UNIVERZITA KOMENSKÉHO V BRATISLAVE

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

WEBOVÝ VIDEO PREHRÁVAČ S PODPOROU 3D

Diplomová práca

2012

Bc. Ivana Uhlíková

UNIVERZITA KOMENSKÉHO V BRATISLAVE

FAKULTA MATEMATIKY, FYZIKY A INFORMATIKY

WEBOVÝ VIDEO PREHRÁVAČ S PODPOROU 3D

Diplomová práca

Študijný program:	Počítačová grafika a geometria
Študijný odbor:	1113 MATEMATIKA
Školiace pracovisko:	Katedra algebry, geometrie a didaktiky matematiky
Školiteľ:	RNDr. David Běhal
Kód práce:	5c23844d-47b8-4c88-80e1-5dd409ce980d

Bratislava 2012

Bc. Ivana Uhlíková

ZADANIE ZÁVEREČNEJ PRÁCE

Meno a priezvisko študenta: Bc. Ivana Uhlíková
Študijný program: počítačová grafika a geometria (Jednoodborové štúdium, magisterský II. st., denná forma)
Študijný odbor: 9.1.1. matematika
Typ záverečnej práce: diplomová
Jazyk záverečnej práce: slovenský

Názov: Webový video prehrávač s podporou 3D

Cieľ: Navrhnúť prehrávač videa pre webové rozhranie, ktorý vie prehrávať video a podporuje zobrazenie 3D. Navrhnúť možné projekcie videa na hladké plochy. Naštudovať problematiku, navrhnúť riešenie a implementovať ukážkový prípad. Otestovať a vyhodnotiť riešenie.

Vedúci: Mgr. David Běhal
Katedra: FMFI.KAGDM - Katedra algebry, geometrie a didaktiky matematiky

Dátum zadania: 27.10.2010

Dátum schválenia: 28.10.2010

doc. RNDr. Andrej Ferko, CSc.
garant študijného programu

.....
študent

.....
Vedúci

Čestne prehlasujem, že túto diplomovú prácu som vypracovala samostatne s použitím uvedených zdrojov.

V Bratislave dňa .....

Ivana Uhlíková

Poďakovanie

Týmto by som chcela poďakovať vedúcemu diplomovej práce RNDr. Davidovi Běhalovi za odborný dohľad, cenné pripomienky a usmernenia pri tvorbe aplikácie a písaní diplomovej práce.

Abstrakt

Táto diplomová práca sa venuje tvorbe a popisu tvorby webovej aplikácie, ktorá slúži ako bežný video prehrávač, ale má aj netradičnú funkciu, a tou je, že video sa mapuje na plochu v 3D priestore. Túto plochu je možné otáčať v priestore a meniť jej tvar. V práci vysvetľujem pojmy spojené s touto problematikou a uvádzam možnosti tvorby aplikácie spájajúcej video s 3D obsahom. Cieľom tejto diplomovej práce je návrh a vytvorenie aplikácie umožňujúcej prehrávať video s možnosťou rozšírenej interakcie, uploadovať vlastné video a publikovať aplikáciu na internete. Chcem tým ukázať nové možnosti využitia dostupných technológií.

Kľúčové slová: video prehrávač, 3D, web, mapovanie videa

Abstract

This diploma thesis is devoted to development and describing creation of a web application that serves as a standard video player, but has also unusual feature, that is video mapped to the patch in 3D space. This patch can be rotated in space and we can change its shape. The thesis explains the concepts related to these issues and present the possibilities of creating application, which combines video with 3D content. The aim of this thesis is to design and create application allowing play video with possibility of enhanced interaction, uploading own videos and publish application on the Internet. I want to show the new possibilities of utilization of available technologies.

Keywords: video player, 3D, web, video mapping

Obsah

Úvod

3D filmy a hry už sú bežnou realitou, dajú sa kúpiť, je možné ich stiahnuť z internetu, 3D filmy môžeme sami natáčať 3D kamerami. S takýmto obsahom sa najčastejšie stretávame v kine (3D filmy) alebo doma na vlastnom počítači (3D hry). Zatiaľ je však málo rozšírené spojenie 3D obsahu a videa priamo na internete. Moja práca sa bude snažiť o priblíženie tejto problematiky. Konkrétne sa budem venovať možnostiam mapovania videa na hladkú plochu v 3D priestore na internete. Zameriam sa na možnosti implementácie 3D obsahu a videa na webe. Vytvorím ukážkovú aplikáciu, ktorá bude mapovať video na hladkú plochu a popíšem postup tvorby tejto aplikácie.

Cieľ

Cieľom mojej diplomovej práce je navrhnúť prehrávač videa pre webové rozhranie, ktorý vie prehrávať video a podporuje zobrazenie 3D. Navrhnúť možné projekcie videa na hladké plochy. Naštudovať problematiku, navrhnúť riešenie a implementovať ukážkový prípad. Otestovať a vyhodnotiť riešenie.

Motivácia

Na internete je možné nájsť niekoľko aplikácií, v ktorých je video mapované na steny rotujúcej kocky, prípadne na zakrivenú plochu. Rotácia objektu môže byť automatická, alebo objekt môže otáčať samotný používateľ. To je však jediná interaktivita, ktorú tieto aplikácie poskytujú. Ja v mojej práci budem video mapovať na plochu, ktorej tvar je možné meniť. Teda motivácia je v oblasti rozšírenej interakcie s videom, hlavne v prípade zmiešavania videa s 3D obsahom. V súčasnosti existujú virtuálne 3D scény a trojrozmerné vizualizácie rôznych miest dostupné na internete. Virtuálne prostredie sa vyznačuje tým, že používateľ sa v ňom môže pohybovať podobným spôsobom ako v bežnom svete a môže aj manipulovať s rôznymi objektmi. Pre používateľa je teda možnosť meniť tvar plochy, na ktorej je premietané nejaké video, zaujímavým a netradičným pridaním interaktivity. Plochu budem reprezentovať Tenzorovo-súčinovou Bézierovou záplatou, ktorá je zadaná riadiacimi vrcholmi. Posúvaním týchto riadiacich

vrcholov v priestore je možné meniť tvar plochy. Inou možnosťou je preddefinovať pozíciu riadiacich vrcholov a tým určiť tvar plochy. Zaujímavým využitím tejto technológie by bolo vymodelovať ľudskú tvár prostredníctvom tejto záplaty a potom ju otextúrovať videom, ktoré by vizualizovalo starnutie človeka. Iným príkladom využitia by bola vodná hladina, ktorá sa vlní a na nej sa prehráva odraz videa. Video je možné mapovať aj na množstvo iných objektov, ktoré majú nepravidelný alebo meniaci sa tvar.

Stručný obsah

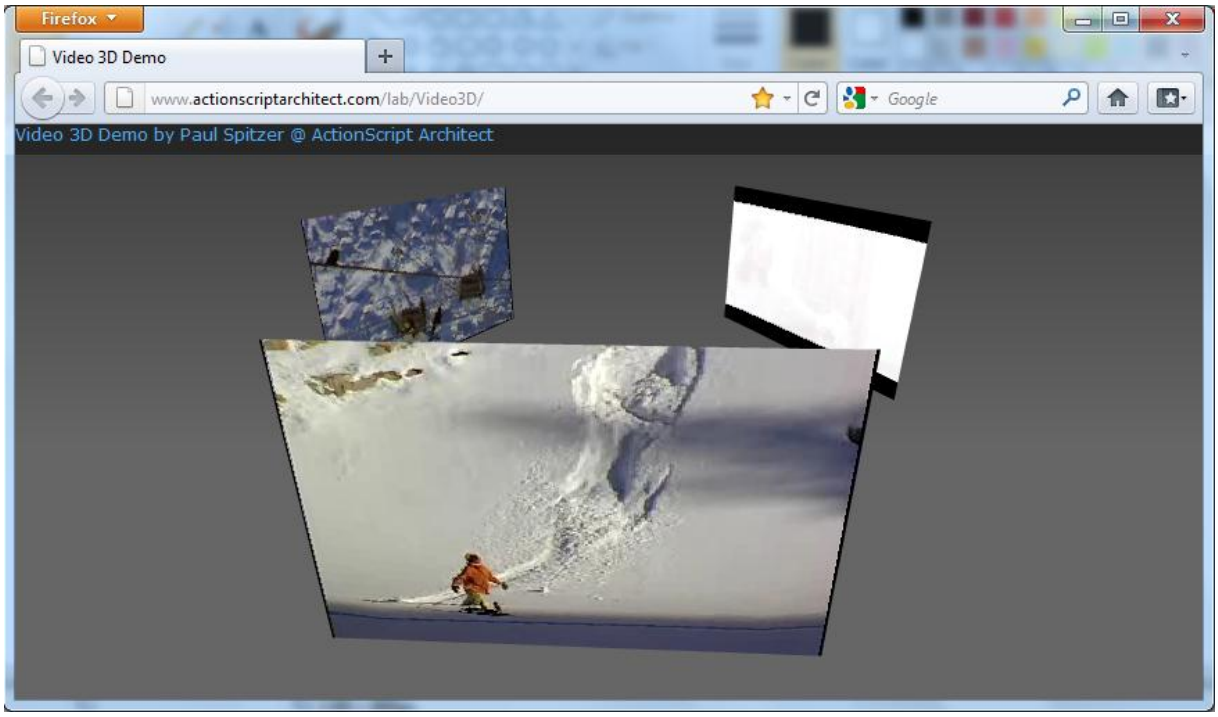
Prvou kapitolou diplomovej práce je prehľad problematiky. Zaoberám sa v nej súvisiacimi prácami, v ktorých sa mapuje video na hladké plochy na internete. Uvádzam softvérové možnosti implementácie a na základe naštudovaných technológií vyberám najvhodnejšiu technológiu pre moju aplikáciu. V druhej kapitole navrhujem aplikáciu a upresňujem jej funkcionalitu. Špecifikujem hardvérové a softvérové požiadavky navrhovanej aplikácie. Postup, ktorý som zvolila pri tvorbe aplikácie sa nachádza v tretej kapitole s názvom Implementácia. Poslednou kapitolou je testovanie. Testovanie bolo vykonané prostredníctvom dotazníka. Respondenti mali možnosť vyskúšať si aplikáciu a na základe toho ju ohodnotiť. V závere popisujem výsledok mojej práce a uvádzam možnosti vylepšenia.

1 Prehľad problematiky

V tejto kapitole sa na úvod budem venovať súčasnému stavu riešenia problematiky mapovania videa na hladké plochy v 3D priestore na internete. V ďalšej časti tejto kapitoly približujem dostupné programy na vytváranie aplikácií, ktoré dokážu prehrávať video a podporujú 3D. Medzi takéto programy patrí napríklad flash v kombinácii s prídavnou externou 3D knižnicou (ako napr. Papervision 3D), konkurenčný softvér Silverlight a VRML. Ja som sa rozhodla vytvoriť moju aplikáciu v najnovšej verzii HTML, teda v HTML 5. HTML 5 umožňuje jednoduchým spôsobom vkladať a načítavať videá na internetové stránky. Ďalšou novinkou je element <canvas>, v preklade plátno, ktorý umožňuje v spolupráci s WebGL renderovanie 3D scén v reálnom čase. WebGL je nízko úrovňové API, čo prináša aj výhody ako multiplatformovosť, ale aj potrebu doprogramovať si vlastné funkcie, prípadne použiť externé JavaScriptové knižnice. V ďalšej časti tejto kapitoly sa preto venujem teórii projekcie z 3D priestoru do 2D priestoru obrazovky. Vo WebGL sa pri textúrovaní používa MIP-mapping, preto aspoň v krátkosti priblížim, o akú techniku sa jedná. Posledná časť tejto kapitoly je venovaná Tenzorovo-súčinovým Bézierovým záplatám, keďže v mojej aplikácii mapujem video práve na takúto plochu.

1.1 Súvisiace práce

Na internete je možné nájsť niekoľko aplikácií, v ktorých je video mapované na steny rotujúcej kocky, prípadne na zakrivenú plochu. Rotácia objektu môže byť automatická, alebo objekt môže otáčať samotný používateľ. To je však jediná interaktivita, ktorú tieto aplikácie poskytujú. Príkladom takejto aplikácie je video mapované na tri vypuklé plochy od architekta ActionScriptu Paula Spitzera. Jeho aplikácia sa nachádza na tejto stránke: http://www.actionscriptarchitect.com/lab/Video3D



Obr.1: Screenshot z aplikácie Paula Spitzera

Táto aplikácia je vytvorená vo flashi s použitím externej knižnice Papervision 3D. Tri rôzne videá sú mapované na tri zakrivené plochy. Tieto plochy je možné otáčať v priestore pohybom myši po obrazovke.

Afinné mapovanie textúr

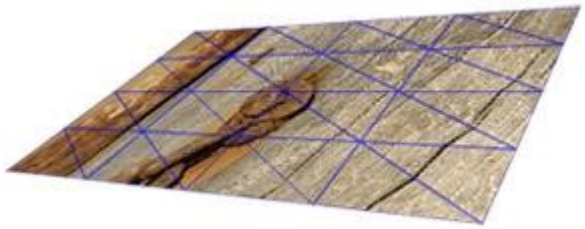
Papervision 3D obsahuje typický súbor objektov, ako aj iné 3D enginy: scény, kamery, mriežky, materiály a iné. Čo robí tento engine takým úspešným, je jeho spôsob mapovania textúr. Namiesto perspektívne správneho mapovania textúr, ktoré je výpočtovo náročné, engine používa tzv. afinné mapovanie textúr. Na nasledujúcich obrázkoch je porovnanie týchto dvoch techník mapovania textúr.

Obr.2: Textúra, ktorú chceme mapovať [Spitzer]

Obr.3: Afinné mapovanie textúry [Spitzer]

Obr.4: Perspektívne správne mapovanie textúry [Spitzer]

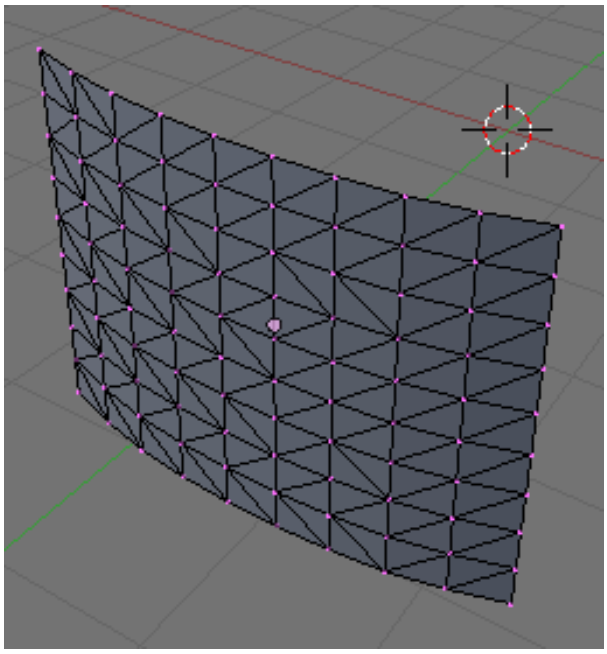
Ako je možné vidieť z porovnania obrázkov 3 a 4, afinné mapovanie textúr nie je tak presné ako perspektívne správne mapovanie textúr. Avšak, prerozdelením plochy na menšie plôšky je deformácia menej viditeľná (pozri obr.5).



Obr.5: Afinné mapovanie textúry na prerozdelenú plochu [Spitzer]

Podpora formátu Collada

Papervision 3D do veľkej miery podporuje formát Collada, ktorý sa používa ako výmenný formát medzi aplikáciami pre prácu s 3D obsahom [collada]. V tejto aplikácii je video mapované na plochu, ktoré bola vymodelovaná v Blenderi (pozri obr.6). Blender je free a open-source, slúži na modelovanie 3D scén a objektov, pre ich animáciu a renderovanie. Je šírený pod licenciou GNU General Public License [blender]. Pred exportom modelu z Blendra do formátu Collada je treba plochu previesť z štvoruholníkov na trojuholníky.



Obr.6: Plocha vytvorená v blendri [Spitzer]

Plocha vytvorená v blendri je však statická plocha, v aplikácii nie je možné meniť jej tvar [Spitzer].

1.2 Flash

V súčasnosti už nájdeme len málo statických a čisto textových internetových stránok. Väčšina má multimediálny obsah, ktorý sa môže meniť v závislosti od správania používateľa. Bežne sa stretávame s animáciami, videami, reklamami, rôznymi užitočnými alebo zábavnými aplikáciami (ako sú napríklad hry). Interaktívne webové aplikácie a multimediálne stránky sú vytvárané najmä pomocou softwaru flash, ktorý je vyvíjaný firmou Adobe už od roku 1996. Ako prvý umožňoval pri vytváraní animácií so zvukovými efektmi použitie vektorovej grafiky, ktorá má v porovnaní s bitmapovou dve dôležité výhody. Prvou je možnosť zväčšovania či zmenšovania bez straty ostrosti a kvality obrazu. Druhou výhodou je veľkosť výsledného súboru, ktorý je rozhodne menší než súbor s animáciou vo formáte GIF. Od roku 1999 bolo možné ukladať zvuk pomocou MP3 kompresie, ktorá zabezpečila prijateľnú veľkosť ozvučeného súboru. Ďalšou funkciou bol tzv. streaming, teda prehrávanie súboru za jeho súčasného sťahovania. Flash sa na internetových stránkach presadil taktiež ako videoprehrávač, podporuje najpoužívanejšie audiovizuálne formáty a má príjemné ovládacie prvky. Avšak na prehranie flashovej prezentácie je nutné mať v prehliadači nainštalovaný plug-in, najlepšie Adobe Flash

Player. Keďže sa jedná o komerčný produkt firmy Adobe, korporácie vyvíjajúce webové prehliadače (Microsoft, Mozilla, Apple, Opera, Google a iné) zaňho platia poplatky [Salvet11].

Flash je multimediálna platforma, využívaná pre prehrávanie videa, animácií a pridávanie interakcie na webové stránky. Hlavnou výhodou tejto technológie je multiplatformovosť, pretože existujú Flashplayer prehrávače pre viacero najznámejších operačných systémov. S tým je spojená ďalšia výhoda, a to veľká rozšírenosť medzi používateľmi internetu. Pod pojmom 3D vo Flashi sa rozumie schopnosť vykresľovať trojrozmerné scény pomocou rôznych vykresľovacích techník v technológii Flash. Konkrétne Flash 11 už má v sebe podporu pre vykresľovanie scény, avšak zatiaľ je to stále bez pokročilejších vykresľovacích techník akými sú napríklad shadere a textúry. Tie sa dajú nájsť hlavne v externých pomocných knižniciach slúžiacich pre vykresľovanie 3D scény. Medzi takéto knižnice, ktoré sa dajú nájsť na internete, patria napríklad open source knižnice: Away3D, Papervision3D a Sandy3D [Kolesár].

1.3 Silverlight

Konkurenčnou technológiu najmä pre Flash je Silverlight, ktorý bol uvedený Microsoftom v apríli 2007. Využíva sa na vytváranie interaktívnych webových aplikácií. Silverlight je postavený na technológii Windows Presentation Foundation (WPF) a .NET Framework 3.0. Silverlight je šírený ako freeware [Salvet11].

Silverlight 5 je najnovšia verzia, jej finálna podoba bola k dispozícii na stiahnutie od 9. decembra 2011. Medzi nové funkcie patrí: GPU akcelerované dekódovanie videa, 3D grafika, ovládanie rýchlosti prehrávania, diaľkové ovládanie a 64-bitová podpora. V čase keď vyšla táto najnovšia verzia, som už pracovala na implementácii pomocou HTML 5 a WebGL. Ale možno do budúcna by bolo zaujímavé vytvoriť podobnú aplikáciu aj pomocou Silverligtu.

1.4 VRML

Jazyk VRML 97 (Virtual Reality Modeling Language) je medzinárodnou normou ISO pre popis statických a dynamických svetov, definuje spôsob zápisu virtuálnych svetov do

súborov v textovom tvare. Je teda súčasne i tzv. formátom, predpisom pre zapisovanie informácií určitého typu. Súbory, ktoré obsahujú VRML svety, majú príponu .wrl (z anglického world).

VRML je tesne zviazané s internetom a prispôsobené jeho prenosovým možnostiam. Pre vstup do VRML aplikácie je potrebné si do internetového prehliadača doinštalovať plug-in, napr. Cosmo Player (Silicon Graphics, Inc.) alebo Cortona3D Viewer (Parallel Graphics). Prehliadač VRML je program, ktorý je schopný previesť textový popis z VRML súboru do obrazu virtuálneho sveta. Navyše umožňuje pohyb v tomto svete a prípadnú interakciu s virtuálnymi predmetmi. Vzhľadom k úzkemu vzťahu VRML a WWW je väčšina dostupných prehliadačov VRML súborov súčasťou internetových prehliadačov. Samostatné prehliadače sú výnimkou, prípadne súčasťou programov pre vytváranie VRML svetov [Žára].

Nasledujúce kapitoly diplomovej práce sú spracované podľa zdroja [w3chools].

1.5 HTML

HTML je jazyk pre popis webových stránok. HTML je skratka pre Hyper Text Markup Language. HTML nie je programovací jazyk, ale značkovací jazyk. Značkovací jazyk je sada tagov (značiek). Jazyk HTML pomocou tagov popisuje webové stránky. HTML tagy sú kľúčové slová obklopené špicatými zátvorkami, ako napr. <html>. HTML tagy sú zvyčajne párové, ako napr. a . Nazývame ich otváracie a uzatváracie tagy.

1.5.1 HTML Elementy

HTML element je všetko od otváracieho tagu až po uzatvárací tag:

Otvárací tag	Obsah elementu	Uzatvárací tag
<h1>	Nadpis	</h1>
	Odkaz	

Tabuľka 1: Časti HTML elementu

Prázdne elementy sú uzavreté už v otváracom tagu (napr.
). Väčšina HTML elementov môže mať atribúty. HTML elementy väčšinou môžu byť vnorené (môžu

obsahovať ďalšie HTML elementy). HTML dokumenty sa skladajú z vnorených HTML elementov:

```
<html>
<body>
<p>Toto je môj odsek.</p>
</body>
</html>
```

1.5.2 História HTML

Okolo roku 1991 spísal pracovník CERN-u (európske stredisko pre výskum atómových častíc) Tim Berners-Lee návrh hypertextového projektu, z ktorého neskôr vzniklo HTML. Zásadné, až prelomové na jazyku HTML bolo to, že sa v ňom dali používať hypertextové odkazy. To umožňovalo používateľom ľahší pohyb medzi jednotlivými dokumentmi uloženými na internetových serveroch. Konečne tak odpadlo prechádzanie adresárových štruktúr, ktoré bolo o to náročnejšie, čím viac sieť rástla a čím bohatší bol obsah serverov pripojených k sieti. Efektívnosť hypertextových odkazov sa tak stala kľúčovým faktorom podieľajúcim sa na fenomenálnom nástupe HTML.

1.5.3 Verzie HTML

Až do verzie 3.2 obsahovalo HTML hlavne tagy pre vyznačenie štruktúry a pre popis informácií, teda sémantické značky, ktoré zo svojej podstaty neumožňovali autorom webových stránok určovať v dostatočnej miere grafickú podobu HTML dokumentov. Vývojári prehliadačov preto do HTML zakomponovali množstvo čisto prezentačných, vzhľad určujúcich tagov, ako napr. (pre zarovnanie obsahu do stredu). Tieto prezentačné tagy boli potom štandardizované práve verziou 3.2. Týmto spôsobom bolo zaistené vytváranie pekných a vzhľadovo variabilných stránok, zvlášť pri využití tabuľkového layoutu.

Tento krok mal však aj svoju odvrátenú tvár. Kvôli nevhodnému používaniu tabuliek pre formátovacie účely a nadmernému používaniu prezentačných tagov strácali HTML dokumenty svoju prehľadnosť. Taktiež sémantická úroveň išla rapídne dole, pretože autori stránok štrukturálne a deskriptívne tagy čiastočne vynechávali, resp. nahradzovali ich prezentačnými.

Štandardizačná webová organizácia W3C (World Wide Web Consortium) teda zahájila ťaženie proti nastolenému trendu a v roku 1997 vydala špecifikáciu HTML 4, kde sa zamerala na vylepšenie štruktúry dokumentu. Pre vzhľad stránok mali slúžiť predovšetkým štýly CSS (Cascading Style Sheets). Nový spôsob definovania vzhľadu sa vďaka svojim výrazným výhodám (vzhľad celého webu bolo možné definovať v externom súbore, čo je prehľadnejšie aj úspornejšie) rýchlo ujal a s prehľadom vytlačil tabuľkový layout i väčšinu prezentačných elementov.

W3C malo ešte odvážnejšie plány. V roku 1999 uviedlo nový jazyk – XHTML. Ten mal nahradiť HTML, ktorého vývoj bol oficiálne prehlásený za ukončený. Tento manéver nebol samoúčelný, ale bol súčasťou širšej stratégie. Konzorcium zamýšľalo zjednotiť značkovacie jazyky pod jedným jednoduchým dátovým formátom, ktorým sa malo stať XML (eXtensible Markup Language). Cieľom bolo dosiahnuť maximálnej konvertibility a umožniť tak ľahšiu prácu s dátami. Prechod na XHTML dával nádej na prekonanie chaosu v syntaxi a na návrat k čisto štruktúrovanému kódu. Avšak vývoj XHTML prestal hneď na samom začiatku. Prvá verzia bola iba reformulovaným HTML. Ďalšia verzia XHTML 2 tak, ako bolo navrhované bolo inkompatibilné s predchádzajúcou verziou, vôbec sa nezaoberalo praktickými potrebami tvorcov webu, najmä sa však nedokázalo vysporiadať s nástupom a rozvojom webových aplikácií. Roky ubiehali a sklamanie rástlo.

V tejto situácii prišlo opäť na rad zavrhnuté HTML. S myšlienkou na znovuobnovenie jeho vývoja vystúpila skupina WHATWG (Web Hypertext Application Technology Working Group), združujúca najmä mnoho tvorcov webových prehliadačov ako Mozilla Foundation, Opera Software či Apple. Nezostalo to len pri myšlienke, prikročilo sa aj k dielu, ktoré sa úspešne rozvíjalo. V roku 2007 sa projekt presunul "pod krídla" W3C. S XHTML 2 to dopadlo horšie. Ani po bezmála desiatich rokoch sa ho nepodarilo dokončiť, taktiež sa nenašiel nikto, kto by javil ochotu ho implementovať [Salvet10].

1.6 HTML 5

HTML 5 bude nový štandard pre HTML. Predchádzajúca verzia HTML, HTML 4.01, prišla v roku 1999. Web sa od tej doby zmenil. HTML 5 je stále vo vývoji a ešte nemá podobu schváleného štandardu. Avšak, významnejšie internetové prehliadače podporujú

mnohé nové HTML 5 elementy a API (rozhranie pre programovanie aplikácií). Vývojári webových prehliadačov majú v pláne aj naďalej pridávať nové funkcie HTML 5 do ich najnovších verzií.

HTML 5 je spolupráca medzi World Wide Web Consortium (W3C) a Web Hypertext Application Technology Working Group (WHATWG). WHATWG pracovalo s webovými formulármi a aplikáciami, W3C pracovalo s XHTML 2.0. V roku 2006 sa rozhodli spolupracovať a vytvoriť novú verziu HTML.

Pre HTML 5 boli stanovené tieto pravidlá:

- Nové funkcie by mali byť založené na HTML, CSS, DOM a JavaScripte
- Znížiť potrebu externých plug-inov (ako napr. flash)
- Lepšie ošetrenie chýb
- Ďalšie značky pre náhradu skriptovania
- HTML 5 by malo byť nezávislé na zariadení
- Vývojový proces by mal byť viditeľný pre verejnosť

Do vývoja novej verzie jazyka zasahovali všetci veľký "výrobcovia" prehliadačov a i vďaka tomu je vidieť posun od textovo orientovaného jazyka smerom ku konkrétnemu využitiu. HTML 5 obsahuje veľa nových funkcií, umožňujúcich napríklad ľahšiu integráciu videa alebo lepšie možnosti zabezpečenia stránok. Vývojári preto označujú HTML 5 ako revolúciu na webe.

Niektoré z najzaujímavejších nových funkcií HTML 5:

- Element <canvas> pre 2D kreslenie, prípadne pre vykresľovanie 3D scén v spolupráci s WebGL
- <video> a <audio> elementy pre prehrávanie médií
- Podpora pre lokálne ukladanie
- Nové obsahovo-špecifické elementy, ako <article>, <footer>, <header>, <nav> a <section>
- Nové ovládacie prvky formulára, ako je kalendár, dátum, čas, email, url, hľadanie

1.6.1 Multimédiá v HTML 5

Špecifikácia HTML 5 ponúka lepšiu integráciu multimediálneho obsahu. Doteraz prehliadače potrebovali pre prehrávanie filmov a hudby na webe plug-iny (ako napr. flash), čo nie je práve optimálne riešenie. Nevýhody plug-inov: Pre svoje fungovanie spotrebujú viac "zdrojov" (CPU, RAM), sú náchylné k pádom a predovšetkým predstavujú nezanedbateľné bezpečnostné riziko. Medzery pri spracovaní formátu flash patria medzi často využívané "cesty", ako sa malware dostáva do PC. A práve tu má priniesť HTML 5 žiadanú zmenu. Nové tagy <audio> a <video> umožnia ľahkú integráciu multimediálnych súborov. HTML 5 špecifikuje štandardný spôsob, ako vložiť video na stránku, pomocou <video> elementu.

```
<video width="320" height="240" controls="controls">
  <source src="movie.mp4" type="video/mp4" />
  <source src="movie.ogg" type="video/ogg" />
  Váš prehliadač nepodporuje video tag.
</video>
```

Obr.7: Príklad použitia <video> elementu

Atribút controls pridá ovládanie videa, ako napríklad prehrať, pozastaviť a hlasitosť. Ovládanie sa líši v závislosti od internetového prehliadača.

Obr.8: Ovládacie prvky video prehrávača vo vybraných internetových prehliadačoch

Vždy je vhodné určiť šírku a výšku <video> elementu. Ak sú tieto atribúty nastavené, tak sa rezervuje miesto potrebné pre video, už keď sa načítava stránka. Avšak, bez týchto atribútov prehliadač nevie rozmery videa a nemôže preňho rezervovať vhodné miesto. Následkom je, že rozloženie stránky sa bude meniť v závislosti od načítania videa. Taktiež je vhodné vložiť text medzi <video> a </video> tagy kvôli prehliadačom, ktoré nepodporujú <video> element.

<video> element dovoľuje viacnásobné použitie <source> elementu. <source> elementy môžu odkazovať na rôzne video súbory. Prehliadač použije prvý rozpoznaný formát.

V súčasnosti existujú 3 podporované video formáty pre <video> element: MP4, WebM a Ogg.

Prehliadač	MP4	WebM	Ogg
Internet Explorer 9	áno	nie	nie
Firefox 4.0	nie	áno	áno
Google Chrome 6	áno	áno	áno
Apple Safari 5	áno	nie	nie
Opera 10.6	nie	áno	áno

Tabuľka 2: Podpora video formátov internetovými prehliadačmi

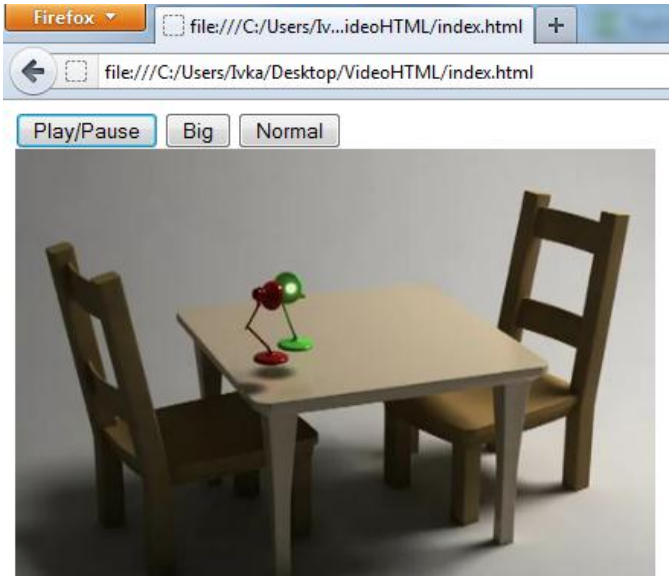
MP4 = MPEG 4 súbory s H264 video kodekom a AAC audio kodekom
WebM = WebM súbory s VP8 video kodekom a Vorbis audio kodekom
Ogg = Ogg súbory s Theora video kodekom a Vorbis audio kodekom

HTML 5 <video> element má taktiež metódy, vlastnosti a udalosti. Existujú napríklad metódy pre prehrávanie (play()), pozastavenie (pause()) a načítanie (load()). Vlastnosti videa sú napr. časová dĺžka (duration), hlasitosť (volume), či je video pozastavené (paused), aktuálny čas už prehraného videa (currentTime), aktuálny zdroj videa (currentSrc). Je možné ich čítať alebo nastaviť. Pre video existujú tiež DOM (Document Object Model) udalosti, ktoré môžu upozorňovať, napríklad keď <video> element začne prehrávanie (play), je pozastavený (pause), je ukončený (ended), boli načítané metadáta (loadedmetadata) atď.

Nasledujúci príklad ilustruje, ako sa dá jednoduchým spôsobom osloviť <video> element, ako čítať a nastavovať vlastnosti a ako volať metódy.

```
<!DOCTYPE html>
<html>
<body>
  <button onclick="playPause()">Play/Pause</button>
  <button onclick="makeBig()">Big</button>
  <button onclick="makeNormal()">Normal</button>
  <br />
  <video id="mojeVideo" width="420">
    <source src="lampy.mp4" type="video/mp4" />
    <source src="lampy.ogv" type="video/ogg" />
    Your browser does not support HTML5 video.
  </video>
<script type="text/javascript">
var myVideo=document.getElementById("mojeVideo");
function playPause() {
  if (myVideo.paused) myVideo.play();
  else myVideo.pause();
}
function makeBig() {
  myVideo.width=560;
}
function makeNormal() {
  myVideo.width=420;
}
</script>
</body>
</html>
```

Obr.9: Ukážka kódu jednoduchého HTML 5 video prehrávača



Obr.10: Ako vyzerá vyššie popísaný videoprehrávač na webe

1.6.2 Multithready: Rýchlejšie a pohodlnejšie surfovanie

Aby nedochádzalo k zbytočnému spomaleniu funkcií a aplikácií podporuje HTML 5 multithreading. Skripty a webové aplikácie môžu bežať paralelne v oddelených threadoch – pomocou funkcie s názvom "Web workers". Tato funkcia umožní rýchle štruktúrovanie stránky a napríklad zaistí nespomalené rolovanie stránky alebo zadávanie textu bez viditeľného oneskorenia [Müller].

1.6.3 Canvas nový element v HTML 5

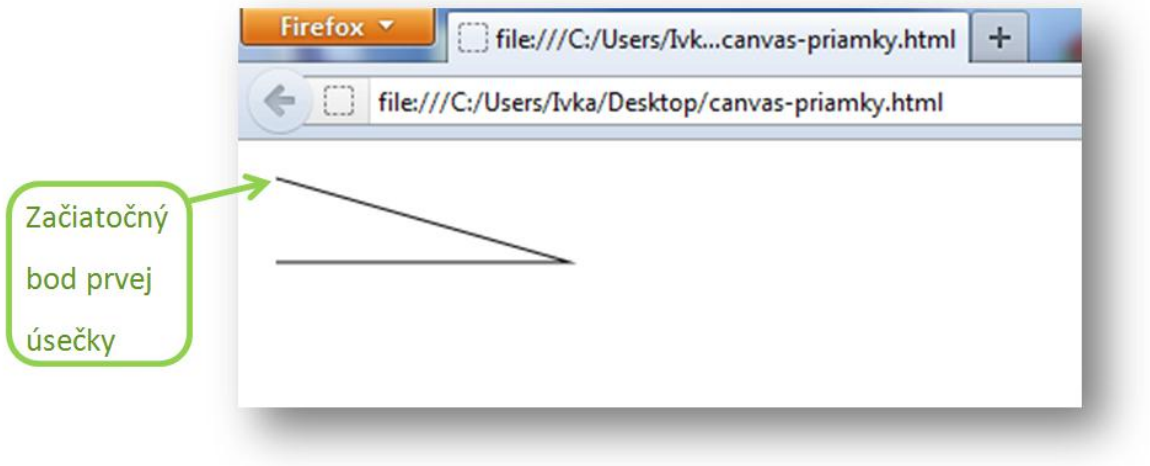
HTML 5 element <canvas> definuje priestor na webových stránkach, kde engine prehliadača renderuje grafiku v reálnom čase. To napríklad umožňuje integrovať do webovej stránky kresliaci program alebo priamo hru, bez špeciálneho doplnku. Element <canvas> sa používa najmä na vykresľovanie interaktívnej grafiky, čo umožňuje skriptovanie napr. v JavaScripte. Element <canvas> je teda len kontajner pre grafiku, je nutné použiť skript pre samotné vykreslenie grafiky. Canvas (v preklade plátno) je kresliaci región, ktorého rozmery sú definované v HTML kóde. Canvas má niekoľko metód pre kreslenie ciest, obdĺžnikov, kruhov, znakov a pre pridávanie obrázkov. Element <canvas> je podporovaný vo všetkých významnejších prehliadačoch.

Kreslenie pomocou JavaScriptu:

```
1 <!DOCTYPE HTML>
2 <html>
3 <body>
4 <canvas id="myCanvas" width="200" height="100">
5 Your browser does not support the canvas element.
6 </canvas>
7 <script type="text/javascript">
8 var c=document.getElementById("myCanvas");
9 var cxt=c.getContext("2d");
10 cxt.moveTo(10,10);
11 cxt.lineTo(150,50);
12 cxt.lineTo(10,50);
13 cxt.stroke();
14 </script>
15 </body>
16 </html>
```

Obr.11: Ukážka kódu pre prácu s canvasom

JavaScript používa id pre nájdenie elementu <canvas> (riadok 8). Potom vytvorí kontext objektu (riadok 9). Objekt getContext("2d") je vstavaný HTML 5 objekt s mnohými metódami pre kreslenie ciest, obdĺžnikov, kruhov, znakov, obrázkov a ďalších. Príkazy moveTo a lineTo slúžia pre určenie začiatočných a koncových bodov úsečiek. Metóda stroke() vykreslí zadané úsečky.



Obr.12: Ako vyzerá vyššie popísané použitie elementu canvas na webe

Element <canvas> sa dá použiť dvomi odlišnými spôsobmi. V predchádzajúcej ukážke bol zvolený "2d" kontext. Ak by sme chceli pracovať s 3D grafikou, treba do objektu getContext uviesť "experimental-webgl". Štandard WebGL umožňuje do canvasu renderovať komplexné 3D scény v reálnom čase pomocou príkazov z grafickej knižnice OpenGL.

```
<html>
  <body>
    <canvas id="myCanvas" width="200" height="100">
      Váš prehliadač nepodporuje element canvas.</canvas>
    <script type="text/javascript">
    var c = document.getElementById("myCanvas");
    var cv = c.getContext("2d");
    var gl = c.getContext("experimental-webgl");
    if (!gl) {
      alert("Váš prehliadač nepodporuje WebGL.");
    }
    </script>
  </body>
</html>
```

Obr.13: Použitie objektu getContext dvomi rôznymi spôsobmi

1.7 CSS

CSS (Cascading Style Sheets) definuje spôsob zobrazenia HTML elementov. CSS štýly vyriešili veľký problém. HTML nikdy nemalo obsahovať tagy pre formátovanie dokumentu. Na vymedzenie obsahu HTML dokumentu malo slúžiť napríklad:

- <h1> Toto je nadpis </ h1>
- <p> Toto je odsek. </ p>

Pridaním tagov ako napr. <font> a atribútov farby do špecifikácie HTML 3.2 sa začala nočná mora pre webových vývojárov. Vývoj veľkých webových stránok, kde sa informácie o písme a farbe pridávajú do každej stránky, sa stal dlhým a nákladným procesom. A práve pre vyriešenie tohto problému World Wide Web Consortium (W3C) vytvorilo CSS. V HTML 4.0 už mohlo byť všetko formátovanie odstránené z HTML dokumentu a uložené v samostatnom CSS súbore. Všetky internetové prehliadače podporujú CSS.

1.7.1 Syntax CSS

Pravidlo CSS má dve hlavné časti: selektor a jednu alebo viac deklarácií:

Selektor: `h1`

`{`

Deklarácia: `color: blue;`

Deklarácia: `font-size: 36pt;`

`}`

Selektor je zvyčajne HTML element. Okrem nastavenia štýlu pre HTML element však CSS umožňuje definovať aj svoje vlastné selektory s názvom "id" a "class".

"id" selektor sa používa na určenie štýlu pre jednotlivý, jedinečný element. "id" selektor používa atribút "id" HTML elementu a je definovaný pomocou mriežky "#". V nasledujúcom príklade bude pravidlo CSS aplikované na element s id="mojNadpis":

```
#mojNadpis
{
text-align: center;
color: red;
}
```

Na rozdiel od "id" selektora sa "class" selektor používa súčasne pre niekoľko elementov. "class" selektor sa teda používa na určenie štýlu pre skupinu elementov. To umožňuje nastaviť osobitný štýl pre množstvo HTML elementov s rovnakou triedou. "class" selektor používa "class" atribút HTML elementu a je definovaný pomocou bodky. V nasledujúcom príklade všetky HTML elementy s class="center" budú zarovnané na stred:

```
.center
{
text-align: center;
}
```

Taktiež je možné určiť, že len konkrétne HTML elementy budú ovplyvnené triedou. V nasledujúcom príklade všetky p elementy s class="center" budú zarovnané na stred:

```
p.center {
text-align: center;
}
```

1.7.2 Vkladanie CSS štýlov

Existujú tri spôsoby vkladania CSS štýlov:

- Externé CSS
- Interné CSS
- Inline štýly (v riadku)

Externé CSS

Externé CSS je ideálne, keď sa štýl aplikuje na viacero stránok. Zmenou externého CSS súboru je možné zmeniť vzhľad celého webu. Tento CSS súbor má koncovku `.css`. Každá stránka musí mať odkaz na CSS dokument pomocou tagu <link>. Tag <link> sa dáva dovnútra sekcie head:

```
<head>
<link rel="stylesheet" type="text/css" href="mojStyl.css" />
</head>
```

Interné CSS

Interné CSS by malo byť použité vtedy, keď jedna stránka webu má jedinečný štýl. Osobitné štýly jednej stránky je možné definovať v sekcii head HTML stránky pomocou tagu <style> takto:

```
<head>
<style type="text/css">
h2 { color: purple; }
p { margin-left: 20px; }
</style>
</head>
```

Inline štýly (v riadku)

Pri tejto metóde dochádza k miešaniu obsahu stránky a štýlov, čím táto metóda stráca najmä výhodu prehľadnosti oproti dvom predošlým spôsobom pripojenia CSS štýlov. A to je aj dôvod, prečo sa táto metóda používa iba málo. Inline štýly sa

implementujú pomocou <style> atribútu v danom tagu. <style> atribút môže obsahovať nejakú vlastnosť CSS. Príklad ukazuje, ako zmeniť farbu odseku a veľkosť ľavého okraja:

```
<p style="color: green; margin-left: 20px">Toto je odsek.</p>
```

1.8 CSS 3

CSS 3 je najnovší štandard pre CSS. Špecifikácia CSS 3 je ešte stále vo vývoji. Avšak, mnoho nových vlastností CSS 3 už bolo implementovaných do moderných internetových prehliadačov.

Niektoré z najzaujímavejších nových funkcií CSS 3:

- je možné vytvoriť zaoblené okraje, pridať tieň boxom a použiť obrázok ako hranicu
- nové vlastnosti pre pozadia: background-size a background-origin
- nové textové efekty: text-shadow a word-wrap
- pomocou CSS 3 transformácií je možné premiestňovať, škálovať, otáčať a skosiť HTML elementy
- CSS 3 prechody: môžeme pridať efekt pri prechode z jedného štýlu do druhého, bez použitia flash animácie alebo JavaScriptu
- s CSS 3 je možné vytvárať animácie, ktoré dokážu nahradiť animované obrázky, flashové a JavaScriptové animácie na mnohých webových stránkach
- s CSS3 je možné rozvrhnúť text do viacerých stĺpcov - podobne ako v novinách

1.9 JavaScript

JavaScript je skriptovací jazyk pre web. JavaScript je použitý na mnohých webových stránkach pre pridanie funkcionality, overovanie formulárov, komunikáciu so serverom a veľa ďalšieho. JavaScript je najpopulárnejší skriptovací jazyk na internete a funguje vo všetkých významných internetových prehliadačoch, ako Internet Explorer, Firefox, Chrome, Opera a Safari. JavaScript je zvyčajne vložený priamo do HTML stránok. JavaScript je interpretovaný jazyk (to znamená, že skripty sa spúšťajú bez predchádzajúcej kompilácie). Každý môže využívať JavaScript bez nutnosti zakúpenia licencie.

JavaScript poskytuje HTML dizajnérom programovací nástroj. Autori HTML zvyčajne nie sú programátori, ale JavaScript je skriptovací jazyk s veľmi jednoduchou

syntaxou. Takmer každý je schopný dať malé kúsky z kódu do svojich HTML stránok. JavaScript môže reagovať na udalosti. Môže byť nastavený na spustenie, keď sa niečo stane, ako napr. keď sa načíta stránka alebo keď používateľ klikne na HTML element. JavaScript môže čítať a meniť obsah HTML elementu. JavaScript je možné použiť pre overenie údajov vo formulári ešte predtým, ako sú odoslané na server. JavaScript môže byť použitý na detekciu internetového prehliadača návštevníka a v závislosti na prehliadači načítať ďalšiu stránku špeciálne navrhnutú pre daný prehliadač. JavaScript je možné použiť na vytvorenie cookies, teda pre ukladanie a načítanie informácií o počítači návštevníka.

1.9.1 História JavaScriptu

JavaScript je implementácia štandardu jazyka ECMAScript. ECMA-262 je oficiálny štandard JavaScriptu. JavaScript bol vynájdený Brendanom Eichom v internetovom prehliadači Netscape Navigator 2.0 a od roku 1996 sa objavil vo všetkých prehliadačoch. Oficiálna štandardizácia bola prevzatá organizáciou ECMA (priemyselná štandardizačná asociácia) v roku 1997. ECMA štandard (tzv. ECMAScript-262) bol schválený ako medzinárodná norma ISO v roku 1998. Vývoj stále prebieha.

1.9.2 Syntax JavaScriptu

HTML tag <script> sa používa pre vloženie JavaScriptu do HTML stránky. Vnútri tagu <script> je atribút `type` pre definovanie skriptovacieho jazyka. Pre manipuláciu s HTML elementmi JavaScript používa DOM metódu getElementById(). Táto metóda pristupuje k elementu so zadaným id. V nasledujúcom príklade sa zapíše aktuálny dátum do existujúceho <p> elementu:

```
<html>
<body>
<p id="demo"></p>
<script type="text/javascript">
document.getElementById("demo").innerHTML=Date();
</script>
</body>
</html>
```

V tomto príklade bol JavaScript vložený do elementu <body>. JavaScript je umiestnený v spodnej časti stránky pre uistenie, že nebude vykonaný skôr, ako sa vytvorí <p> element. JavaScript umiestený v HTML stránke bude vykonaný pri načítaní stránky. Niekedy chceme spustiť JavaScript, až keď dôjde k udalosti, ako napr. keď používateľ klikne na tlačidlo. Udalosť sa zvyčajne používa v kombinácii s funkciami. Keď nastane udalosť, tak sa zavolá JavaScriptová funkcia. V nasledujúcom príklade sa zavolá funkcia `zobrazDatum()` pri kliknutí na tlačidlo:

```
<html>
<head>
<script type="text/javascript">
function zobrazDatum()
{
document.getElementById("demo").innerHTML=Date();
}
</script>
</head>
<body>
<h1>Moja prvá webová stránka</h1>
<p id="demo"></p>
<button type="button" onclick="zobrazDatum()">Zobraz
dátum</button>
</body>
</html>
```

Skripty v elementoch <head> a <body>

Do HTML stránky je možné umiestniť neobmedzený počet skriptov a skripty je možné mať v časti <head> aj <body> zároveň. Skript je možné vložiť aj do funkcie. Všetky funkcie sú zvyčajne umiestnené v časti <head> alebo v dolnej časti stránky. Vďaka tomu sú všetky na jednom mieste a nemiešajú sa s obsahom stránky.

Použitie externého JavaScriptu

JavaScript môže byť tiež umiestnený v externých súboroch. Externé JavaScript súbory často obsahujú kód, ktorý sa používa na niekoľkých rôznych webových stránkach. Externý JavaScript súbor má príponu `.js`.

Pre použitie externého JavaScriptu je treba vložiť názov súboru s obsahom JavaScriptu do atribútu "src" v tage <script>:

```
<script type="text/javascript" src="nazov.js"></script>
```

1.10 DOM

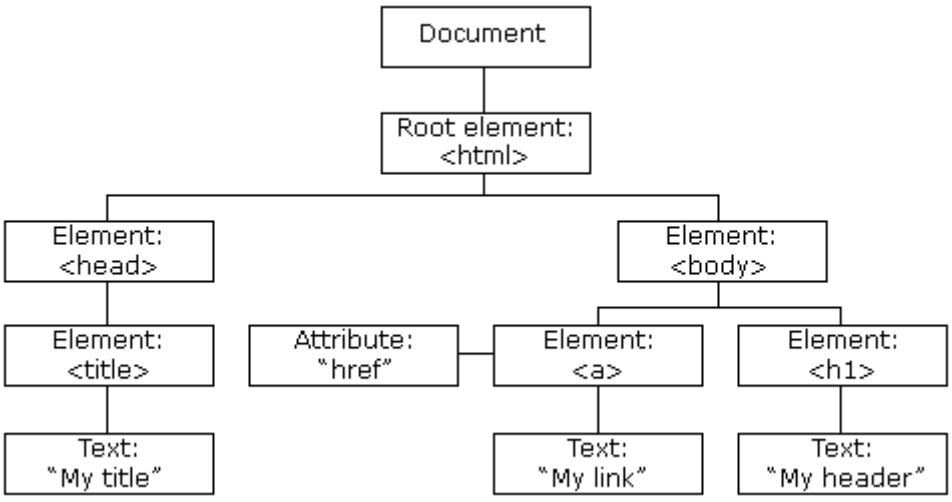
DOM - Document Object Model je W3C (World Wide Web Consortium) štandardom. DOM definuje štandard pre prístup k dokumentom, ako sú HTML a XML. DOM je platforma a jazykovo neutrálne rozhranie, ktoré umožňuje programom a skriptom dynamický prístup a možnosť aktualizovania obsahu, štruktúry a štýlu dokumentu.

DOM je rozdelený do 3 rôznych častí alebo úrovní:

- Jadro DOM - štandardný model pre štruktúrovaný dokument
- XML DOM - štandardný model XML dokumentov
- HTML DOM - štandardný model pre HTML dokumenty

1.10.1 HTML DOM

HTML DOM definuje štandardný spôsob pre prístup a prácu s HTML dokumentmi. DOM predstavuje HTML dokument ako stromovú štruktúru.



Obr.14: Príklad HTML DOM stromu [w3chools]

HTML DOM je štandardný objektový model a štandardné programovacie rozhranie pre HTML. Je nezávislé na platforme a jazyku. HTML DOM definuje objekty a vlastnosti

všetkých elementov HTML a metódy pre prístup k nim. Inými slovami: HTML DOM je štandard, pomocou ktorého je možné zmeniť, pridať alebo odstrániť elementy HTML.

Niektoré DOM vlastnosti:

- element.innerHTML - textová hodnota elementu
- element.nodeName - meno elementu
- element.nodeValue - hodnota elementu
- element.attributes - atribúty elementu

Niektoré DOM metódy:

- document.getElementById(id) - získať prvok s uvedeným id
- document.getElementsByTagName(názov) - získať všetky prvky s uvedeným názvom

Udalosti

Každý prvok na webovej stránke má určité udalosti, ktoré môžu spustiť JavaScriptové funkcie. Napríklad môžeme použiť udalosť onClick na <button> element ako indikáciu funkcie, ktorá sa spustí, keď používateľ klikne na tlačidlo.

Príklady udalostí:

- Kliknutie tlačidlom myši
- Načítanie webovej stránky alebo obrázka
- Umiestnenie kurzora nad hot spot na webovej stránke
- Výber input boxu v HTML formulári
- Odoslanie HTML formulára
- Stlačenie tlačidla na klávesnici

Udalosti sa obvykle používajú v kombinácii s nejakou JavaScriptovou funkciou, ktorá sa vykonaná až vtedy, keď dôjde k udalosti.

1.11 PHP

PHP je výkonný nástroj pre vytváranie dynamických a interaktívnych webových stránok. PHP je skratka pre PHP: Hypertext Preprocessor. PHP je skriptovací jazyk na

strane servera, PHP skripty sú teda vykonávané na serveri. PHP podporuje množstvo databáz (MySQL, Informix, Oracle, Sybase, Solid, PostgreSQL, ODBC Generic, atď). PHP je open source softvér. Beží na rôznych platformách (Windows, Linux, Unix). PHP je kompatibilné s takmer všetkými dnes používanými servermi (Apache, IIS, apod.).

PHP súbory môžu obsahovať text, HTML tagy a skripty. PHP súbory sú vrátené do internetového prehliadača ako obyčajný HTML text. PHP súbory majú príponu `.php`, `.php3`, alebo `.phtml`. PHP skript vždy začína s `<?php` a končí `?>`. PHP skript môže byť umiestnený kdekoľvek v dokumente [w3chools].

1.12 WebGL - OpenGL ES 2.0 pre web

Organizácia Khronos Group v marci minulého roku uvoľnila prvú finálnu špecifikáciu rozhrania WebGL vo verzii 1.0 pre využívanie 3D grafiky priamo na webových stránkach, ktorá sa implementuje prostredníctvom HTML 5 elementu canvas. WebGL umožňuje webovým aplikáciám pracovať s hardvérovo akcelerovanou 3D grafikou a priamo vytvárať a renderovať 3D objekty a scény. WebGL je cross-platformový, royalty-free webový štandard pre 3D grafické API (Application programming interface) založené na OpenGL ES 2.0. WebGL prináša 3D na webe bez plug-inov, je implementované priamo do prehliadača. WebGL je rozhranie pre webové aplikácie využívajúce hardvérovú akceleráciu pomocou grafických kariet podporujúcich OpenGL ES 2.0, teda nie sú to iba výkonné grafické karty desktopových PC ale aj grafické karty v mobilných PC aj menších mobilných zariadeniach, tabletoch a moderných smartphonoch. Spomedzi popredných webových prehliadačov je WebGL v súčasnosti podporované prehliadačmi Chrome, Firefox a Safari. WebGL umožňuje vývojárom webových stránok vytvárať napríklad hry, rôzne graficky bohaté webové aplikácie alebo aplikácie s inovatívnym používateľským rozhraním. Vývojári majú už v súčasnosti k dispozícii aj viaceré knižnice vyššej úrovne, respektíve ďalšie nástroje umožňujúce efektívne programovať a vytvárať webové aplikácie využívajúce WebGL, sú to napríklad C3DL, CopperLicht, EnergizeGL, GammaJS, GLGE, GTW, O3D, OSG.JS, SceneJS, SpiderGL, TDL, Three.js a X3DOM. Veľmi užitočné sú aj maticové knižnice: Sylvester, mjs a glMatrix. Všetko sú to JavaScriptové súbory [WebGL].

1.13 Premietanie (projekcia)

Premietanie je transformácia z n-rozmerného priestoru do m-rozmerného, kde m < n. V počítačovej grafike sa väčšinou jedná o premietanie z trojrozmerného priestoru (3D) do dvojrozmerného (2D). Priemetňou nazývame rovinu v priestore, do ktorej transformujeme (premietame) objekty. Najčastejšie sa používajú nasledujúce spôsoby premietania:

- rovnobežné
- stredové
- axonometrické

1.13.1 Rovnobežné premietanie

Rovnobežné premietanie je transformácia trojrozmerného priestoru do roviny. Pri tomto premietaní sú všetky premietacie lúče rovnobežné. Je zadané priemetňou (rovinou) a smerom premietania (vektorom), ktorý nemôže byť rovnobežný s priemetňou. Rovnobežné premietanie zachováva rovnobežnosť. Veľkosť priemetu nezávisí od vzdialenosti premietaného objektu od priemetne. Rovnobežné premietanie sa používa najmä v technických aplikáciách, kde je zachovanie rovnobežnosti výhodou. Podľa toho, aký uhol zvierajú premietacie lúče s priemetňou, rozlišujeme:

- kolmé premietanie (pravouhlé)
- šikmé (kosouhlé)

Najčastejšie sa používa **kolmé premietanie**, pri ktorom sú premietacie lúče kolmé na priemetňu. Metóda kolmého priemetu zanedbáva jednu zo súradníc.

Kolmé rovnobežné premietanie do roviny xy zanedbáva z-ovú súradnicu. Bodu P = (x, y, z) zodpovedá v priemete bod P’ = (x, y). Maticový zápis tejto transformácie je:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Takto získaný priemet (to čo dostaneme na priemetni) predstavuje pôdorys premietaného objektu. V prípade, že premietame do roviny xz a teda zanedbávame y-ovú súradnicu

premietaného objektu, dostaneme nárys objektu a v prípade premietania do roviny yz to bude bokorys.

Šikmé rovnobežné premietanie bodu (x, y, z) do roviny xy môžeme vyjadriť:

x' = x + z . **a** . cos α,

y' = y + z . **a** . sin α,

kde parameter **a** určuje predĺženie pre os z a uhol α odchýlku od osi x.

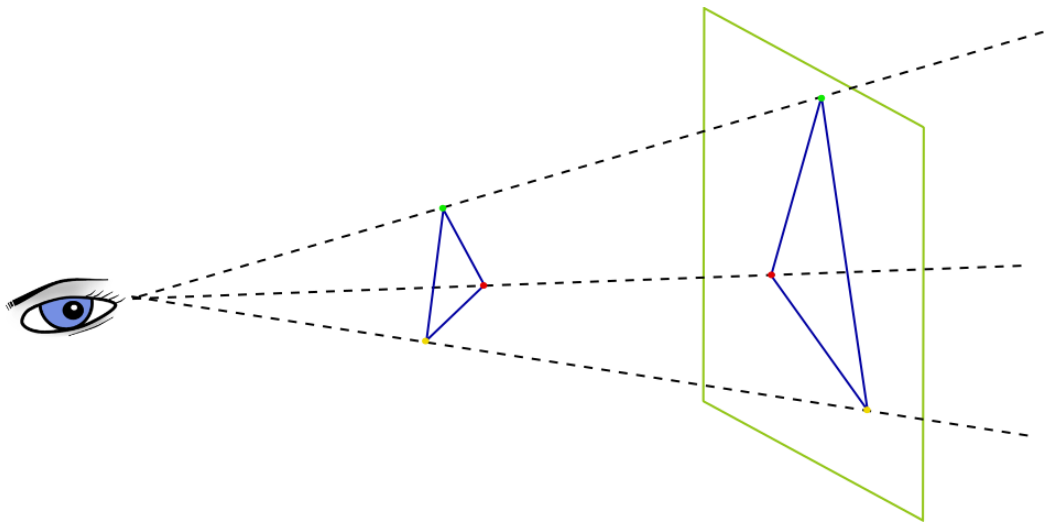
Ak parameter **a** = 0, tak ide o kolmé premietanie.

Šikmé premietanie do roviny xy môžeme vyjadriť nasledujúcou maticou:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ a.\cos\alpha & a.\sin\alpha & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

1.13.2 Stredové (perspektívne) premietanie

Stredové premietanie je dané priemetňou a stredom premietania. Pri tomto premietaní vychádzajú všetky premietacie lúče z jedného bodu, ktorý nazývame stredom premietania. Táto projekcia sa používa najmä v architektúre a všade, kde je potrebné realistické zobrazenie, pretože vytvára obrazy podobné tým, ktoré vidí ľudské oko v reálnom svete. Objekty, ktoré sú od stredu premietania vzdialenejšie, budú po premietnutí menšie. Tento typ premietania nezachováva rovnobežnosť úsečiek. Pôvodne rovnobežné úsečky sa môžu zbiehať.



Obr.15: Stredové premietanie

Majme bod v 3D so súradnicami (x, y, z), ktorý chceme zobraziť v 2D. Za priemetňu si zvolíme rovinu xy. Potom z-ová súradnica premietaného bodu je vzdialenosť tohto bodu od priemetne. Vzdialenosť oka (pozorovateľa) od priemetne označme **d**. Stredové premietanie bodu (x, y, z) do roviny xy môžeme potom vyjadriť:

x' = **d** . x / (**d** - z),

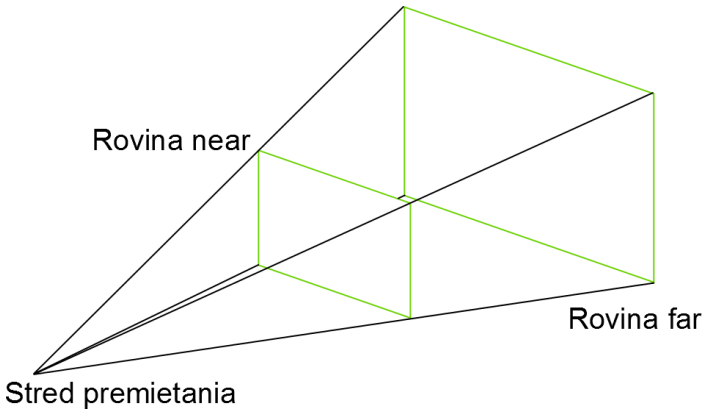
y' = **d** . y / (**d** - z)

Tieto rovnice predpokladajú, že oko pozorovateľa leží na osi z.

1.13.3 Pohľadový objem (View Frustum)

Pri zobrazovaní scény nezobrazujeme všetky objekty, ktoré sa v scéne nachádzajú, ale len tie objekty alebo časti objektov, ktoré sú viditeľné a pre pozorovateľa zaujímavé. Celú scénu teda orežeme tzv. pohľadovým objemom. Pri stredovom premietaní má pohľadový objem tvar zrezaného ihlana. Hranice ihlana tvorí:

- zorný uhol (uhol pri vrchole ihlana) – obyčajne 40°-60°
- rovina „far" – táto rovina odstraňuje objekty, ktoré sú od kamery príliš vzdialené
- rovina „near" – touto rovinou môžeme odstrániť objekty, ktoré sú príliš blízko a bránia vo výhľade



Obr.16: Pohľadový objem

[RTR]

1.14 MIP-mapping

Jedným zo základných problémov pri mapovaní textúr hlavne v časovo kritických aplikáciách je nutnosť častého zväčšovanie a zmenšovanie textúry, podľa toho, či je textúrovaný objekt blízko či ďaleko. Najmä pri aplikácii viac textúr môže táto operácia spotrebovávať podstatnú časť výpočtového výkonu, resp. brzdiť spracovanie grafických operácií. Jednou z možností ako sa tomu aspoň čiastočne vyhnúť je vopred spočítať a uložiť textúru v rôznych veľkostiach (rôznom rozlíšení). Táto technika sa nazýva MIP-mapping. Písmená "MIP" v názve sú akronymom latinskej frázy multum in parvo, ktorá znamená veľa v malom. Pri mapovaní sa potom vyberie najvhodnejšia textúra, prípadne sa zmenší alebo zväčší, avšak už nie z jediného pôvodného obrazu, ale z najbližšieho možného. Princíp MIP-mapping teda spočíva v tom, že namiesto opakovaného zmenšovanie textúry počas zobrazovania sa vykoná táto operácia raz pred jej použitím a výsledok sa uloží. Za zrýchlenie výpočtu sa samozrejme platí zväčšením priestoru, ktorý je potrebný pre uloženie textúry [Žára a kol.].

1.15 Plochy

Trojuholník je základný útvar slúžiaci pre vykresľovanie objektov v 3D scéne. Avšak objekty, ktoré sú vytvorené pri modelovaní scén môžu byť popísané rôznymi spôsobmi. Zakrivené povrchy možno presne popísať rovnicami. Prepočítaním týchto rovníc nájdeme množinu trojuholníkov, ktorú je potom možné vyrenderovať. Výhody použitia rovníc pre popis plôch:

- majú kompaktnejšiu reprezentáciu ako sady polygónov
- dajú sa jednoducho škálovať
- poskytujú hladší a kontinuálnejší povrch ako rovinné mnohouholníky
- animácie a detekcia kolízií sú jednoduchšie a rýchlejšie

Takáto reprezentácia plôch ponúka množstvo výhod pre realtime renderovanie. V prvom rade jej vďačíme za úsporu pamäte pre uloženie modelu. Toto je obzvlášť užitočné pre herné konzoly, ktoré majú zvyčajne menšiu pamäť v porovnaní s PC. Ďalšou veľkou výhodou týchto plôch je, že sú škálovateľné. Plochy môžeme popísať dvomi alebo aj dvesto trojuholníkmi. Záleží, ako detailne chceme plochu zobraziť. Keď je objekt so

zakriveným povrchom blízko, vytvoríme viac trojuholníkov z rovníc popisujúcich jeho tvar. Prípadne, keď chceme znížiť časovú náročnosť vykreslenia scény, môžeme znížiť počet trojuholníkov [Farin].

1.15.1 Parametrické plochy a ich vlastnosti

Parametrické plochy možno použiť na modelovanie objektov sa zakriveným povrchom. Parametrická plocha je definovaná niekoľkými riadiacimi vrcholmi.

Pod bodovou rovnicou parametrickej plochy $Q(u,v)$ dvoch parametrov u a v rozumieme funkciu $Q(u,v) = [x(u,v), y(u,v), z(u,v)]$, kde $x(u,v), y(u,v)$ a $z(u,v)$ sú funkcie dvoch parametrov u a $v \in \langle 0,1 \rangle$. Bod Q so súradnicami $[x,y,z]$ v trojrozmernom karteziánskom priestore má v parametrickom priestore súradnice $[u,v]$. Funkcie $x(u,v), y(u,v)$ a $z(u,v)$ sú obyčajne polynomiálne, s ohľadom na výhodné vlastnosti pri modelovaní a nadväzovaní.

Dotykový vektor $\vec{q}_u(u, v)$ v smere parametra *u* k ploche Q(*u, v*) a dotykový vektor $\vec{q}_v(u, v)$ v smere parametra *v* sú určené vzťahmi:

$$\vec{q}_u(u,v) = \frac{\partial Q(u,v)}{\partial u} = \left(\frac{\partial x(u,v)}{\partial u}, \frac{\partial y(u,v)}{\partial u}, \frac{\partial z(u,v)}{\partial u}\right) \quad (1)$$

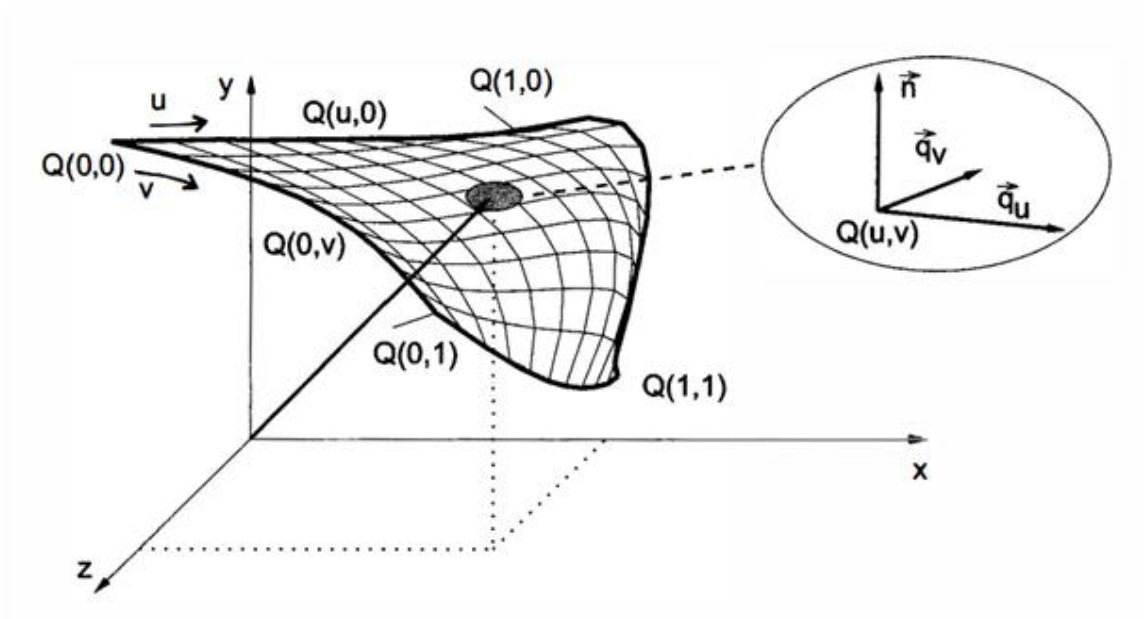
$$\vec{q}_v(u,v) = \frac{\partial Q(u,v)}{\partial v} = \left(\frac{\partial x(u,v)}{\partial v}, \frac{\partial y(u,v)}{\partial v}, \frac{\partial z(u,v)}{\partial v}\right) \quad (2)$$

Pri riešení úlohy nadväzovania plátov majú význam **skruty**, **skrutové vektory**, ktoré charakterizujú "vyklenutie" plochy v miestach napojenia. Sú definované pomocou zmiešaných parciálnych derivácií podľa parametrov *u* a *v* vzťahom:

$$\vec{q}_{uv}(u,v) = \frac{\partial^2 Q(u,v)}{\partial u \partial v} = \left(\frac{\partial^2 x(u,v)}{\partial u \partial v}, \frac{\partial^2 y(u,v)}{\partial u \partial v}, \frac{\partial^2 z(u,v)}{\partial u \partial v}\right) \quad (3)$$

Pre výpočet **normály** (kolmice) $\vec{n}$ k ploche v bode Q (lupa na obr. 17) využijeme poznatky o dotykových vektoroch $\vec{q}_u$, $\vec{q}_v$ a normálu určíme ako ich normalizovaný vektorový súčin:

$$\vec{n} = \frac{\vec{q}_u \times \vec{q}_v}{|\vec{q}_u \times \vec{q}_v|} \quad (4)$$



Obr.17: Parametrická plocha [Žára a kol.]

Hlavná krivka plochy v smere parametra *u* je každá krivka určená rovnicou Q(*u, k*) s pevným parametrom *v = k* a premenným parametrom *u* a analogicky hlavná krivka plochy v smere parametra *v* je každá krivka určená rovnicou Q(*k, v*) s pevným parametrom *u = k* a premenným parametrom *v*.

Rohy plochy sú body Q(0, 0), Q(1, 0), Q(0, 1) a Q(1, 1).

Strany plochy Q(*u, v*) sú hlavné krivky v smere *u* pre hodnoty *v* = 0, resp. *v* = 1 a v smere *v* pre hodnoty *u* = 0, resp. *u* = 1. Všetky strany plochy spolu tvoria jej **okraj**.

Podobne ako sa krivky pri nadväzovaní skladajú zo segmentov, tak aj plochy sa skladajú z častí, ktorým hovoríme **záplaty** (**patch**). Zložitejšie plochy získavame nadväzovaním záplat, prípadne delením plochy na záplaty, ktoré môžeme podrobnejšie tvarovať.

Plochy sa zadávajú **riadiacimi bodmi** alebo **bázovými funkciami**. Väčšina plôch, ktoré sa v trojrozmernom modelovaní používajú, sú plochy **aproximačné**. **Interpolácia** v dimenziách vyšších ako dva je prekvapivo zložitou úlohou, a preto sa pre interpoláciu riadiacich bodov v troch dimenziách (surface fitting alebo scattered data interpolation), zvyčajne používa interpolácia plôškami, najčastejšie trojuholníkmi. Pri aproximácii určuje poloha riadiacich bodov tvar výslednej plochy, táto plocha nimi však nemusí prechádzať. Pre nadväzovanie plôch je dôležité, aby boli známe dotykové podmienky na ich stranách.

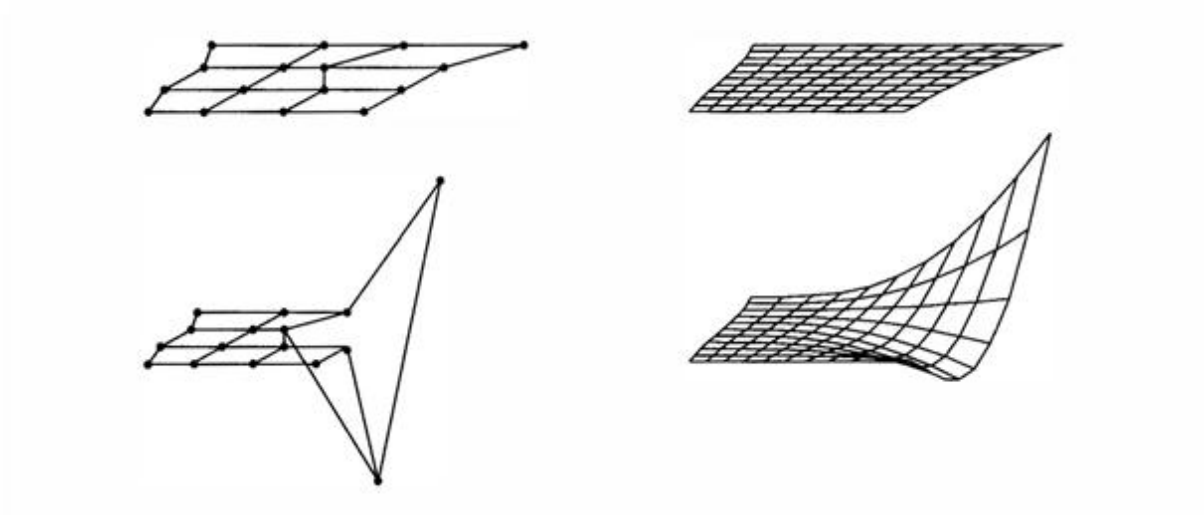
Bázové funkcie sú najčastejšie polynómy, pretože sú ľahko diferencovateľné a je možné ich rýchlo vyčísliť. Pri pomenovaní plôch sa používa stupeň použitého polynómu, a pretože sa jedná o plochy, je nutné v ich názve uviesť vždy dva údaje. Tak napríklad bikubická plocha má ako hlavné krivky v oboch smeroch **kubiky**, kvadraticko-lineárna plocha má ako hlavné krivky v smere parametra *u* **kvadriky**, zatiaľ čo hlavnými krivkami v smere parametra *v* sú úsečky. Najčastejšie sa používajú polynómy tretieho stupňa. Polynóm menšieho stupňa ako tri nie je možné určiť tak, aby pri prechádzaní dvomi bodmi mal zároveň "vhodné" dotykové vlastnosti. Kubika je polynóm najmenšieho stupňa, ktorý tieto vlastnosti má, umožňuje C^1 a C^2 spojitosť a poskytuje uspokojivé modelovacie možnosti. Použitie polynómov vyššieho stupňa vedie k zvýšeniu časovej náročnosti výpočtov. Pretože kubiky umožňujú C^2 spojitosť, je ľahké ich nadväzovať, teda skladať zložité plochy z jednoduchších (zvyčajne bikubických) plátov.

Pre parametrické plochy sa volí reprezentácia, v ktorej sa namiesto uchovávania koeficientov jednotlivých polynómov kombinujú riadiace body, dotykové vektory a ďalšie geometrické parametre s bázovými funkciami:

$$Q(u,v) = \mathbf{U}\mathbf{M}_B P \mathbf{M}_B^T \mathbf{V}^T = [u^3\ u^2\ u\ 1]\mathbf{M}_B P \mathbf{M}_B^T \begin{bmatrix} v^3 \\ v^2 \\ v \\ 1 \end{bmatrix}$$

Matica *P* obsahuje riadiace body a prípadne ďalšie prvky určujúce geometriu plochy, bázová matica $\mathbf{M}_B$ obsahuje koeficienty a_i polynómov bázových funkcií. Bázová matica $\mathbf{M}_B$ sa nemení, zatiaľ čo riadiace body $P_{i,j}$ svojou polohou v priestore určujú tvar plochy.

Modelovanie zvyčajne prebieha zadávaním a úpravami siete riadiacich bodov (obr. 18), ktorá tvorí v trojrozmernom priestore mnohosten. Zmenou polohy riadiacich bodov meníme tvar plochy.



Obr.18: Editácia Bézierovej bikubickej plochy (vpravo). Riadiaca sieť so šestnástimi bodmi (vľavo) určuje zmenu tvaru plochy po zmene polohy dvoch riadiacich bodov [Žára a kol.]

Medzi často požadované vlastnosti plôch patrí:

1. **Invariantnosť** k lineárnym transformáciám a projekciám, ktorá zaručuje, že napríklad otáčanie siete riadiacich bodov a následné generovanie plochy má rovnaký výsledok, ako otáčanie každého bodu z vygenerovanej plochy.
2. **Vlastnosť konvexnej obálky**
   (a) *silná podmienka* - plocha leží v konvexnej obálke všetkých svojich riadiacich bodov,
   (b) *slabá podmienka* - časť plochy leží v konvexnej obálke niektorých riadiacich bodov.
3. **Lokalita zmien** - zmenou polohy (pri racionálnych plochách aj váhy) riadiaceho bodu sa mení iba časť plochy, nie celá plocha.
4. Plocha môže **prechádzať krajnými bodmi** siete riadiacich bodov.

1.15.2 Tenzorovo-súčinové Bézierove záplaty

Niektoré modelovacie systémy používajú k reprezentácii povrchov telies Tenzorovo-súčinové Bézierove záplaty. Tieto plochy sú ľahko diferencovateľné, jednoducho a intuitívne sa modelujú a relatívne ľahko sa vypočítava priesečník s lúčom. Aj keď sa v súčasných systémoch používajú pre uchovanie plôch zložitejšie reprezentácie založené na technológii NURBS, tradičný používateľský pohľad a možnosť práce s Bézierovými modelovacími nástrojmi sú bežnou súčasťou týchto systémov. Je to

umožnené predovšetkým vďaka tomu, že Tenzorovo-súčinové Bézierove záplaty sú špeciálnym prípadom plôch NURBS. Tenzorovo-súčinová Bézierova záplata *m* x *n*-tého stupňa je určená (*m* + 1) x (*n* + 1) riadiacimi bodmi $P_{i,j}$ a vzťahom

$$Q(u,v) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{i,j} B_i^m(u) B_j^n(v),$$

kde $B_k^n(t) = \binom{n}{k} t^k (1-t)^{n-k}$ je *k*-ty (0 ≤ *k* ≤ *n*) Bernesteinov polynóm *n*-tého stupňa. Bernsteinove polynómy sú nezáporné a ich súčet je rovný jednej. Z týchto dvoch vlastností možno dokázať, že Tenzorovo-súčinová Bézierova záplata leží celá v konvexnom obale svojich riadiacich bodov. Strany Tenzorovo-súčinovej Bézierovej záplaty tvoria Bézierove krivky.

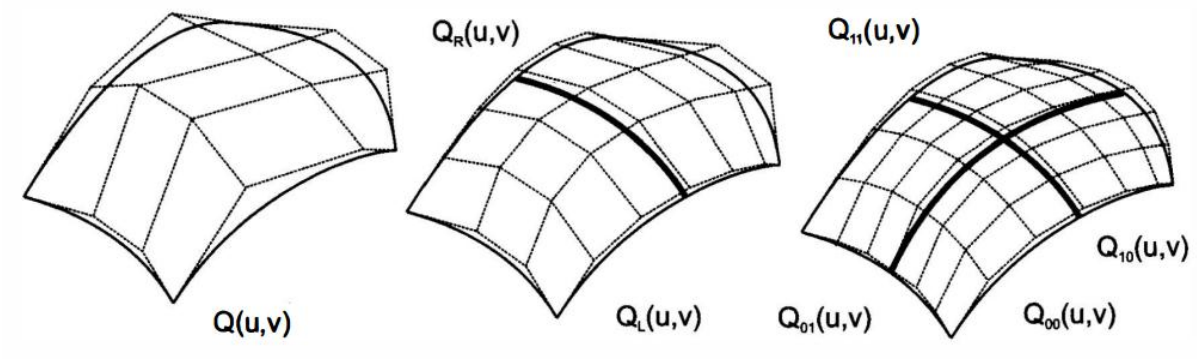
Pre hodnoty $Q(0,0), Q(1,0), Q(0,1)$ a $Q(1,1)$ sú rohy Tenzorovo-súčinovej Bézierovej záplaty totožné s bodmi $P_{0,0}, P_{1,0}, P_{0,1}$ a $P_{1,1}$ riadiacej siete. Tenzorovo-súčinová Bézierova záplata teda týmito bodmi prechádza.

Ďalšou vlastnosťou Tenzorovo-súčinovej Bézierovej záplaty je, že zmení celý svoj tvar pri zmene polohy jediného riadiaceho bodu. Táto vlastnosť je nevýhodná a je jedným z dôvodov, prečo sa Tenzorovo-súčinové Bézierove záplaty plátujú. Pri zmene polohy jedného riadiaceho bodu sa potom zmení tvar iba jediného plátu (ak nie je tento bod z dôvodov spojitosti zviazaný s bodmi zo susedného plátu).

Prevod Tenzorovo-súčinových Bézierových bikubických záplat na sieť trojuholníkov

Jednoduchou metódou zobrazovania Tenzorovo-súčinových Bézierových záplat je ich polygonizácia postupným dosadzovaním do rovnice Bézierovej záplaty za u a v pevným krokom Δu a Δv. V každom kroku týmto spôsobom získame štvorec určený rohmi Bézierovej plochy, ktorý môžeme rozdeliť na dva trojuholníky. Nevýhodou tohto postupu, podobne ako pri prevode kriviek na úsečky je, že nerešpektuje zakrivenie plochy.

Adaptívna metóda polygonizácie Bézierovej plochy pracuje na princípe rekurzívneho delenia plátu (patch splitting). Táto metóda využíva **de Casteljauov algoritmus**. Delenie plátu je schematicky znázornené na obr. 19.



Obr.19: Rekurzívne delenie Bézierovej bikubickej plochy [Žára a kol.]

Vstupom algoritmu je matica riadiacich bodov (pre prípad bikubického plátu je ich 16). V prvom kroku rozdelíme plochu v smere *u* na dve časti (obr. 19 uprostred) a týmto delením získame dve skupiny nových riadiacich bodov. Tým sme rozdelili plochu reprezentovanú riadiacimi bodmi z matice **P** na dve plochy Q_L a Q_R, ktoré sú reprezentované maticami $\mathbf{P}_L$ a $\mathbf{P}_R$. V druhom kroku aplikujeme predchádzajúci postup v smere *v* na matice $\mathbf{P}_L$ a $\mathbf{P}_R$, čím získame celkom štyri matice $\mathbf{P}_{00}$, $\mathbf{P}_{01}$, $\mathbf{P}_{10}$ a $\mathbf{P}_{11}$, obsahujúce celkom 4 x 16 bodov a zodpovedajúce plochy Q_{00}, Q_{01}, Q_{10} a Q_{11} (obr. 19 vpravo).

Po n deleniach získame z pôvodnej matice riadiacich bodov 4^n matíc, ktoré reprezentujú rovnakú plochu zloženú z častí. Riadiace siete uložené v maticiach predstavujú aproximáciu pôvodnej plochy. Tohto faktu využijeme pri jej zobrazení. Každú štvoricu susedných bodov rozdelíme na dva trojuholníky. Pre ďalšie operácie s týmito trojuholníkmi je vhodné ešte dopočítať hodnoty normálových vektorov v ich rohoch. Zostáva určiť kritériá pre zastavenie delenia plochy. Zjavne najjednoduchšie je určiť pevný počet delení *n* a tým v podstate stanoviť aj počet vygenerovaných trojuholníkov na hodnotu $2x4^n$.

Rekurzívny algoritmus rozdelenia Bézierovej bikubickej plochy:

```
DeleniePlátu(P)
Vstup: P - matica 4 x 4 riadiacich bodov
1. rozdeľ maticu P v smere parametra u (výsledkom sú dve matice PL a PR)
2. rozdeľ maticu PL v smere parametra v (výsledkom sú dve matice P00 a P01)
3. rozdeľ maticu PR v smere parametra v (výsledkom sú dve matice P10 a P11)
4. DeleniePlátu(P00); DeleniePlátu(P01)
5. DeleniePlátu(P10); DeleniePlátu(P11)
```

[Žára a kol.]

2 Špecifikácia

Cieľom mojej diplomovej práce je vytvorenie webového video prehrávača, ktorý spája video a 3D netradičným spôsobom. Video sa bude mapovať na Tenzorovo-súčinovú Bézierovu záplatu v 3D priestore. Takáto plocha je určená riadiacimi vrcholmi, teda bodmi, ktoré je možné posúvať v priestore a tým meniť tvar plochy, na ktorú sa mapuje video. V tejto kapitole sa budem venovať hardvérovým a softvérovým požiadavkám mojej aplikácie. Podrobnejšie opíšem požiadavky na funkcionalitu aplikácie, ktoré sú dané cieľom mojej práce.

2.1 Hardvérové a softvérové požiadavky

Aplikácia bude vyžadovať zariadenie s grafickou kartou, ktorá podporuje OpenGL ES 2.0, teda nebude určená iba pre desktopové PC ale aj pre mobilné PC aj menšie mobilné zariadenia, ako sú tablety a moderné smartphony. Aplikácia bude dostupná cez internet, to znamená, že zariadenie bude musieť mať pripojenie na internet. Všetky súbory sa budú nachádzať aj v CD prílohe tejto diplomovej práce. Druhou možnosťou, bude teda spustenie aplikácie pomocou servera (napr. WampServer). Server bude nutný z dôvodu, že uploadovanie videa bude sprostredkované pomocou PHP, čo je skriptovací jazyk na strane servera.

Medzi softvérové požiadavky patrí internetový prehliadač, ktorý podporuje HTML 5, CSS 3 a WebGL. Webový prehliadač musí podporovať nový <video> tag HTML 5 a videá vo formáte Ogg. Ďalej bude nutná podpora nového elementu HTML 5 <canvas>, pomocou ktorého sa implementuje WebGL do internetovej stránky. Medzi takéto internetové prehliadače patrí napríklad Chrome a Firefox.

Aplikácia bude vo veľkej miere naprogramovaná v JavaScripte, pomocou ktorého bude do aplikácie pridaná predovšetkým interaktivita. K WebGL sa dá v súčasnosti najjednoduchšie pristupovať taktiež práve pomocou JavaScriptu. JavaScript je interpretovaný jazyk (to znamená, že skripty sa spúšťajú bez predchádzajúcej kompilácie) a funguje vo všetkých významných internetových prehliadačoch, ako Internet Explorer, Firefox, Chrome, Opera a Safari. Na formátovanie obsahu stránky budú použité CSS štýly. CSS štýly podporujú všetky webové prehliadače, avšak niektoré nové funkcie CSS 3, ako

napríklad zaoblenie okrajov, ešte nie sú implementované vo všetkých prehliadačoch. Vo Firefoxe a v Chrome sú už tieto nové funkcie CSS 3 implementované.

2.2 Funkcionalita aplikácie

Aplikácia bude predovšetkým webový video prehrávač. Ovládanie videa bude podobné ako pri bežných video prehrávačoch. Používateľ bude mať možnosť pozastaviť video, spustiť prehrávanie videa od začiatku, alebo od akéhokoľvek zvoleného času. Video prehrávač bude zobrazovať časovú dĺžku už prehraného a celého videa. Používateľ si bude môcť nastaviť hlasitosť prehrávaného videa. Používateľ bude mať k dispozícii niekoľko videí na prehrávanie, ktoré bude možné vybrať kliknutím na ľubovoľný obrázok symbolizujúci dané video vo video galérii. Prípadne si používateľ bude môcť do prehrávača uploadnúť vlastné video. Podporované video formáty <video> elementom HTML 5 sú Ogg, MP4 a WebM. Všetky tieto video formáty podporuje iba prehliadač Chrome, ostatné prehliadače podporujú iba niektoré video formáty (napr. Firefox podporuje iba Ogg a WebM). Keďže sa bude jednať o upload na server, tak veľkosť uploadovaného súboru bude nutné prispôsobiť obmedzeniam tohto servera. Stránka bude na školskom serveri: st.fmph.uniba.sk.

Video prehrávač bude mať aj ďalšie netradičné funkcie. Video bude mapované na Tenzorovo-súčinovú Bézierovu záplatu, ktorá je vo všeobecnosti zadaná pomocou riadiacich vrcholov. Používateľ bude môcť meniť tvar plochy posúvaním týchto riadiacich vrcholov v priestore, to znamená, že bude umožnené posúvanie vo všetkých troch smeroch. V smere x-ovej a y-ovej osi sa bude vybraný bod posúvať držaním ľavého tlačidla myši a jej hýbaním po podložke. Pre posun riadiaceho vrchola v smere z-ovej osi bude nutné držať stlačené stredné tlačidlo myši (koliesko). Ďalšou interaktívnou vlastnosťou aplikácie bude tzv. zoomovanie alebo škálovanie, teda možnosť približovania a oddiaľovania plochy. Zoomovanie bude zabezpečené otáčaním kolieska myši. Plochu bude možné otáčať okolo zvislej alebo vodorovnej osi o zvolený uhol. Otáčanie okolo zvislej osi bude fungovať na základe stláčania šípky vľavo, respektíve vpravo na klávesnici. Každé stlačenie šípky na klávesnici otočí plochu o uhol zvolený používateľom. Otáčanie okolo vodorovnej osi bude fungovať obdobne, pre ovládanie budú slúžiť šípky hore a dole na klávesnici. Bude možné spustiť aj automatickú rotáciu okolo vodorovnej alebo zvislej osi.

Používateľ bude mať možnosť výberu z preddefinovaných tvarov plochy. Preddefinované tvary plochy budú napr. vypuklá alebo dutá plocha, valec, sud a iné. Ďalej bude možné nastaviť váhy pre jednotlivé riadiace vrcholy (ide o racionálnu Tenzorovo-súčinovú Bézierovu záplatu). Používateľ bude mať možnosť zvoliť si stupeň Tenzorovo-súčinovej Bézierovej záplaty a tzv. parameter "hladkosti". Hladkosť plochy závisí od zvoleného parametra - počet vzorkovacích bodov v jednom smere. Čím väčšie číslo, tým hladšie bude plocha pôsobiť.

3 Implementácia

V tejto kapitole sa budem venovať popisovaniu môjho postupu pri vytváraní aplikácie. Na začiatku mojej práce som sa zamerala na možnosti implementácie Tenzorovo-súčinovej Bézierovej záplaty na internete pomocou WebGL. Ďalším krokom bolo oboznámenie sa s postupom pri textúrovaní objektov v prostredí WebGL. Namapovaním videa na Tenzorovo-súčinovú Bézierovu záplatu som vytvorila základnú aplikáciu s minimálnou funkcionalitou. Následne som do aplikácie pridala ovládacie prvky pre video, vytvorila som video galériu, doplnila som možnosť voľby rôznych parametrov plochy a pridala som ďalšie prvky slúžiace na interakciu (ako napr. škálovanie, rotovanie, preddefinované tvary plochy a iné). Posledným krokom bola implementácia možnosti uploadovania vlastných videí. V tejto kapitole sa zameriam aj na dôvody vzniku komplikácií a chýb, ktoré sa vyskytli pri vytváraní aplikácie a na spôsob ich vyriešenia alebo odstránenia. V závere tejto kapitoly porovnám vytvorenú aplikáciu s navrhovanou aplikáciou, ktorá bola bližšie popísaná v predošlej kapitole. Vyhodnotím, či sú splnené požiadavky na funkcionalitu aplikácie stanovené cieľmi diplomovej práce.

3.1 Obsah elementu <canvas>

Hlavnou časťou body (tela stránky) je element <canvas>, teda priestor na renderovanie 3D grafiky. V elemente canvas je definované jeho id=”glcanvas”, rozmery a dve HTML DOM udalosti: `onmousedown` a `onmouseup`.

Udalosť `onmousedown` nastáva pri stlačení ľubovoľného tlačidla myši. Táto udalosť volá funkciu `mycanvasclick(event)`, ktorá zisťuje, či sa kliklo na nejaký riadiaci vrchol. Ak sa kliklo na nejaký riadiaci vrchol, tak do globálnej premennej `index`, vloží index vybraného vrchola, do premennej `drzi` vloží hodnotu 1 (znamená to, že sa drží stlačené tlačidlo myši). Do jednej z premenných `leftmb` alebo `middlemb` tiež vloží hodnotu 1. Slúžia na určenie, ktoré tlačidlo myši sa drží stlačené. Ak je stlačné ľavé tlačidlo myši, jej pohybom sa riadiaci vrchol hýbe v x-ovom a y-ovom smere. V prípade, že je stlačené stredné tlačidlo myši, je možné posúvať riadiaci vrchol v z-ovom smere.

Udalosť `onmouseup` nastáva pri pustení tlačidla myši. Táto udalosť volá funkciu `mycanvasup(event)`, ktorá iba vynuluje premenné `drzi`, `leftmb` a `middlemb`.

```
<body onload="start()">
<div>
 <div id="left">
  <canvas id="glcanvas" width="500" height="500"
      onmousedown="mycanvasclick(event)" onmouseup="mycanvasup(event)">
    Váš prehliadač nepodporuje HTML5 <code>&lt;canvas&gt;</code> element.
  </canvas>
```

Obr.20: Prvá časť tela stránky: element <canvas>

3.2 Príprava na renderovanie 3D scény

V elemente <body> je udalosť `onload`, ktorá nastáva pri načítaní tela stránky. Táto udalosť zavolá funkciu `start()`.

```
function start() {
  var canvas = document.getElementById("glcanvas");

  // Inicializácia kontextu GL
  gl = null;   // Inicializácia globálnej premennej gl
  try {
    gl = canvas.getContext("experimental-webgl");
  }
  catch(e) {}
  if (!gl) {
    alert("Nie je možné inicializovať WebGL. Váš prehliadač ho nepodporuje.");
  }

  // Pokračujeme iba ak je WebGL dostupné a funkčné
  if (gl) {
    gl.clearColor(0.0, 0.0, 0.0, 1.0);
    gl.enable(gl.DEPTH_TEST);
    gl.depthFunc(gl.LEQUAL);
    gl.clear(gl.COLOR_BUFFER_BIT|gl.DEPTH_BUFFER_BIT);
  }
}
```

Obr.21: Funkcia `start()` pripravuje WebGL kontext

Funkcia `start()` najprv načíta element <canvas> do premennej `canvas`. Ak chceme získať WebGL kontext pre canvas, od premennej `canvas` žiadame kontext s názvom "experimental-webgl". Ak sa to nepodarí, zobrazí sa upozornenie, aby používateľ vedel, že jeho prehliadač nepodporuje WebGL. V tomto bode je premenná `gl` buď null (čo znamená, že nie je WebGL kontext k dispozícii), alebo je odkazom na WebGL kontext, do ktorého budeme renderovať.

Ak je kontext úspešne inicializovaný, tak ho vyčistíme zvolenou farbou (ja som zvolila čiernu nepriesvitnú farbu) a zapneme testovanie hĺbky objektov v scéne, teda upresníme, že bližšie objekty budú zakrývať objekty, ktoré sú ďalej.

Potom, čo sme úspešne vytvorili WebGL kontext, môžeme do neho začať renderovať.

3.3 Základné osvetlenie scény

Na začiatok je potrebné vytvoriť shadere, ktoré osvetlia scénu a tým zviditeľnia objekty v nej. Shadere určujú, ako budú osvetlené objekty. Funkcia `initShaders()` spracováva túto úlohu. Shadere sú špecifikované pomocou tzv. OpenGL ES Shading Language.

```
function initShaders() {
  var fragmentShader = getShader(gl, "shader-fs");
  var vertexShader = getShader(gl, "shader-vs");

  // Vytvorenie shader programu
  var shaderProgram = gl.createProgram();
  gl.attachShader(shaderProgram, vertexShader);
  gl.attachShader(shaderProgram, fragmentShader);
  gl.linkProgram(shaderProgram);
  if (!gl.getProgramParameter(shaderProgram, gl.LINK_STATUS)) {
    alert("Nie je možné inicializovať shader program.");
  }
  gl.useProgram(shaderProgram);

  vertexPositionAttribute = gl.getAttribLocation(shaderProgram, "aVertexPosition");
  gl.enableVertexAttribArray(vertexPositionAttribute);

  textureCoordAttribute = gl.getAttribLocation(shaderProgram, "aTextureCoord");
  gl.enableVertexAttribArray(textureCoordAttribute);

  vertexNormalAttribute = gl.getAttribLocation(shaderProgram, "aVertexNormal");
  gl.enableVertexAttribArray(vertexNormalAttribute);

  vertexColorAttribute = gl.getAttribLocation(shaderProgram, "aVertexColor");
  gl.enableVertexAttribArray(vertexColorAttribute);
}
```

Obr.22: Funkcia `initShaders()` inicializuje shadere

Funkcia `initShaders()` najprv načíta dva shader programy, prvý, `fragment shader`, je načítaný z elementu s ID "shader-fs". Druhý, `vertex shader`, je načítaný z elementu s ID "shader-vs". Potom vytvoríme `shader program` volaním WebGL objektu `createProgram()`, pripojíme naňho dva shadere a prilinkujeme ho k `gl` objektu. Skontrolujeme parameter LINK_STATUS `gl` objektu, aby sme si boli istí, že sa

program úspešne pripojil a ak áno, aktivujeme nový `shader program`. Následne vytvoríme polia pre ukladanie súradníc bodov, pre normálové vektory, textúru a farby. Využijeme pritom preddefinované WebGL funkcie.

3.3.1 Načítanie shaderov z DOM-u

Funkcia `getShader()` použitá v predošlej časti vyvolá `shader program` so zadaným názvom z DOM, vráti skompilovaný `shader program`, alebo null, ak sa nedá načítať alebo skompilovať.

```
function getShader(gl, id) {
    var shaderScript, theSource, currentChild, shader;

    shaderScript = document.getElementById(id);

    if (!shaderScript) {
        return null;
    }

    theSource = "";
    currentChild = shaderScript.firstChild;

    while(currentChild) {
        if (currentChild.nodeType == currentChild.TEXT_NODE) {
            theSource += currentChild.textContent;
        }
```

Obr.23: Prvá časť funkcie `getShader`

Akonáhle je nájdený element so zadaným ID, načíta sa jeho obsah do premennej `theSource`.

```
if (shaderScript.type == "x-shader/x-fragment") {
    shader = gl.createShader(gl.FRAGMENT_SHADER);
} else if (shaderScript.type == "x-shader/x-vertex") {
    shader = gl.createShader(gl.VERTEX_SHADER);
} else {
    // Neznámy typ shadera
    return null;
}
```

Obr.24: Druhá časť funkcie `getShader`

Po načítaní kódu shadera sa pozrieme na typ shadera pre určenie, či je to vertex shader ("x-shader/x-vertex"), alebo fragment shader ("x-shader/x-fragment"), potom vytvoríme vhodný typ shadera z načítaného zdrojového kódu.

```
gl.shaderSource(shader, theSource);

  gl.compileShader(shader);

  if (!gl.getShaderParameter(shader, gl.COMPILE_STATUS)) {
      alert("Objavila sa chyba pri kompilovaní shaderov: "
       + gl.getShaderInfoLog(shader));
      return null;
  }

  return shader;
}
```

Obr.25: Posledná časť funkcie `getShader`

Nakoniec príde premenná `theSource` do shadera a skompiluje sa. Pokiaľ dôjde k chybe pri kompilácii shadera, zobrazí sa varovanie a funkcia getShader vráti hodnotu null, inak vráti novo skompilovaný shader.

3.3.2 Shadere

Ešte je treba pridať samotné shader programy, ktoré sú napísané v OpenGL ES Shading Language (už to nie je JavaScript, preto sú v súbore index.php).

Fragment shader

Každý pixel v polygóne sa v žargóne GL nazýva fragment. Úlohou fragment shadera je stanoviť farbu pre každý pixel.

```
<script id="shader-fs" type="x-shader/x-fragment">
  void main(void) {
    gl_FragColor = vec4(1.0, 1.0, 1.0, 1.0);
  }
</script>
```

Obr.26: Základný fragment shader

V tomto prípade je nastavená biela farba pre každý pixel. V ďalších kapitolách to ešte zmeníme.

Vertex shader

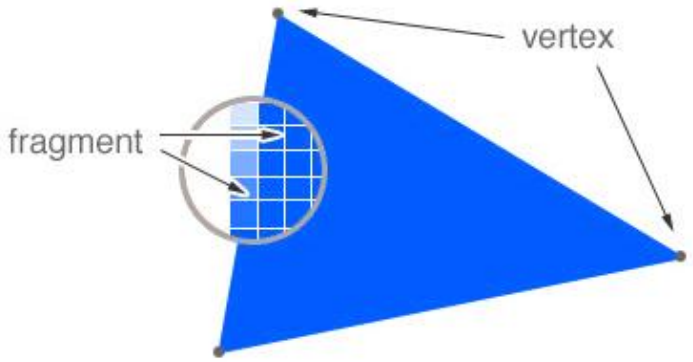
Vertex shader definuje pozíciu každého vrchola.

```
<script id="shader-vs" type="x-shader/x-vertex">
  attribute vec3 aVertexPosition;

  uniform mat4 uMVMatrix;
  uniform mat4 uPMatrix;

  void main(void) {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
  }
</script>
```

Obr.27: Základný vertex shader



Obr.28: Fragmenty (pixle) a vertexy (vrcholy) polygónu

3.4 Vytvorenie jednoduchého objektu

Predtým ako budeme môcť vyrenderovať nejaký objekt, musíme vytvoriť buffer, ktorý bude obsahovať vrcholy tohto objektu. Na to slúži funkcia `initBuffers()`.

```
function initBuffers() {
  squareVerticesBuffer = gl.createBuffer();
  gl.bindBuffer(gl.ARRAY_BUFFER, squareVerticesBuffer);

  var vertices = [
    1.0,  1.0,  0.0,
    -1.0, 1.0,  0.0,
    1.0,  -1.0, 0.0,
    -1.0, -1.0, 0.0
  ];

  gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
}
```

Obr.29: Funkcia `initBuffers()`

Funkcia `initBuffers()` na začiatku volá metódu `createBuffer()` `gl` objektu na získanie buffera, do ktorého budeme ukladať vrcholy objektu (v tomto prípade štvorca). Tento buffer potom zviažeme s kontextom volaním metódy `bindBuffer()`. Ďalej vytvoríme JavaScriptové pole obsahujúce súradnice každého vrchola objektu. To je potom prevedené do poľa floatov WebGL a presunuté do metódy `bufferData()` `gl` objektu, čím sa určia vrcholy pre daný objekt.

3.5 Vykreslenie scény

Akonáhle sú zriadené shadere a je skonštruovaný objekt, môžeme vyrenderovať scénu.

```
function drawScene() {
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  perspectiveMatrix = makePerspective(45, 500.0/500.0, 0.1, 100.0);

  loadIdentity();
  mvTranslate([0.0, 0.0, -6.0]);

  gl.bindBuffer(gl.ARRAY_BUFFER, squareVerticesBuffer);
  gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
  setMatrixUniforms();
  gl.drawArrays(gl.TRIANGLE_STRIP, 0, 4);
}
```

Obr.30: Funkcia `drawScene()`

Prvým krokom je vyčistenie / prefarbenie kontextu na našu farbu pozadia, potom nastavíme perspektívu kamery (pohľadový objem viď kap. 1.13.3). Zorné pole (field of

view) je nastavené na 45°, ďalej pomer šírky k výške je 500/500 (rozmery canvasu) a ešte určíme, že kamerou chceme vidieť len objekty, ktoré sú vzdialené aspoň 0,1 jednotiek a maximálne 100 jednotiek. Nechceme vidieť objekty, ktoré sú príliš blízko, lebo by nám bránili vo výhľade a ani objekty, ktoré sú príliš ďaleko, lebo tie nie sú pre nás zaujímavé a boli by veľmi malé.

Ďalším krokom je načítanie jednotkovej matice. Potom nastavíme polohu objektu posunutím o 6 jednotiek ďalej od kamery. Následne zviažeme `vertex buffer` objektu s kontextom a vykreslíme objekt volaním funkcie `drawArrays()`.

3.6 Matrix utility operácie

Funkcia `drawScene()` používa niekoľko funkcií pre prácu s maticami, ako `loadIdentity()`, `mvTranslate([0.0, 0.0, -6.0])` a `setMatrixUniforms()`. Keďže WebGL je nízko-úrovňové API, nemá v sebe preddefinované takéto funkcie. Všetko treba napísať v JavaScripte. Veľkú časť práce si je možné uľahčiť použitím externej JavaScriptovej knižnice `Sylvester.js`, ktorá definuje vektory, matice a rôzne operácie s nimi. Ďalší súbor `glUtils.js`, ktorý používame, je tiež JavaScriptová knižnica. Táto knižnica zjednodušuje používanie knižnice `Sylvester.js` tak, ako pri pridávaní metód pre budovanie špeciálnych typov matíc (napr. funkcie: `Matrix.Translation` a `Matrix.Scale`), rovnako tak aj pri výstupe vo formáte HTML pre ich zobrazenie (napr. funkcie: `makePerspective` a `makeFrustum`). Pozrime sa teraz bližšie na funkcie spomenuté v úvode tohto odseku, ktoré používa funkcia `drawScene()`.

```
function loadIdentity() {
  mvMatrix = Matrix.I(4);
}
```

Obr.31: Funkcia `loadIdentity()`

Funkcia `loadIdentity()` vloží do globálnej premennej `mvMatrix` (model view matica) štvorrozmernú jednotkovú maticu.

```
function multMatrix(m) {
  mvMatrix = mvMatrix.x(m);
}

function mvTranslate(v) {
  multMatrix(Matrix.Translation($V([v[0], v[1], v[2]])).ensure4x4());
}
```

Obr.32: Funkcie `multMatrix(m)`, `mvTranslate(v)`

Funkcia `multMatrix(m)` vynásobí `mvMatrix` maticou `m` (vstupný parameter), výsledok tohto násobenia matíc vloží do `mvMatrix`.

Funkcia `mvTranslate(v)` zavolá funkciu `multMatrix`, ako parameter nestačí dať vektor `v`, najprv ho treba previesť na štvorrozmernú maticu posunutia.

```
function setMatrixUniforms() {
  var pUniform = gl.getUniformLocation(shaderProgram, "uPMatrix");
  gl.uniformMatrix4fv(pUniform, false, new Float32Array(perspectiveMatrix.flatten()));

  var mvUniform = gl.getUniformLocation(shaderProgram, "uMVMatrix");
  gl.uniformMatrix4fv(mvUniform, false, new Float32Array(mvMatrix.flatten()));

  var normalMatrix = mvMatrix.inverse();
  normalMatrix = normalMatrix.transpose();
  var nUniform = gl.getUniformLocation(shaderProgram, "uNormalMatrix");
  gl.uniformMatrix4fv(nUniform, false, new Float32Array(normalMatrix.flatten()));
}
```

Obr.33: Funkcia `setMatrixUniforms()`

Funkcia `setMatrixUniforms()` hovorí WebGL, aby načítalo našu súčasnú model view maticu (`mvMatrix`) a tiež projekčnú maticu (`perspectiveMatrix`). Tento krok je nutný, pretože ani jedna z týchto matíc nie je vstavená do WebGL. Spôsob, ako sa na to pozerať je, že môžeme robiť všetky pohyby objektov zmenou premennej `mvMatrix`, ktoré chceme, ale všetko sa to odohráva v súkromnom priestore JavaScriptu. Funkcia `setMatrixUniforms()` presunie obe matice do grafickej karty.

3.7 Vytvorenie zložitejšieho objektu

Plocha, na ktorú sa bude mapovať video, nemá byť iba obyčajný štvorec (viď kap. 3.4), ale má to byť Tenzorovo-súčinová Bézierova záplata. Je to plocha určená riadiacimi vrcholmi, ich posúvaním sa dá meniť jej tvar. Preto nie je možné definovať vrcholy priamo, ale pomocou premenných.

```
patchVerticesBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, patchVerticesBuffer);
var vertices = new Array();
var ind = 0;
var indexSample = 0;
  for (j = 0; j < samples-1; j++) {
    for (i = 0; i < samples-1; i++) {
      indexSample = j * samples + i;
      vertices[ind] = fSamples[indexSample].x;
      vertices[ind+1] = fSamples[indexSample].y;
      vertices[ind+2] = fSamples[indexSample].z;

      vertices[ind+3] = fSamples[indexSample+1].x;
      vertices[ind+4] = fSamples[indexSample+1].y;
      vertices[ind+5] = fSamples[indexSample+1].z;

      vertices[ind+6] = fSamples[indexSample+samples].x;
      vertices[ind+7] = fSamples[indexSample+samples].y;
      vertices[ind+8] = fSamples[indexSample+samples].z;

      vertices[ind+9] = fSamples[indexSample+samples+1].x;
      vertices[ind+10] = fSamples[indexSample+samples+1].y;
      vertices[ind+11] = fSamples[indexSample+samples+1].z;
      ind = ind + 12;
    }
  }
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertices), gl.STATIC_DRAW);
```

Obr.34: Vzorky Tenzorovo-súčinovej Bézierovej záplaty

`fSamples` je pole vzoriek vypočítaných pomocou de Casteljauovho algoritmu z poľa riadiacich vrcholov (viac v nasledujúcej kap. 3.8). Vrcholy a aj vzorky tvoria sieť, ktorú si môžeme predstaviť, ako maticu. Avšak `fSamples` je jednorozmerné pole, preto prvky tohto poľa sú uložené za sebou, tak ako idú po riadkoch zdola nahor. Do poľa vertices ich chceme uložiť postupne tak, ako je vždy pri sebe štvorica týchto vrcholov (dva z jedného riadku a dva z riadku nad ním). Každú štvoricu ďalej rozdelíme na dva trojuholníky.

```
patchVerticesIndexBuffer = gl.createBuffer();
gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, patchVerticesIndexBuffer);
var patchVertexIndices = new Array();
var indInd = 0;
for (i = 0; i < (samples-1)*(samples-1); i++) {
  patchVertexIndices[indInd] = 0+4*i;
  patchVertexIndices[indInd+1] = 1+4*i;
  patchVertexIndices[indInd+2] = 3+4*i;
  patchVertexIndices[indInd+3] = 0+4*i;
  patchVertexIndices[indInd+4] = 3+4*i;
  patchVertexIndices[indInd+5] = 2+4*i;
  indInd = indInd + 6;
}
gl.bufferData(gl.ELEMENT_ARRAY_BUFFER,
 new Uint16Array(patchVertexIndices), gl.STATIC_DRAW);
```

Obr.35: Indexy vrcholov

Pole `patchVertexIndices` definuje každú štvoricu vrcholov ako dva trojuholníky, špecifikovaním jednotlivých vrcholov trojuholníka pomocou indexov v poli vzoriek záplaty. Týmto spôsobom popíšeme záplatu ako súbor trojuholníkov.

Ďalej musíme pridať kód do našej funkcie `drawScene()`, aby použila `index buffer` pre záplatu, pridaním ďalšieho `bindBuffer`-a. Záplatu vykreslíme pomocou funkcie `drawElements()`.

```
function drawScene() {
  gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT);

  perspectiveMatrix = makePerspective(45, 640.0/480.0, 0.1, 100.0);

  loadIdentity();
  mvTranslate([-0.0, 0.0, -6.0]);

  gl.bindBuffer(gl.ARRAY_BUFFER, patchVerticesBuffer);
  gl.vertexAttribPointer(vertexPositionAttribute, 3, gl.FLOAT, false, 0, 0);
  gl.bindBuffer(gl.ELEMENT_ARRAY_BUFFER, patchVerticesIndexBuffer);
  setMatrixUniforms();
  gl.drawElements(gl.TRIANGLES, 6*(samples-1)*(samples-1), gl.UNSIGNED_SHORT, 0);
}
```

Obr.36: Zmena a doplnenie funkcie `drawScene()`

3.8 Algoritmus de Casteljau

Algoritmus de Casteljau je vložený do funkcie `recompute()`, ktorá sa volá vo funkcii `start()` hneď za definovaním súradníc riadiacich vrcholov záplaty. Riadiace

vrcholy záplaty sú na začiatku rovnomerne rozmiestnené do siete (matice) tvoriacej obdĺžnik v rovine. Keďže sa jedná o racionálnu Tenzorovo-súčinovú Bézierovu záplatu, prvým krokom je prenásobenie súradníc vrcholov vlastnou váhou (váha je akoby štvrtou súradnicou každého vrchola). Algoritmus de Casteljau (viď kap. 1.15.2) opakujeme pre toľko dvojíc parametrov `u` a `v`, koľko je druhá mocnina počtu vzoriek v jednom smere. A to je aj počet vypočítaných vzoriek (prvky poľa `fSamples` použitého v kap.3.7), pomocou ktorých vykreslíme záplatu. Tento algoritmus má v zdrojovom kóde mojej aplikácie viac ako 150 riadkov, preto nie je vhodné vkladať ho do tejto písomnej časti diplomovej práce. Čitateľ si zdrojový kód môže otvoriť priamo z CD prílohy. Týmto algoritmom je možné vypočítať pre každú vzorku deriváciu v smere `u` aj v smere `v`. Vektorovým súčinom týchto derivácií dostaneme normálový vektor v danej vzorke. Na konci algoritmu zase predelíme jednotlivé súradnice vrcholov ich vlastnou váhou.

3.9 Osvetlenie vo WebGL

WebGL na rozdiel od širšieho štandardu OpenGL nemá vlastnú podporu pre osvetlenie. To znamená, že osvetlenie musíme urobiť sami. V aplikácii je pre zjednodušenie osvetľovacieho modelu použité len jednoduché priame (directional) a okolité (ambient) osvetlenie. Pre implementáciu priameho svetla potrebujeme vedieť dva typy informácií:

- Musíme poznať povrchovú normálu každého vrcholu. Normála, alebo normálový vektor je vektor, ktorý je kolmý na plochu v danom bode.
- Potrebujeme poznať smer, ktorým sa vyžaruje svetlo. Smer je definovaný smerovým vektorom (direction vector).

3.9.1 Definovanie normál pre jednotlivé vzorky

Do funkcie `initBuffers()` pridáme pre každú vzorku k nej prislúchajúci normálový vektor z poľa `fNormals`, ktorý sme vypočítali pomocou de Casteljaouvho algoritmu.

```
patchVerticesNormalBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, patchVerticesNormalBuffer);
var vertexNormals = new Array();
var ind = 0;
var indexSample = 0;
  for (j = 0; j < samples-1; j++) {
    for (i = 0; i < samples-1; i++) {
      indexSample = j * samples + i;
      vertexNormals[ind] = fNormals[indexSample].x;
      vertexNormals[ind+1] = fNormals[indexSample].y;
      vertexNormals[ind+2] = fNormals[indexSample].z;

      vertexNormals[ind+3] = fNormals[indexSample+1].x;
      vertexNormals[ind+4] = fNormals[indexSample+1].y;
      vertexNormals[ind+5] = fNormals[indexSample+1].z;

      vertexNormals[ind+6] = fNormals[indexSample+samples].x;
      vertexNormals[ind+7] = fNormals[indexSample+samples].y;
      vertexNormals[ind+8] = fNormals[indexSample+samples].z;

      vertexNormals[ind+9] = fNormals[indexSample+samples+1].x;
      vertexNormals[ind+10] = fNormals[indexSample+samples+1].y;
      vertexNormals[ind+11] = fNormals[indexSample+samples+1].z;
      ind = ind + 12;
    }
  }
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(vertexNormals), gl.STATIC_DRAW);
```

Obr.37: Doplnenie poľa normál do funkcie `initBuffers()`

Ďalej je treba pridať kód do funkcie `drawScene()`, aby sme zviazali pole normál so shader atribútom, aby shader kód k nim mohol získať prístup:

```
gl.bindBuffer(gl.ARRAY_BUFFER, patchVerticesNormalBuffer);
gl.vertexAttribPointer(vertexNormalAttribute, 3, gl.FLOAT, false, 0, 0);
```

Obr.38: Doplnenie normál do funkcie `drawScene()`

3.9.2 Aktualizácia shaderov

Všetky údaje, ktoré shadere potrebujú, sú im k dispozícii, teraz je potrebné aktualizovať kód v samotných shaderoch.

Vertex shader

Prvá vec, ktorú je treba urobiť, je aktualizovať vertex shader tak, aby generoval shading hodnotu pre každú vzorku. Táto hodnota vychádza z ambientného osvetlenia a rovnako aj z priameho osvetlenia.

```
<script id="shader-vs" type="x-shader/x-vertex">
  attribute highp vec3 aVertexNormal;
  attribute highp vec3 aVertexPosition;
  attribute highp vec2 aTextureCoord;

  uniform highp mat4 uNormalMatrix;
  uniform highp mat4 uMVMatrix;
  uniform highp mat4 uPMatrix;

  varying highp vec2 vTextureCoord;
  varying highp vec3 vLighting;

  void main(void) {
    gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);
    vTextureCoord = aTextureCoord;

    // Apply lighting effect

    highp vec3 ambientLight = vec3(0.6, 0.6, 0.6);
    highp vec3 directionalLightColor = vec3(0.5, 0.5, 0.75);
    highp vec3 directionalVector = vec3(0.85, 0.8, 0.75);

    highp vec4 transformedNormal = uNormalMatrix * vec4(aVertexNormal, 1.0);

    highp float directional = max(dot(transformedNormal.xyz, directionalVector), 0.0);
    vLighting = ambientLight + (directionalLightColor * directional);
  }
</script>
```

Obr.39: Zmenený vertex shader

Akonáhle poznáme pozíciu vrchola a dostaneme súradnice texelu zodpovedajúce vrcholu, môžeme pracovať na výpočte tieňovania pre tento vrchol. Ako prvé treba transformovať normálu založenú na aktuálnej pozícii a orientácii záplaty vynásobením normály danej vzorky normálovou maticou. Množstvo priameho svetla, ktoré má byť aplikované na vzorku môžeme vypočítať pomocou skalárneho súčinu (dot product) transformovanej normály a smerového vektora priameho osvetlenia. Ak je výsledok skalárneho súčinu záporný tak, ako výsledok použijeme nulovú hodnotu.

Keď už máme vypočítané množstvo priameho osvetlenia, môžeme vypočítať osvetľovaciu hodnotu (premenná `vLighting`) sčítaním ambientného svetla a súčinu farby priameho svetla a množstva priameho osvetlenia. Ako výsledok máme RGB hodnotu, ktorá bude použitá fragment shaderom na pridanie farby pre každý pixel, ktorý renderujeme.

Fragment shader

Fragment shader je tiež potrebné aktualizovať, aby bral do úvahy osvetľovaciu hodnotu vypočítanú vertex shaderom.

```
<script id="shader-fs" type="x-shader/x-fragment">
  varying highp vec2 vTextureCoord;
  varying highp vec3 vLighting;

  uniform sampler2D uSampler;

  void main(void) {
    vec4 texelColor = texture2D(uSampler, vec2(vTextureCoord.s, vTextureCoord.t));
    gl_FragColor = vec4(texelColor.rgb * vLighting, texelColor.a);
  }
</script>
```

Obr.40: Zmenený fragment shader

Najprv načítame farbu texelu, ale pred nastavením farby fragmentu, vynásobíme farbu texelu osvetľovacou hodnotou (premenná `vLighting`) pre nastavenie farby pixla s ohľadom na vplyv svetelných zdrojov.

3.10 Textúra

V predchádzajúcich ukážkach shaderov, už boli urobené zmeny potrebné kvôli textúre. Vo vertex shaderi bolo treba nahradiť dáta obsahujúce farbu dátami obsahujúcimi súradnice textúry (texely). Texel indikuje miesto v textúre korešpondujúce k danej vzorke. Vo fragment shaderi namiesto priradenia farebnej hodnoty bolo treba priradiť texel.

3.10.1 Mapovanie textúry

Pre použitie textúry je potrebné namapovať ju na jednotlivé vzorky záplaty. To urobíme vo funkcii `initBuffers()`.

```
patchVerticesTextureCoordBuffer = gl.createBuffer();
gl.bindBuffer(gl.ARRAY_BUFFER, patchVerticesTextureCoordBuffer);
var textureCoordinates = new Array();
var indTex = 0;
var indexSampleTex = 0;
  for (j = 0; j < (samples-1); j++) {
    for (i = 0; i < (samples-1); i++) {
      indexSampleTex = j * samples + i;
      textureCoordinates[indTex] = fTexels[indexSampleTex].x;
      textureCoordinates[indTex+1] = fTexels[indexSampleTex].y;

      textureCoordinates[indTex+2] = fTexels[indexSampleTex+1].x;
      textureCoordinates[indTex+3] = fTexels[indexSampleTex+1].y;

      textureCoordinates[indTex+4] = fTexels[indexSampleTex+samples].x;
      textureCoordinates[indTex+5] = fTexels[indexSampleTex+samples].y;

      textureCoordinates[indTex+6] = fTexels[indexSampleTex+samples+1].x;
      textureCoordinates[indTex+7] = fTexels[indexSampleTex+samples+1].y;
      indTex = indTex + 8;
    }
  }
gl.bufferData(gl.ARRAY_BUFFER, new Float32Array(textureCoordinates), gl.STATIC_DRAW);
```

Obr.41: Doplnenie mapovania textúry do funkcie `initBuffers()`

Najprv vytvoríme `gl` buffer `patchVerticesTextureCoordBuffer`, do ktorého budeme ukladať súradnice textúry pre každú štvoricu vzoriek, potom pripojíme tento buffer ako pole, do ktorého budeme zapisovať.

Pole `textureCoordinates` definuje súradnice textúry korešpondujúce ku každej vzorke na záplate. Súradnice textúry sú definované v poli `fTexels`, ktoré naplníme pri volaní funkcie `recompute()`, teda hneď po načítaní stránky. Súradnice textúry sú rovnomerne rozložené a ich počet musí byť rovnaký ako počet vzoriek, preto do poľa `fTexels` rovnomerne načítame dvojrozmernú mriežku súradníc. Pre účely mapovania textúry je rozsah súradníc textúry vždy od 0.0 do 1.0, rozmery textúry sú normalizované bez ohľadu na ich aktuálnu veľkosť.

```
for (var i=0; i<samples; i++) {
  for (var j=0; j<samples; j++) {
    fTexels[pocetTV] = {x: 1.0/samples*j, y: 1.0/samples*i};
    pocetTV = pocetTV+1;
  }
}
pocetTV = 0;
```

Obr.42: Naplnenie poľa `fTexels`

3.10.2 Načítanie video textúry

Prvým krokom je pridanie <video> elementu do súboru index.php pre načítanie snímok videa.

```
<video id="video" src="kockac.ogv" poster="kockac.jpg">
  Váš prehliadač nepodporuje HTML5 <code>&lt;video&gt;</code> element.
</video>
```

Obr.43: <video> element v HTML kóde

Aby sa video nezobrazilo priamo na stránke, v CSS štýloch pridáme <video> elementu atribút: display: none.

V súbore s JavaScriptovými funkciami definujeme premennú `myMovie` a vkladáme do nej obsah video elementu s id=”video”:

```
myMovie = document.getElementById('video');
```

Obr.44: Načítanie <video> elementu do premennej `myMovie`

Nechceme, aby sa začalo s prehrávaním videa, kým nie je načítaná dostatočne veľká časť videa, ktorá by mohla byť prehrávaná bez prerušovania. Preto je videu pridaný listener udalosti `canplaythrough`. Táto udalosť nastane práve vtedy, keď je načítaná dostatočne veľká časť videa. Listener potom zavolá funkciu `startVideo()`. Videu je pridaný ešte jeden listener udalosti `ended`. Táto udalosť nastane, keď video skončí prehrávanie. Vtedy listener zavolá funkciu `videoDone()`.

```
myMovie.addEventListener("canplaythrough", startVideo, true);
myMovie.addEventListener("ended", videoDone, true);
```

Obr.45: Pridanie listenerov premennej myMovie

Funkcia `startVideo()` spustí prehrávanie videa a nastaví interval volania funkcie `drawScene()` na každých 15 milisekúnd. Do budúcna by bolo lepšie nastaviť túto konštantu v závislosti na frame rate konkrétneho videa. V súčasnosti HTML 5 element <video> nemá takýto atribút.

```
function startVideo() {
  myMovie.play();
  intervalID = setInterval(drawScene, 15);
}
```

Obr.46: Funkcia `startVideo()`

Funkcia `videoDone()` jednoducho zavolá funkciu `clearInterval()`, aby sa už scéna neprekresľovala.

Ďalším krokom je inicializácia textúry:

```
function initTextures() {
  patchTexture = gl.createTexture();
}
```

Obr.47: Funkcia `initTexture()`

Funkcia `initTexture()` vytvorí prázdny textúrový objekt pre neskoršie použitie vo funkcii `updateTexture()`:

```
function updateTexture() {
  gl.bindTexture(gl.TEXTURE_2D, patchTexture);
  gl.pixelStorei(gl.UNPACK_FLIP_Y_WEBGL, true);

  gl.texImage2D(gl.TEXTURE_2D, 0, gl.RGBA, gl.RGBA, gl.UNSIGNED_BYTE, myMovie);

  //gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MAG_FILTER, gl.LINEAR);
  //gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR_MIPMAP_NEAREST);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_MIN_FILTER, gl.LINEAR);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_S, gl.CLAMP_TO_EDGE);
gl.texParameteri(gl.TEXTURE_2D, gl.TEXTURE_WRAP_T, gl.CLAMP_TO_EDGE);

  gl.generateMipmap(gl.TEXTURE_2D);
  gl.bindTexture(gl.TEXTURE_2D, null);
}
```

Obr.48: Funkcia `updateTexture()`

Na vytvorenie textúry špecifikujeme, že nová textúra je aktuálna textúra (`patchTexture`), na ktorej chceme operovať, jej zviazaním s `gl.TEXTURE_2D`.

V ďalšom riadku povieme WebGL, že všetky obrázky, ktoré načítame ako textúru, musia byť vertikálne preklopené. Robíme to z dôvodu rozdielnych súradnicových systémov; súradnice našej textúry narastajú smerom hore popri vertikálnej osi, avšak súradnice video textúry narastajú smerom dole popri vertikálnej osi. Horizontálna os je rovnaká v oboch súradnicových systémoch.

Potom načítame aktuálnu snímku videa pomocou funkcie `gl.texImage2D`. Je to WebGL funkcia a slúži na definovanie dvojrozmernej textúry, posledný z parametrov tejto funkcie je premenná myMovie obsahujúca <video> element. WebGL vie, ako vytiahnuť aktuálnu snímku z videa a použiť ju ako textúru.

Ak by sme chceli použiť dva zakomentované riadky slúžiace na filtrovanie textúry (v tomto prípade je použité lineárne filtrovanie pri zväčšovaní a MIP-mapping (pozri kap. 1.14) pri zmenšovaní), šírka aj výška video textúry by museli byť mocninou dvojky (napr. 64, 128, 256, …). WebGL nepodporuje textúry, ktorých rozmery sú iné ako mocniny dvojky (NPOT = non-power of two). Funkcia `generateMipmap()` vygeneruje v tomto prípade chybu INVALID_OPERATION.

Vzorkovanie textúry, ktorej rozmery nie sú mocninou dvojky v shaderi vyprodukuje RGBA farbu (0, 0, 0, 1) ak:

- je zmenšovací filter nastavený na NEAREST alebo LINEAR: inými slovami, ak je použitý jeden z MIP-mapovacích filtrov
- je nastavený repeat mode pre CLAMP_TO_EDGE, opakovanie NPOT textúr nie je podporované

Keďže väčšina videí má rozmery NPOT, bolo treba nejako vyriešiť tento problém. Riešením bolo nepoužiť MIP-mapovací filter. Namiesto toho sme použili wrap mode bez opakovania pre CLAMP_TO_EDGE. Kvôli tomuto opatreniu sú niektoré videá trochu orezané.

Volaním funkcie `generateMipmap()` sa z textúry vygeneruje MIP-mapa. Posledným riadkom funkcie `updateTexture()` povieme WebGL, že sme skončili s manipuláciou s textúrou (do `gl.TEXTURE_2D` vložíme parameter `null`).

Funkciu `updateTexture()` voláme vždy, keď chceme prekresliť scénu funkciou `drawScene()`. Do funkcie `drawScene()` pridáme volanie funkcie `updateTexture()` predtým, ako sa urobí čokoľvek iné. Vo funkcii `drawScene()` nastanú kvôli textúre aj ďalšie zmeny:

```
gl.bindBuffer(gl.ARRAY_BUFFER, patchVerticesTextureCoordBuffer);
gl.vertexAttribPointer(textureCoordAttribute, 2, gl.FLOAT, false, 0, 0);
gl.activeTexture(gl.TEXTURE0);
gl.bindTexture(gl.TEXTURE_2D, patchTexture);
gl.uniform1i(gl.getUniformLocation(shaderProgram, "uSampler"), 0);
```

Obr.49: Doplnenie funkcie `drawScene()` pre vykreslenie plochy s textúrou

Zmena vo funkcii `drawScene()` je jednoduchá. WebGL poskytuje 32 textúrových registrov. Prvý z nich je `gl.TEXTURE0`. Prilinkujeme našu predtým načítanú textúru tomuto registru, potom nastavíme shader sampler uSampler (špecifikovaný v shader programe), aby použil textúru.

Doteraz opísaný postup pri vytváraní aplikácie vytvorí iba základnú aplikáciu s minimálnou funkcionalitou. Jedná sa iba o namapované video na obdĺžnik, ktorý je reprezentovaný Tenzorovo-súčinovou Bézierovou záplatou. Kliknutím na nejaký riadiaci vrchol a jeho následným posunutím je možné meniť tvar tejto plochy. Pri posúvaní nejakého riadiaceho vrchola sa volá funkcia `recompute()`, ktorá prepočíta nové vzorky pre vykreslenie plochy. Zatiaľ chýba ovládanie videa a ďalšie ovládacie prvky, ako napr. rotácia plochy. Preto v nasledujúcich častiach opíšem postup pre pridanie interaktivity do aplikácie.

3.11 Ovládacie prvky pre video

Medzi ovládacie prvky pre video patria napr. spustiť prehrávanie, pozastaviť, prehrať od začiatku (replay), spustiť prehrávanie od zvoleného času (kliknutím na progress bar) alebo nastavenie hlasitosti.

Obr.50: Ovládacie prvky pre video

Vzhľad ovládacích prvkov, ktoré sú reprezentované HTML elementmi, je formátovaný pomocou CSS štýlov a použitím obrázkov.

Všetky tieto ovládacie prvky pre video sa dajú celkom ľahko naprogramovať s využitím metód, vlastností a udalostí <video> elementu (viď kap. 1.6.1).

Hneď, ako sa načíta stránka voláme funkciu `doFirst()`:

```
function doFirst() {
  maxBarSize = 330;
  myMovie = document.getElementById('video');
  progressBar = document.getElementById('progressBar');
  playButton = document.getElementById('playButton');
  bar = document.getElementById('defaultBar');
  mute = document.getElementById('mute');
  playButton.addEventListener('click', playOrPause, false);
  bar.addEventListener('click', clickedBar, false);
  mute.addEventListener('click', toggleMute, false);
  updateBar=setInterval(update, 500);
}
```

Obr.51: Funkcia `doFirst()`

Funkcia `doFirst()` načíta do premenných `myMovie`, `progressBar`, `playButton`, `bar` a `mute` k nim prislúchajúce HTML elementy. Premenným `playButton`, `bar` a `mute` priradíme listenery na udalosť `click`. Ak nastane táto udalosť, listener zavolá danú funkciu (`playOrPause()`, `clickedBar()` alebo `toggleMute()`). V poslednom riadku funkcie `doFirst()` načítame do premennej `updateBar` funkciu `setInterval`, ktorá bude volať funkciu `update()` každý 500 milisekúnd.

```
function playOrPause() {
  if(!myMovie.paused && !myMovie.ended) {
    myMovie.pause();
    playButton.src = "play.jpg";
    window.clearInterval(updateBar);
  } else {
    myMovie.play();
    playButton.src = "pause.jpg";
    updateBar=setInterval(update, 500);
  }
}
```

Obr.52: Funkcia `playOrPause()`

Funkcia `playOrPause()` zistí, či sa práve prehráva video, ak áno, tak ho zastaví. Zmení zdroj obrázka pre tlačidlo `playButton` a zastaví `updateBar`. Ak sa video neprehrávalo, funkcia spustí prehrávanie, zmení obrázok na tlačidle a spustí `updateBar`.

```
function update() {
  if (!myMovie.ended) {
    var size = parseInt(myMovie.currentTime*maxBarSize/myMovie.duration);
    progressBar.style.width=size+'px';
    var time = myMovie.currentTime;
    var m = Math.round(time/60-0.5);
    var s = Math.round(time - m*60);
    m=checkTime(m);
    s=checkTime(s);
    var celkovytime = myMovie.duration;
    var cm = Math.round(celkovytime/60-0.5);
    var cs = Math.round(celkovytime - cm*60);
    cm=checkTime(cm);
    cs=checkTime(cs);
    document.getElementById("cas").innerHTML = m+":"+s+"/"+cm+":"+cs;
  }
  else {
    progressBar.style.width='0px';
    playButton.src = "play.jpg";
    window.clearInterval(updateBar);
  }
}
```

Obr.53: Funkcia `update()`

V prípade, že video ešte neskončilo, funkcia `update()` najprv vypočíta veľkosť progress baru (oranžový obdĺžnik) a vypočítanú hodnotu vloží do premennej `size`. Následne zmení šírku elementu s id progressBar (oranžový obdĺžnik) pomocou CSS štýlov.

Do premennej `time` vložíme aktuálny čas prehrávania videa. Tento čas je v sekundách, preto ho prepočítame na sekundy a minúty a vložíme do premenných `s` (sekundy) a `m` (minúty). V prípade, že je v niektorej z týchto premenných číslo menšie ako 10, funkcia `checkTime()` pridá na začiatok nulu:

```
function checkTime(i) {
  if (i<10) i="0" + i;
  return i;
}
```

Obr.54: Funkcia `checkTime(i)`

Podobne do premennej `celkovytime` vložíme celkovú časovú dĺžku videa. Aj tento atribút je v sekundách, preto aplikujeme rovnaký postup na prevod na sekundy a minúty.

Následne do HTML elementu s id `cas` vypíšeme: aktuálny čas prehrávania / celkový čas.

V prípade, že video už skončilo prehrávanie, nastavíme šírku oranžového obdĺžnika na nulu, zmeníme zdroj obrázka pre playButton a zastavíme updateBar.

Ďalšou funkciou je možnosť prehrávania videa od zvoleného času kliknutím na bar (sivý obdĺžnik). Toto kliknutie zavolá funkciu `clickedBar(e)`:

```
function clickedBar(e){
  if(!myMovie.paused && !myMovie.ended){
    var mouseX=e.pageX-bar.offsetLeft;
    var newtime=mouseX*myMovie.duration/maxBarSize;
    myMovie.currentTime=newtime;
    progressBar.style.width=mouseX+'px';
  }
}
```

Obr.55: Funkcia `clickedBar(e)`

Funkcia `clickedBar(e)` načíta do premennej mouseX x-ovú súradnicu bodu, kde používateľ klikol na bar. Do premennej `newtime` načíta nový čas, vypočítaný na základe premennej `mouseX`. Vlastnosť aktuálny čas nášho videa prepíšeme hodnotou premennej `newtime`. Posledným krokom je zmena šírky progress baru (oranžový obdĺžnik).

Ak si chce používateľ prehrať video od začiatku, môže kliknúť na ikonku replay, čo zavolá funkciu `replay(e)`:

```
function replay(e){
  myMovie.currentTime=0;
  myMovie.play();
  updateBar=setInterval(update, 500);
}
```

Obr.56: Funkcia `replay(e)`

Funkcia `replay(e)` nastaví aktuálny čas videa na nulu, spustí prehrávanie videa a nastaví interval na 500 milisekúnd pre opakované volanie funkcie `update()`.

Posledným prvkom pre ovládanie videa je možnosť vypnúť a zapnúť zvuk kliknutím na ikonku so zjednodušeným reproduktorom. Vtedy sa zavolá funkcia `toggleMute(e)`:

```
function toggleMute(e) {
      if(!myMovie.paused && !myMovie.ended){
          if(myMovie.volume > 0) {
                myMovie.volume = 0;
                mute.src = "unmute.jpg";
          } else {
          myMovie.volume = 1;
          mute.src = "mute.jpg";
          }
      return false;
      }
}
```

Obr.57: Funkcia `toggleMute(e)`

V prípade, že je hlasitosť videa väčšia ako nula, funkcia `toggleMute(e)` nastaví hlasitosť na nulu a zmení obrázok ikonky. Inak nastaví hlasitosť na hodnotu jedna (maximálna hlasitosť) a tiež zmení zdroj obrázka pre ikonku.

3.11.1 Video galéria

Video galéria je súbor šiestich obrázkov reprezentujúcich rôzne videá. Videá sú od mojich spolužiakov, menovite: Bc. Marián Rojko, Bc. Alžbeta Martinkovičová, Bc. Matúš Ždanský, Bc. Martina Patašiová, Bc. Tomáš Csütörtöky a jedno z videí je moje. Sú to krátke animácie vytvorené v blendri na tému propagácia školy alebo vizualizácia nejakého algoritmu. Týmto by som sa chcela poďakovať mojim spolužiakom za ochotu poskytnúť mi ich autorské diela.

Obr.58: Video galéria

Kliknutím na niektorý z obrázkov sa zavolá jedna zo šiestich funkcií, napr. kliknutím na prvý obrázok sa zavolá funkcia `kockac()`:

```
function kockac() {

  myMovie.removeEventListener("canplaythrough", startVideo, true);
  myMovie.pause();
  window.clearInterval(updateBar);

  width = 960;
  height = 540;

  //vypocet bodov plochy
  for (var j=0; j<=degV; j++) {
    for (var i=0; i<=degU; i++) {
      fVertices[pocetRV] = {x: 2.0/degU*i-1.0,
      y: (2.0*height/width)/degV*j-(1.0*height/width), z: 0.0, w: 1.0};
      pocetRV = pocetRV+1;
    }
  }
  pocetRV = 0;

  recompute();

  myMovie.src = "kockac.ogv";
  playButton.src = "pause.jpg";
  myMovie.addEventListener("canplaythrough", startVideo, true);
  myMovie.play();
  updateBar=setInterval(update, 500);

}
```

Obr.59: Funkcia pre zmenu zdroja video textúry

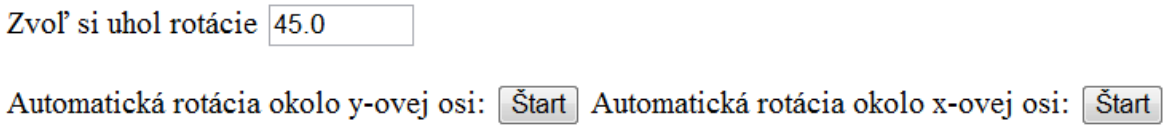
Funkcia pre zmenu video textúry najprv odstráni listener videa na udalosť `canplaytrough`, zastaví prehrávané video a zastaví updateBar. Ďalej načíta rozmery videa a v závislosti na týchto rozmeroch prepočíta nové súradnice riadiacich vrcholov. Následne zavoláme funkciu `recompute()`, ktoré prepočíta vzorky. Až keď sú nastavené správne rozmery plochy, načítame nový zdroj videa. Keby sme to urobili skôr, tak by sa video namapovalo na predošlý tvar plochy, čo by vo väčšine prípadov nebolo dobre, lebo by sa nám video natiahlo do výšky alebo do šírky. V prípade kliknutia na iný obrázok vo video galérii, sa zavolá iná funkcia, ale jediný rozdiel je v rozmeroch a zdroji videa.

Ďalej zmeníme zdroj obrázka na tlačidle playButton. Znova pridáme listener videu na udalosť `canplaytrough`, spustíme video a nastavíme interval volania funkcie `update()` na 500 milisekúnd.

3.12 Ďalšie interaktívne prvky aplikácie

3.12.1 Rotácia plochy

Asi najdôležitejšou funkciou bolo pridanie možnosti otáčania plochy. Vďaka rotácii je možné lepšie vidieť tvar plochy v priestore. Plochu je možné otáčať šípkami na klávesnici (v tomto prípade je možné zvoliť uhol rotácie) alebo spustením automatickej rotácie okolo x-ovej alebo y-ovej osi.



Obr.60: Ovládacie prvky pre rotáciu

Prvá vec, ktorú budeme potrebovať je premenná patchRotationX, v ktorej bude uložený aktuálny uhol rotácie okolo x-ovej osi. Podobne v premennej patchRotationY bude uhol rotácie okolo y-ovej osi. Tieto premenné chceme aktualizovať pri každom stlačení niektorej zo šípok na klávesnici:

```
$(document).keydown(function(event) {
    switch (event.keyCode) {
      case 37: // left
        patchRotationY -= uhol;
        break;
      case 38: // up
        patchRotationX -= uhol;
        break;
      case 39: // right
        patchRotationY += uhol;
        break;
      case 40: // down
        patchRotationX += uhol;
        break;
    }
  } )
```

Obr.61: Zmena uhlov rotácie pri stlačení šípok na klávesnici

Druhá možnosť je zmena uhlov v priebehu času. To sa dá urobiť vytvorením novej premennej `lastpatchUpdateTime` na sledovanie času, v ktorom sme naposledy animovali. Túto premennú využijeme vo funkcii `automaticRotation()`:

```
function automaticRotation() {
  var currentTime = (new Date).getTime();
  if (lastpatchUpdateTime) {
    var delta = currentTime - lastpatchUpdateTime;
    if (rotovanieY==true) patchRotationY += (30 * delta) / 1000.0;
    if (rotovanieX==true) patchRotationX += (30 * delta) / 1000.0;
  }
  lastpatchUpdateTime = currentTime;
}
```

Obr.62: Funkcia `automaticRotation()`

Tento kód používa množstvo času, ktorý ubehol od posledného času, kedy sme aktualizovali hodnotu patchRotationX alebo patchRotationY, pre určenie, o aký veľký uhol máme rotovať. Veľkosť uhlov aktualizujeme len v prípade, že je spustená rotácia okolo x-ovej alebo y-ovej osi (používateľ klikol na jedno z tlačidiel Štart a tým sa nastavila jedna z premenných rotovanieX alebo rotovanieY na true).

Funkciu `automaticRotation()` voláme na konci funkcie `drawScene()`.

Ďalej ešte treba aktualizovať funkciu `drawScene()` pre aplikáciu aktuálnej rotácie na záplatu. Po posunutí na začiatočnú vykresľovaciu pozíciu záplaty, aplikujeme rotáciu:

```
mvPushMatrix();
mvRotate(patchRotationY, [0, 1, 0]);
mvRotate(patchRotationX, [1, 0, 0]);
```

Obr.63: Doplnenie rotácie do funkcie `drawScene()`

Tento kód uloží aktuálnu model view maticu, potom použije funkciu `mvRotate` pre rotáciu okolo jednotlivých osí o aktuálne uhly rotácie.

Po vykreslení záplaty potrebujeme obnoviť pôvodnú maticu. Na to slúži funkcia `mvPopMatrix()`.

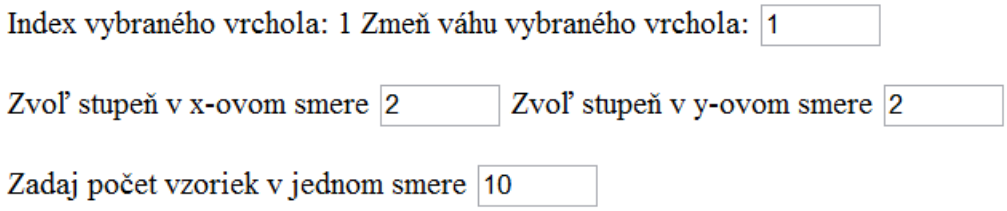
3.12.2 Škálovanie

Škálovanie sa používa, keď chceme priblížiť alebo oddialiť objekty v scéne. Vo WebGL sa to dá urobiť jednoducho pridaním funkcie `mvScale(scale)` do funkcie `drawScene()`. Funkcia `mvScale(scale)` prenásobí model view maticu diagonálnou maticou, ktorej prvky na diagonále majú hodnotu `scale`. Parameter `scale` sa mení

v závislosti na otáčaní kolieska myši používateľom. Väčšina internetových prehliadačov podporuje vlastnosť `wheelDelta`, ktorá má v sebe uloženú informáciu o otočení kolieska myši. Prehliadač Firefox túto premennú nepozná, preto treba vždy zistiť, v akom prehliadači si používateľ prezerá stránku a v prípade Firefoxu sa na aktualizáciu parametra `scale` použije iná premenná, a to `mousewheel`.

3.12.3 Zmena parametrov Tenzorovo-súčinovej Bézierovej záplaty

Medzi parametre plochy patrí stupeň v x-ovom smere a v y-ovom smere (tzv. bistupeň), ďalej počet vzoriek v jednom smere a váha jednotlivých riadiacich vrcholov. Všetky tieto parametre má používateľ možnosť meniť vpísaním ním zvolenej hodnoty v aplikácii do polí na to určených (viď obr.64).



Obr.64: Parametre plochy

Po zmene hodnoty a uzavretí poľa (kliknutím niekde mimo) nastáva HTML DOM udalosť `onblur`. Udalosť `onblur` zavolá funkciu, ktorá najprv vloží novú hodnotu do premennej na to určenej a následne zavolá funkciu `recompute()`. V prípade zmeny stupňa plochy je ešte pred volaním funkcie `recompute()` nutné vypočítať nové riadiace vrcholy.

3.12.4 Preddefinované tvary plochy

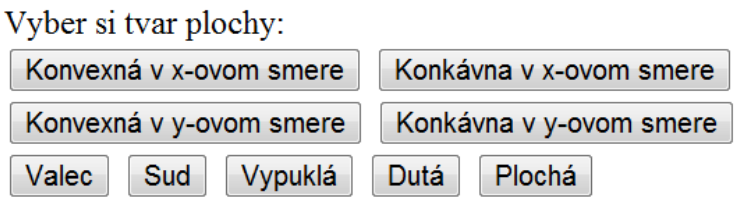
Používateľ má možnosť vybrať si z týchto preddefinovaných tvarov plochy:

- Konvexná v x-ovom smere
- Konkávna v x-ovom smere
- Konvexná v y-ovom smere
- Konkávna v y-ovom smere
- Valec
- Sud
- Vypuklá

- Dutá
- Plochá

Obr.65: Jedna z preddefinovaných tvarov plochy

Preddefinované tvary plochy sa dajú vyberať kliknutím na jedno z tlačidiel, ktoré majú v názve niektorý z tvarov plochy:

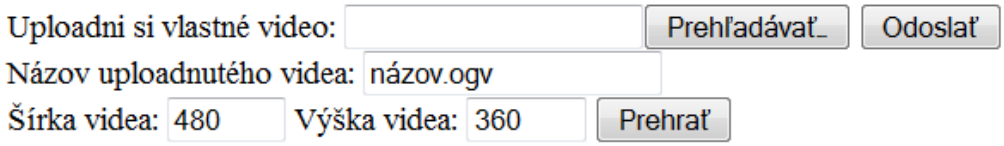


Obr.66: Preddefinované tvary plochy

Po kliknutí na jedno z tlačidiel preddefinovaných tvarov plochy sa zavolá JavaScriptová funkcia, ktorá nastaví stupeň Tenzorovo-súčinovej Bézierovej záplaty, následne prepočíta nové riadiace vrcholy a zavolá funkciu `recompute()`.

3.13 Upload vlastného videa

Posledným krokom bola implementácia možnosti uploadovania vlastných videí.



Obr.67: Upload vlastného videa

Kliknutím na tlačidlo prehľadávať sa otvorí dialógové okno, kde je možné vybrať hocijaký súbor uložený v počítači používateľa. Po výbere súboru je treba kliknúť na tlačidlo odoslať. Vtedy zareaguje PHP funkcia `isset()`, pretože ako parameter má tlačidlo Odoslať (submit).

```
<?php
if(isset($_POST['submit']))
{
if ((($_FILES["file"]["type"] == "video/ogg")
|| ($_FILES["file"]["type"] == "video/mp4")
|| ($_FILES["file"]["type"] == "video/webm"))
&& ($_FILES["file"]["size"] < 1700000))
  {
  if ($_FILES["file"]["error"] > 0)
    echo "Return Code: " . $_FILES["file"]["error"] . "<br />";
  else
    {
    if (file_exists("upload/" . $_FILES["file"]["name"]))
      echo $_FILES["file"]["name"] . " už existuje. ";
    else
      {
      move_uploaded_file($_FILES["file"]["tmp_name"],
       "upload/" . $_FILES["file"]["name"]);
      chmod("upload/" . $_FILES["file"]["name"],0777);
      echo "Video bolo úspešne uploadnuté na server.
       Po zadaní názvu a rozmerov, si ho môžete dať prehrať.";
      }
    }
  }
else
   echo "Video nie je možné uploadnúť. Podporované formáty sú iba:
   ogg, mp4 a webm. Maximálna veIkosť súboru je 1,6 MB.";
}
?>
```

Obr.68: PHP kód pre spracovanie uploadovaného videa

Ako prvé sa zisťuje, či je uploadovaný súbor video, konkrétne to môže byť jedine jeden z troch podporovaných formátov <video> elementom (ogg, mp4 alebo webm). V prípade, že používateľ chcel uploadnúť súbor v inom formáte, alebo je jeho súbor príliš veľký, vypíše sa o tom oznam. Pri uploadovaní môže nastať aj nejaká chyba, vtedy sa vypíše oznam o tejto chybe. Ak nenastala žiadna chyba, zisťujeme, či už neexistuje súbor s rovnakým názvom uploadnutý na server. V tomto prípade by sa vypísal oznam, že súbor už existuje. Ak je všetko v poriadku, presunieme uploadovaný súbor do priečinka s názvom upload. Ďalej voláme funkciu `chmod(file,mode)`, ktorá slúži na zmenu oprávnení pre uvedený súbor. `mode` parameter pozostáva zo štyroch čísel:

- Prvé číslo je vždy nula
- Druhé číslo špecifikuje oprávnenia pre vlastníka
- Tretie číslo špecifikuje oprávnenie pre používateľskú skupina vlastníka
- Štvrté číslo špecifikuje oprávnenie pre všetkých ostatných

Možné hodnoty (pre nastavenie rôznych oprávnení):

- 1 – oprávnenie spustiť súbor
- 2 – oprávnenie zapisovať
- 4 – oprávnenie čítať

Ak chceme zvoliť viac ako jedno oprávnenie (napr. kombináciu čítať a zapisovanie) je nutné tieto hodnoty sčítať.

Po zmene oprávnení je už video k dispozícii pre používateľa. Preto sa o tom vypíše oznámenie spolu s ďalšími inštrukciami, ako prehrať toto uploadnuté video.

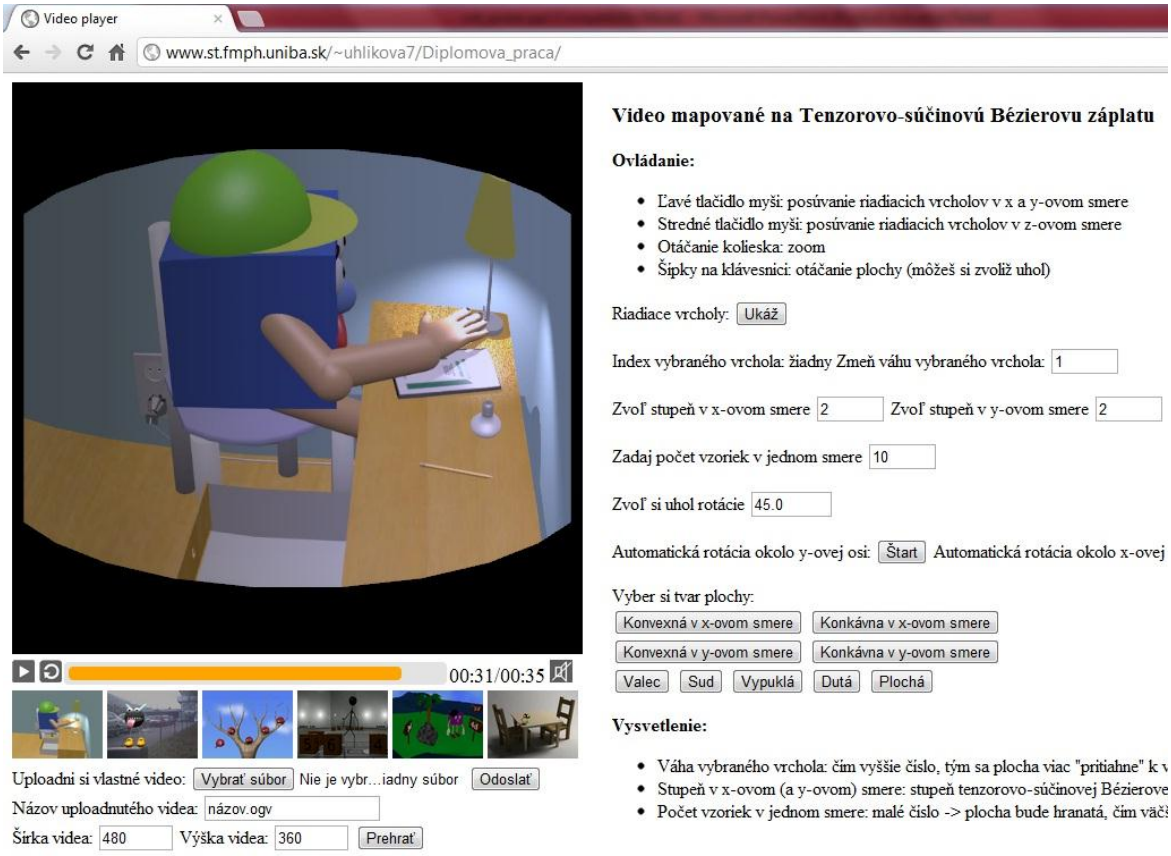
3.14 Porovnanie vytvorenej aplikácie s navrhovanou aplikáciou

V tejto časti porovnám vytvorenú aplikáciu s navrhovanou aplikáciou, ktorá bola bližšie popísaná v špecifikácii. Vyhodnotím, či sú splnené požiadavky na funkcionalitu aplikácie uvedené v špecifikácii.

Aplikácia prehráva video, sú implementované základné ovládacie prvky (pozastavenie videa, prehrávanie od začiatku alebo od akéhokoľvek zvoleného času). Nie je možné nastaviť hlasitosť. Zvuk sa dá iba vypnúť alebo zapnúť. V aplikácii je možné vyberať videá z

video galérie, prípadne uploadnúť vlastné video. Pri uploadovaní sú obmedzenia pre formát a veľkosť videa. Formát videa je obmedzený na formáty videa, ktoré podporuje <video> element. Veľkosť videa je obmedzená nastaveniami servera, na ktorý sa video uploaduje.

V aplikácii sa video mapuje na Tenzorovo-súčinovú Bézierovu záplatu, ktorá je zadaná pomocou riadiacich vrcholov. Vďaka tomu je možné meniť tvar plochy posúvaním riadiacich vrcholov v priestore, plocha sa dá priblížiť, oddialiť, otáčať okolo zvislej alebo vodorovnej osi o zvolený uhol, prípadne sa dá spustiť automatická rotácia. Dá sa vyberať z preddefinovaných tvarov plochy. Je možné nastaviť váhy pre jednotlivé riadiace vrcholy (je implementovaná racionálna Tenzorovo-súčinová Bézierova záplata). Dá sa zvoliť si stupeň Tenzorovo-súčinovej Bézierovej záplaty a počet vzorkovacích bodov v jednom smere, tzv. parameter "hladkosti".



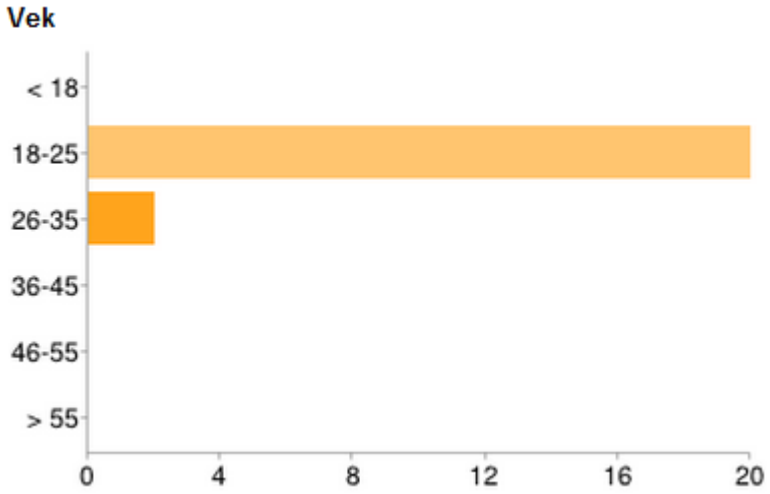
Obr.69: Screenshot vytvorenej aplikácie zobrazenej v internetovom prehliadači

4 Testovanie

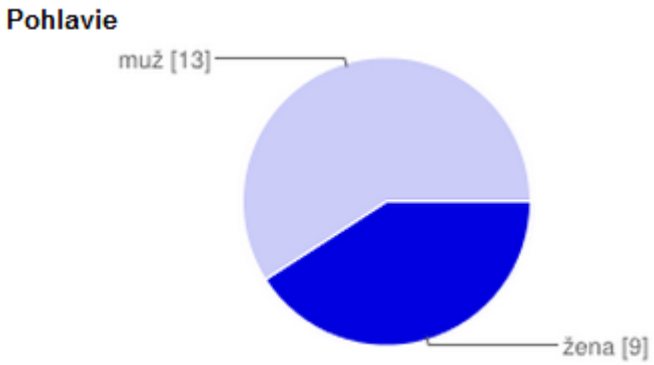
Vytvorenú aplikáciu som sa rozhodla otestovať prostredníctvom dotazníka. Dotazník je výskumný nástroj skladajúci sa zo série otázok. Slúži na zisťovanie informácií v populácii alebo v nejakej menšej skupine osôb. Na základe odpovedí respondentov sa vyhodnotia určité skutočnosti (názory, postoje, preferencie) a určí sa smerovanie ďalších krokov. Dotazníky majú výhody oproti niektorým iným druhom prieskumov v tom, že sú lacné a nevyžadujú toľko úsilia od anketára ako slovné alebo telefónne prieskumy. Často majú štandardizované odpovede, ktoré uľahčujú zhromažďovanie údajov.

Dotazník na zhodnotenie mojej aplikácie som vytvorila pomocou internetovej aplikácie Google Dokumenty. Prvá časť otázok sa týkala demografických údajov respondentov. V druhej časti boli otázky zamerané na samotnú aplikáciu, najmä na jej funkcionalitu.

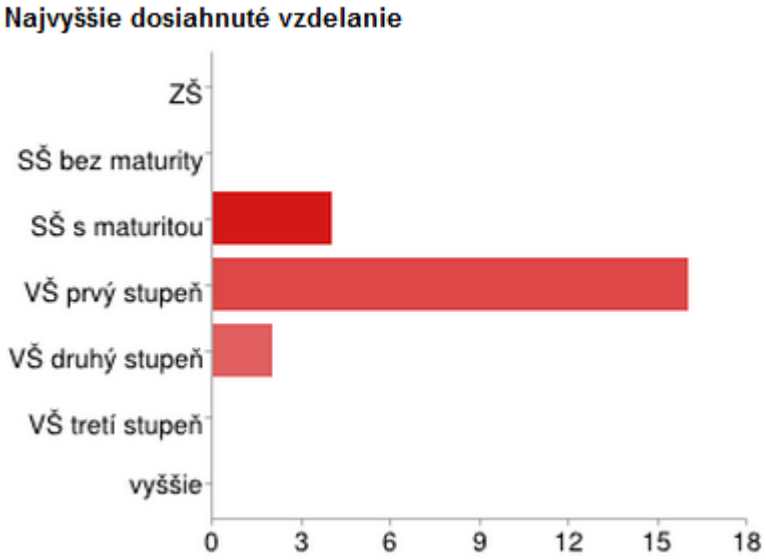
Na vypĺňaní dotazníka sa zúčastnilo dvadsaťdva respondentov. V nasledujúcich grafoch je znázornené zastúpenie respondentov podľa veku, pohlavia, dosiahnutého vzdelania a jeho zamerania. Posledným ukazovateľom je kraj, kde respondenti bývajú.



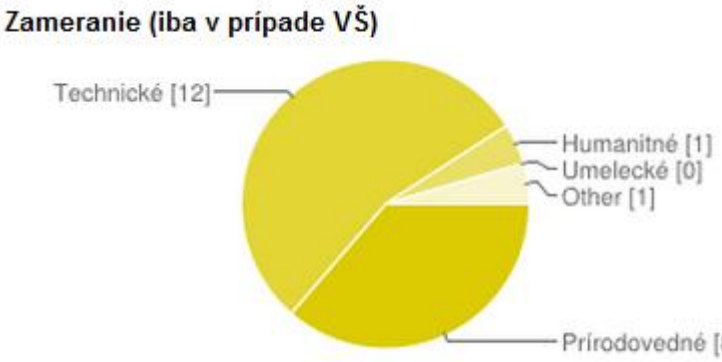
Graf 1: Vek respondentov



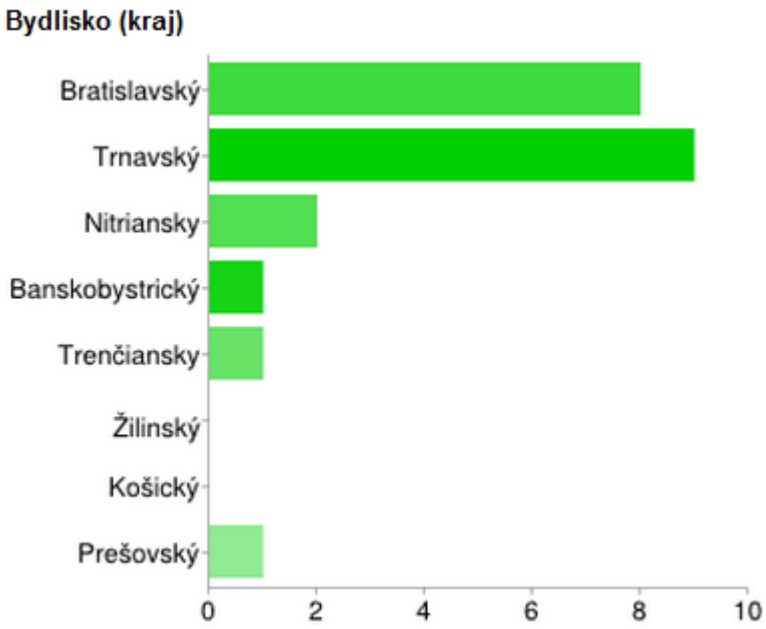
Graf 2: Pohlavie respondentov



Graf 3: Najvyššie dosiahnuté vzdelanie respondentov



Graf 4: Zameranie vysokej školy navštevovanej respondentmi

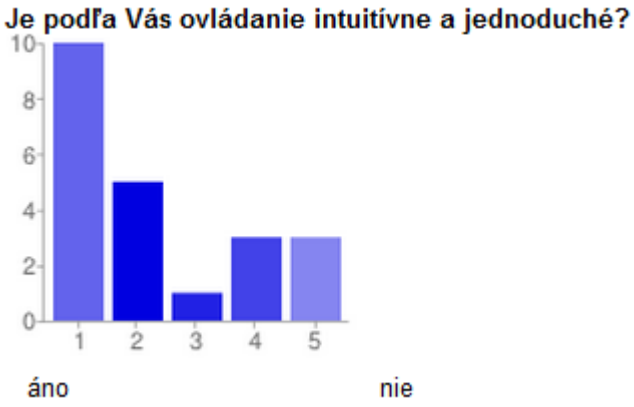


Graf 5: Zastúpenie krajov, v ktorých bývajú respondenti

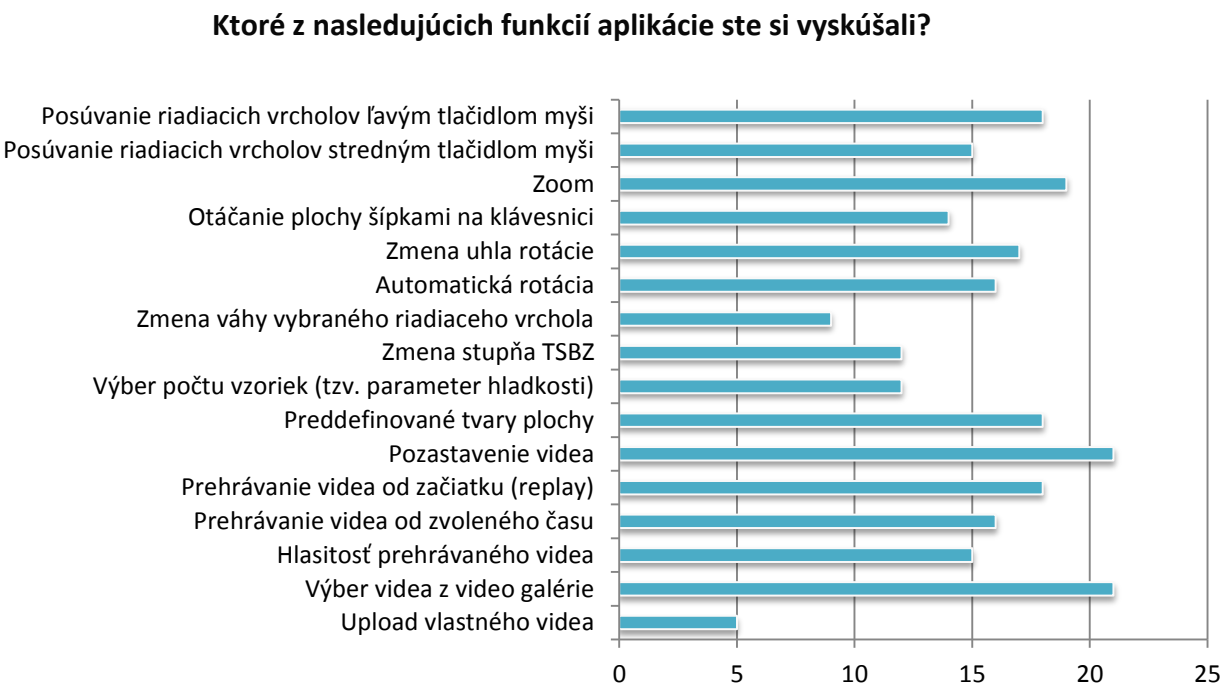
Z grafov vidíme, že vek respondentov sa pohyboval v rozmedzí 18 až 35 rokov, do ankety sa zapojilo o niečo viac mužov, najviac respondentov malo najvyššie dosiahnuté vzdelanie vysokoškolské prvého stupňa, respondenti mali hlavne technické alebo prírodovedné zameranie školy. Hlavné zastúpenie respondentov bolo v Trnavskom a Bratislavskom kraji. Tieto výsledky odzrkadľujú to, že medzi respondentmi boli najmä moji spolužiaci a známi.

Druhá časť dotazníka bola zameraná na aplikáciu. Respondenti boli požiadaní, aby si ju vyskúšali a na základe toho ju zhodnotili odpovedaním na otázky v dotazníku, prípadne mali možnosť dopísať komentár. Výsledky sú vizualizované prostredníctvom grafov.

Graf 6: Zaujímavosť aplikácie



Graf 7: Ovládanie aplikácie



Graf 8: Počet respondentov, ktorí si vyskúšali jednotlivé funkcie aplikácie

Graf 9: Formát uploadovaných videí

Väčšina respondentov zhodnotila aplikáciu ako zaujímavú, o trochu horšie to bolo s jednoduchosťou ovládania. Tomu sa dá celkom rozumieť, pretože aplikácia má rôzne dosť netradičné funkcie, ako možnosť rotácie plochy, na ktorú sa mapuje video, alebo možnosť meniť tvar plochy a možnosť meniť parametre tejto plochy. Plocha je Tenzorovo-súčinová Bézierova záplata, čo bol pre mnohých respondentov nový pojem a teda nevedeli, čo môžu od toho očakávať. Aj práve komentáre, ktoré dopísali niektorí respondenti na nasledujúce otázky, sa týkali jednoduchosti ovládania.

Máte nejaké návrhy, čo by som mala v aplikácii zmeniť? Alebo máte nejaký nápad, čo by som tam mohla doplniť?

„Dalo by sa vypracovať jednoduché používateľské rozhranie systému: čím menej, tým viac. “

„Musela som často čakať, aj keď som využívala Google Chrome. Nie vždy na mňa reagovali tlačidlá. Napr. zmena stupňa. Môže to súvisieť s mojim pomalým PC. Inak sa mi aplikácia veľmi páči. Je to super nápad. Skúsim si niekedy uploadnúť aj moje video :) “ - dôvod prečo tlačidlá nereagujú hneď je, že väčšina polí, kam sa dajú vpisovať hodnoty zareaguje až keď používateľ klikne mimo, jedná sa o HTML DOM udalosť `onblur`, bolo by možné použiť aj udalosť `onchange`, ale tam by bolo treba dôslednejšie ošetriť vstup, pretože v prípade, že používateľ pred prepísaním poľa toto pole vymaže, tak by tam v nejakom okamihu nebola žiadna hodnota, čo by mohlo spôsobiť problém.

„Číselné údaje v poliach by sa mali dať meniť šípkami, treba definovať rozsah (min a max hodnoty). “ – toto je veľmi dobrá rada, určite by bolo vhodné takto vyriešiť v aplikácii možnosť zmeny parametrov plochy alebo uhol rotácie plochy.

Napriek menším problémom s ovládaním, si každý respondent vyskúšal aspoň zopár funkcií aplikácie, väčšina respondentov si vyskúšala pomerne veľa funkcií (viď graf 8). Najčastejšie používané funkcie aplikácie boli: výber videa z video galérie, pozastavenie videa, zoom, posúvanie riadiacich vrcholov plochy, preddefinované tvary plochy a prehrávanie videa od začiatku (replay). Najmenej respondentov skúšalo uploadnúť svoje video.

Kompletné znenie a vzhľad dotazníka sa nachádza v prílohe diplomovej práce.

Záver

Cieľom mojej diplomovej práce bolo navrhnúť prehrávač videa pre webové rozhranie, ktorý vie prehrávať video a podporuje zobrazenie 3D. Ďalej navrhnúť možné projekcie videa na hladké plochy. Aplikácia, ktorú som vytvorila má všetky tieto funkcie a navyše plocha, na ktorú sa mapuje video je reprezentovaná Tenzorovo-súčinovou Bézierovou záplatou, ktorej tvar je definovaný pomocou riadiacich vrcholov a ich umiestnenia v priestore. Súradnice týchto vrcholov v mojej aplikácii nie sú konštantné, ale premenné a vďaka tomu bolo možné pridať interaktivitu v podobe možnosti posúvania týchto vrcholov v priestore, čím sa mení aj tvar celej plochy.

Predtým, ako som začala vytvárať aplikáciu, som si musela naštudovať množstvo postupov implementácie aplikácií s čiastočne podobnou funkcionalitou. Nenašla som žiadnu aplikáciu, ktorá by mapovala video na plochu, ktorej tvar sa dá meniť. Pri mojej práci som skombinovala viaceré naštudované postupy, ktoré riešili iba čiastočne moju úlohu.

Aplikácia je predovšetkým webový video prehrávač, teda prehráva video a sú v nej implementované základné ovládacie prvky pre video (pozastavenie videa, prehrávanie od začiatku alebo od akéhokoľvek zvoleného času, zapnutie alebo vypnutie zvuku). Aplikácia zobrazuje časovú dĺžku už prehraného a celého videa. V aplikácii je možné vyberať videá z video galérie, prípadne uploadnúť vlastné video. Pri uploadovaní sú obmedzenia pre formát a veľkosť videa. Formát videa je obmedzený na formáty videa, ktoré podporuje HTML 5 <video> element. Veľkosť videa je obmedzená nastaveniami servera, na ktorý sa video uploaduje.

Plocha, na ktorú sa mapuje video, sa dá priblížiť alebo oddialiť otáčaním kolieska myši. Plochu je možné otáčať okolo zvislej alebo vodorovnej osi o zvolený uhol šípkami na klávesnici, prípadne sa dá spustiť automatická rotácia. Dá sa vyberať z ôsmich preddefinovaných tvarov plochy, ako napríklad vypuklá, dutá, sud a valec. Je možné nastaviť váhy pre jednotlivé riadiace vrcholy (ide o racionálnu Tenzorovo-súčinovú Bézierovu záplatu). Dá sa zvoliť si stupeň Tenzorovo-súčinovej Bézierovej záplaty v jednotlivých smeroch, čo ovplyvní počet riadiacich vrcholov. Je možné zmeniť počet

vzorkovacích bodov v jednom smere (tzv. parameter "hladkosti"), čím väčšie číslo zvolíme, tým bude plocha pôsobiť hladšie. Parametre plochy sa dajú meniť prepisovaním aktuálnych hodnôt v textových poliach. Ako ukázala anketa, nie je to najlepší spôsob. Pre používateľov by bolo jednoduchšie, keby mohli hodnoty zvyšovať alebo znižovať šípkami.

Aplikácia je plne funkčná a je dostupná na mojej internetovej stránke: http://www.st.fmph.uniba.sk/~uhlikova7/Diplomova_praca/

Použitá literatúra

[Spitzer] SPITZER, P. *Mapping Flash video to 3D objects*,
URL < http://www.adobe.com/devnet/flash/articles/flash_video_3d.html > (21.4.2012)

[collada] COLLADA – oficiálna stránka, URL < http://collada.org/mediawiki/index.php > (12.4.2012)

[blender] Blender – oficiálna stránka, URL < http://www.blender.org > (12.4.2012)

[Salvet11] SALVET, P. 2011. BYE, BYE, FLASH. In *CHIP*. 2011, roč. 2011, č. 2, s. 52-54.

[Kolesár] KOLESÁR, I. 2010. 3D v technológii Flash pomocou open source knižníc. In *Konferencia OSSConf 2010*, s. 101-108.

[Žára] ŽÁRA, J. 1999. *Laskavý průvodce virtuálními světy* [online], Dostupné na internete: <http://www.cgg.cvut.cz/LaskavyPruvodce/>

[Zrzavý] ZRZAVÝ, J. 1999. *VRML tvorba dokonalých WWW stránek: Podrobný sprůvodce*. Grada Publishing. ISBN 80-7169-643-9.

[Salvet10] SALVET, P. 2010. HTML 5 A DĚDICTVÍ MINULOSTI. In *CHIP*. 2010, roč. 2010, č. 11, s. 78-80.

[Müller] MÜLLER, C. 2010. HTML 5: Nový jazyk na internetu. In *CHIP*. 2010, roč. 2010, č. 5, s. 14-15.

[w3chools] Free tutoriály o technológiách pre vytváranie webu,
URL < http://www.w3schools.com/ > (22.4.2012)

[WebGL] WebGL – oficiálna stránka, URL < http://www.khronos.org/webgl/ > (9.10.2011)

[mdn] MOZILLA DEVELOPER NETWORK, URL < https://developer.mozilla.org/en/WebGL > (8.2.2012)

[LearningWebGL] Lekcie WebGL, URL <http://learningwebgl.com/blog/> (8.2.2012)

[CookBook] Návody pre WebGL, URL < http://learningwebgl.com/cookbook/index.php > (17.3.2012)

[DEV.OPERA] Stránky pre vývojárov webu, URL < http://dev.opera.com/ > (7.4.2012)

[RTR] AKENINE-MOLLER, T., HAINES, E., AND HOFFMAN, N. 2008. *Real-Time Rendering.* Third Edition. Wellesley :A K Peters, Ltd., 2008. ISBN-13: 978-1-56881-424-7

[Farin] FARIN, G. 2001. *Curves and Surfaces for CAGD, A Practical Guide,* 5th Edition: Morgan Kaufmann, 2001. ISBN 9781558607378.

[Žára a kol.] ŽÁRA, J. - BENEŠ, B. - SOCHOR, J. - FELKEL, P. 2004. *Moderní počítačová grafika*. 2. vyd. Brno : Computer Press, 2004. 608 s. ISBN 80-251-0454-0.

Prílohy

I Dotazník

Webový video prehrávač s podporou 3D

Na mojej stránke: http://www.st.fmph.uniba.sk/~uhlikova7/Diplomova_praca/
je aplikácia, ktorú som vytvorila v rámci mojej diplomovej práce.
Môžete si ju vyskúšať a potom sa vrátiť k tomuto dotazníku.
Dotazník slúži ako súčasť kvantitatívneho testovanie mojej aplikácie.
Prosím odpovedzte na nasledujúce otázky:

Pohlavie

- ◯ žena
- ◯ muž

Vek

- ◯ < 18
- ◯ 18-25
- ◯ 26-35
- ◯ 36-45
- ◯ 46-55
- ◯ > 55

Najvyššie dosiahnuté vzdelanie

- ◯ ZŠ
- ◯ SŠ bez maturity
- ◯ SŠ s maturitou
- ◯ VŠ prvý stupeň
- ◯ VŠ druhý stupeň
- ◯ VŠ tretí stupeň
- ◯ vyššie

Zameranie (iba v prípade VŠ)

- ◯ Prírodovedné
- ◯ Technické
- ◯ Humanitné
- ◯ Umelecké
- ◯ Other:

Bydlisko (kraj)

- ◯ Bratislavský
- ◯ Trnavský
- ◯ Nitriansky
- ◯ Banskobystrický
- ◯ Trenčiansky
- ◯ Žilinský
- ◯ Košický
- ◯ Prešovský

Nasledujú otázky na samotnú aplikáciu:

Je podľa Vás aplikácia zaujímavá?

	1	2	3	4	5	
áno	◯	◯	◯	◯	◯	nie

Je podľa Vás ovládanie intuitívne a jednoduché?

	1	2	3	4	5	
áno	◯	◯	◯	◯	◯	nie

Ktoré z nasledujúcich funkcií ste si vyskúšali?

Posúvanie riadiacich vrcholov ľavým tlačidlom myši

áno

Posúvanie riadiacich vrcholov stredným tlačidlom myši

áno

Zoom

áno

Otáčanie plochy šípkami na klávesnici

áno

Zmena uhla rotácie

áno

Automatická rotácia

áno

Zmena váhy vybraného riadiaceho vrchola

áno

Zmena stupeňa Tenzorovo-súčinovej Bézierovej záplaty

áno

Výber počtu vzoriek (tzv. parameter hladkosti)

áno

Preddefinované tvary plochy

áno

Pozastavenie videa

áno

Prehrávanie videa od začiatku (replay)

áno

Prehrávanie videa od zvoleného času (kliknutím na progress bar)

áno

Hlasitosť prehrávaného videa

áno

Výber videa z video galérie

áno

Upload vlastného videa

áno

Máte nejaké návrhy, čo by som mala v aplikácii zmeniť? Alebo máte nejaký nápad, čo by som tam mohla doplniť?

V prípade, že ste si uploadovali vlastné video, odpovedzte prosím ešte na ďalšie tri otázky:

Aký formát malo videa / videá, ktoré ste uploadovali?

- .ogg / .ogv
- .mp4
- .webm

Aké malo video rozmery? (napr. 640x360)

Koľko MB malo uploadované video?

Ďakujem Vám za Váš čas.

V prípade otázok alebo pripomienok ma neváhajte kontaktovať na emailovej adrese ivanauhlikova@gmail.com.

Submit

II CD príloha

K diplomovej práci prikladám CD, ktoré obsahuje vytvorenú aplikáciu. Ďalej obsahuje textovú časť diplomovej práce vo formáte `.pdf` a WampServer pre možnosť spustenia aplikácie a prezeranie vytvorenej stránky.