Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Zobrazení 3D modelů s využitím WebGL

Bakalářská práce

Autor: Jan Kváš
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Milan Košťák

Hradec Králové duben 2022

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 29.4.2022 Jan Kváš

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Milanu Košťákovi za metodické vedení práce, za odbornou pomoc, praktické připomínky a trpělivost během vzniku této práce.

Anotace

Bakalářská práce se zabývá prozkoumáním možností zobrazení 3D modelů na webu pomocí WebGL a dalších knihoven. V práci jsou popsány techniky zobrazování 3D objektů. Dále jsou popsány druhy formátů souborů obsahující zakódovaná data o 3D modelech a OBJ formát je prozkoumán detailně. Jsou popsány a porovnány knihovny Three.js a Babylon.js, které slouží k zobrazení 3D modelů na webu. V praktické části je popsáno, jak se dají modely zobrazovat za využití kombinace jazyků JavaScript a GLSL nebo za využití zmíněných knihoven. Byl vytvořen vlastní algoritmus pro načtení 3D modelu, jehož data jsou uložena v OBJ souboru. Dále bylo provedeno podrobné otestování možností a schopností knihoven Three.js a Babylon.js. Na základě testování byla pro implementaci zvolena knihovna Babylon.js.

Klíčová slova: WebGL, 3D, Three.js, Babylon.js, GLSL, JavaScript, renderování

Annotation

Title: Rendering 3D models in WebGL

The bachelor thesis deals with the study of how 3D models could be displayed on the web using WebGL and auxiliary libraries. The work describes the techniques for displaying 3D objects. The types of file formats containing encoded data about 3D models are described, and the OBJ format is discussed in detail. The auxiliary libraries Three.js and Babylon.js are described and compared. The practical part describes how 3D models can be displayed using a combination of JavaScript and GLSL languages or using auxiliary libraries. A custom loader has been written to load a 3D model whose data is stored in an OBJ file. Detailed testing of the possibilities and capabilities of libraries Three.js and Babylon.js was performed. Library Babylon.js was chosen for the implementation based on the results of the tests.

Keywords: WebGL, 3D, Three.js, Babylon.js, GLSL, JavaScript, rendering

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

WebGL	- Web Graphics Library
DOM	- Document Object Model
API	- Application programming interface
HTML	- HyperText Markup Language
ES	- Embdded Systems
2D	- Two Dimensional
3D	- Three Dimensional
GLSL	- OpenGL Shading Language
NURBS	- Non-Uniform Rational B-spline
AR	- Augmented Reality
VR	- Virtual Reality
GPU	- Graphics Processing Unit
CPU	- Central Processing Unit
AJAX	- Asynchronous JavaScript And XML
OOP	- Objektově orientované programování
CAD	- Computer Aided Design
CAM	- Computer Aided Manufacturing
IDE	- Integrated Development Environment
XML	- Extensible Markup Language
NPM	- Node package manage
CDN	- Content Delivery Network
FPS	- Frames per second

1 Úvod

Dnes jsou webové stránky používány kromě poskytování informativního obsahu, také čím dál častěji k prezentaci interaktivního grafického obsahu nebo používají 3D grafické efekty k zaujmutí koncového uživatele a potenciálních zákazníků.

S příchodem WebGL a postupným vývojem grafických karet se množství grafického obsahu stále zvyšuje. Na trhu se začaly objevovat knihovny usnadňující zobrazování a tvoření komplexnějších a barvitějších 3D scén. Tím se ale také zvyšují nároky na technologie související s vývojem webových aplikací.

S vyššími nároky na grafiku a zobrazování 3D scén na webových stránkách se zároveň také zvyšují nároky na technologie související s vývojem webových aplikací. Tyto nároky by nám měly usnadňovat knihovny, které se touto problematikou zabývají.

Největší nároky jsou kladeny na grafický hardware, který je určený k výpočtům a zpracovávání grafických dat. Proto se zpracování grafiky přesunulo na specializovaný hardware a tím je grafická karta. WebGL využívá tzv. shaderové programy pro to, aby se grafické výpočty zpracovávaly právě na grafické kartě. Programovatelné shaderové programy umožňují optimalizovat funkcionalitu dle potřeb aplikace a tím lze docílit menších nároků na grafickou kartu.

Některé webové stránky používají zobrazování 3D scén jako svou základní funkcionalitu. Jedná se například o stránky s nábytkem, kdy je umožněno zákazníkovi manipulovat s objekty v prostoru, aby dostal lepší představy o rozměrech a vzhledu nábytku. Pro představu takové webové aplikace lze uvést například https://kitchen.planner.ikea.com/planner/#/nl/en/, kde si uživatel v rámci několika kroků vybere kuchyň, kterou by si představoval, a po dokončení mu je zobrazena výsledná 3D scéna. Dále lze technologii využít například při tvorbě počítačových her tvořených pro webové prostředí.

V teoretické části této práce je nejprve představena problematika vykreslování 3D objektů ve webovém prostředí s využitím WebGL. Dále jsou představeny dvě

knihovny, které jsou na WebGL založené – Three.js a Babylon.js. Znalosti z teoretické části jsou následně využity k návrhu a realizaci testového prostředí pro knihovny a také k zobrazení 3D modelů na webu. Dále jsou popsány formáty souborů, které se používají pro ukládání 3D modelů. Seznámení se s formáty a jejich kódování dat bylo využito pro naprogramování vlastního parseru jednoho z formátů.

2 Cíl práce

Cílem práce je prozkoumat problematiku a praktiky zobrazování 3D modelů pomocí WebGL a frameworků a následně implementovat webovou aplikaci, která bude modely zobrazovat pomocí popsaných technik a dále porovnat výhody a nevýhody daných technik a porovnání pomocných knihoven, které jsou určeny k zobrazování 3D modelů na webu.

3 WebGL

Web Graphics Library (WebGL) je technologie a standard pro 3D grafiku na webu umožňující vykreslování, zobrazování a interakci s 3D nebo 2D grafickými objekty prostřednictvím webového prohlížeče. Tento standard je vyvíjen a spravován organizací Khronos Group, která je neziskovou organizací zabývající se vytvářením otevřených standardů pro 3D grafiku, AR, VR, paralelní programování nebo také strojové učení. Zobrazování neboli renderování 3D objektů je výpočetně náročný úkol, který vyžaduje speciální, k tomu určený, hardware, kterým je nejčastěji grafická karta. Pro přístup k tomuto hardware se využívají speciální API, což je označení pro rozhraní určené pro programování aplikací. Jde o sbírku procedur, funkcí, tříd či protokolů nějaké knihovny, které může programátor využívat. API určuje, jakým způsobem jsou funkce knihovny volány ze zdrojového kódu programu. Typickými příklady grafických API jsou OpenGL a DirectX. [1; 2]

JavaScriptové API rozhraní pro zobrazování 3D grafiky bylo ve svých začátcích postavené na OpenGL ES 1.0. Postupným vývojem se v současné době API staví na OpenGL ES 2.0. Zkratka ES v anglickém jazyce znamená embedded systems, což můžeme přeložit jako vestavěné systémy. To znamená, že technologie OpenGL ES je původně určena primárně pro použití v malých výpočetních zařízeních, nejčastěji v mobilních telefonech či tabletech. Pro použití WebGL není nutná instalace jakéhokoliv plug-inu. Dnes je již kompatibilní se všemi významnými prohlížeči, což nebylo v minulosti samozřejmé. Dříve bylo nutné instalovat do prohlížeče požadované pluginy a rozšíření, aby bylo možné zobrazení interaktivní 3D scény. WebGL pomocí JavaScriptu umí pracovat s DOM (Document Object Model) elementy. [1; 2]

3.1 Technická specifika WebGL

Scénu objektů je možné vykreslovat do HTML elementu označeném značkou <canvas>. WebGL aplikace je sestavena z JavaScriptového kódu a shaderů. Konkrétně z vertex a fragment shaderů napsaných v GLSL jazyce. Zdrojový kód pro programování shaderů je obvykle zaobalen do datového typu string a je definován jako globální datová proměnná pro každý shader zvlášť. Nebo tento kód můžeme

napsat do textového souboru a následně ho načíst pomocí techniky AJAX (Asynchronous JavaScript And XML). Tato technika nám umožňuje asynchronní aktualizaci webových stránek výměnou dat s webovým serverem na pozadí. To znamená, že je možné aktualizovat části webové stránky, aniž by bylo nutné znovu načítat celou stránku.

Protože je WebGL rozhraní nízko-úrovňové, v němž je velice pracné a časově náročně napsat složitější scény, byla od jeho vydání v roce 2011 vytvořena celá řada knihoven a frameworků, které mají práci usnadnit, zpřehlednit kód a umožnit jednodušší vykreslování složitějších scény. Mezi takové frameworky patří například Three.js, Babylon.js a <model-viewer> .

3.2 Canvas

HTML element canvas je v oblasti grafiky a vizualizace velmi důležitý a mocný nástroj v kombinaci spolu s JavaScriptem. Umožňuje vývojářům tvořit animace a vizualizace přímo v prohlížeči bez použití podpůrných programů jako jsou např. Flash. Element canvas je jedním z řady tagů používaných ve značkovacím jazyce HTML. Tagy se zapisují pomocí špičatých závorek, např. <canvas>. Tagy mohou být párové a nepárové a mohou obsahovat různé atributy. Tagu canvas můžeme definovat např. identifikátor, šířku a výšku. Používají se i v jiných jazycích, zejména v XML (Extensible Markup Language), ve kterých vytvářejí vzhled a členění dokumentu. Díky tomu, že je tag přístupný skrze Document Object Model (DOM), můžeme pomocí JavaScriptu získat referenci na tento tag. To nám umožní získat tzv. kontext plátna. Kontext je klíčová vlastnost canvasu, díky které může vykreslovat 2D nebo 3D obrazce ve webovém prohlížeči podle toho, jaký typ prostoru se zvolí. Dnes již všechny moderní prohlížeče tento tag podporují. Canvas umisťujeme mezi párový tag <body>. JavaScriptový kód pro obsluhu kreslení v plátně se zpravidla umisťuje těsně před konec tagu <body> nebo do tagu <head>. Umístění na konec tagu <body> volíme z toho důvodu, že chceme, aby se načetla celá stránka dříve, než kód začne přistupovat k daným tagům, ostatně to chceme v obou případech. V případě umístění do tagu head, musíme ale nejdříve ověřit, že stránka je již

načtena, aby byl zajištěn správný běh kódu, než se příslušný kód začne vykonávat. Kód vložíme do tagu <script>. [3]

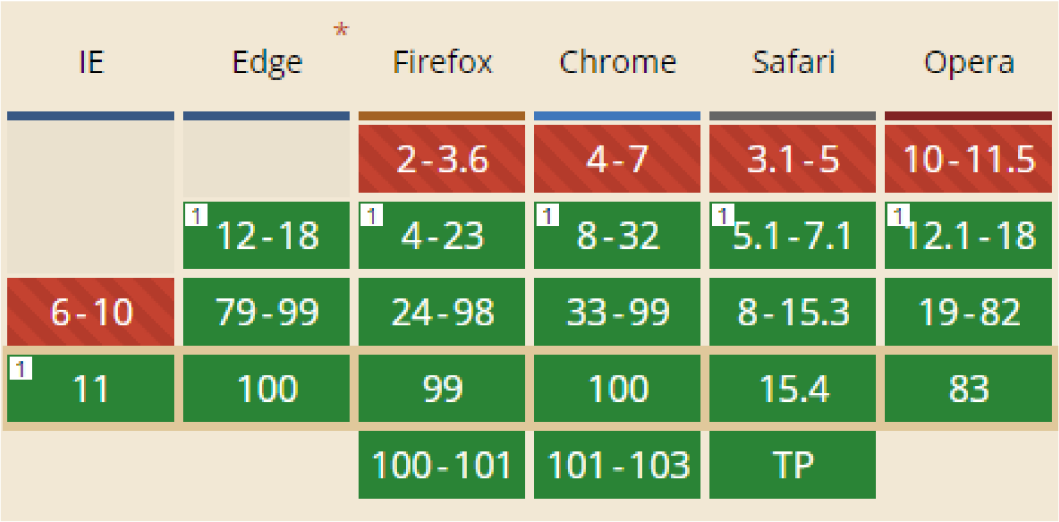
3.3 WebGL 1.0 a WebGL 2.0

Od doby, kdy byla vydána první verze WebGL uběhla řada let a tudíž v roce 2017 vznikl standard další verze WebGL, a to WebGL 2.0. Tato verze je vylepšením a rozšířením verze WebGL 1.0. [4; 5]

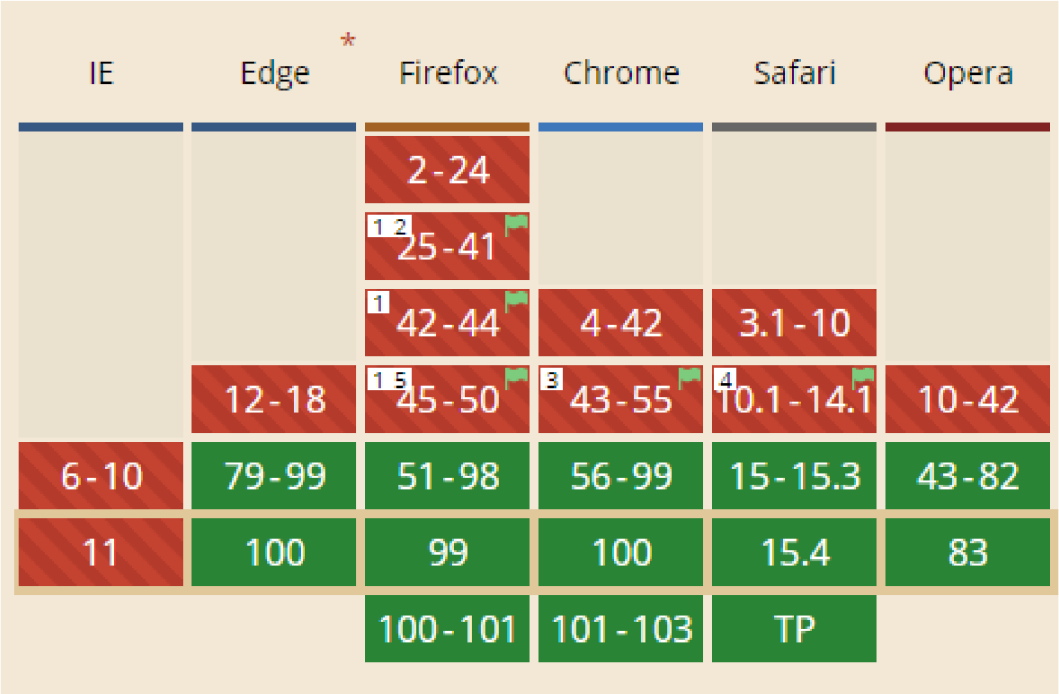
WebGL 2.0 nabízí jako vestavěné funkcionality mnoho rozšíření, které byly v předchozí verzi jako volitelná a musela se získat pomocí metody getExtension(), kde se jako argument metody vložil název chtěného rozšíření. Následně se mohla vytvořit instance tohoto rozšíření. WebGL 2.0 toto usnadňuje a díky tomu můžeme mnoho takových rozšíření využívat pohodlněji, protože jejich podpora je jistá. [4; 5]

Významným rozdílem mezi těmito verzemi je využití jiných verzí GLSL jazyka. Jak se vyvíjelo WebGL, tak se také vyvíjel GLSL jazyk. Novější verze WebGL využívá verzi GLSL 3.0 oproti starší verzi WebGL, která využívá verzi GLSL 1.0. To přináší změny, které souvisejí s odlišnostmi v syntaxi jazyka GLSL. Tudíž se budou lišit zdrojové kódy shaderů. Při využití GLSL 3.0 je nutné jako první řádek zdrojového kódu shaderového programu napsat textový řetězec #verison 300 es, tím je deklarováno využití této verze. Následně je potřeba změnit deklaraci datových typů proměnných. Tato změna se týká např. datových typů attribute a varying. Deklarace datového typu se změnila z attribute na in. Deklarace datového typu varying se změnila na in nebo out v závislosti na tom, v jakém shaderovém programu jsou deklarovány. [4; 5]

Není nutné se obávat vzájemné nekompatibility WebGL 1.0 a WebGL 2.0. Pokud při využití WebGL 2.0 budeme využívat pouze funkcionalit, které jsou ve WebGL 1.0 nebudou vznikat problémy, verze je zpětně kompatibilní. Novější verze GLSL 3.0 také přináší potíže s podporou prohlížečů. Z následujících obrázků zobrazující prohlížeče, které podporují WebGL 1.0 a WebGL 2.0, je patrné, že WebGL 1.0 je k dispozici téměř ve všech prohlížečích a jejich verzích. WebGL 2.0 takovou podporu nemá. [4; 5]



Obr. 1 Podpora WebGL 1.0 ve webových prohlížečích.
Zdroj: [6]



Obr. 2 Podpora WebGL 2.0 ve webových prohlížečích.
Zdroj: [7]

Dalším rozdíl je při deklaraci jejich využití při získávání kontextu elementu canvas. Po použití WebGL 1.0 se zadává řetězec „webgl“ jako argument metody getContext(). Oproti tomu pro použití WebGL 2.0 píšeme do argumentu téže metody řetězec „webgl2“. [4; 5]

4 OpenGL Shading Language

OpenGL Shading Language (GLSL) je vyšší programovací jazyk, který usnadňuje psaní zdrojového kódu. Jazyk GLSL má své kořeny v C a C++, tudíž má podobnou syntaxi. Každý program, podobně jako u jazyku C, musí obsahovat hlavní metodu. Tato metoda má název „main“. Metoda nemá žádné parametry ani nevrací žádnou hodnotu.

Kód v této metodě se provádí nad každým vstupním vrcholem. Parametry vrcholu se do programu předávají pomocí speciálních globálních proměnných. Hodnoty vstupních proměnných jsou nastaveny dle parametrů zpracovávaného vrcholu před spuštěním hlavní metody. V programu se vypočítají nové hodnoty, které se následně uloží do výstupních globálních proměnných. Název i typ těchto proměnných je již předdefinován a nelze je programově měnit.

Jazyk navíc obsahuje datové typy a funkce, které mají v sobě implementované algoritmy pro výpočty typické pro 3D grafiku a její zobrazení. Umí pracovat se základními datovými typy, jako jsou logický datový typ bool, celočíselný datový typ int nebo float, který je jedním ze zástupců datových typů pracujících s plovoucí desetinnou čárkou. Navíc ale přidává datové typy, kterými jsou například n-násobný vektor, matice nebo tzv. sampler umožňující přístup a práci s texturou. Kromě datových typů a funkcí obsahuje také další prvky. Mezi ně patří struktury a pole. GLSL programy nemohou fungovat samy od sebe, ale musejí být součástí většího OpenGL programu. [8; 9; 10]

Jazyk je nejprve přeložen a následně jeho binární kód je vykonávaný přímo na grafické kartě, tudíž je většina výpočetního vytížení přesunuta na grafickou kartu. Je základní a nedílnou součástí OpenGL API. Jakýkoliv program napsaný v GLSL jazyce a využívající OpenGL se nazývá shader. Tyto shadery jsou zpracovávány na grafické kartě. [11]

Shadery dělíme do několika druhů dle toho, v jaké části grafického řetězce jsou použity a co mají vykonávat. V základními typy jsou vertex shader a fragment shader. Mají za úkol zpracovat a zobrazit vizuální podobu programu. Tyto programy

jsou vykonávány na grafické kartě. Využívají se pro programování tzv. rendering pipeline, která nahradila tzv. fixed function pipeline nebo se můžeme setkat s českým termínem fixní pipeline. To nám umožní zpracování dat dle vlastní potřeby, možnost optimalizace zobrazovacího řetězce a možnost přidat vylepšení, či přidat nové funkcionality. [11]

4.1 Vertex shader

Vertex shader je program, který zpracovává každý vertex. Důvodem zpracování každého vertexu je transformace pozice nejčastěji násobením modelovací, pohledovou a projekční maticí. Pozice je stanovena ve 3D světě, tudíž má tři souřadnice x, y, z. Každý vertex je následně převeden do 2D souřadnic s hodnotou hloubky tak, aby byly správně vykresleny na obrazovce a dalo se správně určit, jak se objekty scény překrývají. Vertex shader může měnit vlastnosti vrcholu jako jsou pozice, barva, normály a světelné charakteristiky, ale nové vertexy vytvářet nemůže. Může pouze transformovat stávající vertexy nebo měnit jejich datové hodnoty. Výstup z vertex shaderu pokračuje do další části grafické pipeline. Základní proměnnou, která je předána dál z vertex shaderu, je gl_Position. Je to proměnná vycházející z jazyka GLSL a je v ní uložena výsledná pozice každého jednoho vertexu, který shaderem prošel. [12]

4.2 Fragment shader

Fragment shader, nebo se také můžeme setkat s označením pixel shader, je program, jehož primárním úkolem je vypočítat a přiřadit pixelu obrazovky barvu. Tento program je v grafické pipeline zařazen za vertex shaderem. Předtím než přijde na řadu fragment shader je provedeno ještě několik kroků jako jsou triangulace, ořezávaní a rasterizace scény. Fragment shader pracuje s tzv. fragmenty, což jsou objekty nesoucí v sobě informace o barvě, pozici nebo průhlednosti. Z fragmentu jsou následně tvořeny pixely jako výstup renderování. Fragment shader má přístup k pozici fragmentu, ale nemůže ji změnit. Může vypočítávat osvětlení, stíny, průhlednost, odrazivost a texturování. [13; 14]

5 Renderování

Renderování může být pochopeno z hlediska, kde se výpočty související se zobrazováním vypočítávají, jakou technikou je model zobrazen nebo jaké kroky jsou v zobrazovacím řetězci vykonány. Tyto způsoby a techniky jsou uvedeny v následujících podkapitolách.

5.1 Způsoby renderování

Renderování pomocí WebGL může být prováděno různými způsoby. Záleží na tom, jaký hardware je pro provádění výpočtů použit a zda se výpočet provádí na lokálním nebo vzdáleném zařízení. Metody související s otázkou, na kterém hardware jsou výpočty prováděny, jsou software renderování nebo hardware renderování. O software metodě lze hovořit v případech, kdy všechny výpočty jsou prováděny na CPU. V případě, že veškeré výpočty v reálném čase související s grafickým objektem, který chceme zobrazit, je prováděn na GPU, hovoříme o hardware metodě. Můžeme říci, že hardware metoda je efektivnější, protože výpočty jsou prováděny na hardware, který je tomuto uzpůsoben a vyroben. [15]

Metoda výpočtu prováděného na vzdáleném zařízení se dle autorů Farhada Ghayoura a Diega Cantora [16] nazývá server-based rendering. Oproti tomu metoda výpočtu na lokálním zařízení se dle stejných autorů nazývá client-based rendering. WebGL je naprogramováno tak, že využívá client-based metodu. WebGL v porovnání s ostatními technologiemi jako například Flash nebo Unity Web Player Plugin má tyto výhody:

- JavaScript – WebGL je napsáno v jazyku JavaScript, který se běžně využívá na webových stránkách, či aplikacích. Díky tomu je snadnější integrace s ostatními knihovnami napsanými v JavaScriptu např. jQuery.
- Automatický management paměti je výhoda, která taktéž vyplývá z použití jazyka JavaScript. Zajišťuje, že není nutné se starat o alokaci a dealokaci paměti.

- Rozšířenost na zařízeních – protože veškeré prohlížeče podporují JavaScript, je podpora WebGL na jakémkoliv zařízení, které disponuje webovým prohlížečem.
- Výkonnost – srovnatelná s ekvivalentními tzv. standalone aplikacemi, a to díky možnosti přístupu ke grafickému hardware na zařízeních, kde je WebGL spouštěno. Standalone aplikace jsou aplikace, které běží na lokálním zařízení a ke své funkčnosti nevyžadují mnoho dalšího než spuštěné zařízení. Veškerá logika je naprogramována v aplikaci, tudíž není vyžadováno např. připojení k internetu nebo instalace dalších aplikací či služeb.

WebGL má také i nevýhody a ty mohou být následující:

- Automatický management paměti – tato vlastnost byla zmíněna mezi výhodami, ale snadno může být brána i jako nevýhoda. Důvodem toho je, že dealokaci musí nějaký automatický proces obstarat. Nejčastěji tuto činnost uvolňování paměti má na starosti tzv. garbage collector, který však přispívá k větší výpočetní náročnosti.

- Nízko-úrovňové programování – WebGL je náročné jak na psaní kódu, tak na hlubší znalosti související s 3D grafikou, jejím zobrazováním a algoritmy s ní spojené. Pro zobrazení složitějších scén je zapotřebí napsat, v porovnání s pomocnými knihovnami a frameworky, více kódu.

5.2 Techniky zobrazování 3D modelů

Existuje mnoho technik vykreslování 3D grafiky. Tou bezesporu nejpoužívanější technikou je vykreslení prostřednictvím polygonové sítě. Další technikou vykreslování je technika pomocí NURBS křivek.

5.2.1 Polygonová síť

Polygonová síť je objekt, který je sestaven z jednoho nebo více mnohoúhelníků. Ty se skládají z vertexů definovaných v 3D prostoru pomocí Kartézské soustavy souřadnic x, y, z. Nejčastěji využívaným mnohoúhelníkem jsou trojúhelník nebo méně častěji čtyřúhelník. Tyto nejelementárnější plošné útvary pak tvoří povrch

modelu. Dělení povrchu modelu na mnohoúhelníky a jejich vyplněním roviny bez překrývání a bez mezer se nazývá teselace. Triangulací se pak nazývá dělení povrchu modelu konkrétním geometrickým útvarem a to trojúhelníkem [17]. Josef Pelikán [18] poukazuje i na možné problémy této techniky, když říká: „*Je třeba si uvědomit, že mnohostěn nemůže dokonale nahradit hladká tělesa – i u tak jednoduchého tělesa, jako je třeba koule, narážíme při aproximaci mnohostěnem na mnohé problémy. Obecně platí: čím přesnější (věrnější) aproximaci chceme dostat, tím větší počet trojúhelníků musíme použít. A tím pomalejší bude vykreslování modelu v grafické kartě.*“ Tato technika se používá častěji, protože není tak výpočetně náročná. Tudíž je renderování modelu rychlejší. Používá se tam, kde nejsou kladeny takové nároky na detailní a kvalitní zobrazení a tam, kde se scéna může překreslovat často.

5.2.2 NURBS křivky

Další technika zobrazování 3D modelů je pomocí NURBS křivek. Zkratka NURBS (Non-Uniform Rational B-spline – neuniformní racionální B-křivky) je označení pro typy parametrických křivek a ploch používaných v počítačové grafice. Jsou zobecněním jak Bézierových, tak tzv. B-spline křivek. Termín neuniformní znamená, že vzdálenosti uzlů nemusí být u těchto křivek konstantní tak, jako například u Bézierových křivek. Základ této techniky vychází z definice Bézierovy křivky. Vytvořená síť pomocí NURBS křivek je tvořena množstvím kontrolních bodů s různou váhou a množstvím tzv. uzlů. Pomocí kontrolních bodů a jejich váhy jsme schopni matematickou interpolací vypočítat povrch modelu a vykreslit mezi uzly zakřivení povrchu modelu [19]. Takto vytvořená plocha vypadá hladce i při jakémkoli zvětšování či zmenšování modelu. Z důvodu složitějších výpočtů trvá renderování modelu pomocí této techniky déle oproti technice polygonové sítě. Proto se vyhýbáme použití této metody v případech, kde se model nebo scéna často překreslují. Z důvodu složitějších výpočtů a případného častého překreslování bude kladena větší výpočetní náročnost pro hardware. Jejich oblibě se však dostalo v Computer Aided Design (CAD) a Computer Aided Manufacturing (CAM) systémech, ve kterých se používají dodnes.

5.2.3 Zobrazovací řetězec

Zobrazovací řetězec (Rendering Pipeline), je označení pro řadu kroků jdoucích po sobě, jejichž cílem je převést data o barvách, pozicích, světlech, stínech a dalších do vizuální podoby a vykreslit je na obrazovce daného zařízení. Zobrazovací řetězec je vykonáván na grafické kartě a nahrazuje tzv. fixní pipeline. Umožňuje optimalizaci zobrazovacího řetězce a přizpůsobení funkcionality shaderů dle vlastních potřeb. V případě WebGL se zpravidla upravují dva shadery a to vertex shader a fragment shader. [11]

6 Formáty pro uložení objektů

V oblasti 3D scén existuje množství formátů, ve kterých je možné model objektu uložit. Nejrozšířenějšími jsou například gLTF/GLB, FBX, OBJ, COLLADA. Jejich oblíbenost se liší na základě jejich vlastností a také v jaké oblasti se zrovna bude daný model využívat. Základní vlastností formátů je uložení informací o 3D modelu, jako jsou geometrie modelu, barva, okolní scéna, světla, textura, animace aj.

6.1 Dělení formátů

Jedním z možných dělení formátů je na otevřené a proprietární. Proprietární jsou vytvořeny komerční společností, která takový formát dále licencuje. Model kódování formátu je optimalizován pro daný způsob použití a lehčí spolupráci s dalšími firemními systémy. Na ostatní 3D zobrazovací a modelovací systémy nebo vývojová prostředí (IDE) poté firma nebere ohled. Na druhé straně stojí otevřené formáty, které jsou vyvíjeny nezávislou organizací nebo komunitou a jejich kompatibilita se zobrazovacími a modelovacími systémy bývá vyšší.

6.2 Konkrétní formáty

V následujících podkapitolách jsou podrobně popsány formáty gLTF/GLB, FBX, OBJ, Collada a jsou uvedeny jejich výhody a nevýhody.

6.2.1 gLTF/GLB

gLTF/GLB je open source formát, který patří do kategorie otevřených formátů. Byl vytvořen neziskovým konsorciem Khronos Group za účelem zobrazování 3D objektů na webu, AR, VR a v počítačových hrách. Formát podporuje geometrii, materiály, textury, barvy i animace. Obsahuje Physical Based Rendering (PBR), a tudíž stíny a světla zobrazuje více realisticky. gLTF je založen na JSON formátu. To znamená, že jsou data o modelu (textury, shadery, geometrie nebo animace) uložena i v jiných souborech. GLB je binární verzí gLTF formátu. Data oproti gLTF ukládá do jednoho souboru a je více využíván v oblasti virtuální reality. [20]

6.2.2 FBX

FBX (FilmBox) patří mezi proprietární formáty. Původně byl vyvinut firmou Kardara, ale následně byl odkoupen společností Autodesk. Jeho oblíbenost spočívá v podpoře animací, a také v možnosti použití jako přechodného formátu mezi různými vývojovými prostředími. Právě díky podpoře animací se často využívá pro vytvoření modelu těla, či jeho části, nebo postav do her. Proto je široce rozšířen ve filmovém a herním průmyslu. [20]

6.2.3 OBJ

Tento formát je jedním z otevřených 3D formátů. Vyvinutý byl firmou Wavefront Technologies. Má široké využití – lze ho použít pro 3D modely, 3D tisk a 3D skenování. Dokáže uložit barvy, geometrii a textury. Datové informace o barvách a texturách jsou obvykle uloženy v odděleném MTL souboru, který je distribuován s původním OBJ souborem. Objektové soubory mohou být ve formátu ASCII s příponou .obj nebo v binárním formátu s příponou .mod. Nevýhodou může být chybějící podpora animací, avšak i přesto patří mezi oblíbené formáty. Formát souboru s příponou .obj podporuje jak polygonální objekty, tak objekty volného tvaru. Polygonální geometrie používá k definování objektů body, čáry a plochy, zatímco geometrie volného tvaru používá křivky a povrchy. [20]

6.2.4 Collada

Toto zkratkové slovo znamená COLLAborative Design Activity. Je to formát pro ukládání 3D objektů a animací, který byl vytvořen firmou Sony Computer Entertainment. Je založen na XML schématu, z čehož vyplývá výhoda snadného přečítání, vytváření a editování v libovolném textovém editoru. Další výhodou využití XML schématu je jednoduchost přenášení dat mezi aplikacemi. Tento formát obvykle používá koncovku .dae. Nevýhodou může být větší velikost souboru a v některých případech delší doba exportování. [21; 22]

7 Knihovny a frameworky

V této kapitole jsou představeny některé z mnoha dostupných knihoven a frameworků, které pracují s grafikou na webu a jejím zobrazováním. Většina pomocných nástrojů je založena na OpenGL a JavaScriptovém API. Jejich použitím v aplikaci může dojít ke zpomalení aplikace, ale zato ušetří mnoho hodin času tím, že není nutné se zabývat nízko úrovňovým programováním. Právě to vyřeší daná knihovna nebo framework. Vytváření velkých 3D scén používáním pouze WebGL je celkem náročné bez použití některé z již naprogramovaných knihoven.

7.1 Three.js

Three.js je WebGL JavaScriptová open-source knihovna pro práci s 3D scénami. Poskytuje API rozhraní s velkým množstvím funkcí. Její vývoj začal v roce 2010. Knihovna je používána pro zobrazování 3D modelů, scén a animací přímo v prohlížeči. Díky tomu, že je celá naprogramována v jazyku JavaScript, nevyžaduje zvláštní instalaci plug-inu do prohlížeče. Sama také není nijak závislá na jiných knihovnách, ale pro její spuštění je nutná podpora WebGL v prohlížeči.

Three.js využívá objektově orientované programování. OOP je programovací styl psaní kódu v tzv. objektech. To jsou prvky odrážející prvky reality. Objekty si pamatují svůj stav a okolí poskytují operace. Ty jsou přístupné jako metody pro volání. Knihovna obsahuje podporu pro práci s širokou paletou 3D formátů. Knihovna je napsána tak, aby využívala nejlepší možná řešení pro výpočty a zobrazování. Není plně uzpůsobena pro vytváření propracovaných 3D her, přesto nabízí mnoho předem připravených metod a tříd umožňujících vytváření jednoduchých her, animací, vizualizací a mnoho dalšího. Poskytuje dobrou interakci s uživatelem a dobré mapování toho, kde se zrovna nachází kurzor. [1; 23; 24]

Three.js také pomáhá s matematikou, která je při vytváření grafických vizualizací klíčová. Obsahuje celou řadu definovaných matic, vektorů, kamer a dalších objektů. Three.js umí vykreslovat do 2D nebo 3D kontextu tagu canvas, či SVG elementu. [1; 23; 24]

Pro vytvoření základní scény potřebujeme vytvořit tři objekty. Těmito objekty jsou renderer, scéna a kamera. Pro vykreslení do SVG elementu je nutné použít specializovaný renderer. Scéna se vykreslí v prohlížeči tak, že je přidána scéna a kamera do metody rendereru, která scénu vykreslí. Pokud bude chtěno přidat nějaký objekt nebo světlo, je zapotřebí inicializovat příslušné objekty a přidat je do scény. [1; 23; 24]

7.2 Babylon.js

Babylon.js je druhým zkoumaným frameworkem. Jedná se o JavaScriptový framework, který používá jazyk TypeScript. TypeScript je kompilovaný a multiplatformní jazyk, který rozšiřuje JavaScript o statické typování, třídy, rozhraní a další věci známé z objektově orientovaného programování (OOP). Díky jeho transpilátoru generuje z jazyka TypeScript čistý JavaScript. Tento jazyk byl vytvořen společností Microsoft. Jak Babylon.js, tak i TypeScript jsou open-source technologie. Využití TypeScriptu bylo vývojářskou firmou zvoleno proto, že tento framework poskytuje mnoho funkcí a některé z nich mohou mít hodně parametrů, aby mohly být více flexibilní. TypeScript je vhodný pro velké aplikace s potenciálem rozrůstáním. Mezi další důvody patří fakt, že psaní takto velkého frameworku je s jeho využitím mnohem přehlednější, čitelnější a jednodušeji udržitelnější.

Framework lze využít pro modelování 3D aplikací i her. Významným faktorem pro zvolení tohoto frameworku pro programování 3D hry může být i to, že má vestavěnou podporu pro enginy modelující fyziku scény. Engine se využívá pro zobrazení a kontroly scény a také pro přímou komunikaci s grafickou kartou. Pro vykreslení scény 3D objektů musí být použit tag canvas, jehož kontext je následně předán do již zmíněného enginu. Ten objekty jako jsou kamera, světla a tělesa přiřadí scéně a následně je zpracuje. [25]

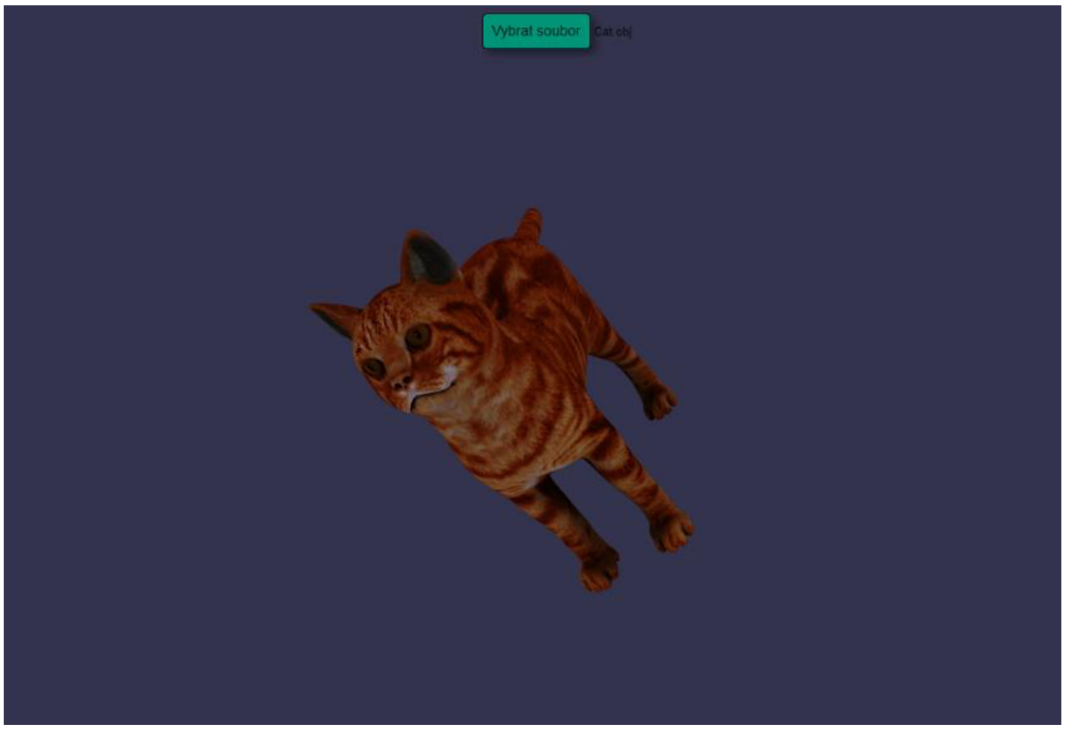
8 Implementace

8.1 Aplikace

Na počátku byly zvažovány dvě varianty, jak aplikaci napsat. Jedna byla za pomoci JavaScriptového frameworku, který by usnadnil práci s psaním logiky a interakce webové aplikace s uživatelem. Druhou variantou bylo psaní webové aplikace klasickým způsobem, a to čistým JavaScriptem.

Jako pomocný framework byl vybrán Vue.js, který je určen pro psaní uživatelských rozhraní webových aplikací. Je také dobře využitelný pro tvorbu single-page aplikací. Snadná integrace různých pomocných knihoven z něho dělá vhodný nástroj pro psaní interaktivních aplikací či rozhraní. Byl vybrán, protože je jedním z nejpopulárnějších front-endových frameworků. Dalším důvodem bylo, že dle průzkumu z článku „15 Best Front end Frameworks for Web Development in 2022“ [26] je tento framework jednoduchý pro naučení, je malý a rychlý a má detailní dokumentaci. Rozvoj aplikace je ve Vue.js snazší nežli v klasickém JavaScriptu a HTML. Integrace knihoven pro tvorbu 3D scén je velmi jednoduchá. Probíhá pomocí Node package manage (NPM), což je správce balíčků pro JavaScript a výchozí správce balíčků pro prostředí Node.js. S vývojem a zkoumáním knihoven pro tvorbu a zobrazování 3D modelů a scén je ale využití potenciálu tohoto frameworku menší. Z hlediska účelu této práce může být spíše možnou komplikací než pomocným nástrojem pro usnadnění psaní.

Vyzkoušeny byly obě varianty, ale následně byla vybrána varianta napsat aplikaci pomocí čistého JavaScriptu bez Vue.js. V průběhu práce se objevilo vícero problémů, se kterými se programováním v JavaScriptu nebylo třeba setkávat. Hlavním důvodem volby byla menší kompatibilita pomocných knihoven pro zobrazování 3D grafiky s frameworkem, než bylo očekáváno. Aplikace je tedy napsána bez využití Vue.js frameworku.



Obr. 3 Výsledný vzhled aplikace
Zdroj: vlastní zpracování

8.2 Implementace WebGL

Pro využití vykreslování scén je nutné do těla stránky tvořené značkovacím jazykem HTML vložit tag canvas. Následně z instance tohoto plátna získat kontext určený pro vykreslování 3D grafiky. Toho je docíleno tak, že do metody getContext() je vložen jako argument řetězec „webgl“. Dalším krokem je vytvoření shaderů. Jak již bylo zmíněno v předchozích kapitolách, k vytvoření je třeba sepsat zdrojový kód pro vertex shader a fragment shader. Zdrojový kód shaderů lze do stránky vkládat pomocí odděleného skriptu, jež se pak načítá pomocí identifikátorů, to znamená pomocí atributu id elementu script nebo je možné zdrojový kód shaderu vkládat jako textový řetězec přímo uvnitř skriptu. Třetí variantou je načtení externího souboru, který kód shaderu obsahuje. Shadery mají přístup ke třem druhům kvalifikátorů datových položek, z nichž každý má své specifické využití. Každý druh proměnné je přístupný buď jedním nebo oběma shadery v závislosti na tom, jaký druh kvalifikátoru je u proměnné zvolen. Druhy kvalifikátorů proměnných jsou tyto:

- Attribute – tímto kvalifikátorem označujeme proměnnou shaderu, jejíž hodnota se může lišit u každého vertexu. Tento kvalifikátor může být použit pouze ve vertex shaderu.
- Uniform – tímto kvalifikátorem se označuje proměnná shaderu, jejíž hodnota není měněna během zpracování primitiva. Tento kvalifikátor může být použit v obou shaderech.
- Varying – používá se pro interpolovaná data mezi vertex shaderem a fragment shaderem. Používá se pro možnost zapisování do proměnné ve vertex shaderu a pro čtení ve fragment shaderu.

Následně jsou shadery inicializovány, a to ve třech krocích. Prvním krokem je vytvoření shaderu metodou createShader(). Do této metody se vkládá argument odpovídající typu shaderu. Druhý krok je vložení zdrojového kódu. To se provede metodou shaderSource(). Do argumentů metody je vložena vytvořená instanci metody createShader() a zdrojový kód. Posledním krokem je kompilace shaderu. To zajistí metoda compileShader() s argumentem, který obsahuje taktéž instanci vytvořenou metodou createShader(). Příklad vytvoření shaderů:

```
var shader = gl.createShader(gl.VERTEX_SHADER);
gl.shaderSource(shader, sourceCode);
gl.compileShader(shader);
```

Obr. 4 Vytvoření instance shaderu při použití WebGL.
Zdroj: vlastní zpracování

Další nutnou součástí je vytvoření tzv. programu. V tomto kroku docílíme toho, že připojíme vytvořené shadery do programu, který následně bude vykreslovat scénu.

Další nezbytným prvkem je vytvoření bufferů. Buffery jsou vytvářeny pro vertexy, textury, normály a také i pro indexy vertexů. Vytvoření bufferu je provedeno metodami createBuffer(), bindBuffer() a bufferDate(). První zmíněná metoda vytvoří instanci bufferu. Druhá zmíněná metoda pomocí argumentů přiřadí, o jaký typ bufferu se bude jednat a přiřadí ho k instanci bufferu. A třetí zmíněná metoda přiřadí k danému bufferu data. Příklad práce s buferrem:

```
var buffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(data),
gl.STATIC_DRAW);
```

Obr. 5 Vytvoření instance bufferu při použití WebGL.
Zdroj: vlastní zpracování

Dalším krokem je nastavení proměnných s kvantifikátorem attribute. Toho je docíleno metodami getAttribLocation(), bindBuffer(), vertexAttribPointer() a enableVertexAttribArray(). Příklad nastavení proměnných s kvantifikátorem attribute:

```
var attribLoc = gl.getAttribLocation(program, attrName);
gl.bindBuffer(gl.ARRAY_BUFFER, buffer);
gl.vertexAttribPointer(attribLoc, size, type, normalize, stride,
offset);
gl.enableVertexAttribArray(attribLoc);
```

Obr. 6 Nastavení proměnných s kvalifikátorem attribute při použití WebGL.
Zdroj: vlastní zpracování

Za atribut attributeName bude dosazen textový řetězec s názvem proměnné. Za atribut buffer bude dosazena instance bufferu, ze kterého bude proměnná získána. Atributem type je definován datový typ dat například gl.FLOAT. Proměnná normalize je datového typu boalean. Tímto atributem se předá informace, zda data jsou normalizovaná či nikoli. Atributy stride a offset ovlivňují, jak budou data čtena.

Posledním krokem před samotným vykreslováním je definování lokace proměnných s kvalifikátorem uniform. K tomu slouží metoda getUniformLocation(). Do atributů se vkládá vytvořená instance programu a textový řetězec s jménem proměnné, která v shaderu má kvalifikátor uniform.

Po tomto kroku už následuje práce s maticemi a samotné vykreslení scény pomocí metody drawElements().

8.2.1 Vlastní OBJ parser

Pro načítání OBJ souborů byl naprogramován vlastní skript. Nejprve bylo nutné se seznámit se strukturou a systémem zapisování dat o vrcholech, normálách, souřadnicích textur, a dalších. Příklad struktury OBJ souboru:

```
mtllib cube.mtl
o Cube
v 1.000000 1.000000 -1.000000
v 1.000000 -1.000000 -1.000000
v 1.000000 1.000000 1.000000
v 1.000000 -1.000000 1.000000
v -1.000000 1.000000 -1.000000
v -1.000000 -1.000000 -1.000000
v -1.000000 1.000000 1.000000
v -1.000000 -1.000000 1.000000
f 2 3 4
f 8 7 6
f 5 6 2
f 6 7 3
f 3 7 8
f 1 4 8
f 1 2 4
f 5 8 6
f 1 5 2
f 2 6 3
f 4 3 8
f 5 1 8
```

Obr. 7 Příklad struktury souboru s příponou .obj.
Zdroj: vlastní zpracování

Soubor obsahuje několik klíčových slov, které pomáhají správně identifikovat, zda se jedná např. o vrchol či normálu. Seznam vybraných klíčových slov/písmen souboru:

- v – Specifikuje vrchol neboli vertex
- vn – Specifikuje normálový vektor
- vt – Specifikuje souřadnice textury
- f – Specifikuje plochu
- g – Specifikuje název skupiny pro prvky, které následují
- o – Specifikuje název objektu pro prvky

- usemtl – Specifikuje název materiálu pro prvek, který následuje
- mtllib – Definuje soubor, kde se nacházejí vlastnosti materiálů, které budou použity [27].

Vrchol je definován souřadnicemi x, y, z. Souřadnice jsou odděleny mezerou. Jeden řádek odpovídá jednomu vrcholu. Pro modely volných forem se využívá čtvrtá souřadnice w[23].

Normálový vektor je definován souřadnicemi i, j, k. Souřadnice jsou odděleny mezerou. Jeden řádek odpovídá jednomu normálovému vektoru vrcholu [27].

Textura je definována souřadnicemi u, v, w. Do souřadnic jsou dosazeny hodnoty, které jsou odděleny mezerou. Písmenko *u* je hodnota pro horizontální směr v textuře. Písmenko *v* je hodnota pro vertikální směr v textuře. Písmenko *w* je hodnota hloubky textury. 2D textury vyžadují definici prvních dvou hodnot. 3D textury vyžadují definici všech tří hodnot [27].

Plocha je definována pomocí indexů vrcholů, normálových vertexů a texturových vertexů. Plochu můžeme definovat tripletem čísel oddělených lomítkem. Triplet obsahuje hodnoty v, vt, vn. Triplety jsou od sebe odděleny mezerou [27].

Skupinu prvků definujeme názvem, kterému předchází písmenko g odděleno mezerou. Takto definujeme všechny následující prvky jako součást skupiny, do toho momentu, něž definujeme skupinu novou. Je možné mít více názvů v jednom řádku oddělených mezerou. To znamená, že následující hodnoty patří do všech skupin definovaných jménem v řádku začínajícím písmenem *g* [27].

Písmenko *o* specifikuje název objektu pro prvky, které následují. Má obdobné využití jako řádek začínající písmenem *g*. Jeho využití je volitelné [27].

Klíčovým slovem usemtl je definován název materiálu pro prvek, který následuje. Pokud je název definován, může být pouze změněn, nikoli odebrán. Pokud není definován, je použit bílý materiál. Název materiálu odpovídá názvu v souboru, který je definován pomocí mtllib. Pod názvem materiálu se v tomto souboru nacházejí vlastnosti materiálu [27].

Klíčové slovo mtllib definuje soubor, kde se nacházejí vlastnosti materiálů, které budou použity [27].

Soubor s daty o OBJ modelu je zpracován metodou parseOBJ(), která je definována v souboru s vlastním skriptem. Tato metoda je nejobsáhlejší metodou scriptu. Script dále obsahuje další dílčí metody, ale tato metoda je stěžejní. Mezi dalšími metodami jsou například takové, které slouží pro zjišťování, o jaký řádek se jedná z hlediska toho, zda se jedná o vertex, normálu či jiná data zakódovaná v OBJ datové struktuře. Seznam metod ve scriptovém souboru pro zpracování OBJ souboru:

- parseOBJ() – Hlavní metoda scriptu. Zpracovává OBJ soubor.
- *is* metody – Metody sloužící k určení, zda se jedná o vertex, normálu, texturovou souřadnici a další nebo také ke zjištění, zda se musí provést úprava indexů pro správné vykreslení. Příklad vytvořených *is* metod:
    - isVertexLine
    - isNormalLine
    - isTextureLine
    - isFaceLine
- setMeshTodefaultValues() – Tato metoda slouží k nastavení hodnot do původního stavu.
- addMesh() – Metoda přidávající do datové struktury informace o celém mesh objektu.
- addIndices() – Metoda přidává k danému do datové struktury informace o indexech k vertexům, textovým souřadnicím a normálám.
- triangulateIndices() – Protože WebGL vykresluje po trojúhelníkách a je nutné znát tři body, které je nutné spojit a v souboru OBJ se může vyskytovat informace, kde se udávají čtyři indexy, je nutné informaci zpracovat a udělat z ní nový trojúhelník.
- convertToIntegers() – Tato metoda převádí textový řetězec na čísla.

V metodě parseOBJ() je spuštěn cyklus, který projde řádek po řádku. Jakmile se identifikuje na začátku řádku klíčové slovo pomocí *is* metody, je tok kódu směrován na příslušnou metodu, která je určená ke zpracování tohoto řádku. Identifikace

klíčového slova je prováděna regulárními výrazy. Regulární výraz je textový řetězec, který nám umožňuje vyhledat, zda textový řetězec definovaný regulárním výrazem je v testovaném textu obsažen. Příklad použitého regulárního výrazu:

```
/^v\s/
```

Řádek je zpracován příslušnou metodou a následně dané informace o modelu uloženy do datové struktury, která se následně předá k dalšímu zpracování. Tato datová struktura obsahuje data o modelu potřebné k jeho vykreslení. Další zpracování už probíhá kódem popsaným v předešlé kapitole s tím, že zpracovaná data modelu jsou předávána do daných bufferů.

Obr. 8 Křeslo zobrazené vlastním OBJ parserem.
Zdroj: vlastní zpracování

8.3 Implementace Three.js

Pro využívání této knihovny je potřeba stáhnout soubor s JavaScriptovým kódem knihovny, který následně vložíme do tagu script umístěného v těle HTML stránky nebo můžeme do tagu script vložit Content Delivery Network (CDN) adresu, která

bude odkazovat na knihovnu Three.js uloženou na vzdáleném serveru. CDN je síť počítačů vzájemně propojených skrze internet. Tímto je uživatelům zvýšena dostupnost dat. Síť se skládá ze zdrojových serverů s originálním obsahem, dalších serverů s replikovaným obsahem. Tyto servery jsou označovány jako point of presence (PoP). Jsou strategicky rozmístěny v různých částech internetové infrastruktury. Další součástí CDN sítě je směrovací systém, který zajišťuje, aby byl obsah uživateli doručen z geograficky nejbližšího PoP uzlu sítě [28]. Po tomto procesu jsme schopni využívat knihovnu. [29]

Vytvoření základní scény se provede v několika málo krocích. Prvním krokem je vytvoření instance scény, která je nejdůležitější součástí. Do této instance budeme následně vkládat světla, různé druhy 3D objektů, které nám nabízí Three.js a 3D modely. Vytvoření instance třídy Scene:

```
var scene = new THREE.Scene();
```

Obr. 9 Vytvoření instance třídy Scene() s použitím knihovny Three.js.
Zdroj: vlastní zpracování

Pro většinu možných objektů se dostaneme přes klíčovou třídu THREE. Obdobným způsobem jako scéna jsou vytvořeny kamera a renderer. Three.js disponuje více druhy kamer. Každá kamera má své specifické chování. V aplikaci je využita ta nejběžnější, a to perspektivní kamera. Její chování má napodobovat způsob vidění reality tak, jak ji vidí člověk svýma očima. Třída PerspectiveCamera, která instanci kamery vytvoří, toto napodobení umožní. Třída vyžaduje čtyři argumenty. Prvním argumentem je číslo, jež definuje šíři pole vidění. Dalším je poměr stran. Poslední dva argumenty jsou čísla, která říkají jak blízko a jak daleko bude objekty ve scéně možné vidět, tzv z-near a z-far. Následně instance třídy PerspectiveCamera je přidána do scény pomocí metody add(). Příklad vytvoření:

```
var ration = window.innerWidth/window.innerHeight;
var camera = new THREE.PerspectiveCamera(75, ration, 1, 1000);
```

Obr. 10 Vytvoření instance třídy PerspectioveCamera() s použitím knhovny Three.js.
Zdroj: vlastní zpracování

Vytvoření rendereru předchází získání elementu canvas, který je následně předán do konstruktoru třídy WebGLRenderer, jež instanci rendereru vytvoří. Argumenty konstruktoru musí být zabaleny do JavaScriptového objektu. To v praxi znamená, že získaný element canvas je vložen do složených závorek a přiřazen k datové proměnné. Ukázka vytvoření rendereru a přiřazení elementu ourCanvasElement k datové položce canvas:

```
var renderer = new THREE.WebGLRenderer({canvas: ourCanvasElement});
```

Obr. 11 Vytvoření rendereru a přiřazení elementu canvas s použitím knihovny Three.js.
Zdroj: vlastní zpracování

Aby bylo možné se ve scéně pohybovat nebo scénou otáčet, musí být importována třída obsahující metody k tomu určené. Po importaci je vytvořena instance třídy OrbitControls a jako argumenty konstruktoru jsou vloženy instance třídy PerspectiveCamera a instance třídy WebGLRenderer. Aby se kontrola scény projevila a byl viditelný pohyb ve scéně či otáčení scény, je nutné v renderovací smyčce volat metodu update(). Ta je přístupna přes tečkovou notaci u vytvořené instance třídy OrbitControls.

```
var controls = new OrbitControls(camera, renderer);
```

Obr. 12 Vytvoření instance třídy Scene() s použitím knihovnyThree.js.
Zdroj: vlastní zpracování

Po provedení tohoto postupu už následuje přidávání světel a 3D objektů dle kýžené představy. Aby se světlo, či 3D model zobrazily ve scéně, je nutné vždy do scény přidat jejich instanci. To lze provést skrze metodu add(), kterou lze zavolat přes tečkovou notaci instance třídy Scene. [29]

Po modelaci scény už zbývá pouze scénu vykreslit. Toho je docíleno zavoláním metody render(), která je součástí instance třídy WebGLRenderer. Jako argumenty vložíme instance tříd Scene a PerspectiveCamera. Renderer z toho bude vědět, co všechno ve scéně má vykreslit, z jaké strany a pod jakým úhlem scénu vykreslit. Tato

metoda je pak vložena do vykreslovací smyčky, aby se případně vykreslovaly některé naprogramované translace či transformace objektů scény. [29]

Three.js disponuje možnostmi načítat již vymodelované 3D modely. Pro jejich načtení je potřeba tzv. loader, který má za úkol přečíst a zpracovat data o modelu. To znamená přečíst a zpracovat jeho vrcholy, normály, indexy vrcholů, normál, textur a načíst správně textury modelu. Three.js poskytuje mnoho loaderů, tak aby pokrylo co nejvíce formátů, v nichž může být 3D model uložen. [29]

Pro použití loaderu musíme importovat třídu, která disponuje metodami, které zpracují daný formát. Například pro načítání souboru s příponou .obj, je potřeba importovat třídu OBJLoader.js a také MTLLoader.js pro správné načtení textur souboru. Po importu těchto tříd vytvoříme jejich instance. Nejprve je vytvořen loader pro textury.

```
const mtlLoader = new MTLLoader();
```

Obr. 13 Vytvoření instance třídy MTLLoader pro možnost načtení textury ze souboru s příponou .mtl s použitím knihovnyThree.js.
Zdroj: vlastní zpracování

Poté je načten soubor a následně v těle anonymní metody vytvořena instanci třídy OBJLoader, které nastavíme získanou texturu. Následně načteme soubor s OBJ modelem a v anonymní metodě jej vložíme do scény.

```
mtlLoader.load('object.mtl',
  ( materials ) => {
    materials.preload();
    const objLoader = new OBJLoader();
    objLoader.setMaterials(materials);
    objLoader.load('object.obj', ( object ) => {
      var model = object;
      scene.add(model);
  });
})
```

Obr. 14 Načtení souboru a jeho textur s použitím knihovny Three.js.
Zdroj: vlastní zpracování

Po této proceduře načítání souboru už zbývá jen scénu vykreslit. Toho docílíme postupem, který je již zmíněn výše. [29]

8.4 Implementace Babylon.js

Pro využití knihovny Babylon.js je nutné vložit do těla HTML stránky scriptový element, jenž bude načítat knihovnu Babylon.js. V projektu je zvolena tato praktika. Načítání knihovny je pomocí adresy CDN.

Pro vytvoření Babylon.js scény je potřeba získat instanci elementu canvas vloženého v HTML stránce. Pro vykreslování scény je jednou z nejstěžejnějších součástí kódu třída Engine. Ta zajistí vykreslování scény. K této třídě je možné se dostat přes tečkovou notaci a třídu BABYLON. Do konstruktoru se vkládá instance elementu canvas a hodnota datového typu boolean, která deklaruje, zda bude či nebude zapnut antialiasing. Antialiasing je technologie zpracování obrazu. Slouží k vyhlazování hran grafických prvků na monitoru obrazovky [30].

```
var canvas = document.getElementById('canvas');
var engine = new BABYLON.Engine(canvas, true);
```

Obr. 15 Vytvoření instance třídy Engine() s použitím knihovny Babylon.js.
Zdroj: vlastní zpracování

Po vytvoření enginu je zapotřebí vytvořit scénu, která bude sdružovat všechny objekty do ní přidané, jenž budou vykresleny. Do konstruktoru třídy Scene se vkládá instance třídy Engine.

```
var scene = new BABYLON.Scene(engine);
```

Obr. 16 Vytvoření instance třídy Scene s použitím knihovny Babylon.js
Zdroj: vlastní zpracování

Prvním objektem vloženým do scény je kamera. Babylon.js nabízí k výběru několik druhů kamer. Každá má své specifika a své využití. Pro zobrazení a manipulaci se scénou je vhodné použití kamery třídy ArcRotateCamera. Tato kamera se chová tak, že vždy je namířena na cíl nebo bod ve scéně, který je možný si libovolně zvolit. Následný pohyb je rotací kolem zvoleného cíle, přičemž cíl je středem této rotace.

```
var targer = new BABYLON.Vector3(0, 0, 0);
var camera = new BABYLON.ArcRotateCamera('camera', 0, 0, 50, target,
scene);
```

Obr. 17 Vytvoření instance třídy ArcRotateCamera s použitím knihovny Babylon.js
Zdroj: vlastní zpracování

Do konstruktoru se vkládají hodnoty v pořadí: jméno instance, alfa, beta, rádius, cíl scény, instance scény. Hodnoty alfa a beta se udávají v radiánech. Tyto hodnoty nám určují počáteční pozici kamery ve vzdálenosti od cíle scény, který je udán hodnotou rádiusu a pohled kamery bude nastaven na zvolený cíl. V příkladě vytvoření instance kamery je tato pozice vložena do proměnné target, jejíž datový typ je třídimenzionální vektor. Přidávání instancí do scény se v této knihovně provádí způsobem, jaký je použit ukázkách kódu výše. To znamená, že do konstruktorů se přidává instance scény, ve které chceme, aby byla instance využita. Zpravidla atribut pro vložení instance scény je jako poslední vkládaný do konstruktoru. Pro zajištění pohybu je potřeba volat metodu attachControl(), která je součástí třídy ArcRotateCamera. Do argumentů je vkládán jako argument element canvas a další hodnotou je proměnná datového typu boolean, která ovlivňuje ovládání. [31]

Do takto vytvořené scény už jen zbývá přidat světla a objekty, které budou chtěny být ve scéně zobrazeny.

Vykreslení scény je prováděno pomocí instance třídy Engine. Do těla metody runRenderLoop() je vložena funkce v jejímž těle je volána metoda render(), která je metodou třídy Scene. Případně pokud ve scéně jsou již vložené nějaké objekty lze zde vložit i nějaké transformace pozice, či rotace daného objektu, které se následně ve scéně projeví. Tudíž je možné zde vytvořit nějaký pohyb tělesa ve scéně. [31]

Babylon.js dále nabízí vkládání 3D modelů ze souboru. Podpora různých formátů je také poskytována. Babylon.js nabízí specializované loadery pro formáty glTF/GLB, stl a obj. Disponuje však jedním obecným loaderem, který dokáže načítat různé druhy formátů pro 3D grafiku. Pro použití je opět potřeba vložit do těla HTML stránky tag script, jenž bude mít v atributu src CDN adresu odkazující na knihovnu poskytující metody pro načítání 3D modelů. Pokud prostředí, ve kterém je aplikace tvořena, umožňuje využití NPM, lze importovat knihovnu skrze NPM. Knihovna disponuje více metodami pro načtení modelu. Lze využít metody Append(), Load(), ImportMesh(), ImportMeshAsync() nebo pro načtení animací využít metodu ImportAnimations().

```
BABYLON.SceneLoader.ImportMesh("", "pathToFolderWithModel/",
"fileName", scene, function (newMeshes) {
    newMeshes.forEach(mesh => {
        Do something with meshes…
    });
});
```

Obr. 18 Načtení 3D modelu pomocí metody ImportMesh() s použitím knihovny Babylon.js
Zdroj: vlastní zpracování

Prvním atributem metody je pojmenování objektů. Tento atribut může být vyplněn i hodnotou *null*. Dalším atributem je cesta k souboru, za kterým následuje atribut, do kterého se vkládá samotné jméno souboru s příponou. Tyto dva parametry jsou datového typu String. Poté je potřeba vložit instanci třídy Scene jako další atribut. Tím je načítaný model vložen do dané scény. Posledním atributem je anonymní

funkce, která však může být v těle prázdná. Nebo můžou být do těla metody vloženy příkazy, které mohou upravit např. pozici načítaného modelu, natočení modelu, upravení textury modelu a dalších. [31]

9 Testování

V kapitole jsou popsány různé druhy testů pro testování software. Následně je popsán test pomocných knihoven Three.js a Babylon.js a jsou prezentovány a vyhodnoceny výsledky testů.

9.1 Výkonnostní testování

Výkonnostní testy se provádí pro účely zjištění výkonnosti, slabin a předností softwaru. Lze měřit mnoho parametrů. Tyto parametry se liší v závislosti na účelu využití daného software. Mezi parametry mohou patřit:

- Využití paměti – parametr udávající množství paměti, které software spotřebovává při běhu [32].
- Doba odezvy – parametr udávající dobu, než software uskuteční požadavek uživatele [32].
- Propustnost – parametr udávající informaci počtu jednotek informací, který systém zpracuje za určitý čas [32].
- Dostupnost – parametr udávající dobu, za kterou je software načtený pro interakci s koncovým uživatelem [32].

Dle výsledků lze následně najít tzv. bottleneck, volně přeloženo jako problémové nebo úzké místo, to následně opravit či předělat tak, aby software splňoval požadavky. [32]

9.2 Druhy testů

V kapitole jsou popsány dva druhy testů, které lze použít pro získání kýžených dat a informací o výkonnosti testovacího subjektu. Tyto dva hlavní druhy testů se nazývají Load Test, do češtiny lze přeložit jako zátěžový test, a Stress Test, který lze přeložit jako test hraniční zátěže.

9.2.1 Zátěžový test (Load Test)

Zátěžový test je proces testování softwaru, ve kterém je testován výkon softwarové aplikace při specifické očekávané zátěži. Lze vysledovat, jak se softwarová aplikace

chová, když k ní přistupuje více uživatelů současně nebo zpracovává větší objem dat. Cílem zátěžového testování je zlepšit výkonová místa a zajistit stabilitu a hladké fungování softwarové aplikace před jejím případným publikováním. Obecně se zátěžové testy používají k ověření stanovených kritérií kvality. [33; 34]

9.2.2 Test hraničních hodnot (Stress Test)

Stress testování je v podstatě zátěžový test, kdy aplikujeme vyšší zátěž, než se očekávalo. Je sledováno, jak se systém chová při vyšší zátěži a při překročení konstrukčních limitů. Dalším cílem je pozorování, kdy se software porouchá a jak začne selhávat. [33; 34]

Typickým průběhem testu je neustálé zvyšování zátěže za účelem zjištění kapacity systému. Zvyšováním zátěže můžeme docílit jak přetěžováním zdrojů systému, tak snižováním kapacity zdrojů systému. Zdroji můžou být např.: operační paměť, diskový prostor, výkon CPU a další. [33; 34]

9.3 Výkonnostní porovnání 3D knihoven

Knihovny byly testovány způsobem, který byl kombinací obou již zmíněných testů. Bylo zvyšováno zatížení knihovny a ponecháno určitou dobu. Před zvýšením byly zaznamenány parametry:

- Frame per second (FPS)
- Čas potřebný k renderování jednoho frame
- Velikost alokované paměti
- Doba načtení scény

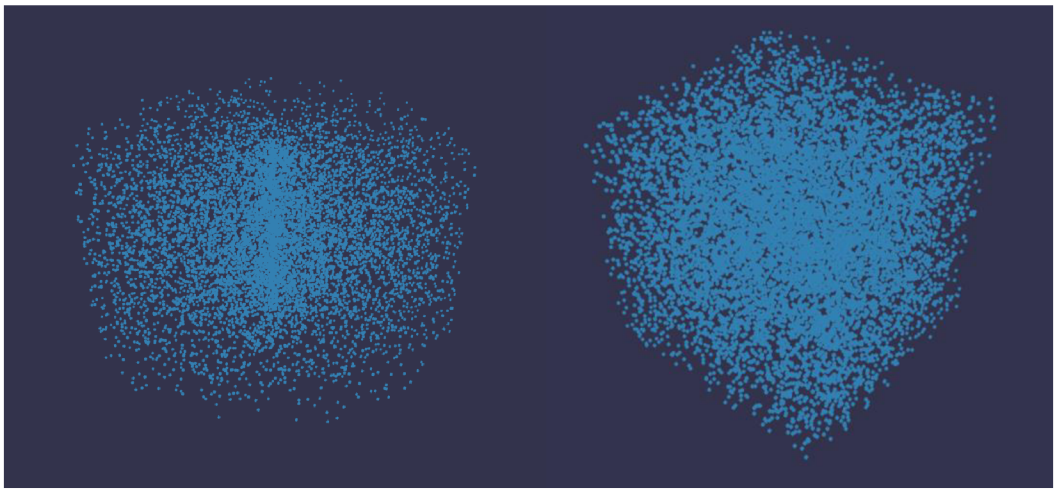
Údaj FPS neboli snímková frekvence indikuje, kolik je zobrazeno snímků za jednu sekundu. Čím vícekrát se zobrazí frame za sekundu, tím bude plynulejší pohyb obrazu. Aby si lidské oko a mozek zhotovily plynulý pohyb, postačí, aby se zobrazovalo 24 FPS. Tomuto parametru dává váhu hodně hráčů počítačových her [35]. S FPS úzce souvisí další parametr, a to je čas potřebný k renderování jednoho frame. Poukazuje na to, kolik času zabere počítači vykreslit jeden frame. Tento údaj lze ověřit matematickým výpočtem z údaje FPS. Jako další parametr byla zvolena

velikost alokované paměti. U tohoto parametru je snaha dosáhnout co nejmenších čísel. Čím větší číslo tohoto parametru bude, tím je program náročnější na výpočetní výkon, z čehož vyplývá, že je náročnější pro zpracování.

Posledním zvoleným parametrem je doba načtení scény. Toto je důležitý faktor pro zhodnocení, zda je knihovna rychlá. Tento parametr je také důležitý pro pohodlí koncového uživatele. Dle článku „Response Times: The 3 Important Limits“ [36], jsou tři důležité hranice. První hranicí je 0,1 sekundy, kdy koncový uživatel nabývá dojmu, že systém reaguje ihned a uživatel si nevšimne žádného zpoždění. Tím je docíleno toho, že uživatel zůstává soustředěný. Další hranicí je 1 sekunda. To je doba, která již je hranicí pro to, aby uživatel mohl nabýt dojmu lehkého zpoždění systému a může být narušena jeho koncentrace a tok myšlenek. Poslední hranicí je doba 10 sekund. Toto je hranice, kde tok myšlenek a koncentrace bývá narušena a uživatel bude mít nutkání začít dělat jiné úkoly. Při takovéto či větší prodlevě je důležité dát uživateli zpětnou vazbu předpokládaného dokončení úkonu. [36]

9.4 Výsledky testu

Pro naměření hodnot byl proveden test knihoven, kde hodnoty byly měřeny na statické a pohyblivé scéně. Scéna se skládala z krychlí, které se do scény postupně přidávaly na náhodné pozice. V pohyblivé scéně krychle opisovaly kružnici s náhodným poloměrem a náhodnou rychlostí. Ve statické scéně krychle zůstávaly na náhodné pozici.



Obr. 19 Ukázka pohyblivé scény (vlevo) a statické scény (vpravo)
Zdroj: vlastní zpracování

Naměřené hodnoty se zaznamenaly po uplynutí doby dvaceti sekund. Pro parametr průměrné doby načtení bylo zhotoveno deset měření, ze kterých pak byl spočítán průměr. Test probíhal na zařízení disponující procesorem Intel® Core™ i7-8650U, 16 GB operační paměti a grafickou kartou Intel UHD Graphics 620 Kaby Lake GT2.

Tabulka 1 Naměřené hodnoty při využití knihovny Three.js

	počet objektů ve scéně	FPS	Doba pro načtení jednoho frame [ms]	Alokovaná paměť [Mb]	Průměr doby načtení scény [ms]
statická scéna	1	60 (60-61)	17 (8-22)	9 (8-14)	384,9
	10	60 (60-60)	17 (4-22)	9 (9-14)	483,2
	100	60 (60-60)	17 (6-28)	9 (9-15)	387,1
	1000	60 (60-61)	19 (7-38)	17 (13-25)	485,1
	10000	37 (29-41)	27 (25-57)	61 (58-65)	898,7
	100000	4 (3-4)	279 (263-349)	560 (519-600)	4450,4
pohyblivá scéna	1	60 (60-61)	17 (3-25)	8 (8-14)	378,2
	10	60 (60-61)	17 (2-40)	9 (9-12)	406,1
	100	60 (60-60)	17 (3-33)	10 (9-15)	387,1
	1000	60 (59-61)	17 (4-39)	14 (13-26)	453,6
	10000	31 (27-33]	30 (28-70)	64 (60-67)	945,3
	100000	3 (3-3)	312 (290-533)	514 (544-637)	4607,6

Zdroj: vlastní zpracování

Tabulka 2 Naměřené hodnoty při využití knihovny Babylon.js

	počet objektů ve scéně	FPS	Doba pro načtení jednoho frame [ms]	Alokovaná paměť [Mb]	Průměr doby načtení scény [ms]
statická scéna	1	60 (59-60)	17	8 (6,8-14,7)	364,3
	10	60 (59-60)	17	9 (7,4-21,4)	379,3
	100	60 (59-60)	17	10 (9,7-22)	412,2
	1000	60 (59-60)	17	29 (28,7-231)	708,6
	10000	60 (59-60)	17	207 (202-212)	3472,1
	100000	Nenačteno			
pohyblivá scéna	1	60 (59-60)	17	8 (7,9-12,2)	401,4
	10	60 (59-60)	17	9 (8,4-9,8)	442,6
	100	60 (59-60)	17	11 (10,3-23,5)	530,3
	1000	60 (59-60)	17	28 (28,5-33,5)	735,1
	10000	60 (59-60)	17	197 (197-210)	4116,3
	100000	nenačteno			

Zdroj: vlastní zpracování

10 Shrnutí výsledků

Z porovnávání knihoven Three.js a Babylon.js z hlediska načítání 3D modelu a jeho textur je snazší využití knihovny Babylon.js, díky jednoduchému komplexnímu procesu zobrazování modelu a načítání textur. Babylon.js do svého loaderu potřebuje znát pouze cestu k souboru a jeho název, a poté dokáže model jakéhokoli formátu správně zobrazit a načíst správné textury. Oproti knihovně Three.js, kde je nutné importovat balíček loaderu pro daný formát modelu a následně vytvořit jeho instanci. Pro zobrazování textur je u formátu OBJ nutné ještě importovat loader pro načtení textur ze souboru MTL a následně vytvořit jeho instanci. Tudíž je potřeba více importů a instancí pro načtení modelů různých typů formátu. To může mít za následek větší množství zdrojového kódu a případné nepřehlednosti či větší časové náročnosti při opravování nebo hledání chyb.

Integrace těchto knihoven s prostředím frameworku Vue.js je vhodnější s knihovnou Three.js. Nastavení scén je u obou knihoven jednoduché. U obou knihoven je možné definovat scénu, kameru a renderer, avšak pro možnost interakce s modelem v knihovně Three.js je navíc definována instance pro kontrolu scény. To v praxi znamená import dalšího balíčku obsahující tuto funkcionalitu. V knihovně Babylon.js je tato funkcionalita již integrována v kameře. Tím je zjednodušen celý proces psaní kódu pro zobrazení scény.

Rozdílem mezi těmito knihovnami je i samotný způsob vykreslení. Při použití knihovny Three.js musíme volat námi vytvořenou metodu. Zde se pak zavolá metoda render(), která je součástí třídy WebGLRenderer, jenž nám scénu vykreslí. Zatímco u knihovny Babylon.js způsob vykreslování jiný. Zde se volá metoda runRenderLoop(), která je metodou instance Engine, kde se následně zavolá metoda render() instance třídy Scene.

Z naměřených hodnot v tabulkách 1 a 2 lze vypozorovat, že Three.js dokáže lépe pracovat s FPS. Oproti knihovně Babylon.js se snížila při velkém množství objektů ve scéně hodnota FPS i přestože pohyb objektů ve scéně a pohyb scénou není plynulý. Bybylon.js si zachovala stejnou hodnotu FPS. Scéna byla díky tomu plynulá.

Plynulý byl jak pohyb objektů, tak pohyb scénou. Ale při velkém množství objektů se již nepodařilo scénu načíst.

Parametr pro alokovanou paměť jde ve prospěch knihovny Three.js, která dle naměřených hodnot nepotřebuje alokaci takového množství paměti pro vykreslení a pohybu scény. Faktem proč Babylon.js má více alokované paměti může být snaha o plynulost scény, kterou koncový uživatel ocení.

Doba načtení scény je u obou knihoven celkem vyrovnaným parametrem až do doby, kdy se ve scéně má vyskytnout mnoho objektů. Zde u knihovny Babylon.js doba zobrazení scény narůstá.

11 Závěr

Cílem práce bylo seznámit čtenáře s problematikou zobrazování 3D grafiky na webu a dále navrhnout a implementovat zobrazování 3D modelu na webu pomocí WebGL. V úvodu byla popsána technologie WebGL a následně je čtenář seznámen s porovnáním knihoven Three.js a Babylon.js, které se využívají pro zobrazení 3D grafiky. Na základě tohoto porovnání byla vybrána knihovna Babylon.js a byla použita pro implementaci zobrazování 3D modelů.

Knihovna byla vybrána po provedených testech na základě dosažených výsledků a jednoduchosti implementace, která byla vyzkoušena při programování scén a v průběhu studování obou zmíněných knihoven. Následné testy byly prováděny jak na statické, tak na pohyblivé scéně. Dosažené hodnoty pak byly zaznamenány. Pro zobrazování modelů byla vybrána knihovna Babylon.js z důvodu, že poskytuje loader pro všechny formáty modelů. Tudíž není třeba řešit různé druhy formátů.

Na začátku práce byla uvažována varianta s využitím některého z open source front-end frameworků. To ale v průběhu implementace bylo přehodnoceno a framework nebyl použit žádný. Aplikace byla napsána pomocí jazyka JavaScript, značkovacího jazyka HTML a knihovny Babylon.js. Důvodem nepoužití frameworku byla větší nekompatibilita s pomocnými knihovnami, než bylo očekáváno. Aplikace využívá verzi WebGL 1.0 a to z důvodu širší podpory prohlížečů.

V aktuálním stavu aplikace dokáže zobrazovat modely. Pro budoucí rozvoj by bylo možné zlepšit celkový vzhled aplikace, uživatelském rozhraní a třeba i větší interakci se zobrazeným modelem.

Bakalářská práce čtenáře podrobně seznámila s používanými formáty pro 3D modely. Formát OBJ byl rozpracován podrobněji. Následně byl pro tento formát napsán vlastní parser. Jedná se o dobře čitelný formát, který je jednoduché strojově zpracovat. Bohužel parser nezobrazuje úplně dokonale všechny modely tohoto formátu. Důvodem může být chybějící implementace zpracování pro další klíčová slova, které nejsou tak běžná. Dalším problémem může být odlišné zakódování některých ploch modelu.

Výslednou aplikaci by dále bylo možné rozšířit o backendovou část. To by vyžadovalo vybrat vhodnou databázi pro uložení 3D modelů a umožňovalo by to jednodušší práci s nahráváním modelů do aplikace. Dále by bylo možné aplikaci rozšířit o větší interakci se zobrazeným modelem. Například zobrazovat drátěný model nebo zvýrazňovat určité části, které by uživatel kurzorem myši určil. Dalším rozšířením by mohla být modifikace textur či barev celého zobrazeného modelu nebo jen určitých částí. Umístění modelu do konkrétní scény, aby si uživatel představil model ve scéně, by bylo také možné rozšíření.

12 Seznam použité literatury

[1] PARISI, Tony, TRESELER, Mary, ed. WebGL: Up and Running. Sebastopol: O'Reilly Media, 2012. ISBN 978-1-449-32357-8.

[2] About The Khronos Group. Khronos.com [online]. The Khronos Group, 2021 [cit. 2022-02-04]. Dostupné z: https://www.khronos.org/about/

[3] FULTON, Steve a Jeff FULTON. HTML5 Canvas: Native Interactivity and Animation for the Web. 2nd ed. Sebastopol: O'Reilly Media, 2013. ISBN 978-1-449-33498-7.

[4] WebGL2 from WebGL1. WebGL2Fundamentals [online]. [cit. 2022-04-26]. Dostupné z: https://webgl2fundamentals.org/webgl/lessons/webgl1-to-webgl2.html

[5] Getting a WebGL Implementation. Getting a WebGL Implementation [online]. [cit. 2022-04-26]. Dostupné z: https://get.webgl.org/get-a-webgl-implementation/

[6] WebGL - 3D Canvas graphics. Can I use [online]. [cit. 2022-04-26]. Dostupné z: https://caniuse.com/webgl

[7] WebGL 2.0. Can I use [online]. [cit. 2022-04-26]. Dostupné z: https://caniuse.com/webgl2

[8] ROST, Randi J. a Bill LICEA-KANE, GINSBURG, Dan, ed. OpenGL Shading Language. 3rd ed. Upper Saddle River: AddisonWesley Professional, 2009. ISBN 978-0-3216-3763-5.

[9] Data Type (GLSL). Khronos [online]. The Khronos Group, 2021 [cit. 2022-01-06]. Dostupné z: https://www.khronos.org/opengl/wiki/Data_Type_(GLSL)

[10] OpenGL Shading Language. Khronos [online]. The Khronos Group, 2021 [cit. 2022-04-06]. Dostupné z: https://www.khronos.org/opengl/wiki/OpenGL_Shading_Language

[11] HALLADAY, Kyle. Practical Shader Development: Vertex and Fragment Shaders for Game Developers. Bristol: Apress, 2019. ISBN 978-1-4842-4456-2.

[12] GOVIL-PAI, Shalini. Principles of Computer Graphics: Theory and Practice Using OpenGL and Maya®. Sunnyvale: Springer Science+Business Media, 2004. ISBN 978-0387-95504-9.

[13] MATSUDA, Kouichi. WebGL Programming Guide: Interactive 3D Graphics Programming with WebGL. Ann Arbor: Edwards Brothers Malloy, 2013. ISBN 978-0-0321-90292-4.

[14] VRTAL, David. Technika 3D: Pixel shadery i pro běžné uživatele. pctuning.cz [online]. Praha: pctuning.cz, 2003 [cit. 2022-01-05]. Dostupné z: https://pctuning.cz/article/technika-3d-pixel-shadery-i-pro-bezne-uzivatele#article-header

[15] CANTOR, Diego a Brandon JONES. WebGL Beginner's Guide. Birmingham: Packt Publishing, 2012. ISBN 978-1-84969-172-7.

[16] GHAYOUR, Farhad a Diego CANTOR. Real-Time 3D Graphics with WebGL 2. 2nd ed. Birmingham: Packt Publishing, 2018. ISBN 978-1-78862-969-0.

[17] CHAKRAVORTY, Dibya. The Most Common 3D File Formats. *All3dp.com* [online]. all3dp.com, 2019 [cit. 2022-01-02]. Dostupné z: https://all3dp.com/3d-file-format-3d-files-3d-printer-3d-cad-vrml-stl-obj/

[18] PELIKÁN, Josef. 3D počítačová grafika na PC. *Cgg.mff.cuni.cz* [online]. Praha: cgg.mff.cuni.cz, 2003 [cit. 2022-01-02]. Dostupné z: https://cgg.mff.cuni.cz/~pepca/lectures/pdf/Grafika2003.pdf

[19] ROGERS, David F. *An introduction to NURBS: with historical perspective*. San Francisco: Morgan Kaufmann Publishers, c2001. ISBN 978-1-55860-669-2.

[20] Top 3D File Formats For 3D Commerce, Social & More. Vntana.com [online]. vntana.com, 2020 [cit. 2021-12-30]. Dostupné z: https://www.vntana.com/blog/demystifying-3d-file-formats-for-3d-commerce-and-more/

[21] COLLADA files. Adobe.com [online]. Adobe, 2021 [cit. 2022-01-07]. Dostupné z: https://www.adobe.com/creativecloud/file-types/image/vector/collada-file.html

[22] COLLADA Overview. Khronos.com [online]. The Khronos Group, 2021 [cit. 2022-01-07]. Dostupné z: https://www.khronos.org/collada/

[23] PARISI, Toni. Programming 3D Applications with HTML5 and WebGL: 3D Animation and Visualisation for Web Pages. Sebastopol: O'Reilly Media, 2014. ISBN 978-1-1449-36296-6.

[24] DIRKSEN, Jos. Three.js Essentials. Birmingham: Packt Publishing, 2014. ISBN 978-1-78398-086-4.

[25] MOREAU-MATHIS, Julien. Babylon.js Essentials. Birmingham: Packt Publishing, 2016. ISBN 978-1-178588-479-5.

[26] SUGANDHI, Abhresh. 15 Best Front end Frameworks for Web Development in 2022. UpGrad knowledge hut [online]. 2022 [cit. 2022-04-28]. Dostupné z: https://www.knowledgehut.com/blog/web-development/front-end-development-frameworks

[27] BOURKE, Paul. Object Files (.obj). Paulbourke.net [online]. [cit. 2022-03-05]. Dostupné z: http://paulbourke.net/dataformats/obj/

[28] HELD, Matthias. Co je síť pro doručování obsahu (CDN)? A je opravdu smysluplná?. Raidboxes [online]. Münster, 2020 [cit. 2022-03-16]. Dostupné z: https://raidboxes.io/cs/blog/hosting-performance/cdn-content-delivery-network/

[29] Three.js [online]. 2010 [cit. 2022-03-14]. Dostupné z: https://threejs.org/

[30] ŠULC, Tomáš. Antialiasing - vyhlazování teoreticky i prakticky. Pctuning.cz [online]. Praha, 2009 [cit. 2022-03-16]. Dostupné z: https://pctuning.cz/article/antialiasing-vyhlazovani-teoreticky-i-prakticky

[31] Babylon.js [online]. 2013 [cit. 2022-03-16]. Dostupné z: https://www.babylonjs.com/

[32] MOLYNEAUX, Ian. The Art of Application Performance Testing: From Strategy to Tools. 2nd edition. Sebastopol: O'Reilly Median, 2015. ISBN 978-1-491-90054-3.

[33] MYERS, Glenford J. The art of software testing. 3rd ed. Hoboken: John Wiley & Sons, 2012. ISBN 978-1-118-03196-4.

[34] COHNEN, Sebastiona. Types of Performance Testing. StormForge [online]. 2021 [cit. 2022-04-12]. Dostupné z: https://www.stormforge.io/blog/types-performance-testing/

[35] WHALEY, Sean. What is Frame Rate and Why is it Important to PC Gaming?. HP [online]. 2018 [cit. 2022-04-12]. Dostupné z: https://www.hp.com/us-en/shop/tech-takes/what-is-frame-rate

[36] NIELSEN, Jakob. Response Times: The 3 Important Limits. Nielsen Norman Group [online]. 1993 [cit. 2022-04-14]. Dostupné z: https://www.nngroup.com/articles/response-times-3-important-limits/

13 Přílohy

1) https://github.com/kvasj/BP

Příloha č. 1

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2019/2020

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

Jméno a příjmení: **Jan Kváš**
Osobní číslo: **I1800195**
Adresa: Krapkova 306/6, Hradec Králové – Malšovice, 50009 Hradec Králové 9, Česká republika

Téma práce: Zobrazení 3D modelů s využitím WebGL
Téma práce anglicky: Rendering 3D models in WebGL

Vedoucí práce: Ing. Milan Košťák
Katedra informatiky a kvantitativních metod

Zásady pro vypracování:

Cíl práce: Prozkoumat problematiku a praktiky zobrazování 3D modelů pomocí WebGL a frameworků. Implementovat webovou aplikaci, která bude modely zobrazovat pomocí popsaných technik a dále porovnat výhody a nevýhody daných technik.
1. Prozkoumat problematiku zobrazování pomocí WebGL
2. Popsat vybrané frameworky a knihovny pro WebGL
3. Implementovat zobrazení 3D modelů ve zvolených knihovnách
4. Porovnat výsledky
5. Závěr a zhodnocení dosažených výsledků

Seznam doporučené literatury:

1) PARISI, Tony, TRESELER, Mary, ed. WebGL: Up and Running. Sebastopol: O'Reilly Media, 2012. ISBN 978-1-449-32357-8.
2) GHAYOUR, Farhad a Diego CANTOR. Real-Time 3D Graphics with WebGL 2. 2nd ed. Birmingham: Packt Publishing, 2018. ISBN 978-1-78862-969-0

Podpis studenta: Datum:

Podpis vedoucího práce: Datum: