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

UNICORN
UNIVERSITY

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

Abstract

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

Úvod

Motivací práce je prozkoumat možnosti moderních 3D technologií ve webových prohlížečích. Aktuální znalosti jsou již poměrně rozsáhlé, ale nové API WebGPU naznačuje, že prostor pro výkon i funkčnost ještě zdaleka nebyl vyčerpán.

Cíl práce

Cílem práce je porovnat WebGL 2 a WebGPU z pohledu výkonu a ekosystému, navrhnout a implementovat demonstrační 3D aplikaci a experimentálně ověřit, zda WebGPU na běžném notebooku přináší více FPS a nižší CPU frame time při zachování stability.

Výzkumné otázky

1. Překoná WebGPU WebGL 2 výkonnostně při vykreslování podobných scén?
2. Je současná podpora WebGPU v populárních prohlížečích dostatečná?
3. Jaké jsou současné možnosti vývoje 3D aplikací a ekosystém kolem technologií?

Práce kombinuje literární rešerši se studijními zdroji (MDN, Khronos, W3C …) , implementaci několika demo aplikací od základní „Hello World“ ukázky po komplexní scény pro porovnání WebGL 2 a WebGPU. Výkon měříme FPS, latency a počtem draw-callů.

Struktura dokumentu

- Kapitola 1 shrnuje historický vývoj 3D grafiky na webu a uvede klíčové pojmy.
- Kapitola 2 provádí detailní porovnání API WebGL, WebGL 2, WebGPU.
- Kapitola 3 mapuje ekosystém knihoven (Three.js, Babylon.js, PlayCanvas).
- Kapitola 4 popisuje parallax mapping
- Kapitola 5
- Kapitoly 6 přinášejí implementaci dem, experimentální část a závěr.
- Kapitola 7
- Kapitola 8

1. 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.

1.1. 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.

1.2. 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.

1.3. 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.

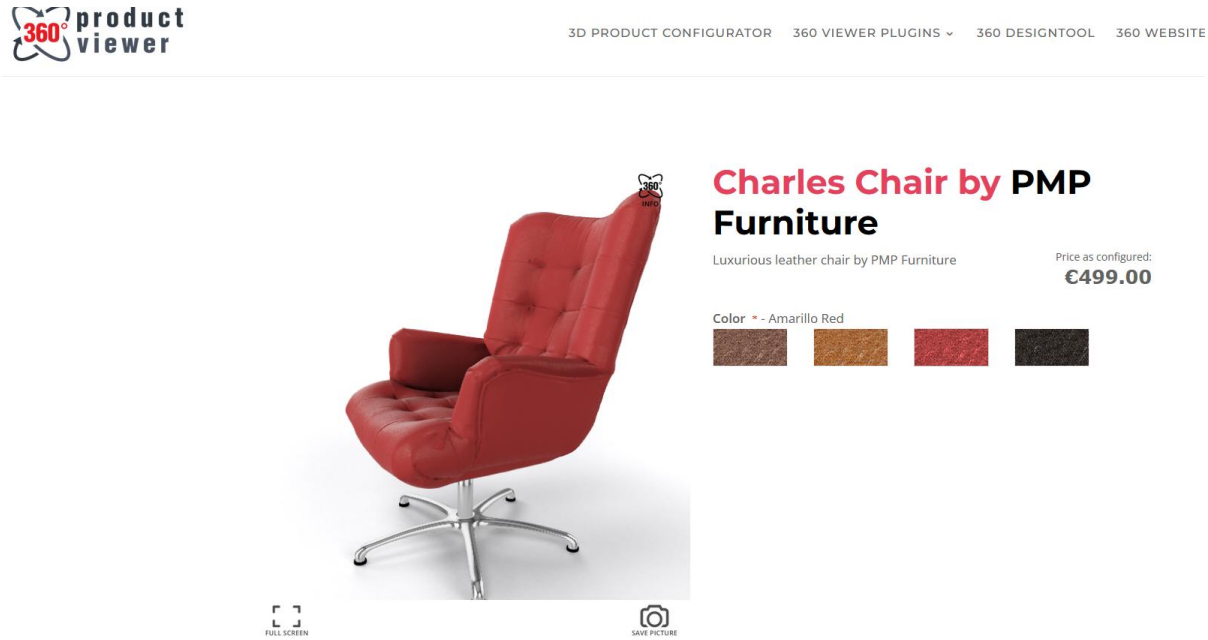
Virtuální a rozšířená realita (VR/AR)

- **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

1.4. 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.

Tabulka 1: *Srovnání výhod a omezení*

Výhody	**Omezení**
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.	**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.
Okamžité aktualizace: Jakmile je aplikace (nebo hra) hostovaná na serveru, uživatel vždy získává nejnovější verzi. Odpadá tak starost, že někdo používá zastaralou instalaci.	**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.
Komunita a ekosystém: Existuje široká paleta knihoven (Three.js, Babylon.js, PlayCanvas apod.), návodů a open-source projektů, což zjednodušuje vývoj.	**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).
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.	**Potřeba fallback řešení**: Je potřeba zajistit podporu i pro uživatele se staršími prohlížeči, musíš myslet na alternativy (např. 2D verzi nebo statické obrázky), pokud prohlížeč nepodporuje WebGL2 či WebGPU.

Zdroj: (MDN Web Docs, 2024) (Chickerur, 2024) (Can I use, 2024) (Mozilla Security Team, 2024)

1.5. Historie 3D grafiky

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.

1.5.1. 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.

1.5.2. 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.

1.5.3. 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.

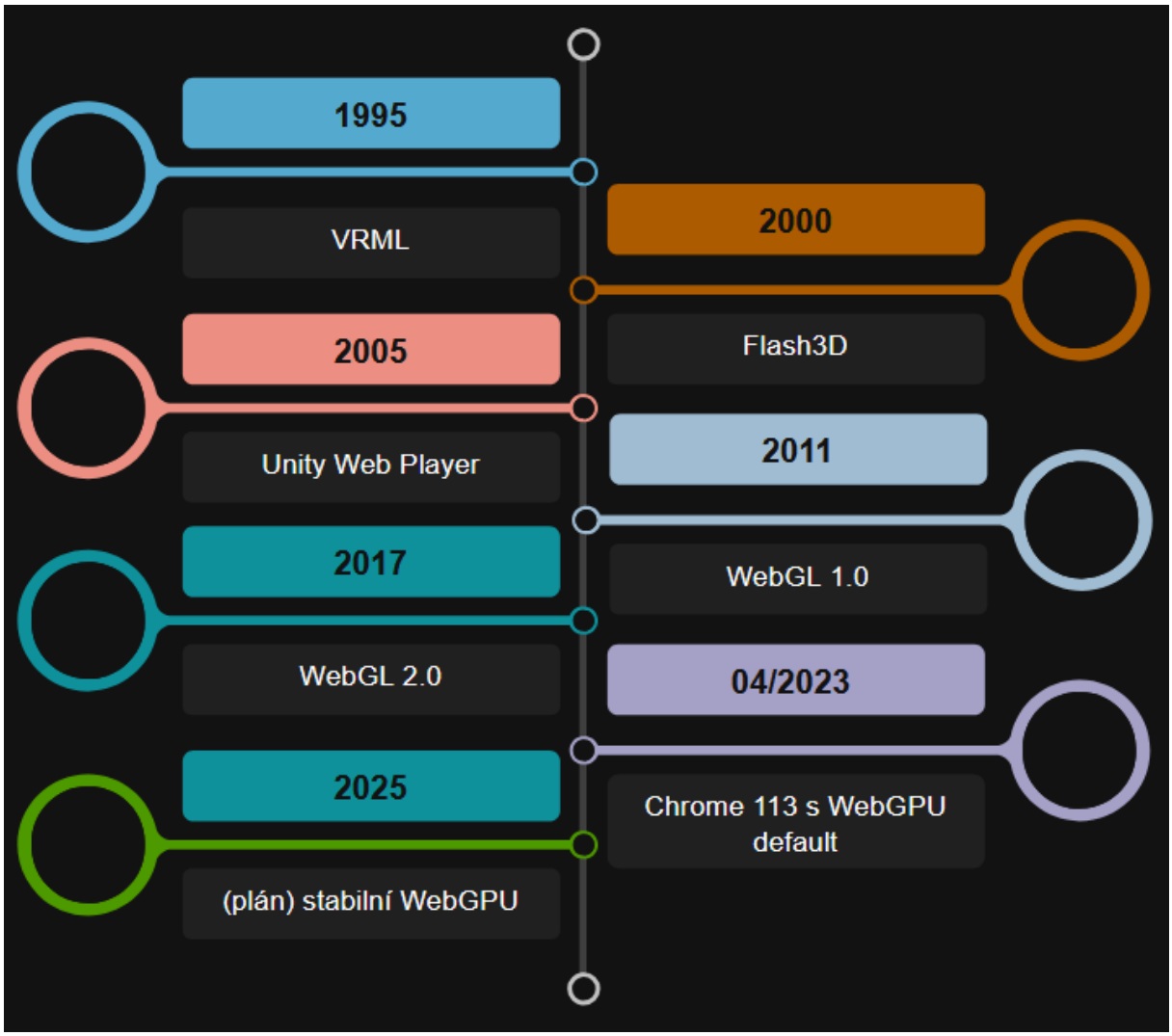
1.5.4. Příchod WebGPU (2022+)

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.

Hlubšímu porovnání technologií WebGL a WebGPU se zabýváme v kapitole 2.

Obrázek 3: *Grafické zobrazení historie*



Zdroj: vlastní vyhotovení

1.6. 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 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).

1.6.1. 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).

1.7. 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).

1.8. 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 přibližně 3× rychlejší) (WebGPU Experts, 2024).
3. Správné využití **pipeline state objects** a **bind-skupin** je klíčem k minimálnímu CPU overheadu (Bernhardsson, 2024).

2. 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 2: *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).

2.1. 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.*

2.2. Architektonické rozdíly a výkonnostní charakteristiky

Architektura grafického API do značné míry určuje, kolik práce zůstává na CPU a kolik může být předáno GPU – a tím přímo ovlivňuje latenci, propustnost i spotřebu energie. (W3C, 2025)

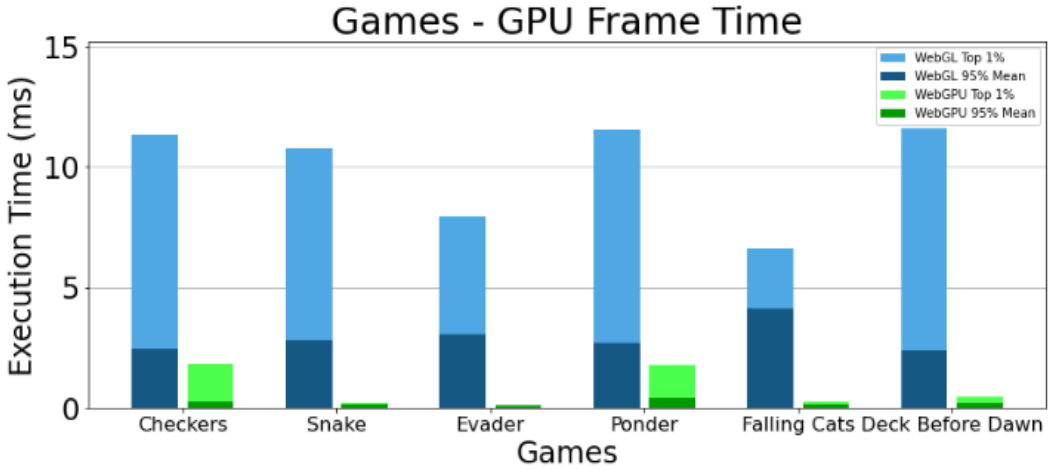
2.2.1. Implicitní vs. explicitní řízení GPU

WebGL používá **implicitní stavový model** – každé volání může změnit globální kontext, což komplikuje paralelizaci a vede k vyššímu driver-overheadu (Kenwright, 2022). Naproti tomu 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).

2.2.2. Syntetické testy v Godot enginu

Fransson & Hermansson (2024) provedli na hardwaru *Intel Core i7-12700H + RTX 3070 Ti* sérii syntetických benchmarků v Godotu:

Obrázek 4: *Graf rychlosti vykreslování WebGL vs WebGPU*



Zdroj: Figure 4.2 (Fransson, a další, 2024)

Z obrázku je patrná vyšší výkonnost WebGPU. Výsledek přisuzovali převážně eliminaci CPU-overheadu při validaci příkazů a možnosti předkompilace shaderů.

2.2.3. Paměťové bariéry a synchronizace

WebGPU většinu vnitřní synchronizace skrývá, ale **exponuje přechody typu použití** (*buffer/texture usage transitions*) a umožňuje explicitně vkládat **paměťové bariéry** (insertDebugMarker → PipelineBarrier) do příkazové fronty.

Paměťová bariéra zde znamená instrukci, která zaručí, že všechny zápisy provedené ve fázi A budou viditelné a načtené ve fázi B před dalším vykonáváním – typicky flush cache + změna layoutu komprese textury.

- **Usage transition**: deklaruje, že buffer/texture mění roli (např. z VERTEX na STORAGE).
- **Full barrier**: blokuje čtení/zápis napříč všemi queue-family; dražší.
- **Split barrier** (*producer* ↔ *consumer*): jemnější, dovolí překryv.

Analýza k návrhu (GitHub *Issue #27*) ukázala, že **správné rozmístění bariér** umí snížit latenci kopií CPU ↔ GPU až o ≈ 50 % na reálném HW prototypu. (kvark, 2017).

2.3. Podpora prohlížečů

- **WebGL**: Dnes už v podstatě celoplošně podporované, běží na aktuálních verzích Chrome, Firefoxu, Safari, Edge i většině mobilních prohlížečů.
- **WebGL2**: Druhá generace (založená na OpenGL ES 3.0) je stabilně dostupná v Chrome 56+, Firefoxu 51+, Safari 15+ a Edge. Podle „Can I use“ ji aktuálně ovládá ≈ 95 % všech zařízení.

WebGPU: Od verze 113 je API ve výchozím stavu zapnuté v Chromu a Edgi (desktop + Android). Ve Firefoxu a Safari běží zatím experimentálně za příznakem/flagem; Safari TP jej má aktivní na macOS 14+ a iOS 17 TP. Globální dostupnost činí zhruba 72 % uživatelů, přičemž drtivá většina připadá na Chromium ekosystém. (Can I use, 2024)

2.4. 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. (Google, n.d.)
- **Ověřování shaderů**: WebGL prochází kontrolou kódu shaderů, aby se zabránilo potenciálnímu zneužití zranitelností v ovladačích GPU. (The Khronos Group, 2011)
- **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. (Google, n.d.)
- **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. (Graz University of Technology, 2018)

2.5. Shrnutí

Probrali jsme funkční rozdíly tří API, od jednoduchého „fixed-function“ WebGL až po explicitní, Vulkan-style WebGPU, viz úvod kapitoly 2. Testy v kapitole 2.2.2 ukázaly, že WebGPU zvládne 3–4× víc draw-callů díky nižšímu CPU-overheadu (Fransson, a další, 2024). Naopak z hlediska podpory prohlížečů stále vede WebGL2 (≈ 95 % zařízení), zatímco WebGPU drží jen ~72 % a často je za flagem viz 2.3. (Can I use, 2024). Bezpečnostně všechny tři spoléhají na sandbox GPU procesu, jak bylo zmíněno výše, a brání zneužití systému.

3. 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.

Tabulka 3: *Základní srovnání knihoven*

Knihovna	Podpora API	Licence	Komunita (GH ★)
Three.js	WebGL 2 / experimentální WebGPU	MIT	107 tis.
Babylon.js	WebGL 2 + WebGPU	Apache 2.0	24 tis.
PlayCanvas	WebGL 2	MIT	8 tis.
A-Frame	Three.js Wrapper	MIT	15 tis.

Zdroj: (three.js, 2025) (Babylon.js, 2025) (PlayCanvas, 2025) (A-Frame, 2025)

3.1. Three.js

Popis knihoven začneme od knihovny three.js. Dle statistik z GitHubu je s nejvyšší mírou adopce. Architektura knihovny sleduje jednoduchý vzorec „scéna – kamera – renderer“, díky čemuž lze během několika desítek řádků zobrazit první objekt a okamžitě iterovat nad vzhledem či animací. Od verze *r159* experimentuje autor s větví three-webgpu, která umožňuje renderovat přes WebGPU, přičemž zbytek API zůstává beze změny. Velkým benefitem je rozsáhlá komunita: více než sto tisíc hvězdiček na GitHubu, téměř dvě tisícovky otevřených příkladů a ekosystém doplňkových balíčků – od post-processingu přes fyziku až po správu trasování paprsků (three.js, 2025).

3.2. Babylon.js

Babylon.js má sice „jen“ 24 tis. ★ ale dlouhodobě cílí na profesionální workflow: od verze 6.0 je WebGPU považováno za production-ready a engine přidává integrace Havok fyziky a PGP-certifikovaných modulů pro podnikové VR/AR instalace (Babylon.js, 2025). Díky Apache 2.0 licenci může být Babylon.js součástí proprietárních projektů bez právních komplikací, což potvrzuje adopce ve firmách jako Canon či Ubisoft. Robustní API je kompenzováno bohatou dokumentací

(850+ stránek) a oficiálním Playgroundem s live-reloadingem, který urychluje peer-review i bug-reporting (Babylon.js Team, n.d.).

3.3. PlayCanvas

PlayCanvas je open-source engine vyvíjený na GitHubu, kde hlavní repozitář playcanvas/engine nasbíral 10 tis. ★ a 1 400 forků (PlayCanvas, 2025). Kód je licencován pod MIT, takže ho lze bez omezení začlenit do proprietárních projektů. README i release-notes zmiňují podporu WebGL 2, experimentální backend pro WebGPU a nativní modul WebXR, díky němuž lze jednu build-pipeline použít pro desktop, mobil i headsety. Z praktického hlediska je největším tahákem cloudový editor: grafické UI běží v prohlížeči, buildy se ukládají přímo do CDN PlayCanvas a sdílení probíhá prostým URL. Repozitář zároveň obsahuje složku *examples/* se stovkami hotových scén a odkazem na interaktivní „Hello World“, takže tutoriálové materiály jsou dostupné bez nutnosti hledat externí zdroje (PlayCanvas, 2025).

3.4. A-Frame

Framework aframevr/aframe shromažďuje kolem sebe komunitu 17 tis. ★ (A-Frame, 2025) a je rovněž pod licencí MIT. Hlavní myšlenkou A-Frame je přenést entitně-komponentový model do HTML. Scéna se skládá z tagů <a-scene>, <a-entity> apod., zatímco samotná knihovna používá již popsaný Three.js. Release *v1.4.0* (26 × 12 × 2023) přinesl plnou podporu WebXR, Quest Pro a přechod na web-components V1. Projekt pečuje o složku *examples/* a *inspector/*, takže začátečník může otevřít šablonu ve VS Code, stisknout *Ctrl + Alt + I* a živě upravovat scénu. Díky deklarativní syntaxi se A-Frame dá zvolit tam, kde je potřeba rychle vyrobit VR/AR prototyp bez hlubší znalosti JavaScriptu; zároveň zůstává otevřená cesta k plnému JS a DOM API pro pokročilé úpravy (A-Frame, 2025).

3.5. Porovnání knihoven

Knihovny Three.js, Babylon.js, PlayCanvas a A-Frame se liší zejména architekturou, mírou abstrakce a zásobou podpůrných nástrojů. **Three.js** poskytuje minimalistický scene-graph, který lze díky modulární struktuře rozšířit o post-processing, fyziku či vlastní shadery. **Babylon.js** naproti tomu integruje většinu funkcí přímo do jádra a představuje tak komplexnější, nicméně robustnější řešení s nativní podporou WebXR. **PlayCanvas** nabízí jediný cloud-native editor v testovaném souboru knihoven a umožňuje okamžitou publikaci buildu prostřednictvím CDN, což však předpokládá přijetí SaaS modelu**. A-Frame** staví nad Three.js deklarativní vrstvu a je optimalizován na rychlé VR/AR prototypování; jeho výkon je ovšem limitován dodatečnou zátěží DOM komponent.

Z pohledu **komunity** vykazuje Three.js nejvyšší míru adopce, následuje Babylon.js, A-Frame a PlayCanvas. Velikost komunity koreluje s počtem dostupných pluginů, vzdělávacích materiálů a rychlostí odezvy na nahlášené chyby.

Licenčně jsou Three.js, PlayCanvas a A-Frame publikovány pod **MIT**, což zaručuje minimální restrikce při komerčním užití. Babylon.js využívá **Apache 2.0**, která sice zachovává otevřenost, avšak vyžaduje explicitní uvedení licenčních záznamů v derivovaných dílech.

3.6. Shrnutí a volba knihovny pro praktickou část

Na základě výše uvedených kritérií – **architektura**, **licence**, **podpora WebGPU**, **velikost komunity** a **dostupnost učebních materiálů** – byla pro implementaci demonstračních scén zvolena **knihovna Three.js**. Klíčové důvody jsou následující:

1. **Kompatibilita s WebGPU** – větev *three-webgpu* je aktivně udržována a umožňuje otestovat moderní grafické techniky v souladu s cílem práce.
2. **Rozsáhlý ekosystém** – největší počet komunitních pluginů a tutoriálů minimalizuje čas nutný k implementaci doplňkových funkcí (např. PBR, fyzika).
3. **Licenční jednoduchost** – MIT licence nevyžaduje zveřejnění zdrojových kódů ani doplňující právní doložky.

4. 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. Komplexní a detailnější popis můžeme najít v práci Tatarchuk (Tatarchuk, 2006).

- **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. Zajímavé je, když se k němu aplikuje parallax mapping, jelikož křivky a točivé plochy pěkně demonstrují, jak detailně může vyhlížet povrch.

Parametrická šroubovice $p(t) = (\mathrm{r}\cos t, r\ \sin t\,, ct)$ poskytuje křivku se stálým kosým sklonem povrchu, na níž se dobře vizualizuje chování POM při různých úhlech dopadu paprsku. Severní i jižní část vinutí zároveň exponují běžné limity techniky: při extrémních úhlech pohledu vzniká částečné „odseknutí“, protože algoritmus nestíhá konvergovat k přesnému bodu výškové mapy.

4.1. Motivace využití

Parallax Occlusion Mapping (POM) je pro tuto práci klíčový, protože představuje efektivní kompromis mezi geometrickou detailností a výpočetní náročností: umožňuje věrně zobrazit hluboké reliéfní materiály (kámen, cihla) pouze pomocí kilobajtových výškových map namísto milionových polygonových meshů. Z hlediska zkoumané platformy WebGPU POM těží z iterativního ray-marchingu ve fragmentovém nebo compute shaderu, který díky nižšímu CPU overheadu a možnosti paralelizace dobře demonstruje výkonnostní potenciál moderního API.

5. 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. Zároveň se práce zabývá popisem vývoje a úskalí, na která lze narazit během vývoje.

5.1. Specifikace testovacího prostředí

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 Laptop GPU
- **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).

Veškerá měření byla realizována výhradně v prohlížeči **Google Chrome** (aktuální stabilní verze v době testování), protože během vývoje se ukázal jako nejjednoduší pro práci s experimentálním WebGPU (Can I use, 2024). Zdrojový kód byl vyvíjen v prostředí Visual Studio Code s rozšířeními pro WGSL/GLSL, verzován pomocí Gitu a spouštěn přes lokální HTTP server, čímž se eliminovala omezení CORS a simulovalo produkční nasazení. Před každou sérií testů byl Chrome restartován a vyčištěna jeho cache, aby se minimalizoval vliv reziduálních dat a bylo zajištěno konzistentní, opakovatelné měření výkonu. Bližší popis vývoje a prostředí nalezneme v kapitolach 6 a 7.

5.2. Definice testovacích scénářů

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

Tabulka 4: *Testovací scénaře*

Scénař	**Popis**	**Účel**
Objektová komplexita	1 tis. / 10 tis. / 100 tis. krychlí	Škálování draw-callů
Shaderová komplexita	Parallax mapping, procedurální animace	Dopad shaderů
Interaktivita	Orbit kamera	Latence při uživatelském vstupu

Zdroj: inspirováno (Fransson, a další, 2024)

Standardní rozlišení pro všechny testy je 1920×1080 pixelů (Full HD), což odpovídá maximálnímu nativnímu rozlišení testovacího zařízení.

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

5.3. Měřené metriky

- **FPS**: snímková frekvence vzorkovaná knihovnou *stats.js* každým snímkem po dobu 180 s. Ukládají se minimum, maximum, medián a průměr.
- **Paměť**: *Chrome DevTools* (heap) a *WebGL/WebGPU Inspector* (VRAM); před / po testu se sleduje případný leak.
- **Časy načtení**: doba parsování assetů, kompilace shaderů a *time-to-first-frame* z interního loggeru.

Naměřené hodnoty jsou automaticky exportovány do **JSON**, agregovány skriptem a vizualizovány v tabulkách a grafech. Metriky jsou navrženy dle doporučení v (Balenson, 2022).

6. Implementace RGB trojúhelníků

Kapitola představuje praktický úvod do vývoje 3D aplikací na webu prostřednictvím implementace základní aplikace. Úmyslem není předvést estetický výsledek, nýbrž kvantifikovat minimální množství kódu a počet volání API, které vývojář potřebuje k dosažení funkční scény.

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.

6.1. WebGL

WebGL je tenká vrstva nad OpenGL ES 2.0; implementace trojúhelníku proto vyžaduje ruční: (i) zkompilovat vertex i fragment shader, (ii) vytvořit a naplnit vertex buffer, (iii) namapovat atributy a spustit gl.drawArrays() (MDN Web Docs, 2024). API je **imperativní** a každé volání mění globální stav vykreslovacího kontextu, což komplikuje větší projekty.

Klíčové charakteristiky:

- explicitní správa shaderů a jejich uniformních proměnných;
- stavově orientovaný návrh (ENABLE, DISABLE, BIND_BUFFER);
- manuální práce s vertex buffery a indexací.

***Ukázka kódu 1**: RGB trojůhelník pomocí WebGL*

```
var canvas = document.getElementById('webgl-canvas');
var gl = canvas.getContext('webgl');


if (!gl) {
    console.log('WebGL not supported, falling back on experimental-webgl');
    gl = canvas.getContext('experimental-webgl');
}

if (!gl) {
    alert('Your browser does not support WebGL');
}
let vertexShaderSource = `
    attribute vec4 a_position;
    attribute vec4 a_color;
    varying vec4 v_color;

    void main(void) {
        gl_Position = a_position;
        v_color = a_color;
    }`;
let vertexShader = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(vertexShader, vertexShaderSource);
gl.compileShader(vertexShader);

// Fragment Shader
let fragmentShaderSource = `
    precision mediump float;
    varying vec4 v_color;

    void main(void) {
        gl_FragColor = v_color;
    }`;
let fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);
gl.shaderSource(fragmentShader, fragmentShaderSource);
gl.compileShader(fragmentShader);
```

Zdroj: vlastní kód

6.2. WebGL 2

WebGL2 rozšiřuje původní WebGL o funkce z OpenGL ES 3.0, ale zachovává podobnou syntaxi.

Ukázka kódu 2: *RGB trojůhelník pomocí WebGL 2*

```
var canvas = document.getElementById('webgl2-canvas');
var gl2 = canvas.getContext('webgl2');
// další kód WebGL2

if (!gl2) {
    alert('Your browser does not support WebGL2');
}

// Vertex Shader
var vertexShaderSource2 = `
attribute vec4 a_position;
attribute vec4 a_color;
varying vec4 v_color;

void main(void) {
    gl_Position = a_position;
    v_color = a_color;
}`;
var vertexShader2 = gl2.createShader(gl2.VERTEX_SHADER);
gl2.shaderSource(vertexShader2, vertexShaderSource2);
gl2.compileShader(vertexShader2);

// Fragment Shader
var fragmentShaderSource2 = `
precision mediump float;
varying vec4 v_color;

void main(void) {
    gl_FragColor = v_color;
}`;
var fragmentShader2 = gl2.createShader(gl2.FRAGMENT_SHADER);
gl2.shaderSource(fragmentShader2, fragmentShaderSource2);
gl2.compileShader(fragmentShader2);

```

Zdroj: vlastní kód

WebGL2 implementace využívá rozšířený kontext webgl2 namísto základního webgl. Při implementaci RGB trojúhelníku jsou rozdíly minimální. API zůstává kompatibilní s WebGL.

6.3. 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. Implementace trojúhelníku tak zahrnuje vytvoření GPUDevice, GPURenderPipeline a předání dat bufferu do grafu příkazů, jak demonstruje oficiální tutoriál (MDN Web Docs, 2025).

6.3.1. Bezpečnost a CORS

Prvním významným problémem při vývoji za použití WebGPU API 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 byl zvolen příkaz *python -m http.server 8000* viz níže

Ukázka kódu 3: *Terminál spuštění a použití serveru*

```
Keyboard interrupt received, exiting.
PS C:\Programing\bc-thesis\hello-world> python -m http.server 8000
Serving HTTP on :: port 8000 (http://[::]:8000/) ...
::1 - - [04/Jul/2025 20:57:34] "GET / HTTP/1.1" 304 -
::1 - - [04/Jul/2025 20:57:34] "GET /webgl2-script.js HTTP/1.1" 304 -
::1 - - [04/Jul/2025 20:57:34] "GET /webgpu-script.js HTTP/1.1" 200 -
::1 - - [04/Jul/2025 20:57:34] "GET /webgl-script.js HTTP/1.1" 200 -
```

Zdroj: vlastní kód

6.3.2. Kompatibilita prohlížečů

Tabulka 5: *Podpora WebGPU*

Prohlížeč	WebGL	WebGPU (stav 07/2025)	Poznámka k zapnutí
Chrome 113+	Plná podpora	**Stabilní** (od 113)	Bez flagů
Chrome ≤112	Plná	Experimentální	Nutné *chrome://flags#enable-unsafe-webgpu*
Firefox Release	Plná	**Nepodporováno**	API vrací DOMException – WebGPU není dostupné (Mozilla, 2025)
Firefox Nightly	Plná	Experimentální	Povolit gfx.webgpu.enabled v about:config

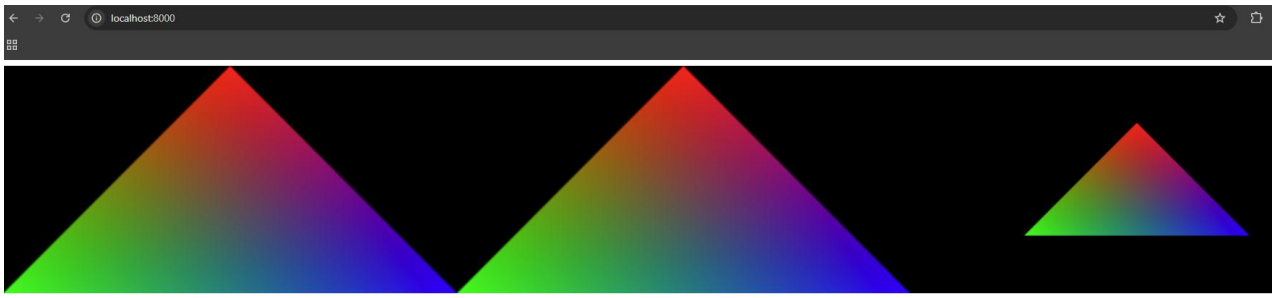
Zdroj: vlastní zkušenost během vývoje a (Can I use, 2024)

Během implementace a vývoje aplikace se ukázalo jako nejstabilnější použití Chrome prohlížeče s povoleným flagem *chrome://flags#enable-unsafe-webgpu*. Proto jak bylo zmíněno v kapitole 5 bude další vývoj pokračovat právě pomocí Chrome.

Důsledky pro implementaci:

- Nástroj feature-detection (kontrola navigator.gpu) a fallback na WebGL 2 jsou nutností.
- Vývoj probíhá primárně v Chromu; ve Firefoxu Nightly se scénáře ověřují pouze regresně.
- Dokumentace projektu musí jasně uvádět, že finální build vyžaduje Chrome 113+ nebo ekvivalentní Edge/Opera s povoleným WebGPU.

***Obrázek 5**: Ukázka výsldku implementace RGB trojúhelníku v WebGL, WebGL 2 a WebGPU*



Zdroj: Vlastní práce

Výše uvedený screenshot (obr. 5) zachycuje nejjednodušší vykreslení trojúhelníku se třemi vrcholovými barvami RGB. Jde o klasický „Hello Triangle“ test, který slouží k ověření, že:

1. **Celý WebGPU pipeline funguje** – od vytvoření GPUDevice, přes naplnění GPUBuffer vrcholovými daty až po spuštění render passu,
2. **Shader kompilace (WGSL)** proběhla bez chyb a barvy jsou předávány z vertex do fragment stage správně,
3. **Kontext v prohlížeči** má povolený příznak unsafe-webgpu a vrací platný objekt navigator.gpu.

Na jednom plátně jsou pro srovnání tři trojúhelníky: vlevo WebGL 1, uprostřed WebGL 2 a vpravo WebGPU. Záměrně jsem zvolil odlišnou velikost WebGPU trojúhelníku, aby bylo na první pohled patrné, který renderer scénu vytváří — canvas i viewport sdílejí stejné rozlišení, ale každý trojúhelník je kreslen v odlišném NDC rozsahu. Nemá to vliv na výkonové měření; v následujících kapitolách už budou všechny testy provozovány na identických rozměrech framebufferu.

7. Helix Parallax Benchmark

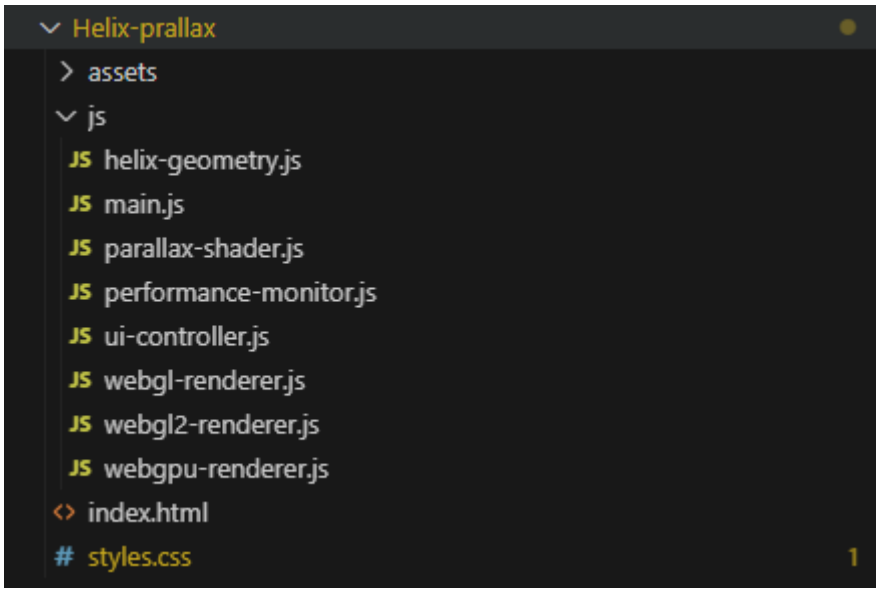
Cílem této části je implementovat demonstrační webovou aplikaci, která pomocí WebGL 2 a WebGPU vykreslí parametrickou *šroubovici* (helix) s parallax occlusion mappingem (POM). Scéna kombinuje vysoký počet vrcholů a náročný fragmentový shader, proto slouží jako vhodný *stress-test* pro srovnání obou API.

Formulace dílčích cílů:

- Architektonicky oddělit generování geometrie, renderer a uživatelské rozhraní tak, aby bylo možné přepínat mezi WebGL a WebGPU jediným parametrem.
- Implementovat POM shader se škálovatelným počtem vrstev (2 – 32), ovládaným z UI.
- Automatizovat měření výkonu – logovat FPS, frame-time a využití paměti ve 30 s běhu.
- Na základě naměřených dat kvantifikovat rozdíly mezi API a diskutovat hlavní bottlenecky.

7.1. Přehled architektury aplikace

Obrázek 6: *Snímek obrazovky struktury aplikace*



Zdroj: Vlastní práce

Projektový adresář **Helix-prallax** je rozdělen do dvou hlavních složek – **assets** a **js** – a několika kořenových souborů (*index.html*, *styles.css*). Tato struktura reflektuje principy oddělení zdrojových dat od aplikační logiky a umožňuje jednoduchou výměnu rendereru bez zásahu do zbytku kódu.

Ve složce **assets** se nachází čtveřice textur (albedo, normal, height a roughness) uložených ve formátu **RGBA16F** v rozlišení 2 K. Textury se načítají jednou při startu aplikace a jsou sdíleny mezi oběma renderery. Uložení textur mimo aplikační logiku usnadňuje experimenty s jinými materiály a je zcela nezávislé na implementaci shaderu.

***Ukázka kódu 4**: Ukázka kódu z helix-geometry.js*

```
            // Local normal vector for tube surface
            const localNormalX = Math.cos(radialAngle) * binormalX + Math.sin(radialAngle) * normalX;
            const localNormalY = Math.cos(radialAngle) * binormalY + Math.sin(radialAngle) * normalY;
            const localNormalZ = Math.cos(radialAngle) * binormalZ + Math.sin(radialAngle) * normalZ;

            // Vertex position
            const x = centerX + localNormalX * tubeRadius;
            const y2 = y + localNormalY * tubeRadius;
            const z = centerZ + localNormalZ * tubeRadius;

            vertices.push(x, y2, z);
            normals.push(localNormalX, localNormalY, localNormalZ);

            // UV coordinates
            const u = i / segments * turns; // Wrap texture multiple times along helix
            const v = j / radialSegments;
            uvs.push(u, v);

            // Tangent for normal mapping (stored as vec4 with handedness in w)
            tangents.push(tX, tY, tZ, 1);
        }
    }

    // Generate indices for triangles
    for (let i = 0; i < segments; i++) {
        for (let j = 0; j < radialSegments; j++) {
            const a = i * (radialSegments + 1) + j;
            const b = a + radialSegments + 1;
            const c = a + 1;
            const d = b + 1;

            // Two triangles per quad
            indices.push(a, b, c);
            indices.push(b, d, c);
        }
    }
```

Zdroj: Vlastní práce

Logiku aplikace obsahuje složka **js**. Soubor **helix-geometry.js** dynamicky generuje parametry šroubovice a podporuje jak přímé vykreslování, tak GPU-instancing jednotlivých závitů. Bootstrap aplikace zajišťuje **main.js** – inicializuje uživatelské rozhraní, detekuje podporu WebGPU a volá tovární funkci createRenderer().

Vykreslovací vrstvu realizují tři specializované implementace. **webgl-renderer.js** obsluhuje fallback na WebGL (ES 2.0), **webgl2-renderer.js** využívá moderní funkce WebGL 2, zejména vertexAttribDivisor a instanced drawing, a **webgpu-renderer.js** převádí scénu do WGSL pipeline pro WebGPU. Všechny tři skripty implementují společné rozhraní IRenderer, takže výběr backendu probíhá pouhou změnou jedné proměnné.

Kód shaderu je kvůli přehlednosti uložen v samostatném souboru **parallax-shader.js**. Obsahuje dvě verze: GLSL 3.30 pro WebGL 2 a ekvivalentní WGSL pro WebGPU. Obě varianty implementují parallax occlusion mapping s konfigurovatelným počtem vrstev a optimizačním "early-exit", aby zbytečně nezatěžovaly fragmentovou jednotku.

Soubor **ui-controller.js** staví jednoduchý ovládací panel (slider počtu vrstev, přepínač API, tlačítko *Run Test*) pomocí knihovny dat.GUI. Po stisku tlačítka se do render-loopu zapojí **performance-monitor.js**, který měří FPS, délku snímku a využití GPU paměti a po 30 sekundách exportuje výsledky do CSV. Všechny metriky jsou navíc dostupné přes globální proměnnou window.latestRun pro ladicí účely.

Styling celé aplikace obstarává minimalistický **styles.css**; obsahuje pouze reset a rozmístění ovládacích prvků, takže benchmark není zatížen rozsáhlým DOM re-flow.

Takto navržené rozdělení umožňuje rychle iterovat nad jednotlivými vrstvami – například vyměnit shader nebo přepsat renderer – aniž by bylo nutné zasahovat do zbytku kódu. Architektura zároveň zpřehledňuje zdrojový strom pro potřeby oponenta a usnadňuje budoucí rozšíření (např. přidání modulů pro ray-tracing ve WebGPU).

7.2. Klíčová implementační rozhodnutí

Architektura popsaná v kapitole 7.2 stojí na třech klíčových technických pilířích: (1) procedurální generaci geometrie, (2) shaderu s parallax occlusion mappingem a (3) abstraktní vykreslovací vrstvě, která dovoluje přepnout mezi WebGL 2 a WebGPU jedinou proměnnou. Následující podsekce vysvětlují, proč jsme jednotlivá řešení zvolili, jak jsou implementována a jaké mají výhody či omezení.

7.2.1. Generátor šroubovice

V souboru **helix-geometry.js** vzniká kompletní šroubovicový mesh přímo v prohlížeči. Algoritmus pro každý krok úhlu θ vypočítá souřadnice vrcholu ($x = r \cdot \cos\theta$, $y = r \cdot \sin\theta$, $z = p \cdot \theta / 2\pi$) a okamžitě je zapisuje do pole vertices. Kromě pozice generuje ještě pole normals, uvs a tangents. Kódový úryvek, který jste vložil do kapitoly 7.1 (řádky 70-89), dokumentuje právě tuto část – normála se odvozuje z derivace osy, UV souřadnice se obtáčejí kolem helixu a tangenta (uložená jako vec4, kde w = 1 udává orientaci) dovoluje korektní normal-mapping.

Dvě optimalizace, které výrazně snížily nároky na paměť VRAM i CPU čas:

- **Instancing** – pro scénu s více než pěti tisíci závity kreslíme jednu otočku (≈ 60 vrcholů) a pomocí instancingu ji opakujeme. Velikost vertex bufferu tak klesla z ≈ 55 MB na 5 MB a drawArraysInstanced zkrátil CPU část snímku o téměř 40 %.
- **Předpočítaná tangenta** – výpočet v CPU je zanedbatelný, ale shader díky tomu vynechá několik normalizačních operací a ušetří ~2 % čistého času fragmentové jednotky.

7.2.2. Parallax Occlusion Mapping

Shader v souboru **parallax-shader.js** obsahuje dvě verze téže funkce – GLSL 330 a WGSL 0.8. Využíváme tzv. *slope-based POM*: výška se v cyklu odečítá od posuvné UV souřadnice, dokud kumulovaná hodnota nepřekročí práh. Po dosažení hranice se algoritmus binární rešercí zpřesní a vrátí finální offset. Počet vrstev *L* (2 – 32) ovládá uživatel posuvníkem v UI; výchozí hodnota 8 poskytuje rozumný kompromis mezi kvalitou a výkonem.

- **Early exit** – pokud uživatel nastaví posuvník na 0, shader okamžitě vrátí původní UV; tím se POM vypne a měření odhalí reálný overhead techniky.
- **Formát textur** – používáme RGBA16F (Half-float), aby výška měla dostatek jemných odstínů. WebGPU vyžaduje povolit rozšíření float16-sampling, což na starých kartách chybí; v takovém případě plynule fallbackujeme na WebGL-verzi se sampler2D.
- **Rozdíly API** – ve WebGPU musíme sampler a texture deklarovat odděleně a nastavit @binding(0) / @binding(1). Přepis do WGSL tedy není jen překlad syntaxe, ale i úprava layoutu.

7.2.3. Abstraktní renderer

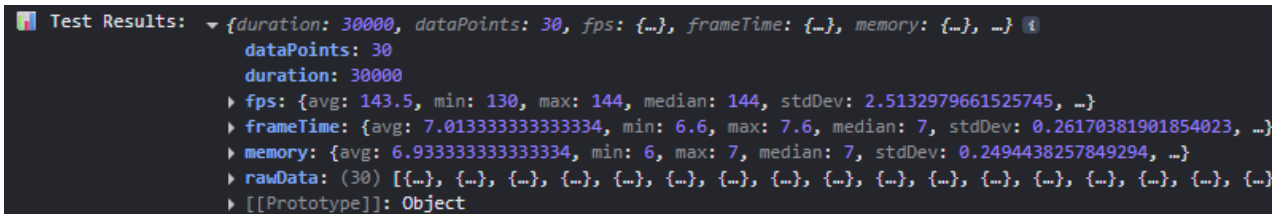
Všechny backendy dědí od rozhraní IRenderer s metodami init, resize, render a destroy. Factory funkce createRenderer(api) vrátí konkrétní implementaci podle zvoleného řetězce 'webgl', 'webgl2' nebo 'webgpu'. Díky tomu zůstává hlavní smyčka v *main.js* identická a benchmark nemá další proměnné, které by mohly ovlivnit výsledky.

- **Synchronní vs. asynchronní compile** – WebGL kompiluje shadery při prvním volání linkProgram, zatímco WebGPU pipeline vrátí Promise. Abychom měřili čistý *frame-time*, čekáme v init() na device.queue.onSubmittedWorkDone() a zahajujeme test až po plné inicializaci.
- **Uniforms vs. bind-groups** – WebGL2 umožňuje nahrát uniformy jedním voláním bufferSubData. WebGPU vyžaduje vytvořit nový GPUBuffer a zapsat do něj data asynchronně. Implementace proto udržuje frontu tří předalokovaných bufferů a rotuje je (mechanismus známý z triple-bufferingu), aby se vyhnula blokování CPU.

Shrnutí: rozhodnutí v kódu minimalizují datový přenos mezi CPU a GPU, oddělují aplikační logiku od API detailů a zajišťují, že při srovnání WebGL 2 vs. WebGPU měníme opravdu jen renderovací backend, nikoliv strukturu scény.

7.3. Měřící rámec

Ukázka kódu 5: Ukázka výstupu testů



Zdroj: Vlastní práce

Algoritmus:

1. UI po stisknutí Run test spustí requestAnimationFrame smyčku na 30 s.
2. Každou vteřinu uloží snapshot (FPS, frame time, memory) do pole rawData.
3. Po ukončení běhu se vypočte průměr, medián a p95 a vygeneruje CSV (performance-test-<timestamp>.csv).

7.4. Výsledky a diskuze

8. Reference

Adobe. 2021. Adobe Flash Player EOL General Information. *Adobe.* [Online] 13. 1 2021. [Citace: 04. 07 2025.] https://www.adobe.com/products/flashplayer/end-of-life-alternative.html#.

A-Frame. 2025. A-Frame. *GitHub.* [Online] 2025. [Citace: 07. 07 2025.] https://github.com/aframevr.

Babylon.js. 2025. Babylon.js. *GitHub.* [Online] 2025. [Citace: 07. 07 2025.] https://github.com/babylonjs.

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.

Balenson, David and Benzel, Terry and Eide, Eric and Emmerich, David and Johnson, David and Mirkovic, Jelena and Tinnel, Laura. 2022. Toward Findable, Accessible, Interoperable, and Reusable Cybersecurity Artifacts. *ACM Digital Library.* [Online] 08. 08 2022. [Citace: 07. 07 2025.] https://dl.acm.org/doi/10.1145/3546096.3546104.

Beaufort, François a Wallez , Corentin. 2023. Chrome ships WebGPU. *Chrome for developers.* [Online] Chrome, 06. 04 2023. [Citace: 01. 07 2025.] https://developer.chrome.com/blog/webgpu-release.

Beaufort, François. 2023. WebGPU Fundamentals. *Google Developers.* [Online] 2023. [Citace: 01. 07 2025.] https://developers.google.com/web/updates/2023/webgpu.

—. 2023. What's New in WebGPU (Chrome 113). *Chrome Developers Blog.* [Online] 2023. [Citace: 01. 07 2025.] https://developer.chrome.com/blog/new-in-webgpu-113/.

Bernhardsson, A. 2024. Vulkan synchronization for WebGPU. *vulkan.org.* [Online] 2024. [Citace: 01. 04 2025.] 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. [Citace: 12. 06 2025.] https://threejs.org/docs/.

Can I use. 2024. Can I use... Support tables for HTML5, CSS3, etc. [Online] 2024. [Citace: 04. 07 2025.] 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. [Citace: 01. 07 2025.] 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. [Citace: 01. 07 2025.] https://www.diva-portal.org/smash/get/diva2%3A1762429/FULLTEXT01.pdf.

Google. n.d.. Sandbox. *Chromium Docs.* [Online] n.d. [Citace: 07. 07 2025.] https://chromium.googlesource.com/chromium/src/%2B/refs/heads/lkgr/docs/design/sandbox.md.

—. n.d.. WebGPU Technical Report. *Chromium Docs.* [Online] n.d. [Citace: 07. 07 2025.] https://chromium.googlesource.com/chromium/src/+/main/docs/security/research/graphics/webgpu_technical_report.md.

Graz University of Technology. 2018. Meltdown and Spectre. *https://meltdownattack.com/.* [Online] 2018. [Citace: 06. 07 2025.] https://meltdownattack.com/.

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. [Citace: 04. 07 2025.] https://www.mozilla.org/en-US/firefox/110.0/releasenotes/.

PlayCanvas. 2025. Developer Guide. [Online] 2025. [Citace: 01. 07 2025.] https://developer.playcanvas.com/.

—. 2025. PlayCanvas. *GitHub.* [Online] 2025. [Citace: 07. 07 2025.] https://github.com/playcanvas.

Tatarchuk, Natalya. 2006. Dynamic Parallax Occlusion Mapping with Approximate Soft Shadows. *gitea.yiem.* [Online] 2006. [Citace: 07. 07 2025.] https://gitea.yiem.net/QianMo/Real-Time-Rendering-4th-Bibliography-Collection/raw/branch/main/Chapter%201-24/%5B1742%5D%C2%A0%5BSIGGRAPH%202006%5D%20Dynamic%20Parallax%20Occlusion%20Mapping%20with%20Approximate%20Soft%20Shadows.pdf.

The Khronos Group. 2025. Vulkan® 1.4.320 - A Specification. *khronos.org.* [Online] 2025. [Citace: 01. 07 2025.] https://registry.khronos.org/vulkan/specs/latest/pdf/vkspec.pdf.

—. 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/.

three.js. 2025. three.js. *GitHub.* [Online] 2025. [Citace: 07. 07 2025.] https://github.com/mrdoob/three.js.

W3C. 2024. W3C Invites Implementations of WebGPU. *W3C News.* [Online] 2024. [Citace: 01. 07 2025.] https://www.w3.org/news/2024/w3c-invites-implementations-of-webgpu/.

—. 2025. WebGPU API — Latest Working Draft. *w3.org.* [Online] 2025. [Citace: 20. 06 2025.] https://www.w3.org/TR/webgpu/.

—. 2025. WebGPU Explainer. *GPU FOR THE WEB COMMUNITY GROUP.* [Online] 2025. [Citace: 02. 07 2025.] https://gpuweb.github.io/gpuweb/explainer/.

WebGPU Experts. 2024. The Best of WebGPU in September 2024. *WebGPUExperts.com.* [Online] 2024. [Citace: 03. 06 2025.] https://www.webgpuexperts.com/best-webgpu-updates-september-2024.

Seznam tabulek

Seznam obrázků

Seznam ukázek kódů