

UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Jure Kolenko

**Orodje za ročno poravnavo 2D in 3D
medicinskih posnetkov**

DIPLOMSKO DELO
UNIVERZITETNI ŠTUDIJSKI PROGRAM PRVE STOPNJE
RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: doc. dr. Matija Marolt

Ljubljana 2015

Fakulteta za računalništvo in informatiko podpira javno dostopnost znanstvenih, strokovnih in razvojnih rezultatov. Zato priporoča objavo dela pod katero od licenc, ki omogočajo prosto razširjanje diplomskega dela in/ali možnost nadaljne proste uporabe dela. Ena izmed možnosti je izdaja diplomskega dela pod katero od Creative Commons licenc <http://creativecommons.si>

Morebitno pripadajočo programsko kodo praviloma objavite pod, denimo, licenco *GNU General Public License*, različica 3. Podrobnosti licence so dostopne na spletni strani <http://www.gnu.org/licenses/>.

Besedilo je oblikovano z urejevalnikom besedil L^AT_EX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Fiddy diddling.

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Jure Kolenko sem avtor diplomskega dela z naslovom:

Vizualizacija medicinskih podatkov

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 24. avgusta 2015

Podpis avtorja:

Thanks Obama.

Posvetilo.

Kazalo

Povzetek

Abstract

1	Uvod	1
2	Metode in orodja	3
2.1	Java in Eclipse	3
2.2	Hranjenje mrežnega modela	4
2.3	Aplikacija za vizualizacijo	5
2.4	Strojno pospešen izris 3D grafike	5
2.5	Projekcija teksture na 3D model	6
2.6	Nabor podatkov	8
3	Implementacija	9
3.1	Programska ovojnica za LWJGL	9
3.2	Samostojna aplikacija	14
3.3	Integracija v aplikacijo za vizualizacijo	15
4	Zaključek	17

Seznam uporabljenih kratic

Povzetek

V vzorcu je predstavljen postopek priprave diplomskega dela z uporabo okolja L^AT_EX. Vaš povzetek mora sicer vsebovati približno 100 besed, ta tukaj je odločno prekratek.

Ključne besede: Muh keyword1, hue.

Abstract

This sample document presents an approach to typesetting your BSc thesis using L^AT_EX. A proper abstract should contain around 100 words which makes this one way too short.

Keywords: Muh keyword1, hue.

Poglavje 1

Uvod

Računalniki so že dolgo pomembni za ljudi, v zadnjih časi pa še toliko bolj, saj so vedno zmogljivejši, močnejši in cenejši. Z razvojem računalnikov se je razvila tudi tehnologija za vizualizacijo, ki močno izboljša predstavo informacij v primerjavi z golimi številkami.

V medicini imamo množico slikovnih podatkov, ki pa jih te težko vizualizirati na zaslonu oziroma papirju, ker so objekti interesa 3 dimenzionalni (od tu dalje 3D), papir pa 2 dimenzionalen (od tu dalje 2D). Tu pride v poštev vizualizacija, tako da posnetke vidimo na zaslonu, tako kot dejansko izgledajo v realnosti.

Cilj moje diplomske naloge je bil narediti orodje za poravnavo 2D in 3D medicinskih podatkov, ki bi zdravnikom pomagal razbrati kje v prostoru se nahajajo deli 2D posnetka. Uporabnik naj bi izbral mrežni model ožilja in pripadajoč rentgenski posnetek, nato pa z rotiranjem in premikanjem modela oziroma slike poskrbel, da se ujemata. V naslednjem poglavju bom opisal orodja, ki sem jih uporabil, ter kako deluje sama projekcija 2D teksture na 3D model, nato pa bom opisal še kako sem nalogo implementiral kot samostojno aplikacijo, ter jo po tem integriral v že obstoječo aplikacijo.

Poglavje 2

Metode in orodja

2.1 Java in Eclipse

Za razvoj programskega dela diplomske naloge sem uporabil programski jezik Java[4] in razvojno okolje Eclipse[2], saj sem z obema že seznanjen.

Java

Java je objektno orientiran programski jezik, ki ga je razvilo podjetje Sun Microsystems leta 1995. Osnovan je na jezikoma C in C++, vendar pa je višjenivojski, saj pred uporabnikom skrije različne konstrukte, kot so naprimer kazalci in upravljanje s pomnilnikom. Prevede se v nižjenivojski jezik, ki ga nato tolmači javanski navidezni stroj (angl. Java virtual machine - JVM). To omogoča princip *'napiši enkrat, izvajaj kjerkoli'*, kar je bil eden izmed ciljev razvoja tega programskega jezika.

Eclipse

Eclipse je odprtokodno integrirano razvojno okolje. V osnovi je namenjeno programiranju Java aplikacij, vendar obstajajo vtičniki tudi za mnoge druge jezike. Prednost uporabe Eclipsea je urejevalnik, ki omogoča samodejno dopolnjevanje izrazov in opozarja na napake, ter razhroščevalnik in prikaz do-

kumentacije.

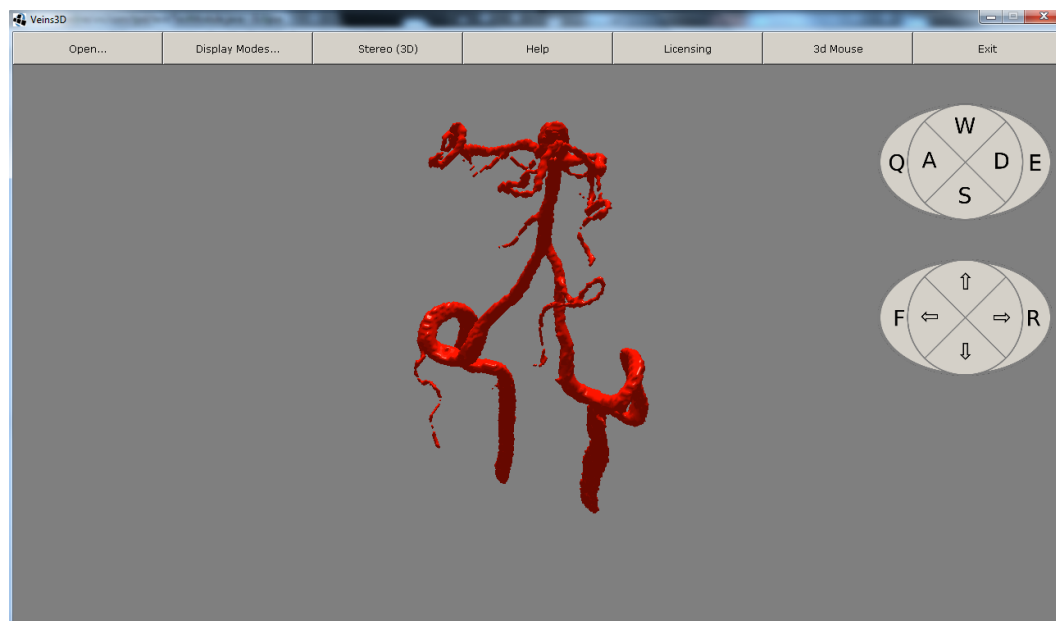
2.2 Hranjenje mrežnega modela

Za hranjenje mrežnega modela je uporabljena spremenjena različica standarda Obj datotek[6], ki ga je razvilo podjetje Wavefront technologies. Standard je odprt in ima široko podporo med orodji za vizualizacijo 3-dimenzionalnih (3D) mrežnih modelov. Standard je tekstoven, tako da so datoteke zlahka berljive, slabost tega pa je večja velikost datotek.

```
1 #Komentarji so vrstice , ki se zacnejo z znakom '#'
2 #Vrstice , ki se zacnejo z 'v' predstavljajo ogljiska
3 v -1 -1 0
4 v 1 -1 0
5 v -1 1 0
6 v 1 1 0
7 #Vrstice , ki se zacnejo z 'vt' predstavljajo teksturne
   koordinate
8 vt 0 0 0
9 vt 1 0 0
10 vt 0 1 0
11 vt 1 1 0
12 #Vrstice , ki se zacnejo z 'vn' predstavljajo normale (
    pravokotnice?)
13 vn 0 0 1
14 vn 0 0 1
15 vn 0 0 1
16 vn 0 0 1
17 #Vrstice ki se zacnejo s 'f' predstavljajo ploskve
18 f 1/1/1 2/2/2 4/4/4
19 f 1/1/1 4/4/4 3/3/3
```

Datoteka 2.1: Primer Obj datoteke, ki predstavlja kvadrat.

Različica podpira samo del standarda, in sicer zapis za položaje vozlišč ter definicijo ploskev, ne podpira pa naprimer teksturnih koordinat in normal, saj se v programu ne uporabljajo oziroma se izračunajo.



Slika 2.1: Glavni prikaz aplikacije za vizualizacijo.

2.3 Aplikacija za vizualizacijo

Aplikacija je namenjena prikazovanju ožilja pacienta. Model se da poljubno obračati in premikati, lahko pa se ga tudi ogleduje v stereo prikazu. Modeli žil so prebrani iz datotek tipa Obj.

Tukaj pride še o Neck_Veinsih nekoliko podrobneje, prav tako lahko kdo vire navedeš tako dosedanja diplomska dela kot tudi članke, ki smo jih s tega področja objavili. Članke najdeš na moji COBISS strani.

2.4 Strojno pospešen izris 3D grafike

Za strojno pospešen izris 3D grafike obstaja več aplikacijskih programerskih vmesnikov, najbolj znana in razširjena pa sta DirectX [1] in OpenGL [7]. DirectX je razvil Microsoft leta 1995 in je namenjen za razvoj aplikacij na operacijskem sistemu Windows, OpenGL pa je leta 1992 razvil Silicon Graphics Inc., sedaj pa ga nadzira neprofitna skupnost Khronos Group, in je namenjen strojno pospešenemu izrisu na večih platformah in v večih jezikih.

V diplomski nalogi sem uporabil odprtokodno knjižnico LWJGL (angl. lightweight java game library), ki omogoča uporabo vmesnika OpenGL.

OpenGL doda višjenivojsko abstrakcijo grafične procesne enote. Obstaja več različnih

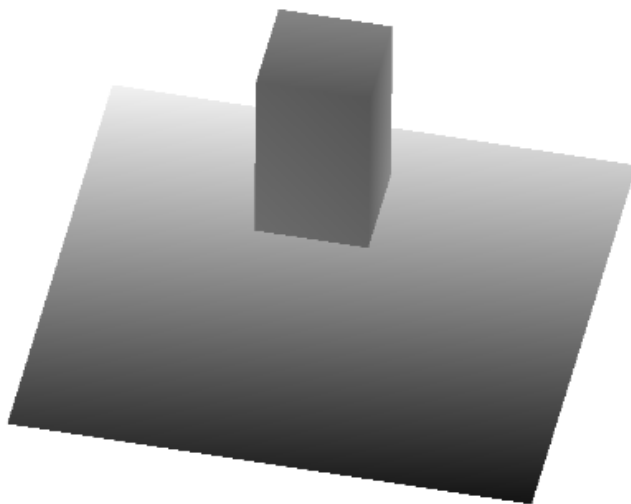
tipov grafičnih procesnih enot z različnimi zmogljivostmi, ki pa jih zaradi abstrakcije lahko programiramo z enotnim programskim vmesnikom.

2.5 Projekcija teksture na 3D model

Projekcija teksture na objekt deluje v treh korakih. Najprej je potrebno izrisati globinsko sliko iz pogleda projektorja [Kaj je projektor - to razloži], nato ugotoviti kaj na modelu dejansko zadanejo žarki projektorja, potem pa še pravilno pobarvati dele, ki jih žarki zadanejo.

Globinska slika

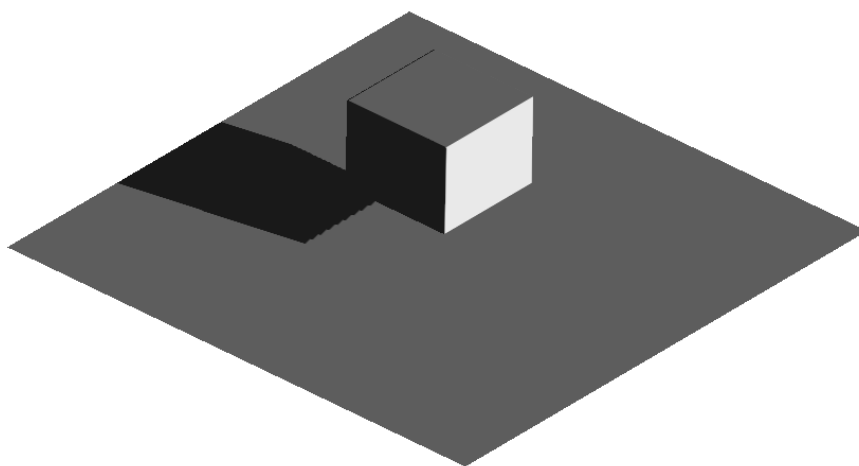
Luč se v računalniški grafiki obnaša enako kot kamera. Objekte se, tako kot pri kameri, projicira v pogled luči, nato pa se objekte izriše, pri tem pa se beleži le razdalja do luči oziroma globina najbližjih objektov v vsaki točki. Slika 2.2 je primer izrisa globinske slike.



Slika 2.2: Primer globinske slike. Svetlejši kot je piksel, dlje od kamere je objekt.

Izris objekta s sencami

Za izris objekta s sencami, se objekt najprej projicira v pogled kamere in pogled luči. Za vsak delec objekta v pogledu kamere se nato v globinski sliki preveri, če je ta delec viden tudi v pogledu luči. Če je delec viden, se ga osvetli z barvo luči, drugače pa je črn. Slika 2.3 je primer tako osenčenega modela.



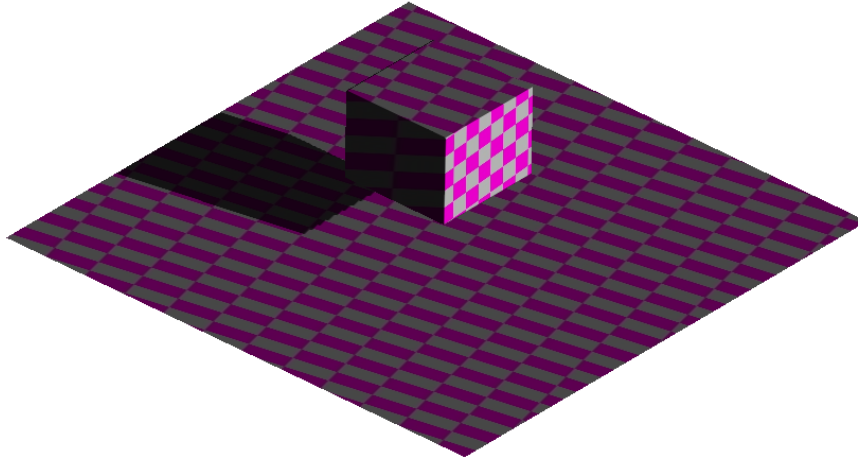
Slika 2.3: Primer osenčenega modela. Luč je na desni strani pred modelom.

Dobro bi bilo dodati lokacijo luči, da se bolje razumejo stvari. Deli geometrije, ki jih luč ne obsije so črni.

Izris objekta s projicirano sliko

Za izris s projekcijo moramo, če je delec viden, izračunati kje v sliki se ta delec nahaja, nato pa namesto bele svetlobe vzeti barvo svetlobe, ki je na tem mestu v sliki. Slika 2.4 je primer izrisa s projicirano sliko, kjer je projicirana slika vijolično-siva šahovnica.

Tukaj dodaj še opis podatkov, ki jih bomo uporabljali. Predstaviš da so podatki v OBJ pretvorjeni iz volumetričnih zajetih podatkov kot tudi kakšen je format v katerem so x-ray slike.



Slika 2.4: Primer projekcije texture na model.

2.6 Nabor podatkov

3D posnetki so volumetrični posnetki ožilja možganov, narejeni z magnetno resonanco, ki so nato s pomočjo funkcionalnosti že obstoječe aplikacije pretvorjeni v mrežni model, 2D posnetki pa so rentgenski posnetki ožilja shranjeni kot slikovna datoteka tipa jpeg [5].

Needs refactoring.

Poglavje 3

Implementacija

3.1 Programska ovojnica za LWJGL

Za lažje delo z OpenGL sem napisal programsko ovojnico, ki temelji na knjižnici LWJGL. OpenGL zahteva veliko knjigovodstva **Ni ravno ustrezen izraz. To opiši.**, saj je potrebno vsa sredstva, kot so medpomnilniki, senčilniki, texture in več, ročno uničiti, da se pomnilnik na grafični enoti ne zasiči. Ovojnica to poenostavi tako, da OpenGL sredstva zavije v smiselno poimenovane razrede. Ovojnica nudi tudi transformacije med kvaternioni in evklidskimi koti in razrede za transformacije in kamere.

Medpomnilniki

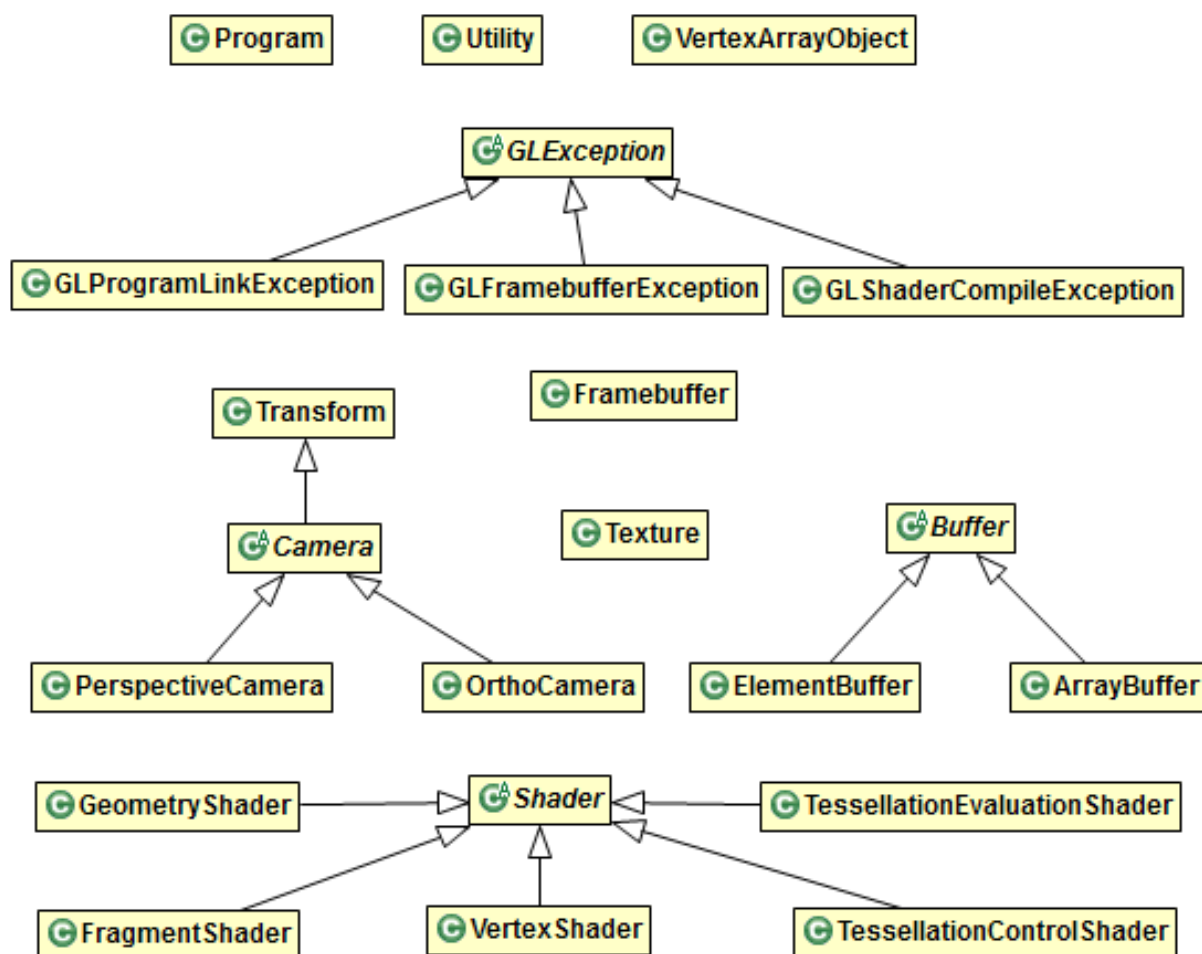
Prenašanje podatkov na grafično enoto in z nje v OpenGL poteka prek medpomnilnikov **Povej kateri vsi medpomnilniki obstajajo in za kaj se uporabljajo. Uporabil sem naslednje OpenGL medpomnilnike:**

Medpomnilnik za sezname hrani podatke potrebne za izris kot so položaji oglišč, pravokotnice v ogliščih, teksturne koordinate in tudi poljubne podatke, ki jih definira uporabnik.

Medpomnilnik za seznam elementov grafični enoti pove kako se sestavljajo osnovni liki.

Medpomnilnik za okvirje je drugačen tip medpomnilnika, saj sam po sebi ne hrani podatkov, vendar določa kam se bodo zapisali podatki iz cevovoda za izris.

Podprti pa so še naslednji tipi medpomnilnikov, ki pa so zunaj obsega predstavljenega dela: za kopiranje, za pakiranje pikslov, za poizvedbe, za texture, za povratno informacijo



Slika 3.1: Diagram dedovanja razredov v programski ovojnici.

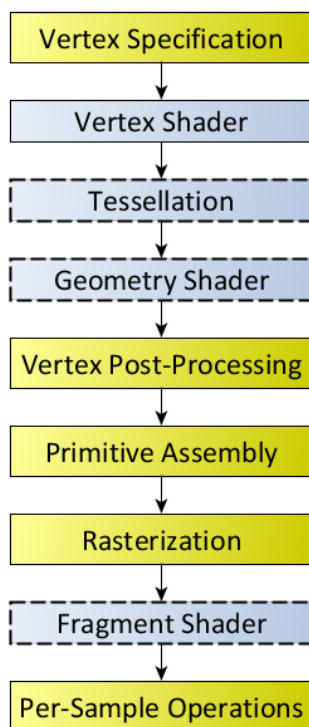
o transformacijah, za enotne spremenljivke, za posreden izris, za atomarne števec, za posredno odpremo ukazov in za hranjenje senčilnikov.

Ovojnica podpira medpomnilnike za sezname, ki se uporabljajo za prenašanje podatkov kot so položaji oglišč, normale v ogliščih in teksturne koordinate, medpomnilnike za elemente, ki grafični enoti povedo iz katerih oglišč so sestavljene ploskve, ter medpomnilnike za okvirje (angl. frame), ki hranijo slike za izris.

Senčilniki

Senčilniki so podprogrami za izris, ki jih mora napisati uporabnik. V starejših različicah se je uporabljal fiksni cevovod za izris, v novejših verzijah pa so nekateri deli programabilni.

OpenGL cevovod je sestavljen iz naslednjih faz, prikazanih v sliki 3.2:



[9]

Slika 3.2: Skica cevovoda za izris. Rumeni pravokotniki so fiksni deli, modri pa so v novejših različicah programabilni.

Specifikacija oglišč je faza cevovoda, v kateri uporabnik poda seznam oglišč in kako se združujejo v osnovne like kot so točke, črte, trikotniki in sezname trikotnikov.

Senčilnik za oglišča so uporabniški podprogrami, ki izvajajo poljubne operacije nad posameznimi oglišči. Uporabnik lahko določa izhodne spremenljivke, vendar pa mora med njimi nujno biti položaj. Ta faza cevovoda je obvezna.

Senčilnik za deljenje ploskev je neobvezen del cevovoda. Uporabnik definira dva senčilnika, senčilnik za nadzor deljenja in senčilnik za izračun deljenja. Prvi lahko deluje samostojno, drugi pa nujno potrebuje še prvega. V senčilniku za nadzor se izračuna kako podrobno se bodo liki delili, v senčilniku za izračun pa se izračunajo novi liki.

Senčilnik za geometrijo so prav tako neobvezni, definira pa jih uporabnik. Kot vhodne podatke dobijo osnovne like, ki pa jih lahko zavržejo, spremenijo in delijo.

Naknadna obdelava oglišč je faza, v kateri se liki iz prejšnjih faz zavržejo, če ne spadajo v pogled kamere, oziramo razrežejo, če so delno v pogledu kamere. Po želji se lahko podatki tu shranijo v medpomnilnik, da jih ni potrebno ponovno računati, če se ne bodo spreminjali.

Sestavljanje osnovnih likov je faza, v kateri se podatki iz prejšnjih faz pretvorijo v osnovne like, ki smo jih hoteli izrisati. Naprimer, če so vhodni podatki trikotniki, želimo pa le izris črt, se bo tu vsak trikotnik razdelil v tri črte. Tu se lahko cevovod konča, če želimo le izračunati podatke za pozneje, ne pa jih tudi izrisati.

Rasterizacija je faza, kjer se osnovni liki razdelijo v diskretne delce, ki služijo izračunu končne barve piksla.

Senčilnik za drobce izračuna barvo piksla in njegovo globina. Ta del cevovoda je neobvezen, vendar se, če ga ni, izračuna le globina delca.

Procesiranje drobcev je faza kjer se, če uporabnik tako nastavi, določi kateri delci, izmed tistih na istem položaju, se bodo obdržali glede na globino, šablono in druge lastnosti. Na koncu se barve prosojnih delcev zmešajo, nato pa se rezultat zapiše na zaslon ali v medpomnilnik za okvirje.

Poleg teh senčilnikov OpenGL podpira še senčilnike za izračun, ki pa se ne izvajajo v cevovodu in so namenjeni izračunu poljubnih podatkov.

Ovojnica podpira pet vrst senčilnikov, in sicer senčilnike za oglišča (angl. vertex shader), za geometrijo (angl. geometry), za drobce (angl. fragment) in za deljenje (angl. tessellation), obstaja pa še senčilnik za izračunavanje (angl. compute shader), ki pa ni podprt. Senčilniki se združujejo v programe, ki tvorijo programabilen del cevovoda za izris.

OpenGL senčilniki se pišejo v jeziku GLSL[3] (angl. OpenGL Shading Language). GLSL je osnovan na jeziku C in omogoča prenosljivost med operacijskimi sistemi in različnimi proizvajalci strojne opreme.

Teksture

Teksture v OpenGL so objekti, ki vsebujejo eno ali več slik, ter so uporabni kot vir slikovnih podatkov v senčilnikih ali kot tarča za izris v medpomnilnikih za okvirje. Ker je inicializacija teksture v OpenGL precej kompleksna, ovojnica poskrbi za inicializacijo osnovnih parametrov.

OpenGL podpira naslednje tipe tekstur:

1D, 2D in 3D teksture so teksture, kjer so slike shranjene v 1, 2 ali 3 dimenzionalnih poljih.

Pravokotna teksture so teksture, ki so nujno dvodimenzionalne in imajo drugačne metode za dostop do podatkov.

Kockaste teksture so teksture, sestavljene iz šestih enako velikih kvadratnih tekstur.

Večvzorčne teksture so teksture, ki za izračun vsakega piksla izračunajo več vzorcev, kar omogoča lepši izris.

Seznami tekstur so teksture, ki vsebujejo sezname slik. Podprte so 1D, 2D, kockaste in večvzorčne teksture.

Ovojnica podpira le 2D teksture, ostale pa v aplikaciji niso uporabljene.

Točkovni nizi

Točkovni nizi (angl. vertex array object) so OpenGL objekti, ki hranijo stanje potrebno za izris objektov. Hranijo le stanje, ne pa tudi samih podatkov, to je reference na medpomnilnike in obliko podatkov v teh medpomnilnikih, ne pa tudi samih podatkov iz medpomnilnikov. Prednost se kaže v hitrosti in velikosti kode, saj je potrebno veliko manj sprememb stanja.

Transformacija

Mogoče bi bilo smiselno kaj več o samih transformacijah napisati v predhodnjem poglavju.

Transformacija (angl. transform) je razred, ki je namenjen hranjenju transformacije objekta, to je rotacija, položaj in velikost. Položaj in velikost sta predstavljena kot vektorja dolžine 3, rotacija pa kot kvaternion. Podprte so rotacije in premiki v lokalnem ter globalnem koordinatnem sistemu, ter rotacije v poljubnem sistemu. Razred vsebuje tudi funkcije za pretvorbo iz kvaternionov v eulerjeve kote, za izračun transformacijske matrike iz stanja transformacije, ter za izračun kvaterniona iz rotacijske matrike.

(Formule za pretvorbe?)

Kot posebna transformacija se obravnava tudi kamera, le da so pri kameri premiki ravno obrnjeni, ima pa še dodatne parametre za pogled kamere. Podprti sta dve vrsti kamere: perspektivna in vzporedna.

Izjeme

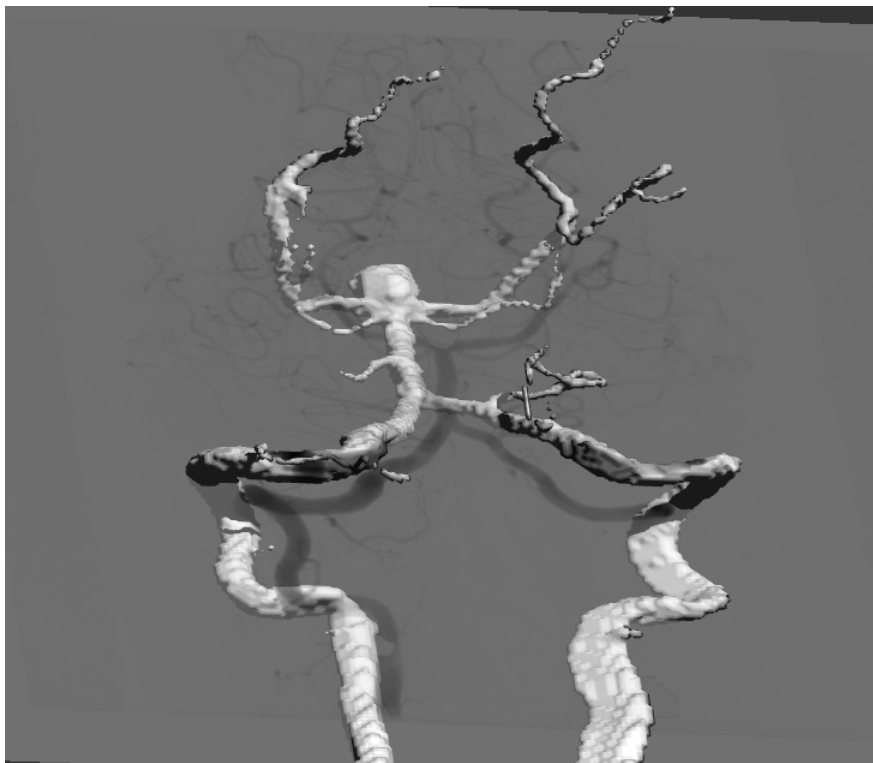
Ovojnica podpira več vrst izjem. OpenGL sam ne ustavi izvajanja programa ob napaki, vendar pa je izris nepravilen, zato ob nekaterih operacijah ovojnica preverja, če je na grafični enoti vse v redu. Izjeme se lahko prožijo, ko se senčilnik ne prevede pravilno, ko se program ne poveže pravilno ali ko se medpomnilnik za okvirje ne inicializira pravilno.

Ostalo

Poleg že naštetega ovojnica vsebuje še funkcijo za razhroščevanje, ki napake, ki jih vrača OpenGL, pretvori v berljive nize ter izpiše vrstico v kateri je bila funkcija klicana, kar močno olajša iskanje napak.

3.2 Samostojna aplikacija

Funkcionalnost orodja sem najprej implementiral kot samostojno aplikacijo, saj je tako lažje preveriti pravilnost delovanja. Aplikacija nima uporabniškega vmesnika, implementiran pa je le osnovni uporabniški nadzor, saj služi le kot primer delovanja, ne pa tudi dejanski uporabi. Sestavljena je iz testnega razreda, razreda ki implementira dejansko funkcionalnost in razredov za branje Obj datotek, ki pa niso uporabljeni v končni aplikaciji, saj ima ta svoj bralnik. Posnetki se v samostojni aplikaciji ob zagonu naložijo avtomatsko iz testnih datotek.



Slika 3.3: Pogled samostojne aplikacije.

Premikanje pogleda

Aplikacija podpira dva pogleda, pogled iz kamere ter pogled iz projektorja. Pogled iz kamere je glavni pogled, v katerem naj bi si uporabnik ogledoval model, na katerega je projicirana tekstura. V tem pogledu je mogoče s kamero poljubno premikati in krožiti okrog modela in projektorja, medtem ko sta ta pri miru. Pogled iz projektorja je namenjen poravnavi projicirane teksture z modelom. V tem pogledu se prizorišče izrisuje iz iste smeri kot sveti projektor, s premikanjem le-tega pa uporabnik poravnava 2D in 3D posnetka.

2D posnetek

Dvo dimenzionalni posnetek je rentgenska slika ožilja. Slika se projecira na pravokotnik, ki je prilagojen velikosti slike. Slika se da v prostoru rotirati neodvisno od modela, kar služi poravnavi, ko pa sta model in slika poravnana, se lahko rotira skupaj z modelom.

3D posnetek

Tri dimenzionalni posnetek je mrežni model ožilja, naložen iz Obj datoteke. 3D posnetek se sam po sebi ne premika, premika se le kamera, kar pa da uporabniku podoben vtis.

3.3 Integracija v aplikacijo za vizualizacijo

Za konec sem funkcionalnost integriral še v aplikacijo za vizualizacijo. Poleg funkcionalnosti samostojne aplikacije je bilo potrebno zagotoviti, da se modul obnaša čim bolj podobno že obstoječi aplikaciji, kar pa ni bilo čisto mogoče, saj modul uporablja drugačno vrsto kamere kot preostala aplikacija.

Premikanje pogleda

Model žil se premika enako kot v že obstoječi aplikaciji, s smernimi tipkami za premik levo, desno, naprej in nazaj ter s tipkama 'r' in 'f' za gor in dol. Obrača se s tipkami 'w', 'a', 's', 'd', 'q' in 'e' ter z miško, tako da se lev miškin gumb drži nad objektom in miško premika. Premikanje z miško je, podobno kot v že obstoječi aplikaciji, narejeno s projekcijo kazalca na navidezno kroglo okoli modela, vendar pa je zaradi drugačne vrste kamere izračun preseka drugačen. (*matematika za izračunom*)

Nalaganje datotek

Funkcionalnost za nalaganje datotek je obstajala že v osnovni aplikaciji. Uporabnik s pritiskom na gumb *'odpri'* dobi pojavno okno, v katerem izbira datoteke. Podprti formati so bili mhd za volumetrične datoteke, ter Obj za mrežne modele. Moral sem dodati še možnost za nalaganje slikovnih datotek v standardnih formatih jpeg in png [8]. Java podpira ta dva formata v svoji standardni knjižnici, tako da je bila implementacija preprosta.

Manjka ti še predstavitev osnovne aplikacije in predstavitev modulov, ki si jih adaptiral, uporabljal pri svojem delu.

Poglavje 4

Zaključek

V diplomskem delu sem implementiral projekcijo 2D rentgenske slike na 3D mrežni model. Vhodni podatki so mrežni modeli v datotekah tipa Obj in rentgenske slike tipa obj. Mrežni modeli so bili pretvorjeni iz MHD datotek v mrežne modele v okviru že obstoječe aplikacije Neck veins. Implementacija je bila napisana v programskem jeziku Java, kar je omogočilo brezhibno integracijo v že obstoječo aplikacijo. Za lažji in hitrejši razvoj sem razvil še programsko ovojnico za LWJGL, ki hrani stanje OpenGL objektov in ima bolj berljivo poimenovane funkcije.

V okviru nadaljnjega dela bi lahko implementiral algoritem za avtomatsko poravnavo 2D in 3D posnetkov[10], razvit na FRIju.

Literatura

- [1] Directx. <https://en.wikipedia.org/wiki/DirectX>. Dostopano 6.9.2015.
- [2] Eclipse. <https://eclipse.org/>. Dostopano 6.9.2015.
- [3] Glsl. <https://www.opengl.org/documentation/glsl/>. Dostopano 6.9.2015.
- [4] Java. https://en.wikipedia.org/wiki/Java_%28programming_language%29. Dostopano 6.9.2015.
- [5] Jpeg format specification. <http://www.w3.org/Graphics/JPEG/jfif3.pdf>. Dostopano 7.9.2015.
- [6] Object format specification. <http://www.martinreddy.net/gfx/3d/OBJ.spec>. Dostopano 7.9.2015.
- [7] Opengl. <https://www.opengl.org/about/>. Dostopano 6.9.2015.
- [8] Png format specification. <http://www.w3.org/TR/PNG/>. Dostopano 7.9.2015.
- [9] Rendering pipeline overview. <https://www.opengl.org/wiki/File:RenderingPipeline.png>. Dostopano 7.9.2015.
- [10] U. Mitrovic', Z. Spiclin, B. Likar, and F. Pernus. 3d-2d registration of cerebral angiograms: A method and evaluation on clinical images. *Medical Imaging, IEEE Transactions on*, 32(8):1550–1563, Aug 2013.