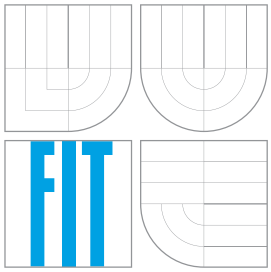
VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

ZOBRAZENÍ 3D SCÉNY VE WEBOVÉM PROHLÍŽEČI

DISPLAYING 3D GRAPHICS IN WEB BROWSER

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

JIŘÍ ZVĚŘINA

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. MICHAL ŠPANĚL, Ph.D.

BRNO 2016

Abstrakt

Bakalářská práce se zabývá návrhem a implementací aplikace zobrazující diagnostické informace o stavu automobilu s využitím 3D vizualizace. Aplikace umožňuje v reálném čase zobrazovat varování související s poruchami dílů automobilu, jako jsou například vady motoru, či nízký tlak v pneumatikách. Aplikace je postavena na technologii WebGL, knihovně Three.js a běží v prostředí internetového prohlížeče.

Abstract

This Bachelor's thesis deals with design and implementation of application displaying car's diagnostic information using 3D visualization. Application allows real-time displaying of warnings related to car parts malfunctions, such as engine failure, or low tire pressure. Aplication is based on WebGL technology, Three.js library and runs in a web browser.

Klíčová slova

WebGL, Javascript, 3D, uživatelské rozhraní, webová aplikace, diagnostika automobilu, shader

Keywords

WebGL, Javascript, 3D, user interface, web application, car diagnostics, shader

Citace

ZVĚŘINA, Jiří. *Zobrazení 3D scény ve webovém prohlížeči.* Brno, 2016. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Španěl Michal.

Zobrazení 3D scény ve webovém prohlížeči

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Michala Španěla, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Zvěřina
17. května 2016

Poděkování

Děkuji vedoucímu mé práce Ing. Michalu Španělovi, Ph.D., za veškeré rady, připomínky a odbornou pomoc při tvorbě této bakalářské práce. Dále také Prof. Dr. Ing. Pavlu Zemčíkovi za podnět zahrnutí automobilů do náplně mé práce.

Obsah

Kapitola 1

Úvod

Zobrazování 3D grafiky v moderních webových prohlížečích začíná být běžným prostředkem sdělování infomací. Využívá se k tomu Javascriptové API WebGL založené na OpenGL ES 2.0. Využívání této technologie s sebou nese spousty výhod, ale také překážky, se kterými je třeba se při práci s WebGL vypořádat. Využití výhod a překonání těchto překážek je náplní této práce.

Hlavním cílem této práce je demonstrovat možnost zobrazení 3D scény ve webovém prohlížeči, dále také ukázat, jaké jsou možnosti pro zobrazení modelu automobilu a jeho diagnostiky. Tyto možnosti jsou v této práci demonstrovány jako tvorba a návrh jednoduchého uživatelského rozhraní. Cílem bylo také vyzkoušet, jakým způsobem se dají využívat některé grafické shadery ve webových aplikacích. Ke zjednodušení práce s WebGL je využita Javascriptová 3D knihovna Three.js. Práce se zmiňuje o technikách využívaných při vytváření scén, animací a tvorby grafického rozhraní ve 3D s využitím této knihovny.

Cílová aplikace umožňuje v reálném čase zobrazovat varování o chybách vzniklých v součástkách automobilu. Základem uživatelského rozhraní je 3D model automobilu se všemi prvky, které mohou být signálem z automobilu označeny jako chybné. Mezi prvky zahrnuté v této aplikaci patří například motor, pneumatiky, popřípadě dveře automobilu. Tyto informace jsou uživateli předávány skrze informační text a zvýraznění konkrétních částí 3D modelu. Uživatel také může s automobilem volně otáčet.

Ve druhé kapitole této práce bude čtenář obeznámen s teorií vykreslování 3D scén ve webových prohlížečích, technologií WebGL, knihovnou Three.js a typy modelů, které jsou v této knihovně používány. Ve třetí kapitole budou zmíněny metody vylepšení vzhledu využívané v 3D aplikacích. Současný stav zobrazování diagnostických informací na palubních počítačích v automobilech bude předmětem čtvrté kapitoly. Pátá kapitola se zabývá návrhem cílové aplikace. Implementace a použité postupy budou objasněny v kapitole šesté. Sedmá kapitola pojednává o testování výsledné aplikace a výsledcích, které toto testování přineslo. Poslední, osmá kapitola, bude shrnovat získané poznatky a výsledky tvorby aplikace a pojednávat o možnostech budoucích úprav.

Kapitola 2

Vykreslování 3D scény ve webovém prohlížeči

Vykreslování 3D grafiky v prohlížečích je výhodné hned z několika důvodů. Prvním z těchto důvodů je možnost zobrazit informace, které běžnými způsoby zobrazit nelze, například model pro 3D tisk s libovolnou možností otáčení nebo místnost s nábytkem, po které se dá volně přesunovat. Dalším důvodem je nezávislost na platformě. Omezujícím faktorem je v tomto ohledu pouze webový prohlížeč. V případě webových aplikací je také velkou výhodou snadná údržba a jednoduchá aktualizace.

2.1 Javascript a WebGL

Jak již bylo zmíněno v úvodu, WebGL je API založené na OpenGL ES 2.0. Toto API rozšiřuje možnosti programovacího jazyka Javascript. Jako místo k vykreslování je využit HTML5 element canvas neboli plátno. O vykreslování se stará přímo grafický procesor, což zaručuje vysokou rychlost. Grafický procesor lze využít i pro rychlé počítání složitých výpočtů.

Javascript je objektově orientovaný skriptovací jazyk navržený pro tvorbu webových aplikací. Kód aplikací psaných v Javascriptu je stahován ze serveru a vykonáván na straně klienta. Výhoda tohoto přístupu je především možnost aplikace okamžitě reagovat na uživatelovi akce bez potřeby přenášet data na server a zpět. Javascript umožňuje vytvářet dynamičtější webové stránky, proto také jeho obliba v posledních letech stále roste a čím dál více aplikací je přesouváno do webového prostředí.

OpenGL ES 2.0, kde ES znamená Embedded Systems (pro vestavěné systémy), je zjednodušenou verzí klasického OpenGL 2.0. Tato verze je navržena k použití na konzolích, mobilních telefonech, popřípadě na palubních počítačích automobilů. Tato verze oproti běžnému OpenGL 2.0 využívá k vykreslování pouze shadery a neobsahuje fixní funkce pro transformace a fragment pipeline.Tímto je docíleno snížení nároků na spotřebu energie, což je u vestavěných systému velmi důležité [5]. Podobného principu si můžeme povšimnout u přechodu z OpenGL 3.0 na OpenGL 3.1.

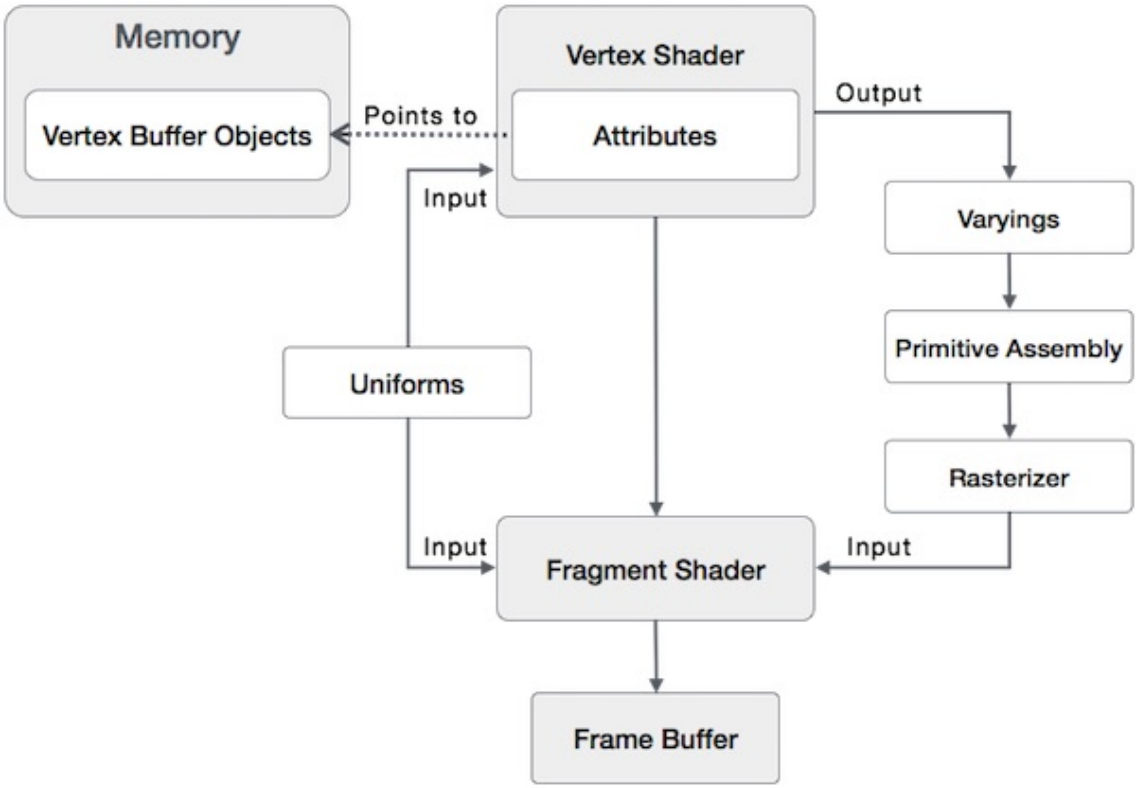
WebGL je již velmi dobře rozšířeno a podporují ho všechny významné webové prohlížeče znázorněné na obrázku 2.1. Velikou výhodou WebGL je, že není třeba instalovat žádné dodatečné knihovny ani doplňky prohlížeče ke správnému chodu 3D aplikací.

V současné době je WebGL využíváno například společností Google a její aplikací Go-

Obrázek 2.1: Prohlížeče, které podporují technologii WebGL.

ogle Maps, popřípadě různými Chrome Experiments[1]. Další uplatnění nachází ve hrách, v aplikacích demonstrující počítačovou grafiku, a dokonce ve výukových systémech.

Jelikož je WebGL založeno na OpenGL ES 2.0, využívá také jeho programovatelnou pipeline představenou na obrázku 2.2. Pipeline je tvořen několika po sobě jdoucími stádii, kterými musí grafický procesor projít při renderování scény. Rychlost pipeline je určena jejím nejpomalejším stádiem [1].



Obrázek 2.2: Grafická pipeline WebGL. Převzato z [9].

GLSL ES

K implementaci shaderů viditelných v grafické pipeline OpenGL ES 2.0 na obrázku 2.2 je nutné použít jazyk GLSL ES neboli OpenGL ES Shading Language. Jedná se ve skutečnosti o dva blízce provázané jazyky velmi podobné jazyku C++. Jelikož se všechny popisované

[1] https://www.chromeexperiments.com/webgl

vlastnosti týkají obou těchto jazyků, odkazuje se na ně jako na jeden ucelený jazyk. Oproti jazyku C++ zde není možná práce s ukazateli. Naopak zde přibývají možnosti využití vektorových typů a operací, bez kterých by byla implementace grafických funkcí výrazně složitější [8].

Shadery v jazyce GLSL ES jsou psány jako krátké programy většinou navržené k vykreslování jednoho materiálu, nebo efektu vykreslovaného přes celou obrazovku.

2.2 Three.js

Protože je API WebGL nízkoúrovňové, vzniklo několik knihoven, které práci s ním značně usnadňují. Jednou z těchto knihoven je Three.js. Tato knihovna je volně dostupná a vytváří jednoduché rozhraní pro práci s WebGL.

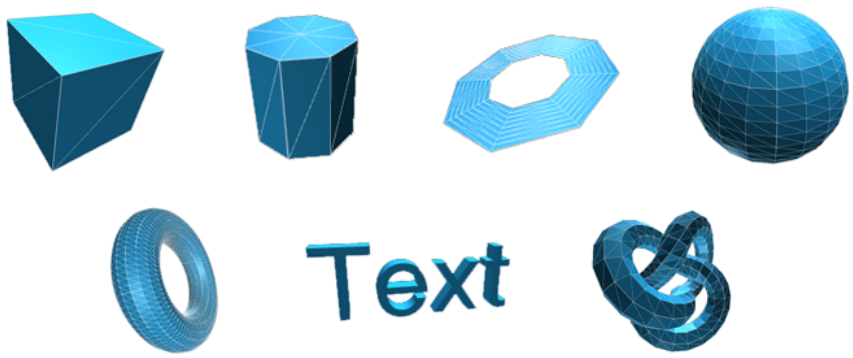
Možnosti knihovny

Three.js usnadňuje spoustu operací při práci s 3D grafikou. Patří mezi ně:

- Tvorba **scény**, práce s ní a správa objektů v ní.
- Vytváření a manipulace s **kamerami** a **světly**.
- Přidávání **modelů** a editace jejich geometrií, materiálů a závislostí.
- Generování základních **animací**.
- Načítání 3D modelů pomocí **loaderů** o kterých bude řeč později.

Kromě základního použití s WebGL lze generovat 3D grafiku i pomocí CSS. Tento postup je možné zvolit v případě, kdy webový prohlížeč nepodporuje WebGL a i tak je potřeba, aby se všem uživatelům zobrazovala stejná stránka. Tento přístup je oproti vykreslování s pomocí WebGL mnohem pomalejší.

Knihovna Three.js kromě vkládání vytvořených modelů umožňuje také vytvářet některé přednastavené základní objekty. Skládáním těchto jednoduchých objektů se dá vytvořit komplexní scéna. Na obrázku 2.3 je vidět většina těchto objektů.



Obrázek 2.3: Základní objekty poskytované knihovnou Three.js.

Formát modelů

K přenosu modelů a scén mezi modelovacím programem a výslednou aplikací existuje několik různých formátů. Knihovna Three.js některé z těchto formátů umí využívat. Nejpoužívanější formáty jsou níže popsány.

Nejuniverzálnějším formátem pro přenos modelů je formát **COLLADA**, který je momentálně spravován organizací Khronos Group[2]. Tato organizace má také na svědomí již několikrát zmiňované OpenGL, a tedy i WebGL samotné. Tento formát je založen na XML a je možné v něm definovat celé scény včetně kamer a světel [4]. Jedinou nevýhodou tohoto formátu je neaktuálnost loaderu v knihovně Three.js od verze r73. Three.js prošlo ve verzi r73 změnou systému animací, která není aplikována na COLLADA loader, tudíž jsou animace nefunkční.

Naopak nejzákladnějším formátem, respektive dvojicí formátů, jsou **OBJ** a **MTL**. Tyto formáty bývají většinou využity společně. Jeden z nich definuje geometrie a druhý materiály. Oba jsou textově založené [4]. Hlavní oblastí využití těchto formátů je pouze jednoduchý přenos modelů a materiálů.

Nejvýhodnějším formátem pro přenos funkčních animací je momentálně formát **JSON**, pro který autoři Three.js vytvořili exportér[3] z modelovacího programu Blender[4]. Tento formát je navržen přímo pro potřeby Three.js. Pomocí formátu JSON je možné přenášet geometrie i materiály.

Existuje široká škála dalších formátů, pro které jsou v knihovně Three.js napsány loadery. Tyto formáty ale nejsou tak často používány jako formáty dříve zmíněné. Patří mezi ně například formát **MD2** známý především svým využitím ve známé počítačové hře Quake, kde sloužil jako formát pro modely postav. Za zmínku stojí také formát **STL** využívaný pro technické modely exportované nástroji CAD.

2.3 Možnosti použití shaderů ve WebGL

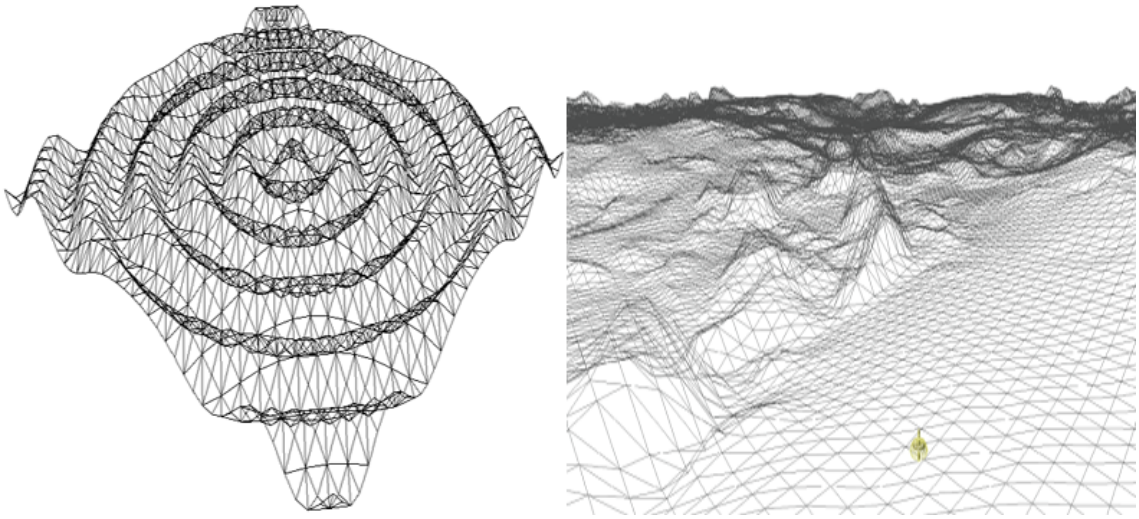
Shader je program umožňující řídit určitá stádia programovatelné pipeline. Ve WebGL jsou používány 2 shadery. Jsou to vertex shader a fragment shader. Tyto shadery jsou schopny pracovat s proměnnými z aplikace, kterým se říká uniforms. Každý z těchto shaderů má jedinečnou funkci.

Vertex shader je program prováděný nad každým vertexem dané scény. Provádí základní transformace pro dosažení požadovaného zobrazení objektů scény. Tento shader má kromě základní transformace také spoustu dalších využití. Jedním z nich je rozvlnění rovné plochy představující vodní hladinu jako na obrázku 2.4.

Fragment shader je program prováděný nad každým pixelem rasterizované scény. Provádí barvení objektů pomocí definovaných materiálů, popřípadě aplikuje textury. Použitelnou variantou je například takzvaný toon shader, který vytváří u objektů kreslený vzhled jako na obrázku 2.5.

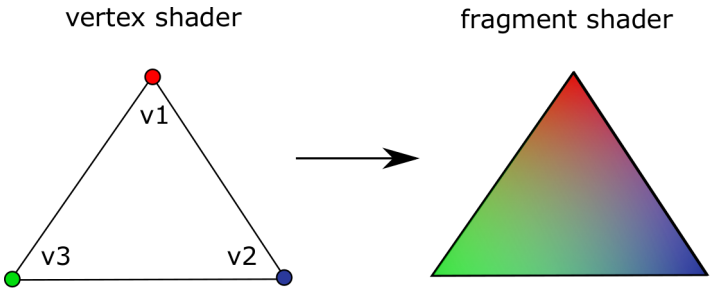
Kromě základních informací předávaných z vertex shaderu do fragment shaderu je možné předávat i proměnné. Používá se k tomu typ proměnných označený jako varying. Jelikož vertex shader je prováděn nad vertexy a fragment shader nad pixely, je proměnná typu varying interpolována. Interpolaci vysvětluje obrázek 2.6 na příkladu využívajícím obarvení vertexů.

[2] https://www.khronos.org/
[3] https://github.com/mrdoob/three.js/tree/master/utils/exporters/blender
[4] https://www.blender.org/



Obrázek 2.4: Možné aplikace vertex shaderu. Zvlnění rovné plochy do podoby vodní hladiny (vlevo) a tvorba hornatého prostředí (vpravo). Zdroj: http://mmmovania.blogspot.cz/2013_05_01_archive.html

Obrázek 2.5: Aplikace fragment shaderu tvořící kreslený vzhled. Zdroj: http://www.lighthouse3d.com/tutorials/glsl-12-tutorial/shader-examples/



Obrázek 2.6: Interpolace při předávání barvy vertexů mezi vertex a fragment shaderem.

Kapitola 3

Grafické efekty pro vylepšení vzhledu aplikace

Při tvorbě aplikace, která vytváří pro uživatele jakékoliv uživatelské rozhraní, je možné zvažovat i metody pro vylepšení vzhledu cílené aplikace. Tyto metody se liší dle typu rozhraní o který se jedná. Tyto metody mohou být jak funkčního charakteru, jakým je třeba zvýraznění objektů, tak čistě estetického jako třeba volba materiálu, popřípadě osvětlení. Tyto metody se týkají uživatelského rozhraní využívajícího 3D grafiku.

3.1 Vizuální efekty pomocí shaderů

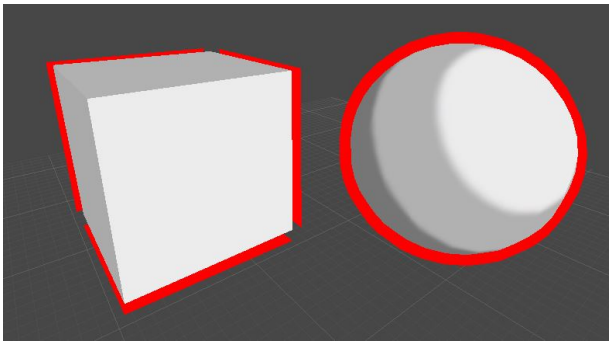
Nejčastěji používanou metodou pro vylepšení vzhledu grafického programu je aplikace shaderů. V této sekci bude popsána teorie potřebná k pochopení shaderů využívaných v této aplikaci. Jedná se o shader pro vytvoření barevného zvýraznění objektu a také o shader pro vytvoření realističtější barvy vozidla.

Hlavní část zvýraznění obrysů součástek je obsažena ve vertex shaderu. Tento shader funguje na principu posouvání vertexů směrem určeným jejich normálami. Jedná se tedy o jednoduchou transformaci. Pro shader je v této části důležitá pouze vzdálenost, o kterou se mají vertexy posunout, tedy jak velký bude výsledný obrys. Druhou částí zvýraznění, obsaženou ve fragment shaderu, je vybarvení tohoto zvětšeného objektu. Shader opět může přijímat informace o tom jakou barvu má využít pro vybarvení. Tento shader je vhodný převážně pro zaoblené objekty u kterých nejsou žádné ostré hrany. Tyto hrany mohou způsobovat v shaderu problémy jako na obrázku 3.1.

Pro vytvoření barvy vozidla je možné použít kruhové mapování. Toto mapování je provedeno spoluprácí vertex a fragment shaderu předávajících si informaci o odrazu světla promítnutého na jednoduchou texturu. Toto počítání je prováděno ve vertex shaderu, tento přístup využívá grafické karty k výpočtu interpolace [7]. Při nedokonalém vykreslování hran některých objektů je možné použít také per-pixel variantu, která počítá odraz pro každý pixel ve fragment shaderu. Ke správnému výpočtu jsou potřeba dva vektory. První je vektor `e` (eye), který jde směrem od kamery k fragmentu. Druhý je normálový vektor plochy, na kterou dopadá náš pohled, dále jako `n` (normal). Odražený vektor `r` lze získat tímto výpočtem:

$$r = e - 2\langle n, e\rangle n \tag{3.1}$$

Výpočet souřadnic bodu textury zajišťují tyto rovnice využívající zmíněný vektor `r`:



Obrázek 3.1: Porovnání zvýrazňování nezaoblených a zaoblených předmětů a demonstrace chyby. Nezaoblený objekt s chybným vykreslením rohů (vlevo), zaoblený objekt (vzpravo). Zdroj: http://answers.unity3d.com/questions/625968/unitys-outline-shader-sharp-edges.html

$$x = \frac{r_x}{2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}} + \frac{1}{2} \tag{3.2}$$

$$y = \frac{r_y}{2\sqrt{r_x^2 + r_y^2 + (r_z + 1)^2}} + \frac{1}{2} \tag{3.3}$$

Tyto souřadnice můžou být předány fragment shaderu, který zvolí barvu podle umístění v textuře. Tímto způsobem je možné vytvořit různé barevné provedení. Popis využívání shaderů v knihovně Three.js se nachází v kapitole o implementaci.

3.2 Světla

Důležitou součástí 3D grafické vizualizace je vhodné použití světel. V současné době existuje několik standartních typů světel, které jsou využívány ke zlepšení vzhledu aplikací a obecné viditelnosti objektů. Mezi tyto typy patří:

- bodová světla
- ambientní světla
- směrová světla
- hemisférická světla

Nejlepší efekt většinou vytváří kombinace těchto světel [2]. Existuje mnoho využitelných přístupů k osvětlení. Pokud je potřeba vytvořit spíše přirozené osvětlení, využívá se hemisférické světlo s lehkým nádechem žluté a červené (v závislosti na denní době), doplněné o ambientní osvětlené zjemňující stíny vytvořené hemisférickým světlem. Tento přístup bychom mohli volit pro prezentaci automobilu ve venkovním prostředí, například při jízdě po silnici. Další způsob – který je zvolen pro tuto aplikaci – je použití technického osvětlení. Tohoto efektu docílíme využitím bílého směrového osvětlení posazeného přímo nad automobil a opět slabým ambientním osvětlením zjemňujícím stíny. Toto osvětlení vytváří stíny

pouze pod automobilem a dodává prezentaci vozidla výstavní dojem. Jedná se o dobře viditelné, esteticky přijatelné zobrazení automobilu. Osvětlování modelu zepředu, popřípadě zezadu, pro zvýraznění rysů automobilu v této aplikaci nemá příliš význam.

Kapitola 4

Současný stav využívání palubních počítačů

Palubní počítače, s displejem, či bez, se dnes nachází téměř ve všech v současnosti prodávaných vozech se standartní výbavou. Tyto počítače mají široké spektrum využití. Kromě měření počtu najetých kilometrů a okolní teploty zobrazují také informace o stavu jednotlivých součástí vozidla. Technologie zobrazování těchto informací se neustále vyvíjí, některé prvky ale zůstávají stále stejné. Důvody jsou popsány v této kapitole.

4.1 Používané způsoby zobrazení informací

Nejběžnějším způsobem, jakým automobily zobrazovaly informace o stavu součástek, byly ikonky zobrazované vedle tachometru automobilu. Tyto ikonky se snaží dodržovat určitý standart, avšak s příchodem složitějších mechanismů využívaných v automobilech a rostoucími možnostmi diagnostiky automobilu, počet těchto symbolů narůstá. V současné době těchto ikon existuje více než 150 [3]. Výhody systému ikon jsou následující:

- Možnost umístit velké množství ikon na malý prostor.
- Pro lidi, kteří jsou s ikonami dobře obeznámeni, nejrychlejší způsob předání potřebné informace.
- Není potřeba přidávat dodatečné obrazovky.

Kvůli narůstajícímu počtu a celkové nepřehlednosti ikon byly některé informace přeneseny na vedlejší displej palubního počítače, který dnes slouží převážně k zobrazování navigačních informací. Informace o součástkách jsou zobrazovány formou textu, popřípadě jednoduchého obrázku zvýrazňujícího daný prblém. Tento způsob zobrazení ilustruje obrázek 4.1. Mezi výhody získané využíváním tohoto systému patří:

- Možnost zobrazit komplexnější informace.
- Netřeba se učit význam ikon, vše může být popsáno na displeji.
- Možnost vzájemné interakce s uživatelem.

Tímto se také dostáváme ke způsobu, který není doposud využívaný. Jedná se o zobrazování informací za pomocí 3D grafiky, které se snaží shrnout tato práce. Tento systém obohacuje výhody systému s obrázky o následující:

Obrázek 4.1: Možný způsob zobrazení diagnostických informací využívaný v současnosti.

- Vytváří modernější estetický dojem a posouvá možnosti interakce s uživatelem na další úroveň.
- S pomocí animací a otáčení vozidla pomáhá uživateli lépe lokalizovat vzniklý problém.

Při těchto přechodech si lze povšimnout jisté redundance. Předchozí způsoby nejsou kompletně nahrazeny, nicméně doplněny o nové možnosti. Je tomu tak převážně z důvodu existence širokého spektra uživatelů automobilů. Ne všechny způsoby vyhovují všem lidem a v situacích kdy je důležité, aby byl problém viditelný pro všechny, a pokud možno okamžitě, je třeba využívat kombinaci všech možných přístupů, na které jsou lidé zvyklí.

Kapitola 5

Návrh grafické vizualizace diagnostických informací o vozidle

Jedním z cílů práce je návrh uživatelského rozhraní, které může být novým způsobem vizualizace diagnostických informací automobilu. Palubní desky auta jsou v dnešní době vybaveny poměrně schopnými malými počítači, které buďto mají zabudovaný internetový prohlížeč, nebo jsou schopny poskytnout potřebnou funkcionalitu pro jeho nahrazení. I v situacích kdy automobil není vybaven palubním počítačem, se dá přidat zařízení jako tablet, nebo mobil, které internetový prohlížeč má.

Při návrhu aplikace je důležité rozhodnout se, jaké nástroje budou používány. WebGL je pro aplikaci vhodnou volbou převážně kvůli jeho univerzálnosti. Výše zmíněná zařízení dnes téměř ve všech případech podporují tuto technologii.

Součástí této práce není návrh sběrnice zprostředkovávající komunikaci mezi autem a aplikací. Veškeré akce jsou simulovány sadou tlačítek. Hlavní náplní aplikace je vizuální demonstrace informací o vozidle.

Zde je v několika odrážkách shrnuto, co by výsledná aplikace měla zahrnovat:

- Zobrazení zvolených varování vizuálně i formou textu. Důraz je zde kladen na viditelnost.
- Možnost volně otáčet kameru kolem automobilu.
- Animace prvků, které si to svojí funkčností vyžadují.
- Ovládání simulující signály z automobilu.
- Pohyby kamery směrem k detailu součástky.

Kromě návrhu prvků, které by aplikace měla mít, můžeme hovořit i o budoucích rozšířeních aplikace. Jedná se hlavně o vytvoření zmiňované sběrnice a nasazení aplikace do reálného prostředí. Další rozšíření se můžou týkat pokročilého vylepšování vzhledu aplikace, buďto použitím lepších modelů, nebo vytvořením nové sady shaderů pro vykreslování částí automobilu.

5.1 Koncept rozhraní

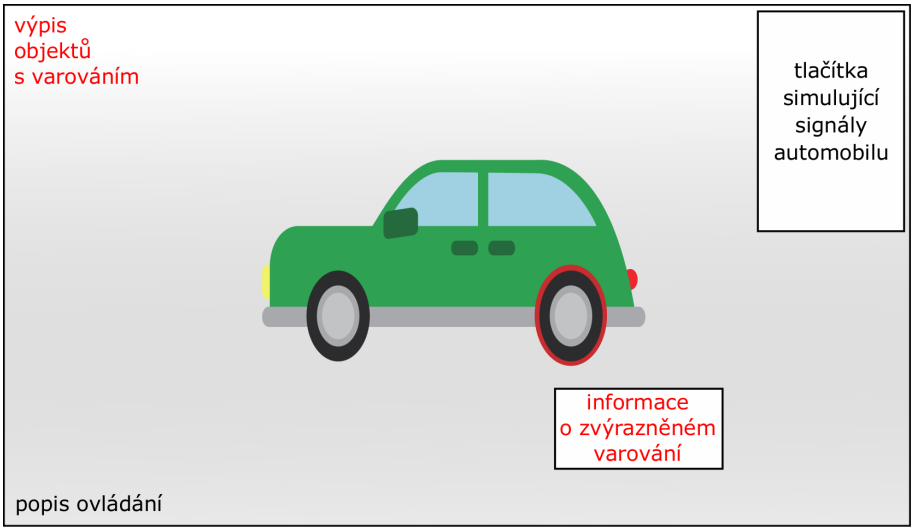
Klíčové prvky pro tvorbu rozhraní zobrazující informace o vozidle jsou jednoduchost ovládání, jednoznačnost zobrazených informací a přehlednost. Jednoduchost ovládání je zajiš-

těna dvojicí přístupů, ovládáním pomocí určených kláves simulujícím tlačítka na volantu automobilu, popřípadě ovládacího kolečka a ovládáním myší simulujícím dotykový displej palubního počítače. Pro jednoznačnost jsou kromě zvýraznění částí modelu zavedeny textové informace o současných varováních v automobilu. Přehlednost pak zajišťuje rozložení a odsazení jednotlivých prvků.

V pracovní verzi aplikace se také nachází panel s tlačítky, který má za úkol nahrazovat signály přicházející z automobilu. Tento panel by se ve verzi pro palubní počítač vůbec nevyskytoval.

Rozložení prvků

Rozložení všech elementů aplikace je ilustrováno na obrázku 5.1. Zobrazení modelu automobilu předchází načítací animace zobrazená uprostřed vykreslovací plochy. Prvek s tlačítky simulující signály je možné schovávat a zobrazovat pro dobrou viditelnost zobrazovaných částí automobilu. Při příchodu signálu z automobilu je zvýrazněna daná součástka a při potvrzení výběru je zobrazen detail této součástky s potřebnými informacemi. Přechod těchto zobrazení je proveden plynulým posunem kamery. Ve fázi detailního náhledu nelze přepínat součástky, ani posunovat kameru. Zrušit detailní náhled umožňuje opětovné potvrzení.

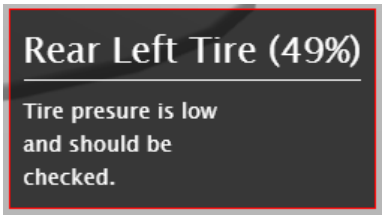


Obrázek 5.1: Rozložení prvků aplikace.

Informace v detailním náhledu musí být zobrazeny v jednotném stylu, stručně s viditelným textem. Součástí těchto informací by měl být nadpis týkající se konkrétní součástky, popřípadě také procentuální hodnoty stavu u součástek, které tyto informace vyžadují. Pod nadpisem by se měl nacházet detailnější popis problému s možným řešením. Příklad informací detailního náhledu využitého v aplikaci je vidět na obrázku 5.2.

Zvýrazňování modelů

Pro jasné označení modelů vyžadující pozornost uživatele lze využít vytažení okrajových hran modelu. Pro toto vytažení byla zvolena červená barva, protože dodává zvýraznění požadovaný varovný efekt. Tmavý a světlý odstín této barvy signalizuje, který zvýrazněný objekt je momentálně zvolený. Měnit zvýrazněný model se dá pomocí šipek na klávesnici.



Obrázek 5.2: Informace o stavu součástky.

Zvolené prvky

Pro demonstrační účely byly pro zvýrazňování zvoleny součásti několika různých typů. Prvním z těchto typů jsou dveře, kapota a kufr, které je potřeba kromě zvýraznění také otevírat. Tyto součástky mění stav v závislosti na příchozím signálu. Druhým typem jsou součásti motoru, které při svém zvýraznění pro dobrou viditelnost také otevírají kapotu. Kapota v takovéto situaci ale není zvýrazněna, aby nedocházelo k misinterpretaci dané situace. Třetím typem jsou součásti, které nereagují na signál, ale mají svoji hodnotu, která se může měnit. V pracovní verzi aplikace je tato hodnota simulována posuvníkem například u nahuštění pneumatik automobilu. Pokud klesne hodnota těchto prvků pod určenou hodnotu, budou zvýrazněny. Informace o stavu tohoto posuvníku jsou promítnuty také do detailu zobrazovaného prvku a je třeba aby se dynamicky měnili se změnou hodnot posuvníku.

Animace

Jedním z nejpřirozenějších způsobů informování uživatele o změnách stavu automobilu jsou animace. Z tohoto důvodu jsou některé pohyblivé části automobilu v mojí aplikaci animovány. Jedná se o dveře, kufr a kapotu automobilu. Využití animací v knihovně Three.js s sebou přináší několik obtíží, se kterými jsem se musel vypořádat. Tyto problémy jsou zdokumentovány v následující kapitole. Na obrázcích 5.3 a 5.4 lze pro ilustraci vidět efekt využívání animací.

Obrázek 5.3: Vizualizace automobilu v klidovém stavu, bez aplikace animací.

Obrázek 5.4: Vizualizace automobilu po aplikaci animací, zvýrazňujících otevřené dveře a kufr.

Kapitola 6

Implementace

Tato kapitola se zabývá samotnou implementací aplikace a problémy, které bylo třeba v průběhu řešit. Základní kostra aplikace je psána v HTML5, veškerá funkcionalita je psána v jazyce Javascript. Shadery využité v aplikaci jsou psány v jazyce GLSL ES.

6.1 Grafické jádro aplikace

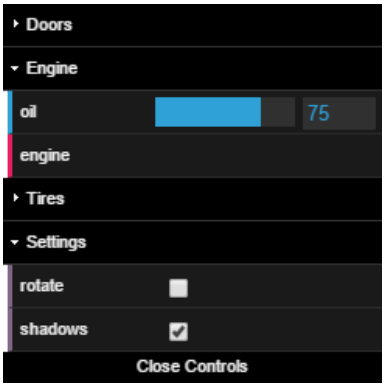
Aplikace je založena na prolínání jednoduchých HTML prvků s 3D prvky vykreslenými do canvasu, o který se stará knihovna Three.js. První operace, která je provedena po spuštění aplikace je kontrola webového prohlížeče. Pokud prohlížeč není schopen pracovat s WebGL, je uživatel přesměrován na stránku o podpoře WebGL[1]. K této detekci slouží přídavná knihovna `Detector` vytvořená pro potřeby Three.js.

Po detekci WebGL je vytvořena scéna a všechny sounáležitosti jako jsou světla a kamera. V aplikaci jsou využity 3 druhy světel, ambientní osvětlení, směrové světlo a světlo generující stín automobilu. Důvody využití těchto světel jsou popsány v kapitole o grafických efektech. Po načtení všech modelů zmíněných v další sekci je spuštěna renderovací smyčka. Tato smyčka se stará o aktualizaci všech potřebných hodnot. Mezi tyto hodnoty patří ošetření stisků kláves, zajištěné knihovnou `KeyboardState`, pro procházení objektů a jejich potvrzování. Dalšími hodnotami, které je třeba aktualizovat jsou statistiky vytvořené knihovnou `Stats` pro účely testování výkonu výsledné aplikace. Na obrázku 6.1 jsou vidět informace zobrazované v těchto statistikách. Je důležité pamatovat také na samotné vykreslování scén.

Obrázek 6.1: Statistiky poskytované knihovnou Stats. Počet snímků za sekundu (vlevo), doba potřebná k vykreslení jednoho snímku (uprostřed) a operační paměť využitá aplikací (vpravo).

Potřebu existence tlačítek, simulujících signály přicházenící z automobilu, naplňuje knihovna `dat.gui`. Tato knihovna poskytuje jednotný styl pro vytváření základu uživatelského rozhraní mezi aplikacemi. Ukázku tohoto stylu je možné vidět na obrázku 6.2. Tato knihovna nabízí kromě standartního typu tlačítek, také posuvníky a přepínače.

[1] http://get.webgl.org



Obrázek 6.2: Demonstrace jednotného stylu rozhraní vytvořeného knihovnou dat.gui.

6.2 Načítání modelů

Loadery využívané v Three.js zajišťují správné načtení modelů. Načítání těchto modelů funguje asynchronně. Načítají se všechny najednou a zároveň nechávají program plynout dál i bez nich. Je proto třeba nějakým způsobem ověřovat, zdali jsou všechny modely načtené a až poté ukončit načítací obrazovku, popřípadě povolit veškerou funkčnost aplikace.

K ověření načtení všech modelů slouží objekt `LoadingManager`. Tento objekt má ve své funkci `onProgress` přístup k počtu modelů, které se načetly, které mají být načteny a také k názvu souboru posledního načteného modelu. Toho je v mé aplikaci využito při ověřování přerušení načítání aplikace a také k vytváření pomocných proměnných, ve kterých si uchovávám pořadí všech načtených modelů vyžadujících pozdější přístup.

Pokud nejsou všechny modely načteny, aplikace nevykresluje scénu, ale jednoduchý HTML prvek, který pomocí CSS stylů vytváří animaci. Tato animace signalizuje uživateli načítání aplikace. Jakmile je načítání všech modelů dokončeno, tento prvek je nahrazen základním elementem pro vykreslování. Navržený vzhled animace demonstruje obrázek 6.3.

Obrázek 6.3: Ukázka stavů jednoduché načítací animace.

Přidání modelů, na jejichž změnu stavu je třeba v automobilu upozorňovat, lze následujícím způsobem. Do adresáře *model* nahrajeme soubor obsahující potřebný model. Tento model buďto nese název *door_x* pro animované části, nebo *part_x* pro části bez animací. X v názvu těchto souborů udává identifikační číslo modelu. Tyto čísla jsou společná pro animované i neanimované modely. Model nepohyblivé části musí mít vždy číslo vyšší než modely animovaných částí a čísla musí po sobě následovat. Pro danou funkcionalitu není možné některá čísla přeskočit. Poté stačí inkrementovat konstantu `DOOR_COUNT`, popřípadě `PART_COUNT`, a vytvořit odpovídající enumeraci v `PARTS_ENUM`.

Totohoto přístupu je docíleno využitím cyklů pro načítání modelů se správnou implementací `LoadingManageru`. Limitem pro počet možných načtených modelů je v mé aplikaci hodnota 99.

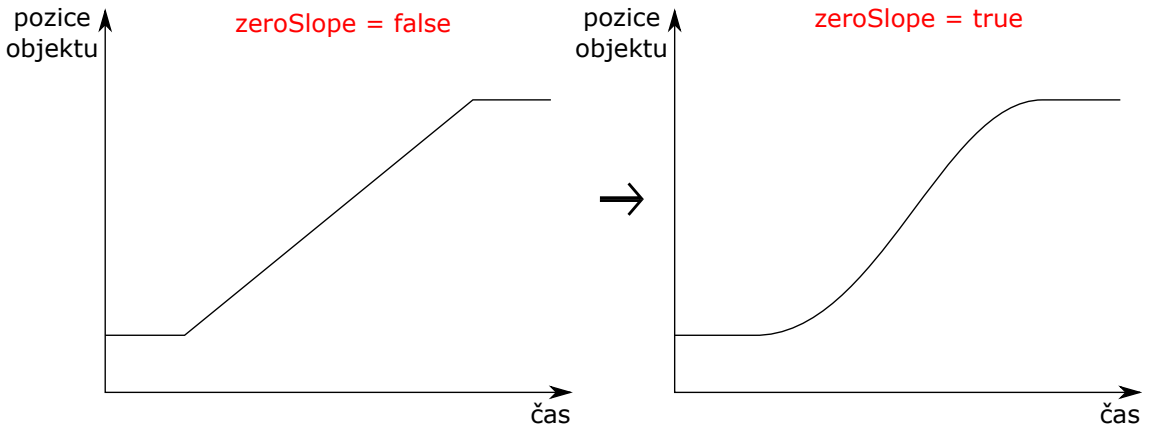
6.3 Realizace animací

Jelikož animace nejsou v současné verzi Three.js dostatečně zdokumentované, pokusím se v této sekci vysvětlit základní principy, na které jsem přišel zkoumáním zdrojových kódů a příkladů vytvořených na oficiální stránce Three.js[2].

Při vytváření modelu pro aplikaci je v současné době nejvýhodnější použít exportér do formátu JSON. Tento exportér nyní existuje pro modelovací nástroj Blender, který byl využit pro úpravu všech modelů využitých v aplikaci, a také pro tvorbu animací. Pro správný export animací je třeba zaškrtnout políčka **morph animation**, **keyframe animation** a **embed animation**. Bez těchto modifikátorů nebude výstupný soubor obsahovat správně formátované animace pro Three.js.

K ovládání animací je třeba vytvořit pro každý mesh `AnimationMixer`. Tento objekt uchovává informace o současném stavu modelu. Nacházejí se v něm také jednotlivé akce vytvořené pomocí `clipAction()`. Výhodou tohoto systému je možnost mezi jednotlivými akcemi plynule přecházet. Každá akce má proto svoji váhu. U jednotlivých akcí je možné nastavit jejich délku trvání `duration`, rychlost přehrávání `timeScale` s možností využití záporných hodnot pro přehrávání pozpátku, popřípadě ovlivňovat plynulost jejich ukončení pomocí hodnot `zeroSlope` demonstrovaných na obrázku 6.4 [6].

V mém systému animací jsou objekty typu `AnimationMixer` uloženy v poli, ke kterému přistupuji pokaždé, když chci aktivovat nějakou animaci. Správný index tohoto pole určuji pomocí proměnných, jež nesou pořadí načítaných modelů. O těchto proměnných jsem se zmiňoval v předešlé sekci. Jaký směr přehrávání aplikace je třeba zvolit, není třeba v mé implementaci řešit. V případě příchodu signálu z automobilu je proměnná `timeScale` vynásobena hodnotou -1, což zajišťuje, že se animace přehraje vždy v opačném směru, než tomu bylo doposud. Tato technika funguje i v případě příchodu signálu ve chvíli, kdy animace ještě nebyla zcela dokončena.



Obrázek 6.4: Průběh animace při rozdílných hodnotách zeroSlope.

V případě využití shaderů, je potřeba u každého vertexu zohlednit jeho současnou polohu vzhledem ke stavu animace. Slouží k tomu proměnné `morphTarget`. Při aktivním využívání směru normál musíme zohledňovat i proměnné `morphNormal`.

[2] http://threejs.org/examples/

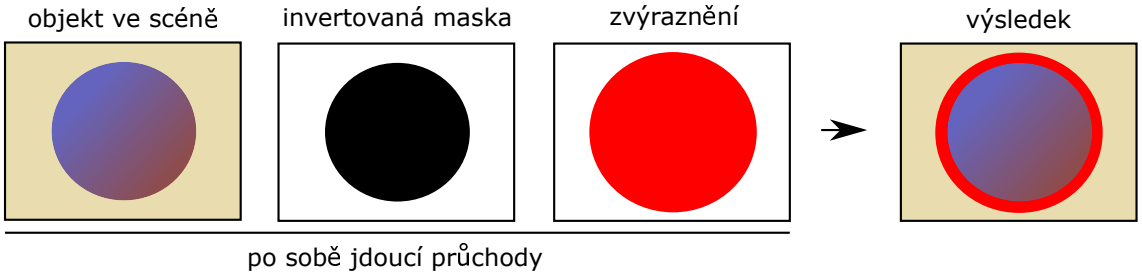
6.4 Využití shaderů

Shaderů je v aplikaci využíváno jak k dosažení potřebné funkcionality návrhu, tak k vylepšení vzhledu cílové aplikace. K použití shaderů v knihovně Three.js je třeba vytvořit materiál typu `ShaderMaterial`. Tento materiál vyžaduje objekt uniforms, který obsahuje všechny proměnné, které budou shaderu předávány z aplikace. Tento objekt je vytvořen jako asociativní pole s definicí proměnných, jejich hodnot a typů. Další dva objekty, které vyžaduje, jsou vertex a fragment shader. Ty jsou definovány jako řetězce s kódem shaderu v jazyce GLSL ES. Tento materiál je poté možno aplikovat na jakýkoli mesh ve scéně.

Tímto způsobem je vytvořen shader pro zobrazení materiálu karosérie vozidla. Jedná se o shader sphere mapping, u kterého jsou odrazy světla promítány na jednoduchou texturu kulovitého tvaru jako na obrázku 6.5. Pro barevné variace můžu volit jiné textury `matcap` ve složce s texturami.

Obrázek 6.5: Demonstrace využití sphere mappingu.

Shader zvýrazňující hrany objektu pro správný chod vyžaduje mnohem více operací. Kromě samotné implementace shaderu, jejíž teorie je popsána v části o grafických efektech pro vylepšení vzhledu aplikace, musím vytvořit také další dvě scény. První pro vytvoření masky objektu a druhou pro samotné zvýraznění. Pro zobrazení žádoucího výsledku je tedy třeba použít více renderovacích průchodů. K tomu nám slouží objekt `EffectComposer`. Tyto průchody demonstruje obrázek 6.6.



Obrázek 6.6: Využití renderovacích průchodu pro zvýraznění objektu.

Kromě běžných průchodu `RenderPass` je důležité přidat i průchod `ShaderPass` bez kterého není možné vykreslovat scénu. Důvodem je implementace a návrh zmíněného `RenderPassu`,

který nemá žádný příznak `renderToScreen`. Je renderován pouze do bufferu `EffectComposeru`, tedy není určen k používání bez dalších průchodů. V mé implementaci jsem si jako `ShaderPass` zvolil předpřipravený FXAA shader.

6.5 Pohyb kamery

Při zobrazování detailů součástek automobilu je třeba posunout kameru směrem k nim, aby bylo zřejmé, na co se mají uživatelé zaměřit. Kromě toho je také třeba řešit plynulý pohyb kamery kolem automobilu v případě, že žádná součást nebyla vybrána. K tomuto slouží dva přístupy, mezi kterými je možno přepínat v závislosti na současné situaci.

První přístup řeší přídavná knihovna k Three.js `OrbitControls`. Tato knihovna zajišťuje automatické rotování kamery kolem objektu. Zejména je třeba určit, kterou kameru a renderer má knihovna využívat, definovat limity volného posunování a při renderování scény aktualizovat hodnotu tohoto ovládání.

Druhý přístup je poněkud složitější. Při přepínání na tento přístup je nutné deaktivovat ovládání prvního přístupu, jinak by se oba přístupy hádaly a způsobovaly nedefinované chování. Pro tento přístup je využita knihovna `Tween`. Pomocí této knihovny jsou vytvářeny abstraktní body v prostoru, které je možné přesunovat z místa na místo plynulou animací. Na každý úsek přesunu těchto bodů je možné navázat pozici kamery s využitím jejich funkce `onUpdate` a vytvořit tak plynulý kamerový přechod z místa na místo. Knihovna nabízí spoustu možností práce s grafem průběhu animace, například zeroSlope zmíněný v sekci realizace animací. S využitím funkce `lookAt` je také možné držet pozornost kamery v určeném bodě.

Kapitola 7

Testování a výsledky

Tato kapitola se zabývá testováním výsledné aplikace. Na testování bylo nahlíženo ze dvou úhlů pohledu. Jedním z nich bylo, jak si aplikace vede, co se týče výkonu na různých strojích. Druhým bylo testování schopnosti uživatelů zorientovat se při používání uživatelského rozhraní aplikace.

7.1 Výkon

Cílem této práce není pouze vytvořit aplikaci pro grafickou vizualizaci diagnózy automobilu, ale také demonstrovat využití WebGL k zobrazení scény ve webovém prohlížeči. Z toho důvodu byl testován i výkon aplikace, a to především kvůli možnosti získání informací pro další vývoj aplikací využívajících WebGL, ale také kvůli budoucím optimalizacím aplikace, která je náplní této práce. Testování bylo prováděno formou dotazníku šířeného skrze sociální sítě. Aplikaci tímto způsobem otestovalo 48 uživatelů. Úkolem uživatelů bylo spustit si danou aplikaci a zaznamenat následující údaje:

- Délka načítací doby.
- Množství vykreslovaných snímků za sekundu v klidovém stavu aplikace.
- Množství vykreslovaných snímků za sekundu při otevírání dveří automobilu.
- Množství vykreslovaných snímků za sekundu při pohybu kamery směrem k zobrazenému detailu.
- Celkový dojem z vizuální stránky aplikace.

Načítací doba

Tato položka dotazníku slouží k ověření závislosti rychlosti internetového připojení na velikosti načítaných modelů aplikace. Otázka v tomto dotazníku se netýkala konkrétní doby potřebné pro načtení všech modelů, ale spíše subjektnivního pocitu. Tato informace může mít vliv na složitost 3D modelů v budoucnu využívaných ve webových aplikacích. Možné odpovědi proto byly:

- Velmi krátká načítací doba.
- Přijatelná načítací doba.

- Příliš dlouhá načítací doba.

Výsledkem této části dotazníku bylo, že doba potřebná pro načtení modelu automobilu, jehož velikost se pohybuje okolo 12,5 MB, je pro většinu uživatelů krátká. Je tedy možné využívat i složitější modely za cenu možné ztráty výkonu aplikace.

Rychlost vykreslování

Rychlost vykreslování byla testována v různých stavech aplikace. Hlavním cílem bylo ověřit, zdali některé složitější úkony zahrnuté v běžném chodu aplikace nebudou mít zásadní vliv na rychlost vykreslování, tedy jestli některé části bude třeba v budoucnu optimalizovat, popřípadě zjednodušit pro plynulý chod aplikací. K záznamu těchto informací byly v dotazníku tři políčka pro vyplnění konkrétní hodnoty počtu snímků za sekundu. K zjištění těchto hodnot sloužil uživatelům rámeček v levém horním rohu aplikace vykreslující vývoj počtu snímků za vteřinu. Kromě těchto políček se v dotazníku nacházel také přepínač zjišťující, jestli bylo testování prováděno na notebooku, nebo na stolním počítači.

Průměr hodnot získaných od uživatelů můžeme vidět v tabulce 7.1.

Případ	**Stolní počítače (počet snímků za vteřinu)**	**Notebooky (počet snímků za vteřinu)**
Klidový stav	46	17
Otevírání dveří	31	12
Pohyb kamery	46	16

Tabulka 7.1: Průměrné hodnoty rychlosti vykreslování na různých strojích.

Z výsledků je patrné, že v rámci optimalizací je potřeba zaměřit se na část kódu zahrnující otevírání dveří, které způsobuje největší zpomalení. Toto zpomalení může být způsobeno přehráváním animací, aktivací dosud neviditelného materiálu s shaderem pro vytažení hran, popřípadě potřebou vykreslovat několik scén najednou.

Celkový dojem

Tato část dotazníku se týkala spíše toho, jakým způsobem se uživatelé staví k využívání 3D grafiky ve webových prohlížečích, jak na ně působí rozhraní využité v mé aplikaci a obecně jaký celkový dojem aplikace vytváří. Jelikož si mohli tuto otázku uživatelé volně interpretovat, byly odpovědi různé. Převážná většina odpovídajících uvedla, že se jim líbí nápad využití 3D grafiky pro sdělování informací skrze webový prohlížeč. Mnoho odpovědí také pozitivně hodnotilo jednoduchost uživatelského rozhraní. Jediné nepříznivé odpovědi zmiňovaly nedostatečnou plynulost mé aplikace.

7.2 Rozhraní

Pro testování intuitivnosti uživatelského rozhraní byla vytvořena zcela odlišná testovací sada. Toto testování probíhalo pod mým dohledem a zúčastnilo se ho pět uživatelů, což je optimální počet testovaných subjektů [10]. Uživatelé předtím s aplikací nepřišli do styku. Mezi testovanými byli lidé různých věkových skupin i pohlaví, pro dosažení co nejobjektivnějších výsledků.

Testovaným byla představena základní funkčnost aplikace, tedy jakým způsobem se ovládá a jak poznat zvýrazněný objekt. Poté jim byla spuštěna aplikace se 3 vybranými objekty a jejich cílem bylo, v co nejkratším čase zjistit, co je s vozidlem špatně. Postup, jakým testovaní uživatelé zjišťovali problémy, byl zaznamenáván pro následnou analýzu. Testování nebylo prováděno za skutečného provozu, výsledky tedy neodpovídají skutečné využitelnosti aplikace v automobilovém provozu.

Výsledkem tohoto testování byly následující poznatky:

- Největší problém s orientací v aplikaci měl nejstarší testovaný subjekt. Aplikace by byla vhodná převážně pro lidi zvyklé děnně využívat moderní technologie.
- Hlavní potíže činila testovaným potřeba vystoupit z náhledu detailu pro vybírání dalších objektů. Možnost volně přecházet mezi objekty i při zobrazení detailu by mohla být součástí budoucích aktualizací aplikace.
- Mezi prvními detaily, které upoutali pozornost testovaných, byly otevřené dveře. Tento jev by mohl činit problémy při reálném nasazení aplikace, kdy uživatelé mohou přehlédnout závažnější problémy. Bylo by tedy vhodné v budoucích úpravách aplikace navrhnout jasnější zobrazování důležitějších prvků.
- Testovaní neměli problémy s rychlým zjišťováním informací o tom, jak se mají s problémem vypořádat, jakmile ho objevili. Přispívá tomu jednotný styl rozhraní s informacemi.

Kapitola 8

Závěr

Tato práce se zabývala návrhem a implementací 3D diagnostické vizualizace stavu automobilu. Bylo toho dosaženo vytvořením internetové aplikace využívající WebGL a knihovnu Three.js. Pro model automobilu byl zvolen formát souboru JSON, umožňující animovat vybrané části automobilu. Signály přicházející z automobilu byly simulovány sadou tlačítek poskytnutých knihovnou dat.gui. Pro vizuální vylepšení aplikace byly navrženy a implementovány shadery umožňující zvýrazňovat objekty a vytvářet zajímavé materiály promítáním odrazu do kruhových textur. Pohyby kamery byly řešeny vytvořením abstraktních bodů v prostoru, na něž se navázala pozice kamery. Tyto body byly poté přesouvány dle potřeb aplikace.

Z testování vyplynulo, že mladší uživatelé nemají problémy s tímto typem uživatelského rozhraní. Hlavní problémy byly způsobeny stejným zvýrazněním různě závažných problémů, popřípadě některými omezeními voleb, které skýtalo detailní zobrazení jednotlivých součástek. Při opravě těchto nedostatků, které mohou být prostředkem budoucích úprav aplikace, lze považovat výsledný způsob zobrazování informací o stavu vozidla za použitelný. Vizuální stránku aplikace většina uživatelů hodnotila kladně. Testování také ukázalo, že v současné době je zobrazování složitějších scén ve webových prohlížečích spíše záležitost stolních počítačů, u kterých si rychlejší grafický adaptér poradí s plynulým vykreslováním. Standartní notebooky u výkonu tohoto typu spíše zaostávají.

Budoucnost aplikace nabízí několik možností. První z těchto možností je vytvoření sběrnice pro komunikaci automobilu s touto aplikací a tedy i odebrání sady tlačítek zprostředkovávajících danou funkcionalitu. S touto možností souvisí také nové způsoby testování. Jde o testování aplikace v reálném prostředí, nebo jeho simulaci. Toto testování nám může poskytnout dodatečné informace o využitelností tohoto způsobu vizualizace. Další možné formy budoucího vylepšování aplikace se týkají tvorby nových a kvalitnějších modelů, u kterých nebude shader chybně zvýrazňovat ostré hrany. Lze toho docílit jemným vyhlazením těchto hran. Kromě popsaných možností se také nabízí využití jádra aplikace pro jiné účely. Může to být například webová prezentace nového vozidla, popřípadě 3D konfigurátor vozidla pro jeho prodej.

Výslednou aplikaci můžeme vidět na obrázku 8.1.

Obrázek 8.1: Demonstrace vytvořeného uživatelského rozhraní.

Literatura

[1] Bao, H.; Hua, W.: *Real-Time Graphics Rendering Engine*. Springer Berlin Heidelberg, 2011, ISBN 978-3-642-18341-6.

[2] Birn, J.: *Digital Lighting & Rendering*. New Riders Publishing, 2014, ISBN 978-0321928986.

[3] DashboardSymbols.com: *Dashboard Symbols*. [online], Navštíveno 3.5.2016.
URL http://dashboardsymbols.com/home/contact/

[4] Dirksen, J.: *Learning Three.js: The JavaScript 3D Library for WebGL*. Packt Publishing Ltd., 2015, ISBN 978-1-78439-221-5.

[5] Khronos Group: *OpenGL ES 2_X*. [online], Navštíveno 1.5.2016.
URL https://www.khronos.org/opengles/2_X/

[6] Parent, R.: *Computer Animation: Algorithms and Techniques*. Morgan Kaufmann, 2012, ISBN 978-0124158429.

[7] Sanchez, Jaume: *Spherical Enviroment Mapping*. [online], Navštíveno 3.5.2016.
URL https://www.clicktorelease.com/blog/creating-spherical-environment-mapping-shader

[8] Simpson, R. J.; Kessenich, J.: *The OpenGL ES Shading Language*. [online], Navštíveno 2.5.2016.
URL https://www.khronos.org/files/opengles_shading_language.pdf

[9] TutorialsPoint.com: *WebGL Graphics Pipeline*. [online], Navštíveno 1.5.2016.
URL http://www.tutorialspoint.com/webgl/webgl_graphics_pipeline.htm

[10] Virzi, R. A.: Refining the Test Phase of Usability Evaluation: How Many Subjects is Enough? *Hum. Factors*, ročník 34, č. 4, Srpen 1992: s. 457–468, ISSN 0018-7208.
URL http://dl.acm.org/citation.cfm?id=141691.141700

Přílohy

Seznam příloh

Příloha A

Obsah CD

Na přiloženém disku se nachází následující soubory a složky:

- /src/ – složka se zdrojovými soubory aplikace
- /doc/ – složka se zdrojovými soubory technické zprávy
- /poster_src/ – složka s editovatelným souborem plakátu
- /LICENSE.txt – soubor s licencí
- /README.txt – soubor s instrukcemi k instalaci a ovládání aplikace
- /poster.png – demonstrační plakát aplikace
- /xzveri04.pdf – elektronická verze této technické zprávy

Příloha B

Plakát

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

3D ZOBRAZENÍ DIAGNOSTICKÝCH INFORMACÍ O STAVU AUTOMOBILU

UŽIVATELSKÉ ROZHRANÍ

Rozhraní je navrženo jako aplikace internetového prohlížeče využívající technologii WebGL s knihovnou Three.js. Dále byly využity tyto knihovny:

DAT.GUI

Tato knihovna zprostředkovává sadu tlačítek simulující příchozí signály z automobilu.

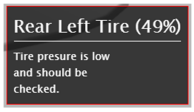
JQUERRY

Knihovna usnadňuje práci s HTML elementy zobrazovanými v aplikaci. Konkrétně se jedná o textové informace o zobrazených součástkách automobilu.

TWEEN

Pomocí této knihovny jsou vytvářeny abstraktní body v prostoru na jejichž pozici se váže pozice kamery při pohybu směrem k detailům součástek.

ZOBRAZOVANÉ INFORMACE O SOUČÁSTCE



GRAFICKÉ SHADERY

Vizuální návrh aplikace vyžadoval implementaci následujících grafických shaderů:

OUTLINE SHADER

Tento shader zvětšuje objekt, který má být zvýrazněn posunutím všech vertexů ve směru jejich normálových vektorů. Na tento zvětšený objekt je poté aplikována maska tvořena objektem původním. Barvu zvětšeného objektu lze libovolně měnit.

SPHERE MAPPING

Shader tvořící výsledný vzhled materiálu karosérie automobilu je implementován jako promítání odrazu pohledu do kruhovité textury. Toto promítání je počítáno pomocí interpolace na grafické kartě.

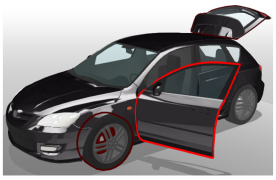
POUŽITÉ TECHNOLOGIE

VÝSLEDNÝ VZHLED APLIKACE

KRUHOVÁ TEXTURA PRO MATERIÁL VOZIDLA

PRŮCHODY VYKRESLOVÁNÍ SCÉNY PRO OUTLINE SHADER

ZVÝRAZNĚNÍ OBJEKTŮ A MATERIÁL VOZIDLA



VYPRACOVAL: Jiří Zvěřina
VEDOUCÍ PRÁCE: Ing. Michal Španěl, Ph.D.

http://www.stud.fit.vutbr.cz/~xzveri04/bp/