Optimizing the Unreal Engine 4 “Soul” Demo for Galaxy Note 10.1

Jack Porter
Engine Development and Support Lead
Epic Games Korea
Introduction

• Epic Games
  – Founded 1991 by Tim Sweeney
  – HQ in Cary, North Carolina
  – Subsidiaries in Utah and Seattle, Korea, Japan, UK and Poland

• Jack Porter
  – Originally from Australia, came to Korea in 2003
  – Founding member of Epic Games Korea
  – 15 years of experience working on Unreal Engine
Content

• Unreal Engine

• Soul Demo

• How we made the demo
  – Unreal Engine 4 pipeline
  – Next-gen mobile rendering techniques in Unreal Engine 4
Unreal Engine 1-3

History

- Unreal Engine 1 (1998)
  - PC
    - Unreal, Unreal Tournament

- Unreal Engine 2 (2002)
  - PC online
    - Lineage 2
  - PS2/Xbox

- Unreal Engine 3:
  - PC online
    - TERA, BLESS, Blade & Soul
  - PS3/Xbox 360
    - Gears of War series
  - iOS
    - Infinity Blade series
  - Android
    - Blade for Kakao
Unreal Engine 4

• **Subscription model**
  – Engine and full source code available for $19/month subscription

• **New Features**
  – Redesigned editor tool
  – DirectX 11 / OpenGL 4
  – Physically based lighting and shading
  – Deferred rendering
  – “Blueprints”

• **CHALLENGE**: how can we bring these features to mobile?

“Infiltrator” demo
“Soul” Demonstration UE4 Content

Purpose:

• See what’s possible in mobile rendering on today’s high end mobile devices
  – iOS/Android
  – Mali/ImgTec/Adreno

• Push the limits of draw calls and driver features to drive innovation in mobile industry (benchmark)

• Using a regular PC content pipeline

• Demonstrated at GDC 2014
Soul Demo Video
Soul Statistics

- 490 MB apk size
  - mostly textures
- Around 400 draw calls per frame
- Around 400,000 triangles
Forward rendering instead of deferred shading

- Bandwidth cost for multiple render targets on ES 3.0 still too high in current devices.

High quality static lighting

- Global Illumination captured on PC
- Cubemaps for image-based lighting

Physically based shading model
Physically Based Shading

- Metallic 0 to 1
- Metal with roughness 0 to 1
- Non-metal with roughness 0 to 1
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)

Material Pipeline

Artist creates shader graph in Unreal Editor tool.
Network graph generates HLSL functions

Combined with hand-written HLSL code for our Physically Based lighting model

We use this directly on PC

```
half3 GetMaterialBaseColor(FMaterialPixelParameters Parameters)
{
    MaterialFloat3 Local5 = (1.00000000 - Material.VectorExpressions[0].rgb);
    MaterialFloat4 Local6 = ProcessMaterialColorTextureLookup(Texture2DSample(Material.Texture2D_1, Material.Texture2D_1Sampler, Parameters.TexCoords[0].xy));
    MaterialFloat3 Local7 = (Local6.rgb + MaterialFloat3(0.00000000, 0.00000000, 0.00000000));
    MaterialFloat Local18 = dot(Local6.rgb, MaterialFloat3(0.30000000, 0.58999997, 0.11000000));
    MaterialFloat Local19 = (Local18 + -0.40000000);
    MaterialFloat Local10 = (Local19 * -5.00000000);
    MaterialFloat Local11 = min(max(Local10, 0.00000000), 1.00000000);
    MaterialFloat Local12 = (Local10 + dot(MaterialFloat3(0.00000000, 0.00000000, 0.00000000), MaterialFloat3(0.30000000, 0.58999997, 0.11000000)));
    MaterialFloat Local13 = (dot(MaterialFloat3(0.00000000, 0.00000000, 0.00000000), MaterialFloat3(0.30000000, 0.58999997, 0.11000000)) / Local12);
    MaterialFloat Local14 = (Local13 - 0.50000000);
    MaterialFloat Local15 = (Local14 * 4.00000000);
    MaterialFloat Local16 = (Local15 + 0.50000000);
    MaterialFloat Local17 = min(max(Local16, 0.00000000), 1.00000000);
    MaterialFloat Local18 = (Local11 * Local17);
    MaterialFloat3 Local19 = lerp(Local6.rgb, Local7, MaterialFloat(Local18));
    MaterialFloat Local20 = (Local19 * Local5);
    return Local20;
}
```
Material Pipeline

For mobile, we cross-compile HLSL to GLSL

Version and GLSL extensions enabled using preprocessor

```glsl
#version 300 es
out mediump vec4 out_FragColor;

uniform highp samplerCube ps4;
uniform highp sampler2D ps0;
uniform highp sampler2D ps1;
uniform highp sampler2D ps2;
varying highp vec4 var_TEXCOORD0;
varying highp vec4 var_TEXCOORD1;
varying highp vec4 var_TEXCOORD2;
varying vec4 var_TEXCOORD7;

void main()
{
    vec4 t0;
    vec4 t1;
    t1.xyzw = vec4(gl_FragCoord);
    t0.xyzw = t1;
    vec4 t2;
    highp vec4 t3;
    t3.xy = var_TEXCOORD4.zw;
    t3.zw = var_TEXCOORD5.zw;
    vec3 t4;
    vec3 t5;
    t5.xyz = vec3(t3.xyz);
    t4.xyz = t5;
    highp vec4 t6;
}
```
Material Pipeline: Mobile Problems

- GLSL shaders are compiled by the device

- Shader compiler bugs on device can cause failures and require workaround
  - eg Register allocation very slow or fails for some shaders

- Extensions we want may not be available
  - GL_EXT_texture_lod
  - GL_EXT_shader_framebuffer_fetch
Image Based Lighting
Image Based Lighting
Image Based Lighting

- Image Based Lighting captured on PC and stored in cubemap
  - PC uses FP16 cubemaps
  - Mobile uses Decode(RGBM)^2 cubemap encoding

- Mipmap choice based on material roughness
  - Same as PC, filtering same as PC
  - We use OpenGL ES 3.0 on T628, GL_EXT_shader_texture_lod where supported on other devices

- Mobile supports one infinite distance cubemap per object

- PC blends multiple cubemaps per surface and provides parallax correction using (texture arrays)
HDR Lighting

• On PC we use a 32-bit floating point framebuffer and a post-processing tonemapper

• On Mobile we would like to use FP16 framebuffer
  – Not available on Mali T-628
  – `GL_EXT_color_buffer_half_float` / `GL_RGBA16F_EXT` supported on a few devices

• On Mali we use “Mosaic Exposure” technique to simulate a 12-bit /channel linear framebuffer on `GL_RGBA8`
Linear Blending at 32-bpp

- Used on mobile GPUs without FP16 and without sRGB support to support linear blending
  - Supports {0.0 to 2.0} dynamic range
  - Forward shading and linear blending with exposure mosaic based on gl_FragCoord
  - De-mosaic in tonemapping pass
HDR Lighting: 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 ETC/PVRTC compression
  – No 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\}
Terrain Rendering
Terrain Rendering
Terrain Rendering
Optimizations for Mali-T628: Terrain

Landscape uses continuous LOD (morphing in vertex shader)

- To avoid duplicating data, vertices are reused for lower LOD levels
- Terrain divided into chunks
Vertex Buffer Layout

Because the vertices are shared between LODs, at low LOD levels the indices are sparse.

The Mali driver runs the vertex shader for **ALL** vertices between minimum and maximum vertex!
Vertex Buffer Layout

Simple rearrangement* of the vertex order to place the shared vertices first increased the frame from 10 to 30 FPS!

We don’t see this behavior on other mobile GPUs.

*Actual layout is more complex to support more than these two LOD levels
UE4 Mobile: Challenges

• Driver differences between devices and vendors
  – Shader compiler differences
    • Extensions and workarounds
    • Compile time and bugs

• Different performance characteristics
  – Resource and framebuffer management
  – Drawcall overhead
  – Thermal throttling behavior

• Version fragmentation – many UE4 issues are solved by doing a firmware upgrade
  – Updates are a big problem in the Android ecosystem!
UE4 Mobile: Challenges

• Draw call overhead
  – Soul demo kept drawcalls to under 400 which seems to be the upper limit
  – This needs to get better!

• Memory management
  – Difficult to get an idea of memory available to app
  – Soul demo used texture streaming from flash into a fixed pool, to ensure a more consistent memory footprint
Thank You!

• Please come and see how we authored the Soul content using the Unreal Engine 4 Editor

• Q & A