Writing Energy-Efficient Windows* Store Applications for Mobile Devices
Impact of Graphical Intensive Application on Processor Power

Outlines
1. Introduction ...........................................................................................................................................1
2. Programming Model ............................................................................................................................2
3. Present1 API Impact............................................................................................................................2
   3.2 UI frame rate and power comparison ...........................................................................................3
4. Event Driven Render vs. Render Loop ...............................................................................................4
   4.1 Rending/Presenting Implementation ............................................................................................4
   4.2 UI Thread Frame Rate and Power Comparison ...............................................................................4
5. Summary .............................................................................................................................................5
6. Future Discussion ................................................................................................................................5
7. About the Author .................................................................................................................................5
8. Notices ..............................................................................................................................................6
9. Optimization Notice ............................................................................................................................7
10. Performance Notice ............................................................................................................................7

1. Introduction

In the series of articles on the “Writing Energy-Efficient Windows* Store Applications for Mobile Devices”, we have discussed the timer usage impact on the processor
In this section, we will discuss the impact of graphical intensive applications on the processor power.

Windows* 8 enables the hardware acceleration of graphical intensive applications through Microsoft* DirectX* SDK if available. We first discuss the programming model by using DirectX* SDK. Then we discuss several Windows Runtime* API usage with an impact on the processor package power.

2. Programming Model

The programming model for graphical intensive applications includes two key steps – rendering and presenting. The render step is to fill the rendering objects in a swap chain and then set tasks for rendering pipelines. The present step is to display the rendered frame to screen. The two steps are done interactively.

Developers can implement possible power saving designs or tune parameters for energy efficiency. We can have hardware acceleration concurrency to improve energy efficiency. For example, the deferred rendering is supported in DirectX* 11 if hardware supported. The deferred rendering is designed to enable breaking up complex scenes into concurrent subtasks, which can be executed simultaneously. The performance of rendering can be improved. Also power could potentially be reduced because the GPU active time is reduced due to the concurrent scene subtask execution. Another easy option is to use the provided APIs with tunable parameters. These parameters can impact the render frame rate. We also introduce the event-driven rendering concept in some application scenarios.

3. Present1 API Impact

Here is a case study on the IDXGISwapChain1::Present1 API usage in Windows* Store apps. The IDXGISwapChain1 is the new data structure in DirectX11 for swap chains. To present the rendered frame to the display, generally Present1 API is utilized.

3.1 Present1 API

The API is declared as the following.

```c
HRESULT Present1(
    [in] UINT SyncInterval,
    [in] UINT Flags,
    [in] const DXGI_PRESENT_PARAMETERS *pPresentParameters
);
```

Sample Code 1. Present1 API in Windows Runtime **

The API takes three input parameters. The first parameter is the synchronized presentation interval of a frame. In general model, 0 means the presentation occurs immediately without any synchronization. 1,2,3,4 are the synchronize presentation after the nth vertical blank. The second and third parameters specify options for
presenting frames to the output and the frame information to be present. If tuning the first parameter, the API can change the rendering frame rate, which has a big impact on the processor power.

3.2 UI frame rate and power comparison

We ran experiments on an Ultrabook™ with a 3\textsuperscript{rd} generation Intel® Architecture Core™ processor. The processor package includes a graphical processor. Through Intel® Power Gadget tool, we can read the processor package power. The package power includes the power drained by the processor package. We compare the processor package power for different settings on the first parameter for \textit{Present1} API. The larger number leads to the lower rendering frame rate.

![Effects of Frame Rate on Processor Power](image)

**Figure 1: Effect of the rendering frame rate on the processor package power in a Windows® Store app**

The figure 1 plots the processor power impact of the \textit{Present1} API parameter settings. X axis is the rendering frame rate triggered by the parameter and y axis is the relative processor power. As we can see, both 0 and 1 are synched with display refresh rate 60 fps/second, which has identical power consumption. When the parameter setting is increased from 1 to 4, the fps is decreased and the processor package power is also decreased. Thus, we should be careful to choose the API parameter. It is better to choose the one to satisfy the user experience need of the app and also maintain a low power cost.
4. Event Driven Render vs. Render Loop

There is a kind of mobile casual gaming apps like painting, drawing or puzzle apps. The animated image on the screen changes only when user touches it. This means the UI updates are event driven, either by touch event or by other gesture events. The animated image updates usually include the steps to render and present an image.

4.1 Rending/Presenting Implementation

Windows runtime allows developers programming the image rendering and presenting in two ways. The example below is the user-event driven rendering (run render once and present once). When user touches screen, render once. The frame rate is low.

```
Run{
    render();
    present();
}
OnTouched()
{
    render();
    present();
}
```

**Sample Code 2. Infrastructure of event driven rendering implementation**

Another option is shown in the sample code below. A while loop is utilized to continuously render and present. When a touch happens, the UI image is updated accordingly by a flag called in render(). The rendering frame rate is synchronized with the display refresh rate, which is 60 Hz in most display. 60 Hz is fast enough to let the user feel the update is driven by the touch action.

```
Run{
    While(!m_windowClosed)
    {
        render();
        present();
    }
}
```

**Sample Code 3. Infrastructure of loop rendering implementation**

4.2 UI Thread Frame Rate and Power Comparison

We coded the two implementations for the same feature as a 2D animation text enabled by a touch action in a Windows* Store app. Figure 2 shows the average processor power comparison between two implementations for an identical UI update feature. The event-driven rendering implementation consumes much less power than the render loop one. The render loop implementation consumes 1.45 times the processor power than the event-driven one. This is because the render loop implementation causes the UI rendering frame rate to be the same as the maximum
display refresh rate (60Hz). The event-driven implementation has near zero rendering frame rates unless the user touches the screen.

The comparison shows whenever possible we should utilize the event-driven implementation for event-driven gaming apps. Thus less UI update caused by the implementation can lead to a big power saving on the processor package power.

5. Summary

In the article we explain the available Windows* Runtime API usage and introduce the event-driven rendering implementation for energy efficiency. We expect developers can apply this guideline in the software development cycle.

6. Future Discussion

As mentioned in the series of the articles, we will discuss the platform power impact of other hardware components in future articles in this series. We will address how to save power in other components for various types of Windows* Store apps like file-access based apps, network I/O-based apps and sensor usages.

7. About the Author

Sushu Zhang is a software engineer in Intel’s Software and Services Group. She is working on ISV scale enabling for Intel-based platforms with Android* / Windows OS. She developed the Intel® Power Monitoring Tool for Android Devices. Most recently, Sushu has been involved with several energy-efficiency projects at Intel. Prior to
joining Intel, Sushu worked at Microsoft on Windows energy efficiency. Sushu earned a Ph.D in Computer Science at the Arizona State University. Her research area was system-level power and thermal management.

8. 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 AFFILATES, 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

Intel, the Intel logo, VTune, Cilk and Xeon 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© 2012 Intel Corporation. All rights reserved.

**This sample source code is released under the Intel OBL Sample Source Code License (MS-LPL Compatible) and Microsoft Limited Public License. Copyright© 2012 Intel Corporation. All rights reserved.

9. Optimization Notice

<table>
<thead>
<tr>
<th>Optimization Notice</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel.</td>
</tr>
<tr>
<td>Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.</td>
</tr>
<tr>
<td>Notice revision #20110804</td>
</tr>
</tbody>
</table>

10. Performance Notice

For more complete information about performance and benchmark results, visit www.intel.com/benchmarks