WEBGPU: A NEW GRAPHIC API FOR 3D WEBGIS APPLICATIONS

Z. Usta

Department of Geomatic Engineering, Engineering Faculty, Artvin Çoruh University, Artvin, Turkey - ziyausta@artvin.edu.tr

KEY WORDS: WebGPU, Visualization, 3D Graphics, 3D Rendering, 3D City Model, WebGIS, Digital Twins.

ABSTRACT:

As smart city applications increase today, the importance of web-based representations of data that serve as inputs for these applications, such as 3D city models and digital twins, is growing. Since 2011, GPU hardware has greatly advanced, and detailed 3DCMs and digital twins have increased the size of the data to be displayed, which in turn increases performance requirements. WebGL, which couldn't fully harness the capabilities and power of modern GPUs, began to struggle to meet the increasing performance demands over time and become a bit outdated. Consequently, a new graphic API named WebGPU was developed by W3C as a response to limitations of WebGL and went live with Chrome 113. Now, we have a brand-new graphic API called WebGPU which harness the full power of modern GPUs and more performant than WebGL. As WebGPU is a new graphic API, its potential enhancements and what it can bring over WebGL in the terms of WebGIS have not been examined yet. Hence, the main idea and contribution of this work is to investigate WebGPU in real-world use cases for WebGIS applications and discuss what it brings over WebGL. For this purpose, a side-by-side performance comparison has been made. The comparisons have been made in the terms of API differences and performance. And finally, an experiment has been carried out how much data can be rendered can be rendered at minimum 60fps in both APIs. The experiments show that WebGPU is more low-level and way more performant than WebGL and it has a lot to offer in the terms of WebGIS applications.

1. INTRODUCTION

1.1 Background

HTML5 and WebGL are two de facto technologies that are used together to display 3D content on the web. HTML is a markup language that defines the structure and content of web pages, and HTML5 is the latest version of this language. On the other hand, WebGL (Web Graphic Library) is a graphic application programming interface (API) used to create browser-based 3D graphics that run on the 'canvas' element of HTML5 without using any software or additional plug-ins. Before the invention of WebGL, in order to visualize 3D content via browsers and developing 3D web applications, additional plug-ins must be used or standalone software had to be installed on the client's device, such as Flash and Silverlight. Plugins for browsers such as Cortona3D, FreeWRL, or Java applets such as XNavigator have been used for visualizing 3D contents on the web. Nasa WorldWind and Google Earth were able to work web-based but had to be downloaded and installed. After being redeveloped using WebGL, Google Earth now can be used without any additional installation. Another example is Unity Web Player which and had to be installed for displaying video games that were developed using Unity3D in the browsers. is now deprecated.

Using HTML5 and WebGL technologies, many studies have visualized 3D city models (3DCMS) and digital twins on the web without any software or plug-in installation and visualizing these 3D geospatial data on the web has become an import- ant topic in the field of WebGIS. Gesquie`re and Manin (2012) visualized CityGML data using WebGL. Jaillot (2020) visualized time-dynamic data along with 3DCMs and Gaillard et al. (2020) developed an approach which visualizes 3DCMs in multi-scale resolutions.

WebGL is based on OpenGL, originally developed in 1992 and started to get a bit old in today's technology stack. Today, modern GPUs are more complex and powerful hence, to better take advantage of modern GPUs' advanced features WebGPU has emerged as a new graphic API for the web. WebGPU a brand-new graphic API, represents a significant evolution in web graphics technology, offering a powerful and modern approach to rendering on the web. As an emerging standard, WebGPU aims to overcome the limitations of its predecessor, WebGL, by providing developers with a lower-level, explicit API for accelerated graphics and parallel computation on the web platform. Hence, WebGPU introduces a paradigm shift by providing developers with a sophisticated and efficient API designed to fully exploit the capabilities of modern GPUs for immersive 3D experiences. Unlike WebGL, which is based on OpenGL and designed for immediate mode rendering, WebGPU is built on a more modern and efficient foundation. It provides a lower-level abstraction that allows developers to take advantage of the full capabilities of modern GPUs while being more closely aligned with the design principles of contemporary graphics hardware which undergone significant development since the release of WebGL in 2011.

WebGPU is developed by the W3C GPU for the Web Community Group with engineers from big software vendors such as Apple, Mozilla, Microsoft, and Google and draft version of the API has been released in 2021 (W3C, 2021). WebGPU, which has been in development for a while, went live and started to be supported by the standard versions of major browsers with the release of Chrome 113. WebGPU offers developers relatively direct access to GPU resources that were previously inaccessible, and it additionally offers general computation functionality beyond rendering that was not previously accessible in WebGL, thanks to the inclusion of compute shaders (Fransson and Hermansson, 2023). WebGPU is a cutting-edge technology that is poised to revolutionize 3D graphics on the web. It offers low-level, general-purpose access to GPUs, enabling the development of sophisticated web-based graphics applications (Galera, 2023).

As a completely new API, WebGPU is not investigated extensively hence, the aim of this study is to examine the WebGPU API in the context of 3D WebGIS applications. The render performance of the WebGPU API has been tested with real-world datasets, and the differences from the WebGL API both in the terms of API design and in the terms of performance have been analysed.

1.2 Related Work

Goh et al. (2022) states that although WebGPU is in an experimental phase it can be used for front-end deep learning apps. Franke and Haehn (2020) emphasize the use of WebGPU for three-dimensional computer graphics and augmented/virtual reality devices, indicating its potential for modern scientific visualization. Bohak et al (2023) have been developed an engine based on WebGPU. Beyer et al. (2022) mention the potential of WebGPU for building high-performance terascale frameworks, further underlining its significance in the field of computer science. Furthermore, Wang and Durrant (2022) emphasize the role of WebGPU in enabling GPU-accelerated graphics and calculations, indicating its relevance in computer-aided drug discovery. Additionally, Kenwright (2022) provides an introduction to the WebGPU API, further underlining its significance in the field of computer science. Hidaka et al. (2017) dis- covered that their execution of a deep neural network (DNN) with WebGPU achieved approximately 36 times greater speed (91 ms compared to 3297 ms) when compared to another widely used DNN implementation on the web, which relies on the emulated compute capabilities of WebGL. Usher and Pascucci (2020) conducted a comparison between the compute capabilities of WebGPU and native Vulkan, revealing a notable similarity in performance, especially when handling compute-intensive tasks. The study utilized the marching cubes algorithm applied to a scalar field as a representative example of compute-intensive operations. The findings demonstrated com- parable performance, with WebGPU typically falling within the same order of magnitude and frequently even closer to the time-to-render values achieved by the Vulkan implementation. Dyken et al. (2022) examined the comparative rendering performance of large-scale graph layouts on the web using libraries relying on different technologies, including WebGPU (GraphWaGu), WebGL (NetV and Stardust), and non-GPU- accelerated counterparts (like D3 Canvas). With the computational capabilities of WebGPU, GraphWaGu stands out as the sole GPU-utilizing library capable of concurrently computing iterations of graph algorithms. When handling 100,000 nodes and 2,000,000 edges, only GraphWaGu sustains an interactive rendering frame rate of ten or more. In contrast, NetV achieves a frame rate of three, and StarDust is unable to render the graph layout altogether. Pushing GraphWaGu to its limits, it successfully renders a maximum of 200,000 nodes and 4,000,000 edges a remarkable achievement unmatched by any other tested library, whether WebGL-based or not. Fransson and Hermansson (2023) examined and quantified the performance diference between WebGL and WebGPU using Godot Engine as backhand. Five different games namely Checkers, Snake, Evader, Ponder and Falling Cats Deck Before Dawn were rendered with using both WebGL and WebGPU in Godot Engine environment. In every game, WebGPU sur- passes WebGL in terms of both average CPU and GPU frame times. The acceleration achieved by employing the WebGPU Rasterizer varies between approximately 6.8 and 35.6, indicating a significant outcome. Moreover, WebGPU is expected to open new possibilities for 3D games in web browsers, lever- aging the advantages of GPU APIs, as highlighted by (Mehannaand Rudametkin, 2023). Helmrich and Käll, (2023) have parallelized Boolean operations using WebGPU. Yee t al. (2023) optimized occlusion culling based on WebGPU. Additionally, Ammann et al (2022) have developed a cross platform map renderer based on Rust and WebGPU and Erazo et al (2023) have been developed a high-performance client-side computational library specifically designed for web-based hydrological and environmental science applications using WebGPU, Web Assembly, and native JavaScript.

When the related work investigated, it can be seen that most of the works focused on the computation capabilities of the WebGPU utilizing compute shader. However, by comparing only computation capabilities, full render performance is not compared and rasterizing capabilities have been neglected. It is also important for 3D WebGIS applications to compare full render performance. In this context only Fransson and Hermansson (2023) compares rendering performance of WebGL and WebGPU. But in their work, they compared using Godot engine. A side-by side comparison in the browser environment have not been done yet. Additionally, none of the works have examined differences between two APIs in the context of geo-spatial applications.

2. METHODOLOGY

2.1 Examining Key API Differences Between WebGPU and WebGL

One of the biggest differences between WebGL and WebGPU APIs lies in the way they handle resource management, work preparation, and GPU (graphics processing unit) submission. While WebGL relies on a single context object that oversees all aspects and encompasses a considerable amount of associated state, WebGPU takes a different approach by segregating these functionalities into distinct contexts (Figure 1). In summary, this division enables sophisticated web applications to stream data through one or more workers. This aligns with multi-threading scenarios seen in native graphics-intensive applications, facilitating the effective utilization of multi-core processors hence, this makes WebGPU more parallel than WebGL. Additionally, Since WebGL's global state model posed challenges in developing resilient, modular libraries, and applications due to its complexity and fragility, WebGPU has notably minimized the volume of state that developers must manage when issuing commands to the GPU.

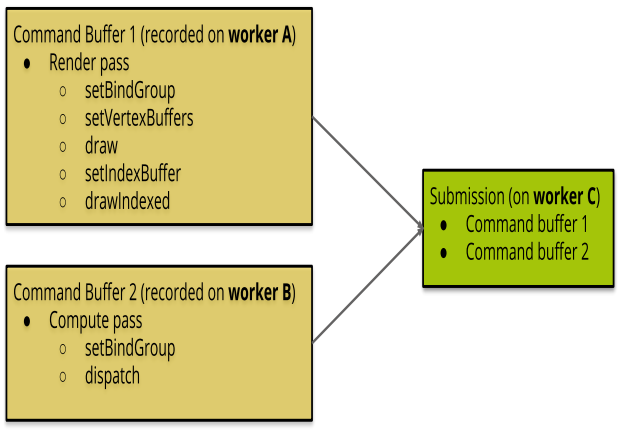


Figure 1. WebGPU Different Resources (URL 1).

The difference between the two graphic APIs starts at the beginning in the initialization. WebGL is initialized using the canvas method "get context()" on the canvas element then by selecting "WebGLRenderingContext". WebGPU initializzation starts with creating a GPU device, a swap chain, and a com-mand encoder. "The navigator.gpu.requestAdapter()" method is used to obtain a GPU adapter. Unlike WebGL, The GPU which WebGPU runs on can be selected at this stage. This is useful for modern machines with multiple GPUs.

In these graphic APIs, there are small piece of programs called "Vertex Shader" and "Fragment Shader". Vertex shader is responsible for calculation of the 2D screen coordinates of the object' vertices from 3D object coordinates and fragment shader operates on each fragment (or pixel) generated by the rasterization process, determining the final colour and other attributes of the pixel. Fragment shaders are responsible for tasks such as computing lighting effects, applying textures, and handling transparency. They receive interpolated data from the vertex shader, such as colour, texture coordinates, and normals, allowing for detailed and dynamic rendering. GLSL (OpenGL Shading Language) is used for programming shaders in WebGL and WGSL (WebGPU Shading Language) is used for same purpose in WebGPU. GLSL has a C-like syntax while WGSL has a Rust-like syntax.

In WebGL operations are generally synchronous, with some asynchronous capabilities through the use of callbacks. WebGPU provides more explicit support for asynchronous operations, especially in the context of parallel computation and multithreading.

There are some differences in space convention that effects the calculations and must be taken into account when migrating from WebGL to WebGPU. First, In WebGL texture coordinates starts from bottom-left and in WebGPU texture coordinates starts from upper-left. Same difference is true for clip space convention (Figure 2).

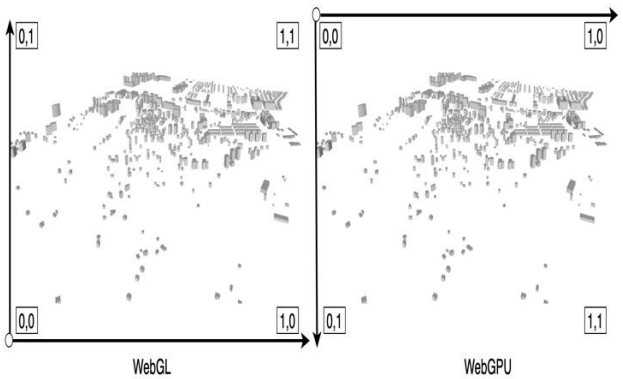


Figure 2. Viewport Space Convention

Second, in WebGL, the Z clip space range spans from -1 to 1, whereas in WebGPU, it ranges from 0 to 1. Consequently, objects with a z value of 0 are considered the closest to the camera, while those with a z value of 1 are perceived as the farthest away. Hence there are no negative values in WebGPU (Figure 3).

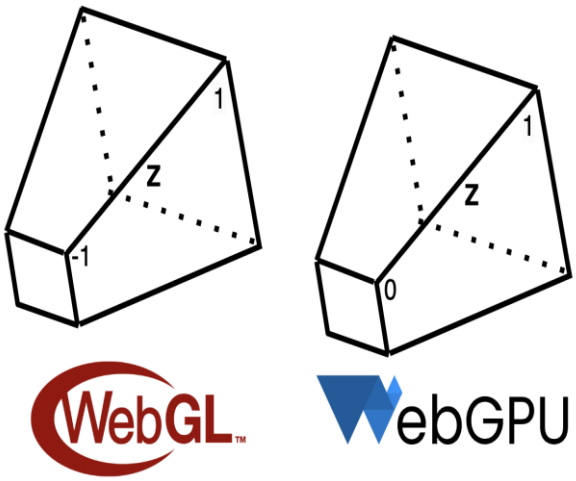


Figure 3. Z values in clip space in both APIs.

While, WebGL automatically handles the canvas upon the creation of a WebGL context and the provision of context attributes like alpha, antialias, colorSpace, depth, preserveDrawingBuffer, or stencil, WebGPU requires lower level manual canvas management. WebGL takes care of canvas management for you. When establishing the WebGL context, you specify parameters such as antialias, preserveDrawingBuffer, stencil, depth, and alpha. Following this setup, WebGL automatically handles the canvas, and your only responsibility is providing canvas.width and canvas.height parameters. In WebGPU, a significant portion of these tasks requires manual intervention. If you require a depth buffer, you must create it independently, with the option of including a stencil buffer. Similarly, for anti-aliasing, you need to generate your own multisample textures and then resolve them into the canvas texture. To illustrate, achieving antialiasing in WebGPU involves the creation of a multisample texture for rendering. Subsequently, the multisample texture needs to be resolved to a standard texture, which is then drawn onto the canvas. This hands-on canvas management in WebGPU offers the flexibility to output to multiple canvases from a single GPUDevice object, a capability not present in WebGL, which is limited to creating only one context per canvas. Hence, in WebGPU multiple canvases can be created for a single GPUDevice object.

In WebGL, numerous elements are linked through names. In contrast, within WebGPU, all connections are solely established through byte offsets or indices, commonly referred to as locations. It is the burden of the developer to maintain synchronization between the locations in the WGSL code and JavaScript.

WebGPU API uses 4-byte-aligment for memory access hence, buffer sizes must be a multiple of 4. This 4-byte-aligment make memory access faster but in return while creating buffers, buffer sizes must be round up to the closest multiple of four.

In WebGL, buffers and textures can be resized. It is possible to create a buffer or texture and modify its size at any point. For instance. However, in WebGPU, the sizes, usage, and formats of textures and buffers are immutable. While you can alter their contents, other aspects cannot be modified.

In WGSL, if you do not explicitly define the type of a variable, it will be inferred from the type of the expression on the right. This

is in contrast to GLSL, where you are obligated to always specify the type of a variable.

Another distinction between WebGL and WebGPU is that in WebGPU, it is possible to include multiple shaders within the same source. In WebGL, the entry point for a shader was invariably named "main," whereas in WebGPU, when utilizing a shader, you specify the particular function to invoke. Additionally, multiple shaders can be compiled at once in WebGPU which is not possible in WebGL.

In WebGL, when your shader didn't compile, you had to manually verify the COMPILE_STATUS using gl.getShaderParameter. If it failed, you had to extract the error messages by calling gl.getShaderInfoLog. If this check wasn't performed, no errors would be displayed, and you might encounter an error later when attempting to use the shader program.

In WebGPU, most implementations will automatically display an error in the JavaScript console. While you can still manually check for errors, it's advantageous that even without explicit checks, you'll still receive useful information if an issue occurs.

2.2 Comparing WebGL and WebGPU performance

In this section same datasets, different 3D city models which consist of different number of objects, rendered using both WebGL and WebGPU, and runtimes have been measured. For this purpose, an experiment has been designed using pure WebGL and WebGPU avoiding to use third party libraries such as THREE.js or Babylon.js. This decision has been made to exclude post-processing and any other advanced optimization techniques which these kinds of libraries heavily use and only to focus raw rendering performance differences of two APIs. Additionally, since WebGPU respectively new graphic API, renderers of such libraries based on WebGPU is not as mature as renderers of such libraries based on WebGL. Hence, it is not fair to compare a mature fully optimized renderer with a new not fully optimized one.

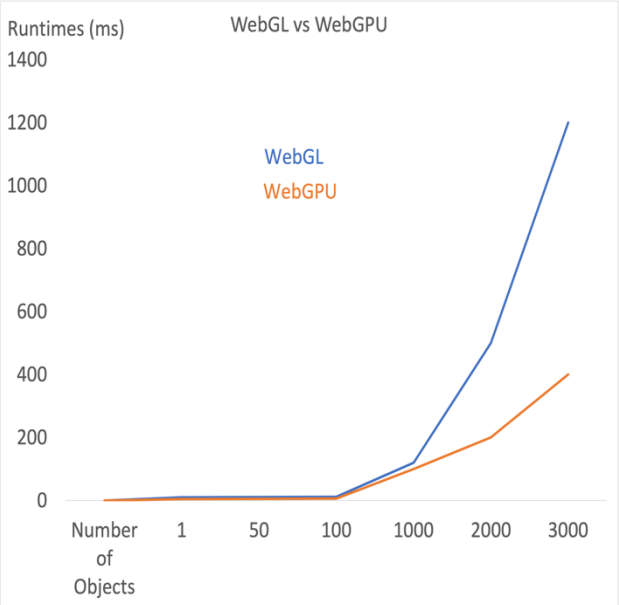
First, started with the simplest single building which consists of 8 vertices and 12 indices. Then, by gradually increasing the data size, rendering performance has been measured as runtime in milliseconds. Larger datasets mean more vertices and indices to render for the APIs. Table 1 shows the sizes of the test datasets in the terms of number of vertices and number of indices.

Number of Objects	Number of vertices	Number of Indices
200	2462	4124
400	5096	8592
600	7298	12196
800	10724	18248
1000	14380	24760
2000	28570	49362
3000	43280	74420

Table 1. Datasets used in the comparison.

3. RESULTS

WebGPU outperforms WebGL in all datasets. Results of the comparison has been showed in Figure 4. Even for a simplest single building which consist of 8 vertices and 12 indices WebGPU is 2.5x faster than WebGL. The difference between performances increases even more as the data size grows. Experiments shows that for rendering 3D city models in a browser environment without using any optimization techniques WebGPU performs 2.5x-3x better performance than WebGL.



4. DISCUSSION AND CONCLUSION

This study has demonstrated that, although the performance difference may vary depending on the type of application, the hardware specifications of the test machine, and the features utilized by the rendering application, WebGPU is three times more efficient than WebGL in terms of raw performance when it comes to displaying 3D city models in the browser. There are several reasons why WebGPU exhibits better performance.

First, WebGPU is designed with a lower-level API, providing developers with more direct access to GPU resources. This allows for finer control and optimization opportunities, contributing to improved performance by providing better tuning opportunities.This allows for optimized control over the rendering pipeline and efficient utilization of the underlying hardware

WebGPU is explicitly designed to support parallelism and multithreading, making more efficient use of multi-core processors, hence, takes advantage of modern multi-core processors more efficiently. This is particularly beneficial for graphics-intensive tasks and computations which are very suitable for parallelism such as matrix multiplications. Historically, WebGL lacked explicit support for parallelism, which could limit its performance in scenarios involving parallel processing.

Tailored to leverage the capabilities of modern graphics hardware, WebGPU supports advanced features and optimizations. This ensures that it can take full advantage of the latest technologies and hardware enhancements, hence it can align with modern graphic hardware more efficiently, While

WebGL is compatible with a wide range of devices, it may not fully exploit the capabilities of the latest graphics hardware due to its design origins. WebGPU incorporates newer technologies and optimizations designed to enhance graphics rendering. These innovations contribute to faster and more efficient execution of graphics operations. Being an earlier technology, WebGL may not benefit from the latest optimizations and advancements introduced in more recent graphics APIs like WebGPU.

WebGPU provides developers with explicit control over resource management, allowing for efficient handling of textures, buffers, and other GPU resources. This flexibility contributes to better performance in diverse scenarios. While WebGL automates certain aspects of resource management, the level of control may not be as granular as in WebGPU, potentially affecting performance in specific use cases.

It is worth mentioning, before designing the experiment, WebGPU renderer of THREE.js library has been utilized for testing purposes. Due to the observing significant performance differences among various versions of the THREE.js library, it was decided to conduct tests without using the library. For instance, rendering same data with THREE.js r154 runtime has been measured as 3,2ms while rendering same data with THREE.js r156 was 6,7ms. This result indicates that WebGPU renderers of libraries are still not sufficiently optimized.

On the other hand, when it comes to the disadvantages of WebGPU, as a side effect of lower-level API design, it is much more verbose than WebGL. For instance, for handling canvas or generation mipmap, a lot of calculations or operations must be done by developer itself while WebGL automatically handle these kinds of tasks for developers.

Additionally, it has been observed that WebGPU maintains some of the limitations that WebGL has in terms of web-based 3D geospatial applications. First, as in WebGL, there is only triangle as primitive hence, 3D solids or multi-surfaces in CityGML have to be triangulated before passing the data to the WebGPU. Second, there is a limitation for float values such as vertex coordinates. The decimal number precision used in WebGPU is typically based on the 32-bit floating-point format known as 'float32.' This implies that decimal numbers have an approximate accuracy of 7 digits. In other words, a floating-point number of this type can provide precise results up to 7 decimal places. In terms of geospatial applications, georeferenced coordinates usually have more digits than 7 hence, to solve a technique called high precision rendering must be applied. Hence, vertex data Translated to local coordinates for which 32-bit floating point precision is adequate. At runtime, the model-view matrix is then computed in a way that avoids 32-bit subtraction of large translation components on the GPU (Schilling et al, 2016).

In summary, WebGPU's superior performance compared to WebGL can be attributed to its lower-level API design, explicit support for parallelism, compatibility with modern hardware, innovative optimizations, explicit resource management, and improved error handling capabilities. These factors collectively contribute to a more efficient and powerful graphics rendering engine. Still, we are not as comfortable as in desktop environments in browsers and the large dataset are needed to decompose into smaller data chunks. But what WebGPU brings over WebGL is that data size of the chunks can be 3 times bigger than WebGL without any performance drop.

More test must be done as a future work such as testing RenderBundle feature of WebGPU. Because there are a lot of room for optimization in WebGPU and WebGPU based renderers will be more and more performant in the near future.

REFERENCES

Ammann, M., Drabble, A., Ingensand, J., & Chapuis, B. (2022). Maplibre-rs: toward portable map renderers. The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences; Proceedings of Free and Open Source Software for Geospatial (FOSS4G) 2022–Academic Track.

Beyer, J., Troidl, J., Boorboor, S., Hadwiger, M., Kaufman, A., & Pfister, H. (2022, June). A Survey of Visualization and Analysis in High-Resolution Connectomics. In Computer Graphics Forum (Vol. 41, No. 3, pp. 573-607).

Bohak, C., Kovalskyi, D., Linev, S., Tadel, A. M., Strban, S., Tadel, M., & Yagil, A. (2023). RenderCore--a new WebGPU-based rendering engine for ROOT-EVE. arXiv preprint arXiv:2312.11729.

Dyken, L., & Poudel, P. (2022, June). GraphWaGu: GPU Powered Large Scale Graph Layout Computation and Rendering for the Web. In Eurographics Symposium on Parallel Graphics and Visualization.

Erazo, C. V., Sermet, M. Y., & Demir, I. (2023). HydroCompute: An Open-Source Web-Based Computational Library for Hydrology and Environmental Sciences.

Franke, L., & Haehn, D. (2020, September). Modern scientific visualizations on the web. In Informatics (Vol. 7, No. 4, p. 37). MDPI.

Fransson, E., & Hermansson, J. (2023). Performance comparison of WebGPU and WebGL in the Godot game engine. Master of Science in Engineering: Game and Software Engineering. Blekinge Institute of Technology

Gaillard, J, Peytavie, A., and Gesguiere G., 2020. Visualization and Perdonalization of Multi-Representations City Models. International Journal of Digital Earth. Vol 13. No.5. P.627-644. 2020.

Galera, A. P., de Oliveira, D. R., Gutierrez, F. D., Pires, G. P., Passarin, T. A., Guarneri, G. A., & Pipa, D. R. (2023). A WebGPU-base acoustic wave simulator for ultrasound NDT.

Gesquière, G., & Manin, A. (2012). 3D visualization of urban data based on CityGML with WebGL. International Journal of 3-D Information Modeling (IJ3DIM), 1(3), 1-15.

Goh, H. A., Ho, C. K., & Abas, F. S. (2023). Front-end deep learning web apps development and deployment: a review. Applied Intelligence, 53(12), 15923-15945.

Helmrich, M., & Käll, L. (2023). Parallelization of boolean operations for CAD Software using WebGPU.

Hidaka, M., Kikura, Y., Ushiku, Y., & Harada, T. (2017, October). Webdnn: Fastest dnn execution framework on web browser. In Proceedings of the 25th ACM international conference on Multimedia (pp. 1213-1216).

Jaillot, V., 2020. 3D, Temporal and Documented Cities: Formalization, Visualization and Navigation. PhD Thesis. University of Lyon. 2020.

Kenwright, B. (2022, December). Introduction to computer graphics and ray-tracing using the webgpu api. In 15th ACM SIGGRAPH Conference and Exhibition on Computer Graphics and Interactive Techniques in Asia 2022.

Mehanna, N., & Rudametkin, W. (2023). Caught in the Game: On the History and Evolution of Web Browser Gaming. arXiv preprint arXiv:2304.14791.

Schilling, A., Bolling, J., & Nagel, C. (2016, July). Using glTF for streaming CityGML 3D city models. In Proceedings of the 21st International Conference on Web3D Technology (pp. 109-116).

URL 1. A Taste of WebGPU in Firefox. https://hacks.mozilla.org/2020/04/experimental-webgpu-in-firefox/. Accessed 25.11.2023.

Usher, W., & Pascucci, V. (2020, October). Interactive visualization of terascale data in the browser: Fact or fiction?. In 2020 IEEE 10th Symposium on Large Data Analysis and Visualization (LDAV) (pp. 27-36). IEEE.

W3C, 2021. WebGPU [Online]. Available: https://www.w3.org/TR/2021/WD-webgpu-20210518/

Wang, A., & Durrant, J. D. (2022). Open-Source Browser-Based Tools for Structure-Based Computer-Aided Drug Discovery. Molecules, 27(14),

Ye, L., Liu, G., Chen, G., Li, K., Chen, Q., Fan, W., & Zhang, J. (2023). 3D Model Occlusion Culling Optimization Method Based on WebGPU Computing Pipeline. Computer Systems Science & Engineering, 47(2)