Case Study: Optimizing King of Soldier* with Intel® Graphics Performance Analyzers on Intel® HD Graphics 4000

Intel Corporation: Cage Lu, Kiefer Kuah   Giant Interactive Group, Inc.: Yu Nana

Abstract

The performance of King of Soldier* (兵王) was improved by 1.75x on a 3rd generation Intel® Core™ processor PC, from 12 frames per second (fps) to 21 fps in a graphics intensive scene by optimizing bottlenecks found in the game. We used Intel® Graphics Performance Analyzers (Intel® GPA) to find the hotspots and run state override experiments to quickly assess potential performance gains. Optimization opportunities were found in drawing shadows, effects from firing weapons and casting spells, and characters. Optimizing these led to fewer draw calls and fewer Direct3D* state changes, which resulted in 1.75x frame rate improvement.

Introduction

King of Soldier is a massively multiplayer online role playing game (MMORPG) set in the future. It was developed by Shanghai JuXian Co., a subsidiary of Giant Interactive, a major game publisher in China. It uses a proprietary 3D game engine featuring advanced visual effects. Intel and Giant worked together to optimize the game on Intel® HD 4000 Graphics. Systems with Intel HD Graphics constitute a growing market share of graphics processors. Properly tuned games will run well on these graphics chips. To help game developers optimize their game, Intel developed a set of performance analysis tools called Intel GPA. They are freely available and easy to use. With Intel GPA, game developers can quickly find bottlenecks in their game and run experiments to assess potential performance benefits. Intel GPA are available at http://www.intel.com/software/GPA. They are the main tools we used in optimizing King of Soldier.

This analysis and optimization were done on a PC with the 3rd gen Intel Core i7 processor (for details on the platform, see Appendix A), which has Intel HD 4000 Graphics. Intel HD Graphics 4000 launched in 2012, and its performance is comparable to mainstream discrete graphics cards.

Giant was interested in the game’s performance when there were dozens of players in combat. So we chose the game scene shown in Figure 1 for performance analysis and optimization. The scene had about 100 players in combat. They were fighting with weapons and magic spells, filling the scene with multiple effects from firing weapons and casting spells. The initial frame rate was 12 fps. Our goal was to increase the frame rate to >15 fps without significantly reducing visual quality.
Figure 1. The scene in King of Soldier used in performance tuning. It has about 100 players and runs at 12 fps.

System level analysis

The Intel® GPA System Analyzer Heads Up Display (HUD) provides a quick analysis of the game performance using CPU, Direct3D, and GPU metrics. It also supports real-time Direct3D pipeline state overrides, which are helpful in pinpointing the performance bottlenecks. We got the following results by using System Analyzer HUD:

1. The frame rate was 12 fps. Applying the Null Hardware override, the frame rate reached 15 fps, which meant that the game could only reach 15 fps even with infinitely fast graphics hardware. This implied that although there were bottlenecks in the GPU, there were more significant bottlenecks elsewhere. With the Disable Draw Call override, which simulates an infinitely fast driver and GPU, the frame rate reached 32 fps. As the driver runs on the CPU, we deduced that there were some bottlenecks on the CPU.

2. The number of draw calls per frame was ~7800, and the number of state changes per frame was ~29k. Both were very high for a scene of this complexity. Draw calls and state changes add a performance cost to the application, the D3D runtime, and the graphics driver. These likely contributed to the CPU bottlenecks we found in item 1.
Results from System Analyzer suggest that we could speed up the frame rate by reducing the number of draw calls and state changes. This reduction could come from eliminating redundant state changes or removing draw calls that had little visual effect. To figure out what these draw calls and state changes were doing, we turned to the Intel GPA® Frame Analyzer.

**Frame Analysis**

The Intel GPA Frame Analyzer can give a more in-depth analysis of the performance within a frame. We captured a frame of our target scene and opened it in Frame Analyzer as shown in Figure 2. The chart labeled A in Figure 2 is a plot of the time taken by the GPU to draw the various elements in the scene. Most of these correspond to Direct3D draw calls. The X and Y axes can be customized to change how the graph is displayed. In Figure 2, we set GPU Duration on the X axis and GPU Breakdown on the Y axis. The draw calls are also displayed in the list in Frame Analyzer, as shown by the table labeled B. Frame Analyzer also displays the content of the render target. The render target labeled C shows the scene captured. The area labeled D has a series of tabs containing performance data, textures, shaders, states, API logs, etc.

*Figure 2. A captured frame from King of Soldier in Frame Analyzer before optimizations.*

System Analyzer HUD indicated that the game had a high number of draw calls and state changes, which likely contributed to the CPU bottlenecks. From the Frame Analyzer, we obtained a breakdown of the draw calls and the time consumed in the frame and found that:
1. Generating shadow maps incurred \(~1198\) draw calls and \(14.5\%\) of the frame time. Through the Texture tab of Frame Analyzer, we found the game calculated shadow maps for effects from firing weapons and casting spells. The selected textures in Figure 3 are effect textures. Calculating shadows for effects is expensive with little visual impact. We can remove these calls without significantly changing the visuals.

2. Drawing other objects in the scene incurred \(5526\) draw calls and \(72.6\%\) of the frame time. More than \(4100\) of the draw calls were for rendering effects from weapons and magic spells and cost \(48.7\%\) of the frame time. This was very costly and was our main target for performance optimization. Figure 4 highlights these draw calls.

Figure 3. Shadow map generation. The game calculates shadows for effects that have little visual impact.
Figure 4. These effects incurred more than 4100 draw calls and 48.7% of the frame time.

Although most of the draw calls each consumed a small amount of GPU time, they added up to 48.7% of the GPU time. Zooming into these draw calls showed an inefficiency that can be optimized. Figure 5 shows the effect rendered by 10 neighboring draw calls. All 10 of the draw calls use the same 128x128 textures. We can merge these draw calls and textures to reduce the number of draw calls and state changes. Another set of objects in the scene, the characters, were also found to be drawn by multiple draw calls. In the same way, we can merge the draw calls for the characters.
Optimization Strategy and Results

1. Shadow Map Generation

Generating shadow maps for weapons and spells effects is unnecessary. We removed them with minimal impact to the visual quality. The number of draw calls for shadow map generation was reduced from 1198 to 860, and the time consumed was reduced from 6.86 ms to 5.58 ms.

2. Merge draw calls and small textures

According to the frame analysis above, the weapon and magic spell effects were the most expensive part of rendering the frame. We merged the draw calls and the textures for these effects as shown in Figure 6 and Figure 7. The number of draw calls was reduced from 37 to 1, and the number of textures was reduced from 4 to 1.
Figure 6. Before optimization, it took 37 draw calls and 4 textures to render the effect.

Figure 7. The draw calls and textures were merged to use one draw call and one texture to render the effect.

After applying this optimization to the whole frame, the number of draw calls for the effects was reduced from ~4100 to ~700. The number of SETTEXTURE API calls was reduced from 3764 to 2029. The GPU time of rendering these effects was reduced from 38.7 ms to 16.1 ms, a significant improvement. Figure 8 shows the results after optimization.
Figure 8. After optimization, the number of draw calls for the effects is ~700, and GPU time was reduced from 38.7 ms to 16.1 ms.

3. Merge draw calls for characters

Drawing the various characters in the scene could be optimized by merging their buffers and draw calls. Figure 9 shows that it took 3 draw calls to render this character and these 3 draw calls used the same texture. We merged the index buffers and the vertex buffers and used 1 draw call to render it, as shown in Figure 10. This saved ~200 draw calls in whole frame. We did the same optimization on player characters and saved an additional ~200 draw calls.
Figure 9. Before optimization, it took 3 draw calls to render this character.

Figure 10. After optimization, it takes 1 draw call to render the same character.

4. Remove unnecessary SETTEXTURE API calls

Reducing state changes could reduce time the CPU spends on Direct3D runtime and driver. In another game scene, we found that the game set the textures in every draw call, even when the textures did not change. Figure 11 shows 2 neighboring draw calls rendering the terrain and each draw call used SETTEXTURE 5 times to set textures. The textures did not change between the 2 draw calls and did not need to be set for the second draw call. We eliminated the redundant SETTEXTURE calls as shown in Figure 12. Optimizing this reduced the number of SETTEXTURE API calls from 1837 to 554.
Figure 11. Two neighboring draw calls for the terrain that use the same set of textures.

Figure 12. After optimization, redundant SETTEXTURE calls were removed.
Summary

With the above optimizations, the number of draw calls per frame was reduced from ~7800 to ~3800, and the number of state changes was reduced from 29k to 14k. Frame rate improved from 12 fps to 21 fps, which is a 1.75x gain in performance and more than meets Giant’s target of >15 fps. Table 1 is a summary of the optimizations done. The elapsed times shown in the table were GPU time as measured by Intel GPA. Optimizing draw calls and state changes not only improved performance of the computations done on the GPU, but also on the CPU. The CPU runs the game code, the Direct3D runtime, and the graphics driver. We measured the CPU time per frame and GPU time per frame to be 85.81 ms and 76.78 ms, respectively, before optimizations. After optimizations, they improved to 48.14 ms and 47.55 ms, respectively. The performance analysis and optimization of King of Soldier is an example of how Intel GPA can be used to improve performance in many games.

Table 1. Summary of items optimized.

<table>
<thead>
<tr>
<th>Optimization items</th>
<th>Optimization method</th>
<th>Before optimization</th>
<th>After optimization</th>
<th>Improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Shadow map generation</td>
<td>Remove shadows for weapons and magic spells effects</td>
<td>6.86 ms</td>
<td>5.58 ms</td>
<td>1.28 ms</td>
</tr>
<tr>
<td>Effects</td>
<td>Merge draw calls and textures</td>
<td>38.7 ms</td>
<td>16.1 ms</td>
<td>22.6 ms</td>
</tr>
<tr>
<td>Characters</td>
<td>Merge draw calls</td>
<td></td>
<td>Save ~400 draw calls</td>
<td>3.8 ms</td>
</tr>
<tr>
<td>SETTEXTURE</td>
<td>Remove redundant SETTEXTURE</td>
<td></td>
<td>Save ~1300 SETTEXTURES</td>
<td>~2.0 ms</td>
</tr>
</tbody>
</table>

About the Authors

Cage Lu is an Application Engineer at Intel. He has been working with big gaming ISVs in China for several years to help them optimize game client performance on Intel platforms.

Nana Yu is the technical manager of Giant. She participated in the development of “仙剑奇侠传-问情篇”, “Giant,” and “The King of Soldier” games. She is currently focusing on the research and development of a 3D game engine.

Kiefer Kuah is a software engineer at Intel. He optimizes games on Intel platforms.

Appendix A:

3rd Generation Intel® Core™ Processor, code-named Ivy Bridge, Platform Configuration
CPU: Intel Core i7 processor @ 2.30 GHz, with HD 4000 graphics
Memory: 8 GB DDR3 1333 Mhz
HD: Intel 160 G SSD
OS: Windows 7 Professional, 64 bit
Graphics Driver: 8.15.10.2618
Notices

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED NOR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

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.

Any software source code reprinted in this document is furnished under a software license and may only be used or copied in accordance with the terms of that license.

Intel, Core, and the Intel logo are trademarks of Intel Corporation in the US and/or other countries.

Copyright © 2012 Intel Corporation. All rights reserved.

*Other names and brands may be claimed as the property of others.