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

Srovnání knihoven jazyka JavaScript pro práci s 3D grafikou

Bakalářská práce

Autor: Markéta Šťastná
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Ing. Filip Malý, Ph.D.

Hradec Králové duben 2015

Prohlášení:

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

V Hradci Králové dne 30. 4. 2015 Markéta Šťastná

Poděkování:

Děkuji svému vedoucímu bakalářské práce, panu docentu Malému, za cenné připomínky a odborné vedení při tvorbě této práce.

Anotace

Bakalářská práce se zabývá knihovnami pro práci s trojrozměrnou grafikou, které jsou postavené na WebGL. Práce obsahuje přehled těchto knihoven a jejich porovnání na základě vybraných aspektů, mezi které patří nástroje, které knihovny poskytují, naměřené hodnoty snímkových frekvencí, licence a přístupnost zdrojových kódů, příklady užití a dokumentace, aktuálnost a vývojáři a velikost komunity. Podrobněji se práce věnuje zejména nástrojům těchto knihoven, jako jsou například poskytované geometrie, světla a nástroje pro import již vytvořených objektů do scény. Součástí práce jsou praktické příklady, implementované v každé z vybraných knihoven, které ukazují práci s těmito nástroji. Práce také knihovny porovnává na základě naměřených snímkových frekvencí při vykreslování různě náročných scén.

Annotation

Title: The comparison of JavaScript language frameworks for working with 3D graphics

This Bachelor Thesis focuses on frameworks for working with three-dimensional graphics, which are based on WebGL. The thesis contains a list of these frameworks and their comparison based on selected aspects. These aspects include tools, which are provided by the frameworks, measured values of frame rate, licenses and openness of source codes, usage examples and documentation, contemporaneousness and developers and size of the community. The thesis focuses mainly on the tools, for example on provided geometries, lights and tools for importing objects into the scene. Practical examples, which are implemented in each selected framework, are included in the thesis. The thesis also compares the frameworks based on the frame rates measured when rendering various scenes.

Obsah

Seznam grafů

Seznam obrázků

Seznam zdrojových kódů

1 Úvod

Vývoj v oblasti standardů pro tvorbu webových stránek a s ním související vývoj webových prohlížečů v nedávné době přinesl a stále přináší nové možnosti a technologie. Tyto změny přicházejí s novou verzí specifikace značkovacího jazyka HTML (HyperText Markup Language), označovanou jako HTML5.

Jedním z nových prvků, o které byl jazyk obohacen, je i element canvas, který umožňuje tvorbu grafiky na webu. Právě s tímto elementem totiž pracuje grafická knihovna WebGL (Web Graphics Library), díky které je možné programovat interaktivní dvourozměrnou i trojrozměrnou grafiku a výsledek zobrazovat přímo ve webovém prohlížeči bez nutnosti použití zásuvných modulů, což je z pohledu uživatele jistě zajímavá výhoda. Je také jasné, že interaktivní grafický obsah je pro návštěvníky webových stránek více atraktivní, než textový či statický. Již zde vyvstávají otázky, které tato práce zodpovídá ve druhé kapitole.

- Jak se programují grafické aplikace pomocí knihovny WebGL.
- Jaká je aktuální podpora WebGL ve webových prohlížečích.

Přes tyto nutné počáteční otázky se práce posunuje k jejímu hlavnímu tématu, tj. srovnání knihoven jazyka JavaScript pro tvorbu 3D grafiky na webu. V poslední době probíhá vývoj velkého množství knihoven jazyka JavaScript, které fungují jako nadstavba nad WebGL a usnadňují tak programování grafických aplikací. Vzhledem k počátečnímu nadšení nad možnostmi, které WebGL umožnila, začalo vznikat i množství knihoven, které nepřinášejí žádnou novou přidanou hodnotu, často jsou vyvíjeny pouze jedním vývojářem a zanedlouho i jejich vývoj zaniká. Tato práce se proto zabývá srovnáním vlastností knihoven pro tvorbu grafiky na webu, a to v několika aspektech. Tyto aspekty se snaží co nejlépe reflektovat potřeby programátora, který chce obohatit webové stránky o interaktivní trojrozměrnou grafiku. Zde se tedy objevila hlavní témata, kterými se práce zabývá.

- Jaké knihovny jazyka JavaScript, které usnadňují práci s WebGL, existují.
- Jaké prostředky pro tvorbu interaktivní trojrozměrné grafiky poskytují nejkomplexnější knihovny.

- Jaké jsou rozdíly v hodnotách snímkových frekvencí u jednotlivých knihoven při vykreslování výpočetně náročných scén.

Posledním dvěma otázkám se věnuje většina práce. Je tedy důležité vysvětlit, že knihovny jsou porovnány zejména z hlediska komfortnosti programátora při vývoji grafických aplikací s jejich pomocí a samozřejmě z hlediska nástrojů, které poskytují. Mezi další aspekty porovnání patří například licence knihoven či rozsáhlost a srozumitelnost jejich dokumentace. Aspekty porovnání jsou vybrané i s přihlédnutím ke spokojenosti uživatele, proto je součástí práce i měření snímkové frekvence v různých prohlížečích a na různých počítačových sestavách.

1.1 Cíl práce

Cílem práce je srovnání knihoven jazyka JavaScript pro práci s 3D grafikou, postavených na WebGL. Práce si klade za cíl vytvoření přehledu existujících knihoven a jejím hlavním cílem je srovnání vybraných knihoven, a to z několika aspektů. Jejím výstupem je popis a porovnání těchto knihoven, aby bylo jasné, která má jaké vlastnosti, jaké nástroje poskytuje a k jakému účelu se hodí. Má tak usnadnit programátorovi výběr té pravé knihovny, která bude nejvíce odpovídat jeho požadavkům.

Práce se věnuje především nástrojům, které knihovny poskytují, jako jsou předdefinované geometrie objektů, světelné zdroje a tak dále. Další její podstatnou částí je také srovnání snímkových frekvencí při vykreslování scén u různých knihoven.

1.2 Metodika zpracování

Protože se téma práce týká oblasti, která se objevila celkem nedávno, neexistuje k ní velké množství knižních zdrojů, natož pak v českém jazyce. Proto bylo při tvorbě práce čerpáno převážně ze zdrojů na Internetu[1], a to hlavně z manuálů a dokumentací v anglickém jazyce.

Z knižních zdrojů se tématu programování s knihovnou WebGL věnuje kniha od Andrease Anyuru s názvem Professional WebGL Programming a podtitulem Developing

[1] Internet zde ve smyslu názvu celosvětově propojených počítačových sítí, proto s velkým počátečním písmenem.

3D Graphics for the Web [1]. Na začátku práce bylo využito informací z oficiálních stránek tvůrců HTML5 ze skupiny Web Hypertext Application Working Group [2] a také z webu organizace W3C [3].

V další části (tj. přehled knihoven) bylo čerpáno ze seznamu knihoven při oficiálních stránkách skupiny Khronos (vývojářů WebGL) zaměřených na WebGL [4] a z výzkumného blogu, zaměřeného hlavně na 2D a 3D frameworky s různým, především vědeckým využitím [5]. Při porovnávání knihoven bylo využito hlavně informací z oficiálních webových stránek těchto knihoven, popřípadě z dokumentací či článků nebo webových příspěvků věnovaných daným knihovnám. Grafické nástroje byly přiblíženy pomocí citací z knihy Moderní počítačová grafika [6].

Pro samotné porovnání byla využita multikriteriální analýza, která zohlednila různé váhy jednotlivých porovnávacích kritérií. Mezi nejdůležitější kritéria patřily právě nástroje a výkon knihoven, ale součástí analýzy byla i kritéria jako licence, dokumentace, komunita a podobně. Při porovnávání snímkových frekvencí byly využity jednoduché aplikace, implementované v každé z vybraných knihoven, které vznikly jako součást této práce.

2 Vznik a možnosti WebGL

Knihovna WebGL vznikla díky experimentům s prvkem canvas jazyka HTML5. Je spravována neziskovou skupinou Khronos Group a je postavena na rozhraní OpenGL ES 2.0, které je také spravováno touto skupinou. V této kapitole jsou stručně přiblíženy pojmy jako HTML5, canvas a WebGL. Součástí kapitoly je také příklad vykreslení scény ve WebGL a srovnání podpory WebGL v nejpoužívanějších prohlížečích.

2.1 HTML5 a canvas

„Za úspěchem webu stojí jednoduchý značkovací jazyk, který se lze snadno naučit a podporují jej téměř všechna zařízení – jazyk HTML.“ [7, s. 15]

Počátky jazyka HTML sahají až do roku 1990, kdy byly navrženy první elementy pro tvorbu webových stránek. S rostoucím počtem těchto elementů se zvyšovalo i číslo verze specifikací jazyka HTML, jejíž nejnovější revize je známá jako HTML5. Tato verze obsahuje nové elementy, díky kterým je například možné vkládat na stránku multimediální obsah bez použití zásuvných modulů [3].

Tento nový standard vyvíjí skupina WHATWG (The Web Hypertext Application Technology Working Group). Dle [8] se tato skupina věnuje návrhu nových webových technologií, přičemž se zaměřuje zejména na vývoj HTML. Skupina WHATWG začala nový standard vyvíjet díky nespokojenosti s vývojem standardu XHTML 2.0 konsorciem W3C (World Wide Web Consortium). Toto konsorcium stanovuje mimo jiné právě standardy jazyka HTML a dne 28. října 2014 oznámilo dokončení standardu HTML5. Přesto se tyto dvě skupiny v některých oblastech specifikace HTML5 rozcházejí díky odlišné filozofii vývoje [9].

Značkovací jazyk byl ve verzi HTML5 obohacen o nové prvky, mezi které se zařadil i element canvas. Skupina WHATWG [2] definuje tento element takto (volně přeloženo): *„Element canvas poskytuje skriptům bitmapové pozadí závislé na rozlišení, které může být použito pro renderování grafů, herní grafiky, umění nebo jiných vizuálních obrazů.“*

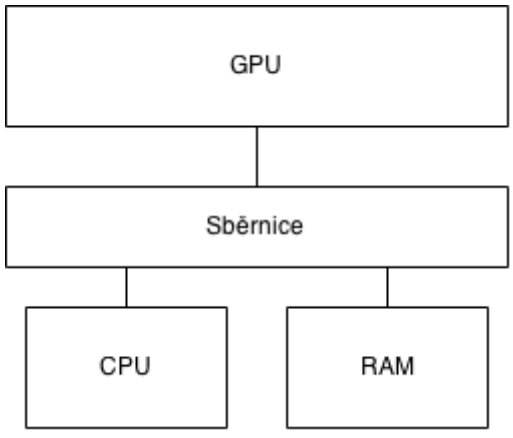
2.2 WebGL

K účelům představeným v předchozí kapitole, mezi které patří vykreslování 3D grafiky, poskytuje HTML5 a jeho element canvas prostředky dynamickému programovacímu jazyku JavaScript. Kód tohoto jazyka se vkládá do HTML kódu stránky a vykonává se obvykle na straně klienta. Právě elementu canvas využívá JavaScriptové grafické API s názvem WebGL.

Zkratka API (Application Programming Interface) označuje skupinu nástrojů (například funkcí), které využívá programátor pro programování aplikací. Knihovna WebGL (Web Graphics Library) slouží k zobrazování interaktivní dvourozměrné i trojrozměrné grafiky v jakémkoli webovém prohlížeči, a to bez použití plug-inů (zásuvných doplňkových modulů). 3D grafiku je tak možné zakomponovat do jakékoliv webové stránky v HTML5. To sebou přináší tu výhodu, že takovou stránku je možné zobrazit například i na některých mobilních zařízeních (WebGL ale musí být podporována prohlížečem na tomto mobilním zařízení).

2.2.1 Využití hardwarových prostředků

WebGL je nízko-úrovňové API, které, protože je založeno na OpenGL ES 2.0, pracuje se skutečným grafickým hardwarem počítače. WebGL aplikace je prováděna na centrální procesorové jednotce (CPU neboli central processing unit) a používá operační paměť (RAM). Pokud se má vykreslit trojrozměrná grafika, pak aplikace zavolá právě WebGL API, které obratem zavolá softwarový ovladač [1]. Tento ovladač odešle grafická data přes sběrnici grafickému procesoru (GPU neboli graphics processing unit).



Obrázek 1: Prostředky využívané WebGL, zdroj: vlastní zpracování dle [1]

O prostředcích WebGL je možné říct následující (volně přeloženo): *„Celkově lze předpokládat, že výkon WebGL bude podobný jako výkon nativní aplikace pro GPU (grafický procesor), protože WebGL užívá GPU pro hardwarově akcelerované vykreslování - navíc je ale potřeba překlad volání WebGL API a shaderů do grafického API operačního systému […] Výkon dále záleží na JavaScriptovém jádru, které využívá webový prohlížeč.“* [10]

2.3 Programování 3D grafiky s WebGL

Aplikace napsaná s pomocí WebGL se skládá nejen z kódu jazyka JavaScript, ale také z tzv. shaderů. Shader kód se píše ve vlastním programovacím jazyce GLSL (Graphics Layer Scripting Language) a je vykonáván na grafické kartě počítače. Díky delegování práce z procesoru na grafickou kartu dochází k nezanedbatelnému zvýšení výkonu aplikace.

Existují dva druhy shaderů [11]: prvními jsou vertex shadery, které jsou používány k práci s vrcholy objektů, a druhými jsou fragment shadery, které umožňují pracovat s barvou a dalšími vlastnostmi pixelů, které jsou vykreslovány.

Knihovna WebGL je plně na shaderech založena. Dle [1] jsou i k vykreslení základních objektů potřeba oba shadery (vertex i fragment shader), což je hlavní důvod toho, proč i k vykreslení jednoduchých scén je třeba většího množství zdrojového kódu. Jako demonstrace tohoto faktu je uveden malý příklad, a to vykreslení prostého bodu na obrazovce. V příkladech v této práci bude kvůli typové kontrole a možnosti použití tříd využit TypeScript (rozšíření JavaScriptu).

2.3.1 Použití TypeScriptu

„TypeScript je open-source programovací jazyk vytvořený a spravovaný firmou Microsoft. Jedná se o nadstavbu nad jazykem JavaScript, která jej rozšiřuje o statické typování a další atributy, které známe z objektově orientovaného programování (třídy, moduly, a další).“ [12]

Příklady v této práci budou napsány s využitím TypeScriptu, protože se v něm dá programovat přehledněji a s využitím datových typů. TypeScript nabízí rozšiřující vlastnosti oproti standardu ECMAScript 5, a to například lambda výrazy, které slouží pro zkrácení anonymních funkcí. TypeScript se kompiluje do čistého JavaScriptu, který běží

v jakémkoliv prohlížeči. K vývoji ho využívá i jedna z rozebíraných knihoven, knihovna Babylon.js.

2.3.2 Příklad vykreslení bodu s využitím WebGL

V této kapitole je ukázán jednoduchý příklad (vykreslení jednoho bodu) naprogramovaný s WebGL. Do HTML dokumentu je potřeba nejdříve vložit `canvas` element s daným identifikátorem.

```
<canvas width="100" height="100" id="canvas"></canvas>
```

Zdrojový kód 1: Element canvas v HTML dokumentu

Na začátek TypeScript dokumentu se vloží reference na element `canvas` do stejnojmenné proměnné. Kvůli typovosti TypeScriptu je také potřeba přetypovat výsledný element na `HTMLCanvasElement`. Metoda `getContext("webgl")` popřípadě `getContext("experimental-webgl")`, zavolaná na tomto elementu, vrátí kreslící WebGL kontext, který se v příkladu vloží do proměnné s názvem `gl`. Na té se volají další metody.

```
var canvas = <HTMLCanvasElement> document.getElementById('canvas');
var gl = canvas.getContext("webgl") || canvas.getContext("experimental-
     webgl");
```

Zdrojový kód 2: Proměnná canvas a WebGL kontext

Dalším krokem je vytvoření shaderů, a to vertex i fragment shaderu, pomocí metody `createShader`. Metoda `shaderSource` poté slouží k nastavení či nahrazení shader kódu v shader objektu. Shader kód se vloží jako parametr této metody ve formě řetězce. U vertex shaderu se vytvoří atribut `position`, který určuje pozici vykreslovaného bodu, a také se nastaví velikost bodu. U fragment shaderu je nastavena barva bodu, tj. zelená barva.

```
var vertexShader = gl.createShader(gl.VERTEX_SHADER);
var fragmentShader = gl.createShader(gl.FRAGMENT_SHADER);

gl.shaderSource(vertexShader,
 'attribute vec2 position;' +
 'void main() {' +
 'gl_Position = vec4(position, 0.0, 1.0);' +
 'gl_PointSize = 10.0;' +
 '}');
```

```
gl.shaderSource(fragmentShader,
 'void main() {' +
 'gl_FragColor = vec4(0.0, 1.0, 0.0, 1.0);' +
 '}');
```

Zdrojový kód 3: Vytvoření shaderů

Pomocí metody `compileShader` dojde ke zkompilování kódu, který byl vložen do shader objektu jako řetězec. Poté je v příkladu vytvořen `shaderProgram`, ke kterému se připojí `vertexShader` i `fragmentShader`.

```
gl.compileShader(vertexShader);
gl.compileShader(fragmentShader);

var shaderProgram = gl.createProgram();

gl.attachShader(shaderProgram, vertexShader);
gl.attachShader(shaderProgram, fragmentShader);
```

Zdrojový kód 4: Kompilace a připojení shaderů u WebGL

Ze shaderProgramu se vrátí lokace atributu pozice. Dále se vytvoří vertex buffer typu `ARRAY_BUFFER`, do kterého se vloží souřadnice pozice bodu (v příkladu jde o souřadnice [0, 0], což znamená, že bod bude uprostřed elementu canvas).

```
var location = gl.getAttribLocation(shaderProgram, "position");
var vertexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, vertexBuffer);
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array([0.0, 0.0]),
gl.STATIC_DRAW);
```

Zdrojový kód 5: Vytvoření vertexBufferu u WebGL

Metoda `clear(gl.COLOR_BUFFER_BIT)` vyčistí color buffer, tj. buffer používaný ke zjištění barvy vykreslovaných pixelů. Dále je třeba určit lokaci dvou souřadnic pozice, zadaných v datovém typu `float`. Nakonec je bod vykreslen pomocí metody `drawArrays`, sloužící k vykreslení grafických primitiv (vykreslení bodů určuje konstanta `gl.POINTS`).

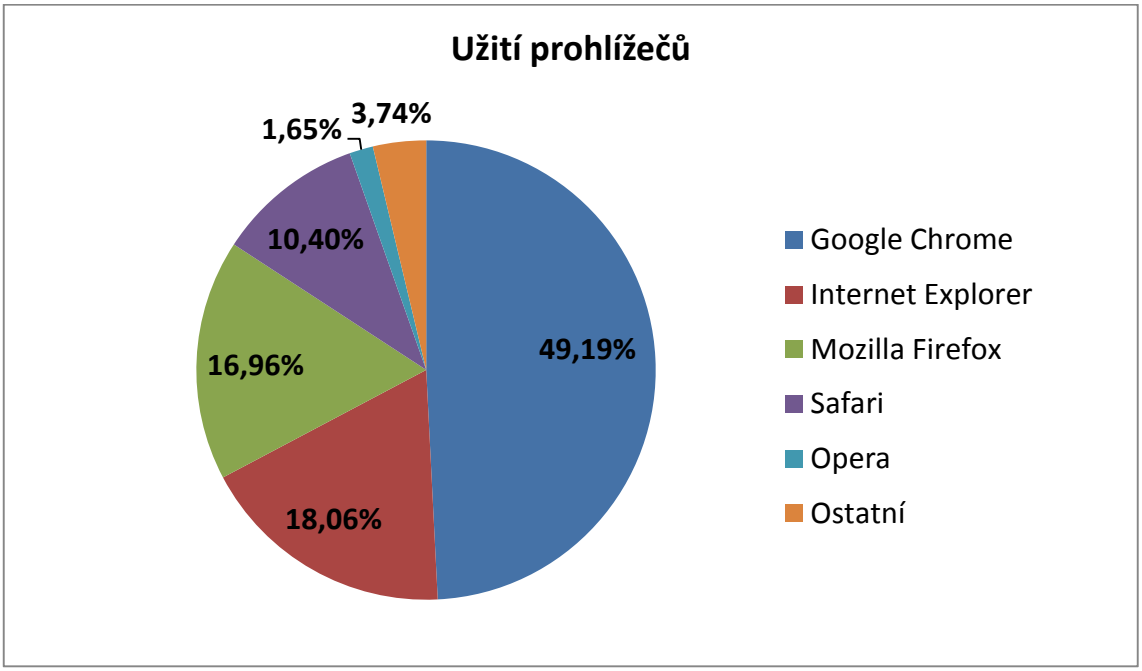
```
gl.clear(gl.COLOR_BUFFER_BIT);
gl.vertexAttribPointer(location, 2, gl.FLOAT, false, 0, 0);
gl.enableVertexAttribArray(location);
gl.drawArrays(gl.POINTS, 0, 1);
```

Zdrojový kód 6: Vykreslení bodu u WebGL

Příklad, spočívající pouze ve vykreslení jednoho bodu, tak zabral desítky řádků kódu. WebGL je API nízko-úrovňové, což sice umožňuje přímou kontrolu hardwarových prostředků, ale také to znamená, že jednoduchý program může vyústit v rozsáhlý kód, jak příklad výše dokazuje. Proto také vzniká mnoho knihoven nad WebGL, které jsou u programování 2D a 3D aplikací vítanou pomocí.

2.4 Podpora WebGL v prohlížečích

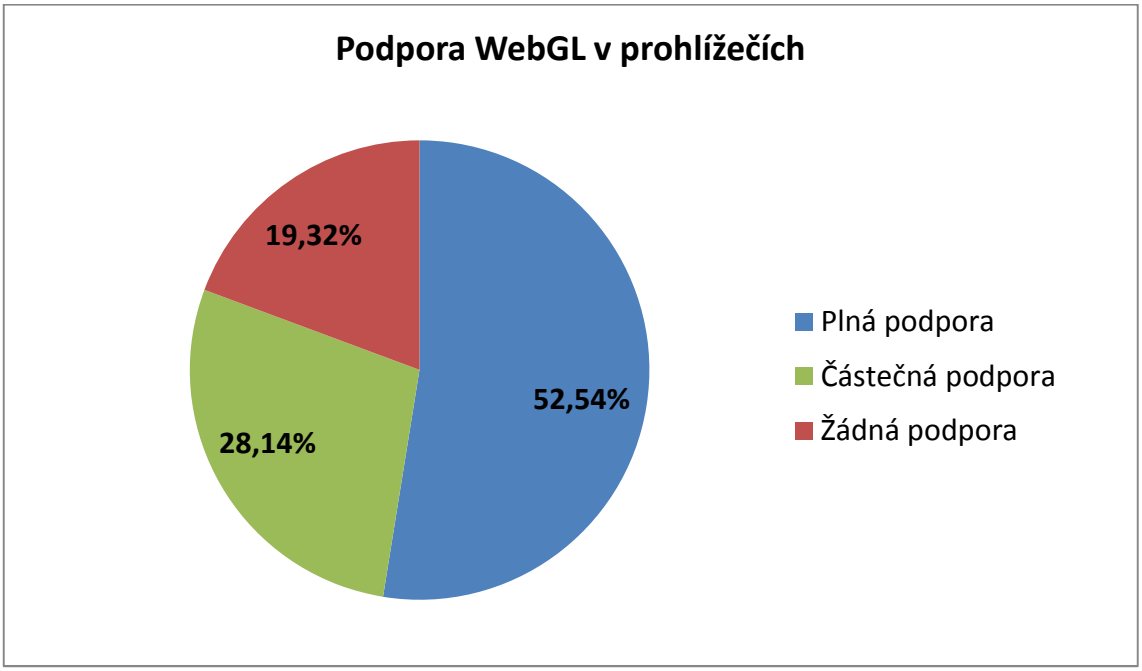
WebGL patří mezi novější technologie, a proto může být problém s její podporou v různých webových prohlížečích. Dle [13] ji k datu vydání této práce naštěstí podporují všechny nejpoužívanější desktopové prohlížeče a dokonce i některé mobilní prohlížeče. Nejčastěji používaným prohlížečem je Google Chrome, za ním následují Internet Explorer a Mozilla Firefox, dalšími prohlížeči v pořadí jsou Safari a Opera [14]. Podíly užití k březnu 2015 ukazuje následující graf.



Graf 1: Užití prohlížečů, zdroj: vlastní zpracování dle statistik [14]

Na základě informací ze zdroje [13] byl také vypracován graf, který ukazuje procentní podíl uživatelů s plnou, částečnou nebo žádnou podporou WebGL v jejich prohlížečích. Částečná podpora znamená, že ne všichni uživatelé v této skupině mohou mít přístup k WebGL – její podpora je totiž ovlivněna i možnostmi jejich hardwaru (grafické karty a jejích ovladačů) a operačního systému.

Tento problém vyřešil mezi desktopovými prohlížeči například Google Chrome, který již od verze 18 plně podporuje knihovnu WebGL, a to díky vykreslovací, čistě softwarové technologii SwiftShader. I uživatelé se zastaralým grafickým hardwarem a operačními systémy (jako například Windows XP) si tak mohou dopřát 3D grafiku přímo v prohlížeči [15].



Graf 2: Podpora WebGL v prohlížečích, zdroj: vlastní zpracování dle statistik [13]

Graf ukazuje podíl uživatelů s různou podporou WebGL, přičemž v úvahu jsou brány všechny prohlížeče ze statistik [13] ve všech stále používaných verzích. Z grafu vyplývá, že největší skupinou jsou uživatelé s plnou podporou WebGL, kteří tvoří většinu s 52,54 procenty. Částečnou podporu WebGL má 28,14 procent uživatelů a žádnou podporu má 19,32 procent uživatelů.

Z nejčastěji užívaných prohlížečů poskytují plnou podporu v nejaktuálnější verzi Google Chrome (verze 41) a Internet Explorer (verze 11). Opera (verze 27), Mozilla Firefox (verze 37) a Safari (verze 8) patří mezi prohlížeče s částečnou podporou. Žádnou podporu již v nejnovější verzi neposkytuje ani jeden z nejužívanějších prohlížečů, jak dokazuje přehled [13].

Jistě je zajímavá také podpora WebGL v mobilních zařízeních. Plnou podporou WebGL se na tomto poli mohou chlubit mobilní prohlížeče Blackberry Browser a Opera Mobile, částečnou podporou pak Chrome pro Android a Firefox pro Android [13].

3 Přehled a porovnání knihoven

Knihoven v jazyce JavaScript, které slouží jako nadstavba nad WebGL a umožňují snazší ovládání jejích prostředků, existují desítky. V této kapitole je vytvořen přehled těchto knihoven a vybrané knihovny jsou porovnány na základě daných aspektů.

3.1 Přehled knihoven

Seznam knihoven nad WebGL přináší oficiální web skupiny Khronos Group [4]. Ač v něm lze k datu vydání této práce nalézt 35 knihoven včetně těch nejznámějších, stále nejde o úplný seznam. Navíc jsou zde uvedeny knihovny pracující jak s dvourozměrnou, tak s trojrozměrnou grafikou. Další zajímavý zdroj i s částečným srovnáním knihoven je výzkumný blog, který vytvořily studentky biomedické vizualizace Autumn Kulaga a Semay Johnstonová [5]. Na základě těchto dvou zdrojů a vlastního hledání byl vytvořen seznam, který zahrnuje knihovny nad WebGL, umožňující práci s trojrozměrnou grafikou. Protože se jedná o velké množství knihoven, je celý seznam uveden v příloze A.

Mnoho z těchto knihoven je jednostranně zaměřených – například knihovna MathBox poskytuje nástroje pro vizualizaci matematických diagramů, knihovna Cesium slouží k vytváření 3D glóbů a 2D map, knihovna XTK je zaměřená na vědeckou vizualizaci (hlavně pro účely biologie a medicíny). Proto se tato práce zaměřila hlavně na knihovny, které jsou orientované obecněji a poskytují komplexní nástroje pro tvorbu trojrozměrných aplikací. Pro další srovnání tak byly vybrány knihovny Three.js, Babylon.js a SceneJS.

3.2 Aspekty porovnávání

První skupinou jsou aspekty, které lze vztáhnout na výběr jakékoli knihovny. Například webová stránka [16] uvádí deset kritérií, která pomohou obecně určit, jaký framework je ten pravý. Podobné, konkrétnější informace nalezneme v článku [17] o vývoji webových aplikací.

Mezi základní obecné aspekty při výběru jakéhokoli frameworku je možné řadit nástroje, které framework poskytuje, licenci, pod kterou je vydáván, dokumentaci, příklady jeho užití, velikost komunity a aktuálnost. Dá se předpokládat, že u některých kritérií (např. kvalita dokumentace) může být porovnání spíše subjektivní, ale přesto se

práce pokusí zaměřit na co nejobjektivnější aspekty a nejobjektivnější porovnání pomocí multikriteriální analýzy. Závěrečné zhodnocení knihoven na základě zkušenosti z práce s knihovnami pak bude logicky subjektivní.

Dle příspěvku Theo Armoura [18] je vhodné se u 3D knihoven zaměřit na tyto aspekty.

1. Je zdrojový kód na serveru GitHub?
2. Vytváří knihovnu více než jeden vývojář?
3. Jak se dají do knihovny importovat 3D data?

Tím hlavním kritériem porovnání jsou samozřejmě možnosti, které knihovna poskytuje k práci s 3D grafikou. Mezi ně patří například nástroje na tvorbu grafických primitiv i složitějších objektů a možnosti světla či kamery. Dalším kritériem je využití hardwarových prostředků při vykreslování, které se projeví na rychlosti vykreslování. Proto je v práci porovnáván tzv. počet snímků za sekundu (FPS – Frames Per Second), který se liší u různých knihoven při vykreslování různých příkladů, a to jak statických, tak dynamických. Všechny tyto aspekty porovnání jsou sloučeny do následujících celků:

1. nástroje knihoven,
2. snímková frekvence,
3. licence a přístupnost zdrojových kódů,
4. příklady užití a dokumentace,
5. vývojáři a aktuálnost,
6. velikost komunity.

3.3 Multikriteriální rozhodování

„V úlohách vícekriteriálního (multikriteriálního) rozhodování máme určenou konečnou množinu n *variant, které jsou ohodnoceny na základě* m *kritérií. Cílem rozhodování je vybrat variantu, která je podle daných kritérií ohodnocena nejlépe. Neboli vybrat tzv. optimální variantu.“* [19]

Pro stanovení vah kritérií byla využita bodovací metoda (tzv. Metfesselova alokace). Při ní uživatel ohodnotí každé kritérium hodnotou z předem dané škály, přičemž platí čím vyšší hodnota, tím závažnější kritérium. Dle [19] se tedy jedná o metodu s kvantitativními vahami kritérií.

Jako metoda ohodnocení alternativ byla zvolena taktéž bodovací metoda, kdy se všechny alternativy ohodnotí na základě předem stanovené stupnice. Pomocí této metody se ohodnotí jak kvantitativní, tak kvalitativní charakteristiky. Navíc je jednoduchá na provedení, a proto byla zvolena pro multikriteriální rozhodování v této práci. Hodnotící škála je v obou případech od jedné do deseti bodů.

Zvolené váhy jednotlivých kritérií, jejich hodnoty u daných knihoven a součty hodnocení lze nalézt v příloze C. Výsledná hodnota kritéria u dané knihovny byla určena za pomoci menších kritérií (například počet praktických příkladů) a jejich hodnot (například 25). U kritérií, u kterých nebylo možné určit hodnotu pomocí počtu (například světla), byla hodnota stanovena ze zvolené stupnice jedna až deset bodů.

Je nutné poznamenat, že se nelze vyhnout určitému subjektivnímu hodnocení, a to jak u vah kritérií, tak u ohodnocení alternativ. To souvisí s tím, že ne každý programátor by zvolil stejné váhy kritérií (pro někoho může být více důležitá kvalita dokumentace, pro jiného praktické příklady užití a podobně). Pro více použitelné srovnání by metoda musela být provedena s větším množstvím expertů. I tak zůstává volba vhodné knihovny záležitostí, která se odvíjí od požadavků programátora i situace, ve které je využita.

3.4 Porovnání knihoven

Vzhledem k velkému množství různě zaměřených knihoven jsou v této práci porovnány hlavně obecněji zaměřené knihovny: knihovna Three.js[2], která je již celkem dobře známá a vývojáři oblíbená, dále knihovny Babylon.js[3] a SceneJS[4].

Původně byla zvažována i knihovna CopperLicht[5], která je od verze 1.8.1 nově open-source (tj. s otevřeným zdrojovým kódem) a je zveřejněna na serveru GitHub. S ní je možné využít editor CopperCube, který ale již není pod licencí „free“, ale „trial“, tj. na vyzkoušení, a to pouze na čtrnáct dní. CopperLicht sám o sobě nemá například třídy vy-

[2] http://threejs.org/
[3] http://www.babylonjs.com/
[4] http://scenejs.org/
[5] http://www.ambiera.com/copperlicht/

tvářející geometrie, které se bez využití editoru tak musí definovat ručně pomocí vrcholů a podobně. Proto nakonec bylo od této knihovny upuštěno.

Stejně tak byla zvažována knihovna PhiloGL, která má mimo nástrojů pro vizualizaci dat sloužit i k vývoji her. Po prohlédnutí jejích nástrojů ale byla pro porovnání zvolena knihovna SceneJS, která slouží primárně pro datovou vizualizaci. Paleta jejích nástrojů je bohatá, i když její oblíbenost na serveru GitHub je o něco menší než u knihovny PhiloGL. Její výhodou je také větší nabídka praktických příkladů.

3.4.1 Nástroje knihoven

Knihovna Three.js má komplexní nástroje pro tvorbu 3D grafiky a není zaměřena přímo na jednu oblast, jako například na tvorbu her. Jedná se o velice rozšířenou knihovnu, a proto existují moduly, které jí přidávají další funkcionalitu. Herní rozšíření se označují THREEx[6] a patří sem například rozšíření k řešení kolizí. Ke knihovně Three.js existuje i editor[7], pomocí kterého je možné vytvořit vlastní scénu bez znalosti kódu.

Další knihovnou je Babylon.js, která slouží i k tvorbě her, a proto obsahuje například moduly zabývající se fyzikou a dalšími herními prvky. K této knihovně také existuje editor scény[8]. Navíc ke knihovně Babylon.js existuje také speciální editor pro materiály[9], do kterého je možné nahrát soubor s objektem a texturami a editor umožní měnit materiál objektu.

Knihovna SceneJS se zaměřuje spíše na vizualizaci komplexních 3D modelů ve vysokém detailu. K této knihovně zatím chybí online editor scény. Zvláštností knihovny je styl jejího API, který je podobný psaní JSON formátu. Poskytovanými nástroji knihoven se podrobněji a v příkladech zabývá kapitola 4.1.

3.4.2 Snímková frekvence

Snímková frekvence se udává se v jednotkách FPS (počet zobrazovaných snímků za sekundu) nebo v hertzích. Počtem snímků za sekundu se udává obrazová odezva,

[6] http://www.threejsgames.com/extensions/
[7] http://threejs.org/editor/
[8] http://www.babylonjs.com/editor/
[9] http://materialeditor.raananweber.com/

jednoduše řečeno to, zdali na uživatelův podnět reaguje aplikace dostatečně rychle a vykreslovaná grafika působí plynule. Lidské oko potřebuje minimálně třicet snímků za sekundu, aby vidělo obraz plynule, přičemž plně dostačující snímková frekvence se pohybuje kolem šedesáti snímků za sekundu [20].

Další jednotkou, kterou se dá měřit výkon grafických aplikací, je počet milisekund potřebných k vykreslení jednoho snímku. Vývojář rozhraní DirectX, Robert Dunlop, se k měření výkonu pomocí snímků za sekundu vyjadřuje následovně (volně přeloženo): *„Jedním z nejčastějších způsobů, jak poskytnout jednoduchou míru grafického výkonu ve hrách, je počet snímků za sekundu. Avšak tato míra může být docela zavádějící, zejména s dnešním rychlejším grafickým hardwarem. Může poskytnout určitou míru výkonu, ale pokud jde o provádění rozsudků týkající se optimalizace, je FPS špatným způsobem měření výkonnosti aplikací.“* [21]

Autor v příspěvku naráží na nelineárnost snímkové frekvence a ukazuje převod snímků za sekundu na počet milisekund potřebných k vykreslení jednoho snímku. Na nich je lépe vidět úbytek výkonu, protože tyto hodnoty klesají lineárně. Jsou proto lepší pro rozsudky týkající se optimalizace výkonu. U vybraných testovacích příkladů v jednotlivých knihovnách jsou tedy měřeny jak snímky za sekundu, tak i počet milisekund k vykreslení snímku, a to na různých platformách a v různých prohlížečích. Následně jsou výsledné snímkové frekvence porovnány pomocí grafů, kde je snímková frekvence přehlednější a nejspíše i užívanější mírou. Podrobnosti měření jsou uvedeny v kapitole 4.2.

3.4.3 Licence a přístupnost zdrojových kódů

Všechny tři knihovny jsou open-source, tj. s otevřeným zdrojovým kódem. Knihovna Three.js je vydávána pod svobodnou MIT licencí[10], knihovna Babylon.js pod Apache licencí[11], která je taktéž svobodnou softwarovou licencí. Obě licence umožňují užívání softwaru k různým účelům (k úpravám a podobně).

[10] http://opensource.org/licenses/MIT
[11] https://www.apache.org/licenses/LICENSE-2.0

MIT licence požaduje zahrnutí textu licence při distribuci softwaru, který knihovnu používá. Apache licence má vcelku podobné podmínky, jen má více definovaná omezení. Například, pokud distributor změnil zdrojový kód knihovny, je potřeba tuto skutečnost uvést. Dále licence ošetřuje situaci s patenty a s použitím projektového jména.

Knihovna SceneJS je vydávána pod tzv. duální licencí (dual-license), takže je přístupná jak pod MIT licencí, tak pod GNU GPL (GNU General Public License)[12] licencí svobodného softwaru. GNU GPL licence zajišťuje, že všechna díla odvozená od původního díla budou také svobodným softwarem pod GNU GPL licencí.

3.4.4 Příklady užití a dokumentace

U knihoven Three.js a Babylon.js je dostupná dokumentace. Dokumentace Three.js[13] je přístupná z oficiálních webových stránek. Místy je sice neúplná, ale u nejužívanějších tříd je velice podrobná a často i jsou v ní uvedené i příklady kódu a implementace (výborně je zpracována například část dokumentace o materiálech).

Dokumentace Babylon.js[14] má oproti Three.js v dokumentaci daleko méně komentářů a vysvětlení. Je také těžší se v ní zorientovat a zjistit, která třída k čemu slouží. Příklady kódu přímo v dokumentaci uvedeny nejsou, ale nachází se v podobě „Wiki“ příspěvků na serveru GitHub[15].

Oproti tomu knihovna SceneJS nemá dokumentaci řádně zpracovánu a je těžké najít informace k jejím nástrojům. K dispozici jsou tutoriály od jejího hlavního vývojáře[16], tzv. „Wiki“ příspěvky na serveru GitHub[17] a několik podrobnějších tutoriálů[18], které popisují dané nástroje, ale které se nachází v odlišné repository na serveru GitHub, než SceneJS.

[12] https://www.gnu.org/copyleft/gpl.html
[13] http://threejs.org/docs/
[14] http://doc.babylonjs.com
[15] https://github.com/BabylonJS/Babylon.js/wiki
[16] http://xeolabs.com/articles/learning-scenejs/
[17] https://github.com/xeolabs/scenejs/wiki
[18] https://github.com/xeolabs/xeolabs.github.com/tree/master/_drafts

Z hlediska praktických příkladů nejvíce vyniká Three.js, na jejímž webu lze nalézt desítky různých příkladů užití se zdrojovými kódy[19]. Příklady knihovny SceneJS[20] jsou zpracovány ve stejném stylu a s obdobnou kvalitou. Babylon.js také nabízí množství příkladů užití. Ty jsou implementovány v rámci aplikace „Babylon.js Playground“[21], ve které je také možné přímo naprogramovat scénu s pomocí knihovny Babylon.js a ihned spatřit výsledek.

3.4.5 Vývojáři a aktuálnost

Knihovna Three.js přišla na scénu trojrozměrné grafiky v prohlížeči v roce 2009. Její poslední verzí ke dni vydání této práce je verze 71. Jejím hlavním vývojářem je Ricardo Cabello, známý pod přezdívkou „Mr. Doob“. Na vývoji se podílí hodně dalších vývojářů, na serveru GitHub má projekt celkem 442 přispěvatelů.

Knihovna Babylon.js je daleko novější, poprvé byla vydána v roce 2013. Jejím hlavním vývojářem je David Catuhe, dalšími vývojáři jsou David Rousset a Pierre Lagarde. Na serveru GitHub má knihovna 44 přispěvatelů. Knihovnu SceneJS vyvíjí hlavně softwarový vývojář Lindsay Kay a vznikla v roce 2009. Na serveru GitHub do ní přispívá 12 uživatelů.

Knihovna Three.js měla za měsíc březen 2015 108 tzv. „commitů“ (odeslání změn), knihovna Babylon.js 115 a knihovna SceneJS osm. Poslední aktualizace proběhla u knihovny Three.js 16. března, u Babylon.js 16. dubna a u SceneJS 15. dubna, takže se na vývoji všech knihoven stále aktivně pracuje.

3.4.6 Velikost komunity

Všechny tři knihovny se nachází na serveru pro open-source projekty jménem GitHub. Dle oblíbenosti (hodnocení pomocí tzv. hvězdiček) uživatelů na tomto serveru můžeme knihovny k datu vydání této práce seřadit následovně:

1. Three.js (19 135 hvězdiček),
2. Babylon.js (2 024 hvězdiček),

[19] http://threejs.org/examples/
[20] http://scenejs.org/examples/
[21] www.babylonjs-playground.com/

3. SceneJS (370 hvězdiček).

Dle oblíbenosti i velikosti komunity a přispěvatelů je tedy jasně vidět převaha knihovny Three.js, což už samo o sobě vypovídá i o přívětivosti knihovny. Babylon.js je ale novější knihovna, proto se dá předpokládat, že její oblíbenost ještě poroste. Přesto je Three.js z těchto knihoven nejvíce užívaná, a proto je možné rychleji najít řešení problémů při vývoji na různých diskuzních fórech a komunitních serverech jako je StackOverflow a podobně. Právě na tomto serveru je přibližně 12 474 otázek s tématikou Three.js, zatímco pouze přibližně 87 týkajících se Babylon.js a 52 souvisejících se SceneJS.

4 Porovnání nástrojů a snímkových frekvencí

Cílem této kapitoly je porovnání knihoven na základě poskytovaných nástrojů a naměřených snímkových frekvencí.

4.1 Nástroje vybraných knihoven v praktických příkladech

Součástí kapitoly jsou praktické příklady, které se vztahují k daným poskytovaným nástrojům knihoven. Tyto nástroje se lépe porovnávají přímo na základě příkladů a práce s nimi, než jen například pomocí dokumentace. Příklady jsou vytvořeny v knihovnách Three.js, Babylon.js a SceneJS, které byly zvoleny k porovnání v předchozí kapitole. Všechny příklady jsou umístěny na přiloženém CD.

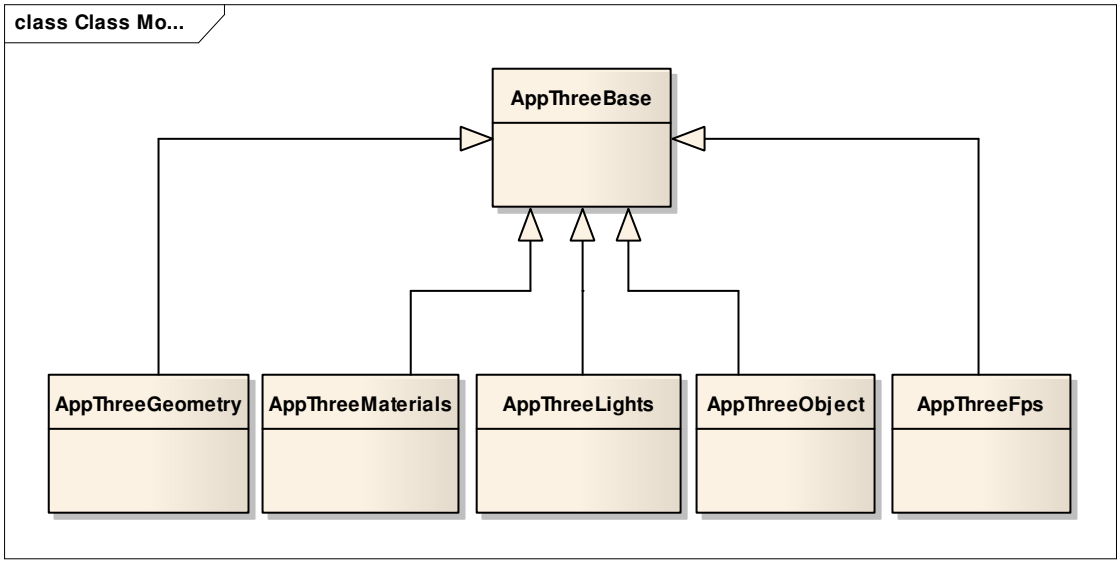
Příklady slouží k prozkoumání nástrojů, které knihovny poskytují, a jako ukázky jejich použití. Pomocí příkladů byly také získány zkušenosti s tvorbou v jednotlivých knihovnách, takže je možné je i subjektivněji porovnat na základě toho, jak dobře se s nimi pracuje. V neposlední řadě je s jejich pomocí možné ilustrovat některé pojmy z oblasti počítačové grafiky jako typy geometrií a podobně. Součástí této kapitoly jsou drobné ukázky zdrojových kódů, které byly vybrány z praktických příkladů. Při tvorbě této kapitoly bylo využito informací z dokumentací knihoven Three.js [22], Babylon.js [23] a z tutoriálů zabývajících se knihovnou SceneJS [24].

4.1.1 Implementace příkladů

Příklady jsou implementovány v jazyce JavaScript, resp. v nadstavbě TypeScript. Při implementaci se objevilo několik problémů, které je nutné vyjasnit ještě před samotnými ukázkami.

4.1.1.1 Třídy

TypeScript umožňuje mimo jiné i podporu tříd, které jsou tak v příkladech naprogramovány standardně s konstruktorem, atributy a metodami. Každá složka v projektu se věnuje jedné knihovně a obsahuje několik tříd, které představují jednotlivé příklady v dané knihovně. Všechny třídy v každé složce jsou potomky jedné základní třídy, která obsahuje například metodu pro přizpůsobování obrazu při změně velikosti okna prohlížeče, která je pro všechny potomky stejná.



Obrázek 2: Diagram tříd ve složce příkladů s knihovnou Three.js

4.1.1.2 Typové definice

Aby bylo možné pracovat s knihovnami napsanými v JavaScriptu typově, je nutné vložit do projektu jejich typové definice. Ty je možné nalézt již vytvořené na Internetu nebo je ke knihovně napsat. Největší databází typových definic, ze které bylo čerpáno i v příkladech v této práci, je Definitely Typed[22] na serveru GitHub.

U knihovny Babylon.js je nutné navíc přidat typové definice užívaných knihoven[23], aby se předešlo chybám. Knihovna SceneJS bohužel zatím nemá připravenou typovou definici, proto je v příkladech pracováno přímo s knihovnou v JavaScriptu.

4.1.1.3 Klíčové slovo this

Užití klíčového slova `this` může být v JavaScriptu někdy problémové. V JavaScriptu existují tzv. callback funkce, což jsou funkce vložené do parametru jiné funkce. Tato funkce callback funkci uloží do proměnné a zavolá, když je potřeba.

Klíčové slovo `this` uvnitř metody standardně značí objekt, nad kterým se metoda volá. Pokud je ale vložena jako parametr funkce jiná funkce (callback) a je použito klíčové slovo `this`, tak toto klíčové slovo neoznačuje objekt, nad kterým se má callback funkce zavolat, ale objekt, ze kterého je později volána (po uložení do proměnné).

---

[22] https://github.com/borisyankov/DefinitelyTyped
[23] https://github.com/BabylonJS/Babylon.js/tree/master/References

Řešení tohoto problému existuje více. V této práci byl zvolen postup, kdy se callback funkce obalí anonymní funkcí zapsanou pomocí lambda výrazu (ten je dostupný v TypeScriptu). Tak se jako hodnota proměnné `this` nastaví odkaz na objekt, ze kterého je funkce, jejímž parametrem je callback funkce, volána. Tento objekt je poté v přeloženém souboru v JavaScriptu reprezentován proměnnou `_this`.

4.1.2 Příprava scény

Tato kapitola ukazuje přípravu scény a její vykreslení v různých knihovnách.

4.1.2.1 Three.js

V HTML dokumentu stačí vložit odkaz na soubor knihovny Three.js, viz příklad níže.

```
<script src="../lib/three.js"></script>
```

Zdrojový kód 7: Odkaz na knihovnu Three.js v HTML dokumentu

V souboru s příkladem je nutné inicializovat scénu, do které se budou objekty vkládat, pomocí třídy `Scene`. Následně se inicializuje `renderer`, který scénu vykresluje. Poté je nastavena jeho barva a velikost. Nakonec je DOM element rendereru (tj. canvas, na který se vykresluje) vložen do stránky do tagu `body`.

```
// inicializace sceny a rendereru
this.scene = new THREE.Scene();
this.renderer = new THREE.WebGLRenderer({ alpha: true });

// nastaveni barvy a velikosti
this.renderer.setClearColor(0xffffff, 1);
this.renderer.setSize(window.innerWidth, window.innerHeight);

document.body.appendChild(this.renderer.domElement);
```

Zdrojový kód 8: Příprava scény a rendereru s knihovnou Three.js

Nyní je již možné vytvářet objekty a vkládat je do objektu scény. Nakonec je potřeba postarat se o samotné vykreslení scény pomocí metody `render`. V metodě `requestAnimationFrame` nastala problémová situace s klíčovým slovem `this`, která je popsána v kapitole 4.1.1.3. Díky použití lambda výrazu se zde výrazem `this` myslí objekt, ve kterém je umístěna funkce `render`.

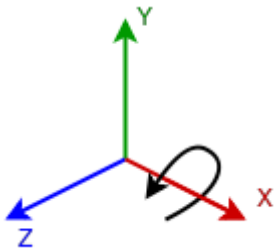
Metoda `requestAnimationFrame` má jako parametr funkci `render` (tj. jedná se o callback). Tato funkce aktualizuje animaci před dalším vykreslením. Funkce `render` obsahuje také zavolání stejnojmenné funkce `render` na rendereru, která se postará o vykreslení scény.

```
render() {
  requestAnimationFrame(() => this.render());
  this.renderer.render(this.scene, this.camera);
}
```

Zdrojový kód 9: Vykreslování s knihovnou Three.js

4.1.2.1.1 Soustava souřadnic

Při následném vkládání objektů do scény je třeba vědět, jaký typ soustavy souřadnic knihovna využívá. Knihovna Three.js užívá pravotočivou soustavu prostorových kartézských souřadnic (tu používá i OpenGL ve všech prostorech mimo prostor obrazovky, kde se soustava po transformaci změní na levotočivou).



Obrázek 3: Pravotočivá soustava souřadnic, zdroj: překresleno dle [25]

K rozpoznání pravotočivé soustavy souřadnic slouží pravidlo pravé ruky. Pokud zvedneme pravou ruku dlaní k sobě, palcem ukážeme doprava a prostředníček otočíme tak, aby směřoval k nám, pak palec označuje osu x, ukazováček osu y a prostředníček osu z.

Levotočivá a pravotočivá soustava se liší nejen v pozitivním směru osy z, ale také ve směru rotace kolem os. Pozitivní směr rotace kolem osy x je znázorněn na obrázku výše.

4.1.2.1.2 Přizpůsobení scény dle velikosti okna

Three.js neposkytuje již připravenou metodu použitelnou při změně velikosti okna, která by upravila vykreslenou scénu dle nové velikosti okna. Tu je možné vytvořit například tak, jako na příkladu níže.

```
registerResizeHandler(): void {
 window.addEventListener('resize',() => {
  if (!this.renderer
   || !this.camera
   || !this.camera['aspect']
   || !this.camera['updateProjectionMatrix'])
   return;
  this.renderer.setSize(window.innerWidth, window.innerHeight);
  (<any>this.camera).aspect =
  window.innerWidth / window.innerHeight;
  (<any>this.camera).updateProjectionMatrix();
 }, false);
}
```

Zdrojový kód 10: Funkce při změně velikosti okna s knihovnou Three.js

Tato metoda zaregistruje funkci, která se zavolá v případě změny okna. Tato funkce způsobí změnu velikosti rendereru a popřípadě i aktualizuje kameru, pokud je k dispozici. Tuto metodu je nutné zavolat po vytvoření scény.

4.1.2.2 Babylon.js

Do HTML souboru je třeba vložit `script` tagy s odkazy na `.js` soubory, stejně jako v případě knihovny Three.js. Jako první je uvedena knihovna Babylon.js, druhá je knihovna hand.js, kterou Babylon.js používá. Třetí je volitelná knihovna cannon.js, která zahrnuje fyziku do herních aplikací.

```
<script src="../lib/babylon.2.0.js"></script>
<script src="../lib/hand-1.3.8.js"></script>
<script src="../lib/cannon.js"></script>
```

Zdrojový kód 11: Odkaz na knihovnu Babylon.js a volitelné knihovny

Dále je třeba vložit do HTML dokumentu `canvas` element:

```
<canvas id="canvas"></canvas>
```

Zdrojový kód 12: Vložení canvas elementu

V samotném `.js` (v případě příkladů v této práci nejdříve `.ts`) souboru je potřeba právě element `canvas`, kterému je dále v příkladu nastavena šířka a výška.

```
this.canvas = <HTMLCanvasElement>document.getElementById("canvas");
this.canvas.width = window.innerWidth;
this.canvas.height = window.innerHeight;
```

Zdrojový kód 13: Práce s canvas elementem

Následně je vytvořena instance třídy `Engine`. V příkladu je také vytvořena instance scény `scene`, přičemž v konstruktoru je jí předán objekt `engine`. Následně je pomocí metody `window.addEventListener` zaregistrována anonymní funkce obsluhující událost `resize`, tj. změnu velikosti okna. V této anonymní funkci se zavolá metoda `resize` na vytvořené instanci `engine`, která se postará o příslušnou úpravu scény.

```
this.engine = new BABYLON.Engine(this.canvas, true);
this.scene = new BABYLON.Scene(this.engine);
window.addEventListener("resize", () => {
 this.engine.resize();
});
```

Zdrojový kód 14: Vytvoření instancí tříd Engine a Scene s knihovnou Babylon.js

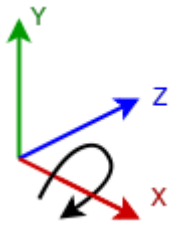
O samotné vykreslování při každém snímku se postará právě `engine` pomocí metody `runRenderLoop`, která jako callback zaregistruje anonymní funkci volající metodu `render` na scéně.

```
this.engine.runRenderLoop(() => {
            this.scene.render();
        });
```

Zdrojový kód 15: Vykreslování s knihovnou Babylon.js

4.1.2.2.1 Soustava souřadnic

Babylon.js využívá na rozdíl od SceneJS a Three.js levotočivou kartézskou soustavu souřadnic (tu využívá například také rozhraní DirectX).



Obrázek 4: Levotočivá soustava souřadnic, zdroj: překresleno dle [25]

U levotočivé soustavy souřadnic směřuje kladná osa z opačným směrem než u levotočivé. Směr rotace kolem os je opačný než u pravotočivé soustavy souřadnic, viz obrázek výše.

4.1.2.3 SceneJS

SceneJS obsahuje na rozdíl od ostatních knihoven dynamické načítání dalších částí knihovny. O tom informuje i text o knihovně na serveru GitHub (volně přeloženo): *„Aby byla udržena malá velikost jádra knihovny, SceneJS načítá na požádání ostatní funkcionalitu dynamicky ze složky s plug-iny na GitHub repository.“* [26]

U knihovny SceneJS není nutné vytvářet element `canvas`, knihovna totiž vytvoří jak `div` element s vnořeným elementem `canvas`, tak `div` element pro prvky zobrazující načítání jednotlivých plug-inů. Dokonce automaticky zařídí i přizpůsobení obrazu při roztahování okna.

Do HTML dokumentu je třeba vložit soubor s knihovnou:

```
<script src="../lib/scenejs.js"></script>
```

Zdrojový kód 16: Odkaz na knihovnu SceneJS v HTML dokumentu

SceneJS má jiný styl API než Three.js a Babylon.js, scéna se zde vytváří pomocí uzlů (tzv. `nodes`). V příkladu níže je vytvořen jeden uzel typu `renderer`, kterému je nastavena barva a čištění `depth` a `color` bufferu.

```
this.scene = SceneJS.createScene({
 nodes: [
  {
   type: "renderer",
   clearColor: { r: 0, g: 0, b: 0 },
   clear: {
   depth: true,
   color: true
  }
 ]
});
```

Zdrojový kód 17: Vložení rendereru s knihovnou SceneJS

Nakonec je třeba scénu vykreslit pomocí metody `render`.

```
render() {
 requestAnimationFrame(() => this.render());
}
```

Zdrojový kód 18: Vykreslení s knihovnou SceneJS

4.1.2.3.1 Soustava souřadnic

Scene.js využívá pravotočivou soustavu souřadnic, stejně jako Three.js. Pravotočivá soustava souřadnic je popsána v kapitole 4.1.2.1.1.

4.1.3 Geometrie

Všechny tři vybrané knihovny poskytují již předdefinované geometrie, které usnadňují tvorbu základních objektů jako je krychle, koule a další. U každé knihovny je samozřejmě možné vytvořit vlastní objekt pomocí definování vrcholů a indexů, protože knihovny poskytují třídy pro práci s vektory a různé matematické operace.

Tato kapitola je zaměřena na třídy reprezentující trojrozměrné, popřípadě i dvourozměrné objekty. Tyto třídy usnadňují vývojáři přidávání základních objektů do scény a odstraňují tak nutnost „ruční“ definice objektů.

4.1.3.1 Three.js

Knihovna Three.js obsahuje řadu tříd, pomocí kterých lze vytvořit geometrické objekty bez nutnosti definování vrcholů a hran. Objekty se v knihovně tvoří pomocí třídy `Mesh` (těleso), do které se parametrem zadá geometrie a materiál. Tato kapitola se zabývá právě geometriemi, materiály se zabývá kapitola 4.1.4.

Třída `BoxGeometry`, původně nazvaná `CubeGeometry`, slouží k vytvoření trojrozměrného tělesa – kvádru, popřípadě krychle. Konstruktor má šest parametrů: `width` určuje šířku tělesa, `height` výšku a `depth` hloubku. Další parametry `widthSegments`, `heightSegments` a `depthSegments` určují, z kolika segmentů se bude daná strana skládat. Standardně je nastaven jeden segment.

V příkladu níže je do scény přidána krychle `cube`. Je vytvořeno nové těleso `Mesh` a jako parametr je mu předána geometrie `BoxGeometry` a materiál, který je popsán v pozdější kapitole. Následně je změněna pozice krychle přímo přes její souřadnice (posun by také bylo možné provést pomocí metody `translate`). Nakonec je krychle přidána do scény.

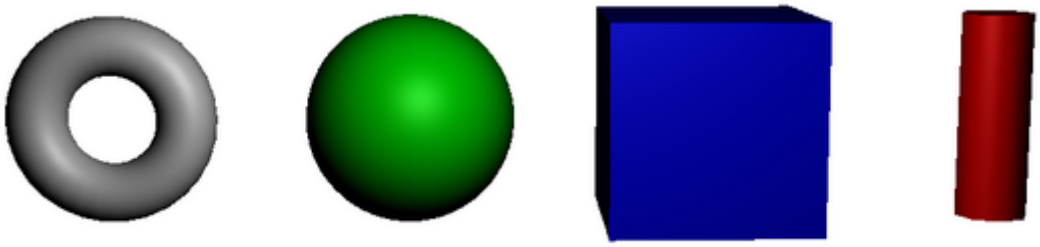
```
var cube = new THREE.Mesh(
 new THREE.BoxGeometry(1, 1, 1),
 new THREE.MeshPhongMaterial({ color: 0xff0000 }));
cube.position.x = -1;
cube.position.z = 0;

this.scene.add(cube);
```

Zdrojový kód 19: Vytvoření krychle s knihovnou Three.js

`SphereGeometry` slouží k vytvoření koule. Má sedm parametrů: `radius` slouží k nastavení poloměru, `widthSegments` a `heightSegments` značí počet segmentů, a to horizontálních, resp. vertikálních. Standardně je nastaveno osm segmentů horizontálně a šest vertikálně. Dalšími parametry jsou `phiStart` a `phiLength`, které specifikují horizontální úhel (jeho začátek a velikost) a `thetaStart` a `thetaLength` obdobně specifikují vertikální úhel. Pomocí modifikace úhlů se dají vykreslit i neúplné koule.

`PolyhedronGeometry` je hlavní třída vytvářející mnohostěn, kterou využívají další třídy: `DodecahedronGeometry` vytvoří dvanáctistěn (mnohostěn se stěnami tvořenými pětiúhelníky), `IcosahedronGeometry` dvacetistěn (stěny jsou tvořeny trojúhelníky), `OctahedronGeometry` osmistěn (stěny tvořeny trojúhelníky) a `TetrahedronGeometry` čtyřstěn (stěny taktéž tvořeny trojúhelníky).

Geometrie `CylinderGeometry` vytvoří válec, `LatheGeometry` reprezentuje osově souměrný tvar (např. váza), `TorusGeometry` torus a `TorusKnotGeometry` torusový uzel. Třída `TubeGeometry` vytvoří rouru kolem křivky.



Obrázek 5: Ukázky trojrozměrných objektů vykreslených pomocí Three.js

Třída `TextGeometry` vytvoří 3D objekt z textu vloženého v parametru konstruktoru a třída `ExtrudeGeometry` umožní převést 2D tvar do 3D geometrie. Geometrie `ParametricGeometry` vytvoří objekt definovaný funkcí.

Další geometrie se pak obdobně zabývají vytvořením dvourozměrných objektů: `CircleGeometry` vytvoří kruh, `PlaneGeometry` obdélník (čtverec), `RingGeometry` prstencový tvar. Třída `ShapeGeometry` vytvoří polygon na základě tvaru, podobně jako `ExtrudeGeometry`.

4.1.3.2 Babylon.js

Knihovna Babylon.js také obsahuje trojrozměrné geometrické objekty. Třída `Box` vytvoří krychli. Mezi parametry jejího konstrukturu patří `id` (identifikátor objektu), `scene` (scéna, do které patří) a `size` (velikost).

V příkladu níže je přidána krychle do scény pomocí metody `CreateBox`. Následně je vytvořen materiál `StandardMaterial`, jehož barva se nastaví pomocí `diffuseColor` a materiál je přiřazen. Dále je nastavena pozice krychle.

```
var box = BABYLON.Mesh.CreateBox("box", 1.0, this.scene);
var materialBox = new BABYLON.StandardMaterial("material1",
 this.scene);
materialBox.diffuseColor = BABYLON.Color3.Blue();
box.material = materialBox;

box.setAbsolutePosition(new BABYLON.Vector3(0, 0, -1));
```

Zdrojový kód 20: Vytvoření krychle s knihovnou Babylon.js

Třída `Sphere` definuje kouli a mezi její parametry patří parametr `segments`, který značí počet segmentů, a `diameter`, který označuje průměr. Třída `Cylinder` reprezentuje válec, `Torus` torus a `Torus Knot` torusový uzel.

Je možné definovat i dvourozměrné objekty. Třída `Plane` reprezentuje čtverec, třída s názvem `Ground` vytvoří obdélník. Zajímavostí je třída `TiledGround`, která vytvoří členěný, „kachlovitý“ povrch.

4.1.3.3 SceneJS

Knihovna SceneJS obsahuje typy uzlů, které umožňují definovat geometrické objekty. Tyto uzly je nutné vložit do stromu scény, a to například již v metodě `createScene`, která je představena v kapitole 4.1.2.3. Do scény je možné také přidávat uzly pomocí metody `addNode`.

V příkladu níže je zobrazen uzel typu `material`, který udává vlastnosti materiálu (materiál zde bude mít červenou barvu). Tento uzel v sobě obsahuje další uzel typu `translate`, který určuje posun uzlů, které jsou v něm vnořené. V tomto uzlu je tedy vnořen samotný uzel typu `geometry/box`, který má vlastnosti `xSize`, `ySize` a `zSize`. Ty nastavují velikost kvádru. Dále může obsahovat vlastnost `wire`, pomocí které se vykreslí drátěný model krychle. Tato vlastnost je standardně nastavena na hodnotu `false`.

```
{
 type: "material",
 color: { r: 1, g: 0, b: 0 },
 nodes: [
  {
   type: "translate",
   x: -2,
   y: 0,
   z: 0,

   nodes: [
    {
     type: "geometry/box",
     xSize: 1,
     ySize: 1,
     zSize: 1
    }
   ]
  }
 ]
}
```

Zdrojový kód 21: Vytvoření krychle s knihovnou SceneJS

Pro vytvoření koule složí typ uzlu `geometry/sphere`. V objektu uzlu jsou poté definovány vlastnosti `latitudeBands`, `longitudeBands` a `radius`. První udává počet horizontálních segmentů, druhá počet vertikálních segmentů a třetí vlastnost udává poloměr koule. Standardně je počet segmentů nastaven na hodnotu třicet a poloměr na hodnotu jedna. Také koule, stejně jako krychle, obsahuje vlastnost `wire`, díky které se vykreslí drátěný model objektu.

Válec se do scény vloží pomocí uzlu typu `geometry/cylinder`, torus pomocí `geometry/torus`. Je možné vytvořit dokonce i známou konvici z Utahu pomocí typu

uzlu `geometry/teapot`. Zajímavostí je také uzel typu `geometry/heightmap`, který vytváří výškovou mapu.

Z hlediska 2D objektů je ve SceneJS typ uzlu `geometry/plane` jako reprezentace obdélníku. Pomocí `geometry/vectorText` je možné vložit do scény text, který se předtím do uzlu napíše jako hodnota vlastnosti `text`.

4.1.4 Materiály

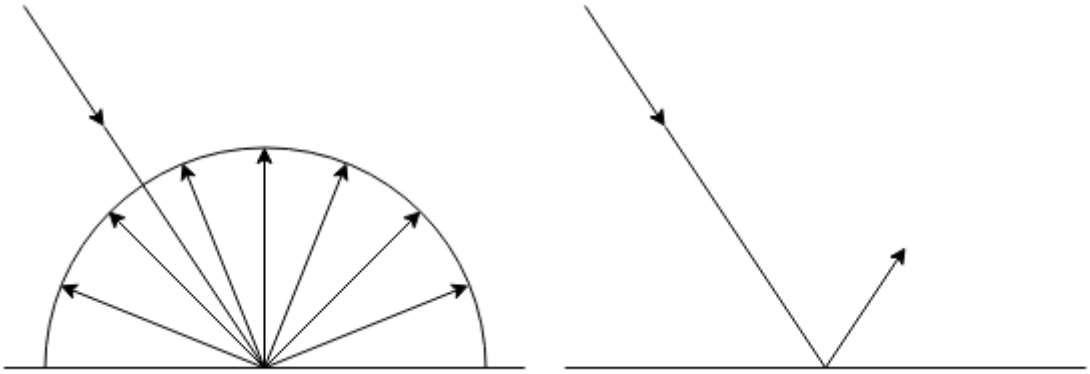
Všechny tři knihovny obsahují třídy (resp. uzly v případě knihovny SceneJS), které umožňují definovat materiály objektů. Ty často pracují s odrazem světla, a to jak difúzním, tak zrcadlovým.

4.1.4.1 Three.js

Knihovna Three.js obsahuje třídy pro definici materiálu určitého objektu. Přidání materiálu k objektu je vidět v kapitole 4.1.3.1. Generickou třídou je třída `Material`. Nejjednodušším materiálem je `MeshBasicMaterial`, který objekt vykreslí pouze v dané barvě (nebo jako drátěný model). `MeshDepthMaterial` vykresluje barvu objektu dle hloubky – část blíže k pozorovateli je vykreslena bílou barvou, část dále je vykreslena černou barvou. `MeshFaceMaterial` slouží k přidání více materiálů k jedné geometrii, tj. na každé straně může být použit jiný materiál.

`MeshLambertMaterial` využívá Gouraudovo stínování, tj. nejdříve vypočítá normálový vektor a (z něho) barvu ve vrcholech, kterou následně interpoluje. Jedná se o Lambertovský povrch, takže způsobuje difúzní odraz, který je vidět na obrázku níže (vlevo).

Difúzní odraz se odráží do všech směrů. Tak se liší od zrcadlového odrazu, který se odráží pod zrcadlově souměrným úhlem ke směru dopadu [6]. Ideálně zrcadlový odraz je zobrazen na obrázku níže (vpravo).



Obrázek 6: Ideálně difúzní a ideálně zrcadlový odraz, zdroj: překresleno dle [27]

`MeshPhongMaterial` využívá Phongovo stínování, u kterého se barva vypočítá na základě již interpolovaných normálových vektorů. Dále je v této třídě využit Phongův osvětlovací model.

„Phongův osvětlovací model rozlišuje tři druhy odrazu světla od materiálu. Z těchto případů se pak skládá výsledný obraz. Odraz je rozdělený na zrcadlový (spekulární), difúzní a ambientní. Zrcadlový odraz není v pravém slova smyslu ideálním zrcadlovým odrazem, ale spíše odpovídá modelu lesklého povrchu.“ [6, s. 333]

Další třída, `MeshNormalMaterial`, mapuje normálové vektory na barvy. K vykreslení částicových systémů slouží materiál s názvem `PointCloudMaterial`. U třídy `ShaderMaterial` je možné nadefinovat vlastní shadery, stejně tak jako u třídy `RawShaderMaterial` (u ní je nutné ručně definovat více atributů).

Materiál `SpriteCanvasMaterial` slouží k vytvoření materiálu, který vykresluje tzv. sprity (malé 2D obrázky) na 2D canvas. `SpriteMaterial` je materiálem pro třídu Sprite, která vytváří obdélník ve 3D scéně, jenž je vždy obrácen směrem k pozorovateli. Dále je možné použít třídy `LineBasicMaterial` a `LineDashedMaterial`, které slouží k vykreslení drátěných modelů celými, resp. přerušovanými čarami.

U tříd `MeshBasicMaterial`, `MeshLambertMaterial`, `MeshPhongMaterial`, `PointCloudMaterial` a `SpriteMaterial` se nachází atribut `map`, který umožňuje definovat texturu, která se má na objekt namapovat. Příklad materiálu s texturou je zobrazen níže.

```
var material = new THREE.MeshPhongMaterial({
 map: THREE.ImageUtils.loadTexture('../images/UHK.png')
})
```

Zdrojový kód 22: Textura s knihovnou Three.js

4.1.4.2 Babylon.js

Knihovna Babylon.js nabízí tři typy materiálů. Prvním z nich je `StandardMaterial`, který obsahuje řadu atributů, které pomáhají tento materiál definovat. Použití tohoto materiálu je zobrazeno v kapitole 4.1.3.2. Jeho atributem je dle článku [28] například `diffuseColor`, který určuje základní barvu objektu (s difúzním odrazem světla). Výsledná barva závisí nejen na difúzní barvě materiálu, ale také na difúzní barvě světla. Dalším atributem je `specularColor`, což je barva zrcadlové složky. Ta závisí na zrcadlové barvě světla i materiálu. `AmbientColor` je atribut určující barvu objektu vzhledem k ambientní barvě scény. Atribut `emissiveColor` definuje barvu objektu, která je dána, i když objekt není osvětlen.

`StandardMaterial` obsahuje i atributy pro definici textur. Atribut `diffuseTexture` je zobrazen na příkladu níže.

```
var material = new BABYLON.StandardMaterial("material",
 this.scene);
material.diffuseTexture = new BABYLON.Texture("images/UHK.png",
 this.scene);
cube.material = material;
```

Zdrojový kód 23: Textura s knihovnou Babylon.js

Mimo difúzní textury je zde k dispozici i zrcadlová textura (mapuje texturu na místo zrcadlového odrazu), ambientní (okolní) textura, emisivní (textura bez světla), textura odrazu a bump textura, která je mapována pomocí tzv. bump mapování. To vytváří simulaci nerovného povrchu.

Dalším typem materiálu je `ShaderMaterial`, který umožňuje nadefinování vlastních shaderů, a to samozřejmě bez nutnosti složité práce s nimi ve WebGL kódu. Posledním typem materiálu je třída `MultiMaterial`, která umožňuje definici více materiálů pro jeden objekt (část objektu má jeden materiál, druhá část jiný materiál a podobně). `MultiMaterial` se vytvoří složením z materiálů typu `StandardMaterial`.

4.1.4.3 SceneJS

Stejně jako ostatní knihovny obsahuje i knihovna SceneJS prostředek pro definování materiálů. Jedná se o uzel typu `material`, jehož použití je vidět v kapitole 4.1.3.3. Objekty v podstromu tohoto uzlu poté mají nastaven tento materiál.

Uzel `material` obsahuje atribut `color`, kterým je definována základní barva materiálu (tj. difúzní) pomocí RGB složek barvy (v hodnotách od 0.0 do 1.0). Další vlastností je `specularColor`, která udává barvu zrcadlového odrazu. Vlastnost `specular` slouží k definici intenzity zrcadlového odrazu. Vlastnost `emit` značí celkovou zářivost a `shine` velikost zrcadlového odrazu (při větších hodnotách se odraz více soustředí do jednoho bodu). `Alpha` slouží k definici průhlednosti.

U knihovny SceneJS je definován následující materiál standardně již při vytvoření scény. Tento materiál bude tedy nastaven pro objekty ve scéně i v případě, pokud bude úplně chybět uzel typu `material`.

```
{
 type: "material",
 color: { r: 1.0, g: 1.0, b: 1.0 },
 specularColor: { r: 1.0, g: 1.0, b: 1.0 },
 specular: 1.0,
 shine: 70.0,
 emit: 0,
 alpha: 1.0,
}
```

Zdrojový kód 24: Standardní materiál s knihovnou SceneJS

SceneJS obsahuje také uzel typu `texture`, jehož textura se aplikuje na vnořené uzly typu `geometry`, jak je vidět v příkladu níže. Textura může mít více vrstev [29], přičemž každá může být aplikovaná na dané vlastnosti materiálu (jedná se o vlastnosti `baseColor`, `specular`, `alpha` a `emit`). Textura může být také aplikována přímo na vlastnost geometrie s názvem `normals`. To slouží k provedení tzv. bump mapování, které simuluje nerovný povrch.

```
{
   type: "texture",
   src: "images/UHK.jpg",
   applyTo: "color",

   nodes: [
    {
     // Objekty, na ktere bude namapovana textura
    }
   ]
  }
```

Zdrojový kód 25: Textura s knihovnou SceneJS

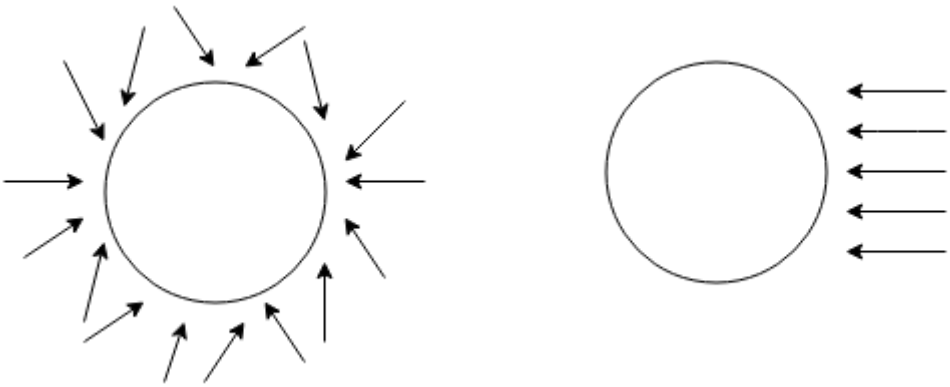
4.1.5 Světla

Tato kapitola se zabývá poskytovanými světelnými zdroji v knihovnách Three.js, Babylon.js a SceneJS. V knize Moderní počítačová grafika je o světelných zdrojích řečeno: *„Světelný zdroj vyzařuje světelné záření. […] Nejjednodušším případem světelného zdroje je bodový světelný zdroj, který svítí konstantní barvou a intenzitou izotropně do všech stran.“* [6, s. 336]

Bodový zdroj, stejně jako rovnoběžný zdroj, poskytují všechny tři knihovny.

4.1.5.1 Three.js

Třída `Light` je abstraktní třídou v knihovně Three.js, ze které dědí ostatní světelné zdroje. Třída `AreaLight` vytváří plošný zdroj, který osvětluje scénu ze svého povrchu. Tato třída funguje pouze s rendererem s názvem `WebGLDeferredRenderer`. Třída `HemisphereLight` reprezentuje světelný zdroj umístěný nad scénou. Knihovna poskytuje i tzv. ambientní (okolní) světlo v podobě třídy `AmbientLight` (okolní světlo je vidět na obrázku níže vlevo). Barva tohoto světla se započítává do barev všech objektů ve scéně [30].



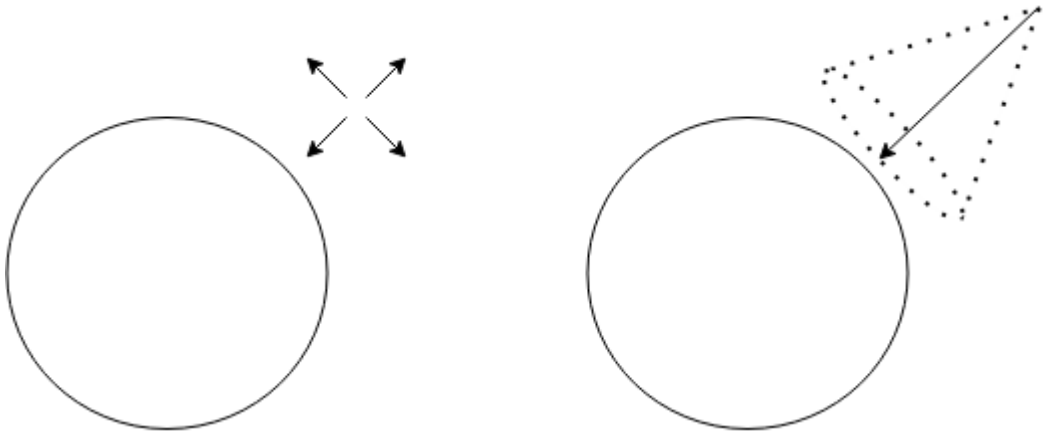
Obrázek 7: Okolní světlo a rovnoběžný světelný zdroj, zdroj: překresleno dle [31]

`DirectionalLight` produkuje světlo, které míří jedním směrem (jedná se o rovnoběžný zdroj). Tento světelný zdroj je vidět na obrázku výše (vpravo). Světelný zdroj `PointLight` je bodovým zdrojem světla, tj. jeho světlo „svítí“ do všech okolních směrů. Níže je uveden příklad přidání bodového světla do scény v knihovně Three.js.

```
var pointLight = new THREE.PointLight(0xffffff, 1);
pointLight.position.set(3, 0, 1);
this.scene.add(pointLight);
```

Zdrojový kód 26: Bodový světelný zdroj s knihovnou Three.js

Třída `SpotLight` reprezentuje světelný zdroj známý jako reflektor, který je podobný kuželu a je směrově závislý [6]. Na obrázku níže lze vidět bodový zdroj světla (vlevo) a reflektor (vpravo). Je třeba říci, že poslední tři světelné zdroje umí ovlivnit pouze materiály `MeshLambertMaterial` a `MeshPhongMaterial` [30].



Obrázek 8: Bodový světelný zdroj a reflektor, zdroj: překresleno dle [31]

4.1.5.2 Babylon.js

Světla v knihovně Babylon.js produkují difúzní a zrcadlovou barvu odrazu. Materiály poté s pomocí těchto barev spočítají barvu každého pixelu. Sice je možné do scény přidat libovolný počet světel, je ale třeba si uvědomit, že jeden materiál typu `StandardMaterial` dokáže vypočítat výslednou barvu pouze pro čtyři zároveň působící světla [32].

Knihovna Babylon.js nabízí tato světla: `PointLight` (bodové), `DirectionalLight` (směrové), `SpotLight` (reflektor) a `HemisphericLight`. Poslední jmenované slouží k realistickému zobrazení ambientního světla a zadává se pomocí barvy pixelů, které jsou otočeny „nahoru“, a barvy pixelů, které jsou otočeny „dolů“. Níže je uveden příklad vložení bodového světla do scény.

```
var pointLight = new BABYLON.PointLight("pointLight",
 new BABYLON.Vector3(3, 0, 1), this.scene);
pointLight.diffuse = new BABYLON.Color3(1, 1, 1);
pointLight.specular = new BABYLON.Color3(1, 1, 1);
```

Zdrojový kód 27: Bodový světelný zdroj s knihovnou Babylon.js

4.1.5.3 SceneJS

SceneJS vytváří některá světla již automaticky, stejně tak jako materiály. Bez jakéhokoliv definování světel poskytuje dva směrové zdroje světla a jeden ambientní zdroj světla [33]. Tyto zdroje je možné v kódu přepsat.

Knihovna SceneJS poskytuje uzel typu `lights`, do kterého se vkládají světla. Níže je příklad bodového světla. To má nastaveno vlastnost `mode` na hodnotu `point`.

```
type: "lights",
lights: [
 {
  mode: "point",
  color: { r: 1.0, g: 1.0, b: 1.0 },
  diffuse: true,
  specular: true,
  pos: { x: 6.0, y: 0, z: 10.0 },
  space: "world"
 }
]
```

Zdrojový kód 28: Bodový světelný zdroj s knihovnou SceneJS

Knihovna dále poskytuje světlo rovnoběžné (`dir`) a ambientní (`ambient`). U směrového i bodového světla se dá nastavit vlastnost `space` na hodnotu `view` nebo `world`. Hodnota `world` zajistí pozici světel v souřadnicích světa, takže se pohybují zároveň se scénou a pozorovatel může vidět neosvětlenou část scény, což nejde u hodnoty `view`.

4.1.6 Kamery

Kamera může scénu promítat perspektivně (středově) a ortogonálně (rovnoběžně). U rovnoběžného promítání mají promítací paprsky stejný směr, u středového vycházejí z jednoho bodu. Středové promítání odpovídá lidskému vidění světa, protože při vzrůstající vzdálenosti od pozorovatele se zmenšuje velikost objektů [6].

Všechny knihovny obsahují mimo základních kamer také speciální třídy, které umožňují ovládat kameru, a to například pomocí myši.

4.1.6.1 Three.js

Knihovna Three.js obsahuje abstraktní třídu `Camera`. Z ní jsou odvozeny třídy `OrthographicCamera` (využívá rovnoběžné promítání) a `PerspectiveCamera` (využívá středové promítání). Dále je k dispozici `CubeCamera`, která vytvoří jednu perspektivní kameru pro každou stranu krychle.

Three.js poskytuje velké množství tříd pro ovládání kamery. Jednou z nich je například třída `OrbitControls`, kde se kamera pohybuje po „orbitu“ kolem scény. Dalšími třídami jsou například `TrackballControls` (umožňuje otočení kamery vzhůru nohama) a `FirstPersonControls`.

4.1.6.2 Babylon.js

V knihovně Babylon.js jsou kamery děleny do dvou základních tříd: `ArcRotateCamera` a `TargetCamera`. V první třídě kamera rotuje kolem daného místa. Do druhé třídy patří například kamera `FreeCamera`, která reprezentuje kameru s pohledem z první osoby. Dále sem patří kamery, které se často soustředí na ovládání kamery ve speciálním zařízení (například `OculusCamera` a `GamepadCamera`). U kamer lze nastavit i ortogonální promítání.

4.1.6.3 SceneJS

V knihovně SceneJS je základní kamera v jádře knihovny a je definována pomocí uzlu typu `camera`. Tento uzel má vlastnost `optics`, ve které je blíže specifikován typ promítání. Promítání může být buď `perspective` (středové), `frustum` (také středové, ale s definicí komolého promítacího jehlanu) a `ortho` (rovnoběžné).

Typy uzlů k ovládání kamery jsou umístěny ve složce s plug-iny a jsou dynamicky načítány. Mezi ně patří uzly typu `cameras/orbit`, `cameras/trackball` a `cameras/pickFlyOrbit` (zajistí přiblížení kamery k bodu, na který bylo kliknuto).

4.1.7 Import objektů

Trojrozměrné objekty mohou být vytvořené v grafických programech (například program Blender) a uložené v různých formátech, jako například BLEND (koncovka .blend), DAE neboli COLLADA (.dae), OBJ (.obj) a tak dále. Tyto objekty je pak potřeba importovat do scény, k čemuž nabízí knihovny své nástroje.

4.1.7.1 Three.js

Knihovna Three.js má vlastní JSON formát objektů. Do něj je možné objekty přímo exportovat v různých grafických programech pomocí připravených doplňků, které stačí do programu přidat. Tak je možné vyexportovat objekt z programů Blender, 3DS Max, Maya a Autodesk Revit. K přímému konvertování do JSON formátu z různých formátů slouží konvertory, které Three.js taktéž poskytuje ve své repository na serveru GitHub.

Pro následné přidání objektu v JSON formátu do scény slouží třída `JSONLoader` (viz příklad níže) a pro společné přidání objektu ve formátu OBJ s materiály ve formátu MTL slouží třída `OBJMTLLoader`. Three.js dále obsahuje několik dalších tříd k načítání objektů (například samostatný `OBJLoader` a `MTLLoader`).

```
var loader = new THREE.JSONLoader();

loader.load('../objects/deer.json', (geometry, materials) => {
 var material = new THREE.MeshFaceMaterial(materials);
 var deer = new THREE.Mesh(geometry, material);
 this.scene.add(deer);
});
```

Zdrojový kód 29: Vložení objektu do scény s knihovnou Three.js

4.1.7.2 Babylon.js

Knihovna Babylon.js má vlastní formát s JSON daty s příponou `.babylon`. Stejně jako knihovna Three.js, poskytuje i Babylon.js doplňky, pomocí kterých je možné exportovat objekty z grafických programů do tohoto formátu. Mezi grafické programy, ze kterých je export možný, patří Cheetah 3D, Unity 5, Blender a 3DS Max. Dále je možné do požadovaného formátu konvertovat soubory formátů FBX, OBJ a DAE. Všechny tyto doplňky lze opět nalézt v repository knihovny na serveru GitHub.

Soubory s příponou `.babylon` je pak možné nahrát do scény pomocí třídy `SceneLoader`, viz příklad níže.

```
BABYLON.SceneLoader.ImportMesh("", "../objects/", "deer.babylon",
 this.scene);
```

Zdrojový kód 30: Vložení objektu do scény s knihovnou Babylon.js

Zajímavým nástrojem je aplikace Babylon.js Sandbox[24], ve které je možné si vyzkoušet vložení objektu do scény.

4.1.7.3 SceneJS

Knihovna SceneJS poskytuje dynamicky načítané plug-iny pro import objektů. K importu objektů ve formátu OBJ slouží uzel typu `import/obj` (viz příklad níže), k importu objektů ve formátu MD2 typ uzlu `import/md2` a k importu 3DS objektů `import/3ds`.

```
{
 type: "texture",
 src: "../objects/Deer_body_D.png",

 nodes: [
  {
   type: "import/obj",
   src: "../objects/Deer.obj"
  }
 ]
}
```

Zdrojový kód 31: Vložení objektu do scény s knihovnou SceneJS

[24] http://www.babylonjs.com/sandbox/

Pro konvertování COLLADA souborů slouží malá knihovna SceneJS-PyCollada[25].

4.1.8 Zhodnocení knihoven na základě zkušenosti z tvorby příkladů

Díky množství příkladů a celkem propracované dokumentaci se nejlépe pracovalo s knihovnou Three.js. Ta také poskytuje širokou škálu nástrojů. Nejhůře se oproti tomu pracovalo s knihovnou SceneJS, ke které chybí dokumentace a vlastnosti poskytovaných nástrojů se musí více dohledávat. Její API je navíc velice specifické a při tvorbě složitých scén ztrácí na přehlednosti. Nejvíce možností pro import objektů poskytuje také knihovna Three.js.

4.2 Porovnání snímkových frekvencí

Měření snímkových frekvencí proběhlo v jednoduchých aplikacích, které zobrazují scénu s volitelným počtem krychlí. Aplikace (pro každou z knihoven jedna) jsou dostupné na webové stránce na přiloženém CD.

4.2.1 Aplikace

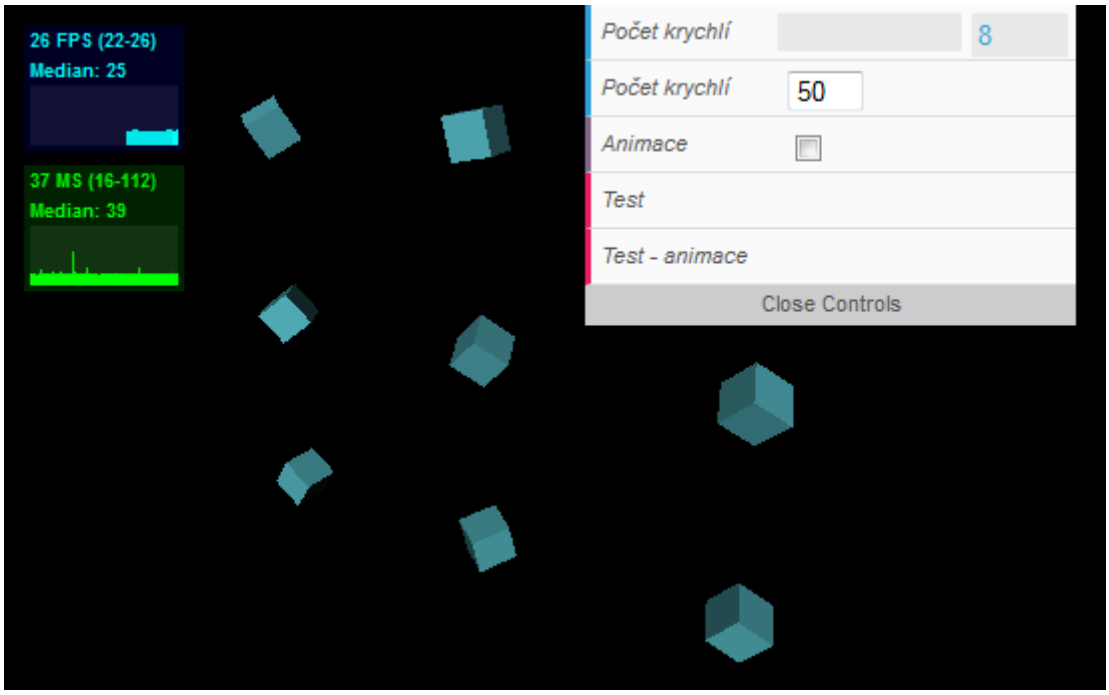
Každá aplikace se skládá z elementu `canvas`, který zabírá celou velikost obrazovky. Na něm je vykreslována daná scéna (ta je implementována v knihovně, ke které se aplikace vztahuje). Dále se zde nachází měření počtu snímků za sekundu a také měření počtu milisekund potřebných k vykreslení jednoho snímku. Měření je implementováno pomocí knihovny Stats.js[26].

V rámci práce byla do knihovny Stats.js přidána metoda na získání mediánu ze snímků za sekundu i milisekund k vykreslení snímku. Právě medián těchto hodnot bude porovnán nejen v jednotlivých knihovnách, ale také v různých prohlížečích a na různých počítačových sestavách.

V pravé části obrazovky jsou umístěny ovládací prvky, které jsou implementovány pomocí knihovny dat.GUI[27]. Pomocí těchto prvků je možné měnit počet vykreslovaných krychlí, zapnout rotaci krychlí, spustit test statické scény (ten automaticky

[25] https://github.com/xeolabs/scenejs-pycollada
[26] https://github.com/mrdoob/stats.js/
[27] https://code.google.com/p/dat-gui/

vykresluje daný počet krychlí a vypisuje medián snímků za sekundu a milisekund do konzole) a test dynamické scény (taktéž, pouze s animovanou scénou).



Obrázek 9: Aplikace pro měření FPS

4.2.1.1 Scéna

Samotná scéna se v každé knihovně skládá ze zvoleného počtu modrých krychlí, jednoho směrového světla a perspektivní kamery. Krychle se přizpůsobují velikosti obrazovky tak, aby byly vždy stejně velké, a také aby nezasahovaly mimo obraz.

V každé aplikaci je možné pohybovat s kamerou pomocí myši, což je implementováno pomocí obdobných nástrojů poskytovaných v různých knihovnách, které umožňují pohyb po „orbitu“ neboli kulové ploše.

4.2.2 Měření

Základní otázkou při porovnávání snímkových frekvencí bylo to, jaké prostředky WebGL využívá. Tak mohlo být provedeno několik měření při různých prostředcích, aby byl výsledek objektivnější. O tom, jak WebGL využívá hardware, pojednává kapitola 2.2.1. Výkon kódu s použitými nadstavbovými knihovnami je samozřejmě nižší, než s použitím čisté nízko-úrovňové WebGL, jak dokazuje i test SceneJS, Three.js, PhiloGL a WebGL [34]. Hodnoty snímků za sekundu souvisí také s obnovovací frekvencí monitoru, a to proto, že je k vykreslování využívána metoda `requestAnimationFrame`.

Komunitní stránky podporované W3C konsorciem tuto metodu popisují následovně (volně přeloženo): *„Na rozdíl od starších animačních technik založených na metodách* `setTimeout` *a* `setInterval` *dochází k animaci ve chvíli, kdy je systém připraven. To vede k hladší animaci a nižší spotřebě energie, protože* `requestAnimationFrame` *bere v úvahu viditelnost webové aplikace a obnovovací frekvenci monitoru.“* [35]

Častá obnovovací frekvence monitorů je dnes 60 Hz, a proto je v této práci největší naměřená hodnota snímkové frekvence rovna šedesáti. Měření FPS a MS proběhlo na třech různých počítačových sestavách a na jednom mobilním zařízení (tabletu). Výkon byl měřen nejdříve pro jednu krychli ve scéně, dále také pro 50, 500, 1 000, 2 000 a 5 000 krychlí. Jedna krychle se skládá ze dvanácti trojúhelníků (trianglů), takže měření proběhlo pro dvanáct trojúhelníků a dále pro 600, 6 000, 12 000, 24 000 a 60 000 trojúhelníků.

Pro zajištění co největší objektivnosti byla před každým měřením aktualizována daná webová stránka (i s vymazáním paměti „cache“) a nebylo tak využito připravených tlačítek pro spuštění testu. Při měření se také objevil zajímavý problém s nástrojem Developer Tools v prohlížeči Google Chrome. Pokud byl tento nástroj otevřen, pak významně stoupala měřená snímková frekvence, a to kvůli technice vykreslování vrstev, kterou Google Chrome používá. Zmenšila se totiž vykreslovací plocha, tím se zmenšil počet vrstev a zvýšila se snímková frekvence. U ostatních prohlížečů podobný problém nenastal.

4.2.2.1 První sestava: středně výkonný notebook

První sestavou byl středně výkonný notebook s integrovanou grafickou kartou spadající do „low-to-mid range“ karet dle online srovnání grafických karet [36]. Notebook měl následující konfiguraci:

- procesor: AMD Athlon™ II P360 Dual-Core Processor 2,30 GHz,
- 2 GB RAM,
- 64-bit,
- grafický adaptér: ATI Mobility Radeon HD 4200,
- operační systém: Windows 7 Professional.

Na této sestavě byla rychlost měřena v těchto verzích prohlížečů:

- Mozilla Firefox: 37. 0. 1,
- Google Chrome: 42.0.2311.90,
- Internet Explorer: 11 11.0.9600.17728.

Výsledné tabulky snímků za sekundu a milisekund potřebných k vykreslení daného snímku v různých prohlížečích jsou uvedeny v příloze B.1. U této sestavy stojí za zmínku ne moc výkonná, starší grafická karta.

Nejvýkonnějším prohlížečem se v tomto testu ukázal prohlížeč Google Chrome a nejméně výkonným Internet Explorer. V prohlížeči Google Chrome si Three.js udržela 60 FPS až do počtu padesáti krychlí ve scéně, a to jak ve statické, tak dynamické. Ostatní knihovny, stejně jako Three.js v ostatních prohlížečích, začínají až na 40 FPS, což je stále po lidské oko dobře pozorovatelná frekvence. Na scénu s jednou krychlí je to ale dost nízká hodnota, která ukazuje, že 3D grafika v prohlížeči není ještě úplně připravena na pohodlné užití na méně výkonných sestavách.

U této sestavy je výkon Three.js o poznání vyšší než výkon ostatních dvou knihoven. U prohlížeče Google Chrome za Three.js následuje Babylon.js a až třetí je SceneJS, u prohlížečů Mozilla Firefox a Internet Explorer jsou síly Babylon.js a SceneJS přibližně vyrovnané, i když za zmínku stojí o něco lepší výkon SceneJS ve statických scénách v těchto prohlížečích. To je dáno tím, že se tato knihovna orientuje na vykreslování komplexních statických modelů.

4.2.2.2 Druhá sestava: výkonný notebook

Druhou sestavou byl výkonný notebook s integrovaným grafickým adaptérem spadajícím do kategorie „high-to-mid“ adaptérů dle online srovnání [36]. Tento adaptér byl dle tohoto srovnání nejvýkonnějším ze všech adaptérů testovaných sestav. Druhá sestava měla následující konfiguraci:

- procesor: Intel® Core™ i7 CPU Q820 1,73 GHz 1,73 GHz,
- 4 GB RAM,
- 64-bit,
- grafický adaptér: ATI Mobility Radeon HD 4670,
- operační systém: Windows 8.1 Pro.

Na této sestavě byla rychlost měřena v těchto verzích prohlížečů:

- Mozilla Firefox: 37. 0. 1,
- Google Chrome: 42.0.2311.90,
- Internet Explorer: 11 11.0.9600.17690.

Výsledná data z měření jsou uvedena v příloze B.2. Stejně jako u ostatních sestav jsou i zde celkově nejlepší výsledky v prohlížeči Google Chrome a nejhorší v prohlížeči Internet Explorer. Oproti první sestavě je v tomto prohlížeči o poznání rychlejší knihovna Babylon.js, zatímco Three.js a SceneJS mají skoro stejné výsledky. V prohlížeči Mozilla Firefox jsou všechny tři knihovny téměř vyrovnané, zatímco u dynamické scény se ukazuje slabina SceneJS v dynamických scénách. V prohlížeči Internet Explorer má nejmenší hodnoty FPS knihovna SceneJS.

4.2.2.3 Třetí sestava: výkonný osobní počítač

Třetí sestavou byl výkonný osobní počítač s grafickou kartou spadající do „high-to-mid" karet dle online srovnání [36]. Počítač měl následující konfiguraci:

- procesor: Intel® Core™ i7-3770 CPU 3,40 GHz 3,40 GHz,
- 16 GB RAM,
- 64-bit,
- grafický adaptér: Intel® HD Graphics 4000,
- operační systém: Windows 8.1 Pro.

Na této sestavě byla rychlost měřena v těchto verzích prohlížečů:

- Mozilla Firefox: 37. 0. 1,
- Google Chrome: 42.0.2311.90,
- Internet Explorer: 11 11.0.9600.17690.

I v této sestavě vykazoval nejlepší výsledky prohlížeč Google Chrome, jak lze vidět v grafech v příloze B.3. V Google Chrome byly naměřeny nejlepší hodnoty FPS u knihovny Babylon.js, nejhorší u SceneJS. Three.js měla nejvyrovnanější výsledky, co se týče porovnání statické a dynamické scény, a to ve všech prohlížečích. Naproti tomu zvláště v prohlížeči Internet Explorer, ale i v prohlížeči Mozilla Firefox, byly opět vidět rozdíly ve výkonu SceneJS u statické a dynamické scény. Rozdíl se začal projevovat přibližně při vykreslení tisíce krychlí. V prohlížeči Mozilla Firefox byla nejvýkonnější

knihovnou právě Three.js, stejně tak jako v dynamické scéně v prohlížeči Internet Explorer, ve statické scéně v tomto prohlížeči byla výkonnější knihovnou Babylon.js.

4.2.2.4 Čtvrtá sestava: mobilní zařízení

Na závěr bylo měření provedeno na tabletu s konfigurací a prohlížečem:

- procesor: ARMv7 Processor rev 2 (v7l) Kortex A9 1000 MHz,
- 464 MB RAM,
- grafický adaptér: PowerVR SGX 540,
- operační systém: Android 4.3.1,
- prohlížeč: Chrome pro Android 40.0.2214.109.

Jak ve statické, tak v dynamické scéně se ukazuje převaha Three.js při počtu více jak padesáti krychlí. Při menším počtu krychlí měla nejlepší hodnoty knihovna Babylon.js (až 50 FPS), následována SceneJS (až 47 FPS). Tyto výsledky je možné vidět v příloze B.4.

4.2.3 Výsledky měření

Z výsledků měření lze vyvodit několik zajímavých závěrů. Na testovaných středně až méně výkonných sestavách (včetně mobilních zařízení) měla nejlepší výsledky knihovna Three.js, a to v jakémkoliv prohlížeči. Na výkonnějších sestavách (tj. na druhé a třetí sestavě) si všechny tři knihovny udržují výborné hodnoty FPS v prohlížečích Google Chrome a Mozilla Firefox až do počtu 1 000 krychlí, kdy začne mít v prohlížeči Google Chrome výkonnostní převahu knihovna Babylon.js.

Prohlížeč Google Chrome, který je mimochodem nejrozšířenějším prohlížečem, se na jakékoliv sestavě ukazuje jako nejvýkonnější prohlížeč pro zobrazování 3D grafiky v prohlížeči. Zvláště v prohlížečích Mozilla Firefox a Internet Explorer spuštěných na výkonnějších sestavách se projevilo zaměření knihovny SceneJS na vykreslování statických modelů, protože v dynamické scéně její hodnoty snímkových frekvencí dosahují nižších hodnot.

5 Shrnutí výsledků

Cíle zmíněné ve druhé kapitole byly naplněny pomocí zvolení vhodných kritérií na porovnání knihoven a následného zjištění, jak dané knihovny kritéria splňují. Pro co nejlepší srovnání nástrojů byly vytvořeny praktické příklady jejich použití ve vybraných knihovnách. Tyto knihovny tak bylo možné porovnat také na základě vlastní zkušenosti při tvorbě příkladů.

K porovnání rychlosti vykreslování ve snímcích za sekundu byla vytvořena jednoduchá aplikace v každé knihovně, pomocí které je možné měnit počet objektů ve scéně. Nejrychlejší knihovnou ve většině příkladů byla knihovna Three.js. Zvláště u výkonných sestav a v prohlížeči Google Chrome měla výborný výkon také knihovna Babylon.js.

Subjektivně řečeno je nejlepší knihovnou z hlediska pohodlnosti při programování knihovna Three.js, a to i díky množství příkladů a zodpovězených otázek na Internetu. Po použití zvolené metody multikriteriálního rozhodování vyšla z porovnání nejlépe knihovna Three.js následovaná knihovnou Babylon.js a SceneJS.

6 Závěry a doporučení

Použití vhodné knihovny se odvíjí jak od osobních preferencí programátora, tak od situace, ve které se daná knihovna využije. Je ale vhodné zanalyzovat, jaký prohlížeč a jaký hardware používá skupina uživatelů, která bude vytvořenou aplikaci užívat, protože výkon knihoven se velice liší v závislosti na použité počítačové sestavě a webovém prohlížeči.

Celkově z porovnání při daných kritériích vyšla nejlépe knihovna Three.js. Knihovna Babylon.js prokázala také dobrý výkon a rozsáhlé nástroje. Obě knihovny lze doporučit pro tvorbu 3D grafiky na webu, a to jak pro vizualizaci modelů, tak například pro tvorbu her. Knihovna SceneJS se hodí spíše pro vizualizaci statických modelů, než pro tvorbu dynamických scén a her, a to jak kvůli stylu svého API, tak kvůli nižším naměřeným hodnotám snímkových frekvencí v testu dynamických scén.

Seznam použité literatury

[1] ANYURU, Andreas A. *Professional WebGL Programming: Developing 3D Graphics for the Web*. 1st ed. Indianapolis, IN: Wiley Pub., Inc., 2012, p. cm. ISBN 11-199-6886-0.

[2] HTML: Living Standard. WEB HYPERTEXT APPLICATION TECHNOLOGY WORKING GROUP. *WHATWG* [online]. 2014 [cit. 2014-08-13]. Dostupné z URL: http://www.whatwg.org/specs/web-apps/current-work/multipage/

[3] HTML5. WORLD WIDE WEB CONSORTIUM. *W3C* [online]. © 2015 [cit. 2015-04-27]. Dostupné z URL: http://www.w3.org/TR/html5/

[4] User Contributions. In: *WebGL* [online]. 2012, 8. 8. 2014 [cit. 2014-08-16]. Dostupné z URL: http://www.khronos.org/webgl/wiki/User_Contributions

[5] List of Frameworks. *WebGL Frameworks: A Research Blog* [online]. © 2014 [cit. 2014-08-16]. Dostupné z URL: http://webglframeworks.org/framework-documentation/framework-list/

[6] ŽÁRA, Jiří, Bedřich BENEŠ, Jiří SOCHOR a Petr FELKEL. *Moderní počítačová grafika*. Brno: Computer Press, 2004, 609 s. ISBN 80-251-0454-0.

[7] CASTRO, Elizabeth a Bruce HYSLOP. *HTML5 a CSS3: názorný průvodce tvorbou WWW stránek*. 1. vyd. Brno: Computer Press, 2012, 439 s. ISBN 978-80-251-3733-8.

[8] FAQ. In: *WHATWG Wiki* [online]. [2015] [cit. 2015-04-21]. Dostupné z URL: https://wiki.whatwg.org/wiki/FAQ

[9] W3C oznámilo dokončení HTML5!. In: *Interval.cz* [online]. [2014] [cit. 2015-04-21]. Dostupné z URL: https://www.interval.cz/clanky/w3c-oznamilo-dokonceni-html5/

[10] WebGL performance considerations. *Unity - Manual* [online]. © 2015 [cit. 2015-04-16]. Dostupné z URL: http://docs.unity3d.com/Manual/webgl-performance.html

[11] WebGL With Three.js: Shaders. In: SOPYŁO, Maciej. *Tuts+* [online]. 2013 [cit. 2014-08-15]. Dostupné z URL: http://code.tutsplus.com/tutorials/webgl-with-threejs-shaders--net-36054

[12] TypeScript. In: *Wikipedie: Otevřená encyklopedie* [online]. 2001- [cit. 2015-01-31]. Dostupné z URL: http://cs.wikipedia.org/wiki/TypeScript

[13] WebGL - 3D Canvas graphics. *Can I use...* [online]. 2015 [cit. 2015-04-16]. Dostupné z URL: http://caniuse.com/#feat=webgl

[14] StatCounter: Global Stats. *Top 5 Desktop, Table & Console Browsers* [online]. © 1999 - 2015 [cit. 2015-04-16]. Dostupné z URL: http://gs.statcounter.com/#browser-ww-monthly-201403-201504

[15] GPU accelerating 2D Canvas and enabling 3D content for older GPUs. In: *The Chromium Blog* [online]. 2012 [cit. 2014-08-16]. Dostupné z URL: http://blog.chromium.org/2012/02/gpu-accelerating-2d-canvas-and-enabling.html

[16] 10 criteria for choosing the correct framework. In: Symfony [online]. 2014 [cit. 2014-11-02]. Dostupné z URL: http://symfony.com/ten-criteria

[17] 15 Most Important Considerations when Choosing a Web Development Framework. In: *Tuts+* [online]. 7. 12. 2009 [cit. 2015-01-30]. Dostupné z URL: http://code.tutsplus.com/tutorials/15-most-important-considerations-when-choosing-a-web-development-framework--net-8035

[18] ARMOUR, Theo. Guidelines on Choosing a WebGL Framework: Using Three.js as an Example. In: *Skupiny Google* [online]. 2011 [cit. 2015-01-30]. Dostupné z URL: https://groups.google.com/forum/#!msg/webgl-dev-list/3mGz27VXJfs/b3vzdiJkBpAJ

[19] Vícekriteriální rozhodování. KŘUPKA, Jiří, Miloslava KAŠPAROVÁ a Renata MÁCHOVÁ. *Rozhodovací procesy* [online]. © 2011 [cit. 2015-04-17]. Dostupné z URL: http://www.rozhodovaciprocesy.cz/vicekriterialni-rozhodovani.html

[20] Snímková frekvence. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-04-17]. Dostupné z URL: http://cs.wikipedia.org/wiki/Sn%C3%ADmkov%C3%A1_frekvence

[21] DUNLOP, Robert. FPS: A common yet flawed metric of game performance. In: *MVPs.org* [online]. 2003 [cit. 2015-04-17]. Dostupné z URL: https://www.mvps.org/directx/articles/fps_versus_frame_time.htm

[22] *Threejs - documentation* [online]. [2015] [cit. 2015-04-26]. Dostupné z URL: http://threejs.org/docs/

[23] *BabylonDoc* [online]. [2015] [cit. 2015-04-26]. Dostupné z URL: doc.babylonjs.com

[24] SceneJS Tutorials. *XeoLabs* [online]. © 2015 [cit. 2015-04-26]. Dostupné z URL: http://xeolabs.com/articles/learning-scenejs/

[25] Left-hand & Right-hand Coordinate Systems. KESSON, Malcolm. *CG References & Tutorials* [online]. © 2002 [cit. 2015-04-15]. Dostupné z URL: http://www.fundza.com/rib/example4/example4.html

[26] SceneJS. *GitHub* [online]. 2013 [cit. 2015-04-15]. Dostupné z URL: https://github.com/xeolabs/scenejs

[27] Stínování a viditelnost. *Algoritmy počítačové grafiky* [online]. 2000 [cit. 2015-04-20]. Dostupné z URL: https://cent.felk.cvut.cz/courses/APG/skripta/kap11/kap11.html

[28] CATUHE, David. Babylon.js: Unleash the StandardMaterial for your babylon.js game. In: *MSDN Blogs* [online]. © 2015 [cit. 2015-04-21]. Dostupné z URL: http://blogs.msdn.com/b/eternalcoding/archive/2013/07/01/babylon-js-unleash-the-standardmaterial-for-your-babylon-js-game.aspx

[29] Texture. In: *GitHub* [online]. © 2015 [cit. 2015-04-21]. Dostupné z URL: https://github.com/xeolabs/scenejs/wiki/texture

[30] STIER, Greg. WebGL and Three.js: Lighting. In: *Solution Design Group (sdg)* [online]. [2014] [cit. 2015-04-21]. Dostupné z URL: http://solutiondesign.com/blog/-/blogs/webgl-and-three-js-lighting/

[31] OfLight. *OpenFrameworks* [online]. 2015 [cit. 2015-04-21]. Dostupné z URL: http://openframeworks.cc/documentation/gl/ofLight.html

[32] CATUHE, David. Babylon.js: Using lights in your babylon.js game. In: *MSDN Blogs* [online]. © 2015 [cit. 2015-04-21]. Dostupné z URL: http://blogs.msdn.com/b/eternalcoding/archive/2013/07/08/babylon-js-using-lights-in-your-babylon-js-game.aspx

[33] The shading model. In: *GitHub* [online]. © 2015 [cit. 2015-04-25]. Dostupné z URL: https://github.com/xeolabs/xeolabs.github.com/blob/master/_drafts/2013-11-5-scenejs-lights-and-materials.md

[34] SceneJS, Three.js and PhiloGL – WebGL performance test. In: *Steffe.se* [online]. 2011 [cit. 2015-04-27]. Dostupné z URL: http://steffe.se/?p=475

[35] RequestAnimationFrame. WORLD WIDE WEB CONSORTIUM. *WebPlatform Docs* [online]. [2014] [cit. 2015-04-27]. Dostupné z URL: https://docs.webplatform.org/wiki/dom/Window/requestAnimationFrame

[36] PASSMARK SOFTWARE. *Video Card (GPU) Benchmark Charts* [online]. © 2015 [cit. 2015-04-27]. Dostupné z URL: http://www.videocardbenchmark.net

Seznam příloh

A. Seznam knihoven nad WebGL umožňujících práci s trojrozměrnou grafikou na webu
B. Výsledky měření FPS
C. Multikriteriální analýza
D. Obsah kompaktního disku

A Seznam knihoven nad WebGL umožňujících práci s trojrozměrnou grafikou na webu

Babylon.js

Blend4Web

C3DL

Cesium

CopperLicht

csg.js

CubicVR.js

Curve3D

FRAK engine

GLGE

GlowScript

gwt-g3d

GwtGL

Inka3D

J3D

Jax

JS3D

KickJS

KriWeb

Kuda

Lightgl.js

MathBox

O3D

Oak3D

OpenWebGlobe

OSG.JS

Parallax

PhiloGL

SceneJS

SpiderGL

StormEngineC

taccGL

TDL

Three.js

X3DOM

XTK

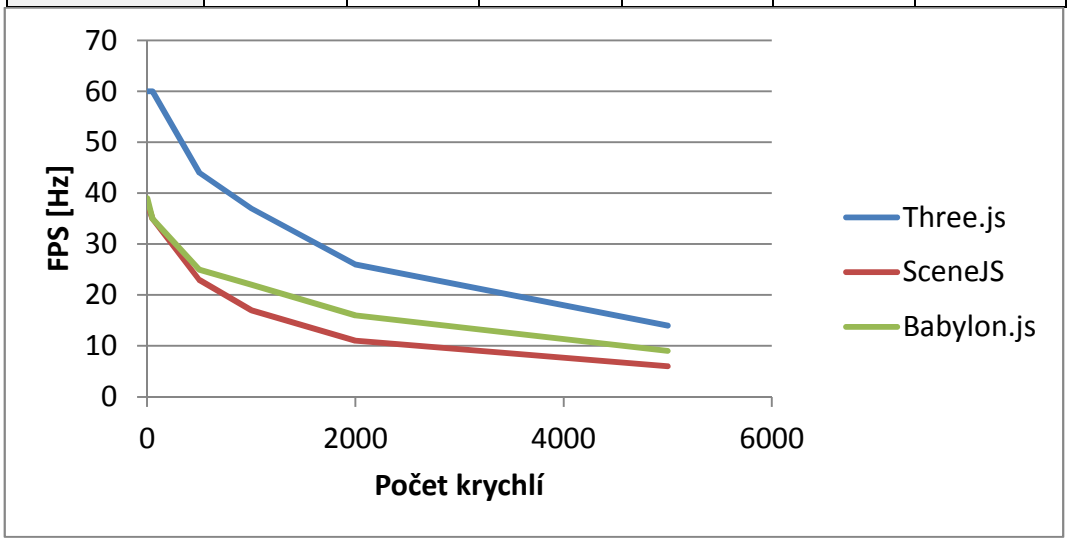
WebGLU

B Výsledky měření FPS

B.1 První sestava: středně výkonný notebook

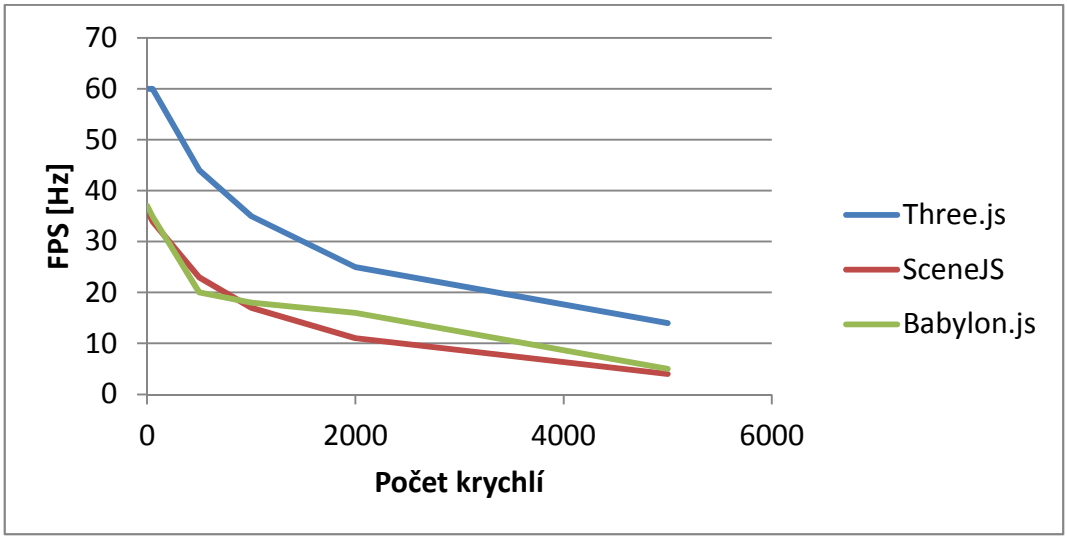
Google Chrome - statická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	60	17	38	27	39	27
50	60	17	35	29	35	29
500	44	21	23	36	25	40
1000	37	26	17	55	22	46
2000	26	36	11	61	16	61
5000	14	73	6	150	9	105



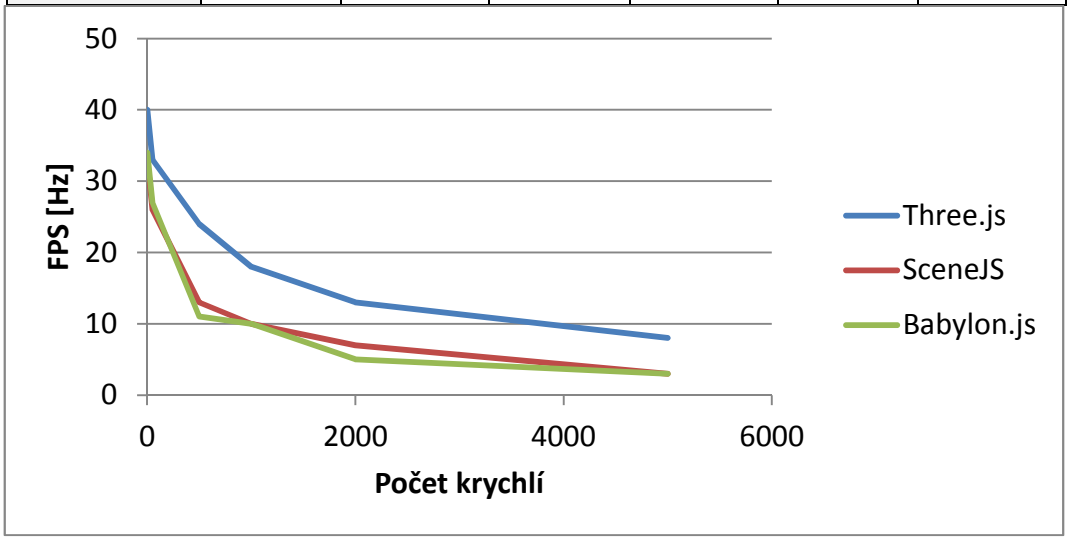
Google Chrome - dynamická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	60	17	36	28	37	27
50	60	17	34	30	35	30
500	44	23	23	37	20	49
1000	35	28	17	55	18	48
2000	25	40	11	61	16	63
5000	14	72	4	192	5	192



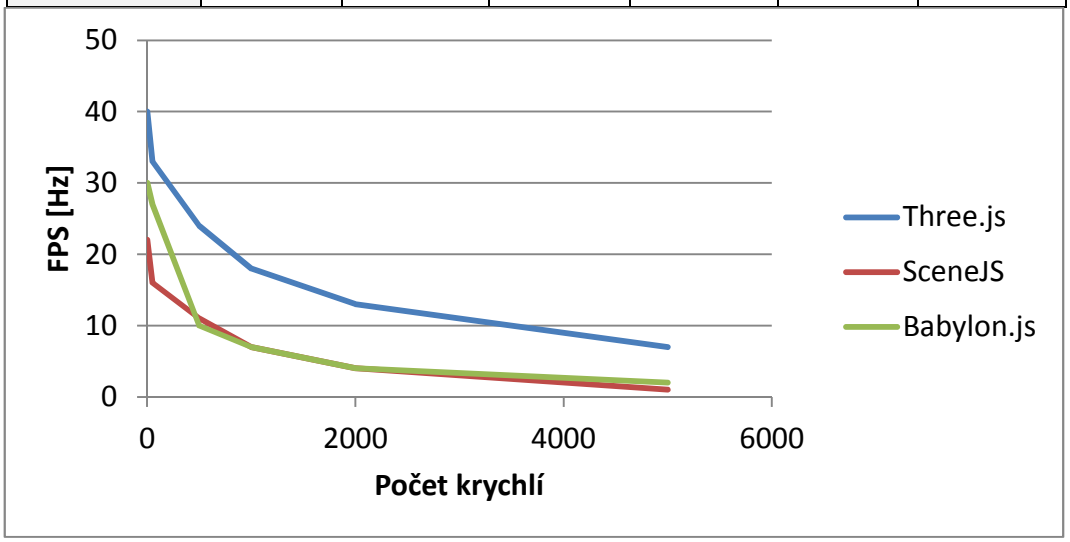
Mozilla Firefox - statická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	40	25	33	30	34	28
50	33	30	26	38	27	37
500	24	40	13	75	11	84
1000	18	52	10	96	10	103
2000	13	73	7	147	5	188
5000	8	130	3	287	3	390



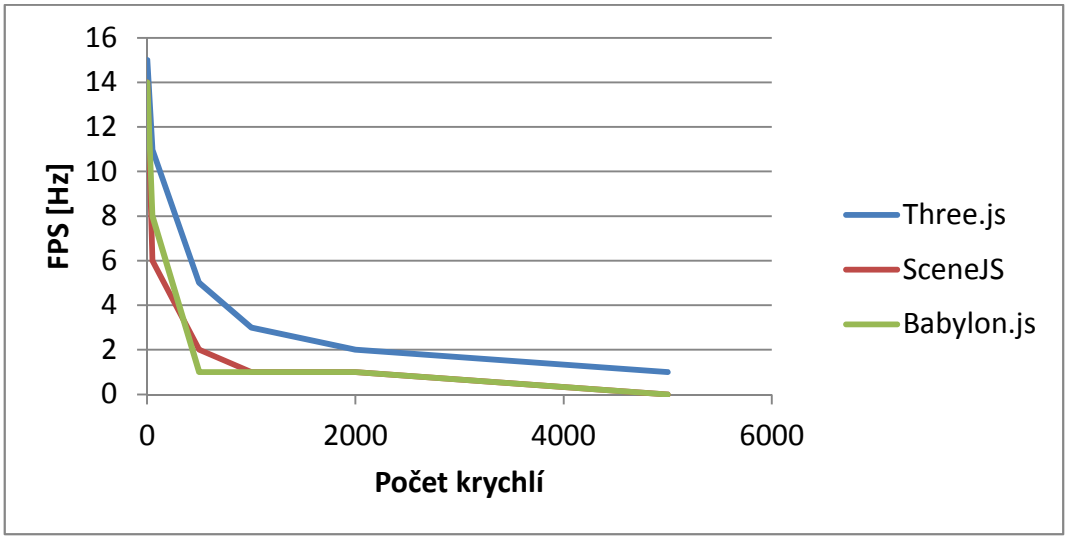
Mozilla Firefox - dynamická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	40	25	22	42	30	32
50	33	30	16	54	27	37
500	24	41	11	88	10	101
1000	18	53	7	125	7	148
2000	13	76	4	220	4	241
5000	7	136	1	832	2	513



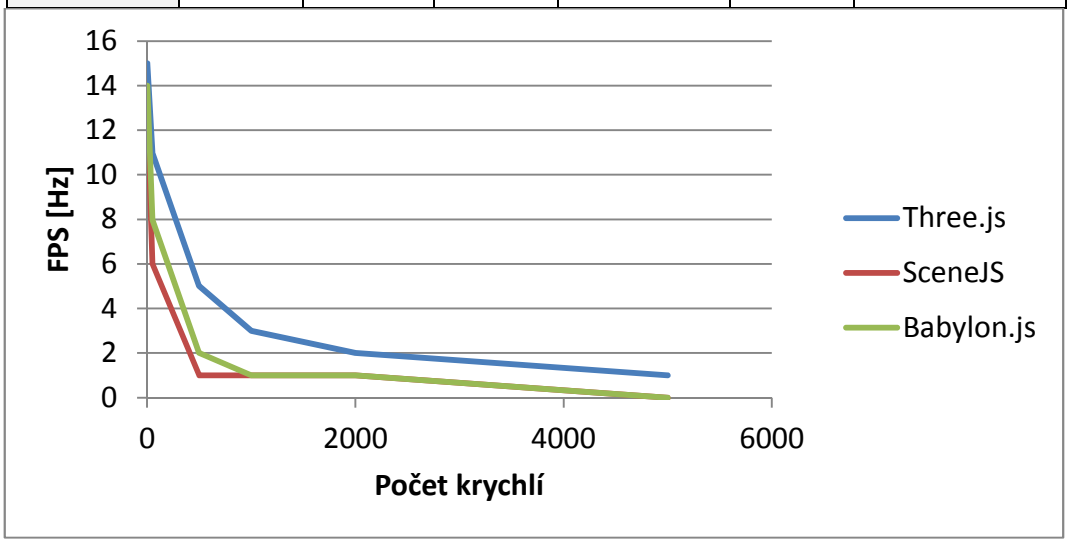
Internet Explorer - statická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	15	64	14	73	14	74
50	11	93	6	153	8	124
500	5	200	2	644	1	437
1000	3	312	1	950	1	720
2000	2	466	1	1378	1	1181
5000	1	888	0	2121	0	2987



Internet Explorer - dynamická scéna

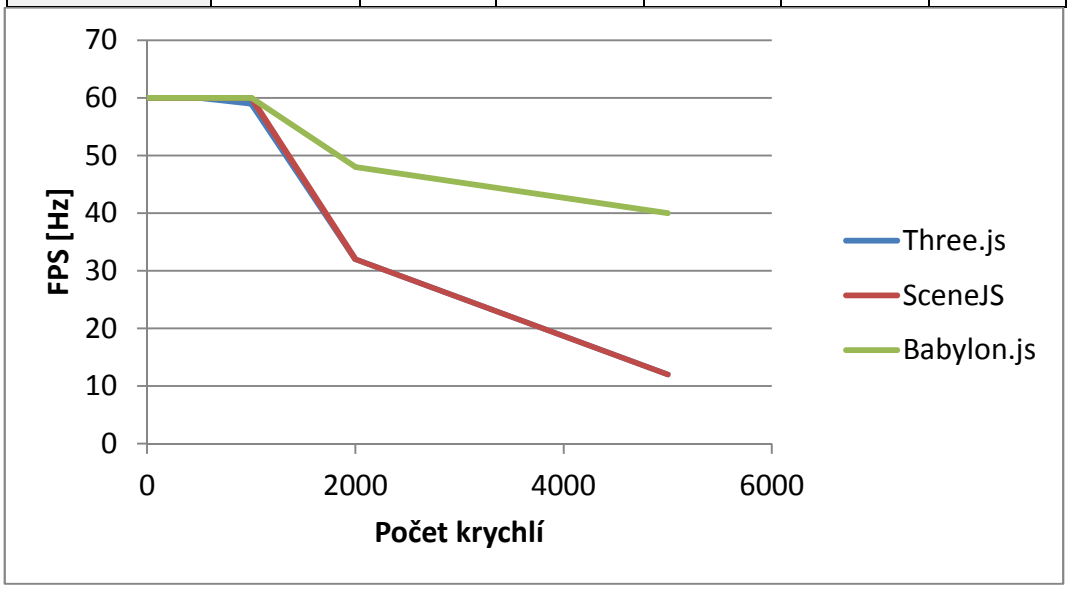
	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	15	66	13	78	14	74
50	11	93	6	161	8	121
500	5	203	1	710	2	491
1000	3	321	1	1207	1	837
2000	2	475	1	1752	1	1464
5000	1	940	0	3257	0	10241



B.2 Druhá sestava: výkonný notebook

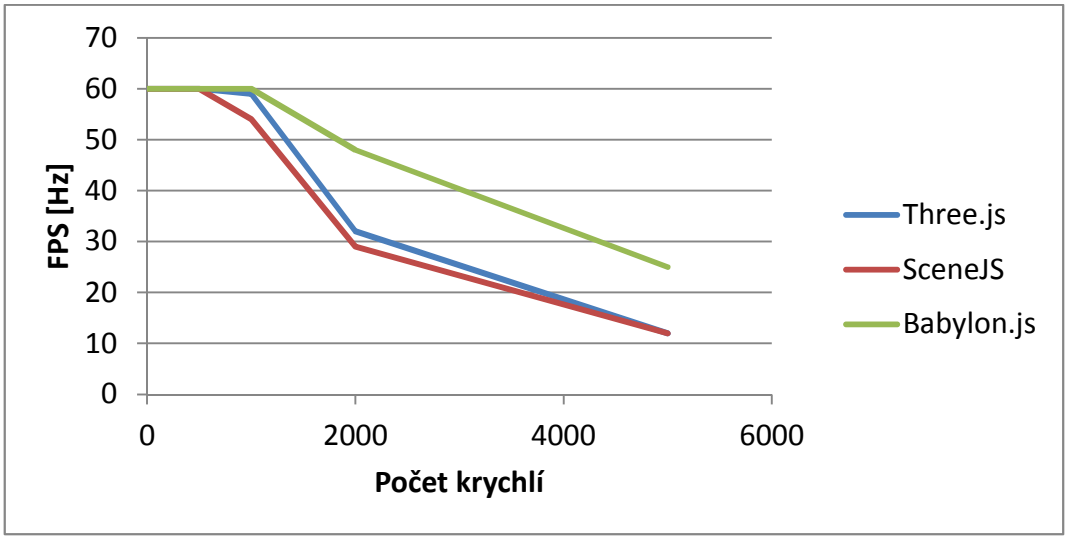
Google Chrome - statická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	60	17	60	17	60	17
50	60	17	60	17	60	17
500	60	17	60	17	60	17
1000	59	17	60	17	60	17
2000	32	30	32	31	48	17
5000	12	80	12	81	40	19



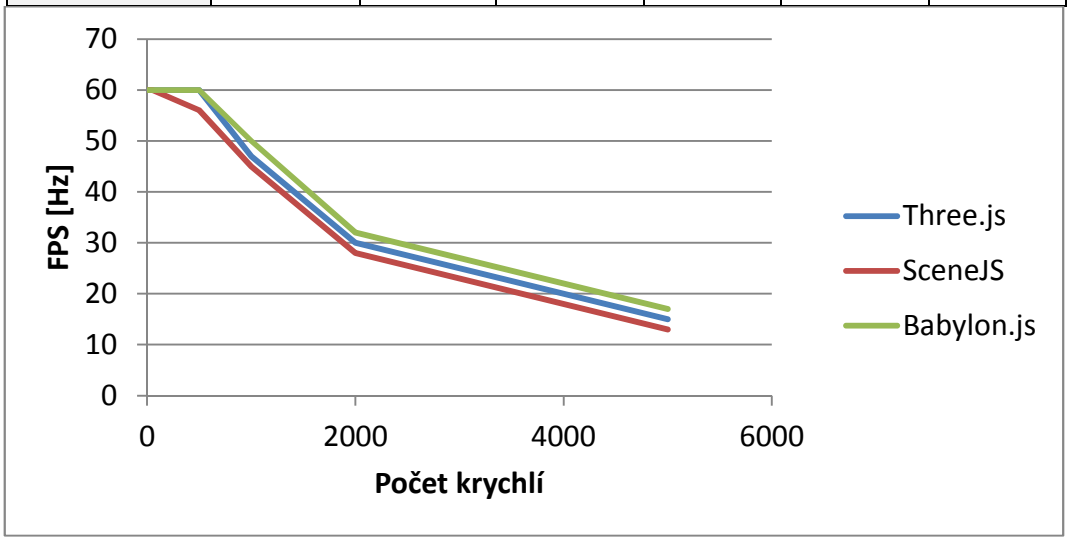
Google Chrome - dynamická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	60	17	60	17	60	17
50	60	17	60	17	60	17
500	60	17	60	17	60	17
1000	59	17	54	18	60	17
2000	32	31	29	34	48	18
5000	12	80	12	83	25	40



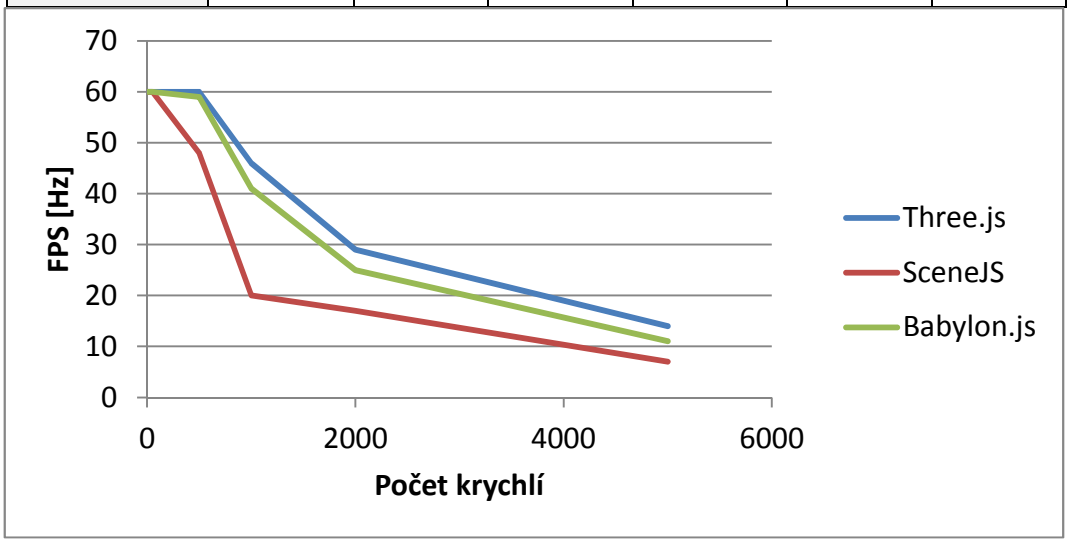
Mozilla Firefox - statická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	60	17	60	17	60	17
50	60	17	60	17	60	17
500	60	17	56	18	60	17
1000	47	21	45	22	50	19
2000	30	33	28	35	32	31
5000	15	68	13	74	17	60



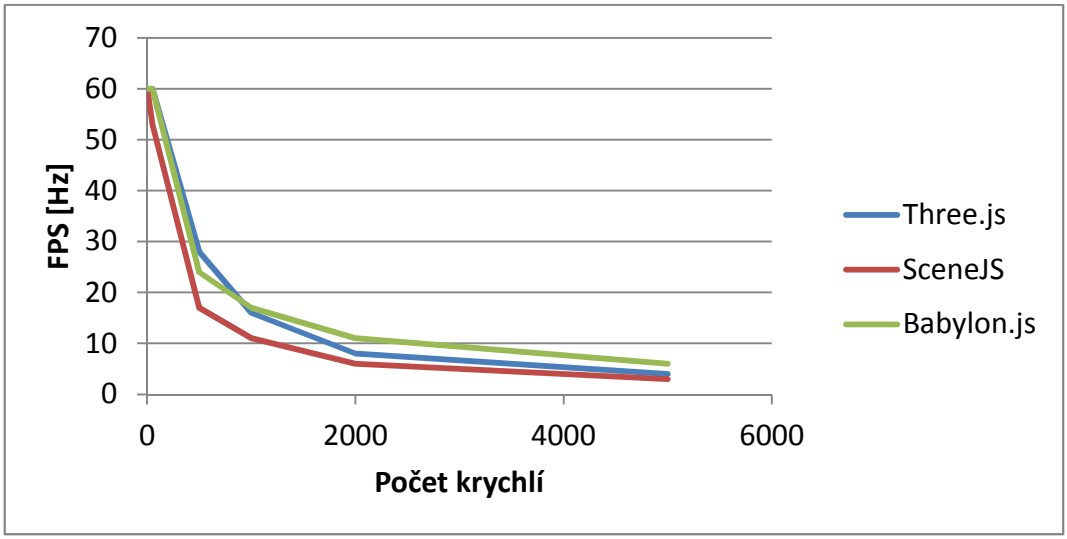
Mozilla Firefox - dynamická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	60	17	60	17	60	17
50	60	17	60	17	60	17
500	60	17	48	21	59	17
1000	46	22	20	49	41	24
2000	29	34	17	56	25	40
5000	14	71	7	141	11	88



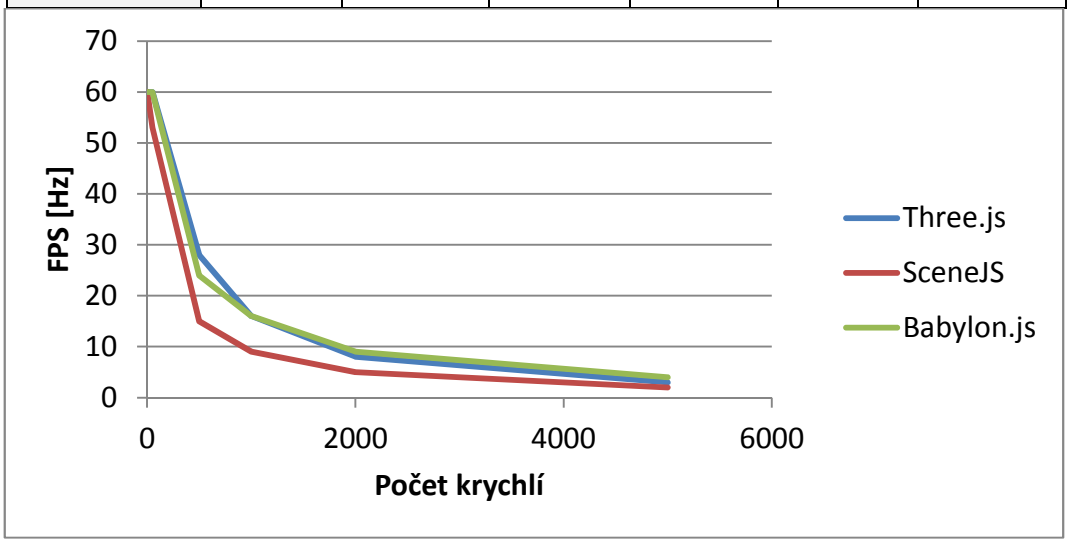
Internet Explorer - statická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	60	17	60	17	60	17
50	60	17	53	18	60	17
500	28	35	17	57	24	30
1000	16	64	11	93	17	53
2000	8	119	6	163	11	91
5000	4	281	3	365	6	167



Internet Explorer - dynamická scéna

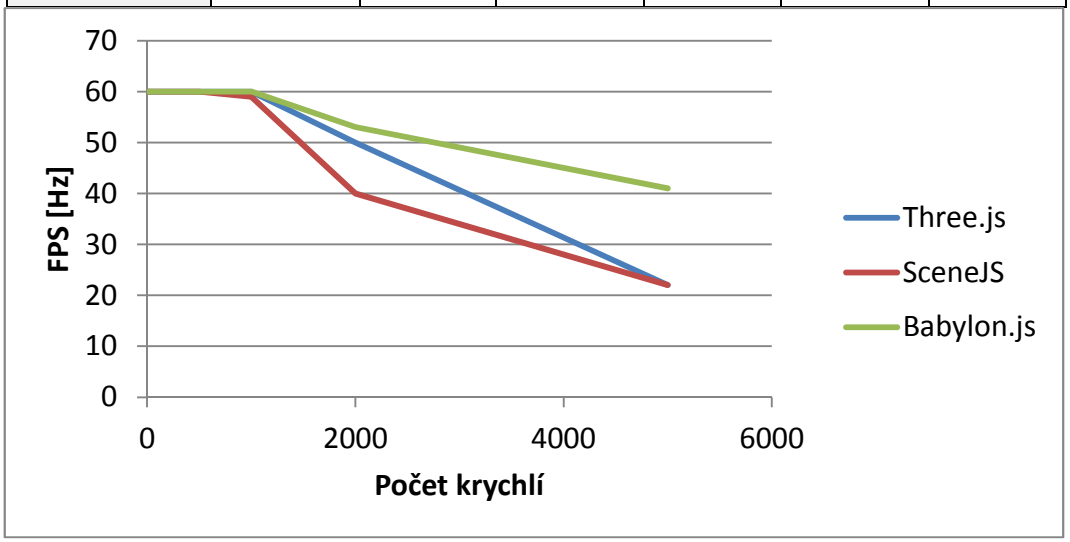
	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	60	17	60	17	60	17
50	60	17	53	18	60	17
500	28	35	15	68	24	32
1000	16	63	9	112	16	56
2000	8	120	5	200	9	104
5000	3	286	2	488	4	229



B.3 Třetí sestava: výkonný osobní počítač

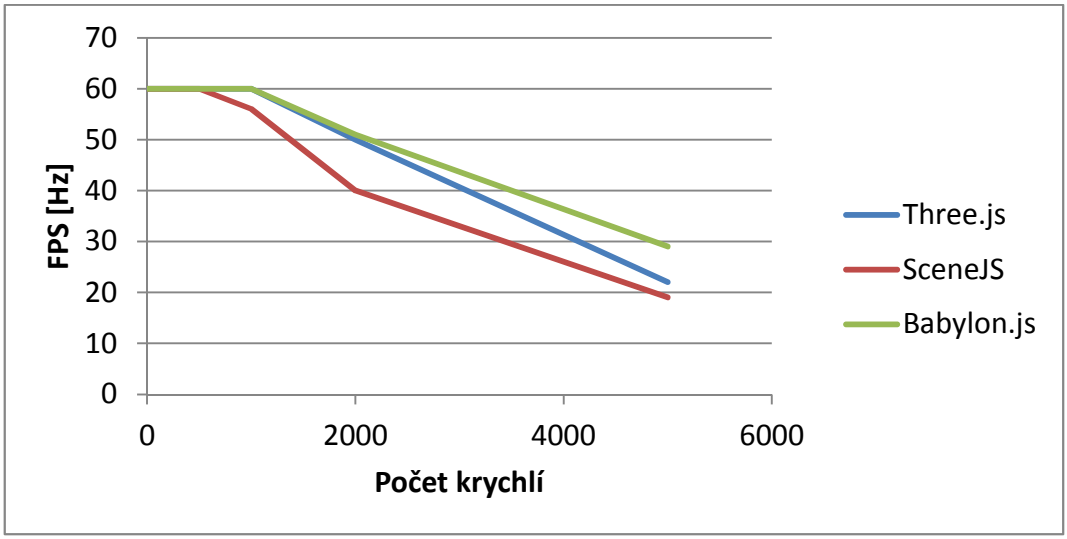
Google Chrome - statická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	60	17	60	17	60	17
50	60	17	60	17	60	17
500	60	17	60	17	60	17
1000	60	17	59	17	60	17
2000	50	19	40	23	53	17
5000	22	45	22	46	41	20



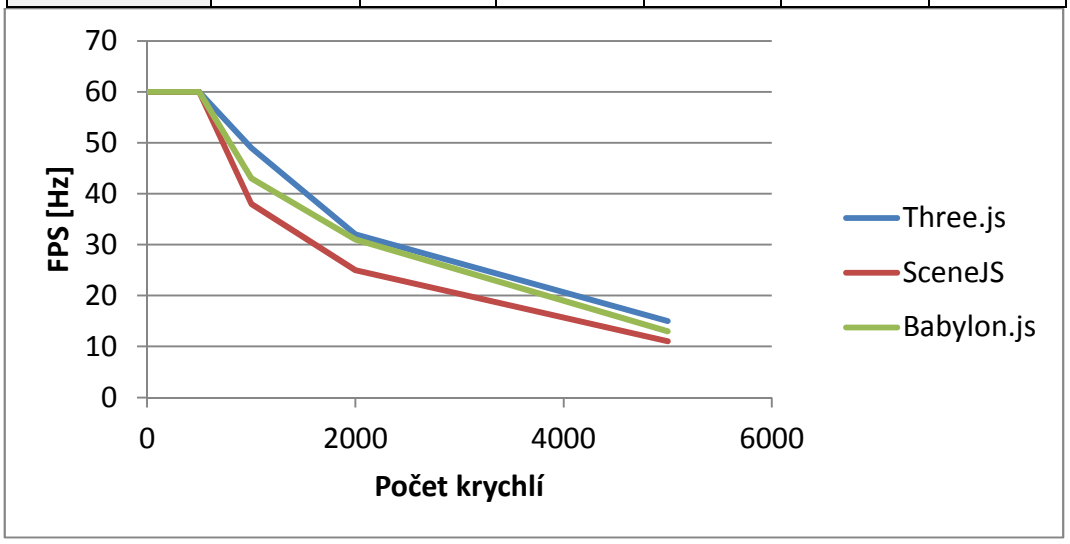
Google Chrome - dynamická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	60	17	60	17	60	17
50	60	17	60	17	60	17
500	60	17	60	17	60	17
1000	60	17	56	17	60	17
2000	50	19	40	23	51	20
5000	22	45	19	51	29	34



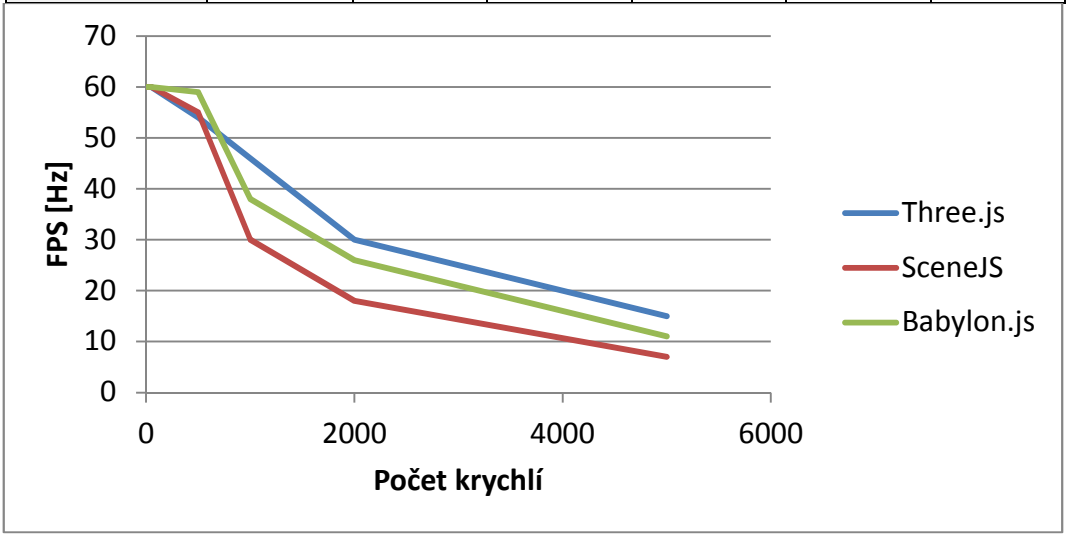
Mozilla Firefox - statická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	60	17	60	17	60	17
50	60	17	60	17	60	17
500	60	17	60	17	60	17
1000	49	20	38	24	43	22
2000	32	31	25	40	31	31
5000	15	67	11	94	13	64



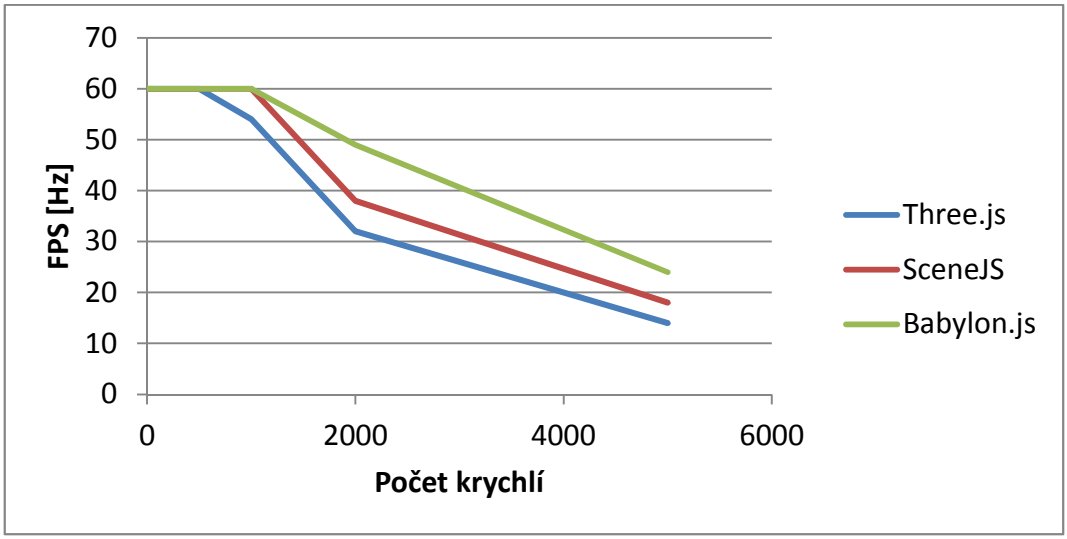
Mozilla Firefox - dynamická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	60	17	60	17	60	17
50	60	17	60	17	60	17
500	54	18	55	18	59	17
1000	46	21	30	32	38	26
2000	30	33	18	55	26	37
5000	15	67	7	135	11	85



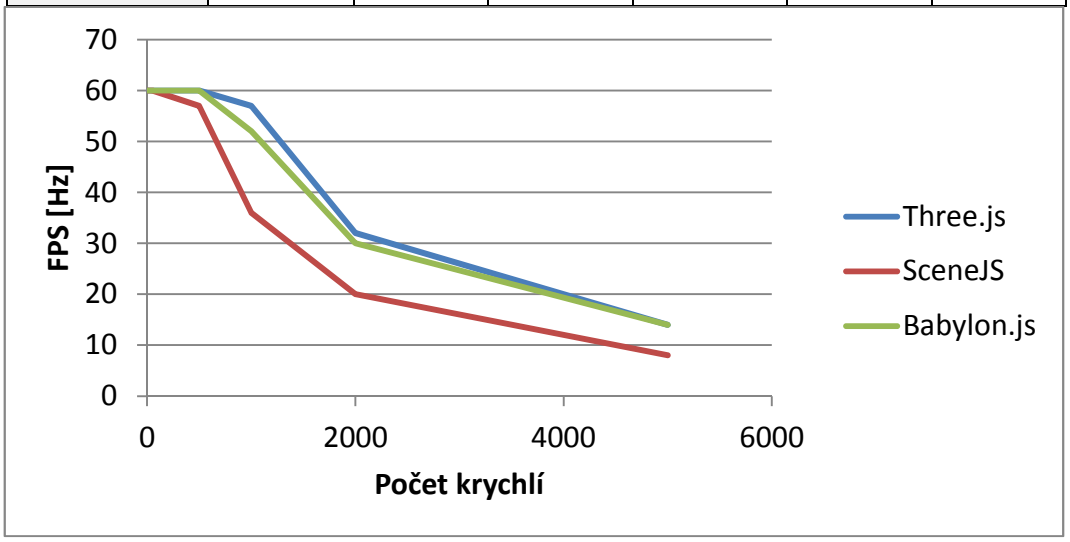
Internet Explorer - statická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	60	17	60	17	60	17
50	60	17	60	17	60	17
500	60	17	60	17	60	17
1000	54	18	60	17	60	17
2000	32	30	38	26	49	20
5000	14	69	18	54	24	41



Internet Explorer - dynamická scéna

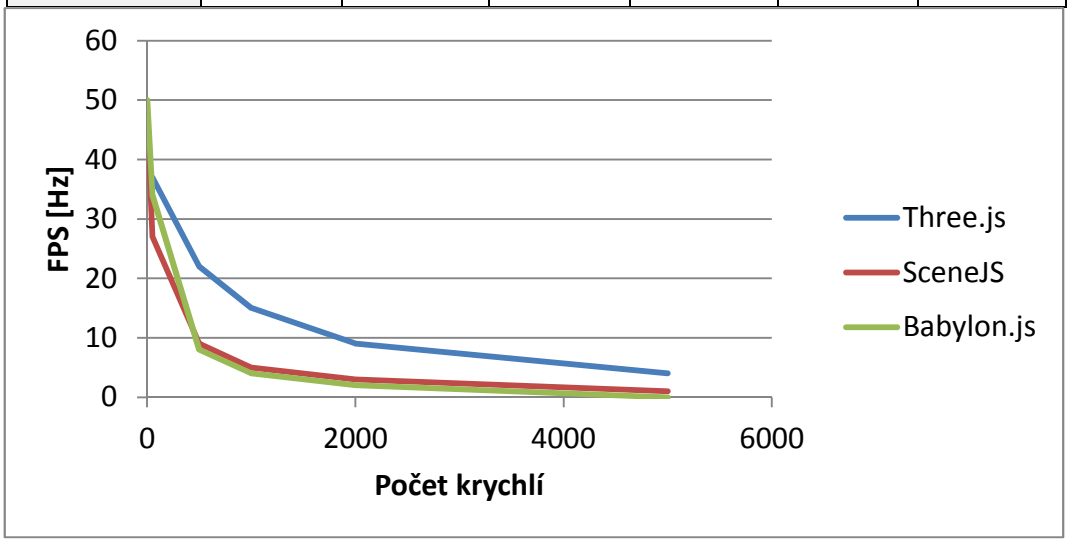
	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	60	17	60	17	60	17
50	60	17	60	17	60	17
500	60	17	57	17	60	17
1000	57	17	36	27	52	19
2000	32	31	20	48	30	31
5000	14	69	8	120	14	68



B.4 Čtvrtá sestava: mobilní zařízení

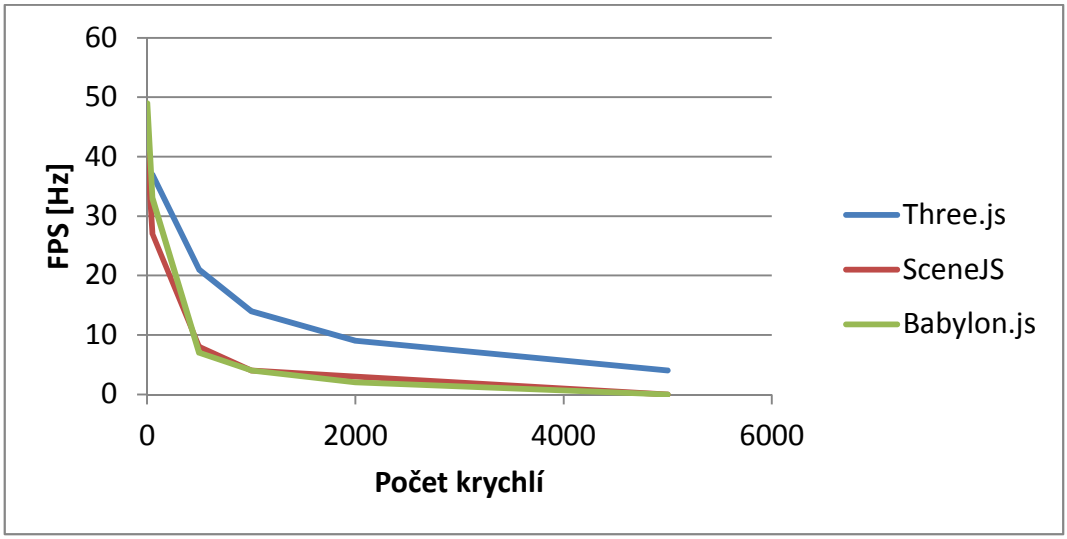
Google Chrome pro Android - statická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	37	23	47	20	50	19
50	37	27	27	34	34	30
500	22	46	9	117	8	121
1000	15	65	5	201	4	223
2000	9	112	3	384	2	449
5000	4	237	1	850	-	-



Google Chrome pro Android - dynamická scéna

	Three.js		SceneJS		Babylon.js	
	FPS	MS	FPS	MS	FPS	MS
1	34	29	41	20	49	19
50	37	27	27	33	33	30
500	21	46	8	122	7	134
1000	14	71	4	286	4	250
2000	9	115	3	382	2	507
5000	4	254	-	-	-	-


60
50
40
30
20
10
0
FPS [Hz]
0
2000
4000
6000
Počet krychlí
Three.js
SceneJS
Babylon.js

C Multikriteriální analýza

Three.js

Škála: 1 - 10

Kritéria	Váhy kritérií	Podkritéria	Hodnota	Hodnocení	Celkem
Nástroje	10	geometrie - hodnocení	10	10	**100**
		materiály - hodnocení	10		
		světla - hodnocení	10		
		kamery - hodnocení	10		
		import objektů - hodnocení	10		
		editor	ano		
Snímková frekvence	10	počet testů s nejvyšší hodnotou FPS	12	10	**100**
Licence, přístupnost zdrojových kódů	9	svobodná licence	ano, MIT	10	**90**
		otevřený zdrojový kód	ano		
Dokumentace, příklady	9	dokumentace, tutoriály - hodnocení	9	9	**81**
		počet příkladů	259		
Aktuálnost, vývojáři	5	poslední aktualizace	16. 3. 2015	10	**50**
		počet aktualizací za poslední měsíc (březen 2015)	108		
		počet vývojářů	442		
Komunita	6	oblíbenost na serveru GitHub	19 135	10	**60**
		otázky na serveru StackOverflow	12 474		
					481

Babylon.js

Škála: 1 - 10

Kritéria	Váhy kritérií	Podkritéria	Hodnota	Hodnocení	Celkem
Nástroje	10	geometrie - hodnocení	9	9	**90**
		materiály - hodnocení	10		
		světla - hodnocení	8		
		kamery - hodnocení	10		
		import objektů - hodnocení	9		
		editor	ano		
Snímková frekvence	10	počet testů s nejvyšší hodnotou FPS	8	7	**70**
Licence, přístupnost zdrojových kódů	9	svobodná licence	ano, Apache	9	**81**
		otevřený zdrojový kód	ano		
Dokumentace, příklady	9	dokumentace, tutoriály - hodnocení	8	6	**54**
		počet příkladů	25		
Aktuálnost, vývojáři	5	poslední aktualizace	16. 4. 2015	9	**45**
		počet aktualizací za poslední měsíc (březen 2015)	115		
		počet vývojářů	44		
Komunita	6	oblíbenost na serveru GitHub	2 024	5	**30**
		otázky na StackOverflow	87		
					370

SceneJS

Škála: 1 - 10

Kritéria	Váhy kritérií	Podkritéria	Hodnota	Hodnocení	Celkem
Nástroje	10	geometrie - hodnocení	8	6	**60**
		materiály - hodnocení	8		
		světla - hodnocení	6		
		kamery - hodnocení	5		
		import objektů - hodnocení	5		
		editor	ne		
Snímková frekvence	10	počet testů s nejvyšší hodnotou FPS	0	0	**0**
Licence, přístupnost zdrojových kódů	9	svobodná licence	ano, MIT + GNU GPL	10	**90**
		otevřený zdrojový kód	ano		
Dokumentace, příklady	9	dokumentace, tutoriály - hodnocení	3	5	**45**
		počet příkladů	217		
Aktuálnost, vývojáři	5	poslední aktualizace	15. 4. 2015	4	**20**
		počet aktualizací za poslední měsíc (březen 2015)	8		
		počet vývojářů	12		
Komunita	6	oblíbenost na serveru GitHub	370	3	**18**
		otázky na StackOverflow	52		
					233

D Obsah kompaktního disku

K práci je přiložen kompaktní disk, který obsahuje web s praktickými příklady nástrojů knihoven a s jednoduchou aplikací pro měření snímkové frekvence. Na disku se také nachází zdrojové kódy k těmto příkladům.

FIM UHK
UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Rokitanského 62, 500 03 Hradec Králové, tel: 493 331 111, fax: 493 332 235

Zadání k závěrečné práci

Jméno a příjmení studenta: **Markéta Šťastná**

Obor studia: Aplikovaná informatika
Jméno a příjmení vedoucího práce: **Filip Malý**

Název práce:
Srovnání knihoven jazyka JavaScript pro práci s 3D grafikou

Název práce v AJ:
The comparison of JavaScript language frameworks for working with 3D graphics

Podtitul práce:

Podtitul práce v AJ:

Cíl práce: Cílem práce je přehled a srovnání knihoven pro práci s 3D grafikou postavených na WebGL, a to z několika aspektů a na základě praktického příkladu.

Osnova práce:
- Úvod
- Představení knihovny WebGL
- Přehled knihoven postavených na WebGL pro práci s 3D grafikou
- Porovnání knihoven na základě daných aspektů
- Vytvoření praktického příkladu ve vybraných knihovnách
- Výběr nejvíce vyhovující knihovny a shrnutí výsledků
- Závěr a literatura

Projednáno dne: 13. 10. 2014

Podpis studenta Šťastná

Podpis vedoucího práce Malý