Optimizing Unity* Games on Android* OS for Intel® Architecture: A Case Study

Unity is one of the most popular game engine for the mobile environment (Android and iOS*), and many developers are using it to develop and launch games. Before Unity supported Android on Intel platforms, games were executed on an emulator that changed ARM* native code to Intel native code. Some non-native x86 games running on Intel platforms did not work at all and others had performance issues. With the growth in mobile market share of Intel processors, many developers are now interested in supporting Android on x86 architecture and want to know how to optimize their games.

This article will show a performance gain with native support on Android and share some tips for increasing performance on Intel architecture using Hero Sky: Epic Guild Wars as an example.

Figure 1. Hero Sky: Epic Guild Wars

Innospark, maker of Hero Sky: Epic Guild Wars, has significant experience in mobile game development using a variety of commercial game engines and also has its own in-house game engine. Hero Sky: Epic Guild Wars is its first Unity-based game launched for the global market. With an increasing number of downloads from the Google Play* store, the company began to get complaints that the game did not work and that it lagged on some Intel processor-based devices with Android . So Innospark decided to port and optimize the game for Android OS on Intel architecture. This article explains what Innospark did for optimization with profiling results from Intel® Graphics Performance Analyzers (Intel GPA), like changing drawing order and removing unneeded alpha blending.

Introduction

Hero Sky: Epic Guild Ward is an online combat strategy style game supporting full 3D graphics. Innospark developed and optimized it on an Intel Atom processor-based platform (code named Bay Trail). The Bay Trail reference design and specifications are listed below.

CPU	Intel Atom processor Quad Core 1.46 Ghz
OS	Android* 4.4.4
RAM	2GB
Resolution	1920x1200

3Dmark* ICE Storm Unlimited Score	10,386
Graphics score	9,274
Physics score	17,899

Table 1. Bay Trail 8”reference design specification and 3Dmark* score

Below is a graph showing a performance comparison between non-native x86 and native x86 code on the Bay Trail reference design.

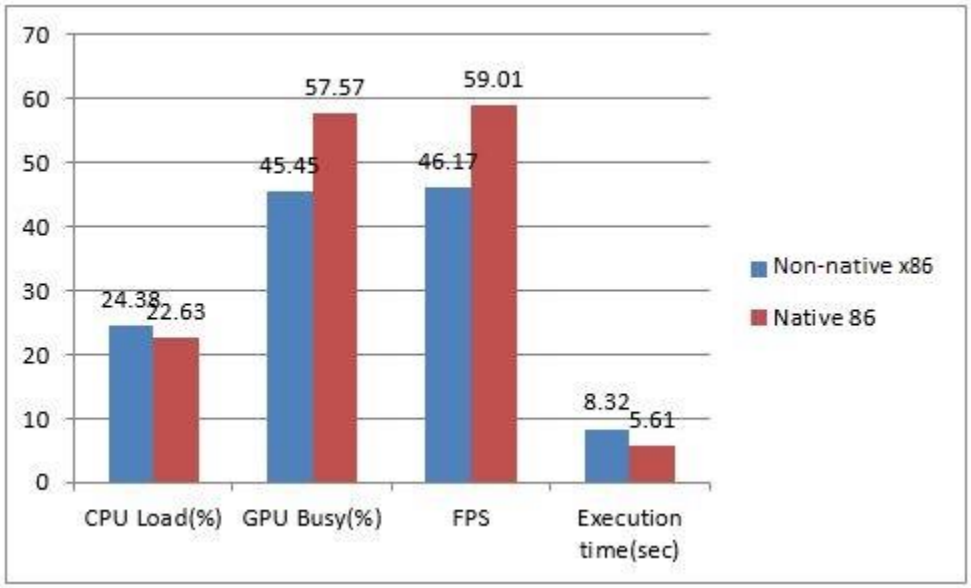

Figure 2. Performance gains with x86 native support

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark* and MobileMark*, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more information go to http://www.intel.com/performance.

After the game was ported for Android on Intel architecture, the CPU load decreased about 7.1%, FPS increased about 27.8% and execution time decreased about 32.6%. However, GPU Busy increased about 26.7% because FPS increased.

Innospark used Intel GPA to find CPU and GPU bottlenecks during development and used the analysis to solve graphics issues and performance.

Intel GPA System Analyzer measured 59.01 FPS as the baseline performance. Graphics Frame Analyzer, which measures FPS only on the GPU side, measured 120.9 FPS. The reason the FPSs are different is that System Analyzer is monitoring live activity of the process, which includes both CPU and GPU work and Graphics Frame Analyzer includes GPU-related work with the CPU activities directly related to submission of data to the driver and GPU.

Deep-dive analysis using Graphics Frame Analyzer

Figure 3 Screen capture of the baseline version

After being ported, the game showed 59.01 FPS. We analyzed it in more detail using the Graphics Frame Analyzer in order to decrease the GPU Busy and CPU Load. The tables below show the information captured using the Graphics Frame Analyzer.

Total Primitive Count	4,376
GPU Duration, ms	8.56 ms
Time to show frame, ms	9.35 ms

Table 2. Baseline frame information

Type	Erg	GPU Duration (ms)	GPU Memory Read(MB)	GPU Memory Write(MB)
Sky	1	1.43 ms	0.2 MB	7.6 MB
Terrain	5	1.89 ms	9.4 MB	8.2 MB

Table 3. the high draw call cost of the baseline version

Analyze and optimize high draw call

Remove unneeded alpha blending

When a display object uses alpha blending, the runtime must combine the color values of every stacked display object and the background color to determine the final color. Thus, alpha blending can be more processor-intensive than drawing an opaque color. This extra computation can hurt performance on slow devices. So we want to remove unneeded alpha blending.

The Graphics Frame Analyzer can enable or disable each drawing call so a developer can test and measure without source modification. This feature is in the Blend State tab under the State tab.

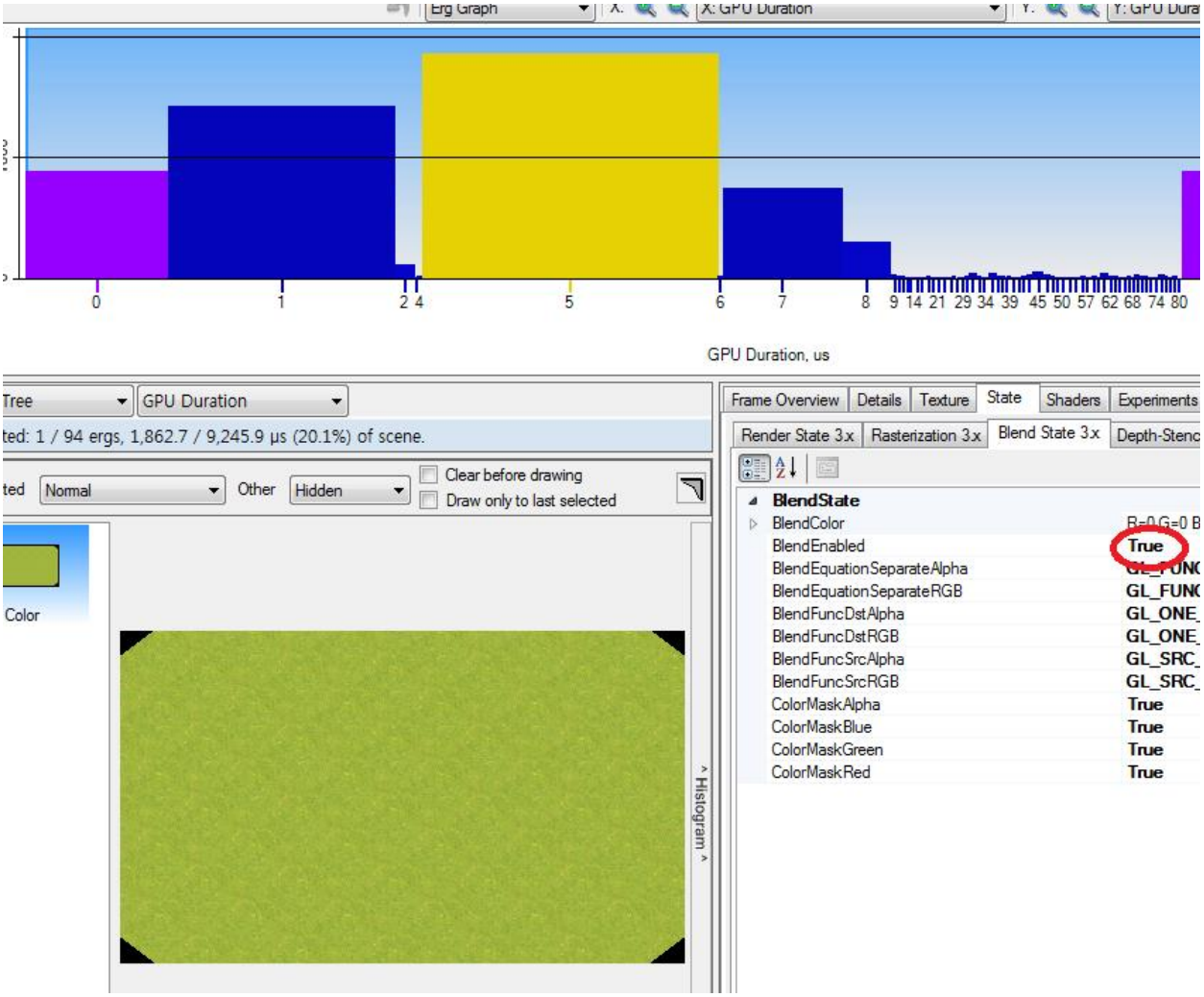

Figure 4. How to experiment Enable/Disable alpha blending on Graphics Frame Analyzer without source modification.

The table below shows more detailed information about drawing call of the grass after disabled alpha blending and the GPU Duration of the grass is decreased about 26.0%. Also notice that the GPU Memory Read is decreased about 97.2%.

	Baseline	Changed drawing order(sky)
GPU Clocks	1,466,843	1,085,794.5
GPU Duration, us	1,896.6 us	1,398.4 us
GPU Memory Read, MB	7.6 MB	0.2 MB
GPU Memory Write, MB	8.2 MB	8.2 MB

Table 4. Detailed information of drawing call after disabled alpha blending

Apply Z-culling efficiently

When an object is rendered by the 3D graphics card, the 3D data is changed into 2D data (x-y), and the Z-buffer, or depth buffer, is used to store the depth information (z coordinate) of each screen pixel. If two objects of the scene must be rendered in the same pixel, the GPU compares the two depths and overrides the current pixel if the new object is closer to the observer. The process of Z-culling reproduces the usual depth perception correctly by drawing the closest objects first so that a closer object hides a farther one. Z-culling provides performance improvement when rendering hidden surfaces.

Game has two kinds of terrain drawing: sky and grass drawing. The Erg 1 drawing call is for the sky and Erg 5 is the drawing call for the grass. Because large portions of sky are behind grass, lots of sky areas never show during the game. However, the sky was rendered earlier than the grass, which prevented efficient Z-culling.

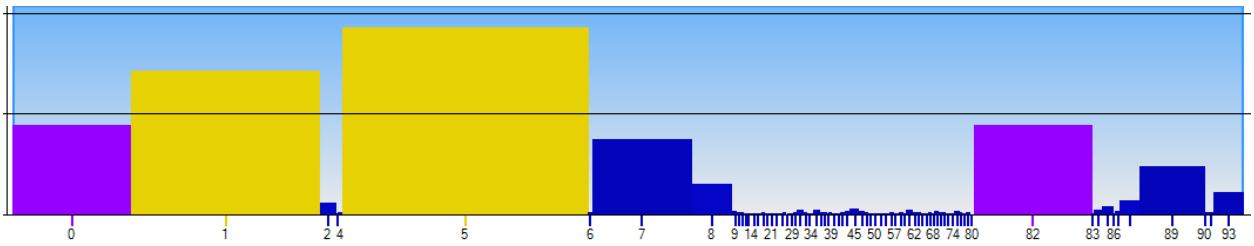

Figure 5. Drawing call for sky(erg 1) and grass(erg5)

Below is the GPU duration of the sky after changed the drawing order.

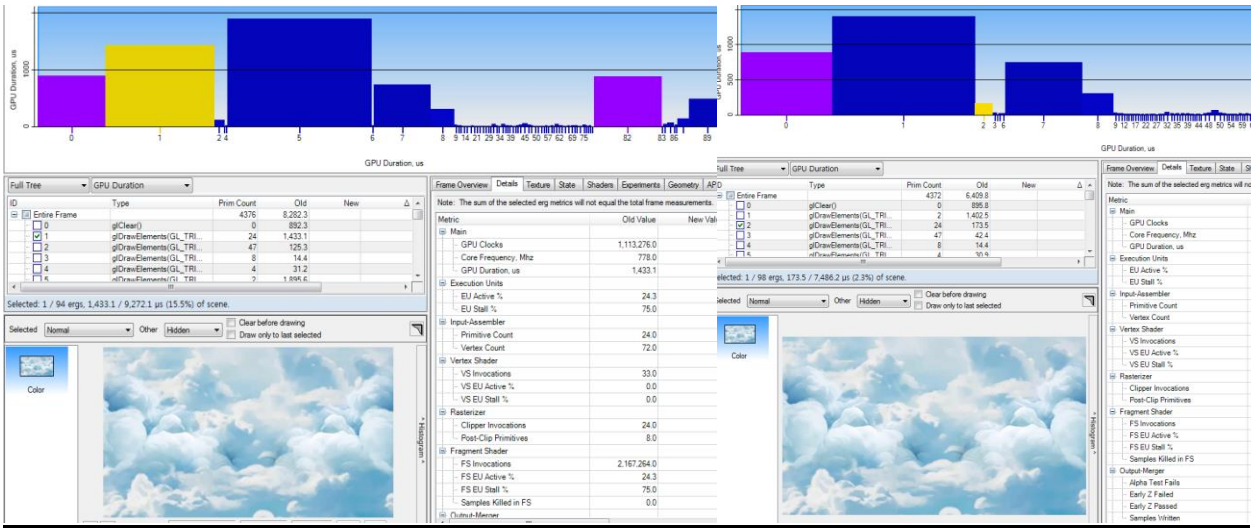

Figure 6. Result after changing the drawing order of sky on Graphics Frame Analyzer.

The table below shows more detailed information about the sky after changing the drawing order, and the GPU Duration of grass is decreased about 88.0%. Notice how the GPU Memory Write is decreased about 98.9%.

	Baseline	Changed drawing order(sky)
GPU Clocks	1,113,276	133,975
GPU Duration, us	1,433 us	174.2 us
Early Z Failed	0	2,145,344
Sample Written	2,165,760	20,416

GPU Memory Read, MB	0.2 MB	0.0 MB
GPU Memory Write, MB	9.4 MB	0.1 MB

Table 5. Detailed information of drawing call after changed drawing order(sky)

Results

The next table shows the more detailed data of x86 optimization after removing unneeded alpha blending and changing the drawing order. GPU Duration is decreased about 25% and GPU Memory Read/Write is decreased about 42.6% and 30.0%, respectively. System Analyzer showed the FPS only increased 1.06 because Android uses vsync mode and max FPS is 60 fps, but the FPS on Graphics Frame Analyzer increased about 29.7%.

	X86 Baseline	X86 optimized
GPU Clocks	6,654,210	4,965,478
GPU Duration, us	8,565.2 us	6,386 us
Early Z Failed	16,592	2,248,450
Sample Written	6,053,311	2,813,997
GPU Memory Read, MB	20.9 MB	12.0 MB
GPU Memory Write, MB	28.6 MB	20.0 MB
FPS on System Analyzer	59.01	60.07
FPS on Graphics Frame Analyzer	120.9	156.8

Table 6. Performance gains after disable alpha blending and changing drawing order(sky)

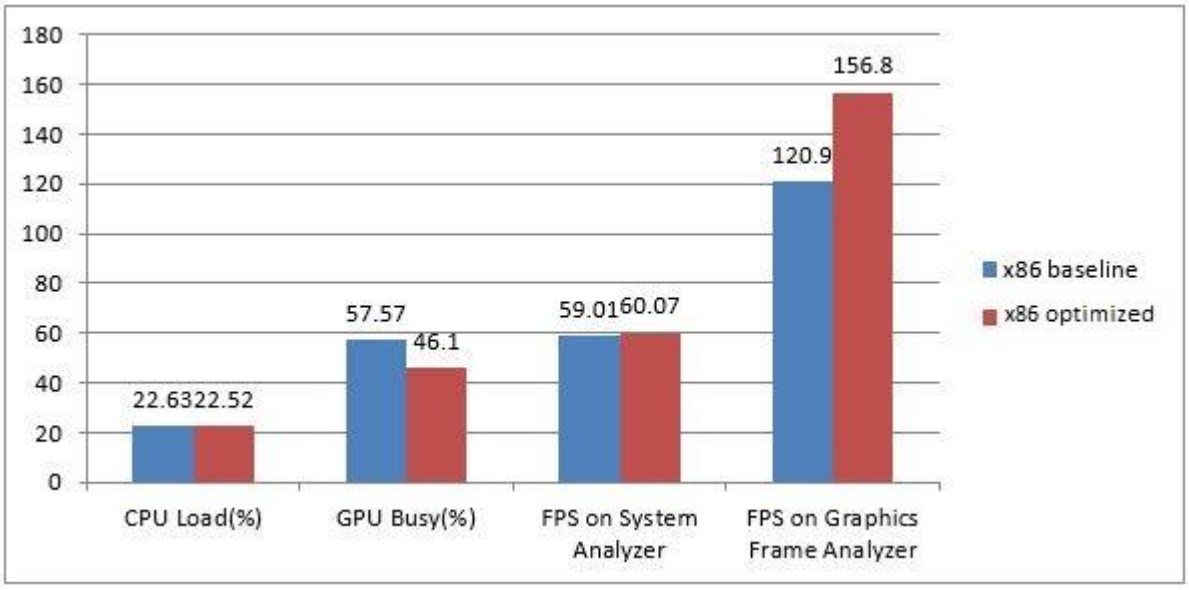

Figure 7. Performance gains after optimized x86 native support

Conclusion

When you start to optimize a game on Android x86, first developers should port their games for Android x86 and next determine where the application bottleneck is. Profiling tools can help you measure performance and see more easily where performance issues are on the GPU side. Intel GPA's powerful analytic tools can provide the ability to experiment without any source modification.

About the Authors

Jackie Lee is an Applications Engineer with Intel's Software Solutions Group, focused on performance tuning of applications on Intel Atom platforms. Prior to Intel, Jackie Lee worked

at LG in the electronics CTO department. He received his MS and BS in Computer Science and Engineering from Chung Ang University.

References

Intel® Graphics Performance Analyzers
https://software.intel.com/en-us/gpa

Innospark
http://www.innospark.com/#!home-en/c1vtc

Hero Sky: Epic Guild Wars
https://play.google.com/store/apps/details?id=com.innospark.herosky

Unity
http://unity3d.com

Unity* Native X86 Support Shines for Square Enix's Hitman GO*
https://software.intel.com/en-us/articles/unity-native-x86-support-shines-for-square-enix-s-hitman-go

Alpha Blending
http://help.adobe.com/en_US/as3/mobile/WS4bebcd66a74275c36c11f3d612431904db9-7ffe.html

Notices