Project SHIELD and Tegra 4: Redefining AFK

Andrew Edelsten (Manager, Tegra Developer Technologies)
Paul “Hodge” Hodgson (Manager, Tegra Developer Technologies)
Overview

Andrew
- Tegra 4 & Project SHIELD
- Game considerations for Project SHIELD
- NVIDIA development tools for Android

Hodge
- Tegra 4’s new GPU features
- Anatomy of Tegra 4’s GPU
Tegra 4

“NVIDIA Tegra 4 is a promising processor that’s going to bring a whole new level of gaming to mobile devices.”

“if you enjoy the web browsing experience on your iPad, you’re going to be pretty pleased what NVIDIA has to offer here.”

“If you want to take better pictures on your mobile device, NVIDIA’s Chimera computational photography engine is the technology you’ve been waiting for.”

Hottest gadgets
MWC 2013

Better photography:
NVIDIA Tegra 4 HDR camera
Tegra 4 Family

Tegra 4 ("Wayne")
World's Fastest Mobile Processor

Tegra 4i ("Grey")
1st Integrated Tegra 4 LTE Processor

Superphone / Tablet

Smartphone

Quad CPU
Cortex A15, 4+1
72 Core
Optional with i500

NVIDIA GPU
Cortex A9 r4, 4+1
60 Core
Integrated i500

LTE
Chimera*

* Chimera is NVIDIA’s Computational Photography
Mobile Processor, Ultrabook Performance

<table>
<thead>
<tr>
<th>Mobile Processor</th>
<th>Geekbench2 Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tegra 3 1.3GHz</td>
<td>1479</td>
</tr>
<tr>
<td>A6X 1.4GHz</td>
<td>1774</td>
</tr>
<tr>
<td>APQ8064 1.5GHz</td>
<td>2031</td>
</tr>
<tr>
<td>Exynos5250 1.7GHz</td>
<td>2433</td>
</tr>
<tr>
<td>Core-i3 1.5GHz</td>
<td>3082</td>
</tr>
<tr>
<td>Core-i3 1.8GHz</td>
<td>3648</td>
</tr>
<tr>
<td>Core-i5 1.6GHz</td>
<td>3807</td>
</tr>
<tr>
<td>Core-i5 1.7GHz</td>
<td>4592</td>
</tr>
<tr>
<td>Tegra 4 1.9GHz</td>
<td>4679</td>
</tr>
</tbody>
</table>

Intel Core i3-2377m 1.5GHz, Core i3-3217U 1.8GHz & Core i5-2467m 1.6GHz, Core i5-3317U 1.7GHz all have 17W maximum TDP
Competitive data published on Geekbench website; Tegra 4 1.9GHz measured on reference platform
Project SHIELD
Project SHIELD

- Tegra 4 powered
- 5 inch 720p & multitouch display
- Console grade controller
- High speed Wi-Fi
- Full connectivity (HDMI, USB, microSD, headphone)
- Pure Android (currently Jellybean)
Tuned Port, Bass Reflex Speakers
Two Open Platforms — One Amazing Portable

Android

PC
SHIELD Development Considerations

- Support landscape screen orientation
  - Don’t assume device is a phone and lock to portrait based on DPI
- Don’t require touch (for Android games) or a mouse (for PC games)
- Test using HDMI
  - Is everything possible without getting up?
  - How does it look on a big screen?
- Optimize your PC game for Streaming (see next NVIDIA session in this room)
- Controller is King!
Controller is King

- Auto-detect the controller
- Include a controller map overlay
- Sub 20fps extremely noticeable
- Remove all on-screen touch elements
- Use Android Input and code to the Built for Tegra standard

UI should:
- Have visual focus indicator (highlight, arrows, etc)
- Use classic standards for navigation (9 & 6 for OK etc)
- Allow use of all elements (menu items, checkboxes, sliders etc)
- Include “exit” in the main and pause menus

Controllers Everywhere
See the Tegra developer documentation “How To: Support Android Game Controllers” and NativeGamepad sample for a great guide on how to handle multiple controllers on all Android devices
Developing for Android

- Setting up an Android development environment can be tricky
- Android SDK, NDK, ANT, Eclipse, adb.. Grrr!
- Native debugging.. Double grrrr!
- Is that gcc configuration quite right?
Tegra Android Development Pack

GET STARTED in minutes NOT hours
INSTALLED all tools required for Tegra Android

CPU DEBUGGING with Nsight Tegra
GPU DEBUGGING with PerfHUD ES
OPTIMIZE applications with Tegra Profiler
REFERENCE docs, samples & tutorials

OPTIMIZED for Tegra Android development
FLASHES Tegra DevKit with OS Image
CONFIGURED for debugging and profiling
INCLUDES Kernel symbols and DS-5 support

http://developer.nvidia.com/develop4tegra
Native Code Samples

- **Android lifecycle**
  - Lifecycle can be tricky
  - Highly recommend using “Native Basic” as a base

- **OpenGL ES**

- **Input device handling**
  - Multitouch
    - Beware the stylus!
    - Use `getToolType()`
    - See [http://goo.gl/eRdIC](http://goo.gl/eRdIC)
  - Sensors
  - Gamepad

Tegra Developer Tools

Nsight Tegra
Visual Studio and Eclipse integrations
Full Android build management
Native Android CPU debugging
Breakpoints in both Java and Native

Tegra Profiler
Maximize multi-core CPU utilization
Quickly identify CPU “hot spots”
Identify thread contention issues

PerfHUD ES
Examine and debug OpenGL ES frames
Automated bottleneck analysis
Edit shaders at runtime

http://developer.nvidia.com/develop4tegra
The Tegra 4 GPU

Paul “Hodge” Hodgson
Tegra Developer Technologies
Depth on Tegra 4

Many additional extensions supported
- OES_depth24
- OES_depth_texture
- OES_depth_texture_cube_map
- EXT_shadow_samplers
- NV_shadow_samplers_cube
- NV_shadow_samplers_array

Hardware PCF
Tegra 4 - Shadows

- Depth textures
- Percentage-closer filtering
- Soft-edged, AA shadows
- No fragment shader emulation

Unfiltered

Percentage Closer Filtered
Tegra 4 - Shadows

**Shadows**
- Depth textures
- Percentage-closer filtering
- Soft-edged, AA shadows
- No fragment shader emulation

**Comparison**
- Unfiltered
- Percentage Closer Filtered
HDR on mobile

- Improved half float support
  - OES_texture_half_float_linear
  - OES_texture_half_float
  - EXT_color_buffer_half_float

- Introducing sRGB
  - EXT_sRGB
  - NV_sRGB_formats
  - NV_generate_mipmap_sRGB
## Tegra 4 GPU Features

<table>
<thead>
<tr>
<th>ES Features</th>
<th>Tegra 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>FBO_render_mipmap</td>
<td>✓</td>
</tr>
<tr>
<td>Uniform Buffer Objects</td>
<td>✓</td>
</tr>
<tr>
<td>Separate Shader Objects</td>
<td>✓</td>
</tr>
<tr>
<td>Framebuffer Blit</td>
<td>✓</td>
</tr>
<tr>
<td>Copy Buffer (ARB_copy_buffer)</td>
<td>✓</td>
</tr>
<tr>
<td>Explicit Attribute Locations</td>
<td>✓</td>
</tr>
<tr>
<td>Surface-less context creation</td>
<td>✓</td>
</tr>
<tr>
<td>Texture Storage</td>
<td>✓</td>
</tr>
<tr>
<td>Pixel Buffer Objects</td>
<td>✓</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ES Features</th>
<th>Tegra 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>24-bit Depth</td>
<td>✓</td>
</tr>
<tr>
<td>FP16 Texture Filtering</td>
<td>✓</td>
</tr>
<tr>
<td>Multisampling</td>
<td>✓</td>
</tr>
<tr>
<td>Occlusion Queries</td>
<td>✓</td>
</tr>
<tr>
<td>Non-square Matrices</td>
<td>✓</td>
</tr>
<tr>
<td>Multiple Render Targets</td>
<td>✓</td>
</tr>
<tr>
<td>R8, RG8, RGB8, RGBA8, RGB565</td>
<td>✓</td>
</tr>
<tr>
<td>SRGB8_ALPHA8, RGBA4, RGB5_A1</td>
<td>✓</td>
</tr>
<tr>
<td>{R, RG, RGBA}{8}{I,UI}</td>
<td>✓</td>
</tr>
</tbody>
</table>
Vertex and primitive processing

- ~60 cycles in vertex shader per visible primitive
  - 60 DP4
  - 15 vec4*mat4
- Parallel to fragment shader
- Vertex cache
- Use optimized triangle lists
Raster and early z

- 8 fragments per clock
- Can skip fragment shader if depth/stencil killed
- Can skip fragment shader if no surface writes
- Depth compression
- Depth pre-pass, consider it now
Fragment shader

- 4 “mad” units with a single VLIW instruction, can for example:
  - 4x MAD
  - 2x DP2A + MFU
  - 1x DP3A + 2x MFU
  - 1x DP4 + MFU

- Use ‘lowp’ precision specifier where possible:
  - 2x ‘lowp’ DP4 + MFU

- Single clock lerps per mad unit
- Input & output modifiers
- Free precision conversion
- Ideal ALU/TEX ratio of 3
Tegra 4 GPU - Improved Effective Scaling

- Consumes half the vertex attribute bandwidth of Tegra 3
- Texture L2 cache added
  - reduces over-fetch across pipes
  - increases total effective cache size
- Improvements to pixel pipe to increase ALU utilization

<table>
<thead>
<tr>
<th>Improvement</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU local register state</td>
<td>Reduces power and perf cost of allocating registers</td>
</tr>
<tr>
<td>Increase max pixel shader registers over Tegra 3</td>
<td>Up to 24 vs. Tegra 3’s 16 fp20 registers per pixel (more threads in flight)</td>
</tr>
<tr>
<td>Increase instruction tables for ALU</td>
<td>Improves the efficiency of long programs</td>
</tr>
<tr>
<td>Add Multi-Function Unit (MFU) to ALU</td>
<td>Better MFU scaling, improve ALU utilization</td>
</tr>
</tbody>
</table>
## Tegra 4 vs Tegra 3 GPU stats

<table>
<thead>
<tr>
<th></th>
<th>Tegra 4/ Tegra 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Vertex Shader</td>
<td>8x</td>
</tr>
<tr>
<td>Fragment ALU</td>
<td>8x</td>
</tr>
<tr>
<td>Pixel Rate</td>
<td>2.6x</td>
</tr>
<tr>
<td>Texture Rate</td>
<td>2.6x</td>
</tr>
<tr>
<td>Memory Rate</td>
<td>2.3x</td>
</tr>
<tr>
<td>Z-Kill Rate</td>
<td>1.3x</td>
</tr>
<tr>
<td>Triangle Rate</td>
<td>1.3x</td>
</tr>
</tbody>
</table>

**Tegra 4 - 72 Core GPU @ 672 MHz**  
4 pixel pipes * 3 ALUs/pipe * 4 MADS/ALU + 6 VPEs * 4 MADS/VPE

**Tegra 3 - 12 Core GPU @ 520 MHz**  
2 pixel pipes * 1 ALU/pipe * 4 MADS/ALU + 1 VPE * 4 MADS/VPE
Wrapping Up

Questions/Comments?

Resources
- NVIDIA Developer Zone - https://developer.nvidia.com
- NVIDIA Developer Forums - https://devtalk.nvidia.com

Presentation References
- “Moving Games into the Cloud, Technologies and Architectures”
- NEXT UP IN THIS ROOM
- “Eliminating Texture Waste: Borderless Realtime Ptex”
  - Friday, March 29, 10:35 am - 11:00 am in Room 307, South Hall

NVIDIA’s GDC Exhibit Booth: #1602