|  |  |  |
| --- | --- | --- |
| **Univerzita Hradec Králové**  **Fakulta informatiky a managementu**  **Katedra Informatiky a kvantitativních metod** | | |
| **Vizualizace digitálních dvojčat s technickými řezy v Three.js**  Bakalářská práce | | |
| Autor: Šimon, Žanta  Studijní obor: Aplikovaná informatika | | |
| Vedoucí práce: Ing., Jakub, Beneš | | |
| Hradec Králové | měsíc rok | |
| Prohlášení:  Prohlašuji, že jsem bakalářskou/diplomovou práci zpracoval/zpracovala samostatně a s použitím uvedené literatury. | | | |
| V Hradci Králové dne 10.11.2024 | | *vlastnoruční podpis*  Jméno a Příjmení | |
| Poděkování:  Děkuji vedoucímu bakalářské/diplomové práce titul, jméno, příjmení za metodické vedení práce a…. | | | |

Abstrakt

Text abstraktu – shrnuje cíl práce, její metodiku, výsledky v ní dosažené a jejich diskusi. Délka minimálně 100 a maximálně 200 slov. Abstrakt je vhodné sepsat až po celkovém dokončení práce a má výstižně charakterizovat celou práci včetně jejich výsledků.

Abstract

Title: Visualization of digital twins with technical cut in Three.js

Abstrakt v anglickém jazyce. Délka minimálně 100 a maximálně 200 slov. Pro označení kvalifikační práce v anglickém jazyce se používají výhradně překlady „Bachelor Thesis“ pro bakalářskou práci a „Diploma Thesis“ pro práci diplomovou.

Klíčová slova: Three.js, shader, norma, technický řez, TypeScript, digitální dvojče

Key words: Three.js, shader, norm, technical cut, TypeScript, digital twin

Obsah

[1 Úvod 1](#_Toc147739360)

[2 Cíl a metodika práce 2](#_Toc147739361)

[3 Kapitola - Vlastní text práce 3](#_Toc147739362)

[3.1 Podkapitola 3](#_Toc147739363)

[3.1.1 Podřazená podkapitola 3](#_Toc147739364)

[4 Shrnutí a diskuse výsledků 4](#_Toc147739365)

[5 Závěry a doporučení 5](#_Toc147739366)

[6 Seznam použité literatury 6](#_Toc147739367)

[7 Přílohy 7](#_Toc147739368)

[*8* Zadání práce z IS (eVŠKP) 1](#_Toc147739369)

Obsah práce – strany práce jsou číslovány arabskými číslicemi tak, že kapitola

"Úvod" začíná na straně 1. Je možné číslovat i předchozí strany, v tom případě se použijí např. římské číslice I, II, III. Strany přílohy práce se nečíslují a nezahrnují se tak do celkového počtu stran práce.

# Úvod

V aktuální době vývoje a implementace industry 4.0 je čím dál tím větší snaha digitalizovat různorodé oblasti vývoje.

Jednou z těchto oblastí je například analýza vývoje a výkonosti výrobních linek. Jelikož samotná analýza je vcelku složitá, to tedy zejména z důvodu ohromného objemu dat, které z linek chodí, je snaha tuto analýzu digitalizovat a ucelovat v rámci komplexnějších aplikací.

# Cíl a metodika práce

Cílem práce je vyřešení určitého zvoleného problému, podložené získanými teoretickými poznatky. Toto propojení obecných poznatků a vlastním výzkumem/vývojem získaných poznatků je součástí popisu cílů práce. Při formulací cíle je vhodné používat tzv. aktivní slovesa (např. vytvořit, porovnat, stanovit, zpracovat apod.)

Zvolená metodika musí být adekvátní řešenému problému. Proto je vhodné vysvětlit, proč si autor zvolil právě konkrétní přístup, a případně také vysvětlit, proč ne nějaký jiný. Dále je vhodné si zároveň uvědomit a popsat možnosti a omezení zvoleného přístupu k problému. Součástí metodiky je popis práce se zdroji a způsob koncipování teoretické části práce. Využití nástrojů umělé inteligence při tvorbě práce je možné, avšak pouze za předpokladu důkladného popsání metodiky jejich použití. Pokud je součástí práce empirický výzkum, musí být uveden jeho charakter (kvalitativní, kvantitativní). U kvantitativního výzkumu je nutné zvážit cílové skupiny šetření a velikost a strukturu reprezentativního vzorku.

Během zpracování bakalářské/diplomové práce se může vyskytnout řada potíží, některé otázky nemůže autor rozpracovat vůbec, naopak jiné musí dodatečně do zpracování zahrnout. Toto je pak nutné rovněž popsat v metodice práce.

# Teoretická východiska

## Grafické API pro web

Pro programování jakékoliv grafiky je třeba získat přístup k prostředkům GPU (graphical proses sing unit). Pro desktopové aplikace je možné využít OpenGL (open graphical library), které vytváří abstrakci, tzv API. Pro web je třeba použít podobné API, které staví nad OpenGL.

### WebGL

WebGL je nejznámější JavaScript API pro renderování 3D a 2D objektů do HTML <canvas>. WebGL je úzce spjato s OpenGL, tedy API pro komunikaci s HW grafických karet. Toto spjatí s OpenGL umožňuje rychlý render time i v rámci prohlížeče. (WebGL: 2D and 3D graphics for the web).

WebGL tedy umožňuje práci s abstrakcí nad HW za pomocí zmiňovaného OpenGL. Díky tomu je práce s tímto API zjednodušená a není třeba aby uživatel byl seznámen s HW prostředky. WebGL využívá synchronní přístup k programování, což může způsobovat zpomalení. (From WebGL to WebGPU, 2023)

WebGL má v rámci webu různé použití. Nejzřejmější je renderování 3D a 2D scén. Nicméně se také využívají např. pro urychlení operací při renderování, a to třeba při. nahrazení renderování grafů do SVG. Hlavní výhodou využití grafů a frameworků, které renderují grafy pomocí WebGL je nesrovnatelná rychlost vyrenderování a často snadná implementace zobrazování streamu dat.

### WebGPU

WebGPU je aktuální nástupce WebGL, z něj dědí většinu základních funkcionalit a staví na nich. Má za úkol vylepšovat WebGL např. v nahrazení synchronního přístupu za asynchronní.

WebGPU má větší přístup k prostředkům GPU, to má za následek větší možnosti. To také umožňuje využívat GPU i k jiným úkonům než renderování a to např. složité výpočty, či simulace. K tomu je možné využít computed shader. Tento speciální shader se paralelně zpracovává na stovkách až tisících vláknech. (WebGPU API, 2024 )

To vše má však i nevýhodu, a to že WebGPU je teprve ve své aktivní fázi vývoje a není zdaleka tak podporovaný jako WebGL. Další nevýhodu je větší komplexnost v programování na této platformě a zejména z důvodu potřeby větší znalosti a zkušenosti s víc low-level programováním.

### Shadery

V moderním přístupu programování grafiky se již nevyužívá zastaralý způsob programování pomocí fixní pipeline, nicméně pomocí moderních shaderů. Fixní pipeline se již nevyužívá od verze OpenGL 3.0, aktuálně se tomuto přístupu říká *legacy OpenGL.* Specifické funkce legacy OpenGL je možné najít i v novějších verzích moderního OpenGL, nicméně dle oficiální dokumentace není vhodné jej již využívat.

Hlavními důvody, proč se vyhnout při vývoji legacy OpenGL a místo něj využívat shadery je hned několik. Nejzásadnějším důvodem je nejspíš výkon aplikace, to zejména díky způsobu předávání vertex atributů, tedy setu vlastností jednotlivých bodů na daném 2D nebo 3D modelu. Ten se ve fixní pipeline GPU předával, každý frame, to je s využitím moderního přístupu zjednodušeno a díky shaderům je možné tyto vlastnosti předat pouze jednou. Dalším důvodem je flexibilita využití shaderů a to zejména v možnosti předávání malých programů do GPU, kde je díky ohromnému výkonu možné rychle a real-timově zpracovat, tyto programy se nazývají shadery. Díky těmto shaderům je možné vytvářet mnohem komplexnější vizuální efekty, a to například osvětlení. Veškerý shader code se píše pomocí speciálního jazyku GLSL. (Legacy OpenGL, 2016)

#### A screenshot of a computer screen Description automatically generated Rendering Pipeline

Pro renderování pomocí shaderů se využívá tzv. *Rendering Pipeline,* ta funguje na podobném principu jako *Fixní Pipeline,* tedy v různých krocích této pipeline se zpracovávají různé části a shadery.

Prvním krokem je zpracování vertexů, které se prvně nahrají, následovně tyto nahrané vertexy projdou [Vertex Shaderem](#_Vertex_Shader). Tam se jednotlivé vertexy jednotlivě zpracují. Z Vertex Shaderu mohou putovat data do [Tessalation Shaderu](#_Tessalation_Shader) a [Geometry Shaderu](#_Tessalation_Shader). Tyto shadery jsou nepovinné a mají za úkol upravit nebo jinak manipulovat s grafickými primitivy. Následuje Post-processing vertexů, v něm jde hlavně o přípravu na rasterizaci. Po rasterizaci následuje zpracování posledního shaderu, tedy [Fragment Shader](#_Fragment_Shader), ten zpracovává tzv. fragmenty a určuje, jakou barvu budou mít. Po výstupu z Fragment shaderu nastávají pouze Per-Sample operace, které mají za úkol např. blending barev, nebo depth testing. (Rendering Pipeline Overview, 2022)

Obrázek https://www.khronos.org/opengl/wiki/File:RenderingPipeline.png

#### Vertex Shader

Jak již jejich název napovídá vertex shadery se zabývají manipulací s jednotlivým vertexy. Je to jeden ze dvou základních shaderů, který se v moderní grafice používá. Princip je takový, že na inputu dostane jeden vertex, ten následovně zpracuje a na výstupu dopět dostane jeden zpracovaný vertex. Je třeba aby byl zachován poměr 1:1 vstupních a výstupních vertexů. Také je možné je využít k pre-processingu, aby následující shadery mohli dělat další operace.

Tímto pre-procesingem je většinou myšleno to, že vertex shader umožňuje dopočítání dalších atributů, například dopočítání normál pro jednotlivé vertexy následovně slouží k výpočtům souvisejících s osvětlením či normálovými texturami. (Vertex Shader, 2017)

#### Tessellation Shadery

Tessellace je rozdělování shluků vertexů na menší primitiva, to má za následek zvýšení detailu daného objektu. Proces tessellace má 3 části, z nich jsou dvě provedené pomocí shaderů, mezi nimi je fixní funkce. První částí je *Tessellation Control Shader* (TCS), ta určuje, kolik tesselace se má provést, zároveň hlídá, aby úroveň tesselace byla mezi všemi objekty stejná, díky tomu je zachována celistvost objektů a zabraňuje vytváření mezer. Tato fáze je nepovinná, je možné využít výchozí hodnoty tessellace. Následuje fáze generování primitiv. V této fázi se vezme výstup z TCS a za jeho pomocí se vytvoří nová primitiva. V poslední fázi *Tessellation Evaluation Shader* (TES) se vezmou vygenerovaná primitiva a propočítá se pro ně hodnoty vertexů. (Tessellation, 2020)

#### Geometry Shader

Nepovinný shader, který slouží k manipulaci a vytváření grafických primitiv. Nastává po zpracování Vertex shaderu. Jako vstup využívají jedno grafické primitivum a na výstupu jich může být libovolný počet počínaje nulou. Příklady těchto primitiv jsou například points, lines či triangles. Častým využitím geometry shaderů může být například debugování normál. Jelikož je množné ke každému vertexu přidat kolmici a díky ní následovně vidět, jestli se normály vypočítávají správně. Nicméně není vhodné využívat tento typ shaderu pro generování komplexnějších tvarů. (Geometry Shader, nedat)

#### Fragment Shader

Fragment shader je shader, který se využívá k zpracování fragmentu na set barev. Pozice každého fragmentu je již ve window space. Tento shader v pomyslné pipeline zpracování shaderů navazuje na Vertex Shader, z toho důvodu obsahuje veškerý z něj získaný výstup. Výpočet tohoto shaderu nicméně nastává až po rasterizaci. Jejich výstupem je tedy set barev. Jejich největší zastoupení je například při vytváření výpočtů pro světlo. Pokud nebude existovat výstup z fragment shaderu, bude barva nedefinovaná. (Fragment Shader, 2020)

#### Compute shader

Speciální typ shaderu, který narozdíl od ostatních shaderů neslouží primárně k renderování, ale k provádění různých výpočtů. Má také úplně jiný přístup ke zpracování. Na rozdíl od normálních shaderů, které se spouští v nějakém, předem definovaném okamžiku, se tyto shadery spouští v závislosti na tom, kde jsou implementovány. Z toho důvodu mají také vlastní definovaný „prostor“. Jako vstupy jim neslouží nějaké předdefinované vstupy, nýbrž je potřeba aby si je každý shader získal. To může udělat pomocí vstupu do textury nebo třeba blok pro ukládání shaderů, další možností je také si pro to vytvořit vlastní Interface. (Compute Shader, 2019)

## Framework Three.js

### Frameworky pro práci s webGL

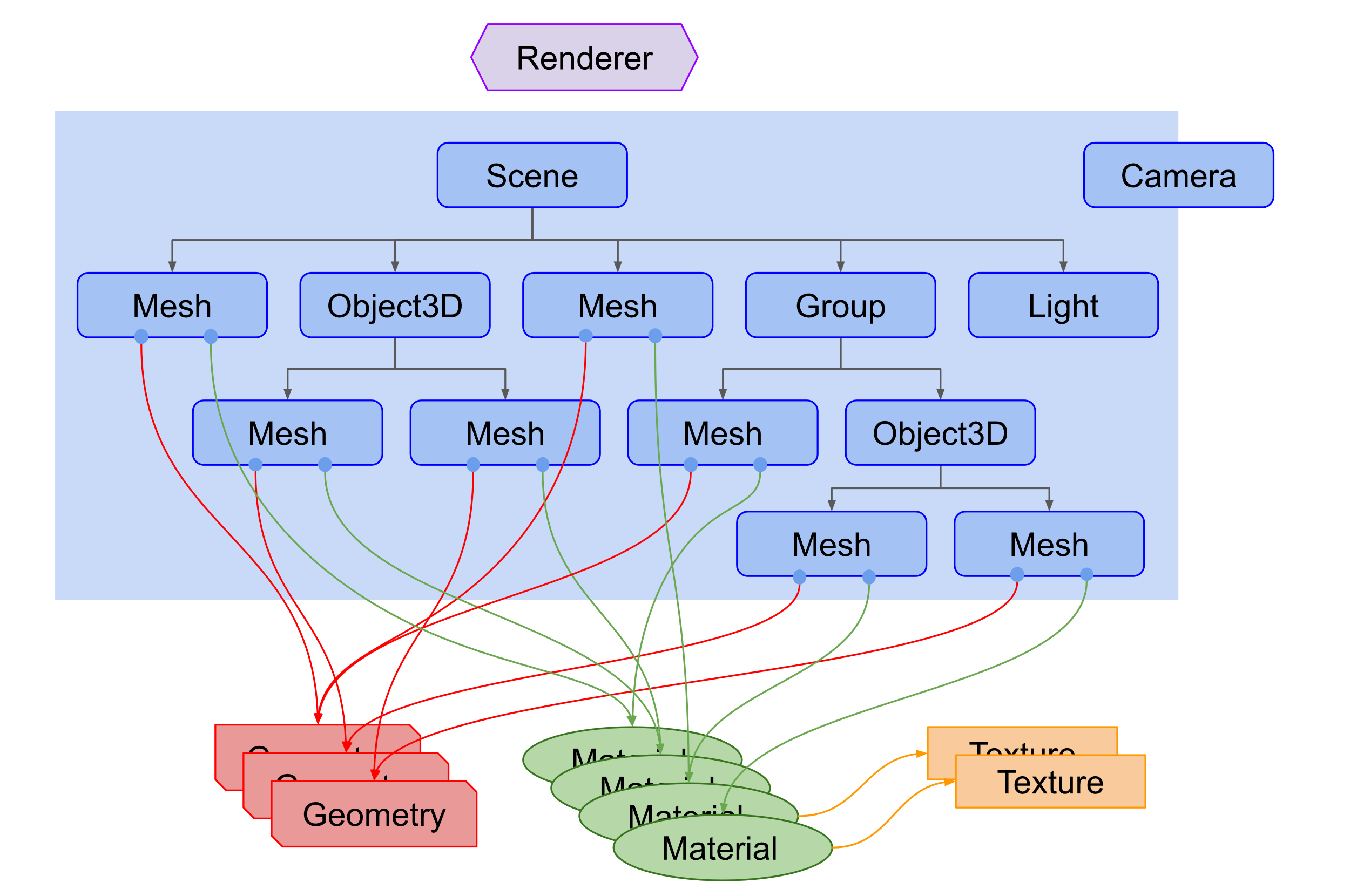
Jelikož může být pro část vývojářů složité nebo zbytečně zdlouhavé naučit se se samotným webGL vznikly frameworky. Ty tuto práci značně usnadňují a umožňují rychlejší vytvoření uceleného programu. Jejich úkolem je zejména usnadnit určité specifické problémy, či repetitivní úkony. Každý z těchto frameworku povětšinou implementuje nějakou formu vizualizační pipeline a k ní nějaký druh vlastní funkcionality. To je například lepší podpora VR či AR a to za pomocí frameworku A-Frame, nebo třeba podpora pro snazší vizualizaci grafů ve frameworku D3.js. Nicméně pokud není potřeba nějaké speciální funkcionality, je možné využít obecný framework jako je například právě framework Three.js. (WebGL-WebGPU-frameworks-libraries, 2017)

### Framework obecně

Framework Three.js je asi nejznámější framework pro práci s webGL. Implementuje spoustu základních i pokročilých funkcí z webGL, tedy se stará o scénu, materiály a textury, osvětlení a stíny a shadery. Je napsaná v jazyku JavaScript, nicméně má i plnou podporu TypeScriptu pro hlídání typování. Díky tomu je tato knihovna podporována většinou moderních prohlížečů.

### Renderer

Nejpodstatnější část jakéhokoliv grafického programu je renderer, jelikož ten vše vizualizuje. Hlavními součástmi rendereru je scéna a kamera. V rámci threejs se využívá WebGLRenderer. Tento renderer umožňuje například i buid in funkce pro optimalizaci zvanou culling. (fundamentals, nedat)



Obrázek https://jakearchibald.github.io/svgomg/ https://threejs.org/manual/resources/images/threejs-structure.svg

#### Scéna

Scéna slouží jako takový obal pro všechny objekt, které se budou připravovat na vykreslení nebo v ní budou pouze uloženy. Obsahuje tedy základní útvary ve formě mesh, tedy povrch, který je vytvořen pomocí základních geometrických útvarů, nejčastěji tedy z trojúhelníků. V rámci mesh je také možné definovat materiál a texturu, které definují vzhled daného mesh. Dále Object3D, což je ve své podstatě uskupení meshů do celistvého objektu. Pro zjednodušení operací s každým z těchto objektu mají také naimplementováno několik podpůrných funkcí pro snazší manipulaci v rámci scény. Tyto objekty je možné uskupovat do skupin a s těmito skupinami manipulovat. Poslední podstatnou částí scény je světlo, to osvětluje scénu a díky ní je možné vidět materiály a textury v zamýšlené formě. (fundamentals, nedat)

#### Kamera

Kamera je druhá hlavní součást rendereru. Její hlavní účel je promítnutí scény na display zařízení, to za pomocí převodu 3D scény do 2D. Další úlohou je také pohyb ve scéně. Dva hlavní typy kamer jsou perspektivní a ortografická kamera.

Perspektivní kamera je častěji vyžívaný typ kamery, přičemž jsou schopné zachovat perspektivu v rámci scény.

Ortografická kamera je typ kamery, který se využívá, pokud je potřeba aby byla zachována velikost objektů ve scéně a aby se tedy s perspektivou neměnila jejich velikost.

Dalšími typy kamer jsou například stereo kamera, ta kombinuje dvě perspektivní kamery a umožňují vytváření 3D efektu pro diváka, např v kině. Dalším příkladem je například Cube kamera, která vyrenderuje 6 kamer, které se následně pak využívají k vytvoření lepších odrazů. Pokud chceme využít více kamer ve scéně můžeme využít array camera, díky které je možné jich renderovat více. (kamera, nedat)

## ISO 128-3

### ISO normy

# Shrnutí a diskuse výsledků

Obsahuje souhrn a kritickou diskusi vlastních výsledků, získaných v průběhu řešení problému (soulad výsledků s literaturou či předpoklady; výsledky a okolnosti, které zvláště ovlivnily předkládanou práci, nové poznatky včetně jejich odvození ze získaných výstupů atd.).

# Závěry a doporučení

Závěr představuje shrnutí, do jaké míry a jakým způsobem byly naplněny plánované cíle práce. Je vhodné naznačit i případné další (popř. alternativní) možnosti zkoumání dané problematiky a otevřené problémy pro další navazující práce.

# Seznam použité literatury

# References

From WebGL to WebGPU. (2023, September 19). Retrieved from Chrome for Developers: https://developer.chrome.com/blog/from-webgl-to-webgpu

Legacy OpenGL. (2016, 5 11). Retrieved from khronos: https://www.khronos.org/opengl/wiki/Legacy\_OpenGL

WebGL: 2D and 3D graphics for the web. (2024, September 28). Retrieved from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/WebGL\_API

WebGPU API. (2024 , August 2). Retrieved from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/API/WebGPU\_API

Zdroje jsou v seznamu zdrojů na konci práce seřazeny v pořadí dle použité metody odkazování (viz dále), bez třídění podle typu zdrojů. Citace a jejich vyznačení do textu se řídí postupem podle ČSN ISO 690 Bibliografické citace z roku 2022 (<https://citace.zcu.cz/>), případně normou APA (<https://apastyle.apa.org/>).

Pro odkazování na zdroje v textu se používá buď Harvardský citační styl „autor/organizace (rok)“, nebo metoda číselných odkazů formátu „[1]“, nebo „(1)“, v závislosti na zvyklostech ve zkoumané oblasti. Seznam zdrojů na konci práce (bibliografie) se v prvním případě (autor rok) řadí zpravidla abecedně, v druhém případě (číslované odkazy) zpravidla dle pořadí výskytu. Pro systematickou správu citací je možné použít některý z dostupných softwarových nástrojů (např. Citace PRO Free (https://www.citace.com/vytvorit-citaci), Zotero (<https://www.zotero.org/>), nebo v MS Word panel Reference/Citace a bibliografie). Citační styl může být stanoven jednotně pro celý studijní program/specializaci. Student se řídí instrukcemi určenými vedoucím práce. Odkaz na zdroj v textu musí být umístěn tam, kde se začíná pracovat s převzatou informací, nezávisle na tom, jde-li o parafrázování či přímou citaci. U obrázků, schémat, tabulek, které vytvořil autor závěrečné práce na základě vlastního výzkumu, je uvedeno zdroj: vlastní. U vlastních fotografií je uvedeno zdroj: autor, případně odkaz na archiv autora, popsaný v seznamu zdrojů.

Důležitá poznámka: Závěrečné práce jsou po odevzdání automaticky předávány ke kontrole v anti-plagiátorském systému. Protokol o této kontrole je k dispozici vedoucímu i oponentovi práce. Uvedení myšlenky někoho jiného bez řádně provedené citace využitého díla (článku, knihy, dokumentu, zdroje na webu apod.) je trestné (autorský zákon). U bakalářské/diplomové práce je takovéto zjištění důvodem k jejímu automatickému odmítnutí a je to také považováno za disciplinární přestupek. Pokud bylo v práci zjištěno větší množství takových pochybení, je toto zjištění postoupeno disciplinární komisi.

# Přílohy

U každé přílohy musí být v pravém horním rohu uvedeno číslo přílohy (např. Příloha

č. 4). Přílohou bakalářské/diplomové může být i datové médium (CD, DVD, …), které

je nutné vložit do vhodné obálky připevněné k zadní desce práce a popsat také jako

přílohu v textu. Také v přílohách musí být korektně citovány použité zdroje informací. Digitální přílohy v podobě softwarových projektů mohou být také přiloženy formou odkazu na příslušný cloudový repositář projektu (Github, Gitlab apod.).

# Zadání práce z IS (eVŠKP)

„Zadání práce“, které každý student projednal před započetím přípravy závěrečné

práce se svým vedoucím práce a vložil do STAGu, je součástí odevzdávané elektronické (popř. tištěné, je-li to požadováno) verze závěrečné práce. Zadání práce student získá z informačního systému eVŠKP (<https://ris.uhk.cz/evskp/>).