Levering GPGPU and OpenCL Technologies for Natural User Interfaces

A You i Labs White Paper

Rev. 1.0

First Published February 2012

By: Stuart A. Russell

Chief Technology Officer

You i Labs Inc.

303 Terry Fox Drive, Suite 340, Ottawa, ON, Canada, K2K 3J1

Abstract

Natural User Interfaces (NUIs) are vastly more complex than traditional graphical user interfaces and require large computational power to provide an immersive experience for the user. We will examine NUI improvements and design challenges over traditional Graphical User Interfaces and the associated computational complexities encountered during implementation. In order to maximize the hardware's capability for a NUI, making use of available Graphics Processing Unit (GPU) cycles to complement the Central Processing Unit (CPU) is crucial. This process of using GPUs as General Purpose Graphics Processing Units (GPGPU) has been traditionally limited to desktop computing platforms, but as portable devices are becoming more powerful, encompassing multiple-core CPU and GPU elements, implementation becomes crucial for efficient use of the hardware's capability. When effectively utilizing both the GPU and CPU cycles a smooth, fluid experience can be maintained, as well as optimization for best case power consumption. This can be very challenging due to platform constraints, differing CPU / GPU architectures, implementation complexity and cost of integration. We will examine a few aspects of a NUI that could benefit from GPGPU computing and the associated implementation benefits.

GPGPU

"GPGPU stands for General-Purpose computation on Graphics Processing Units. Graphics Processing Units (GPUs) are high-performance many-core processors that can be used to accelerate a wide range of applications."

-http://gpgpu.org/

NUI

"A natural user interface is a user interface designed to use natural human behaviors for interacting directly with content"

-http://nui.joshland.org

"A NUI relies on a user being able to carry out relatively natural motions, movements or gestures that they quickly discover control the computer application or manipulate the digital content."

-http://nuigroup.com

"We started with Command-line Interfaces in the 60s, and then moved to GUIs in the 80s, and we have just entered into the next 25 year evolution - the era of the NUI"

- Jason Flick, CEO, You i Labs.

Utilizing GPGPU for Natural User Interfaces

This paper will cover areas where General Purpose Graphics Processing Unit (GPGPU) functionality could be implemented in a NUI engine to best utilize the available hardware for efficiency and improved user experience. Both emerging trends and common optimization areas will be discussed. Optimizations particular to the You I Labs uSwish framework will also be reviewed. Although the main focus will be on mobile devices, the techniques discussed can easily be applied to other computing platforms. It is assumed that the reader is knowledgeable on the basic concept of heterogeneous and parallel computing. The actual software implementations using OpenCL or OpenGL shaders is outside the scope of this paper and will not be discussed.

NUI Background

In traditional graphical user interfaces (GUI) most of the interactive elements and controllable parts in the interface are static until acted upon. We are all used to this interaction when using a traditional desktop computer environment. The mouse-and-keyboard driven interface forces the user to navigate within the computer's environment for interaction and selection. Most of the information or content is provided to the user in separate applications that are simply launched from the GUI. As modern devices migrate to using touch screens, gesture, and other input methods the user interface must now instead navigate to the user's requests – static icons are now scrollable to navigate virtual screen space and must react in a natural way to provide appropriate feedback. In Natural User Interfaces (NUI) the user should experience interactions with the device that are natural and responsive. Actions such as sliding a finger

across the display must be accurately translated into user requests for motion (list navigation), selection (initiate an event or action), or other potential interpretations. Once the action has been decided, the natural flow of response must happen:

1: Real world physics style response

2: Immediate feedback to the user as acknowledgement of the request

3. Transitions that help to keep things in context

4: Preparation for next user interaction with minimal or no delay

Performing these tasks on modern interfaces can be quite costly in terms of CPU bandwidth. Consider a list style control of recently received emails. In a NUI environment the user would interact directly with the list itself and their motions would be interpreted, such as sliding gestures to navigate the list. The interaction experience is natural and pleasing to the user. This is a crucial difference from traditional GUIs where a separate scrollbar, usually to the far side of the list, would need to be manipulated by the user in an unnatural opposite direction of the list motion. To ensure a natural user response, the NUI would calculate the physical acceleration implied by the user's motion and provide a real-time update of the displayed email content as the list scrolls to a natural feeling friction stop. The update process must achieve, at the very least, 24 updates per second to appear natural and fluid. Modern NUIs normally target 60 frames per second (FPS) for optimal user response. While performing these updates of both content and visual components, the NUI must also continuously update the physical properties in response to other potential inputs from the user *without any discernible delay*. This results in a natural feedback response to the user.

Due to the complexity in providing this fluid flow of content many NUI devices incorporate Graphics Processing Units (GPU) to provide high performance rendering freeing the CPU to handle the user input, content control, and other aspects of the NUI environment. However, in most cases the GPU is utilized only for visual performance enhancement and relatively under-utilized in the normal NUI environment. Many of the available mobile GPUs are capable of very high performance polygon rendering that applies directly to 3D video games whereby hundreds of thousands of polygons would be rendered at high frame rates. When applied to current NUI environments which incorporate perhaps only hundreds of polygons, the GPUs are typically idle while the CPU is in constant use performing background system tasks. Due to the parallel nature of GPU design, making use of their computing power for other tasks in a NUI environment is an optimum application for GPGPU enhancement.

It is very important to understand that the demands of a NUI expand beyond just user inputs and feedback to the handling of information and content, which should flow naturally to the user. More information is being provided within the context of the user interface directly rather than through auxiliary programs, requiring more complexity and interactivity between the interface and user. In traditional GUI design, content for applications exists in silos that need only be accessed and updated when in direct user focus. A common example is an email application where the content is only retrieved and presented in the context of the application itself. Before running the application, the content contained may be out of date, incorrect or simply inaccessible to the user. Having seamless content flow is crucial in creating a natural interaction experience. This adds tremendous complexity as applications must freely pass, update and maintain content information in the background of the NUI at all times.

GPGPU considerations in NUI applications

Unlike traditional CPU architecture, GPU design incorporates a Single Instruction Multiple Data (SIMD) style architecture which works exceptionally well with large data sets using identical computations. Applying this technology for tasks in a NUI environment on mobile devices has the following challenges:

1. The data sets for common NUI operations are relatively small
2. Visual effects are still the largest demand on the GPU
3. Complexity of some calculations is not feasible or efficient on GPU architecture
4. Lack of access to functional hardware components either through open standards such as OpenCL or proprietary APIs
5. Physical limitations in the device platform, such as OpenGL 2.0 compliant shader support and Frame Buffer Object (FBO) support with extensions applicable to GPGPU style implementation for creating computational blocks

It is also important to note that many other uses for GPGPU functionality are outside the scope of the actual NUI itself:

1. Computer vision or OpenCV based intelligent image processing
2. Gesture tracking and interpretation
3. Live content decoding or parsing, such as video or audio content streams, especially for new formats not currently supported directly by the device hardware

4. External device control or data processing, such as video camera image data compression or conversion

We need to consider multiple factors for utilizing GPGPU computing effectively within a NUI environment. We must also share the resources with the normal graphics computations as well as potential GPGPU gains realized by external elements such as computer vision or gesture tracking. In addition consider the CPU and GPU bandwidth and requirements by other applications or external elements (such as networking). Simply stated, we cannot arbitrarily throw computation segments at the GPU and expect improved performance. *Efficient* use of the GPU for computational support in parallel with CPU based computation to obtain the best balance between overall performance and availability for other elements is the key to success. Mobile applications should also consider power consumption and the associated costs of bringing the GPU out of standby (if supported) versus dynamic CPU manipulation for power control. Many modern mobile CPUs support dynamic frequency control for reducing power consumption when idle. We need to consider the consumption differences between pulling the CPU to full rate to process data versus the equivalent GPU power costs for the same operations. The GPU will prove less efficient than the CPU in certain circumstances such as very limited data sets where the overhead costs of preparing the hardware exceed the computational time on the CPU. Managing the best power balance for computational selection is a complex part of utilizing GPGPU for NUI enhancement (this limitation and potential solutions will be addressed in detail in a future white paper). This lends itself to increased code complexity as both CPU and GPU based versions of each supported algorithm must exist and the choice must be made dynamically based on content as to which resource is utilized. This issue will be minimized in

the future through proper OpenCL implementations where the computational device being used is abstracted from the algorithm code, allowing for shared code between the architectures. We must also consider memory sharing and potential for blocking operations between processors. Utilizing multi-core CPU as well as GPU devices adds to this complexity since the architecture of the devices must be understood to make optimum use of the computing power while addressing limitations such as shared resources.

Proposed uses for GPGPU in NUI applications

Several areas of NUI design are directly suited to implementation in GPGPU form. Our "Top 10", in no particular order, are:

1. Visual features not traditionally handled by the GPU, such as font rendering
2. Layout of NUI elements and content frames
3. Enhanced Visual features
4. Sorting of data or other data handling services
5. Compression / decompression of content stores
6. Physical properties of objects – reactive physics effects processing
7. Complex animation or motion schemes for NUI visuals to appear more natural
8. Handling streaming NUI content
9. Real time security encryption / decryption
10. You I Labs uSwish NUI engine specific features

Let's consider each of these potential applications:

Visual features not traditionally handled by the GPU

Making use of available GPU cycles for CPU tedious tasks such as font rendering or layout calculations provides a good starting point for NUI GPGPU optimization. Early mobile devices used bitmap based fonts to remove the complexity and performance cost of dealing with vector font data. This solution had quality and expandability limitations that were reasonably acceptable at the time, but with modern high resolution devices and extensive available content a better solution is required. Modern devices support desktop style vector fonts allowing more compatibility with content and better overall readability for the user, but at a cost of more CPU cycles. This cost is usually offset with caching schemes and other intelligent buffering, but these in turn have their own limitations and complexities of implementation. If one considers the current cycle of font rendering which requires conversion from the original glyph data into raster bitmap information for display based on font face, style and size, followed by final per-pixel rendering, it is obvious there is potential for improvements. Although some support for GPU accelerated vector fonts exists within OpenVG (since version 1.1) and also some modern desktop GPU hardware (with questionable quality [N1]), it is more advantageous to have access to the glyph rendering directly prior to the final rendering to utilize existing support for font effects, formatting, layout and other NUI engine specific capabilities. To provide some cutting edge visual effects such as glowing or shadowed text the uSwish platform supports various text rendering engines for glyph support but handles pixel level drawing internally. Given the large amount of content that requires text rendering, it is a great potential bottleneck for improvement from parallel computing. By processing the font data through the GPU the cost per glyph would be much less especially if the complete cycle from vector data to final rendered image could be processed in a single pass without requiring data transfer back to the CPU. This solution would also decrease the complexity of the font management since the requirement for font caching schemes would be almost moot. Advanced font visual effects such as transitions and distortions would also benefit from encapsulation into a single GPGPU processing cycle.

Layout of NUI elements and content frames

Layout is a common concern in development of a strong NUI solution and requires considerable consideration of the traditional GUI counterparts since dynamic reorganization of content is common. Additional features such as device rotation from portrait to landscape view, usually very cumbersome on traditional GUIs needs to be entertaining and immersive in a NUI environment. Transitions between UI modes or applications must be fluid and contextually useful, requiring very fast layout recalculations. All this requires live updates at high frame rates and applies well to a potential GPGPU implementation. Assumptions made in current CPU based implementations, such as handling large framed zones as single elements rather than each element within each zone can be removed. An example would be a free flowing transitions between 2 view states in an email client, where individual elements (even down to the letters in the email content) could be animated independently allowing for some very compelling visual effects. Current CPU based implementations simply rotate the complete application display or animate only portions of the display between states. As device resolution continues to increase enhancing the amount of usable screen real estate there is an almost exponential increase in the amount of concurrent data presented to the user, demanding increased complexity of the underlying layout handlers making it a prime candidate for GPGPU optimizations.

Enhanced Visual features

A common use of GPGPU functionality is to process image or visual effect data to achieve effects either not supported by the GPU hardware, or not efficient in the normal OpenGL shader paradigm. For example a particle system used to highlight areas of the UI would be fairly costly in terms of the CPU processing, involving collision detection, motion calculations, and finally utilizing the GPU for rendering. Using a GPGPU implementation the GPU could both process and render freeing the CPU for other operations, minimizing memory transfer and optimizing efficiency allowing for more particle elements or complex motion of the particles. An example of this type of particle system is detailed in "Heterogeneous Computing with OpenCL" [R1].

Sorting of data or other data handling services

Data manipulation and sorting is a common theme in most content heavy applications. As an example, many NUI based devices will provide summary data of SMS, email, tweets, etc. to the user in an organized fashion for quick consumption of the data. The representation of this data can be manipulated by sorting the events for date, importance, keywords and other user provided filters. As the amount of available local storage space even on mobile devices is quite large the sorting process becomes a highly CPU intensive process leading to potential breaks in the user experience and high CPU power consumption. By utilizing GPGPU processing we can perform these tasks much more efficiently. Given the latest generation of multi-core GPU devices we would be able to utilize multiple cores to greatly reduce the time associated with this task.

Compression / decompression of content stores

The high cost of memory transfer is a concern with large content stores and streaming data. Local content for NUI elements such as images, pre-rendered SVG content, etc., can be very complex or large depending on the visual impact intended for the user. We can see this example clearly in the Experience Station [*3] NUI implementation where dynamic controls are used for all user interactive elements. It is very common for GPU devices to support texture compression to provide reduced memory footprint of texture data as stored on the local GPU RAM, however the availability of GPGPU computing allows us to utilize compression of data stores of all local content that can be decompressed during the loading phase without reducing access time for the content. Expanding this capability through streaming data would allow for higher potential bandwidth and reduction in cost of data usage. This long term application would require content providers to compress the content in a format compatible with the GPGPU based decompression. This method could be applied to content carrier formats or markup languages such as HTML. Today many web servers provide mobile versions of their content, yet next generation HTML5 based servers could potentially detect the presence of a compression supported mobile device and relay the content in a compressed format, reducing bandwidth and data transfer costs. This process could be dynamic, allowing for optimization between bandwidth and data processing performance of the target device.

Physical properties of objects – reactive physics effects processing

In order to provide a NUI experience it is crucial to provide very accurate “real world” physical properties to elements of the interface. Simple acceleration and velocity based calculations for scrolling lists are being replaced by more accurate mass, gravity, and inertia models or accelerometer driven effects to provide a very compelling interaction for the user. As the complexity of NUIs increases to take advantage of the rendering capabilities of modern GPUs, the complexity of applied mathematics will increase, benefiting greatly from GPGPU acceleration.

Complex animation or motion schemes for NUI visuals

The motion of elements in a NUI design can be very complex, requiring smooth easing motions and high order Bezier curve based animation paths and transitions. These calculations can be very costly as they must be performed for each drawn frame based on true elapsed time. Traditional optimization would involve pre-calculation or prediction algorithms to simplify this task, but this is not always feasible in a true dynamic NUI experience. Since each element affected in the NUI requires individual calculation, the GPU SIMD architecture is a powerful solution which allows for full calculations in real time with minimal performance impact.

Handling streaming NUI content

One of the key areas of concern in an immersive NUI is the seamless flow of content to the user, both from local and remote stores. Dynamic control of content, whether simple text based data, images, video, or other forms should be handled smoothly in the background without interfering with the user’s interactions. A good example is an image viewer application permitting the user to browse perhaps hundreds of photographs stored locally, many stored at very high resolution. In one example application developed using the uSwish NUI system, an almost infinite panel of available images is provided in a grid format, allowing the user to freely navigate and quickly browse large amounts of photo content. Depending on number and layout of the images, very high amounts of processing power are required to decode and display quickly enough to maintain a fluid natural flow to the user. Any operation which breaks the natural flow or motion results in an unresponsive feeling for the user. By making use of available GPU cycles to offload traditional CPU operations, the bandwidth demands can be decreased and the desired look and feel of the interface can be maintained. Even with very high bandwidth demands and complex content stores, the user experience can be kept consistent.

Real time security encryption / decryption

Security in mobile devices has always been a concern, and is becoming increasingly more important as typical users migrate to using them as their main means of connectivity. Using complex encryption algorithms for protecting sensitive information has been challenging at best on mobile platforms in the past. By using GPGPU capabilities, it would be possible to provide much more secure encryption of local content without hindering the experience of the device for the user.

You I Labs uSwish engine specific features

The uSwish platform utilizes a fully dynamic physics-enabled real time interface environment with full dynamic timeline animation control for a rich NUI experience. It is not uncommon for high levels of interconnected elements and effects to be manipulated during transitional events within the scope of the interface. These complex interleaved sequences would benefit from GPGPU acceleration as it would permit much higher numbers of connected objects to be processed and more complex computations to be performed at each step in the animation timeline.

Limitations and Cost of Implementation

The potential benefits from using available GPU cycles for NUI enhancement are obvious: The UI can be more visually appealing, increasingly interactive, yet smooth and responsive. By utilizing GPGPU below the UI layer the platforms could benefit from more elaborate user inputs and gestures as well as support for new video or audio format decoders, expanding the useful life of the device as these standards evolve. Certainly these potential benefits and enhancements would be nice to have, but at what cost?

OpenCL or similar support

One of the biggest challenges today is making use of the GPU within mobile devices. While the performance for processing graphics is quite phenomenal, support for OpenCL or similar vendor specific APIs is not always available. Making use of existing OpenGL 2.0ES graphics APIs to achieve complex GPGPU functionality is very cumbersome and costly to implement. Another concern is the performance of functionality crucial to GPGPU applications that typically is hindered on mobile platforms: Frame Buffer Object (FBO) support and pixel read-back performance. Both of these factors greatly limit the feasibility of using the GPU for outside tasks. We must realize that the throughput of the GPGPU kernel is only one part of the equation: we must consider the costs of transferring data from CPU to GPU space and back, which can be very costly. Even if the complexity of the calculations and amount of data to process make throughput on the GPU seem factors above the comparable CPU implementation, it may be negated by memory transfer speed and other hardware factors.

Code complexity:

When working with large data sets such as encoded video streams, it is typical to optimize the implementation to always utilize the GPU pipeline due to the nature of the content. The same cannot be said for NUI elements where some operations may cross the boundary between requiring GPU performance and actually being hindered by the overhead of the GPGPU process.

In these cases intelligent decisions must be made as to CPU versus GPU for best case data processing. This requires that a proper NUI framework support both implementation methods, adding development costs, code complexity, and associated costs.

Sharing Device Resources:

Another major concern is the use of the available resources on the platform by external devices and applications. A NUI design which makes heavy use of available GPU cycles may encounter usability issues when resources are not available. A device using a complex gesture recognition system may need heavy GPU cycles or have stringent timing constraints that make the use of GPU optimized UI elements simply not accessible while it is running. Applications that expect full GPU cycles to be available may not perform correctly if cycles are taken by the NUI for background tasks or data / content processing.

Conclusions

As multiple processing core devices become more prevalent in the market, the need for parallel computing and intelligent use of the CPU resources has become paramount. Platforms with multiple core CPUs are now commonplace, even in the mobile space. GPUs themselves are no longer simply fast polygon renderers but powerful computational blocks capable of providing tremendous value in offsetting CPU load for certain functions. Natural User Interfaces require large amounts of processing power to provide users with a fluid experience and will easily benefit from the potential performance and efficiency offered by utilizing available GPU cycles. Existing limitations which complicate implementation today, such as support for standards like OpenCL, are quickly being rectified. As mobile devices migrate from being peripheral to desktop computers to effectively replacing them, processing performance must scale to meet user demand while still maintaining efficiency to ensure long battery life. By utilizing GPGPU in NUI applications we can maintain the best user experience by maximizing the hardware's full capabilities.

Notes

1. http://support.mozilla.com/en-US/questions/791489
2. Experience Station NUI: http://youilabs.com/youi-labs-powers-inwindow-outdoor%E2%80%99s-experience-station/

References

1. Benedict Gaster, Lee Howes, David R. Kaeli, Perhaad Mistry and Dana Schaa (2012): *Heterogeneous Computing with OpenCL*. Advanced Micro Devices and Elsevier Inc.