Unity game performance difference between Vulkan and WebGL in Linux

Erik Reider

February 12, 2021

Abstract

In this thesis, I compared the CPU performance differences between Vulkan and WebGL2 in Chromium and Firefox on Linux. The benchmarks were made in the Unity 3D game engine using C#. All tests were run on a high-end and a low-end PC and graphed accordingly (figure[1] and figure[2]). The benchmarks tested two CPU heavy loads, raw CPU performance and draw call performance. While testing, OpenGL was used as a control to more easily verify the results because of WebGL being a more lightweight version of OpenGL. The Vulkan results were significantly faster than the WebGL results because of significant overhead created by the browser while the OpenGL results were comparable to the Vulkan results. The conclusion concluded that compiling to a PC natively is the best option when every last drop of performance matters but not when wanting the best cross-compatibility support where the web version shines the most.

Contents

List of Figures

1 Introduction

When creating a game, a programmer can choose where their game will be played. Depending on the platform, it could either be really simple or extremely difficult. The Unity game engine (also referred to as Unity) is a game creation tool that is considered to be one of the best game engines in the world. Unity can build games for Windows, Linux, Mac OS, Android, iOS, every modern console and your web browser. Should one consider publishing ones game to all platforms using WebGL or seperately?

1.1 Background

When compiling (building the project) towards Linux, the preferred graphics API (an acronym for Application Programming Interface) is Vulkan while the web browser uses WebGL2 (OpenGL ES 3.0 but will be referred to as WebGL). The graphics API makes it easier for the developers to render graphics more easily instead of writing code for every graphics card in existence[6].

OpenGL ES is a stripped down version of the regular more feature rich OpenGL while the differences between the latter and Vulkan is more stark. Vulkan is the successor to OpenGL which brings more asynchronous CPU optimizations and has less overhead but because of it being relatively new, it is not as widely adopted as OpenGL. Ergo, one of the reasons why WebGL uses OpenGL instead of Vulkan.

One of the other reasons why WebGL doesn't use Vulkan is because of the lack of native support for it on some platforms like Mac OS and iOS which uses OpenGL and Apple's Metal API. Because of Vulkan being more efficient than OpenGL, it can process more draw calls in a given time than OpenGL. Draw calls are calls to the CPU that contains new materials, textures, objects, etc which is commonly needed when running a game. Each call is then sent to the GPU for processing [2].

Unity uses C# (pronounced as "see sharp") is a Microsoft built programming language based off of the C languages. C# was originaly intended to only be used on Microsofts Windows dotnet platform but then decided to sponsor an open-sourced implemntation of the language named Mono[7]. Unity uses the IL2CPP (acronym for Intermediate Language To C++) implemntation to be able to compile towards almost any platform. IL2CPP is an alternative to Mono that converts dotnet code to C++[4].

When compiling torwards the web, Unity uses IL2CPP and emscripten to convert the C# code to WASM (acronym for WebAssembly)[3]. WASM is a web standard that aims to make C/C++/Rust binary files able to be executed within a web browser while being fast, efficient and safe[5]. To run the web-game, the user needs to open a web-server to host the game instead of running the game regularly.

1.2 Description

Because WebGL has a tendency of being less resource efficient than Vulkan, there should be an observable performance difference between the two, especially on lower end devices where the CPU performance isn't the best. So focusing on CPU limited situations is an excellent way of benchmarking the differences. The benchmarks should be conducted on a low-end PC and a high-end PC to verify the performance differences. To make the comparison as scientific as possible, the high-end PC's CPU will be underclocked to 2.0GHz to simulate a lower-end PC with the same hardware to eliminate any differences except for the CPU. Those results will then be compared to the stock PC. While testing, the FPS(acronym for frames per second) will be logged.

2 Material

Unity 3D

The game engine used to create and compile the benchmarks
Unity

C#

The programming language used to code all unity scripts.
C#

NeoVim and Visual Studio Code

The text editors used.
NeoVim
Visual Studio Code

Mozilla Firefox and Chromium

The web browsers used to test WebGL
Mozilla Firefox
Chromium

Github

The reposetory host used to store all Unity files
Github

Courses

The relevant courses are Programming 1, 2 and the digital creation course offered by NTI Gymnasiet.

3 Boundaries

There will only be two benchmarks with 3 different APIs to test with three reruns to verify the performance results. One of the APIs will be used as a control. Only two of the most popular browsers will be tested.

4 TimeTable

Task	Time to spend
Research CPU bound benchmarks	1-2 days
Implement said benchmarks	3 days
Run benchmarks, increase difficulty if needed	1 day
Run benchmarks	1 day

5 Method

The C# code will generate each benchmark in realtime within the Unity engine. The first benchmark will generate a large 64x64 grid of cubes with random colors and Y positions. On each tick (frame update), each cube will be repositioned and will gain a new material. This should increase the amount of draw calls called but not to the extent as the next test. The second benchmark generates a ring of 256 cubes. On each tick, a new row gets generated with 2 more cubes than the last, building a cone. This benchmark drastically increases the amount of draw calls du to the sheer amount of cubes being added and being visible at the same time.

Each hardware setup will run the benchmark three times to eliminate any anomalies while testing. All of the benchmarks will be ran with Vulkan, WebGL and with OpenGL as a control. To increase the reliability of the WebGL tests, both Firefox and Chromium (which Google Chrome is based off of) will be used to run the benchmark. Those browsers tend to be the most popular choices[1]. While running the benchmark, the FPS will be logged and averaged into separate files per run to ease the comparison between different results and to eliminate any inevitable mistakes. The first benchmark will run for 30 seconds meanwhile the second benchmark will run until the FPS drops below 10. Before running the benchmarks, all non essential programs like Discord, Spotify, Steam will be killed to further eliminate any strange deviations between each run. To make the benchmark more CPU bound, all settings will be set to the lowest value and the resolution will be set to 1080p which will increase the performance delta if there is one.

5.1 PC specifications

- CPU: AMD Ryzen 7 5800X
- CPU performance profile: Performance

- Graphics driver: Mesa 21.1.0-devel (git-e2608312d3)
- Resolution: 1920x1080
- OS and kernel: Linux 5.10.14-119-tkg-upds

6 Results and analysis

Figure[1] shows a massive difference between both platforms while on each platform, the differences are very minutiae. This graph shows the averaged FPS which means that a higher number is always better but, within around a 8% range the differences do not matter. OpenGL is around 3 FPS faster than Vulkan while both browsers perform the same. The performance drop from stock CPU clock speeds to 2GHz is enormous at a 50% drop in performance. The high-end PCs native performance is around 6.75 times faster than both web browsers. The difference on the low-end PC was exponentially larger at an eye-watering 1000% performace increase.

The results from figure[2] are not as drastically different from each other compared to figure[1] but there is still a difference. This test measures time instead of FPS meaning that a larger number results in a higher and a more stable FPS. Vulkan lasted for 3 seconds longer than OpenGL on both systems while lasting 7.5 averaged seconds longer than the web results. The OpenGL results on both systems lasted an averaged 3.75 seconds longer than both web browsers.

7 Discussion

Using the data collected, The performance difference between WebGL and Vulkan is ginormous, especially on lower-end hardware where performance really matters. In the first test, the differences between Vulkan and OpenGL isn't that large but in the second test Vulkan lasted 3 seconds longer which confirms that the second test is more draw call heavy than the former. This shows that Vulkan is not always faster than OpenGL. These tests show that during CPU intense loads, WebGL struggles to keep up with the native versions but gains ground in draw call heavy situations. Meaning that the first test was unrealistically difficult to run but it still shows that there is performance left on the table due to these browsers large overhead. Because of WebGL being based off of OpenGL, the performance should at least be similar or even better but that was not the case.

8 Conclusion

In conclusion, the native version of the benchmarks always ran significantly faster than the web optimized version but gained ground when more draw call

intensive. Compiling to Vulkan or even OpenGL instead of compiling to web platforms is the best option if in need for every last drop of performance, but when in need for extensive cross-platoform compatibility, the web platform could be very convenient.

9 References

[1] Ataul Ghani. "13 Most Popular Web Browsers in 2021 (Windows, Mac & Linux)". In: *usefulblogging* (2021). DOI: https://www.usefulblogging.com/best-web-browsers-windows.

[2] Tonči Jukić. "Draw calls in a nutshell". In: *Medium* (2015). DOI: https://medium.com/@toncijukic/draw-calls-in-a-nutshell-597330a85381.

[3] Unity. *Getting started with WebGL development*. https://docs.unity3d.com/Manual/webgl-gettingstarted.html. Accessed on 2021-02-05. 2018.

[4] Unity. *IL2CPP*. https://docs.unity3d.com/Manual/IL2CPP.html. Accessed on 2021-02-11. 2018.

[5] W3C. *WebAssembly*. https://webassembly.org/. Accessed on 2021-02-11.

[6] Wikipedia contributors. *API — Wikipedia, The Free Encyclopedia*. [Online; accessed 02-February-2021]. 2021. URL: https://en.wikipedia.org/wiki/API.

[7] Wikipedia contributors. *C Sharp (programming language) — Wikipedia, The Free Encyclopedia*. [Online; accessed 09-February-2021]. 2021. URL: https://en.wikipedia.org/wiki/C_Sharp_(programming_language).

Figures

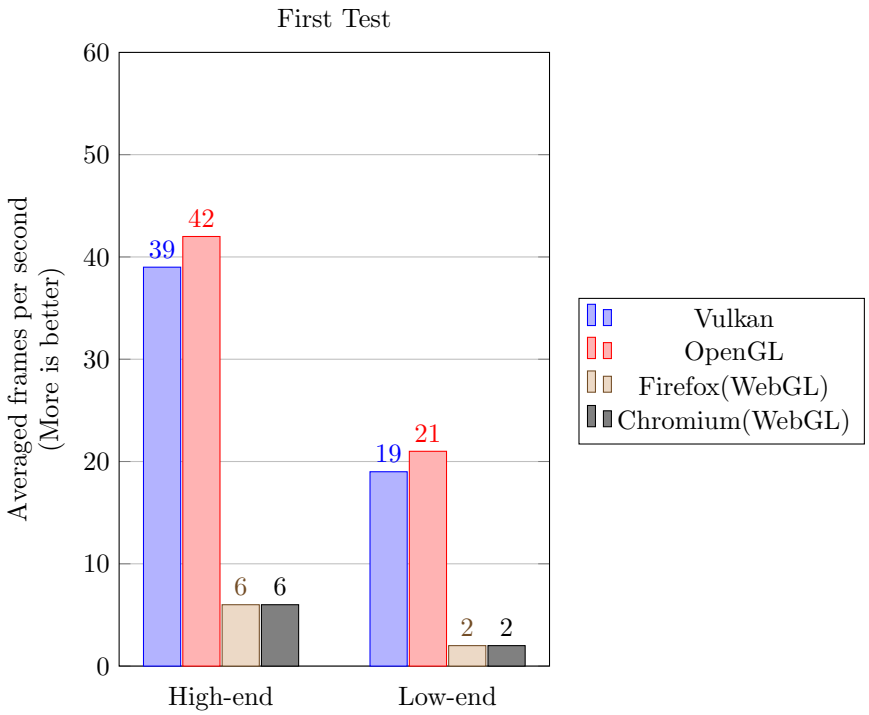

Figure 1: Colored Cubes Test

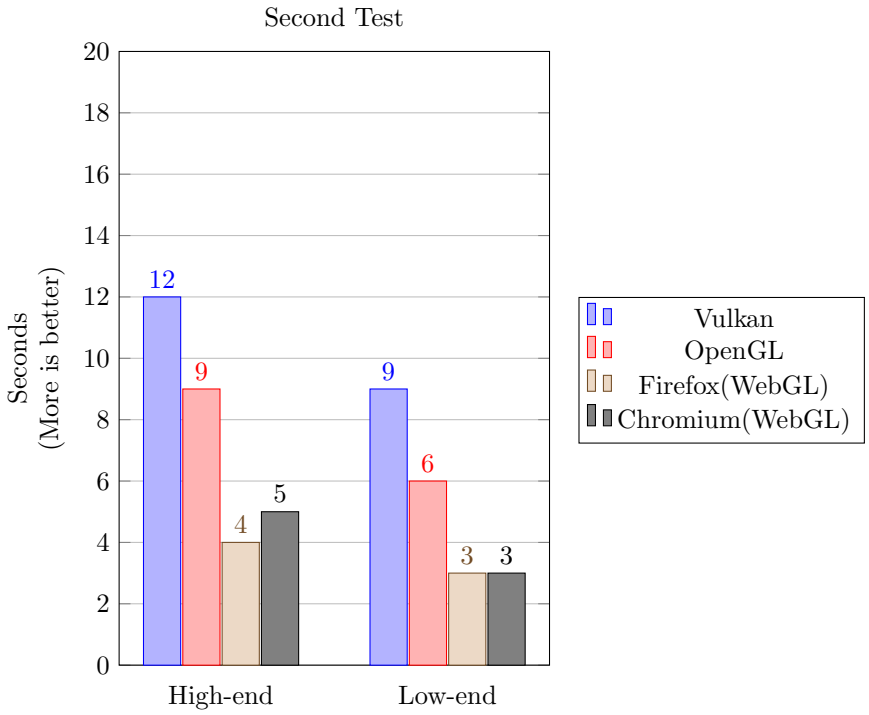

Figure 2: Colored Cone test