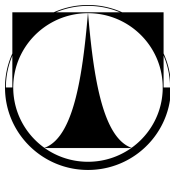
TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií



BAKALÁŘSKÁ PRÁCE

Liberec 2012 **Stanislav Franc**

TECHNICKÁ UNIVERZITA V LIBERCI

Fakulta mechatroniky, informatiky a mezioborových studií

Studijní program: B2646 – Informační technologie

Studijní obor: 1802R007 – Informační technologie

Softwarová knihovna pro vykreslování 3D modelů pomocí HTML5

Software library for rendering of 3D models using HTML5

Bakalářská práce

Autor: **Stanislav Franc**

Vedoucí práce: Ing. Jiří Jeníček, Ph. D.

V Liberci 15. 5. 2012

Originální zadání

Prohlášení

Byl(a) jsem seznámen(a) s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. O právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této zkutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracoval(a) samostatně s použitím uvedené literatury a na základě konzultací s vedoucím bakalářské práce.

Datum: 15. 5. 2012

Podpis:

Poděkování

Chtěl bych poděkovat Thomasovi Gilesovi za jeho tutoriály pro WebGL[1]…

Abstrakt

Cílem práce je vytvořit softwarovou knihovnu pro vykreslování 3D modelů na webu za pomocí HTML5. Knihovna by měla umožňovat základní prohlížení modelu, jako je například otáčení nebo zvětšení. Měla by umožnit vykreslení několika modelů do jedné scény. Jednotlivé modely mohou být různobarevné, nebo otexturované. Knihovna by měla zajišťovat plynulý provoz při použití desítek až stovek modelů.

Výsledkem práce je softwarová knihovna napsaná v Javascriptu využívající grafické knihovny WebGL. Model se vykresluje přímo v prohlížeči, na místě prvku typu canvas, není třeba žádného pluginu. Prohlížení modelu se provádí za pomocí prvků na webové stránce a použitím myši. Model může být osvícen směrovým světlem. Každý model může mít několik barev, nebo textur. Knihovna umožňuje zobrazit několik modelů v jedné scéně.

Abstract

The purpose of this work is creating a software library for rendering 3D models on web page with usage of HTML5. The library should allow basic viewing of model, for example rotation or scaling. It should allow to render several models in one scene. Every model could have many colors or textures. Library should allow smooth run even with use of tens to hundreds of models.

The result of work is software library written in Javascript using graphical library WebGL. The model is rendered inside web browser on place of element canvas. There is no need of any plugin. Viewing of the model is allowed by elements of the web page or by mouse. The model can be lighted by simple directional light. Every model can use several colors or textures. Library allows rendering of several models in one scene.

Obsah

Seznam použitých zkratek

3D	Three-Dimensional space (Trojrozměrný prostor)
HTML	Hyper Text Markup Language (Značkovací jazyk pro hypertext)
MS	Microsoft
VB. NET	Visual Basic. NET
VRML	Virtual Reality Modeling Language (Modelovací jazyk pro Virtuální realitu)
X3D	Extensible 3D Graphics (Rozšiřitelná 3D grafika)
XML	Extensible Markup Language (Rozšiřitelný značkovací jazyk)
WebGL	Web-based Graphics Library (Grafická knihovna pro web)
DOM	Document Object Model (Model objektů dokumentu)
DWG	Drawing (Výkres)
DWF	Design Web Format (Návrhový webový formát)
3DS	3D Studio
XHTML	Extensible Hyper Text Markup Language (Rozšiřitelný značkovací jazyk pro hypertext)
HTTP	Hypertext Trensfer Protocol (Protokol pro přenos Hypertextu)
HTTPS	Hypertext Trensfer Protocol Secure (Zabezpečený protokol pro přenos Hypertextu)
PHP	Původně Personal Home Page, nyní Hypertext Preprocesor
GLSL	OpenGL Shading Language
RGB	Označení uchování barev do složek červené, zelené a moderé
FPS	Frames per second (počet snímků za sekundu)

Seznam Ilustrací a Grafů

1 Úvod

Cílem této práce je vytvořit softwarovou knihovnu, která by umožňovala vykreslování předem vytvořených 3D modelů na webových stránkách. Tato knihovna by měla být založena na prostředcích, které nabízí standard pro psaní webových stránek HTML ve verzi 5.

Knihovna musí umět získat potřebná data k vytvoření 3D modelu ze souboru, vykreslit jej na obrazovku a umožnit prohlížení modelu, ke kterému patří zvětšování, zmenšování, otáčení pohledu na model a posun jednotlivými částmi modelu.

Knihovna bude dále používána pro další účely rozšiřující problematiku vykreslování modelů na webu, ale už ve specifičtějším zaměření, jako je například sesazení a vykreslení desek plošných spojů.

2 Seznámení se s prostředím

2.1 Rozdíly HTML5 oproti předchozí verzi

HTML5[2] je nový standard pro psaní webových stránek. Snaží se řešit problémy minulých verzí HTML a pokrýt potřeby webových aplikací, které v předchozích verzích nebyly umožněny. Přičemž ale zachovává zpětnou kompatibilitu s předchozími verzemi.

Verze 5 přidává mnoho nových HTML značek sémanticky definujících strukturu stránky. Mezi ně patří například <*video*>, <*audio*>, <*canvas*>, tyto prvky umožňují použití a zpracování multimédií a grafiky v prohlížeči bez nutnosti použití přídavných modulů. Další značky, jako například <*section*>, <*article*>, <*nav*>, slouží k obohacení sémantického obsahu dokumentu.

Ze stejného důvodu byly přidány nové atributy, přičemž některé značky a atributy byly odstraněny. HTML5 také definuje některé detaily ke zpracování nesprávných dokumentů tak, aby syntaktické chyby byly jednotně zpracovány všemi prohlížeči a ostatními programy.

HTML5 je stále ve stádiu návrhu.

2.2 Možnosti pro vykreslování 3D na webu

2.2.1 Java

Java[3] je vyšší programovací jazyk odvozený z jazyka C++. Byl navrhován, aby byl více uživatelsky přátelský než jazyk C++. Zjednodušení práce programátorovi nepatrně zpomaluje rychlost chodu programu. Java programy lze spouštět na několika platformách, ve webovém prohlížeči běží na Java appletu.

Java applet je vždy spouštěn v Sandboxu – virtuálním stroji, kde nemá přístup k vnějším datům jako například schránce nebo systému souborů. Java může využívat hardwarovou akceleraci 3D a proto je vhodná k vykreslování 3D modelů.

Výhody:

- Hardwarová akcelerace 3D
- Rychlý chod programu
- Rozšířené používání

Nevýhody:

- Nutnost instalace přídavného modulu do prohlížeče

2.2.2 Flash

Flash[4] byl původně, nástroj pro kreslení obrázků, později se používal pro zobrazování animace, vytvořené vektorovou grafikou, na webu. Flash začal být oblíbený díky malé velikosti souboru s animací. Flash má v sobě implementovaný programovací jazyk ActionScript, který slouží k vytvoření interaktivní animace a vývoji aplikací. Od verze 11 začíná Flash Player podporovat hardwarovou akceleraci.

Výhody:

- Rozšířené používání
- Od verze 11 hardwarová akcelerace 3D

Nevýhody:

- Nutnost instalace přídavného modulu do prohlížeče

2.2.3 MS Silverlight

MS Silverlight[5] je program pro psaní a spouštění zásuvných modulů pro webové prohlížeče, nebo jiných internetových aplikací. Hlavní využití má v business a multimediálních aplikacích. Aplikace mohou běžet v prohlížeči, nebo ve vlastním okně mimo prohlížeč. Jako programovací jazyk lze zde použít C#, VB. NET, Javascript a IronPython. Od verze 5 je přidána hardwarová akcelerace 3D.

Výhody:

- Od verze 5 hardwarová 3D akcelerace

Nevýhody:

- Nutnost instalace přídavného modulu do prohlížeče
- Malá rozšířenost

2.2.4 Google O3D

Plugin pro vykreslování 3D modelů na webu, je napsán v programovacím jazyce C, takže komunikuje přímo s hardwarem[6]. Dne 7. 5. 2010 Google ohlásil, že by O3D měl přejít z pluginu napsaného v C na Javascriptovou knihovnu využívající technologie WebGL

Výhody:

- používá hardwarovou akceleraci

Nevýhody:

- Nutnost instalace přídavného modulu do prohlížeče (v pluginové verzi)
- Minimální rozšíření

2.2.5 Použití 2D obsahu canvasu

HTML5[2] obsahuje element canvas, díky kterému se dají vykreslovat různé dvourozměrné obrazce. Bohužel nemá prostředky na přímé vykreslování 3D a nemá možnost použití grafického adaptéru a všechny úkony tedy provádí procesor. Za pomocí Javascriptu by se dal 3D model přepočítat do 2D prostoru.

Výhody:

- Nevyžaduje instalaci přídavného modulu prohlížeče
- Nepotřebuje prohlížeč s podporou WebGL

Nevýhody:

- Zpracování grafiky běží na procesoru

2.2.6 WebGL

WebGL[7] rozšiřuje možnosti programovacího jazyka Javascript o vytvoření interaktivní 3D grafiky v jakémkoliv kompatibilním prohlížeči, bez použití pluginu.

WebGL je založen na OpenGL ES 2.0, grafické knihovně pro vykreslování 3D grafiky v počítačových aplikacích. Pro vykreslování na stránky využívá elementu canvas a jeho WebGL obsah. Je podporován prohlížeči Google Chrome 9, Mozilla Firefox 4.0, Safari 5.1, Opera 12 (Alfa) a v některých mobilních prohlížečích. Microsoft podporu WebGL pro Internet Explorer neplánuje, ale existují pluginy pro jeho podporu.

Výhody:

- Nevyžaduje instalaci přídavného modulu prohlížeče
- Používá hardwarové akcelerace

Nevýhody:

- Webový prohlížeč musí podporovat WebGL (lze řešit pluginem)

2.3 Použitelné formáty

2.3.1 VRML

VRML[8] je souborový formát pro popisování aktivní i pasivní 3D vektorové grafiky, používané například v aplikacích virtuální reality, ve kterých se rozšířil nejvíce. Zápis dat je založený na deklarativním programovacím jazyce. Pro zobrazování ve webovém prohlížeči byl stvořen plugin, ale mezi webovými aplikacemi není moc rozšířený.

2.3.2 X3D

X3D[9] je nástupcem VRML. Zápis dat souboru je odvozený od standartu XML. Je to ISO standard, který poskytuje způsob pro uložení grafiky. Stejně jako pro VRML i pro X3D existují pluginy na vykreslování na webu. Většina těchto pluginů má zpětnou podporu pro VRML.

2.3.3 Wavefront Object

Wavefront object[10] je textový formát popisující jednotlivé geometrické informace. Každý řádek obsahuje data některé z informací. Z velké části je považován za univerzální formát. Informace o použitých materiálech jsou obsaženy v přídavném souboru.

2.3.4 Collada

Jedná se o textový formát pro interaktivní 3D aplikace. Collada[11] definuje otevřený standard XML schématu. Formát byl určen pro výměnu informací mezi různými programy, které by jinak používali nekompatibilní formáty.

2.4 Programy pro 3D modelování

2.4.1 Autocad

Program vyvinutý společností Autodesk, původně určený na kreslení technických výkresů, později rozšířen o možnost 3D modelování. Program vytváří 3D modely podle přesných geometrických rozměrů, umožňuje vytvářet vazby mezi vrcholy, hranami a plochami. Dále pak umožňuje přichytávání k těmto a dalším prvkům. Program ukládá soubory do formátu dwg a dwf. Program je licencovaný to znamená nutný nákup licence.

2.4.2 3DS Max

Program vyvinutý společností Autodesk, určený pro vytváření 3D modelů a animací. Má možnost přidávání zásuvných modulů a tím rozšíření dostupných funkcí. Program ukládá soubory ve formátu max, ale program umožňuje exportovat modely do jiných formátů, mezi které patří 3ds, dwg, Wavefront Object a VRML.

Je často využívaný při výrobě reklam a filmů, vizualizaci architektury a konstrukcí. Často se používá k tvorbě grafiky počítačových her. Program je licencovaný to znamená nutný nákup licence.

2.4.3 Blender

Program vyvinutý neziskovou organizací Blender Foundation, určený pro vytváření 3D modelů a animací. Umožňuje vytváření různých simulací, jako například pohyb kouře nebo vody, dále umožňuje simulaci fyzikálních zákonů.

Používá se k vytváření animovaných filmů, interaktivních 3D aplikací včetně videoher. Program ukládá soubory ve formátu blend, dále umí exportovat do jiných formátů, mezi které patří Collada, 3ds, Wavefront Object. Program je zdarma a má otevřený kód – možnost vlastních úprav programu.

2.4.4 Cinema 4D

Program vyvinutý společností Maxon, je určen pro vytváření 3D scény a animací. Je oblíbený mezi 3D grafiky díky své jednoduchosti. Má možnost přidávání zásuvných modulů, které rozšiřují základní funkce, nejpoužívanějším z těchto zásuvných modulů je Body Paint 3D pro pokročilé texturování a malování přímo na 3D objekty.

Program ukládá soubory ve formátu c4d, dále umí exportovat do jiných formátů, mezi které patří 3ds, Wavefront Object, VRML. Je dostupný v české lokalizaci. Program je licencovaný to znamená nutný nákup licence.

2.4.5 SketchUp

Program vyvinutý společností Google, je určen pro vytváření 3D modelů. Je navržený pro architekty, stavební a strojní inženýry, ale i pro filmové tvůrce a vývojáře počítačových her. Tento program umožňuje určit geografické umístění pomocí Google Earth.

Výhodou tohoto programu je možnost vyhledat a stáhnout již hotové modely jako třeba dopravní značky, nebo celé budovy. Existuje ve dvou verzích free, která je volně

stažitelná, a professional, která je placená a nabízí více možností jako například export do jiných formátů, mezi které patří 3ds, dwg a Wavefront Object.

2.5 Programovací jazyk JavaScript

Psaní webových aplikací se provádí buď v programovacím jazyce PHP a ASP, které se spouští na straně serveru, nebo Javascript[12], který provádí akce v prostředí prohlížeče, nebo jejich vzájemnou kombinaci. Javascript je interpretovaný jazyk a často se vkládá přímo do HTML kódu stránky. S jeho pomocí se dají ovládat jednotlivé prvky grafického uživatelského prostředí stránky.

Jeho syntaxe je podobná programovacím jazykům C/C++/Java. I když mají podobné jméno, tak JavaScript s programovacím jazykem Jave nemá moc společného. JavaScript poskytuje možnosti objektově orientovaného programování a některé prvky dědičnosti.

3 Požadavky na 3D knihovnu

3.1 Programátorský pohled

Softwarová knihovna[13] je označení pro soubor obsahující funkce, procedury a hodnoty, které mohou být sdíleny a použity více počítačovými programy. Knihovna zjednodušuje programátorovi práci při vytváření programu tím, že programátor nemusí některá řešení vymýšlet a psát sám, ale použít řešení, které už někdo vymyslel.

Knihovna by měla být zapouzdřený celek, u kterého se k jednotlivým funkcím přistupuje voláním metod a předáváním proměnných. Aby programátor dosáhl svého cíle tak by neměl měnit kód knihovny, pouze přistupovat k jednotlivým funkcím. Zapouzdření knihovny také znemožňuje nechtěné přepsání proměnné, definované v knihovně, jinou proměnnou, definovanou uživatelem ve vlastním kódu.

3.2 Požadované funkce

Aby knihovna mohla být dále využívána, je potřeba aby poskytovala funkce, které jsou běžně využívány v problematice vykreslování 3D modelů. Jednou ze základních funkcí by mělo být získání samotného modelu, který by se měl vykreslovat. Toto by se mělo zprostředkovat buď nahráním modelu ze souboru, nebo vytvářením základních geometrických obrazců jako je například kvádr, válec, obdélník, kruh a případně i čára.

Dalším z potřebných funkcí je přiřazení barev nebo textur jednotlivým modelům. Aby se na scénu nedívalo jenom z jednoho pohledu, je třeba vytvořit funkce, které by byly schopny manipulovat se samotnou scénou. Mezi takovéto manipulace patří otáčení s pohledem na scénu, nebo se samotným modelem. Pro prohlížení některých detailnějších částí by byla vhodná funkce zvětšení modelu. A pro prohlížení příliš velkých modelů je třeba funkce která by posouvala pohled na scénu. Ve scénách, která obsahují velké množství modelů by bylo vhodné, aby některé modely nebo skupiny modelů šly zneviditelnit.

V případě jednobarevných nebo monotónně zbarvených objektů může docházet k zanikání viditelnosti hran, pro dosažení lepší viditelnosti hran a tak i porozumění celkového tvaru objektu je potřeba alespoň základní osvětlení.

4 Řešení implementace knihovny

4.1 Volba způsobu vykreslení na webu

Úkolem je vykreslování modelů v HTML5. Již dříve bylo zprostředkováno mnoho možností jak vykreslovat 3D modely ve webovém prohlížeči, ale tyto možnosti byly zprostředkovány použitím dalšího programu, jako je například Java applet nebo Adobe Flash player.

HTML5 nyní zprostředkovává možnost vykreslovat 3D model přímo za použití prohlížeče, nejdříve umožňoval použití 2D canvasu pro vykreslování jednoduchých geometrických obrazců. Tato možnost by teoreticky mohla stačit, protože to co je zobrazeno na monitoru jako 3D obraz je ve skutečnosti několik 2D obrazců uspořádaných tak, aby dohromady složily prostorový obraz.

To je docíleno přepočítáním stěn třírozměrného objektu do dvourozměrné plochy, kdy při výpočtu dojde ke změně tvaru obrazce v závislosti na úhlu pohledu, a výpočtu, který obrazec se bude zobrazovat navrchu, který obrazec bude viditelný jen částečně a který obrazec bude zakryt úplně.

Při použití 2D canvasu by vyžadovalo abych napsal program pro přepočítání obrazců sám, tento program by zpracovával procesor a tím by se nedosáhlo plných možností počítače a to zejména použití grafické karty, která je k těmto výpočtům přizpůsobena. Naštěstí existuje knihovna zvaná WebGL, která řeší přepočítání obrazců a využívá přitom prostředků grafické karty.

Takže při výběru prostředí pro vykreslování 3D modelů na webu jsem zvolil použití WebGL, jednak protože je to nová technologie, ale i protože nevyžaduje instalaci přídavného modulu prohlížeče a využívá prostředků grafické karty.

4.2 Volba souborového formátu modelu

Svět počítačové grafiky má celou škálu různých souborových formátů pro uložení modelů. Možný výběr byl z formátů, které přímo ukládá jeden z výše popsaných grafických programů, ty však většinou bývají binárně uloženy a některé společnosti chtějí za poskytnutí knihovny pro čtení těchto souborů zaplatit. Proto bylo nutné vybírat některou z možností, která je zdarma. Další možností je využití souborových formátů, které jsou ukládány dostatečně jednoduše, aby šlo napsat vlastni funkce pro načítání formátu.

Nakonec jsem vybral souborový formát Wavefront object[12], používající soubory s příponou obj a mtl. Tento souborový formát je textový a ukládá jen základní informace o 3D geometrii modelu, čímž jsou souřadnice jednotlivých vrcholů, UV souřadnice textur, normály a stěny představující jednotlivé polygony, které jsou tvořeny seznamem vrcholů a UV souřadnic textur, které jej tvoří. Model je dále členěn na jednotlivé geometrie, které používají materiál uložený v souboru mtl, který je také textový a ukládá informace o barvě jednotlivých materiálů. Tento souborový formát lze navíc exportovat z mnoha programů pro vytváření 3D modelů.

4.3 Práce se souborem

4.3.1 Načtení souboru modelu

Modelový soubor formátu Wavefront Object je uložený textově a tak se nemusí řešit binární čtení, ani knihovnu pro překlad binárního kódu. Nejprve je nutné získat text souboru, který bude umístěn mezi ostatními soubory potřebnými pro chod webových stránek, pravděpodobně na webovém serveru. Tento text se bude ukládat do proměnné, se kterou se dále bude pracovat a získávat z ní potřebné informace.

Pro tuto potřebu jsem použil XMLHttpRequest, jedná se o rozhraní dostupné v Javascriptu. Toto rozhraní umožňuje prohlížeči poslat http nebo https žádost serveru. Server žádost zpracuje a pošle odpověď, která se předá skriptu. Tímto způsobem zle získat data ze souborů ve dodržující formát XML, nebo získat čistý text ze souboru. Oboje možnosti se spouští příslušnou metodou.

Já konkrétně využiji getResponseText pro získání textu souboru s modelem. XMLHttpRequest má bohužel nějaká bezpečnostní omezení a v některých prohlížečích nedovoluje načíst text ze souboru umístěného na lokálním disku. Toto nám vadí v případě, když webovou aplikaci nepouštíme z webu ale z disku.

Pokud na daném místě nebude soubor nalezen zachytí se chyba a zobrazí se varovná tabulka oznamující, že nelze najít daný soubor.

4.3.2 Získání potřebných dat z obj souboru

V Wavefront Object souboru je na začátku každého řádku, který nás zajímá, znak nebo skupina znaků, které určují typ potřebné informace. Symbol # určuje poznámku, ta pro vykreslení modelu není nijak důležitá. Na začátku souboru bývá informace o materiálové knihovně, ve které jsou uloženy informace o materiálech, jako jsou

například barvy, odrazivost nebo průhlednost. Řádek je označen řetězcem mtllib a za ním následuje název souboru.

```
# Blender v2.62 (sub 0) OBJ File: ''
# www.blender.org
mtllib krychle.mtl
o Cube_Cube.001
v 1.000000 0.000000 -1.000000
v 1.000000 0.000000 -0.000000
v 0.000000 0.000000 -0.000000
v 0.000000 0.000000 -1.000000
v 1.000000 1.000000 -1.000000
v 1.000000 1.000000 0.000000
v 0.000000 1.000000 -0.000000
v 0.000000 1.000000 -1.000000
usemtl
s off
f 1 2 3 4
f 5 8 7 6
f 1 5 6 2
f 2 6 7 3
f 3 7 8 4
f 5 1 4 8
```

Ilustrace 1: Obj soubor popisující krychli

Model je členěn na jednotlivé objekty modelu, nebo na skupiny polygonů. Na začátku řádku je písmeno o, případně g následováno názvem. Následující řádky obsahující geometrické informace patří do příslušné skupiny. Ke každé skupině bývá přiřazen materiál z knihovny materiálů. Tato informace je označena řetězcem usemtl následována jménem materiálu, tak jak je uložen v materiálové knihovně.

Každá skupina obsahuje i geometrické informace. Vrchol je značen písmenem V, za kterým následují souřadnice X, Y a Z daného vrcholu.

Souřadnice textury je značena písmeny VT následovány souřadnicemi textury U a V. Tyto souřadnice vyjadřují bod na textuře, který bude odpovídat bodu reprezentující vrchol polygonu. Při přidělení UV souřadnic textury ke všem vrcholům polygonu můžeme určit, která část textury bude nanesena na polygon.

Normálový vektor je značen písmeny VN, za kterými následují rozměry vektoru X, Y a Z. Normálový vektor slouží k určení přední a zadní strany stěny. Této informace využíváme zejména při osvětlování.

Polygon je označen písmenem F, za kterým následují informace o tom, ze kterých vrcholů je sestaven polygon. Často jsou u vrcholů informace o příslušných UV

souřadnicích textury a normálové vektory. Tyto informace však nejsou povinné a v některých případech nejsou potřebné.

Polygony jsou zapisovány čtyřmi způsoby:

f v(1) v(2) v(3) v(4)… – určeny jsou jenom vrcholy, ze kterých se polygon skládá

f v(1)/vt(1) v(2)/vt(2) v(3)/vt(3)… – určeny jsou vrcholy a souřadnice textury

f v(1)/vt(1)/vn(1) v(2)/vt(2)/vn(3) v(3)/vt(3)/vn(3)…

– určeny jsou vrcholy, souřadnice textury a normála

f v(1)//vn(1) v(2)//vn(3) v(3)//vn(3)… – určeny jsou vrcholy a normála

Číslo v(i) určuje pořadí vrcholu jak je zapsán v souboru, číslo vt(i) určuje pořadí souřadnice textury jak je zapsána v souboru a číslo vn(i) určuje pořadí normály jak je zapsána v souboru. První zápis vrcholu má číslo jedna, druhý zápis vrcholu má číslo dva,…

Před začátkem zpracování textu se připraví pole, které reprezentuje jednotlivé skupiny. Toto pole je sestaveno z objektů, které obsahují další 3 pole, která se naplní informacemi o vrcholech, normálách a souřadnicích textur. Dále obsahuje proměnné do kterých se v případě potřeby uloží název použitého materiálu, adresa textury a samotný soubor s texturou.

Samotné zpracování textu se začne tím, že se text rozdělí na jednotlivé řádky. Potom se každý řádek rozdělí na jednotlivé řetězce podle znaku „mezera“, z prvního řetězce se zjistí, zda-li řádek obsahuje některou z požadovaných informací a o který typ informace se jedná.

Na začátku souboru bývá informace o knihovně materiálů, její název se uloží pro pozdější získávání informací o materiálu. Při nalezení definice nové skupiny polygonů se v poli ukládající skupiny vytvoří nový objekt, který bude uchovávat další informace. Pokud je zjištěno, že řádek obsahuje informace o vrcholu, tak se vrchol s příslušnými souřadnicemi přidá do pomocného pole obsahující vrcholy, podobě se uloží informace o souřadnicích textur, normál nebo materiálu.

V případě, že řádek obsahuje informace o polygonu tak se nejprve musí zjistit způsob zápisu, ten se získá, když se druhý řetězec řádku rozdělí podle znaku „lomítko“. Pokud řetězec nepůjde rozdělit, je zanesena informace pouze o vrcholech. Pokud se řetězec rozdělí na dvě části, tak jsou zaneseny informace o vrcholech a souřadnicích textury. Pokud se řetězec rozdělí na tři části, tak existují dvě možnosti. Když bude druhá

část řetězce prázdná, tak jsou zaneseny informace o vrcholu a normále. Pokud bude v druhé části řetězce číslo, tak jsou zaneseny informace o vrcholech, souřadnicích textur a normále.

Dále je nutné zjistit z kolika vrcholů se polygon skládá, to zjistíme spočítáním řetězců za písmenem F. V případě tří řetězců se jedná o trojúhelník, ten WebGL vykreslit umí. Pokud se jedná o čtverec je nutné jej rozdělit na dva trojúhelníky, první trojúhelník se skládá z prvních třech vrcholů, druhý se skládá z prvního, třetího a čtvrtého vrcholu. Podle jednotlivých čísel vrcholů, souřadnic textury a normál se vytvoří seřazená pole vrcholů, souřadnic textur a normál.

Pole hodnot pro WebGL musí být uspořádána tak, že se za sebe napíšou souřadnice vrcholů/souřadnic normál/textur. Řekněme-li, že x(i, j) y(i, j) z(i, j) jsou souřadnice vrcholu, přičemž i je číslo polygonu a j je číslo vrcholu v polygonu, pole pak vypadá následovně: x(1,1), y(1,1), z(1,1), x(2,1), y(2,1), z(2,1), x(3,1), y(3,1), z(3,1), x(1,2), y(1,2), z(1,2)…

4.3.3 Získání potřebných dat z mtl souboru

Mtl soubor se bude zpracovávat podobně jako soubor obj. V tomto souboru se nacházejí informace o materiálech. Název materiálu je definován řetězcem newmtl na začátku řádku. Na dalších řádcích následují informace o barvách, které jsou uloženy ve formátu RGB.

Ka označuje barvu osvětlení prostředí (ambient), Kd označuje difuzní barvu a Ks s Ns označuje informace k zrcadlové (specular) barvě. D nebo Tr označuje průhlednost. Na začátku se vytvoří pole materiálů, do kterého se bude ukládat jeho jméno a jednotlivé barvy.

4.4 Vytváření grafických primitiv

Nejdříve jsem uvažoval o vložení předem připravených modelů, u kterých bych později upravoval velikost. Potom jsem se rozhodl, že udělám funkce, které vypočítají pozice vrcholů v závislosti na zadaných hodnotách.

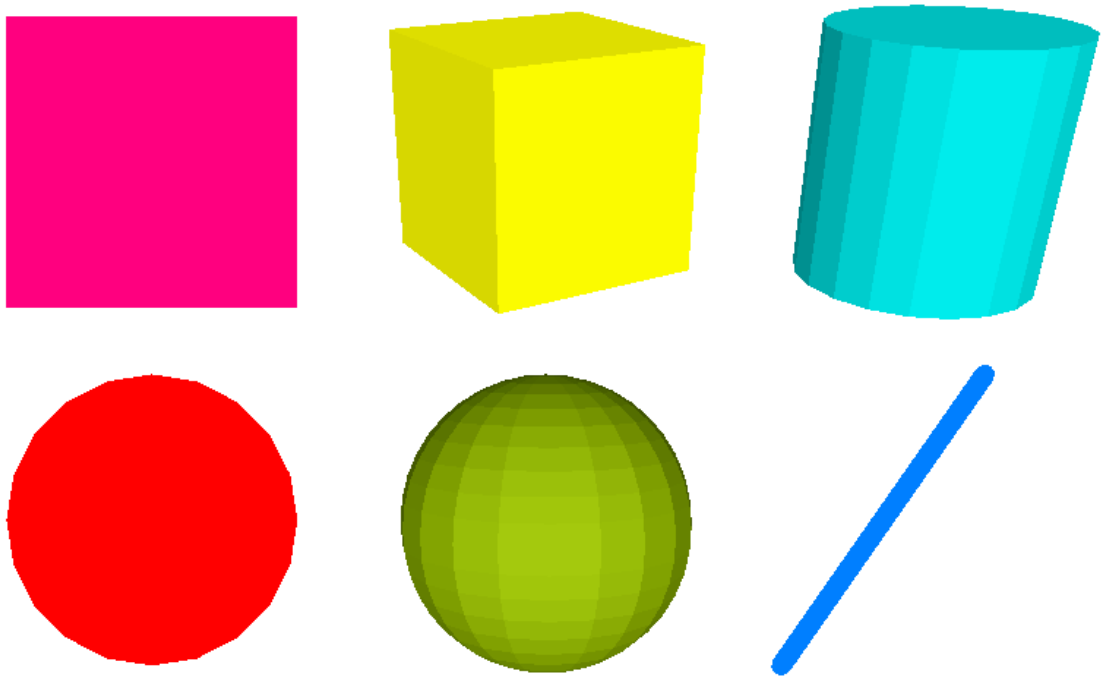
U čtyřúhelníku a trojúhelníku se zadají souřadnice vrcholů. U kruhu se zadá poloměr a počet segmentů kruhu (ve webgl nelze vytvořit přesný kruh, tak se vytváří jako pravidelný n-úhelník). Kruh se rozdělí na jednotlivé rovnoramenné trojúhelníky. Délka ramen trojúhelníku odpovídá poloměru kružnice a úhel, který svírají je 360/počet

segmentů. Pomocí goniometrických funkcí se vypočítají jednotlivé vrcholy. Jedná se tedy o n-úhelník kružnici vepsaný. Toto zajišťuje, že vzniklé modely nebudou zasahovat do jiných modelů.

U hranolu se zadají rozměry a později se provede posun, podle rozměrů se vypočítají pozice vrcholů. Válec je podobný jako kruh, navíc k poloměru a segmentaci se zadá i výška. Obě podstavy se vytvoří stejně jako kružnice a mezi jednotlivými segmenty kružnic se vytvoří obdélníky.

U koule se zadává poloměr a segmentace, aby póly koule nevypadali zvláštně, tak by měla být segmentace dělitelná dvěma. Ta je o něco složitější, ale není to nic jiného než soustava válců bez podstav s různými průměry podstav. Výška jednotlivých válců se určí podle rozdělení kružnice.

U čáry se zadává výchozí bod, koncový bod a tloušťka. V podstatě se jedná o obdélník s šířkou odpovídající tloušťce čáry a půlkruhy na každém konci. Podle úhlu, který čára svírá s hlavní osou se vypočítá pozice rohů obdélníku. Ze začátku jsem ukládal každou čáru jako samostatný model, ale při větším množství čar dochází k zpomalení prohlížení. Tak jsem přidal možnost na zřetězení několika čar do jednoho objektu. Tyto čáry budou mít společnou barvu.



Ilustrace 2: Ukázka některých grafických primitiv

4.5 Uchovávání modelů

Jednotlivé modely se ukládají v poli. Jednotlivé objekty tohoto pole obsahují několik informací a funkcí. Mezi tyto informace patří pole s jednotlivými geometriemi modelu, jejich barva popřípadě textura, informace jestli je textura použita.

Při vytváření modelu se nejdříve vytvoří objekt a potom se u něj některou z funkcí vytvoří obsah, buď z modelu nebo z primitiv. Když se vytváří model ze souboru (získání informace je popsáno výše), tak se za pomoci názvů materiálu z obj souboru a materiálu z mtl souboru přidělí jednotlivým geometriím barva.

Další z informací a funkcí jsou určení a získání x, y, z souřadnice vložení daného modelu, otočení po zmíněných osách, informace o tom jestli se má model zobrazovat, nebo ne a jaký se má použít filtr na textury.

Další funkce, o kterých bych se chtěl zmínit, ale nejsou to přímo metody daného objektu obsahující model, jsou funkce manipulující s naposledy vloženým modelem. Mezi ně patří, změna bodu vložení, otočení, změna barvy jednotlivých geometrií modelu, změna barvy celého modelu a přidělení textury jednotlivým geometriím modelu.

4.6 Shader programy

4.6.1 Vertex Shader

Je to program, který se provádí na straně grafické karty. Vertex Shader se provede u každého vrcholu a provede s ním určité věci. Na začátku kódu se definují proměnné různých typů (atributy, uniformy a proměnné), tady se upravují pozice vrcholů podle matice perspektivy a matice transformace.

Dále se zde vyhodnotí jestli se používá osvětlení nebo ne, tuto informaci mu předáme při vykreslování, které vysvětlím později. Pokud osvětlení není použito tak se předá jednotkový koeficient změny barev (barvy se nezmění). Pokud osvětlení použito je tak vypočítá skalár mezi transformovanou normálou a převráceným směrem osvětlení. Podle tohoto úhlu určí koeficient změny barvy. Vzniklé informace předá Fragment Shaderu.

4.6.2 Fragment Shader

Fragment Shader je program, který určuje jak bude vypadat každý bod představující plochu polygonu. Na začátku kódu se definuje přesnost desetinných míst reálných čísel, se kterými se pracuje. Poté se definují proměnné. V této části se vyhodnotí, jestli se používají textury nebo ne, tuto informaci programu předáme při vykreslování.

Pokud se textury používají získají fragmenty zabarvení podle rozložení textury, toto zabarvení se dále upraví podle koeficientu vypočítaném ve Vertex Shaderu. Pokud textura není použita, tak se použije barva, která je předána shaderu při vykreslování, tato barva se opět upraví podle výše zmíněného koeficientu.

4.7 Inicializace WebGL prostředí

4.7.1 Získání informací o canvasu

Než se začne vykreslovat model, tak se nejdříve musí určit, ve kterém prvku stránky se bude daný model vykreslovat. Použitelný prvek stránky je canvas a přistoupíme k němu pomocí DOM, konkrétně pomocí id elementu canvas.

Když je určen canvas, tak se zavolá jeho WebGL obsah, tomu dále budeme dávat příkazy k vykreslování. Pokud není kontext možné získat prohlížeč pravděpodobně nepodporuje WebGL, model nebude možné zobrazit a zobrazí se upozornění, že nelze inicializovat WebGL. V takovémto případě musí uživatel zkontrolovat jestli webový prohlížeč podporuje WebGL, případně zkontrolovat v nastavení prohlížeče jestli je WebGL povoleno.

4.7.2 Inicializace shaderů

Nejdříve jsem se dozvěděl, že se shaderovací programy dají uchovat jako skript v html souboru, ale to bylo trochu nepraktické. Uživatel knihovny by pak musel nakopírovat shader programy do html kodu.

Nakonec tyto programy uchovávám v textovém souboru, který načítám pomocí výše popsané funkce používající XMLhttpRequest. Kód programu se předá WebGL ke kompilaci. Potom když jsou zkompilované se připojí k WebGL a označí se, že se mají používat. Poté se musí WebGL oznámit, co používají shadery za proměnné abychom je mohli později předat.

4.8 Manipulace s modelem

Protože by nebylo dobré, aby se model vykreslil jenom v jednom pohledu a nedalo by se jím otáčet, tak jsou vytvořeny funkce pro manipulaci s modelem. S modelem bychom měli manipulovat tak, že nezměníme přímo jeho hodnoty jak jsou uloženy v polích.

Takováto manipulace s více objekty by byla velice složitá. Namísto toho Vertex Shaderu poskytneme matici, ve které jsou uloženy všechny manipulace s objektem, této matici říkáme transformační matice. Pro práci s maticemi využívám knihovnu glMatrix-.9.5.min[14], která řeší různé maticové a vektorové funkce, jako jsou například maticové násobení, sčítání matic, transponování matic, ale i vytvoření matic posunu, rotace a zvětšení.

S modelem lze manipulovat buď absolutně, nebo relativně. Absolutní manipulace znamená, že aktuální manipulace s objektem se provede na původních hodnotách objektu, nezáleží kolikrát bylo modelem otáčeno, o kolik stupňů byl otočen, když zadáme, aby byl model otočený o 90° od původní pozice, tak se otočí o 90° od původní pozice.

Relativní manipulace si zase naopak pamatuje všechny předešlé korky a když je potřeba modelem plynule otáčet o jeden stupeň s rozestupem nějakého časového úseku, tak se nemusí zadávat, aby se první vteřinu model otočil na první stupeň, druhou vteřinu na druhý stupeň, ale zadá se mu, že každou vteřinou se má otočit o jeden stupeň.

4.8.1 Funkce pro manipulaci

Jelikož WebGL přijímá pouze absolutní změnu modelu, tak se aktuální posunutí a zvětšení ukládá do proměnných. Pro absolutní manipulaci se tyto proměnné přepíšou, když je potřeba relativní manipulace tak se k těmto proměnným přičte nová hodnota posunu, nebo v případě zvětšení se proměnná vynásobí novým násobkem.

Posun a zvětšení je o něco jednodušší zpracovaní než otočení. Jednoduše se změní čísla kolikrát se model má v daných osách zvětšit, nebo o kolik se má v daných osách posunout.

Otáčení modelem by bylo stejně jednoduché, ale v tom případě by se model otáčel poněkud zvláštně. Kdyby se to dělalo tímto způsobem, tak při změně otočení kolem os X a Y by se měnily úhly otočení podle zkutečných X, Y os zobrazovaného modelu. Výsledný efekt by byl ten, že po otočení modelu o 90° podle osy X, by se při otáčení

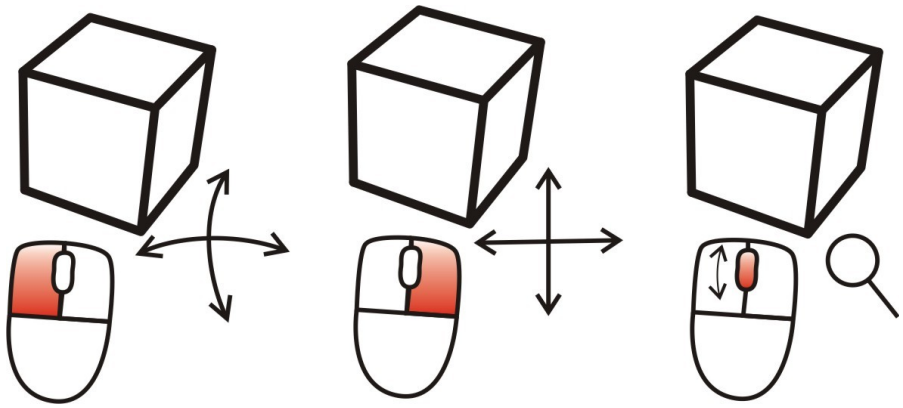
podle osy Y ve skutečnosti model otáčel podle Z osy z pohledu na model, protože při otáčení podle osy X se osa Y také otočila.

Aby se model otáčel podle aktuálního pohledu, je zajištěno tím, že aktuální otočení je zapsáno do matice, když se provede nové otočení tak se vytvoří matice s informacemi o tomto novém otočení. Matice s novým otočením se vynásobí maticí původní pozice a tím se získá nová matice aktuální pozice. Při absolutním otáčení, se matice původní pozice vynuluje, respektive se z ní udělá jednotková matice, a povede se výše popsané násobení matic.

Manipulaci s modelem umožňuji dvěma způsoby. Výše popsané funkce se mohou s nějakou konstantní velikostí přidělit tlačítkům uživatelského rozhraní (při kliknutí proveď funkci), případně se mohou tyto funkce volat v průběhu uživatelem definovaného kódu, chce-li zajistit nějaké fixní otáčení v průběhu času.

4.8.2 Manipulace za pomoci myši

Další možností manipulace s modelem, je za pomocí myši. Při stisknutí tlačítka myši v prostoru canvasu se zaznamená do proměnné, že je stisknuté tlačítko myši. Do proměnných uložíme souřadnice X, Y a uchováme informaci o tom, které tlačítko bylo stisknuté. Při uvolnění tlačítka myši kdekoliv na obrazovce se do proměnné zaznamená, že tlačítko už není nadále stisknuto.



Ilustrace 3: Ovládání pomocí myši

Při každém pohybu myši se zkontroluje zda je stisknuté tlačítko myši. Pokud ano tak se ještě zkontroluje, které tlačítko bylo stisknuto. Bylo-li stisknuto levé tlačítko myši, vypočítá se rozdíl mezi souřadnicemi stávající pozicí a předešlé pozice, tyto rozdíly se upraví a použijí jako úhel otočení v příslušném směru. Nakonec se předchozí pozice kurzoru zaznamená současná pozice kurzoru.

Bylo-li stisknuto pravé tlačítko, provede se něco podobného, ale namísto funkce pro otáčení, se použije funkce pro posun. Poslední ovládání za pomocí myši je zvětšování pomocí kolečka. Kolečko myši je vlastně krokový senzor, při každém pootočení se zaznamená počet kroků a směr je určen znaménkem. Tuto informaci o počtu kroků upravíme v závislosti jakým způsobem jsme jej získali (některé prohlížeče zaznamenávají deltu s jinou hodnotou kroku). Upravenou informaci předáme jako vstup funkci pro zvětšování.

4.9 Textury

4.9.1 Texturové filtry

V případě, že budeme chtít použít texturu na model, musíme ji nějakým způsobem zpracovat. Textura je ve skutečnosti soubor s obrazem. Málokdy odpovídají rozměry části obrazu nanášeného na polygon rozměrům polygonu, proto musíme použít filtrování textur. Možnosti jsou filtrování metodou nejbližšího souseda, bilineární filtrování a trilineární filtrace s mipmapami, existují i další možnosti, ale jejich implementace ve WebGL by nemusela být tak jednoduše uskutečnitelná.

Ilustrace 4: Filtrování metodou nejbližšího souseda při zvětšení

Filtrování metodou nejbližšího souseda je z nich nejjednodušší. Funguje tak, když se objekt bude zvětšovat, tak barva chybějících bodů textury se doplní jako barva nejbližšího souseda. Při zmenšování modelu se body z plochy textury rovnoměrně odstraňují. Textura bude vypadat dobře, když bude mít polygon odpovídající rozměr, a nebude vypadat špatně když bude textura větší než její polygon. Nicméně když bude textura menší než její polygon tak dojde k roztažení obrázků, textura bude vypadat moc čtverečkovaně.

Bilineární filtrování je trochu složitější, ale má lepší výsledky při zvětšování modelu. Při roztažení obrázku se mezery mezi pixely vypočítají jako lineární přechod mezi pixely originální textury, jednoduše řečeno pixel, který je napůl cesty mezi černým pixelem a bílím pixelem dostane šedivou barvu. Tímto dojde při zvětšování modelu k hladšímu roztažení a nebude to kostičkované, nicméně ostré hrany se poněkud rozmažou. Ve skutečnosti, když zvětšujete obraz nikdy nemůžete dostat perfektní výsledek – nemůžete získat detaily, které tam prostě nejsou.

Ilustrace 5: Bilineární filtrování při zvětšení

Bilineární filtrování dává srozumitelné výsledky, když se model zvětšuje, ale když se zmenšuje tak není o nic lepší než nejbližší filtrování oba filtry mohou způsobovat ošklivý Aliasing. Jedná se o ztrátu dat, když se vezme například textura prken poskládaných vedle sebe, při příliš velkém zmenšení může dojít k tomu, že zmizí černá mezera mezi prkny, v dalším kroku by se opět mohla objevit.

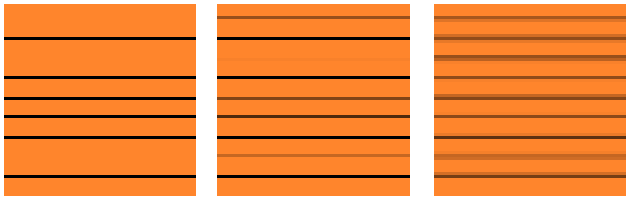
K tomu dochází z určitého důvodu, když se textura zmenší například 10x tak se jako barvu pro polygon použije každý desátý pixel. Aby to vypadalo dobře, musela by být místo barvy každého desátého pixelu, průměrná barva všech deseti pixelů. To je ovšem velmi náročné pro grafiku vykreslovanou v reálném čase a proto se používá Mipmap.

Mipmapy řeší tento problém vytvořením několika obrázků (zvaných mip level), poloviční, čtvrtinové, osminové, atd. velikosti. Každá úroveň mipmapy je zmenšená verze originálního obrázku, přepočítaná pomocí průměrných hodnot pixelů. Přepočet se provede jen jednou na počátku programu, proto už při samotném vykreslování

nezdržuje. Takže, když se obraz zmenšuje tak vybíráme nejbližší mip level a na tom provedeme bilineární filtrování.

Pro další zvýšení kvality se používá trilineární filtrace, kde se mezi hodnotami nejbližších mipmap lineárně interpoluje – trilineární potom znamená lineárně v X, Y a mezi mipmapami.

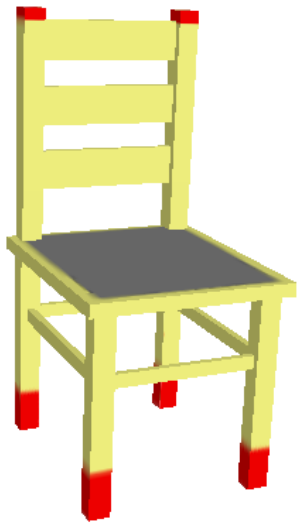
Na níže uvedeném obrázku (Ilustrace 6) je vidět rozdíl při zmenšení mezi filtrováním metodou nejbližšího souseda (vlevo), bilineárním filtrováním (uprostřed) a trilineárním filtrováním (vpravo).



Ilustrace 6: Texturové filtry při zmenšování

4.9.2 Načtení textury

WebGL prostředí má vlastní místo pro textury. Vytvoří se proměnná, do které se vloží textura jako objekt knihovny WebGL. Umístění této proměnné je ve výše zmiňovaném objektu s geometrií. Vytvoří se Javascriptový obrázkový objekt a přidá se jako parametr k textuře. Teď by se měl určit zdroj obrázku, ale nejdřív se přidá funkce, která po dokončení nahrání obrázku, předá obraz jako skutečnou texturu.



Ilustrace 7: Model židle používající vlastní texturu

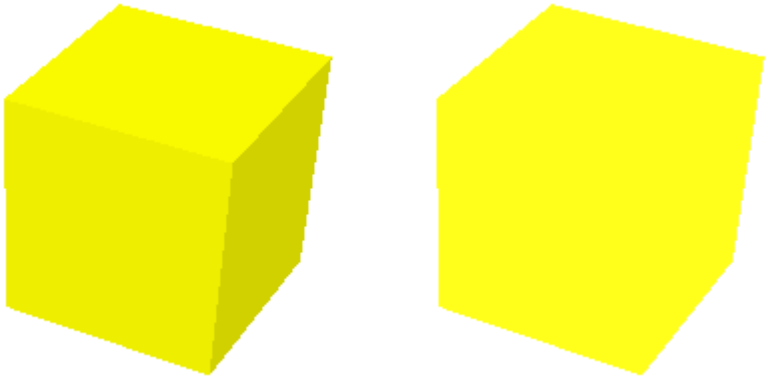
Nejdřív se WebGL musí předat, se kterou texturou se právě bude pracovat. Potom se WebGL oznámí, že je potřeba obrázek vertikálně otočit. Tohle se musí provést protože souřadný sytém textury je stejný jako ten matematický, v levém dolním rohu je počátek a směrem nahoru a doprava stoupají osy X a Y.

Na druhou stranu většina obrázků je uložena s opačným souřadnicovým systémem, kde se začíná v levém horním rohu a hodnoty na osách nabývají směrem dolů a doprava. Dalším krokem je přidělení právě nahraného obrázku do paměti grafické karty. Když je obrázek nahraný je přidělen ještě způsob filtrování. Tímto způsobem je textura připravená pro pozdější použití a může se uvolnit.

4.10 Osvětlení

WebGL samo o sobě metody pro osvětlení nemá, nicméně není až tak složité je implementovat. Mluvíme zde pouze o základním osvětlení, jednak osvětlení prostředí, které osvětluje všechny polygony stejně. A o základním směrovém osvětlení, které způsobuje, že polygony směřující přímo ke zdroji světla jsou světlejší a ty polygony, které se od zdroje světla odklánějí jsou tmavší. Toto závisí na úhlu, který se směrem světla svírají. K vypočítání osvětlení je potřeba normálový vektor, jenž určuje, kterým směrem je polygon namířen.

Nejdříve se zkontroluje jestli je světlo zapnuto a tato informace se předá shaderům jestli mají při výpočtu barvy počítat se světlem. Potom se načtou jednotlivé složky barvy světla prostředí. Poté se načte směr světla. Před posláním směru světla shaderům jej musíme nejdříve normalizovat, zajistit, aby jeho délka byla hodnota jedna. A poté se směr světla obrátit, výchozí směr světla ukazuje kam světlo míří, ale k výpočtu je potřeba směr odkud světlo přichází. Nyní se směr světla pošle shaderům. Poté se nahraje a pošle barva směrového světla.



Ilustrace 8: Krychle s osvětlením a bez osvětlení

K získání konečné barvy ($RGB_{finál}$) vynásobíme původní barvu(RGB) skalárem normálového vektoru (n) s převráceným vektorem směru světla (v), v případě záporného skaláru vynásobíme původní barvu nulou. $RGB_{final} = RGB \cdot max((n \cdot v), 0)$

Další věc, která se musí ošetřit, aby se při otáčení, nebo při pohybu modelu správně přepočítaly normálové vektory, ty se nemůžou přepočítávat stejně jako vrcholy.

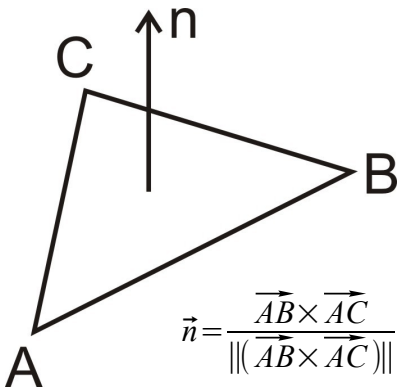
4.11 Vykreslování modelu

4.11.1 Načtení modelu do zásobníku

Než se může začít přímo vykreslovat model, musíme nahrát data prostředku, který je bude vykreslovat, v tomto případě do paměti grafické karty. K vytváření modelů potřebujeme čtyři informace. Pozice vrcholů, souřadnice textur, normály a pořadí vrcholů.

Scéna se skládá z několika modelů a každý model se skládá z několika geometrií. Abychom jim mohli přidělit jiné barvy musíme geometrie načítat každou zvlášť. Připravíme si pole bufferů, do kterých budeme zaznamenávat referenci na buffery jednotlivých geometrií, ještě si musíme zaznamenat z kolika čísel se skládá jeden předmět (vrchol, normála,…) a kolik předmětů v daném poli je.

Chybí-li informace o normálách, vypočítáme normály sami. Není to nic těžkého, v obj souboru jsou vrcholy polygonů zapisovány v protisměru hodinových ručiček. Normála je vlastně jednotkový vektor kolmý na plochu. Plochu máme určenou dvěma vektory (strany AB a AC trojúhelníka ABC) a vektor na ně kolmý získáme vektorovým násobením. Jednotlivé složky vektoru pak vydělíme jeho délkou a tak získáme normalizovaný vektor. Takto vypočítáme normálový vektor pro každý trojúhelník a připíšeme ho ke každému vrcholu.



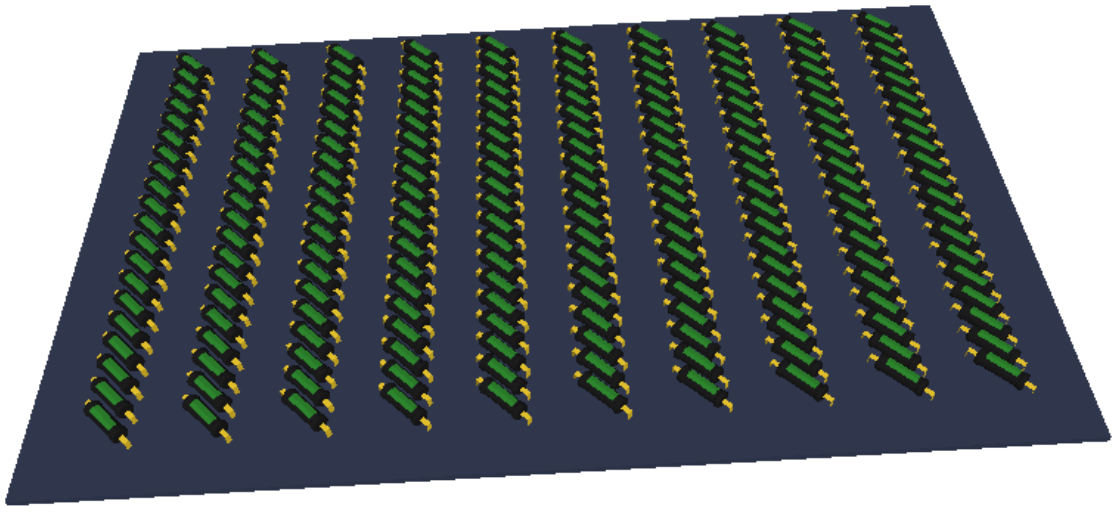
Ilustrace 9: Normálový vektor

4.11.2 Vykreslení modelů

Na začátku funkce, ve které se kreslí scéna, se nejprve vloží výška a šířka zorného pohledu, které získáme z rozměrů canvasu. Je to nutné proto, aby se scéna správně vystředila do canvasu. Poté se musí canvas vyčistit. Nebylo by dobré, kdyby po pootočení modelu zůstal v canvasu model vykreslen v původní poloze, překryt modelem v nové poloze. Na začátku vykreslování scény se také určí matice perspektivy, ve které se bude vykreslený model zobrazovat.

Vyčistí se matice pohybu modelu a poté se na ní aplikují posun, rotace a zvětšení modelu. Tady záleží na pořadí ve kterém akce provádíme. Před začátkem vykreslování modelu zjistíme, jestli daný model chceme vykreslit. U každého modelu získáme informaci o jeho bodu vložení a natočení a aplikujeme tyto informace na transformační matici. Poté projedeme každou geometrii modelu a WebGL oznámíme, že chceme pro vykreslování použít jejich příslušné buffery.

Pokud model má texturu aktivujeme ji a oznámíme shaderu, že ji má použít jako zdroj barvy. Poté provedeme výše popsané akce potřebné k osvětlení scény. A potřebné informace předáme shaderům. Když se vykreslí všechny geometrie modelu, tak musíme vrátit změny, které jsme provedli na transformační matici pro změnu pozice jednotlivých modelů.



Ilustrace 10: 200 modelů v jedné scéně

Aby se nám scéna pravidelně aktualizovala, tak funkci pro vykreslení musíme pouštět v pravidelném intervalu. Pro spouštění nějaké funkce v intervalu existuje v JavaScriptu funkce, ale ta se spouští i když si danou stránku neprohlížíme. Takže by se

počítač zbytečně zatěžoval, i když by jsme si model aktuálně neprohlíželi. Pro tento případ jsem využil funkce z knihovny webgl-utils[15]. Funkce RequestAnimFrame umožňuje aktualizovat scénu tak často jak to počítač dovolí, ale jenom v případě, že je stránka s canvasem zrovna aktivní.

5 Zhodnocení

5.1 Použití knihovny

Pro použití na webových stránkách se na web zkopíruje složka ObjToWebGL a všechny soubory, které obsahuje. Dále se musí do HTML souboru vložit odkaz na skript webGL-links.js v dané složce, který propojí všechny ostatní komponenty knihovny. Nyní je možné ve vlastním skriptu používat funkce knihovny.

Nejdříve se musí knihovně přiřadit canvas. Poté se nastaví barva pozadí canvasu a spustí se inicializace součástí knihovny. Poté se může zapnout použití světla a nastavit jeho hodnoty. Vytvoří se upřesňující vlastnosti pohledu na scénu jako jsou posunutí modelu a úprava jeho velikosti. Vyčistí se paměť modelů a nahrají se nové modely i s texturami. U jednotlivých modelů lze určit jejich bod vložení a otočení. Nakonec se knihovně řekne, že může začít vykreslovat.

5.2 Co knihovna umí a co neumí

Knihovna umí vykreslovat modely uložené ve formátu obj, většina testovacích modelů byla vytvořena v Blenderu, ale modely vytvořené v jiných prostředí pro vytváření 3D modelů by také měly fungovat. Každý model může mít několik textur nebo barev, v závislosti jak byl vytvořen.

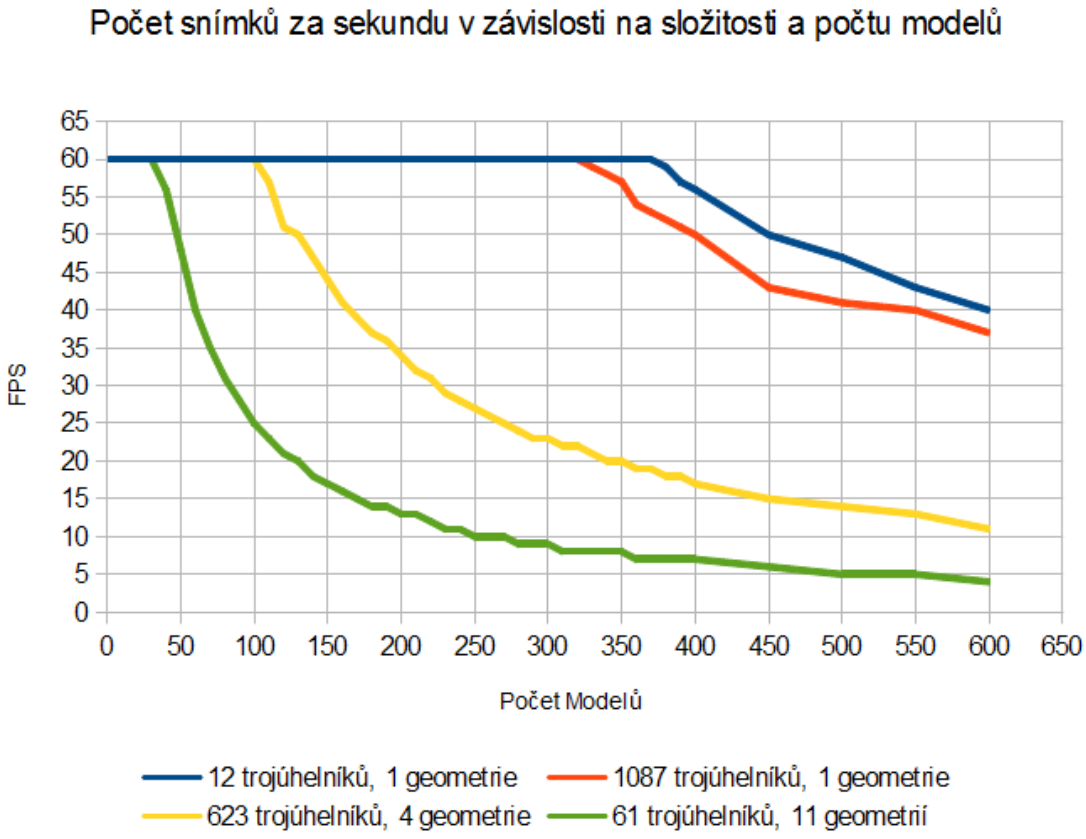
U každého modelu v sestavě se dá určit bod vložení a jeho natočení. Celá scéna se dá natáčet podle os X, Y, Z, posouvat ve směru os X, Y, Z a zvětšovat, buď celkově nebo jenom v určeném směru. Otáčení, posouvání nebo zvětšování se provádí buď za pomocí funkcí pro natočení modelu, nebo za pomocí myši.

Knihovna zvládá vykreslovat desítky až stovky modelů, přičemž se s modelem manipulovat při dostatečném (>24) počtu snímků za sekundu. Pro větší počet modelů by se musel hlouběji optimalizovat kód.

Scéna může být nasvícena jedním směrovým světlem. Umožňuje skrývání jednotlivých modelů. Knihovna neumí složitější osvětlení a efekty, jako je například bodové světlo, nebo Phongovo stínování.

5.3 Test výkonu

Provedl jsem menší testování výkonu knihovny. Testování jsem provedl na svém osobním počítači. Počítač měl následující parametry: dvoujádrový Procesor Pentium Dual-Core s frekvencí 2,5 GHz na každé jádro, 3 Gigabajty RAM, Grafická karta NVIDIA GeForce 9500 GT s 512 Megabajty paměti. Na počítači byly nainstalovány Windows Vista. Bylo nastaveno rozlišení 1920x1080 za obnovovací frekvence 60Hz. Test byl proveden v prohlížeči Mozilla Firefox 12.0.



Graf 1: Počet snímků za sekundu v závislosti na složitosti a počtu modelů

Použil jsem čtyři testovací modely, které jsem vybral z dostupných modelů. Postupně jsem zvyšoval množství modelů po 10. Po dosažení 400 modelů jsem zvyšoval množství po 50 až do konečného množství 600 modelů. Nejdříve jsem použil velice jednoduchý model, krychli, která se skládala z 1 geometrie obsahující 12 trojúhelníků. Do počtu 370 modelů si prohlížeč udržoval přibližně 60 snímků za sekundu, což odpovídá obnovovací frekvenci použitého monitoru. Poté začala klesat.

Jako další testovací model jsem použil model židle, která se skládala z 1 geometrie obsahující 1087 trojúhelníků. Počet snímků za sekundu začal klesat o 40

modelů dřív. Dalším testovacím modelem byl rezistor, který se skládal ze 4 geometrií obsahujících 623 trojúhelníků. Zde začal klesat počet snímků o dost dříve a to při počtu 100 modelů. Jako poslední testovací model jsem použil jiný model židle, který se skládal z 11 geometrií obsahujících 61 trojúhelníků. Počet snímků za sekundu se začal snižovat po pouhých 30 modelech. (viz. Graf 1)

6 Závěr

Vytvořil jsem softwarovou knihovnu pro vykreslování 3D modelů na webových stránkách. Knihovna je napsaná v Javascriptu a využívá grafické knihovny WebGL. Model se vykresluje přímo v prohlížeči a nepotřebuje instalaci přídavného modulu. Scéna se vykresluje na místě prvku typu canvas, jenž je součástí HTML5.

Prohlížení modelů je umožněno pomocí prvků na webové stránce nebo použitím myši. Knihovna umožňuje základní prohlížení modelu, jako je například otáčení, nebo přiblížení pohledu na scénu. Do jedné scény je umožněno vložení několika modelů. Scéna může být volitelně osvícena směrovým světlem zvolené barvy. Každý model může používat několik barev nebo textur. Knihovna zajišťuje plynulý provoz při zobrazování desítek až stovek modelů.

Ke knihovně je nad rámec zadání přiložena ukázka vykreslení modelu na webu. Cílem této stránky není příjemné prostředí pro uživatele, ale ukázka funkčnosti knihovny.

Knihovnu by dále bylo možné rozšířit například o Phongovo stínování, vrhání stínu modelem a dalšími typy zdrojů světla. Dále by se mohl hlouběji optimalizovat kód pro zvýšení počtu snímků za sekundu ve scéně s větším počtem modelů.

Seznam použité literatury

[1] *Learning WebGL* [online]. 2009 [cit. 2012-05-11]. Dostupné z: <http://learningwebgl.com>

[2] HTML5 differences from HTML4: W3C Working Draft 29 March 2012. [online]. [cit. 2012-04-17]. Dostupné z: <http://www.w3.org/TR/html5-diff/>

[3] GOSLING, James, Bill JOY, Guy STEELE, Gilad BRACHA. *The Java™ Language Specification*: *Java SE 7 Edition* [online]. 2011 [cit. 2012-05-09]. Dostupné z: <http://docs.oracle.com/javase/specs/jls/se7/html/index.html>

[4] Adobe Flash. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-04-17]. Dostupné z: <http://en.wikipedia.org/wiki/Adobe_Flash>

[5] *Microsoft Silverlight* [online]. [cit. 2012-05-12]. Dostupné z: <http://www.microsoft.com/silverlight/>

[6] O3D. In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-04-17]. Dostupné z: <http://en.wikipedia.org/wiki/O3D>

[7] *WebGL public wiki* [online]. 2009- [cit. 2012-04-17]. Dostupné z: <http://www.khronos.org/webgl/wiki/Main_Page>

[8] VRML97 and Related Specifications. *Web3D Consortium* [online]. © 1999-2011 [cit. 2012-05-11]. Dostupné z: <http://www.web3d.org/x3d/specifications/vrml/>

[9] What is X3D?. *Web3D Consortium* [online]. © 1999-2011 [cit. 2012-04-17]. Dostupné z: <http://www.web3d.org/realtime-3d/x3d/what-x3d/>

[10] Object Files. *The Graphics File Format Page* [online]. 1994 [cit. 2012-05-11]. Dostupné z: <http://www.martinreddy.net/gfx/3d/OBJ.spec>

[11] *COLLADA*: *Digital Asset and FX Exchange Schema* [online]. 2009 [cit. 2012-05-10]. Dostupné z: <https://collada.org/mediawiki/index.php/COLLADA_-_Digital_Asset_and_FX_Exchange_Schema>

[12] ZAKAS, Nicholas C. *JavaScript pro webové vývojáře*. Překlad Lukáš Krejčí. Brno: Computer Press, 2009, 832 s. ISBN 978-80-251-2509-0.

[13] Library (computing). In: *Wikipedia*: *the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2012-04-17]. Dostupné z: <http://en.wikipedia.org/wiki/Library_(computing) >

[14] Glmatrix: Javascript Matrix and Vector library for High Performance WebGL apps. *GitHub* [online]. © 2012 [cit. 2012-04-19]. Dostupné z: <https://github.com/toji/gl-matrix>

[15] Google WebGL utils. [online]. [cit. 2012-04-20]. Dostupné z: <https://cvs.khronos.org/svn/repos/registry/trunk/public/webgl/sdk/demos/common/webgl-utils.js>

Příloha A – Seznam funkcí pro používání knihovny

setCanvas(id) – přidělí knihovně canvas, který se má využít. Vyplní se id canvasu

webGLInit() – připraví WebGL prostředí pro práci

webGLStart() – započne vykreslování modelů

loadStart() – zobrazí informaci, že se scéna nahrává (obsaženo ve webGLInit())

loadStop() – skryje informaci, že se scéna nahrává (obsaženo ve webGLStart())

setAmbientLight(r, g, b) – nastaví barvu všudypřítomného osvětlení, barva je RGB

setDirectionalLightPosition(x, y, z) – určí pozici ze které bude vycházet směrové světlo

setDirectionalLightColor(r, g, b) – určí barvu směrového světla

setLighting(boolean) – zapne(true)/vypne(false) osvětlení

setBgColor(r, g, b) – nastaví barvu pozadí canvasu

AbsOtoc(x, y, z) – určuje absolutní otočení scény

RelOtoc(x, y, z) – určuje relativní otočení scény

AbsPosun(x, y, z) – určuje absolutní otáčení scény

RelPosun(x, y, z) – určuje relativní otáčení scény

AbsZvetseni(x,[y, z]) – určuje absolutní zvětšení scény

RelZvetseni(x,[y, z]) – určuje relativní zvětšení scény

SetPivot(x, y, z) – změní střed otáčení a zvětšování

nahrajModel(cesta) – nahraje model ze souboru, který se nachází na zadané cestě

modely[číslo].setVisible(boolean) – nastavuje viditelnost daného modelu ve scéně

modely[číslo].bodVlozeni(x, y, z) – změní bod vložení daného modelu

modely[číslo].otoceni(x, y, z) – změní natočení daného modelu

bodVlozeni(x, y, z) – změní bod vložení posledního vytvořeného modelu

otoceni(x, y, z) – změní natočení posledního vytvořeného modelu

setColor(r, g, b) – změní barvu poledního vytvořeného modelu

setColorI(r, g, b, i) – změní barvu i-té geometrii posledního vytvořeného modelu

setTextureI(cesta, i) – nastaví texturu i-té geometrii posledního vytvořeného modelu

setTextureFilter(cislo) – nastaví filtrování textur posledního vytvořeného modelu 1 – filtrování nejbližšího souseda, 2 – bilineární filtrování, 3 – Bilineární s mipmapami

vytvorHranol(x, y, z) – vytvoří model hranolu o rozměrech x, y, z

vytvorObedlnik(x, y) – vytvoří obdélník o rozměrech x, y

vytvorCtyruhelnik(x1, y1, x2, y2, x3, y3, x4, y4) – vytvoří čtyřúhelník ze zadaných vrcholů

vytvorTrojuhelnik(x1, y1, x2, y2, x3, y3) – vytvoří trojúhelník ze zadaných vrcholů

vytvorCaru(x1, y1, x2, y2, t) – vytvoří čáru z prvního bodu do druhého o tloušťce t, s přivrácenou osvětlenou stranou

vytvorCaruB(x1, y1, x2, y2, t) – vytvoří čáru z prvního bodu do druhého o tloušťce t, s odvrácenou osvětlenou stranou

vytvorKruh(r, s) – vytovří kruh o poloměru r s hladkostí s

vytvorValec(r, v, s) – vytvoří válec o poloměru r, výšce v a hladkostí s

vytvorKouli(r, s) – vytvoří kouli o poloměru r a hladkostí s, s by mělo být číslo dělitelné dvěma

MultiCara() – vytvoří instanci pro vytvoření multičáry (multiCara = new MultiCara())

multiCara.pridejCaru(x1, y1, x2, y2, t) – přidá čáru z prvního bodu do druhého o tloušťce t, s přivrácenou osvětlenou stranou

multiCara.pridejCaruB(x1, y1, x2, y2, t) – přidá čáru z prvního bodu do druhého o tloušťce t, s odvrácenou osvětlenou stranou

multiCara.vytvor() – vytvoří model s multičárou