WebCL for Hardware-Accelerated Web Applications

Won Jeon, Tasneem Brutch, and Simon Gibbs
Advanced Technology Lab
Samsung Information Systems America
3000 Orchard Parkway, San Jose, California
{won.jeon, t.brutch, s.gibbs}@samsung.com

ABSTRACT

Mobile devices, such as smartphones and tablets, now run full feature browsers capable of handling rich media and web content. The emergence of HTML5 makes the browser an ever more attractive platform for application developers. In addition, improvements in JavaScript engines are further shrinking the performance gap between native applications, typically written in C and C++, and web apps, those written in web-based technologies (HTML, CSS and JavaScript). However there is still one area where native applications can show significantly better performance: compute-intensive functions, such as complex image and audio processing algorithms, are considered beyond the reach of JavaScript. This work looks at removing this last deficiency from the web application platform. Specifically we show how high-performance compute capabilities of multi-core CPUs and programmable GPUs can be made accessible to web applications and then discuss the standardization of this technology and its implementation for a mobile browser.

Keywords

WebCL, HTML5, WebKit, GPU, OpenCL, WebGL, hardware acceleration, parallel computing, multicore

1. INTRODUCTION

WebCL is a proposed JavaScript binding to OpenCL [1], which allows web applications to leverage heterogeneous parallel computing resources such as multi-core CPU and GPU. It enables significant acceleration of compute- and (therefore) visual-intensive web applications such as image/video processing, advanced physics, gaming, augmented reality, etc.

WebCL is designed to enable high performance, general purpose parallel processing on multicore/manycore platforms with heterogeneous processing elements, for web applications. It provides ease of development, application portability, platform independence, and efficient access to heterogeneous multicore/manycore devices through a standards-compliant solution. WebCL will enable a breadth of interactive web applications with high compute demands, on platforms with multicore and manycore resources.

Samsung and Nokia jointly proposed formation of a WebCL Working Group in Khronos and both companies have made their prototype WebCL implementations open source. The Khronos WebCL working group will define JavaScript APIs for interacting with OpenCL or equivalent computing capabilities. WebCL is intended to be an interface above OpenCL, which will facilitate higher level abstractions on top of the WebCL API. The goal of the WebCL working group is to produce a WebCL specification, IDL definition, implementation document on guidelines for layering WebCL on OpenCL, and the definition of conformance process. Similar to the WebGL standard [2], support for WebCL may be added to browser engines, using the interface definition from the Khronos WebCL working group. WebCL requires a modified browser, OpenCL driver and runtime support, in addition to GPU and/or multicore processor supporting OpenCL or similar technologies. Security will remain the highest priority, and WebCL will be designed for security. The design will ensure that adding the new functionality into the browser does not increase its exposure to attacks. The WebCL working group will work on providing near term provisions to promote robustness, and will work with hardware vendors, providers of OpenCL device drivers, and browser vendors for an in-depth solution supported by hardware/firmware-supported multitasking.

WWW'12Dev, April 16-20, 2012, Lyon, France.

1: Disclaimer: we use the term "WebCL" to refer to Samsung's WebCL APIs and prototype implementation hereafter.

2. Design of WebCL[1]

2.1 Design considerations

Our WebCL prototype uses a JavaScript API for interacting with OpenCL. To ensure portability, the prototype is not biased towards a specific solution space. Our WebCL prototype is not intended to be a higher level API to satisfy everyone. There are several design choices in defining WebCL's programming model.

Single vs. multiple namespaces: For WebCL API design, a single namespace means that there is a single WebCL JavaScript object, whereas multiple namespaces means that WebCL defines multiple JavaScript objects, in which a subset of WebCL APIs are defined accordingly. For example, WebGL has a single namespace, which means that all the WebGL APIs are called from a single object `WebGLRenderingContext`.

The OpenCL specification defines ISO C-99-styled C APIs as well as C++ bindings which internally call underlying C APIs and introduce little execution overhead. The C++ binding defines multiple classes for platform, devices, and contexts in its C++ platform layer, and memory objects, buffer objects, images, samplers, programs, kernels, events, user events, command queues in its runtime layer.

For our WebCL prototype, we provide an object-oriented interface similar to OpenCL's C++ bindings.

Error handling: There are multiple design choices in handling errors when a WebCL function is called. For example, WebGL's error reporting mechanism involves calling `getError()` and checking for errors. It is basically reporting errors from the OpenGL/ES state machine. On the other hand, JavaScript has its own mechanism to throw JavaScript errors. For example, `throw()` creates a user-defined exception or error and `Error` is a built-in JavaScript object commonly used in conjunction with `throw()`. In

addition, JavaScript provides try, catch, and finally statements to catch errors.

We support a JavaScript-like exception handling mechanism in the current version to improve compatibility with JavaScript.

Initialization: OpenCL provides a number of APIs for querying the capabilities of the platform and for selecting and configuring computational resources. As a result, typical OpenCL applications contain lengthy initialization sections. For our WebCL prototype, we provide access to the same initialization APIs as in OpenCL. The WebCL working group is currently simplifying the initialization interface, for enhanced portability; however this aspect of WebCL design is still evolving in the WebCL working group.

Interface to JavaScript objects: In order for WebCL to seamlessly process existing JavaScript objects, such as those representing media content, it has to provide a clean interface to create, read, and write such objects. For instance, OpenCL provides clCreateImage2D() and clCreateImage3D() to create a 2D and 3D image objects respectively. In addition, it has clEnqueueReadBuffer() and clEnqueueWriteBuffer() to enqueue commands to read/write from/to a buffer object to/from host memory, and clEnqueueReadImage() and clEnqueueWriteImage() for the same purpose on 2D image objects. In JavaScript, a 2D or 3D image object could be represented by either an HTMLCanvasElement or ImageData element, or a typed array such as an Int32Array or Float32Array object.

Kernel source vs. binary: In OpenCL, a program executable can be built using source code or a precompiled binary. Using source code for WebCL kernels provides code portability whereas the source code is visible as a human-readable format. On the other hand, using the binary format of the kernel program could eliminate steps for runtime compilation, but it could seriously hinder code portability and increase security vulnerability if the binary is malicious.

Therefore, for security and portability issues, we only allow kernel executables built from source code provided as part of the web page. This design decision may be changed if a portable binary representation, such as LLVM bitcode, is defined for kernels. Note that currently the Khronos OpenCL-SPIR sub-group [3] is discussing a low-level intermediate representation using LLVM for code obfuscation and security.

2.2 Basic Functionality

The functionality of our WebCL implementation covers creating OpenCL objects such as contexts, queues, and buffers, and building and running OpenCL kernels. Our prototype also supports "WebGL interoperability" which refers to sharing objects in GPU memory between WebCL and WebGL.

WebCL initialization: OpenCL APIs are accessed from JavaScript through the WebCLComputeContext class. A WebCL context is initialized by creating a new WebCLComputeContext object in JavaScript. Once the object is created, WebCL's computing platform, device, and context are created within the WebCLComputeContext object.

Kernel creation: WebCL Kernels are the main functions that execute on the computing device(s). Similar to OpenCL, the source code of the kernel is described in C99-like language and the corresponding program is created and built by calling createProgramWithSource() and buildProgram() respectively, The actual kernel is then created by createKernel().

Memory object creation: Memory objects in OpenCL are reserved regions of global device memory that can serve as containers for user data. In WebCL, memory objects are created from different objects in JavaScript, such as canvas image, JavaScript image, and typed arrays [4].

Kernel execution: Once the buffer objects used for input and output data are created by createBuffer(), the data are actually written from the host memory to GPU memory by calling enqueueWriteBuffer(). setKernelArg() sets the argument values for kernel parameters and getKernelWorkGroupInfo() returns information, such as workgroup sizes, used for the kernel execution. The actual execution of the kernel is initiated by enqueueNDRangeKernel().

WebCL clean-up: releaseCLResource() releases all WebCL-related resources that are allocated. Use of this function is optional, as the browser frees all the WebCL objects when the page is unloaded.

2.3 Interoperability with WebGL

3D graphics applications, such as gaming and augmented reality, are an area where we see great potential for WebCL. On web-based platforms, 3D graphics is provided by WebGL, so it is important that WebCL and WebGL do not conflict with each other.

Fortunately OpenCL is designed for interoperation with OpenGL and provides APIs for safely sharing buffers with OpenGL. The shared buffers reside in GPU memory, thus there is no need to copy data back and forth between host memory and GPU memory when switching between OpenGL and OpenCL processing. WebCL/WebGL interoperability builds on that available for OpenCL/OpenGL. First the application creates a WebCL context via the createSharedContext() API. The application can then create a WebCL memory object (buffer) that is associated with a WebGL buffer via the createFromGLBuffer() API. When WebCL processes the buffer, it is acquired and then released by calling enqueueAcquireGLObjects() and enqueueReleaseGLObjects(), respectively.

Note that WebCL also supports a way of handling HTML5's Canvas, Image, and Video elements, so that they can serve as sources for enqueueWrites*(). Canvas elements can serve as destinations for enqueueReads*() as well.

3. IMPLEMENTATION

We use WebKit [5] as a codebase for our WebCL implementation. WebKit is a layout engine designed to allow a web browser/runtime to render web pages and execute web widgets/applications. It has been used as a core of popular web browsers such as Google Chrome and Apple Safari, and runs on both PC and mobile platforms.

The WebKit engine consists of three main components, WebCore, JavaScriptCore, and WebKit. WebCore is a layout, rendering, and Document Object Model (DOM) library for HTML and SVG (Scalable Vector Graphics). JavaScriptCore is a JavaScript engine that interprets or JIT (Just-in-time)-compiles and executes JavaScript. WebKit wraps WebCore and JavaScriptCore to provide a common application programming interface (API) to browser or application developers.

WebCL provides a JavaScript binding for OpenCL to web applications by modifying the internal binding mechanism implemented in WebCore.

The bridge between JavaScript and OpenCL is the WebCLComputeContext class. A WebCLComputeContext object is associated with the current browsing context, specifically Window object, by being inherited from ActiveDOMObject object. WebCLComputeContext object is a main object in WebCL design and implementation, which defines and implements most of JavaScript APIs which is used by web application developers. Other WebCL objects such as WebCLContext, WebCLDevice, WebCLPlatform, WebCLProgram, WebCLKernel, WebCLMemObject, WebCLCommandQueue, etc. basically maintain native OpenCL objects as member variables and are being used as arguments of WebCL APIs defined in WebCLComputeContext object.

We initially integrated our WebCL code to WebKit revision 78407 (release date: February 10, 2011) and the most recent WebCL implementation is integrated to WebKit revision 101696 (release date: December 2, 2011). Our code is open-sourced and available at http://code.google.com/p/webcl/.

4. PERFORMANCE EVALUATION

Currently our WebCL prototype is developed with Apple's Xcode and running on Mac OSX 10.6/10.7 with NVIDIA and AMD's GPU which has OpenCL 1.0/1.1 support. However the code is fully portable, as WebKit is available on most of the operating systems, and OpenCL is supported on different computing processors.

The platform used for WebCL prototyping and benchmarking that we use is MacBook Pro with Intel Core i7 2.66GHz, 8GB of memory, and NVIDIA's GeForce GT 330M GPU. The following three WebCL examples are presented to compare the computational performance of JavaScript and WebCL.

Sobel filter (Fig. 1): For a given image, WebCL applies the Sobel filter which is commonly used within edge detection algorithms.

N-body simulation (Fig. 2): WebCL simulates the dynamics of given number of particles, calculating the positions and velocities of the particles under the influence of mutual gravitational forces.

Deformable body simulation (Fig. 3): WebCL deforms the surface of spheres using a fractal noise function; WebGL renders the deformed spheres with shaders for Fresnel and reflective effects.

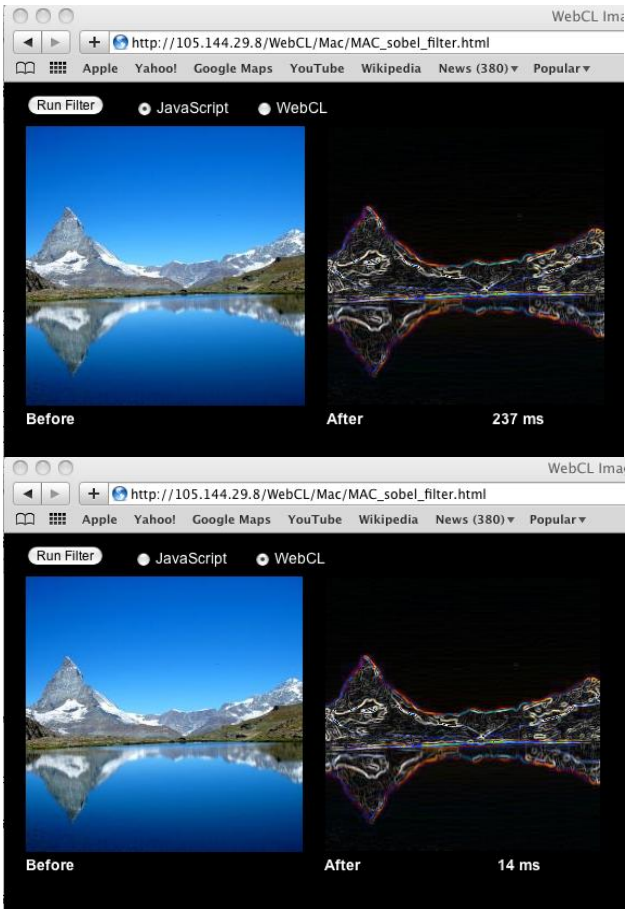

Fig. 1. Sobel filter (top: JavaScript, bottom: WebCL)

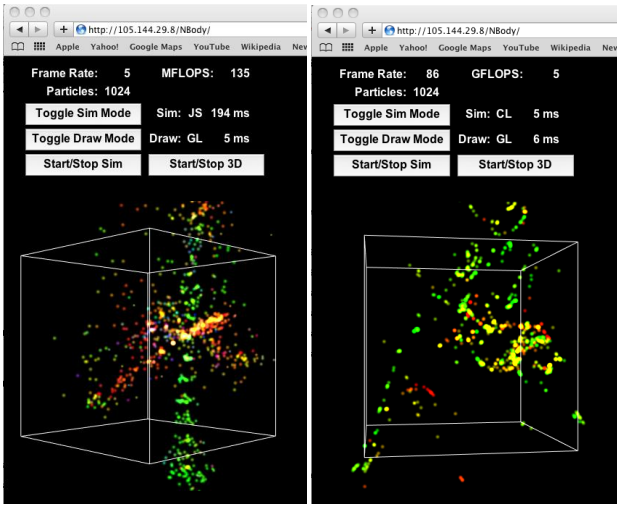

Fig. 2. N-body simulation (left: JavaScript, right: WebCL)

Fig. 3: Deformable body simulation (WebCL)

The performance comparison of JavaScript vs. WebCL is summarized in Table 1. The performance metrics used are: the computation time in Sobel filter, and the frame rate in N-body simulation and deformable body simulation.

TABLE 1
PERFORMANCE COMPARISON OF JAVASCRIPT VS. WEBCL ON PC (INTEL CORE I7 2.66GHZ, 8GB MEMORY, NVIDIA GEFORCE GT 330 M GPU)

Demo Name	JavaScript	WebCL	Speed-up
Sobel filter (with 256x256 image)	~200 ms	~15ms	13x
N-body simulation (1024 particles)	5-6 fps	75-115 fps	12-23x
Deformable body simulation(2880 vertices)	~ 1 fps	87-116 fps	87-116x

The video clips of N-body simulation and deformable body simulation are available on YouTube at http://www.youtube.com/user/SamsungSISA.

5. CONCLUSION

WebCL is a JavaScript binding to OpenCL, which allows web applications to leverage the compute power of multi-core CPUs and GPUs. We are helping drive WebCL standardization in Khronos, and have open sourced a WebCL prototype running in WebKit. Our prototype can be easily ported to smartphones and other devices using WebKit-based browsers.

6. ACKNOWLEDGMENTS

The authors would like to thank Sang-bum Suh and Alan Messer for support of this project plus Niranjan Patil and Siba Samal for their aid in implementing our WebCL prototype.

7. REFERENCES

[1] Khronos OpenCL Working Group, 2011. The OpenCL Specification – Version 1.1. http://www.khronos.org/registry/cl/specs/opencl-1.1.pdf.

[2] Khronos Group, 2011. WebGL Specification- Version 1.0. https://www.khronos.org/registry/webgl/specs/1.0/.

[3] Forshaw, J., Stone, P., and Jordan, M. 2011. WebGL – More WebGL Security Flaws, June 2011. http://www.contextis.com/research/blog/webgl2/.

[4] Khronos, 2011. Typed Array Specification. http://www.khronos.org/registry/typedarray/specs/latest/.

[5] The WebKit Open Source Project. http://www.webkit.org.