Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Aleksandar Bobić

Vizualizacija 3D medicinskih posnetkov na spletu z uporabo tehnologije WebCL

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Matija Marolt

Ljubljana, 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo: Vizualizacija 3D medicinskih posnetkov na spletu z uporabo tehnologije WebCL

Tematika naloge:

V diplomski nalogi preučite aplikacijo za vizualizacijo 3D medicinskih podatkov NeckVeins. Izdelajte spletno različico te aplikacije, pri čemer uporabite tehnologijo WebCL za hitro izvajanje algoritma Marching Cubes. Primerjajte hitrost spletne aplikacije in aplikacije NeckVeins pri izvajanju omenjenega algoritma in pri prikazovanju 3D modelov.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani Aleksandar Bobić sem avtor diplomskega dela z naslovom:

Vizualizacija 3D medicinskih posnetkov na spletu z uporabo tehnologije WebCL

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Matije Marolta,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 15. september 2015 Podpis avtorja:

Zahvaljujem se doc. dr. Matiji Maroltu in asist. mag. Cirilu Bohaku za pomoč pri izdelavi in pisanju diplomskega dela. Zahvaljujem se Juretu Kolenkotu za moralno podporo in pomoč pri izdelavi. Prav tako se zahvaljujem vsem bližnjim in vsem prijateljem za spodbudo pri izdelavi diplomskega dela.

Svoji družini.

Kazalo

Povzetek

Namen diplomske naloge je vgraditi osnovne funkcionalnosti aplikacije NeckVeins v spletno aplikacijo in pri tem dokazati, da je možno vgraditi enakovredne funkcionalnosti na spletu. V diplomskem delu je opisan postopek izdelave spletne aplikacije za vizualizacijo volumetričnih podatkov žil, pridobljenih iz naprave CT. Sprva so predstavljene vse uporabljene tehnologije, nato je predstavljeno predelovanje urejevalnika Three.js za potrebe diplomske naloge. Opisan je postopek vgrajevanja funkcionalnosti prikazovanja in upravljanja objektov. Prav tako je predstavljen glavni del diplomskega dela, ki predstavlja odpiranje in prikazovanje parov datotek Mhd in Raw s pomočjo Marching cubes algoritma s poskusno spletno tehnologijo WebCL. Nazadnje so predstavljeni rezultati meritve hitrosti obdelave podatkov v spletni aplikaciji in namizni aplikaciji NeckVeins ter meritve hitrosti izrisovanja v spletni in namizni aplikaciji. Pri tem je ugotovljeno, da ni bistvene razlike v hitrosti obdelovanja podatkov. Pri samem izrisovanju predmeta, ko miruje in ko ga rotiramo, pa se izkaže, da je spletna aplikacija počasnejša.

Ključne besede: OpenGL, OpenCL, WebGL, WebCL, Three.js, Marching cubes, JavaScript.

Abstract

The main goal of this thesis is to implement the basic functionalities of the NeckVeins application in to a web application and by that proving that it's possible to implement the same functionalities on the web. This thesis describes the process of developing a web application for visualization of volumetric data of veins from a CT machine. In the beginning of this thesis all the used technologies are described. The description of editing the Three.js editor for the purpose of this thesis follows. In it we describe the functionalities of displaying and controlling the objects. Then we describe the biggest part of this thesis which is the support for opening and displaying pairs of Mhd and Raw data with the algorithm Marching cubes implemented with the experimental WebCL technology. Lastly we describe our speed measurements of the web and the desktop NeckVeins application in which we conclude that there is no major difference in speed of processing data, but there is a big speed gap between the speed of drawing frames. It's concluded that the desktop application is much faster when it comes to drawing single frames.

Keywords: OpenGL, OpenCL, WebGL, WebCL, Three.js, Marching cubes, JavaScript.

Poglavje 1

Uvod

V današnjem času ima splet zelo pomembno vlogo v našem življenju tako v zasebnem kot tudi v profesionalnem. Spletne aplikacije za shranjevanje, prikazovanje in obdelovanje podatkov so se usidrale globoko v naše vsakdanje življenje. Uporabljamo jih za komunikacijo s prijatelji, za dostopanje do naših bančnih računov, za shranjevanje pomembnih podatkov in še za veliko drugih namenov. S predstavitvijo WebGL-a leta 2011 [21] se je zmanjšalo število omejitev za uporabo spletnih aplikacij. Od tedaj se lahko na spletu prikazujejo tudi interaktivni tridimenzionalni podatki. Naknadno se je razvilo tudi veliko knjižnic, ki so olajšale delo z WebGL-om, kot sta Three.js [17] in babylon.js [2]. Pozneje pa se je začel tudi razvoj WebCL-a, ki nam omogoča vzporedno obdelavo podatkov. Z njim lahko izdelamo spletne aplikacije, ki nam omogočajo obdelovanje zelo velike količine podatkov, z uporabo računske moči grafičnih kartic, kar znotraj našega spletnega brskalnika, če so seveda izračuni, ki jih želimo izvajati na teh podatkih, razgradljivi v veliko manjših neodvisnih delov.

Za prikazovanje sta nam na voljo dve vrsti tridimenzionalnih podatkov. Tridimenzionalni podatki, ki smo jih sami tvorili in oblikovali (npr. s pomočjo 3D modeliranja) ali pa tridimenzionalni podatki, ki smo jih pridobili iz zunanjega sveta s pomočjo raznih naprav za zajem 3D oblik. Podatki iz zunanjega sveta nam lahko služijo za analiziranje lastnosti, ki jih v realnem svetu

ne bi mogli. Primer tega v medicini je računalniška tomografija ali krajše CT (angl. Computed tomography). Ta nam omogoča prikaz notranjosti človeškega telesa in podrobnejšo analizo kosti, ožilja in drugih delov telesa v primeru poškodb, tumorjev ali infekcij. Slike so pridobljene z obsevanjem človeškega telesa z rentgenskimi žarki [5]. Podatke, ki jih dobimo iz naprave za CT slikanje, moramo obdelati in iz njih sestaviti tridimenzionalni model, ki ga nato lahko gledamo s pomočjo aplikacije.

Na trgu je že veliko aplikacij, namenjenih medicinski vizualizaciji, ki olajšajo delo tako zdravnikom pri preiskovanju simptomov bolezni in posledic raznih poškodb, kot tudi študentom, ki se učijo človeške anatomije ter kako izgledajo razni simptomi bolezni in posledice poškodb v dejanskem človeškem telesu. Primer takšnih programov je aplikacija Bodyviz [3], ki omogoča vizalizacijo podatkov, pridobljenih iz MRI in naprav CT, prikazovanje podatkov glede na vrsto (koža, maščobno tkivo, žile, itd.) in kontroliranje pogleda ter modela s pomočjo xbox upravljalnika. Obstaja tudi spletna aplikacija arivis WebView [1], ki nam omogoča ogledovanje 2D slik in 3D slik, pridobljenih z računalniško tomografijo ali iz mikroskopov. Omogoča odpiranje zelo velikih datotek, ki jih nato predela na strežniški strani in nam jih prikaže znotraj spletnega brskalnika.

Glavni cilj te diplomske naloge je pokazati, da je z uporabo tehnologij WebGL in WebCL možna vizualizacija tridimenzionalnih podatkov, pridobljenih iz CT naprave znotraj spletnega brskalnika ter algoritma Marching Cubes na uporabnikovem računalniku. Pri tem bom uporabil dele že napisane OpenCL C kode iz programa NeckVeins, ki je bil razvit v obliki več diplomskih nalog v Laboratoriju za računalniško grafiko in multimedije in jih integriral v spletno aplikacijo. Za razliko od svojih konkurentov je program NeckVeins še v zgodnjih fazah svojega razvoja in trenutno nima toliko različnih zmogljivosti za prikazovanje oziroma urejanje medicinskih podatkov, vendar pa ima veliko več možnosti za samo upravljanje kamere in objektov.

Spletna različica programa bo ponudila večjo prilagodljivost, saj bo dostopna z vsakega računalnika z internetno povezavo in ne le določenega

računalnika z naloženo aplikacijo. To nam omogoča, da jo zaganjamo tudi na slabših računalnikih, kot so chromebook-i, ki imajo na sebi naložen Googlov operacijski sistem Chrome OS. V osnovi je namenjen za splet in za delo s spletnimi aplikacijami. Razvoj na spletu pa ima tudi nekaj slabih lastnosti. Trenutno je aplikacijo možno razvijati samo na določenih brskalnikih, saj je tehnologija v poskusni fazi. JavaScript v primerjavi z Javo velja za počasnejši jezik, kar pomeni, da je potrebno pri samem razvoju več časa posvetiti optimizaciji. Nekatera podjetja dovoljujejo samo določene spletne brskalnike, kot je Interent Explorer, ki ne podpirajo najnovejših tehnologij. V takšnem primeru nam pomaga namizna aplikacija napisana v Javi.

Začel bom s spletnim urejevalnikom tridimenzionalnih podatkov, zgrajenim z uporabo spletne knjižnice Three.js. Namen je odstraniti vse nepotrebne funkcionalnosti ter urediti uporabniški vmesnik tako, da bo ponujal podobne funkcionalnosti, kot jih ponuja program NeckVeins. Nato spremeniti sam prikaz osnovnih tridimenzionalnih modelov v obliki Obj in ga prilagoditi potrebam aplikacije. Potrebno bo implementirati tudi navigacijo s kamero in rotacijo modela tako, da bo uporabniška izkušnja enaka izkušnji aplikacije NeckVeins. Nazadnje pa bo prišla na vrsto implementacija branja Raw tridimenzionalnih podatkov. Te bom obdelal s pomočjo tehnologije WebCL in OpenCL C kodo že napisanega algoritma Marching cubes. Ta bo volumetrične podatke pretvoril v množico trikotnikov, ki jih bom nato izrisal.

Poglavje 2

Pregled orodij in tehnologij

2.1 JavaScript

JavaScript je programski jezik, ki ga je razvilo podjetje Netscape. Je skriptni jezik, ki se sproti tolmači [8]. Prav tako ni tipiziran jezik, kar pomeni, da lahko neka spremenljivka hrani število, niz ali kakšno drugo vsebino. Nima določenega tipa. Je popularen programski jezik, ki je bil sprva namenjen za programiranje znotraj spletnega brskalnika, zadnje čase pa se njegova uporaba širi tudi na številna druga področja.

2.2 Brackets

Brackets je odprtokodni urejevalnik besedila, ki ga je naredilo podjetje Adobe Systems, namenjen razvoju spletnih aplikacij [4]. Omogoča pregled nad celotnim projektom in nekaj osnovnih funkcionalnosti, ki olajšajo sam razvoj. Tako kot drugi urejevalniki besedila, namenjeni za razvoj aplikacij, dopušča tudi Brackets uporabo vtičnikov, ki razširijo njegovo funkcionalnost in razvijalcu olajšajo delo. Podpira tudi razvoj v programskih jezikih, ki niso namenjeni za splet, vendar se v večini primerov ne uporablja za njih.

2.3 WebGL

WebGL (angl. Web Graphics Library) je ovojnica za JavaScript za programiranje aplikacij, narejena na podlagi OpenGL ES 2 [21]. Služi za prikaz interaktivne dvodimenzionalne in tridimenzionalne grafike znotraj spletnega brskalnika s pomočjo grafične procesne enote ali GPU [15]. Z njim lahko izdelamo dvodimenzionalne in tridimenzionalne igre, programe za obdelovanje slik in še mnogo več znotraj našega brskalnika.

2.4 Three.js

Three.js je JavaScript knjižnica, ki nam olajša delo z WebGL-om in nam posledično omogoča hitrejšo ter bolj optimizirano izdelavo spletnih aplikacij [17]. Omogoča dodajanje luči, materialov, objektov, animacijo objektov, branje več vrst datotek in še mnogo več. Delo nam olajša in pohitri, tako da ovije funkcije WebGL-a, kar pomeni, da potrebujemo manj funkcij za želen rezultat.

2.5 OpenCL in WebCL

OpenCL (angl. Open Computing Language) je ogrodje za pisanje programov, ki se izvajajo na heterogenih platformah, ki vsebujejo centralno procesno in grafično procesno enoto. OpenCL definira programski jezik OpenCL C za programiranje teh naprav in vmesnik za programinaje aplikacij, ki je lahko v različnih jezikih in služi za upravljanje naprav, na katerih se izvajajo OpenCL C programi. Omogoča nam vzporedno obdelavo podatkov, kar pomeni, da nam omogoča sočasno izvajanje več izračunov [14, 13]. WebCL (angl. Web Computing Language) je JavaScript ovojnica za OpenCL, ki služi za heterogeno paralelno računanje znotraj spletnih brskalnikov [20]. Omogoča nam upravljanje naprav, na katerih zaganjamo OpenCL C programe s programskim jezikom JavaScript [7].

2.6 Zapis 3D geometrije v datoteki tipa Obj

V računalniški grafiki obstaja veliko različnih tipov datotek za hranjenje tridimenzionalnih objektov in podatkov povezanih z njimi. Veliko se jih je razvilo kot odprti standardi zaradi potrebe po bolj optimiziranem shranjevanju določene vrste podatkov, nekaj pa tudi preprosto zaradi različnih pristopov podjetij, ki izdelujejo programsko opremo za delo s tridimenzionalnimi podatki. Obj je eden izmed najbolj razširjenih podatkovnih tipov za hranjenje 3D geometrije. Predstavlja položaj vsake 3D točke geometrije v prostoru, položaj vsake teksturne točke v prostoru, normalo vsake točke in ploskve, ki sestavljajo poligone definirane s točkami. Privzeto so vse točke zapisane v nasprotni smeri urinega kazalca. Datoteke Obj nimajo enot, vendar lahko vsebujejo podatke o sami velikosti modela. Vrstice, ki se začnejo s črko "v", vsebujejo koordinate točk v 3D prostoru. Vrstice, ki se začnejo z "vt", vsebujejo koordinate točk teksture. Vrstice, ki se začnejo z "vn" vsebujejo točkovne normale, "vp" vsebujejo podatke, namenjene za krivulje ali površine oz. za njihovo uravnovešanje. Vrstice, ki se začnejo s "f", služijo za definiranje ploskev poligona, pri tem je vsaka vrstica en poligon [19]. V datoteki 2.1 lahko vidimo zgradbo datoteke Obj.

```
1  # To je trikotnik
2  # To so tocke
3  v 0 0 0
4  v 1 0 0
5  v 1 1 0
6  # To so normale tock
7  vn 0 0 1
8  vn 0 0 1
9  vn 0 0 1
10 # Pari tocke in normale, ki predstavljajo ploskev
11 f 1/1 2/2 3/3
```

Datoteka 2.1: Primer zapisa datoteke Obj

2.7 Zapis volumetričnih podatkov

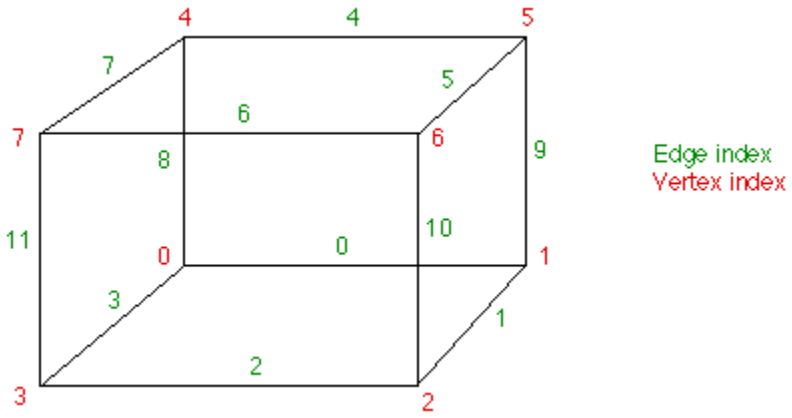
Podatke, pridobljene iz CT naprav, so zdravniki še pred nekaj leti gledali na neintuitiven način v obliki rezin. Z napredovanjem tehnologije pa jim je bil omogočen vpogled v človeško telo v obliki tridimenzonalnih modelov, ki jih lahko obračajo in gledajo iz vseh kotov. CT naprave iz slik tvorijo tridimenzionalne podatke, ki pa jih mi dobimo v obliki parov datotek Mhd in datotek Raw. Datoteko Mhd si lahko predstavljamo kot neke vrste načrt za datoteko Raw. V njej so razni opisni podatki, ki nam povejo lastnosti podatkov, shranjenih v določeni datoteki Raw. Vsebuje dimenzije (velikost) tridimenzionalnega volumna, orientacijo, položaj in podobno. Potrebujemo jo, zato da lahko v programu obdelamo podatke iz datoteke Raw na pravi način. Datoteka Raw vsebuje tridimenzionalen volumen, ki ga obdelujemo. V datoteki 2.2 je prikazan primer zgradbe datoteke Mhd.

```
1  ObjectType = Image
2  NDims = 3
3  DimSize = 512 512 390
4  ElementSpacing = 0.463838 0.463838 0.463838
5  Position = 0 0 0
6  Orientation =  1 0 0
7   0 1 0
8   0 0 1
9  AnatomicalOrientation = R P I
10 ElementByteOrderMSB = False
11 ElementType = MET_SHORT
12 ElementDataFile = Pat2_3D-DSA.raw
```

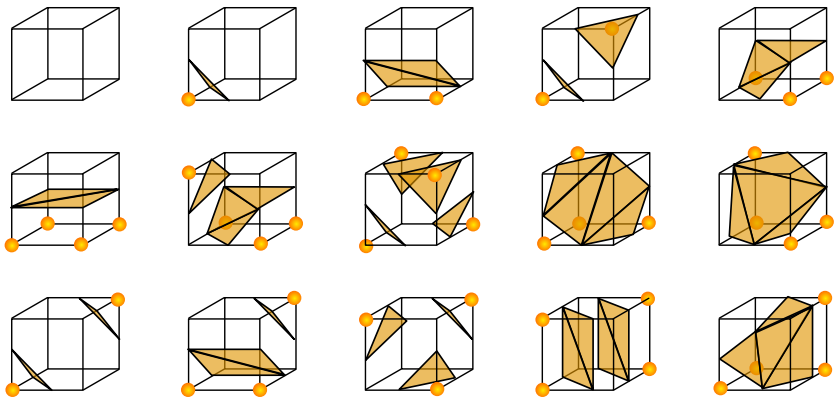
Datoteka 2.2: Primer zapisa datoteke Mhd

2.8 Algoritem Marching cubes

Glavni algoritem, uporabljen za izluščenje tridimenzionalnega modela iz datoteke Raw, je Marching cubes. Izdelala sta ga William E. Lorensen in Harvey E. Cline kot del raziskave za podjetje General Electric [23]. Algoritem razdeli prostor na enako velike kocke, ki so definirane z 8 točkami v prostoru. Nato pregledamo, kje naš tridimenzionalen volumen, definiran v datoteki Raw, seka vsako od kock in ali je del, ki seka kocko, dejansko del objekta, ki ga želimo prikazati ali pa je samo množica nekih podatkov, ki so del ozadja. Primer oštevilčevanja oglišč kocke lahko vidimo na sliki 2.1. Volumen lahko sploh ne seka naše kocke, lahko seka eno oglišče, lahko pa seka več oglišč v eni od vnaprej definiranih kombinacij. Glede na to, katera oglišča seka, izračunamo indeks. Če naš volumen seka kocko v ogliščih 1, 5 in 6, to pomeni, da je to na seznamu površin indeks z binarno vrednostjo 01100010. Na seznamu vseh možnih površin znotraj kocke izberemo tisto, ki se nahaja na izračunanem indeksu. Ta površina bo predstavljala del našega volumna, ki je sekal to kocko. Za to površino imamo 256 možnosti, ki jih je mogoče zmanjšati na 15 možnosti. Vidimo jih na sliki 2.2, če upoštevamo to, da lahko vsako od možnosti rotiramo in zrcalimo [22].



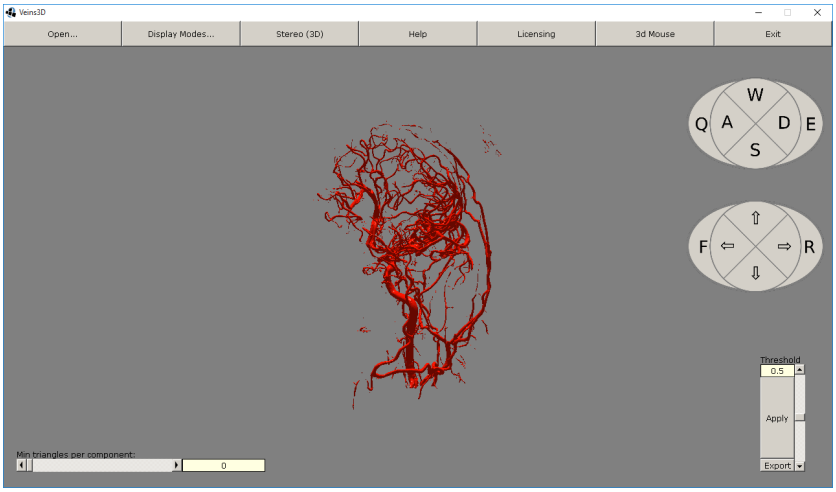
Slika 2.1: Oštevilčevanje kocke v marching cubes algoritmu [22]



Slika 2.2: 15 možnosti postavitve površin v algoritmu Marching cubes [11]

2.9 Aplikacija NeckVeins

Namen aplikacije NeckVeins je omogočiti zdravnikom lažji in bolj intuitiven pogled na slike žil, pridobljene z napravo CT. Omogoča odpiranje datotek Obj, ki so predstavljene s točkami v prostoru in povezavami med njimi. Odpira lahko tudi pare datotek Mhd in Raw. Datoteke Mhd vsebujejo podatke o sestavi volumetričnih podatkov žil v datotekah Raw. Branje datotek Mhd in Raw je bilo implementirano s pomočjo algoritma Marching cubes. Omogoča tudi rotacijo in premikanje kamere v vse smeri ter rotacijo tridimenzionalnega objekta. Uporabnik lahko upravlja kamero in objekt z navadno miško, s tipkovnico, s tridimenzionalno miško ali pa z napravo Leap Motion, lahko pa se tudi odloči uporabljati gumbe izrisane na ekranu. Podpira več resolucij ekrana ter stereo izris modela. Aplikacijo so razvijali številni študentje v Laboratoriju za računalniško grafiko in multimedije v obliki diplomskih nalog. Sprogramirana je bila v programskem jeziku Java in z uporabo knjižnice LWJGL [10]. Osnovno aplikacijo je sprogramiral Simon Žagar [25]. Dodatek za odpiranja datotek Raw pa je dodal Anže Sodja [24]. Aplikacija bo tudi v kratkem dobila nov izgled, ki pa trenutno še ni vgrajen. Na sliki 2.3 lahko vidite izgled trenutne namizne aplikacije NeckVeins.



Slika 2.3: Trenutna NeckVeins aplikacija

Poglavje 3

Implementacija

3.1 Izbira in namestitev primernega okolja

Za spletni brskalnik sem izbral Firefox, in sicer različico 34, saj je to zadnja različica, v kateri WebCL vtičnik še deluje [12]. Pri tem je prišlo do zanimivega problema, saj urejevalnik Three.js ni bil narejen z mislijo na kompatibilnost s starejšimi brskalniki. Ob prvem zagonu se je polje za prikaz tridimenzionalnega sveta vsakič odprlo v majhni obliki. To sem rešil tako, da sem vsakič ob zagonu urejevalnika sprožil dogodek "resize", ki pa je osvežil polje za prikaz in s tem popravil njegovo velikost.

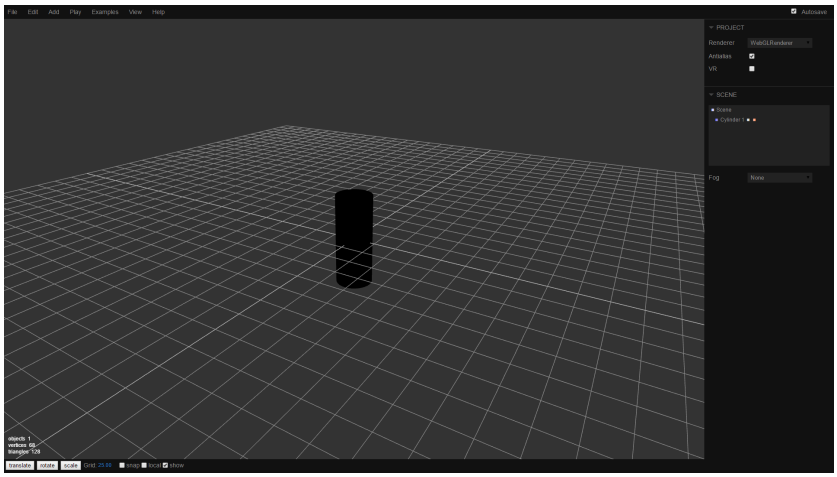
3.2 Prilagajanje urejevalnika Three.js

3.2.1 Urejevalnika Three.js

Urejevalnik Three.js je spletna aplikacija, narejena s knjižnico Three.js, ki omogoča osnovne funkcionalnosti orodja za tridimenzionalno oblikovanje. Omogoča dodajanje osnovnih geometrijskih teles v svet, dodajanje luči in kamer, premikanje in rotiranje objektov, spreminjanje raznih lastnosti objektov, luči, kamer in celotnega sveta ter branje in shranjevanje nekaj osnovnih formatov za predstavitev tridimenzionalnih objektov in njihovih lastnosti.

3.2.2 Odstranjevanje funkcionalnosti

Urejevalnik Three.js ima v svoji osnovni obliki veliko več funkcionalnosti za oblikovanje in dodajanje novih objektov, kot jih je dejansko potrebno za ogledovanje tridimenzionalnih volumnov, zato sem jih večino odstranil. Sprva sem odstranil nekaj nepomembnih menijev iz zgornje orodne vrstice, kot so meni za primere, za spreminjanje samega izgleda urejevalnika in možnost predvajanja dodatnih skript, ki jih sami napišemo ter večino možnosti dodajanja in shranjevanja objektov. Odstranil sem tudi spodnjo orodno vrstico z dodatnimi možnostmi za prikazovanje tridimenzionalne mreže, pripenjanje objektov na njena presečišča in rotiranje, skaliranje ter translacijo objektov. Nazadnje sem odstranil tudi desno orodno vrstico, namenjeno prikazovanju in spreminjanju lastnosti načina izrisovanja, zgradbe našega trenutnega sveta in objektov v njem, lastnosti teh objektov, kot so materiali, trenuten položaj, rotacija in velikost objekta ter dodatne skripte in še nekaj drugih lastnosti. Ko sem vse odstranil, sem moral določiti tudi novo velikost in novo razmerje same izrisovalne površine, na kateri se izrisuje naš objekt. Vse funkcionalnosti sem odstranil tako, da sem jih izbrisal ali zakomentiral. Na sliki 3.1 lahko vidite prvotno stanje urejevalnika pred odstranjevanjem funkcionalnosti.



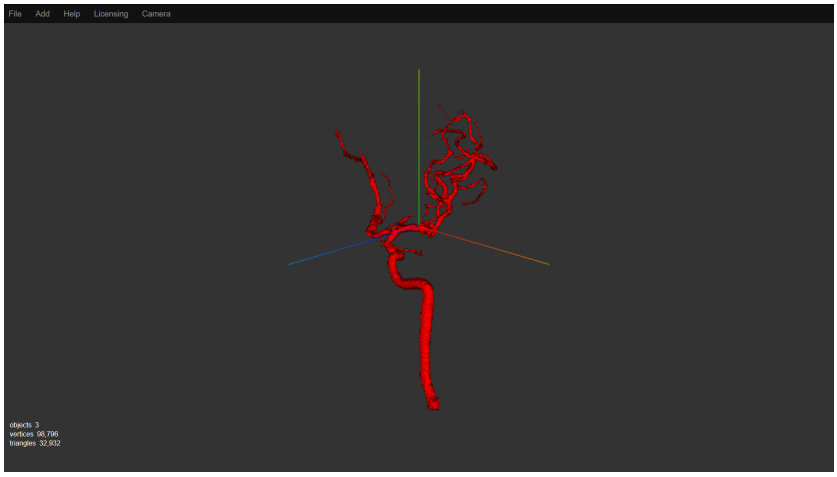
Slika 3.1: Stanje urejevalnika Three.js pred odstranjevanjem funkcionalnosi

3.2.3 Prikaz 3D objektov

Ko sem naložil objekt, sem ga moral sprva postaviti v središče tridimenzionalnega sveta in mu dodati pravilno barvo. Objekt sem postavil v središče, tako da sem izračunal povprečje položajev vseh točk. Ko sem to naredil, sem od vseh točk odštel to povprečje in s tem premaknil objekt nazaj v središče. Nato sem materialu objekta določil rdečo barvo. Zdaj je bil objekt pravilno postavljen in pravilno obarvan, vendar pa ni bil osvetljen. V tridimenzionalni svet sem dodal še točkasto luč kot izvor svetlobe, ki sveti v vse smeri z enako močjo. Njena moč upada z oddaljenostjo od izvora. Točkasta luč mi je omogočila, da objekt vidim bolj jasno in ugotovim, da še vedno ni popolnoma pravilno prikazan. Normale objekta so bile zrcaljene. To je povzročilo, da se je videlo v notranjost objekta. Vse normale sem negiral in s tem dobil pravilen prikaz objekta. Za konec sem objekt še rotiral za -45 stopinj po Z osi in za -132,5 stopinj po Y osi. Končni prikaz tridimenzionalnega modela je mogoče videti na sliki 3.2.

Normale

Vsaka točka na površini ima vektor imenovan normala, ki je pravokotnica na površino. Če se več površin stika skupaj, se normale točk, skupnih z vsemi površinami, povprečijo. Tako dobimo eno samo normalo za vsako točko. Z njimi določimo, katera stran tridimenzionalnega modela je zunanja in katera notranja. Služijo izračunu pri osvetljevanju objektov, računanju fizikalnih simulacij in še marsičemu.



Slika 3.2: Prikaz tridimenzionalnega modela shranjenega v Obj datoteki

3.2.4 Nadziranje kamere

Kameri sem sprva dodal možnost postavitve v prvotno stanje z gumbom "reset". Ko uporabnik klikne gumb, se položaj kamere postavi v prvoten položaj in pogled kamere postavi v središče sveta. Prav tako pazim na to, da luč postavim za kamero, saj tako dobim pravo osvetlitev modela. To sem naredil tako, da sem vzel vrednosti kamerinega vektorja pogleda, ki kaže v smer, v katero kamera trenutno gleda in jih pomnožil s skalarjem ter jih odštel od trenutnega položaja kamere. Luč sem nato postavil na izračunan položaj. Kamera se lahko premika in rotira po vseh oseh. Ko se kamera premika, se z njo premika tudi luč. Tako lahko dobimo dobro osvetljen objekt iz vseh vidnih kotov.

3.2.5 Nadziranje objekta

Rotacijo sem prav tako implementiral pri objektu in s tem ponudil uporabniko še eno možnost več za upravljanja svojega pogleda. Pomagal sem si s kodo iz repozitorija [18]. Pri rotaciji objekta si lahko pomagamo z rotacijsko matriko.

Rotacijska matrika

Rotacijska matrika je matrika, v kateri definiramo rotacijo za naše objekte oziroma za točke naših objektov. Ko izvedemo rotacijo, se točke rotirajo kot celota in posledično ostanejo razdalje med njimi enake. V rotacijski matriki definiramo vse rotacije s sinusi in kosinusi.

Pojav kardanska zapora

Tridimenzionalen objekt lahko rotiramo v treh smereh oziroma treh dimenzijah. Postopek rotacije opišemo z rotacijsko matriko. Ta lahko opiše rotacijo v nekem vrstnem redu, kar pomeni, da so rotacijske dimenzije med sabo odvisne. Bolj natančno, druga dimenzija je odvisna od prve, tretja pa je odvisna od prvih dveh. Torej če predmet zarotiramo za 30 stopinj po X osi, sta se drugi dve osi zarotirali skupaj s predmetom za 30 stopinj po X osi. Pri temu lahko pride do poravnave dveh osi oziroma kardanske zapore (angl. gimbal lock). V takem primeru izgubimo eno os rotacije oz. eno prostostno stopnjo in lahko objekt rotiramo samo še po dveh oseh. Zaradi tega moramo za samo rotacijo uporabljati kvaternione [16]. Ti služijo kot drugačna predstavitev orientacije objekta, s katero se izognemo pojavu kardanske zapore. Na sliki 3.3a lahko vidite normalno stanje, na sliki 3.3b pa je prikazan primer kardanske zapore.

(a) Normalno stanje (b) Kardanska zapora

Slika 3.3: Prikaz normalnega stanja in stanja kardanske zapore [6]

3.3 WebCL implementacija Marching Cubes

3.3.1 Osnoven opis WebCL-a

WebCL je sestavljen iz več osnovnih enot, ki imajo vsaka svoj namen. Delovna enota (angl. work item) je najmanjša enota, ki izvaja en sam izračun na enem samem delu naših podatkov. Več delovnih enot se združi v delovno skupino (angl. work group), kar dejansko pomeni ena skupina delovnih enot, ki se izvaja skupaj. Teh skupin je več. Jedro (angl. kernel) predstavlja funkcijo s programsko kodo ene same delovne enote. Naš program v OpenCL C-ju ima lahko več jeder in drugih funkcij, potrebnih za pravilno delovanje programa. Kontekst je okolje, v katerem se izvajajo naša jedra oz. naše delovne enote. Predstavlja vse naprave, spomine naprav, ukazne vrstice in še mnogo več.

3.3.2 Zgradba pomnilnika

Gostitelj vsebuje več naprav za izračune (angl. compute devices), kot so grafična procesna enota in centralna procesna enota. Vsaka naprava za izračune vsebuje več računskih enot (angl. compute units) ali jeder. Vsako jedro pa vsebuje več elementov za procesiranje ali niti (angl. hardware threads). Vsak gostitelj, naprava za izračune in nit imajo svoj pomnilnik. Pomnilnike delimo na globalni pomnilnik (angl. global memory), pomnilnik za konstantna števila (angl. constant memory), lokalni pomnilnik (angl. local memory) in zasebni pomnilnik (angl. private memory). Vsaka delovna enota ima zasebni pomnilnik, vsaka delovna skupina ima lokalni pomnilnik. Globalni pomnilnik in pomnilnik za konstantna števila si delijo vse delovne enote.

3.3.3 JavaScript tipizirana polja

JavaScript v osnovi ni tipiziran jezik. Tipizirana polja (angl. typed arrays) so objekti, podobni navadnim poljem, ki dovoljujejo dostop do binarnih podat-

kov. Normalna polja svojo velikost dinamično prilagajajo glede na količino podatkov, ki jih shranjujemo, medtem ko za tipizirana polja tega ne moremo reči. Kljub vsem optimizacijam z večanjem velikosti polja in dodajanjem novih elementov upočasnimo aplikacijo. Tipizirana polja so fiksne velikosti in so namenjena za hitre dostope do binarnih podatkov. Razdeljena so v medpomnilnike (angl. buffers) in poglede (angl. views). Medpomnilnik je le kup podatkov in do njihovih vrednosti ne moremo dostopati. Potrebujemo pogled, ki nam bo določil tip podatkov v medpomnilniku in način tolmačenja teh podatkov. Lahko jih tolmačimo kot bajte, cela števila ali števila s plavajočo vejico [9].

3.3.4 Vračanje napak v WebCL-u

Vse WebCL funkcije vrnejo izjeme, za razliko od OpenCL funkcij, ki vrnejo šifro napake. To pomeni, da lahko WebCL funkcije vstavimo v try-catch stavek in ne prekinemo delovanja programa v primeru, da pride do izjeme, ter jo tam obravnavamo. Zaradi tega večino kritičnih WebCL ukazov vstavimo v try-catch stavek.

3.3.5 Začetek razvoja aplikacije z WebCL

Pred začetkom dejanske javascript kode za inicializacijo vseh potrebnih delov WebCL-a moramo preveriti, če trenutni brskalnik sploh podpira WebCL, saj ga trenutno podpira zelo majhen delež brskalnikov. V primeru, da ga ne, o tem obvestimo uporabnika in prenehamo z izvajanjem. Ko dobimo potrdilo o podpori, izberemo seznam vseh naprav, primernih za poganjanje WebCL-a in izberemo željeno napravo. S to napravo inicializiramo kontekst, ki služi kot most med našo WebCL ovojnico in OpenCL C programom. Incializiramo tudi ukazno vrste. Ukazi se pošljejo iz gostiteljskega programa v jedro preko ukazne vrste. Nazadnje prevedemo program. Zdaj imamo vse pripravljeno za nadaljnje pripravljanje in izvajanje jeder.

3.3.6 Splošni postopek inicializacije in zagona jedra s tehnologijo WebCL

Inicializiranje in zagon jedra lahko delimo na štiri glavne dele. Na podajanje vseh podatkov medpomnilniku, zaganjanje samega jedra, pridobivanje podatkov iz medpomnilnikov, ki smo jih definirali kot izhodni medpomnilniki ter brisanje nepotrebnih medpomnilnikov.

Podajaje podatkov medpomnilniku

Sprva moramo podatke pretvoriti v tipizirano polje, saj le tako program tolmači podatke netipiziranega JavaScripta in poda te informacije OpenCL C-ju. Nato moramo s pomočjo konteksta narediti medpomnilnik. Pri tem tudi določimo, kakšne pravice dostopa želimo dati jedrom za ta medpomnilnik. Lahko določimo, da jedro samo bere, samo piše ali pa bere in piše v medpomnilnik. Iz spletne aplikacije ne moremo direktno dostopati do WebCL-ovih oziroma OpenCL-ovih pomnilnikov. Nazadnje, s pomočjo ukazne vrstice, podatek zapišemo iz našega tipiziranega polja v medpomnilnik.

Zaganjanje jedra

Da bi jedro zagnali, ga moramo sprva ustvariti. Ustvarimo jedro, ki mu nato podamo vse medpomnilnike, definirane za njegovo uporabo. Določimo jih kot argumente, podobno kot bi to storili pri normalni funkciji, le da tukaj vsak argument posebej definiramo s funkcijo `setArg`. Nato moramo še jedru povedati, kolikšen je obseg problema in koliko je vseh delovnih enot, ki jih želimo uporabiti za reševanje tega problema. Lahko določimo tudi velikost delovne skupine, vendar se moramo zavedati, da je ta omejena in različna na vsaki napravi.

Pridobivanje podatkov iz medpomnilnikov

Tako, kot smo morali za podajanje podatkov medpomnilniku definirati tipizirano polje, moramo to storiti tudi pri pridobivanju teh podatkov. Nato s

funkcijo te podatke prepišemo iz medpomnilnika v naše polje.

Brisanje nepotrebnih medpomnilnikov

JavaScript ne dovoljuje direktnega posega v podatke, zapisane v pomnilniku. Sam WebCL pa to za svoje medpomnilnike dovoljuje, zato je potrebno poskrbeti za redno čiščenje podatkov iz pomnilnika. Na koncu izvedbe vsakega jedra je potrebno počistiti vse uporabljene medpomnilnike, ki jih ne bomo več uporabljali. Prav tako je potrebno počistiti samo jedro v primeru, da ga ne bomo ponovno zaganjali.

3.3.7 Branje podatkov

Ker javascript nima dostopa do uporabniških podatkov, shranjenih na uporabnikovem računalniku, moramo prebrati obe datoteki hkrati s pomočjo HTML input elementa. Ko preberemo datoteko Mhd, jo dobimo v obliki niza podatkov. Ta niz razdelimo po presledkih, nato pa v zanki pregledamo vse podatke in jih zapišemo v polje. Ob končanem branju preverim, če se ujemata imeni datoteke Raw, navedene v naši datoteki Mhd in dejanske datoteke Raw, ki smo jo prebrali. V primeru ujemanja, začnemo s prenašanjem Raw podatkov v polje, kjer jih preoblikujemo v primerno obliko za jedro.

3.3.8 Predelovanja volumetričnih podatkov

Ko podatke preberemo in shranimo v polje, jih najprej razdelimo v več manjših polj, ki zavzemajo približno 50 milijonov elementov ali manj. To nam bo koristilo pozneje, saj ima WebCL buffer omejitev velikosti in če jo prekoračimo, povzroči napačne rezultate izračunov. Nato vsak del podatkov pošljemo skozi glavno zanko, ki podatke sprva pretvori s pravilom kratkega konca v 16 bitna števila, ki se shranijo v 32 bitni zapis s plavajočo vejico. Na novo izračunane podatke pošljemo skozi algoritem Gaussovega glajenja, ki gladi sprva po dimenziji X nato po dimenziji Y in nazadanje po dimenziji Z. S tem dosežemo manjšo kompleksnost in posledično hitrejše izvajanje Ga-

ussovega glajenja. Algoritem nam vrne zglajene podatke, ki so primernejši za prikaz uprabniku. Nato na zglajenih podatkih za potrebe Otsujevega upragovanja najdemo maksimalno vrednost. Izvedemo prvi del algoritma Otsujevega upragovanja, ki zgenerira histogram različnih vrednosti v naših podatkih. Ob vskem obhodu glavne zanke te histograme shranjujemo v polje. Ko se izvajanje glavne zanke zaključi, seštejemo histograme v enega. Ta histogram nato pošljemo skozi drugi del Otsujeve metode, ki izračuna dejansko mejno vrednost med našim objektom in ostalimi podatki. Nazadnje zaženemo v novi zanki algoritem marching cubes, ki iz zglajenih podatkov in določene meje, pridobljene z algoritmom Otsujevega upragovanja, ter maksimalno vrednostjo, pridobi trikotnike, ki sestavljajo naš objekt, število teh trikotnikov in njihove normale. Pridobljene vrednosti ponovno shranjujemo v polje. Podatke nato združimo skupaj in izrišemo.

3.3.9 Opis Jeder

Jedro za Gaussovo glajenje

Za Gaussovo glajenje imamo tri jedra, ki služijo za izvajanje algoritma Gaussovega glajenja po vseh treh dimenzijah. Rezultat bi bil enak, če bi izvedli algoritem po vseh treh dimenzijah istočasno, vendar pa bi bila sama zahtevnost algoritma veliko večja. S tem dobimo bolj gladke podatke brez ostrih prehodov, ki so bolj primerni za prikaz uporabniku.

Jedro za izračun maksimuma

Jedro iz vseh podatkov najde maksimalno vrednost tako, da iterira skozi vhodne podatke. Ta vrednost nam pozneje pomaga pri izračunu Otsujevega pragu in izvedbi algoritma Marching cubes.

Jedro za Otsujevo upragovanje

To jedro služi za izračun meje, ki določi, kaj bomo upoštevali kot objekt za prikaz in kaj bomo upoštevali kot okolico. Torej vzame volumenske po-

datke datoteke Raw in iz njih s pomočjo algoritma Otsujevega upragovanja izračuna število, ki predstavlja prag med našim objektom in ostalimi podatki. Otsujevo upragovanje se v osnovi uporablja pri pretvorbi sivinskih slik v črno bele slike tako, da določi pragovno vrednost, ki jo nato uporabljamo kot mejo med sivinskimi točkami, ki jih pretvorimo v bele, in sivinskimi točkami, ki jih pretvorimo v črne točke. V našem primeru določi mejo med objektom in ostalimi podatki. Sestavljen je iz dveh delov. Iz jedra, ki nam izračuna histogram, sestavljen iz zglajenih vrednosti datoteke Raw, in funkcije, ki ta histogram predela in iz njega dobi dejanski prag. Histograme izračunamo tako, da vrednosti datoteke Raw raztegnemo s pomočjo maksimuma na vrednosti od 0 do 255. Samo jedro izvajamo znotraj glavne zanke, medtem ko funkcijo za obdelovanje izvajamo posebej na celotnih podatkih.

Marching cubes jedro

To jedro izvede algoritem marching cubes. Algoritem iz zglajenih vhodnih podatkov, s pomočjo maksimalne vrednosti in pragu, pridobljenega z Otsujevim upragovanjem, izračuna trikotnike, ki sestavljajo objekt, njihove normale in število trikotnikov. Pri tem uporabi prag za določanje tega, kaj je objekt in kaj je del okolice, ki nas ne zanima. To jedro se izvede večkrat v svoji zanki. Ob vsakem obhodu zanke se novo pridobljene trikotnike in normale zapiše v dvodimenzionalno polje, število končnih trikotnikov pa se sešteva.

3.3.10 Prikaz podatkov

Podatki, ki sem jih pridobil iz Marching cubes jedra, so se sekali, saj je jedro pri vsakem delu podatkov začelo z indeksiranjem od začetka. Sprva sem vsak del zamaknil po Z osi, tako da so bili poravnani, in jih sproti združeval. Združene podatke sem nato postavil v središče sveta, tako kot pri prikazovanju navadnega objekta iz datoteke Obj. Nazadnje sem jim dodal primeren material in jih izrisal.

3.3.11 Problemi in omejitve

Začetni problem je bil predvsem to, da je na spletu zelo malo materiala povezanega z WebCL-om in njegovim delovanjem. To je pomenilo, da sem moral na začetku izvesti veliko poskusov, ki so mi omogočili pravilno razumevanje WebCL-a in dela z njim.

Omejitev velikosti medpomnilnika

Ugotovil sem, da obstaja omejitev pri velikosti podatkov, ki jih lahko shranimo v medpomnilnik. Podatki, ki so večji od 200MB, bodo povzročili težave pri samih izračunih. To sem rešil tako, da sem osnovne podatke razdelil na enakomerne dele, ki so manjši ali enaki 200MB. Zaradi tega sem moral prilagoditi tudi nekatera jedra in jim podati velikost podatkov za izračune.

JavaScript nadziranje pomnilnika

Tako JavaScript kot večina drugih visokonivojskih jezikov ne ponuja možnosti direktnega nadzorovanja pomnilnika. Zato sem moral paziti, da bom le določene večje podatke hranil v pomnilniku. Podatke za brisanje določi s pomočjo algoritma, imenovanega pobiranje smeti (angl. garbage collection), ki pobriše vse nepotrebne podatke. Podatke za brisanje določi glede na to, ali obstaja možnost za dostop do njih ali ne oziroma ali obstaja referenca do njih nekje v programu ali ne. Aplikacija v trenutni obliki hrani le podatke iz datoteke Raw v enem polju.

3.4 Primerjanje hitrosti in ugotovitev

Po zaključeni izdelavi spletne aplikacije sem izmeril hitrost izvedbe vseh jeder in prvega izrisa v spletni aplikaciji in namizni aplikaciji ter ju primerjal. Nato sem izmeril še hitrost izrisovanja objekta v primeru obračanja objekta in v primeru mirovanja objekta na isti način. Testiral sem na 150MB veliki datoteki Raw in 195MB veliki datoteki Raw. Vsa testiranja sem opravil

na prenosnem računalnik s 64 bitnim operacijskim sistemom Windows 10, z grafično kartico nVidia Geforce GTX 850M, 8GB DDR3 rama, Intel Core i7-4710HQ procesorjem s hitrostjo 2.5 GHz in 1TB velikim trdim diskom s 5400 obratov na minuto. Vsi rezultati so bili pridobljeni tako, da sem izmeril isto stvar petkrat in nato povprečil rezultate. Kot napravo za izvajanje OpenCL jeder sem izbral grafično kartico.

3.4.1 Meritev hitrosti algoritma in prvega izrisa

Rezultati meritev ob prvem zagonu

Sprva sem izmeril čas od branja do prvega izrisa objekta ob svežem zagonu sistema. Vsi programi, razen potrebovanega programa za izris in sistemskih programov, so bili ugasnjeni. Rezultati meritev so vidni v tabeli 3.1.

Velikost datoteke	**Spletna aplikacija**	**Namizna aplikacija**
150MB	10150ms	11398ms
195MB	14175ms	13124ms

Tabela 3.1: Rezultati meritev časa od branja datoteke do prvega izrisa v spletni aplikaciji in namizni aplikaciji ob prvem zagonu.

Rezultati meritev po prvem zagonu

Druga meritev hitrosti je bila po prvem zagonu, ko sem že enkrat izrisali nek model. Večkrat sem zaporedno odpiral model velikosti 150MB in 200MB in meril hitrost. Rezultati meritev so vidni v tabeli 3.2.

Velikost datoteke	Spletna aplikacija	Namizna aplikacija
150MB	4910ms	5090ms
195MB	6356ms	6225ms

Tabela 3.2: Rezultati meritev časa od branja datoteke do prvega izrisa v spletni aplikaciji in namizni aplikaciji po prvem zagonu.

Sklep iz meritev hitrosti algoritma in prvega izrisa

Ugotovil sem, da ni večjih razlik v hitrosti nalaganja in izrisa modelov. Manjše razlike, ki so nastale, lahko pripišemo samemu sistemu, različnem načinu pisanja spletne aplikacije in namizne aplikacije ter razliki v hitrosti jezikov JavaScript in Java.

3.4.2 Meritev hitrosti izrisa pri prikazovanju in rotiranju objekta

Rezultati meritev v stanju mirovanja

Pri tej meritvi sem meril povprečen čas potreben za en obhod glavne zanke za izrisovanje, ko je objekt že naložen in miruje. Izmeril sem petkrat sto obhodov zanke in jih med sabo povprečil. Nato sem pretvoril povprečno hitrost enega obhoda v število obhodov na sekundo. Rezultate meritve so vidni v tabeli 3.3.

Velikost datoteke	Spletna aplikacija	Namizna aplikacija
150MB	227,5 FPS	666,4 FPS
195MB	271,3 FPS	568,4 FPS

Tabela 3.3: Rezultati meritev števila obhodov glavne zanke za izrisovanje, ko objekt miruje

Rezultati meritev med obračanjem

Pri tej meritvi sem meril povprečen čas potreben za en obhod glavne zanke za izrisovanje, ko objekt obračam. Izmeril sem petkrat sto obhodov zanke tako kot pri stanju mirovanja in jih med sabo povprečil. Pretvoril sem povprečno hitrost enega obhoda v število obhodov na sekundo. Rezultati meritve so vidni v tabeli 3.4.

Velikost datoteke	**Spletna aplikacija**	**Namizna aplikacija**
150MB	53,8 FPS	662,9 FPS
195MB	69,6 FPS	567,6 FPS

Tabela 3.4: Rezultati meritev števila obhodov glavne zanke za izrisovanje, ko se objekt rotira

Sklep iz meritev hitrosti algoritma in prvega izrisa

Ugotovil sem, da so pri izrisovanju velike razlike v številu obhodov zanke za izrisovanje pri spletni različici in namizni različici. To se še posebej vidi iz meritev hitrosti pri rotiranju objekta. Pri manj kompleksnih datotekah to seveda ni problem, vendar se bo to lahko pokazalo kot problem pri večjih datotekah. Ta razlika v izrisovanju je lahko posledica drugačnega načina programiranja, saj sta bila različna progamerja, in počasnosti JavaScripta v primerjavi z Javo.

Poglavje 4

Zaključek

V končnem projektu sem implementiral upravljanje s kamero in z modelom ter odpiranje datotek Raw. Pri tem sem poskrbel za pravilno branje podatkov o datoteki Raw iz datoteke Mhd in obdelovanje podatkov v datoteki Raw s pomočjo poskusne tehnologije WebCL in kode iz namizne aplikacije NeckVeins. Dokazal sem, da je možno realizirati isto funkcionalnost tako v namizni aplikaciji kot tudi v spletni aplikaciji, vendar pa je WebCL tehnologija trenutno zelo omejena, saj deluje samo na nekaj brskalnikih in ne na vseh. Delo sem zaključil z meritvijo hitrosti spletne in namizne aplikacije in ugotovil, da sta približno enako hitri, ko pride do obdelovanja volumetričnih podatkov in njihovega prvega izrisa. Pri meritvah hitrosti izrisovanja v mirovanju in pri obračanju predmeta pa sem ugotovil, da je namizna različica bistveno hitrejša. V prihodnosti bo predstavljena tudi nova različica WebGL-a, ki bo morda ponudila podobno funkcionalnost, kot jo ima WebCL. V takem primeru bi bilo dobro trenutno kodo spletne aplikacije prenesti na WebGL 2.0. V nadaljevanju dela bi lahko dodal še več možnosti za prikazovanje objektov in upravljanje z njimi.

Literatura

[1] arivis WebView. http://vision.arivis.com/en/arivis-WebView-3D-3D-Volume-Rendering-in-Web-Browsers. Dostopano 12.9.2015.

[2] Babylon.js. https://github.com/BabylonJS/Babylon.js. Dostopano 6.9.2015.

[3] Bodyviz. https://www.bodyviz.com/. Dostopano 12.9.2015.

[4] Brackets (text editor). https://en.wikipedia.org/wiki/Brackets_(text_editor). Dostopano 6.9.2015.

[5] CT scan. https://en.wikipedia.org/wiki/CT_scan. Dostopano 6.9.2015.

[6] Gimbal lock. https://en.wikipedia.org/wiki/Gimbal_lock. Dostopano 10.9.2015.

[7] Heterogeneous parallel computing in HTML5 web browsers. https://www.khronos.org/webcl/. Dostopano 6.9.2015.

[8] JavaScript. https://en.wikipedia.org/wiki/JavaScript. Dostopano 6.9.2015.

[9] JavaScript typed arrays. https://developer.mozilla.org/en/docs/Web/JavaScript/Typed_arrays. Dostopano 9.9.2015.

[10] LWJGL. http://www.lwjgl.org/. Dostopano 12.9.2015.

[11] Marching cubes. https://en.wikipedia.org/wiki/Marching_cubes. Dostopano 6.9.2015.

[12] Nokia WebCL. http://webcl.nokiaresearch.com/. Dostopano 6.9.2015.

[13] The open standard for parallel programming of heterogeneous systems. https://www.khronos.org/opencl/. Dostopano 6.9.2015.

[14] OpenCL. https://en.wikipedia.org/wiki/OpenCL. Dostopano 6.9.2015.

[15] OpenGL ES 2.0 for the Web. https://www.khronos.org/webgl/. Dostopano 6.9.2015.

[16] Quaternion. https://en.wikipedia.org/wiki/Quaternion. Dostopano 9.9.2015.

[17] Three.js. https://github.com/mrdoob/three.js. Dostopano 6.9.2015.

[18] Three.js-Object-Rotation-with-Quaternion. https://github.com/defmech/Three.js-Object-Rotation-with-Quaternion. Dostopano 8.9.2015.

[19] Wavefront .obj file. https://en.wikipedia.org/wiki/Wavefront_.obj_file. Dostopano 6.9.2015.

[20] WebCL. https://en.wikipedia.org/wiki/WebCL. Dostopano 6.9.2015.

[21] WebGL. https://en.wikipedia.org/wiki/WebGL. Dostopano 6.9.2015.

[22] Paul Bourke. Polygonising a scalar field. http://paulbourke.net/geometry/polygonise/, May 1994. Dostopano 6.9.2015.

[23] William E. Lorensen and Harvey E. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '87, pages 163–169, New York, NY, USA, 1987. ACM.

[24] Anže Sodja. Segmentacija prostorskih medicinskih podatkov na gpe, 2013.

[25] Simon Žagar. Vizualizacija žil tilnika z OpenGL-om, 2012.