Optimizing Puzzle Touch (Casual Game)

Introduction

Puzzle Touch, developed by Greenfield Technologies*, rose swiftly to become one of the top 10 games in the Windows* Marketplace. It is a highly interactive puzzle game where the gamer uses touch or a mouse to move and rotate puzzle pieces into their proper place. The difficulty can be modified by choosing the number of pieces that a puzzle will be broken into, or playing with a moving image (video puzzle). Touch capability was included in the initial release by the developer, and consumers loved the concept.

Three main issues arose during our analysis. First, we noticed that the game loaded slowly. Then, the game lagged in certain spots. Finally, we discovered that using the touch interface consumed more power than when the gamer used a mouse, and there was additional unnecessary power consumption when the game idled. With these issues defined, we got to work and used publicly available tools to optimize the power and responsiveness of the application.

Intel collaborated with Greenfield Technologies to improve the performance and power usage of Puzzle Touch on Ultrabook™ PCs. Prior to contacting Greenfield Technologies, we looked for ways to improve rendering speed and power usage. After analyzing the performance of the game using the Microsoft* Performance Analyzer (WPA) and other tools, we realized there were issues that we could help resolve.
We tested the game on a 3rd Generation Intel® Core™ i5 Mobile Processor, and used Intel® Power Gadget and Battery Life Analyzer to determine the power usage in the game.

**Optimizing the Game**

**Shortening Load Times**
Initially, there was a prolonged load time for Puzzle Touch, which troubled us. We selected a 36-piece puzzle to test the launch time, and with this puzzle, the app took 9.2 seconds to launch.

![Figure 1: Windows Performance Analyzer shows CPU activity spikes up to 40% during 9.2 seconds loading time](image)

Since this time seemed excessive, we analyzed the startup of the app with the Windows Performance Analyzer (WPA) tool. Plotting the 36 piece puzzle with animation on screen that uses graphics for rendering (the game uses XAML/DX framework) should have low CPU utilization (~10-15%) and should take less than four seconds on 3rd Generation ULV Processor. Comparing the performance metrics with the game launch time, we discovered launch optimizations by examining the CPU overtime in xPerf. WPA is useful for showing CPU usage over time, analyzing the call stack, and performing periodic/wait analysis. Figure 1 shows “% CPU Usage” graph for the game launch time. The X-axis represents CPU Utilization in percent, while the Y-axis represents time in seconds. In this case, it shows a series of spikes in CPU activity during the launch period. These spikes consistently reached 40% and higher during the game launch.

After looking through the CPU utilization and loading time, we examined ways to optimize the launch time and lower the resource utilization. We analyzed the game launch application and provided feedback to Greenfield Technologies.

**Optimizing Launch Time**
When a player opens a game, the app places puzzle pieces randomly on the screen. Initially, there was a delay in the placement, so we looked for the cause.
We used the WPA to find the time spent in threads, and the Windows Performance Recording (WPR) to collect the data. We checked CPU Usage and GPU Activity to perform the analysis. We started the collection and switched to the app. As the game load time was part of our study, we opened the “Difficult Level” of the game. Once all puzzle pieces were placed, we switched to WPA and stopped the collection.

![Figure 2: User interface for Windows Performance Recorder](image)

Next, we began our analysis. We used WPA to open the trace file and analyzed the loading time issues.
**Figure 3: The explorer in WPA for the opened trace file.**

To begin the analysis, we dragged and dropped CPU Usage (Precise) to the explorer. Figure 4 shows the overtime view of loading the game. The X-axis denotes time in seconds. The Y-axis shows CPU Utilization. Multiple threads were spawned by the process, as seen by opening the thread stack. We arranged the columns as shown in Figure 4. ReadThreadStack shows the stack of the readying thread, Ready(us) denotes the difference in SwitchInTime(s) and ReadyTime(s). SwitchInTime(s) is defined by the time when the new thread was readied. Wait(s) is defined as difference in ReadTime(s) minus LastSwitchOuttime(s).
Next, we performed a wait analysis and critical path analysis on different threads, which provided insight on which thread took more time. The critical path is the longest path through the set of operations. Reducing the length of critical path directly reduces the time taken for the activity. An activity is a set of operations that flow from the start to the end event. To find critical path in WPA we sorted the column: “CPU Usage, Wait and Ready”.

**Figure 4: Windows Performance Analyzer Overtime view**

**Figure 5: Call Graph for Wait Analysis**
In Figure 6, sorting by CPU Usage (ms) shows a top child row of 1731 milliseconds. Sorting by Ready (us) [Sum] shows a top child row of 67ms. Sorting by Waits (us) [Sum] shows a top child row of 18694 milliseconds, and a second row of 4902 milliseconds. These rows had a dominant and significant contribution, so were the next to be investigated.

Opening the stacks, we saw the calls made by the thread for Ready/Waits and CPU Usage. Figure 7 shows different calls made by the application that caused performance issues. BufferStreamFlush waited for 18msec while accessing the thumbnails for every puzzle.
Another call stack showed critical path where ThreadID 4016 waited for ThreadID 2972 to finish for 11msec before context switched out. Optimizing the interaction between these two threads can gain 11msec for every puzzle drawn on the screen.

Figure 7: Call stack showing Storage Access

Figure 8: Call Stack showing Wait Analysis
Other stacks we observed were related to Semaphore and tick activities. We provided the feedback of our analysis to Greenfield Technologies, and were able to optimize the app’s loading time.

![Figure 9: Call Stack showing top issues identified with Windows Performance Analyzer](image)

Finding the critical path is very important for performance and user experience optimization. The use of an efficient API can boost the game’s performance and save power consumption. Optimizing the calls for loading resulted in a significant performance gain. For a consistent and smooth frame presentation, at a frame rate of 60 frames per second, each frame must complete the presentation in 16 ms (vblank-to-vblank). Out of this time, the CPU budget is 4 ms and the GPU budget is 8 ms. Since much of the work of setting up a frame and rendering it is serialized, this leaves 4 ms of head-room to deal with activities such as cache misses and other core interferences that may prevent achieving a consistent frame rate. Removing the dependency between threads helped to reduce the game loading time from 9.2 seconds to 7.9 seconds.

![Figure 10: Overtime view after Optimization](image)

**Controlling the Power**

When using the touch interface while playing the game, we noticed a significant impact on power consumption as analyzed by the Intel Battery Life Analyzer. Figure 11 shows the mouse vs. touch power impact. The Y-axis is power in watts, while the x-axis is timeline in seconds. During first ~110 seconds the application was running at idle and after that, the graph shows a scenario of playing the game with
touch and mouse. Intel Battery Life Analyzer at idle shows significant wakeup causing C0-state package residency at 9.74% (goal is ~95%).

![Graph showing package power over baseline](image)

**Figure 11:** *Once active input begins, power usage stays consistently high when using touch*

To analyze the cause of the wakeup, we used WPA to analyze the trace. We opened the CPU Usage (Precise) to find the call stack of the calls.

The call stack in Figure 12 shows use of DirectX and XAML render for full surface every monitor refresh rate. Use of Virtual SIS when there is no user input after some time duration can have significant saving at idle. Removing the extra render at idle and decreasing the fps when no user input was present created significant power savings.
Figure 12: Call stack showing Periodic Activity

Result: Significant saving due to change in software architecture.

<table>
<thead>
<tr>
<th>Before Optimization</th>
<th>Touch Playing (%)</th>
<th>After Optimization</th>
<th>Touch Playing (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Package C0-C1</td>
<td>9.74</td>
<td>Package C0-C1</td>
<td>3.74</td>
</tr>
<tr>
<td>Package C2</td>
<td>1.823</td>
<td>Package C2</td>
<td>1.023</td>
</tr>
<tr>
<td>Package C3</td>
<td>0</td>
<td>Package C3</td>
<td>0</td>
</tr>
<tr>
<td>Package C6</td>
<td>1.13</td>
<td>Package C6</td>
<td>0.947</td>
</tr>
<tr>
<td>Package C7</td>
<td>87.31</td>
<td>Package C7</td>
<td>94.09</td>
</tr>
</tbody>
</table>
**Conclusion**

Power and performance are both important metrics for a positive user experience. Windows 8 Modern UI applications must be optimized for lower power and better responsiveness. Tools such as WPA and Intel Battery Life Analyzer are important for optimization of the user experience.

**References**


**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.

A “Mission Critical Application” is any application in which failure of the Intel Product could result, directly or indirectly, in personal injury or death. SHOULD YOU PURCHASE OR USE INTEL'S PRODUCTS FOR ANY SUCH MISSION CRITICAL APPLICATION, YOU SHALL INDEMNIFY AND HOLD INTEL AND ITS SUBSIDIARIES, SUBCONTRACTORS AND AFFILIATES, AND THE DIRECTORS, OFFICERS, AND EMPLOYEES OF EACH, HARMLESS AGAINST ALL CLAIMS COSTS, DAMAGES, AND EXPENSES AND REASONABLE ATTORNEYS’ FEES ARISING OUT OF, DIRECTLY OR INDIRECTLY, ANY CLAIM OF PRODUCT LIABILITY, PERSONAL INJURY, OR DEATH ARISING IN ANY WAY OUT OF SUCH MISSION CRITICAL APPLICATION, WHETHER OR NOT INTEL OR ITS SUBCONTRACTOR WAS NEGLIGENT IN THE DESIGN, MANUFACTURE, OR WARNING OF THE INTEL PRODUCT OR ANY OF ITS PARTS.

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](http://www.intel.com/design/literature.htm)

Intel, the Intel logo and Ultrabook are trademarks of Intel Corporation in the U.S. and other countries.
*Other names and brands may be claimed as the property of others

Copyright© 2013 Intel Corporation. All rights reserved.

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.

Optimization Notice