

- Using cross compiler tool to retarget HLSL into GLSL
Consistency: Physically Based Shading

• Same material model as PC
  – See “Real Shading in Unreal Engine 4” (Brian Karis) [1]

• Mobile adjustments
  – Analytic approximation of Environment BRDF (ALU instead of TEX)
  – Normalized Phong spec distribution (faster and still energy conserving)

Consistency: HDR Directional Lightmaps

• Pair of compressed textures
  – HDR color with log luma encoding
  – World space 1\textsuperscript{st} order spherical harmonic luma directionality

• Optimized for Mobile and PVRTC compression
  – Mobile encoding lacks separately compressed encoding for alpha
  – Therefore mobile encoding for HDR color is different than PC
  – PC uses \{RGB/Luma, LogLuma in Alpha\}
  – Mobile uses \{RGB/Luma * LogLuma, no Alpha\}
Directional Light + SDF Shadows
Directional Light + SDF Shadows

- UE4 mobile applies the primary directional light dynamically
- This light is shadowed by baked signed distance field (SDF) shadows
  - Uncompressed texture stores signed distance to nearest shadow transition
Image Based Lighting
Image Based Lighting

- Mip choice based on roughness
  - Same as PC, filtering same as PC
- PC uses FP16 IBL cubemaps
- Mobile uses Decode(RGBM)^2 IBL cubemap encoding
- PC blends multiple cubemaps per surface and provides parallax correction
- Mobile supports one infinite distance cubemap per object
Mobile Post Processing Pipeline

FP16 RGB=Color A=Depth
Light Shaft (Sun) Mask
A=Depth to A=CoC+Sun
[conversion done on chip if possible]

1/4x1/4 Smart Reduction

1/8x1/8
1/16x1/16
1/32x1/32
1/64x1/64

Bloom Filter Tree

1/8x1/8
1/16x1/16
1/32x1/32
1/64x1/64

1/4x1/4 DOF Near Dilation

1/2x1/2 DOF Filter

1/2x1/2 DOF Downsample

1/4x1/4 Lightshaft Filter Pass 1

1/4x1/4 Lightshaft Filter Pass 2

Tonemapping

1/4x1/4 Final Filter Passes and Merge
{Light Shafts, Bloom, Vignette}

Anti-Aliasing
Mobile Depth of Field
Mobile Depth of Field

• Special mobile algorithm
  – Uses 1/4 x 1/4 resolution near DOF dilation pass
    • Used to enable Bokeh to bleed out over background
  – DOF blur works at 1/2 x 1/2 resolution
    • Later applied to full resolution in tonemapping pass
    • Extra ultra fine blur tap taken in full resolution pass for better transitions
Mobile Depth of Field: Details
Mobile Depth of Field: Filtering

- Filtering is a weighted average of 4 taps
- **4 taps at different distance from pixel center**
  - Enables smooth in-to-out-of-focus transition
  - Bokeh is slightly off center but covers 16 half res texels
- **Remove color bleeding problems when bilinear filtering**
  - Packed circle of confusion size in alpha
  - Pre-weighting color by alpha (undo weighting after fetch during filtering)
  - Watch out for lack of FP16 denormal support on mobile hardware!
Packed Circle of Confusion and Sun Intensity

- Optimization done for Depth of Field and Light Shafts
  - Represent sun intensity and circle of confusion (CoC) in one FP16 value
  - Saves needing an extra render target

Depth (0 to 65504)

CoC (0 to 1)

Sun Intensity (1 to 65504)

0 = Max Near Bokeh
0.5 = In Focus
1 = Max Far Bokeh and No Sun
65504 = Max Sun (still at Max Far Bokeh)
Vignette + Bloom + Light Shafts : Optimized

- Composited at 1/4 x 1/4 resolution
  - Using pre-multiplied alpha FP16
  - Composite shader does last bloom and light shaft filter pass too
    - Optimized to minimize resolves (or round trips off chip)
- 3 effects applied to scene with one full-res TEX fetch
  - Applied in the tonemapper pass
Mobile Bloom
Mobile Bloom

• Lower quality version which limits resolves to work around render target switch limits
  – Limited effect radius and less passes
• Faster and higher quality version for devices without need for the work around
  – Standard hierarchical algorithm with some optimizations
  – Down-sample from 1:1/4 res first (shared with light shaft)
  – Then down-sample in 1:1/2 resolution passes
  – Single pass circle based filter (instead of 2 pass Gaussian)
    • 15 taps on circle during down-sampling
    • 7 taps for both circles during up-sample+merge pass
Mobile Light Shafts

- Filters at 1/4 x 1/4 resolution
- Monochromatic with artist controlled tint on final composite
  - Bloom and light shaft down-sample pass shared
  - RGB = color for bloom, A = light shaft intensity
- Light shaft filter runs in tonemapped space (8-bit/channel)
  - Applied in linear (reverse tonemap before tint and composite)
- Filtering done by 3 passes of 8 taps
Film Post Tonemapping
Film Post Tonemapping: Details

- Same controls as PC
  - Unused controls optimized out (shader permutations)
- Similar controls as high end photo editing software
- Applied in Linear HDR color space before tonemapping
  - Critical for high quality color
- White point and shadow color tint adjustment
- Saturation and channel mixer (applied as 3x3 color matrix)
Film Post Tonemapping: Curve

- 18% grey is the perceptual midpoint
- “Crush” controls film latitude (size of linear segment above and below 18% grey)
  - Higher latitude increases the region of the curve with no distortion on the ratio of color primaries
- Contrast controls slope of linear segment

- 18% Grey Pivot
- Crush Highlights
- Contrast
- Crush Shadows

Input Axis (drawing scaled non-uniformly)
Film Post Tonemapping: Mobile Shader

Film Post
- Shadow Tint (optional)
- Tint, Tonemap Curve

Color Matrix
- {Saturation, Channel Mixer} (optional)

Quantization Grain (optional)

RGBA8 LDR sRGB Output

Film Grain (optional)

Blend in DOF (optional)

1/2x1/2 DOF

Blend in {Bloom, Vignette, Light Shafts} (optional)

Film Grain Jitter (optional)

1/4x1/4 Pre-multiplied Alpha

RGBA16F HDR Linear Color

Linear to sRGB

UNREAL ENGINE
Anti-Aliasing
Anti-Aliasing: Detail

Geometric Edges

Normal Edges
Anti-Aliasing

• Using a spatial & temporal anti-aliasing filter
  – 2x temporal super-sampling (higher quality surface shading)
  – Blending two jittered frames after tonemapping (32 bpp, perceptual colorspace)
    • With some special logic to remove ghosting/judder (not using motion reprojection)

• Not currently using MSAA
  – Needed super-sampled shading for HDR physically based shading model
Exposure Mosaic: Linear Blending at 32-bpp

- Used on GPUs without FP16 and without sRGB support to support linear blending
  - Supports \{0.0 to 2.0\} dynamic range
  - Mosaic mode simulates a 12-bit/channel linear framebuffer
  - Forward shading and linear blending with exposure mosaic based on gl_FragCoord
  - Demosaic in tonemapping pass
Exposure Mosaic: Output Quality
All Techniques Combined
And one more thing...
Unreal Engine 4

Full source code available now!

unrealengine.com

Includes all C++, shaders, tools, content

$19/mo
Bonus Slides

The following slides were not part of the live presentation . . .
GDC 2012 vs. GDC 2014:

**SGX 543MP2:**
- Two GPU cores
- Two SIMDs per core (vec4)
- Two MADD per clock
- 200 Mhz
- 13 GFLOPS

**ImgTec G6430:**
- One GPU core
- Four 16-way SIMDs (scalar)
- Two SOP per clock (FP16)
- 400 MHz
- 154 GFLOPS (FP16)
ImgTec Series 6: Latency Hiding

- Highly multithreaded to hide memory latency
  - Thousands of threads
  - Switch to new thread when kicking off a memory read
- Each SIMD has their own thread manager
- One Wave: 32 vertices/pixels, one instruction pointer
- Switching Wave is free (every two clock cycles)
  - Memory fetch
  - Branching
  - Other dependency