

**Computer Games Technology**

A JavaScript Runtime for Hardware Accelerated Applications

**Computing Honours Project (COMP10034) Interim Report**

William Taylor

**B00235610**

17/10/2016

**Supervisor: Paul Keir**

# Introduction

###### The Topic

In recent history there has been a seismic shift in technology. Processors have stopped getting faster at an exponential rate. Increasing the clock speed of processors has now been abandoned in favour of multicore processors. Due to 3D and high resolution media increasing in popularity we now see GPU’s integrated into modern computers by default. Easily learning and experimenting with this new technology is of great importance if we are to see GPU parallel programming more widely adopted.

###### The Problem

Developing and learning how to utilise this technology is a problem however. There is no easy to use integrated environment to experiment with various GPU API’s such as OpenCL and OpenGL which are used heavily for compute and rendering. What’s more using these API’s tends to be in low level C++ leading to a lot of boilerplate which needs to be written before you actually start writing GPU based programs.

###### The Project

In this project we aim to build an all in one platform suitable for GPU programming experimentation, learning and prototyping. We will provide a JavaScript runtime which aims to provide a bulk of features out the box and native bindings to popular industry standard API’s. Finally, we aim that the techniques developed using the platform should be easily integrated into real world applications.

# Technical Review

###### GPU’s

Before proceeding it is best to summarize why GPU programming is an important subject to address. In recent years GPU’s have become more mainstream. If you for instance look at the latest 6th generation intel processors all ship with integrated GPU chips on the die. As such most computers which ship with an intel processor now have a high performing GPU chip waiting to be utilized.

Much research has been taken in investigating the advantage of taking traditional algorithms and seeing the speeds up available. Research by Yang et all (2008) took bread and butter computer vision algorithms and compared their performance when processed across a CPU and GPU. With a histogram they saw a 44x speed up when computed on the GPU. When it came to edge detection they saw a 200x speed up. Additionally, research Teodoro et all (2009) found that optimising a histopathology application result in a speed factor increase of between 19x to 40x.

In computationally expensive tasks we can see GPU’s can provide unseen speed ups in expensive computations. We can also see how a workbench could be advantageous to experiment and test such optimisations.

###### JavaScript

We choose JavaScript as the language for the platform for various reasons. The first is its speed. JavaScript has benefited from a large amount of investment in compiler development with most browser vendors now opting for JIT compilers over traditional interpreters for JavaScript execution. The result is a tenfold increase in JavaScript speed making the language more suitable for high performance applications. JavaScript is also asynchronous meaning that the complexity of multicore programming will not be found here, instead found is a simple and easy to understand event driven style leading to less complexity. Finally, JavaScript has had a new recent standard ECMAScript 2015 which has sought to remove previous issue and present JavaScript as a clear concise general purpose scripting language rather than a language for document object model manipulation in the browser.

###### Typed Arrays

Recent revisions of the JavaScript standard have added support for objects designed to make low level programming possible. I will summarise the most ground breaking set of objects known as TypedArrays objects as it now allows JavaScript to work with binary data directly.

TypedArray Objects

Typed arrays were added in the JavaScript standard ECMAScript 2015. They provide the ability to interact with raw binary data. They provide the perfect way to work with low level data structures and provide much more control over the underlying data.

ArrayBuffer

ArrayBuffer is the core type for every Typed Array object and it just represents a stream of binary data. Look at the following example we can take the struct person and represent it in memory in JavaScript with the following ArrayBuffer.



 At this point **person** and **p** share the same amount of bytes in memory. This is an important step forward in JavaScript as it allows us to allocate and control bytes which was a concept absent from JavaScript till this point.

TypedArrays

Following the base type ArrayBuffer you can now also represent arrays of bytes with greater precision than before. JavaScript numbers are defined in the standard as 64-bit double precision numbers. This limits control but with TypedArrays you can now control a great range of integral types. Looks at the following C++ arrays.



Previously it was impossible to have variables in JavaScript that natively mimicked these due to JavaScript having one type for all types of numbers. But that’s to TypedArrays this is no longer the case.



In short the additional of these types to JavaScript better enables the language to interact with low level data structures and binary data. As such when building the platform using these objects was prioritised as it stops the need to convert JavaScript data types to the native ones found in C++.

###### V8 JavaScript Compiler

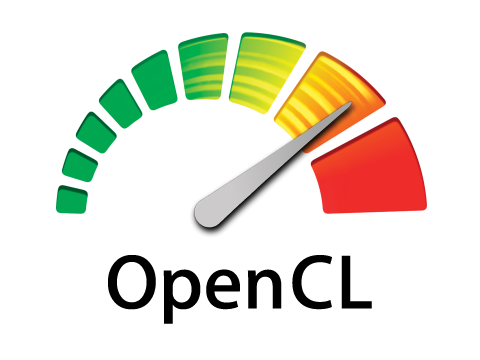
Google back in 2008 set the benchmark for JavaScript compilers. The created a new JavaScript JIT compiler V8 from the ground up to dramatically improve JavaScript execution speed. Internally they built a benchmark called V8 bench and measured performance increases overtime. As you can see from the following graph each subsequent revision of Chrome which in turn has a new version V8 saw massive gains in JavaScript performance.



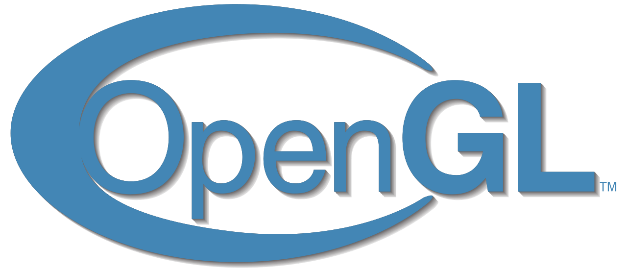
This increase in performance did not go unnoticed. JavaScript can now be found in many environments other than the web. You can now write server side applications in JavaScript with Node.js. You can write full 3D games with the Unity game engine which uses it as its scripting language. Finally, through open source projects with electron you can now write native desktop applications as well. We built our platform on top of the V8 compiler to ensure that the platform is fast and efficient and provides access to the latest JavaScript standard.

###### OpenCL & OpenGL

GPU’s have been traditionally access through industry API’s with most under the umbrella of the Kronos group a non-profit organisation with multiple world leading companies advising and signing off on industry standards. OpenCL and OpenGL are two of their more well-known standards. Although there are other very popular API’s such as CUDA and DirectX they are closed to specific hardware and software.



OpenCL which stands for open compute library is an API designed to allow programmers to process data in parallel across multicore devices such as GPU’s. OpenCL is best used for computer vision and image processing where traditional algorithms can be paralysed to great effect. OpenCL also has a Web variant called WebCL which provides much of the same functionality to web based applications. OpenGL on the other hand stands for open graphics library which aims to provide a pipeline with programmable elements. It’s used for high detailed, high resolution 3D rendering but can also be used for advanced 2D work of general media applications.



The section people learn and experiment the most with are shaders in OpenGL and kernels in OpenCL. These are custom built programs built to take data provided by the program and produce an output either on screen or to an output buffer. The platform we build will be providing a high level environment to mutate and prepare data sent to these programs, but will let the user write the custom shader and kernel programs which can then be imported into any application.

# Current Progress & Plan for Completion

###### Development Technique

###### Gantt Chart Schedule

###### Prototype JavaScript Runtime

###### Current Demonstrations

# Concluding Remarks

###### Excellent Progress

###### Looking forward to demonstrations + presentations

###### Possible future problems

# References

# Appendix

<http://www.ecma-international.org/ecma-262/6.0/>

<http://www.amd.com/en-us/products/processors/desktop/a-series-apu>

<http://www.intel.co.uk/content/www/uk/en/processors/core/core-i7-processor.html>

<http://ieeexplore.ieee.org/document/1425532/>

<http://ieeexplore.ieee.org/document/4722322/>

<http://gernot.web.cse.unsw.edu.au/~keller/Papers/gpugen.pdf>

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.657.4302&rep=rep1&type=pdf#page=45>