Kompjutorski vid u analizi medicinskih slika

Štirjan, Sara

SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Sara Štirjan

KOMPJUTORSKI VID U ANALIZI MEDICINSKIH SLIKA

Diplomski rad

Voditelj rada:
prof. dr. sc. Luka Grubišić

Zagreb, rujan, 2019.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Zahvaljujem se mentoru prof. dr. sc. Luki Grubišiću na razumijevanju, pomoći i savjetima pruženima prilikom pisanja ovog rada.
Zahvaljujem se dr. med. dent. Petri Bučević Sojčić na danim materijalima, odvojenom vremenu i savjetima.
Hvala mojim prijateljima koji su studentsko razdoblje učinili ljepšim.
Veliko hvala mojoj obitelji koja mi je najveća podrška u svemu i uvijek je tu za mene.
Najveće hvala mojem dečku koji me uvijek gura naprijed
i izvlači ono najbolje iz mene.

Sadržaj

Uvod

Istraživanja u području kompjutorskog vida, obrada slike i prepoznavanja uzoraka postigla su značajan napredak u posljednjih nekoliko desetljeća. Također, medicinska snimanja (CT, MR, ultrazvuk...) posljednjih godina privlače sve veću pažnju zbog svoje važnosti u medicini. Budući da su ljudi vizualna bića, 3D vizualizacija medicinskih slika može uvelike pomoći radiolozima i liječnicima u postavljanju dijagnoza, planiranju terapija, operacijama i daljnjem liječenju. Izrada i ispis 3D modela daje puno opipljiviji i prirodniji prikaz stanja od slika u 2D formatu. 3D ispis je već nekoliko godina prisutan u dentalnoj medicini, od ispisa krunica, kirurških šablona, splinteva do različitih pomagala. U budućnosti će, najvjerojatnije, kompjutori zamijeniti dentalne tehničare i bit će moguće isprintati zub u svega nekoliko minuta zbog čega je bitno razviti softver koji je jednostavan za korištenje i dovoljno precizan.

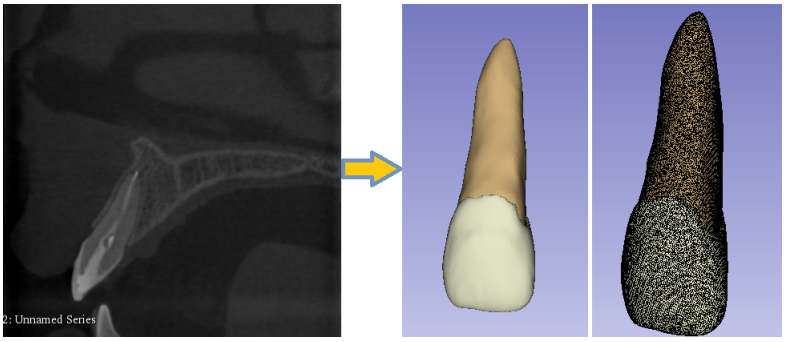
3D Slicer je primjer softvera koji se koristi za vizualizaciju i analizu medicinskih podataka. Njegova glavna prednost je što svima, uključujući i pacijente, daje uvid u njihovo stanje u 3D obliku. Prednost leži u grafičkom korisničkom sučelju koje korisniku dopušta brzo izoliranje regije interesa i vizualizaciju iste bez pisanja jednog reda koda, što aplikaciju čini pristupačnom korisnicima bez programerskog iskustva.

Tema ovog diplomskog rada je analiza sustava 3D Slicer koji je sustav otvorenog koda i koji iz snimke kompjutorske tomografije (CT) konstruira CAD model snimljenog objekta i sprema ga kao STL datoteku (Slika 0.1). Treba uspostaviti sustav te ga testirati na problemu rekonstrukcije ljudskog zuba iz CT snimke.

U prvom poglavlju ukratko su objašnjene metode dobivanja medicinskih slika. Objašnjen je princip rada CT uređaja i rekonstrukcija CT snimke te opisan DICOM format koji je najraširenij standard za pohranu medicinskih podataka.

U drugom poglavlju analiziran je sustav 3D Slicer, njegov razvoj i arhitektura sustava te su navedene biblioteke koje taj sustav koristi.

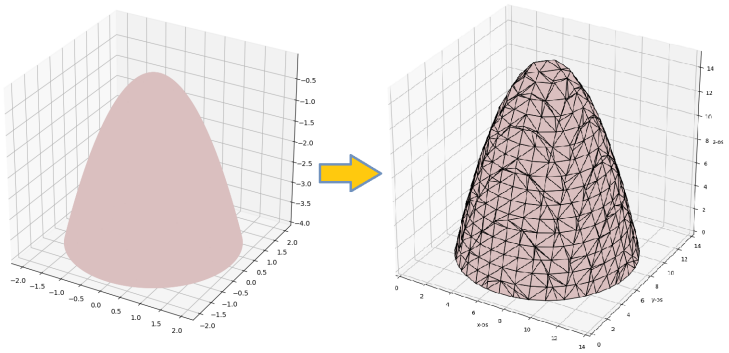
Treće poglavlje je vezano uz praktični zadatak ovog diplomskog rada. Na početku poglavlja je ukratko objašnjena građa zuba, anatomski dijelovi i struktura. Zatim je detaljno



Slika 0.1: Prikaz 3D modela dobivenog iz CT snimke zuba

objašnjen postupak dobivanja CAD modela iz CBCT snimke zuba u okruženju sustava 3D Slicer. U radu smo koristili CBCT snimku čeljusti i dobili CAD model lijevog centralnog sjekutića, točnije model cakline i dentina (Slika 0.1). Segmentacija slike je vrlo bitna na granicama tkiva kako bi segmentirani volumeni glatko prijanjali jedan uz drugi. Time se dobiva precizniji model bez praznina, te je ispravniji za simulacije jer se na dodirnu plohu stavljaju fizikalni uvjeti pod kojima se ostvaruje kontakt. Zbog važnosti glatkog prijanjanja dvaju modela, napravili smo model dodirne površine cakline i dentina. Dodirna površina je također važna u 3D printanju.

U četvrtom poglavlju je opisan *marching cubes* algoritam čije se varijacije koriste za dobivanje mreže trokuta (STL datoteke) iz trodimenzionalnog diskretnog skalarnog polja (voksela). Varijacija algoritma se koristi i u sustavu 3D Slicer. Na kraju poglavlja je primjer *marching cubes* algoritma u Pythonu primijenjen na paraboloid koji predstavlja idealizirani model zuba (Slika 0.2).



Slika 0.2: Primjena *marching cubes* algoritma na paraboloidu

Ključni pojmovi

Kako bi bolje razumijeli princip rada 3D Slicera i postupak dobivanja CAD modela, treba objasniti par pojmova koji se susreću u tom procesu.

Medicinske slike (Medical Imaging)

Medicinske slike kreiraju vizualnu reprezentaciju unutrašnjosti tijela u svrhu analize, dijagnostike i planiranja medicinskih tretmana. Postoji mnogo načina dobivanja medicinskih slika, počevši od klasičnog rendgena do visoko sofisticiranih metoda trodimenzionalne rekonstrukcije podataka koristeći rendgenske zrake (kompjutorska tomografija ili CT) ili magnetsku rezonancu (MR). Detaljnije opisano u poglavlju 1.

Obrada medicinskih slika (Medical Image Processing)

Obrada medicinskih slika uključuje vizualizaciju, analizu i preoblikovanje medicinskih slikovnih podataka, omogućujući im da budu značajniji za daljnju analizu. Obrada medicinske slike ovisi o vrsti medicinskog snimanja. Na primjer, određene anatomske strukture su bolje vidljive na MR nego na CT snimci ili kada su istaknute kontrastom. Slike niske rezolucije ograničavaju mogućnosti naknadne obrade i dobivanja korisne reprezentacije.

Registracija slike (Image Registration)

Registracija slike je proces prostornog (ili vremenskog) usuglašavanja dvije ili više slika scene. Registracija slike je korak u obradi i analizi slike u kojem se krajnja informacija dobiva kombinacijom podataka, kao što je to slučaj pri sjedinjavanju slika, detekciji promjena, restauracije slike iz projekcija i slično. U medicini se registracija slika, između ostalog, koristi pri kombiniranju podataka dobivenih kompjutorskom tomografijom u svrhu dobivanja 3D prikaza tkiva pacijenta.

Segmentacija slike (Image Segmentation)

Segmentacija slike se izvršava u sklopu registracije slike kako bi se istaknula obilježja za uparivanje. Termin segmentacija slike odnosi se na grupu postupaka za podjelu slike na segmente sa sličnim atributima (objekt i pozadina). Ovaj proces se ponavlja za svaki poprečni presjek kako bi se generirao konačan segmentirani volumen. Krajnji cilj segmentacije je pojednostaviti i/ili promijeniti reprezentaciju slike kako bi daljnja analiza bila jednostavnija i značajnija. U ovom radu je korištena metoda segmentacije pomoću praga (engl. threshold) kako bi se izdvojila željena tkiva za prikaz (dentin, caklina...).

Mapa oznaka (Label Map)

Mapa oznaka je reprezentacija područja odabranog tijekom segmentacije, bazirana na boji, pri čemu svaki voksel ima vrijednost koja označava vrstu tkiva na tom području. U 3D Sliceru, volumen mape oznaka (Label Map Volume) je čvor 3D skalarnog volumena gdje je svaki voksel broj koji označava vrstu tkiva na tom području. Volumen mape oznaka je povezan s čvorom "boja" (Color) koji preslikava brojeve u boje i tekstualne stringove.

Thresholding

Jednostavan i popularan pristup za segmentaciju i obradu slika. Thresholding algoritam sliku, koja je u nijansama sive boje, pretvara u binarnu sliku (crno-bijelu). Pikseli čija vrijednost prelazi prag τ dodjeljuju se jednoj, a ostatak drugoj grupi. Slijedno tome, slika se sastoji samo od objekta i pozadine.

Matematički, "Thresholding" možemo shvatiti kao filtriranje slike (funckije) podešavanjem vrijednosti praga. Slika je dvodimenzionalno polje (matrica) vrijednosti koeficijenta atenuacije $\psi : \{1, \ldots, n\}x\{1, \ldots, m\} \to \mathbb{R}$. Slika na koju je primijenjeno filtriranje s pragom τ je preslikavanje

$$\tilde{\psi} = \chi_{[0,\tau\rangle} \circ \psi,$$

gdje je $\chi_{[0,\tau\rangle} : \mathbb{R} \to \{0, 1\}$ karakteristična funkcija intervala $[0, \tau\rangle \subset \mathbb{R}$.

Threshold effect

U 3D Sliceru mape oznaka (Label Maps) se stvaraju opcijom Threshold effect. Korisnik odabire minimalnu i maksimalnu skalarnu vrijednost koje određuju interval čije se vrijednosti nalaze unutar segmenta. Za binarnu segmentaciju, vokseli koji su unutar skalarnog intervala predstavljeni su vrijednošću 1, dok su vokseli izvan skalarnog intervala predstavljeni vrijednošću 0.

CAD modeliranje

CAD modeliranje je trodimenzijalno modeliranje predmeta uz pomoć prostorno određenih dužina, krivulja, ravnina i ploha. Uz numerički vođene alatne strojeve ujedinjeno je projektiranje i proizvodnja u jedinstvenu koncepciju CAD/CAM (engl. Computer Aided Design/ Computer Aided Manufacture)

STL datoteka

STL format dobiva ime po postupku stereolitografije, a poznat je i kao Standard Triangulation Language. Izvorno, format je kreiran od strane 3D Systems-a za njihov CAD softver. STL datoteke opisuju geometriju površine trodimenzionalnog objekta bez prikaza boja, tekstura ili drugih uobičajenih atributa 3D modela [3]. Površina je prikazana u obliku mreže trokuta koja omeđuje CAD model i logički je razbijena u seriju malih trokuta koji imaju svoj smjer, orijentaciju i opisani su trima točkama u prostoru.

Poglavlje 1

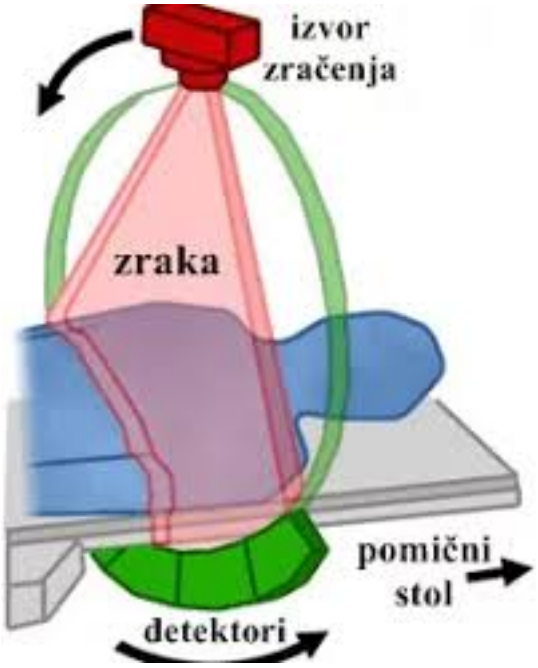
Medicinske slike

1.1 Kompjutorizirana tomografija

Kompjutorizirana tomografija (CT) radiološka je metoda koja koristi ionizirajuće-rendgensko zračenje za dobivanje slojevitog prikaza željenog dijela tijela [13]. CT snimka sastoji se od niza rendgenskih snimaka snimljenih iz različitih kutova i kombinira ih kako bi kreirala presjek ili sloj kostiju, krvnih žila i mekih tkiva unutar ljudskog tijela. Može koristiti za vizualizaciju gotovo svih dijelova tijela i koristi se za dijagnosticiranje bolesti ili ozljeda, kao i za planiranje medicinskih zahvata ili zračenja. CT snimka prikazuje unutarnje organe dvodimenzionalno u različitim nijansama sive boje. Koristeći suvremene CT uređaje (višedetektorske CT uređaje spiralne tehnologije) postoje velike mogućnosti rekonstrukcije slike npr. multiplanarne rekonstrukcije, 3D rekonstrukcije.

Procedura rada CT stroja kod skeniranja pacijenta:

1. Motorizirana podloga pomakne ležećeg pacijenta u unutrašnjost CT stroja.
2. Unutar kućišta stroja izvor rendgenskih zraka i detektor rotiraju se oko pacijenta. Jedna rotacija traje u prosjeku 1 sekundu. Rendgenski izvor proizvodi uski snop zraka u obliku lepeze koji prolazi kroz određeni dio tijela pacijenta (Slika 1.1).
3. Detektor je smješten na suprotnoj strani od rendgenskog izvora, te redovito proizvodi trenutačne snimke (engl. snapshots). Za vrijeme jedne rotacije izradi se veći broj snimaka, iz različitih kutova.
4. Nakon svake rotacije prikupljeni podaci šalju se u računalo koje zatim rekonstruira individualne snimke u jedan ili više poprečnih presjeka (engl. slice).



Slika 1.1: Prikaz lepezastog snopa rendgenskih zraka kod CT skeniranja[1]

Podjela CT uređaja s obzirom na napredak u tehnologiji:

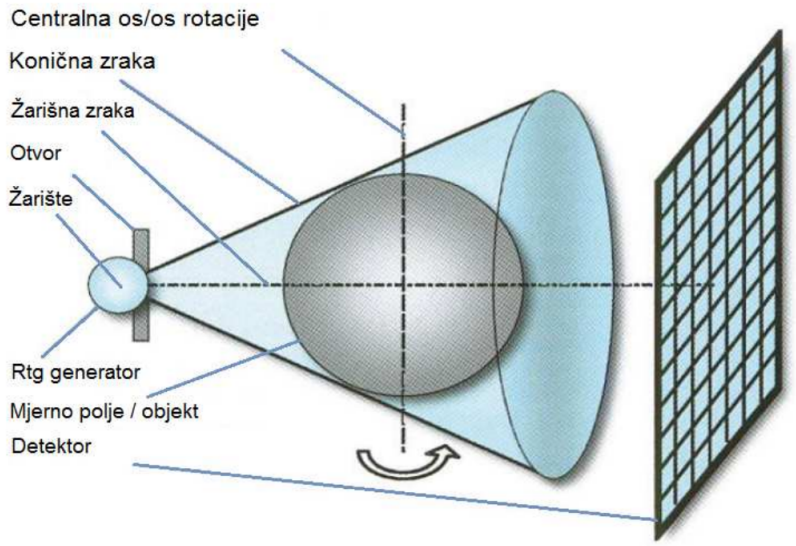
- jednoslojni CT uređaj (Single-Slice CT)
- višeslojni CT uređaj (Multi-Slice CT)
- CT s koničnim snopom (Cone Beam CT)

Single-Slice CT ima mogućnost snimanja samo u jednom sloju te se danas više ne koriste zbog visokih doza zračenja. Multi-Slice CT omogućuje simultan prikaz 4 sloja u rotacijskom vremenu od 0.5 sekundi, pružajući napredak u brzini skeniranja i longitudinalnoj rezoluciji te boljoj upotrebi dostupnih rendgenskih zraka.

Cone Beam CT

CBCT je baziran na koničnim zrakama, koje su usmjerene na usko područje interesa, te tako u odnosu na konvencionalni CT imaju bitno smanjenu efektivnu dozu zračenja, visoku razlučivost detalja, točne kvalitativne i kvantitativne vrijednosti, ekonomičnost i jednostavnost u korištenju. Omogućuje nam vizualizaciju snimaka po slojevima i presjek u sve tri dimenzije.

[1]Izvor slike 1.1: https://mateaandrijic.wixsite.com/doctorinthetardis/untitled-c16bp



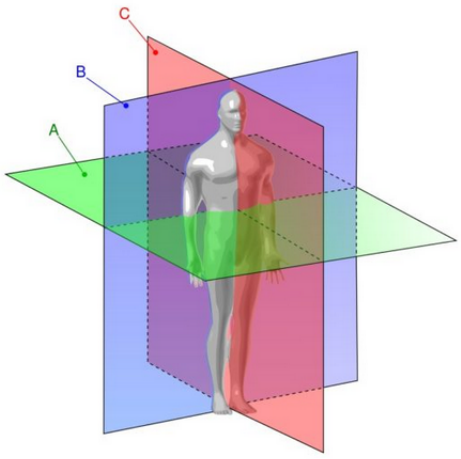
Slika 1.2: CBCT stvara koničan izvor ionizirajućeg zračenja[2]

Sa svojim divergentnim, odnosno koničnim izvorom ionizirajućeg zračenja, pokriva cijelu željenu regiju i dovoljna je jedna cirkularna rotacija, koja traje manje od 30 s, da se prikupe podaci za stvaranje trodimenzionalne slike (slika 1.2).

Rekonstrukcija slike na kompjuteru koja traje oko 2 minute, ovisi o više faktora kao što su širina polja, broj projekcija, veličini voksela koja daje slici dubinu, računalnom programu i samom računalu. CBCT uređaji su sposobni razlikovati najmanje 4096 nijansi sivila. Za svaku pojedinu rotaciju stvori se 100 – 700 individualnih projekcija koji sadrže više od milijun piksela [8].

Uz pomoć trodimenzionalnih prikaza moguća je točna vizualizacija struktura u njihovom stvarnom prostornom prikazu i u mjerilu 1:1, (DICOM format). Ovakva tehnologija omogućuje velik broj kombinacija slika, jer je od trodimenzionalne snimke moguće proizvesti panoramske, aksijalne, transverzalne, poprečne, kose i trodimenzionalne presjeke (Slika 1.3).

[2]Izvor slike 1.2: https://repozitorij.sfzg.unizg.hr/islandora/object/sfzg%3A177/datastream/PDF/view



Slika 1.3: Transverzalni (A), koronarni (B) i sagitalni (C) presjek[3]

Princip rada

Rendgenske zrake pri prolasku kroz snimani dio tijela oslabljuju (engl. attenuate), do čega dolazi zbog apsorpcije i rasapa zračenja. Slabljenje nije jednako u svim vrstama tkiva – ono ovisi o više faktora: atomskom broju, elektronskoj gustoći tkiva i energiji rendgenskih zraka.

Koeficijent apsorpcije označava do koje mjere atenuacije je došlo. Što je veći atomski broj i elektronska gustoća tkiva, koeficijent apsorpcije će biti veći. Nakon prolaska kroz tijelo pacijenta zrake nastavljaju put do detektora rendgenskog zračenja. U njemu se one pretvaraju u električne signale, čija je snaga proporcionalna atenuaciji, koji se šalju u računalo gdje će se izvršiti rekonstrukcija slike [13].

Rekonstrukcija CT snimke

Polje, dobiveno prekrivanjem rendgenskog snopa, prilikom izvođenja svake projekcije podijeljeno je na mrežu kvadrata. Najmanji kvadratić ovog polja se naziva **piksel** (engl. pixel – element slike) i predstavlja osnovnu jedinicu CT snimke. Za svaki piksel je izračunat koeficijent atenuacije. Najmanji dio objekta predstavlja piksel koji ima treću dimenziju u vidu debljine sloja – **voksel** (engl. voxel – volumni element slike). Vrijednost svakog voksela se dodjeljuje odgovarajućem pikselu polja i pohranjuje u memoriju računala. Prilikom rekonstrukcije CT snimke računalo iz memorije uzima podatke koeficijenata atenuacije

[3]Izvor slike 1.3: `http://www.fizibo.hr/2018/02/26/rotacija-trupa/`

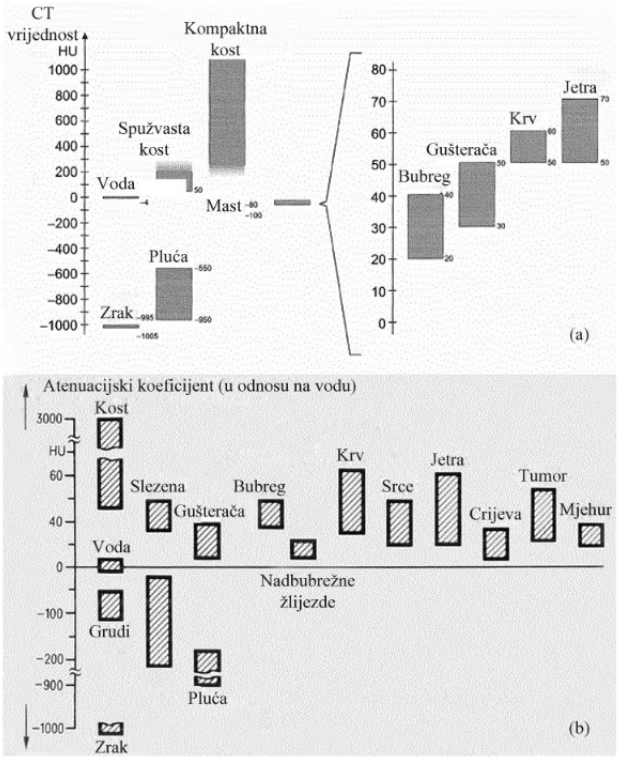
svakog piksela i od njih rekonstruira CT snimku. To znači da se svaka CT snimka sastoji od dvodimenzionalnog niza zapisanih vrijednosti koeficijenata atenuacije [9].

Za rekonstrukciju slike računalo iz memorije uzima podatke o koeficijentu atenuacije rendgenskih zraka za svaki piksel posebno i na osnovu njih rekonstruira CT snimku. Prilikom izračunavanja koeficijenta atenuacije u svakom pojedinom elementu slike, te vrijednosti se izražavaju preko CT broja, odnosno Hounsfieldove skale (relativne atenuacije u odnosu na vodu) prema formuli

$$H = \frac{\mu - \mu_{vode}}{\mu_{vode}} \cdot 1000, \tag{1.1}$$

gdje je μ_{vode} – koeficijent atenuacije za vodu, a μ izračunata vrijednost atenuacije.

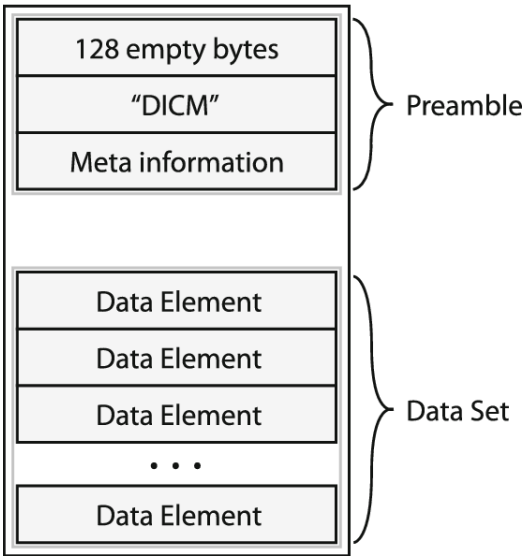
Kako je prikazano na slici 1.4 , vrijednosti atenuacije variraju od -1000 do +3000, dok je za Hounsfieldovu skalu raspon od -1000 do +1000 za pojedina tkiva [12].



Slika 1.4: a) Houndsfieldova skala, b) Vrijednosti atenuacije tkiva [9]

1.2 DICOM format

Najraširenији standard za pohranu podataka u medicinskoj vizualizaciji je DICOM (engl. digital imaging and communications in medicine). To je vrlo opsežni standard koji formalno definira većinu procedura u radu s digitalnim medicinskim podacima: rukovanje, skladištenje, ispis i prijenos podataka. DICOM pokriva definiciju formata podataka i protokol za mrežno komuniciranje – aplikacijski protokol baziran na protokolu TCP/IP. U DICOM formatu (Slika 1.5) sve informacije se spremaju grupirane u skupove podataka (engl. datasets), što znači da će se slikovni podaci nalaziti zajedno s metapodacima (npr. ime i ID pacijenta, datum, itd.). Time je osigurano da se glavni podaci nikad neće odvojiti od informacija koje ih opisuju.



Slika 1.5: DICOM format[4]

Podatkovni objekt u DICOM formatu sadržava razne atribute; većina njih su već spomenuti metapodaci, no jedan specijalni atribut predstavlja slikovne podatke, tj. vrijednosti piksela u medicinskoj snimci. Iako se metapodaci mogu smatrati kao zasebnom cjelinom koja je zapravo zaglavlje datoteke (engl. header), ne postoji nikakve praktične razlike između slikovnih podataka i informacijskih atributa. Jedna DICOM datoteka može sadržavati samo jedan atribut sa slikovnim vrijednostima. Za većinu modaliteta, to znači da će sadržavati jedinstvenu sliku. Ipak, postoji mogućnost zapisa višestrukih okvira (engl. frames) unutar tog atributa [5].

[4]Izvor slike 1.5: https://images.app.goo.gl/RpSs8mAsUbkrGVb36

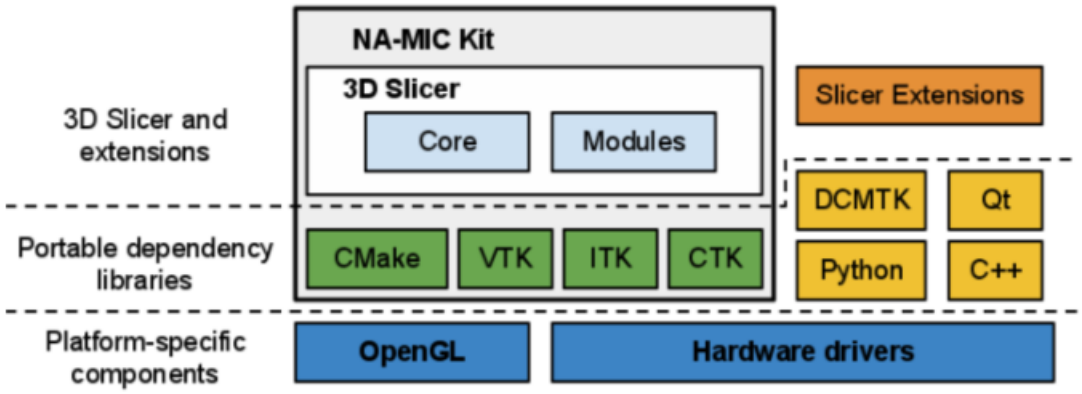
Poglavlje 2

3D Slicer

3D Slicer je besplatni program otvorenog koda koji se koristi za analizu i vizualizaciju medicinskih slika [2]. Kao klinički istraživački alat, 3D Slicer je sličan radiološkoj radnoj jedinici koja podržava razne vizualizacije, ali također nudi napredne funkcije poput automatizirane segmentacije i registracije za niz aplikacijskih domena. Za razliku od standardne radiološke radne jedinice, 3D Slicer je besplatan i nije vezan za određeni hardver. Kao programska platforma, 3D Slicer pojednostavljuje provođenje i procjenu novih kvantitativnih metoda omogućavajući znanstveniku da se usredotoči na implementaciju algoritma i pronalaženje ideja za komunikaciju između podataka, vizualizaciju i razvoj korisničkog sučelja. U usporedbi s drugim alatima koji pružaju aspekte ove funkcionalnosti, 3D Slicer je u potpunosti otvorenog koda i može se lako proširiti. Osim toga, 3D Slicer je dizajniran kako bi olakšao razvoj novih funkcionalnosti u obliku proširenja (engl. Extensions).

2.1 Razvoj 3D Slicer-a

Slicer je nastao kao vrhunac nekoliko neovisnih projekta koji su bili usredotočeni na vizualizaciju slike, kiruršku navigaciju i grafičko korisničko sučelje (GUI). David Gering predstavio je početni prototip Slicer-a u svom magistarskom radu, 1999. godine na MIT-u. Rad se temeljio na ranijim iskustvima istraživačkih grupa MIT-a i Laboratorija za kirurško planiranje (Surgical Planning Lab – SPL). Nakon toga, Steve Pieper preuzima ulogu glavnog arhitekta (engl. Chief Architect) i započinje s radom na pretvaranju 3D Slicer-a u industrijski paket. Od 1999. Slicer je u kontinuiranom razvoju na SPL-u pod vodstvom Rona Kikinsa. Danas Slicer razvijaju profesionalni inženjeri u bliskoj suradnji s programerima i znanstvenicima, uz sudjelovanje Isomics Inc., Kitware Inc. i GE Global Research-a, a značajno pridonosi i rastuća Slicer zajednica. Inicijalno zamišljen kao neurokirurški sustav usmjeravanja, vizualizacije i analize, tijekom posljednjeg desetljeća Slicer se razvio u integriranu platformu koja je primijenjena u raznim kliničkim i pretkliničkim istraživanjima,



Slika 2.1: Arhitektura 3D Slicer-a [4]

kao i za analizu nemedicinskih slika. Poboljšanje i održavanje softvera prvenstveno je moguće podrškom Nacionalnog zavoda za zdravstvo (National Institutes of Health – NIH). Također, njegov razvoj omogućio je i doprinos zajednice jer brojne grupe i pojedini korisnici, iako nisu direktno financirani za razvoj 3D Slicera, kontinuirano poboljšavaju softver prijavljivanjem problema, doprinosom rješenja, idejama za novim dodacima i razvijanjem novih alata.

2.2 Arhitektura 3D Slicer-a

Arhitektura 3D Slicera slijedi modularni i slojeviti pristup kao što je prikazano na slici 2.1. Na nižoj razini arhitekture nalaze se temeljne biblioteke koje daje operacijski sustav i ne dolaze u paketu s 3D Slicerom, primjer je OpenGL i hardverski upravljački programi koji omogućuju učinkovito korištenje prozora i grafičkih resursa računala. Na razini iznad nalaze se programski jezici (prvenstveno C++ i Python, ali sve više i JavaScript) i biblioteke koje pružaju višu razinu funkcionalnosti. Neke od biblioteka korištenih u aplikaciji su Qt[1] (za kreiranje grafičkog korisničkog sučelja), DCMTK[2] (implementira dijelove DICOM standarda i koristi se za interakciju s DICOM podacima) i jqPlot[3] (pruža mogućnosti s grafikonima).

Neke od biblioteka korištenih u 3D Sliceru dizajnirane su u bliskoj suradnji i često dijele istu zajednicu programera. Te se biblioteke distribuiraju kao dio NA-MIC Kit-a, ko-

[1] Qt, https://www.qt.io/
[2] DCMTK, https://dicom.offis.de/dcmtk.php.en/
[3] jqPlot, http://www.jqplot.com/

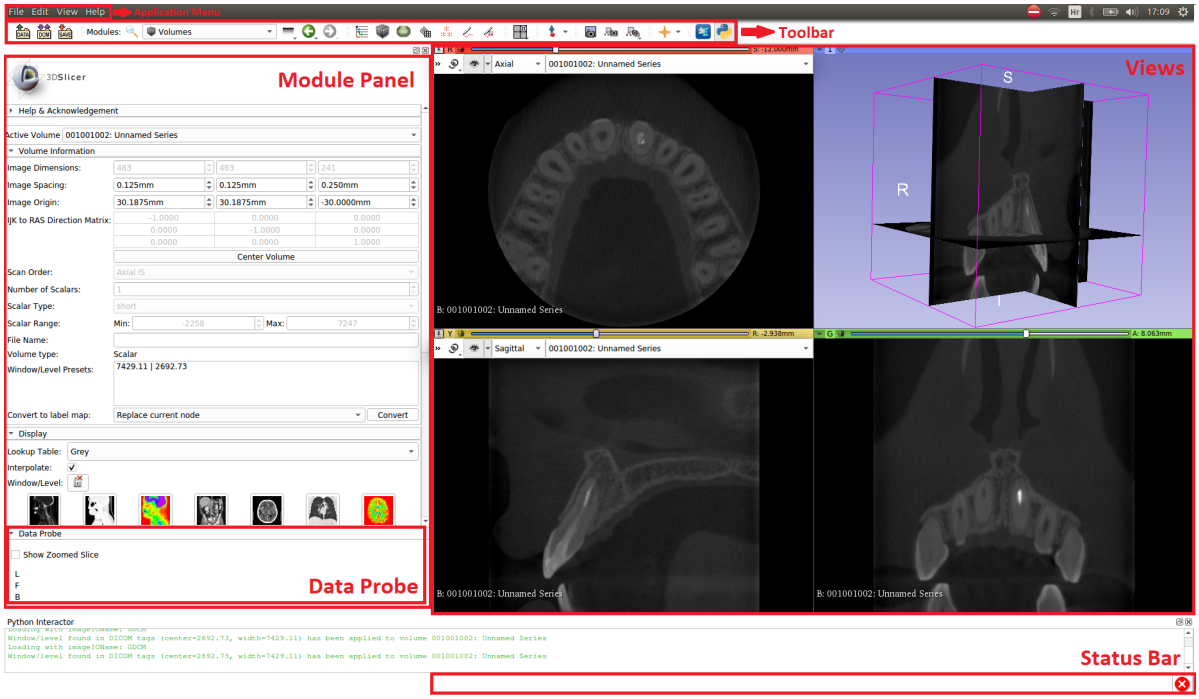
lekcije softverskih alata dijelom podržanih od strane National Alliance for Medical Image Computing (NA-MIC) projekta. CMake[4] omogućava među-platformsku (engl. cross-platform) izgradnju konfiguracije sustava, pakiranje i testiranje 3D Slicera i NA-MIC Kit biblioteka. CDash[5] je web-poslužitelj koji organizira rezultate testiranja softvera. Alat za vizualizaciju VTK[6](Visualization Toolkit) osigurava ključne sastavne dijelove za 3D grafiku računala i vizualizaciju. Alat za uočavanje ITK[7] (Insight Toolkit) je biblioteka razvijena isključivo za zadatke vezane uz registraciju i segmentaciju medicinskih slika te za implementaciju novih algoritama za analizu slika. CTK[8] (Common Toolkit) je biblioteka za obradu biomedicinskih slika s naglaskom na aplikacijskoj podršci DICOM formata.

Sam 3D Slicer sastoji se od aplikacijske jezgre, modula i proširenja. Jezgra (engl. core) implementira korisničko sučelje, pruža podršku za ulaz/izlaz podataka, vizualizaciju i programerska sučelja koja podržavaju proširenja aplikacije s novim priključcima (engl. plugins). Interno, Slicer koristi strukturu podataka ”scena” za organiziranje slika, koordiniranje okvira, napomena i održavanje stanja aplikacije. Jezik MRML (Medical Reality Markup Language) temelji se na XML-u i koristi za serijalizaciju sadržaja ”scene” [4]. Slicer moduli su priključci čija implementacija novih funkcionalnosti ovisi o jezgri Slicer-a. Pojedini moduli mogu biti neovisni ili mogu ovisiti o drugim modulima (npr. modul koji pruža funkcionalnost segmentacije lezije može ovisiti o modulu za vizualizaciju volumena - ”Volume Rendering” kako bi se omogućila 3D vizualizacija anatomije i segmentirane strukture). Za razliku od modula koji su dio Slicer-a, Slicer proširenja su vanjski priključci instalirani ”na zahtjev” korisnika, slično kao i proširenja web-preglednika.

2.3 Korisničko sučelje

Slicer sve učitane podatke pohranjuje u repozitorij podataka koji nazivamo ”scenom” (engl. scene). Svaki skup podataka, poput volumena slike (engl. image volume), površinskog modela (engl. surface model) ili skupa točaka, predstavljen je u sceni kao ”čvor” (engl. node). Slicer nudi veliki broj ”modula” (engl. modules), pri čemu svaki implementira određeni skup funkcija za stvaranje ili manipuliranje podataka u ”sceni”. Moduli obično ne djeluju izravno jedan na drugoga, već svi rade na istim podacima, koji su pohranjeni u ”sceni”. Slicer sadrži preko 100 ugrađenih modula, a dodatni moduli se mogu instalirati preko upravitelja proširenja (engl. extension manager).

[4]CMake, http://www.jqplot.com/
[5]CDash, https://www.cdash.org/
[6]VTK, https://vtk.org/
[7]ITK, https://itk.org/
[8]CTK, http://www.commontk.org/



Slika 2.2: Korisničko sučelje u programu 3D Slicer

Modul panel (engl. Module panel)

Modul panel nalazi se na lijevoj strani glavnog prozora i prikazuje sve mogućnosti koje trenutni modul nudi korisniku. Trenutni modul se može odabrati pomoću alatne trake za odabir modula.

Data probe se nalazi na dnu modul panela. Prikazuje informacije o prikazu sadržaja na položaju pokazivača miša.

Prikazi (engl. Views)

Slicer prikazuje podatke u različitim prikazima. Korisnik može birati između više unaprijed definiranih prikaza koji mogu sadržavati 3D prikaze, grafikone i tablice.

Alatna traka prikaza nudi padajući izbornik u kojem se može izabrati vrsta prikaza, ovisno o potrebi korisnika.

Izbornik programa (engl. Application Menu)

- Datoteka (File): funkcije za učitavanje prethodno spremljene scene ili pojedinačnih skupova podataka raznih vrsta te za preuzimanje primjera skupova podataka s interneta. Ovdje se nudi i opcija za spremanje scena i podataka. Dodavanje podataka (Add Data) omogućava učitavanje podataka iz datoteka. DICOM modul preporučuje se za dodavanje podataka iz DICOM datoteka i učitavanje već dodanih DICOM podataka. Klikom na "Save" otvara se prozor koji nudi razne mogućnosti za spremanje svih podataka ili odabranih skupova podataka.

- Uredi(Edit): Sadrži opciju za prikazivanje postavki aplikacije koja omogućuje korisnicima prilagođavanje izgleda i ponašanja Slicer-a, poput modula prikazanih u alatnoj traci, veličine fonta aplikacije, privremene lokacije direktorija, lokacije dodatnih modula koje treba uključiti.

- Prikaz (View) : Funkcije za prikazivanje/skrivanje dodatnih prozora i widgeta, poput Upravitelja proširenja za instaliranje proširenja iz trgovine aplikacija Slicer (Slicer App Store), Dnevnik grešaka (Error Log) za provjeru je li aplikacija naišla na potencijalne pogreške, Python Interactor za dobivanje Python konzole za interakciju s učitanim podacima ili modula, prikazivanje/skrivanje alatnih traka ili prebacivanje izgleda prikaza.

Poglavlje 3

Izrada CAD modela u 3D Slicer-u

U ovom poglavlju je opisan postupak izrade CAD modela zuba u programu 3D Slicer. Za potrebe diplomskog rada, koristili smo CBCT snimku gornje čeljusti spremljenu u DICOM formatu.

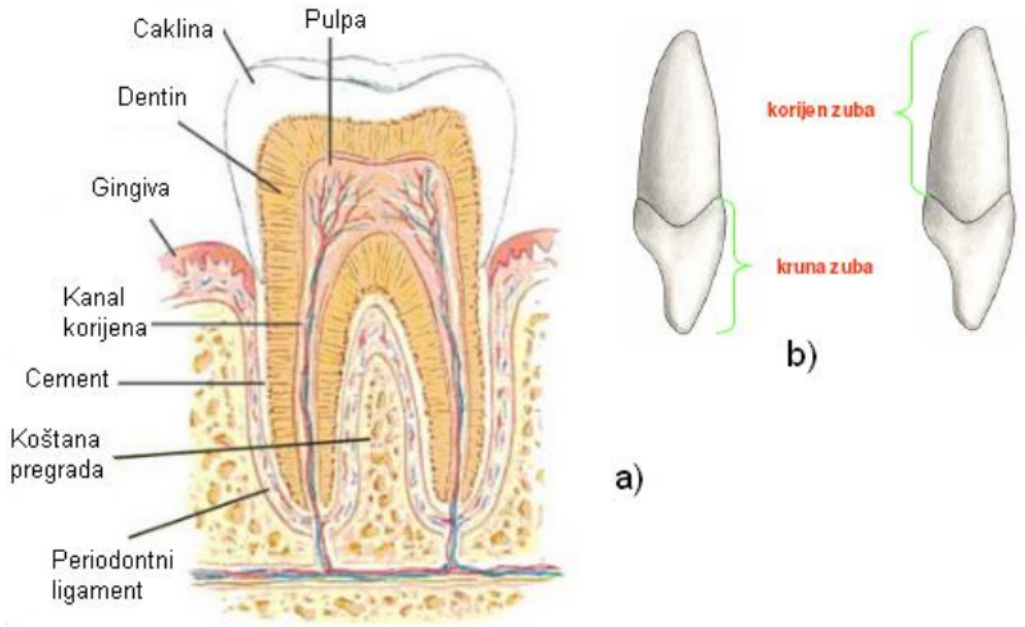
3.1 Građa zuba

Anatomski dijelovi zuba

Na svakon zubu se razlikuju tri osnovna dijela: kurna, vrat i korijen.
Kruna zuba se označava s dva termina: anatomska i klinička kruna. Anatomska kruna je dio zuba prekriven caklinom. Ona se kod intaktnog zubnog niza ne vidi cijela, jer joj je vratni dio prekriven desnima. Klinička kruna je vidljivi dio zuba i može, ali ne mora, odgovarati anatomskoj kruni. Ona se obično mijenja tijekom života uslijed resorpcije okolnog koštanog tkiva, povlačenja gingive i sl. Vrat zuba je suženje na granici cakline i cementa, u obliku plitkog žlijeba i u fiziološkim je uvjetima prekriven sluznicom usta. Korijen zuba također je označen sa dva termina, a služi učvršćivanju zuba. Anatomski korijen je dio zuba prekriven cementom i usađen je u zubnu jamicu (alveolu). Klinički korijen je onaj dio zuba koji se ne vidi pri pregledu zuba i može, ali ne mora, odgovarati anatomskom korijenu. Broj i veličina korijena varira od zuba do zuba.

Struktura zuba

Četiri osnovna tkiva tvore zub: caklina, dentin, cement i pulpa. Prva tri tkiva su tvrde građe, sastoje se od različitog udjela mineralnih tvari koje im daju čvrstoću. Caklina i dentin čine krunu zuba, dok cement i pulpa čine korijen zuba. Pulpa je specijalizirano tkivo i sastoji se od vezivnog tkiva, krvnih žila i živaca te ima prehrambenu, oblikovnu i zaštitnu ulogu.



Slika 3.1: Građa zuba; a) dijelovi zuba, b) osnovna raspodjela [7]

Periodontni ligament sa zubom čini anatomsku i funkcionalnu cjelinu. On veže zub za kosti. Gingiva okružuje zub i prekriva kost u koju su zubi umetnuti (slika 10).

Caklina

Caklina je najtvrđa te najviše mineralizirana tvar na tijelu, a ujedno čini jednu od četiri najbitnijih dijelova zuba. Caklina je vidljiva izvana (ona je uočljiva samim pogledom u usta), a „drži“ je dentin koji nije vidljiv i u potpunosti se nalazi ispod cakline. 96% cakline se sastoji od minerala, a ostatak čine voda i organske tvari. Normalna boja cakline varira od svijetlo žute do sivkasto bijele boje. Budući da je caklina prozračna, boja dentina i bilo kojeg restorativnog materijala ispod cakline bitno utječu na sam izgled zuba. Debljina cakline ovisi o dijelu površine zuba koji prekriva. Obično je deblja na vrhu, gdje njena debljina iznosi obično oko 2.5 mm, a najtanja je na rubovima što se klinički vidi kao spoj cementa i cakline.

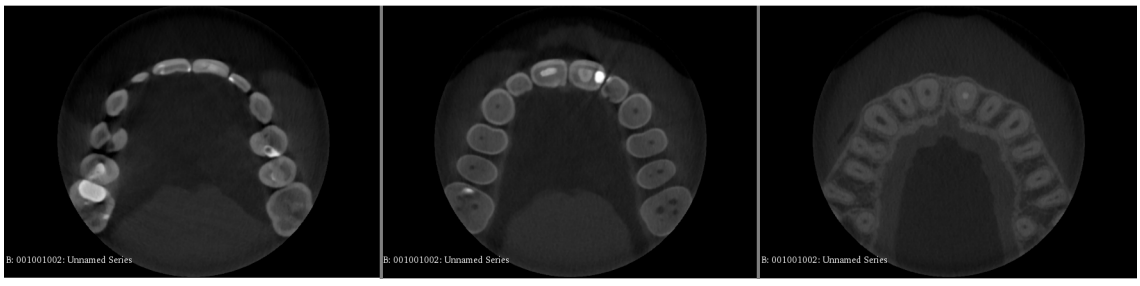
Dentin

Dentin ili zubna kost je čvrsto tkivo koje izgrađuje najveći dio zuba. Ne vidi se pri pregledu zuba jer je pokriven caklinom (u području krune) i cementom (u području korijena). Ima približan oblik zuba, a u središnjem dijelu formira zubnu šupljinu. Po sastavu je sličan kostima, te sadrži 28% organskih i 72% neorganskih materija. Sastoji se uglavnom od kristala hidroksiapatita, koji su ugrađeni u čvrstu mrežu kolagenih vlakana, a soli kalcija čine ga veoma otpornim na pritisak. Kroz dentin prolaze mnogobrojni Haversovi kanalići promjera $2 - 5\mu m$, kroz koje se protežu Tomasova vlakna (ogranci krvnih žila i živaca) i fibra-vlakna (periferni produžeci stanica odontoblasta koje grade dentin). Dentin se stvara kontinuirano tokom cijelog života (sekundarni i tercijarni dentin), a boja mu varira od žućkaste do bijelosive.

3.2 Učitavanje podataka u 3D Slicer

Pokretanjem 3D Slicer-a otvara se "Welcome to Slicer" modul u kojem su ponuđene opcije učitavanja podataka. Za učitavanje DICOM datoteka preporučeno je koristiti "DICOM" modul pa ćemo u padajućem izborniku izabrani modul "DICOM". Klikom na "Import" odaberemo direktorij u kojemu su spremljene CT snimke, odaberemo studiju koju želimo učitati te klikom na "Load" otvorimo podatke.

Nakon učitavanja podataka, u desnom dijelu prozora (Slika 2.2) su vidljiva tri dvodimenzionalna prikaza - transverzalni, sagitalni i koronarni koji su redom prikazani u crvenom, žutom i zelenom prozoru. Pomoću pokazivača miša se može lako prelaziti između pojedinih poprečnih presjeka (Slika 3.2), mijenjati svjetlina i kontrast (Slika 3.3).



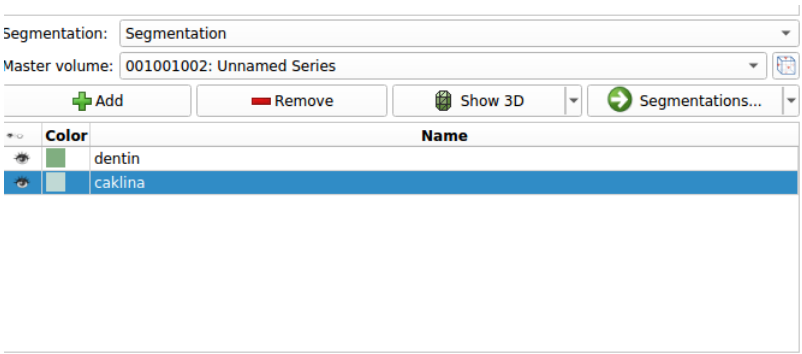
Slika 3.2: Tri različita poprečna presjeka u transverzalnoj ravnini

Slika 3.3: Prikaz promjene svjetline i kontrasta

U modulu ”Volumes”, u lijevom dijelu prozora mogu se vidjeti informacije o ”volumenu” (Slika 2.2). Vidimo da je dimenzija CT snimke korištene u ovom diplomskom radu $483 \times 483 \times 241$, a razmak između pojedinih poprečnih presjeka je $0.125mm$ u sagitalnoj i koronarnoj ravnini, te $0.250mm$ u transverzalnoj ravnini. U odjeljku ”Display>>Window/Level” možemo odabrati unaprijed postavljene postavke boje i kontrasta kako bi bolje naglasili područje interesa.

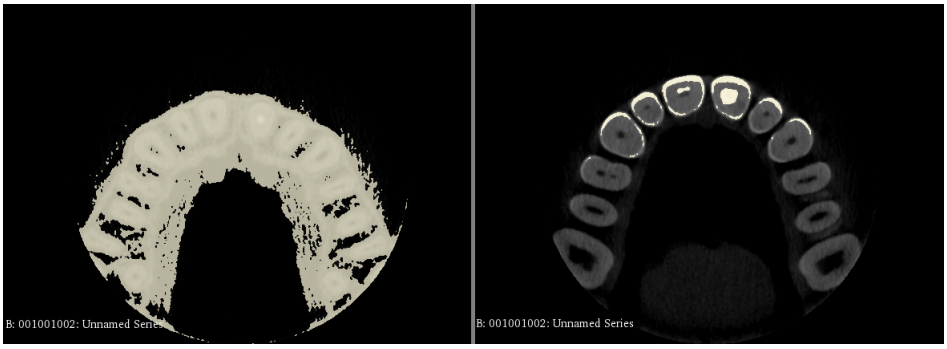
3.3 ”Segment Editor” modul

Segmentaciju ćemo započeti koristeći ”Segment Editor” modul. Cilj nam je dobiti CAD model u kojem ćemo odvojiti caklinu od dentina zuba i spremiti to kao dvije STL datoteke. Pomoću ”Add” gumba dodajemo novi segment. Prvi ćemo nazvati ”caklina” i odabrati boju koja će predstavljati segment (Slika 3.4).



Slika 3.4: Dodavanje novog segmenta

Sada koristimo ”Threshold effect” koji nam omogućava da stvorimo binarnu mapu oznaka (binary label map). Na dvodimenzionalnim prikazima se pojavljuje trepereća regija u

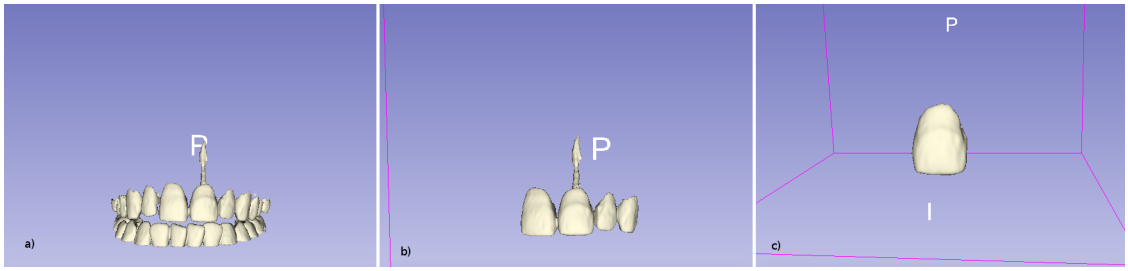


Slika 3.5: Trepereća regija koja se pojavljuje odabirom opcije ”Threshold effect” ovisno o odabranom intervalu

boji odabranog segmenta. Alat nam nudi opciju da odaberemo interval, minimalnu i maksimalnu vrijednost, unutar kojeg se nalaze vrijednosti voksela koji ulaze u odabrani segment. Mijenjajući ”Threshold” interval vidimo da se mijenja trepereća regija (Slika 3.5). Vrijednosti voksela koji su izvan intervala će poprimiti vrijednost 0, dok će vrijednosti voksela unutar intervala poprimiti vrijednost 1. Kod odabira intervala dolazi do kompromisa između hvatanja detalja i čistoće segmenta. Imamo mogućnost ručnog doradivanja segmenta, pa neke dijelove možemo ostaviti izvan intervala kako bi dobili što čišći model.

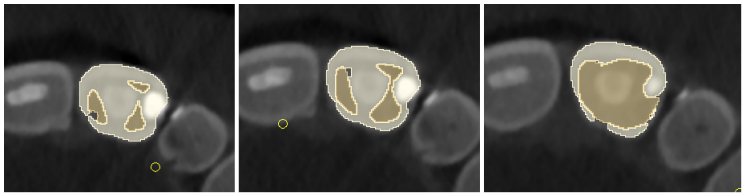
Kada smo potvrdili vrijednost intervala, kliknemo na gumb ”Show 3D” kako bi vidjeli odabrani segment (Slika 3.6a). Odabirom ”Show 3D” obojana regija, koju zovemo binarna mapa oznaka, pretvara se u ”closed surface” model. Klasični algoritam za ovu pretvorbu je *marching cubes* algoritam koji je opisan u poglavlju 4. Pretvorba se izvršava svaki puta kada napravimo promjenu na segmentu.

Pomoću dodatnih alata kao što su ”Islands”, ”Paint”, ”Erase” i ”Scissors” možemo prepraviti nepravilnosti u segmentu i odstraniti dijelove koji nam nisu potrebni za model (Slika 3.6).

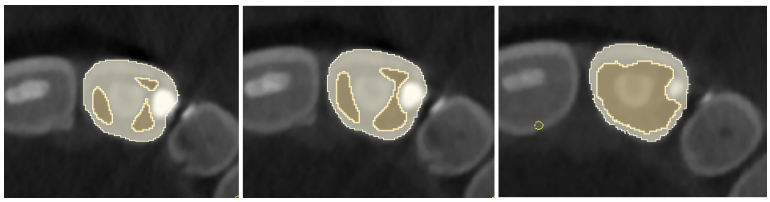


Slika 3.6: a) Rezultat ”Threshold effect-a”, b) Rezultat opcije ”Islands”, c) Rezultat opcije ”Scissors”

Kada caklina poprimi zadovoljavajući oblik, možemo prijeći na drugi segment kojeg ćemo nazvati ”dentin” Sada kod odabira intervala u alatu ”Threshold” odabiremo komplementarnu vrijednost kako bi kruna i korijen bili spojeni (Slika 3.9). Prilikom segmentacije svakog poprečnog presjeka, bitno je dobro segmentirati presjek zuba kako ne bi došlo do razmaka između dentina i cakline (vidi sliku 3.7 i 3.8).

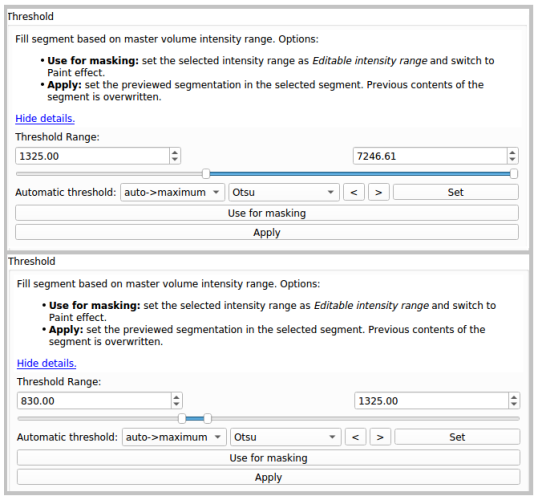


Slika 3.7: Primjer loše segmentirane plohe

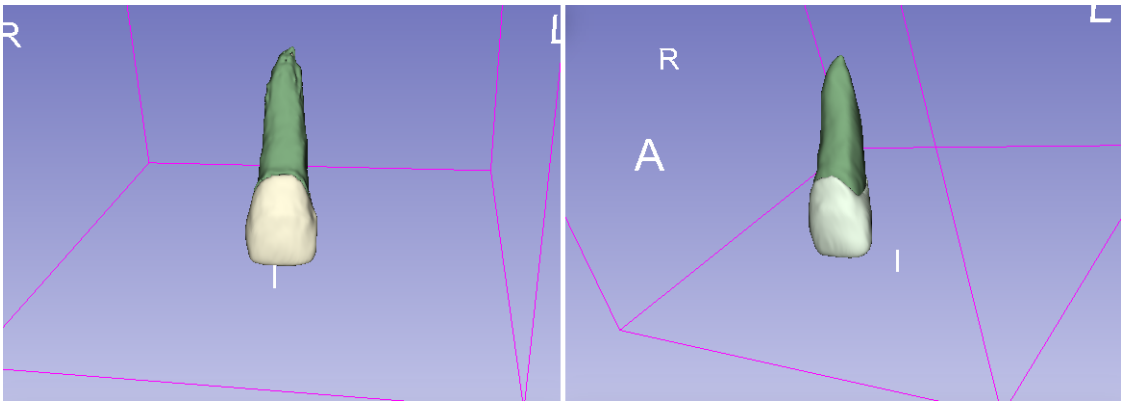


Slika 3.8: Primjer dobro segmentirane plohe

Nakon dodatnog uređivanja, kao što smo radili za prvi segment, i nakon što smo zadovoljni izgledom segmenta, preostale neravnine možemo zagladiti odabirom opcije ”Smoothing” (Slika 3.10).



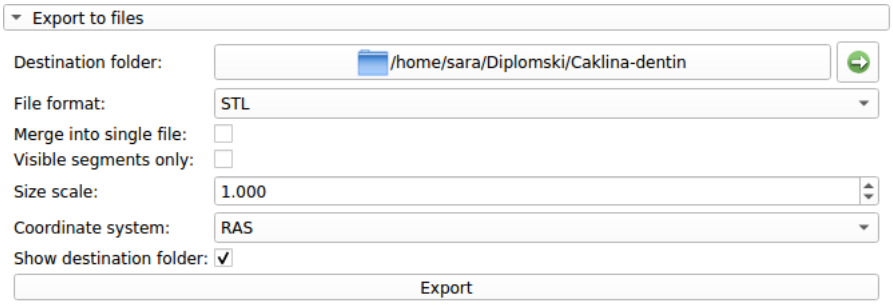
Slika 3.9: Odabir ”threshold” intervala za caklinu (gore) i dentin (dolje)



Slika 3.10: Rezultat opcije ”Smoothing”

3.4 Spremanje modela u STL formatu

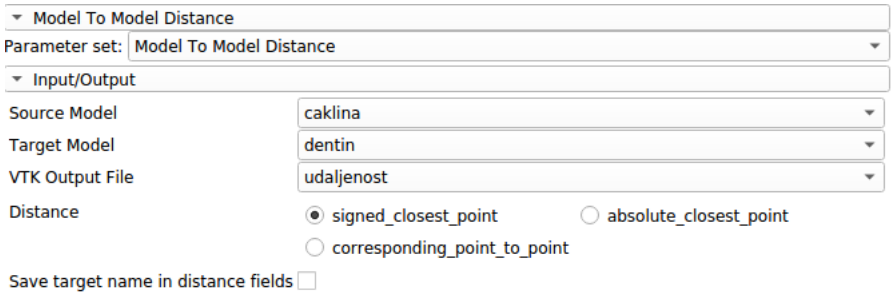
Segmentacija slika se često koristi za 3D ispis modela. 3D Slicer nudi trivijalnu i automatsku konverziju modela u STL format. Kada završimo s izradom modela, prelazimo na modul ”Segmentations” gdje pronalazimo prozor ”Export to File”, odaberemo STL format datoteke i direktorij u koji želimo spremiti datoteku te kliknemo na ”Export” (Slika 3.11). CAD model zuba se nalazi u odabranom direktoriju i spreman je za 3D ispis.



Slika 3.11: Spremanje modela u STL formatu

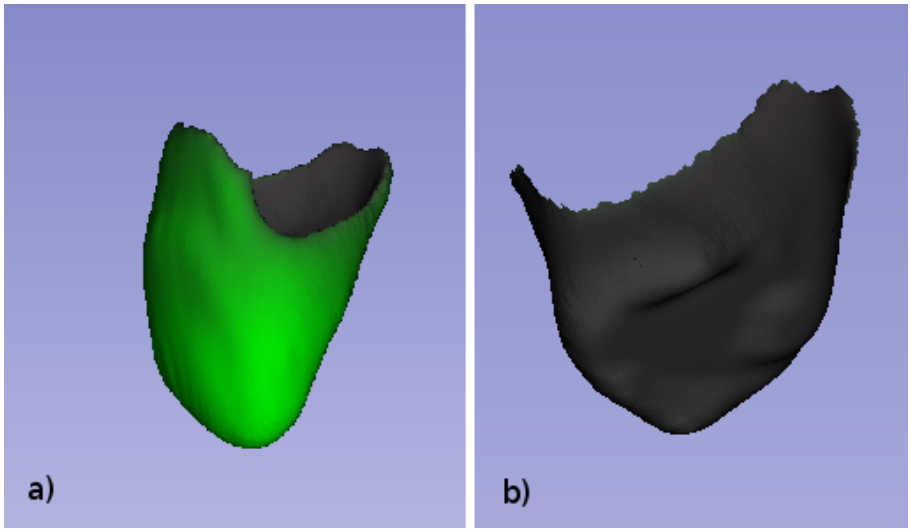
3.5 Dodirna površina modela

Kada smo napravili 3D model dentina i cakline, uz pomoć 3D Slicera možemo dobiti dodirnu površinu dvaju modela. To ćemo napraviti uz pomoć modula ”Model to model distance” kojeg možemo instalirati preko ”Upravitelja proširenja”. Ovaj modul izračunava udaljenost od točke do točke između dva modela učitana u Slicer. Temelji se na *vtkDistancePolyDataFilter*-u. Udaljenost može biti sa predznakom ili bez predznaka. Iako *vtkDistancePolyDataFilter* izračunava i udaljenost između točaka i udaljenost između ćelija, ovaj modul sprema samo udaljenost između točaka. Izlazni volumen ima isti broj točaka kao i prvi ulazni volumen (Source Model). Udaljenosti se spremaju kao niz točaka pod imenom ”Distance” [1].



Slika 3.12: Modul za računanje udaljenosti dvaju modela

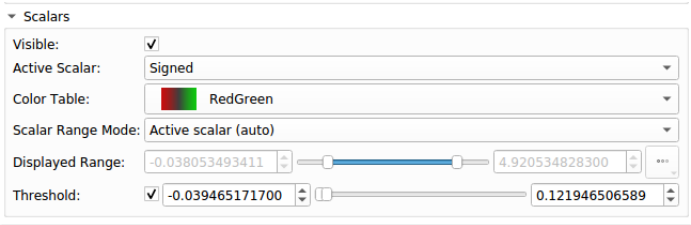
Kao ”Source Model” odaberemo caklinu, a kao ”Target Model” dentin. U ”VTK Output File” odaberemo novi model i spremimo ga kao ”udaljenost” (Slika 3.12). Odaberemo ”signed closest point”, algoritam koji računa Hausdorffovu[1] udaljenost između točaka cakline i dentina. Klikom na ”Apply” stvorimo novi model koji je prikazan na slici 3.13a.



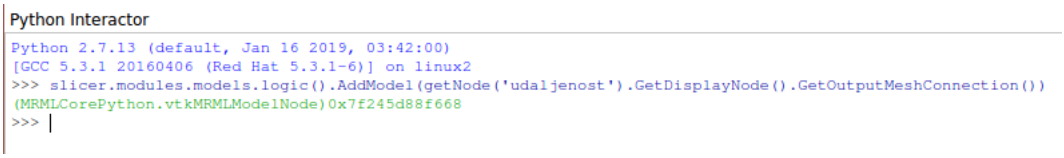
Slika 3.13: a) Model koji prikazuje udaljenost cakline od dentina, b) Dodirna ploha između cakline i dentina

Cilj je dobiti dodirnu površinu cakline i dentina pa ćemo model udaljenosti, pomoću smanjenja ”Threshold” intervala (Slika 3.14), ograničiti na dio između cakline i dentina i time dobiti vizualizaciju dodirne površine (Slika 3.13b). Kako bismo od toga dobili model, moramo u ”Python Interactor” upisati kod dan na slici 3.15.

[1] više na https://arxiv.org/abs/1812.06740

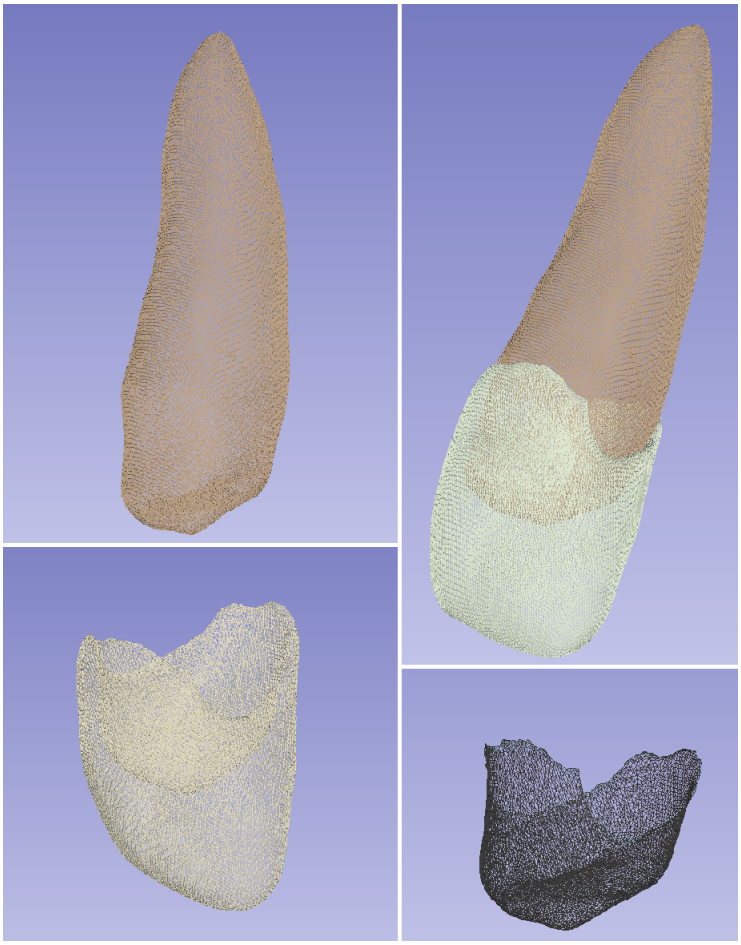


Slika 3.14: ”Threshold” interval koji trebamo smanjiti kako bi dobili dodirnu plohu



Slika 3.15: Kod u ”Python Interactoru” za kreiranje modela dodirne površine

Kao rezultat smo dobili STL datoteku koja predstavlja dodirnu površinu između cakline i dentina (Slika 3.13b). Za opcije prikaza možemo odabrati ”Wireframe” i vidjeti kako modeli izgledaju kao mreže trokuta (Slika 3.16).



Slika 3.16: Prikaz STL modela dentina, cakline, dodirne plohe i spojenog dentina i cakline

Dobivena dodirna ploha je bitna u 3D printanju jer 3D modeli moraju prijanjati upravo u tim točkama. Također je bitna za simulacije jer na tim dijelovima plohe stavljamo fizikalne uvjete pod kojima se ostvaruje kontakt. U program Gmsh[2] su učitane STL datoteke i napravljena je 3D mreža, označeni su rubovi i kontaktna površina, a rezultati su prikazani na kraju diplomskog rada, u dodatku A.

[2]http://gmsh.info/

Poglavlje 4

Kreiranje mreže trokuta

U ovom poglavlju opisati ćemo *marching cubes* algoritam koji kreira mrežu trokuta površina konstantne gustoće (izopovršina) iz trodimenzionalnog diskretnog skalarnog polja (voksela).

Mreža poligona se u računalnoj grafici definira kao linearna površinska aproksimacija objekta iz stvarnog svijeta. Sastoji se od vrhova (točaka), bridova koji spajaju te vrhove i poligona. Mreža trokuta je podskup mreže poligona u kojoj su svi poligoni trokuti.

4.1 *Marching cubes* algoritam

Najpopularnija metoda za rekonstrukciju površine je *marching cubes* algoritam (algoritam marširajućih kocaka), predstavljen 1987. godine od Lorensena i Clinea [11]. Osnovna ideja algoritma je da područje slike podijeli u regije - kvadrate u 2D, kocke u 3D - te, ovisno o konturi objekta, odredi je li regija homogena ili ne. Ukoliko je regija homogena, to je interijer ili eksterijer objekta. U slučaju kada nije homogena, uvodi se konturni element kako bi objekt odvojio od pozadine.

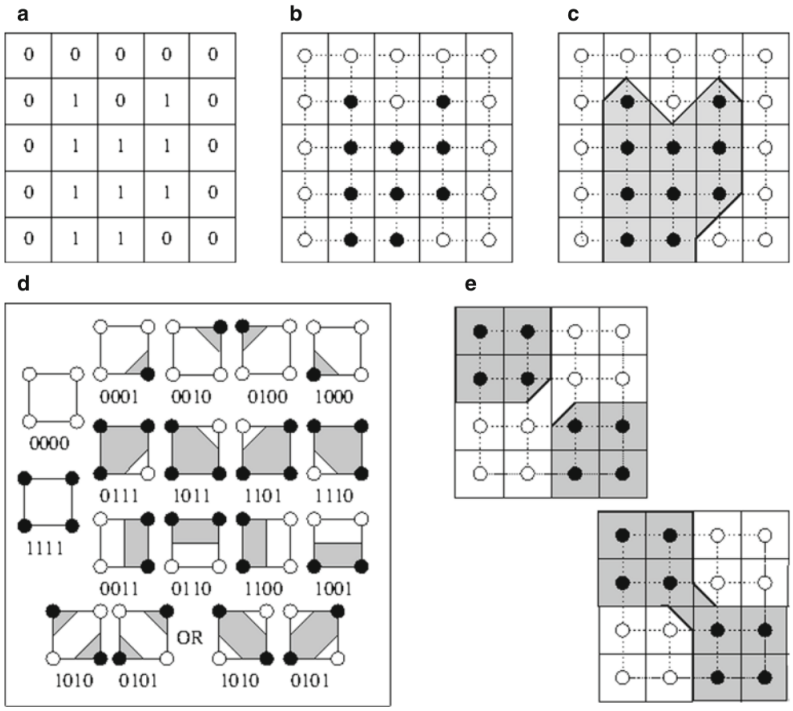
Marching squares algoritam

Kako bi shvatili *marching cubes* algoritam, prvo ćemo objasniti analognu, jednostavniju verziju algoritma u 2D okolini, *marching squares* algoritam (algoritam pokretnih kvadrata).

Binarna slika je interpretirana kao mreža kvadrata (Slika 4.1a), čije vrhove čine centri četiri susjedna piksela. Svaki kvadrat (ćelija) ima četiri binarne vrijednosti, svaki vrh ima vrijednost 0 (bijelo) ili 1 (crno) ovisno o vrijednosti odgovarajućeg piksela (Slika 4.1b). Kada su četiri vrha ćelije crna (bijela), površina ćelije je dio objekta (pozadine). Kada su

vrijednosti jednog ili više vrhova ćelije različite, tada je ćelija dio objekta i dio pozadine. U tom slučaju, dio konture objekta mora biti prisutan unutar ćelije kako bi odvojio površinu koja priprada objektu od površine koja pripada pozadini.

Da bi utvrdili koji dio ćelije je unutar, a koji izvan objekta, uzimamo u obzir vrijednosti vrhova i njihove susjedne vrhove. Na primjer, ako je donji desni vrh crni, a preostala tri bijela, tada mora postojati dio konture koji odvaja donji desni vrh od preostala tri. Najjednostavnija kontura je segment pravca čije se krajnje točke nalaze na pola puta između crnog i bijelih vrhova (Slika 4.1d, slučaj 0001). To odvaja kvadratnu ćeliju na trokut (dio unutar objekta) i peterokut (dio izvan objekta).



Slika 4.1: *Marching squares*

Analizom možemo uočiti da postoji $2^4 = 16$ mogućih kombinacija za vrijednosti vrhova te dobivamo sve moguće konfiguracije ćelija (Slika 4.1d). Svaka konfiguracija ćelija ima jedinstveni četveroznamenkati binarni indeks formiran od binarnih vrijednosti vrhova ćelije, u redoslijedu suprotnom od kazaljke na satu gdje je najmanje značajan bit u donjem desnom kutu. Uzmimajući u obzir rotaciju, simetriju i komplementarnost, 16 slučajeva

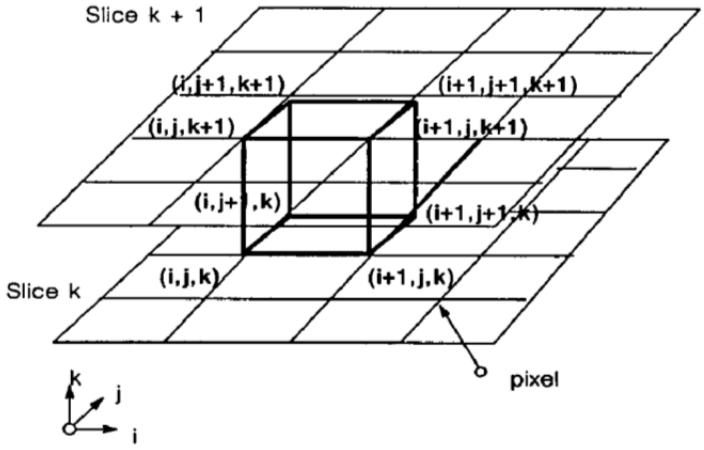
možemo reducirati na 4: svi vrhovi su jednaki (slučajevi 0000 i 1111), jedan vrh je različit (slučajevi 0001,0010,0100,1000,0111,1011,1101 i 1110), dva susjedna vrha su jednaka (slučajevi 0011,0110,1100 i 1001), i dva nasuprotna vrha su jednaka (slučajevi 1010 i 0101). Uočimo da zadnji slučaj ima dvije moguće interpretacije, kao što je prikazano u zadnjem redu na slici 4.1d).

Granica objekta se konstruira sekvencijalno ispitujući svaku ćeliju, izračunavajući njen indeks na temelju binarnih vrijednosti vrhova i utvrđujući koji se granični segment pojavljuje u svakoj ćeliji, ukoliko ih ima. Rezultat je po dijelovima linearna aproksimacija granice objekta (Slika 4.1c). Primjetimo da će susjedni granični segmenti uvijek biti povezani jer dijele krajnju točku koja je na polovištu između bijelog i crnog vrha. Također, redoslijed kojim se obilaze ćelije je nevažan, sve dok se sve ćelije obiđu barem jednom. Složenost algoritma je linearna u ovisnosti o broju ćelija, a time i veličini slike (broju piksela).

Slučajevi 5 (0101) i 10 (1010) se mogu interpretirati na dva načina, što dovodi do dvosmislenosti u sedlastim točkama (Slika 4.1e). Dvosmislenost se može riješiti koristeći prosječnu vrijednost piksela (a ne binarnu) za centar ćelije te pretvoriti tu vrijednost u binarnu kako bi utvrdili pripada li unutrašnjost ćelije objektu ili pozadini. Isto tako, umjesto polovišta stranica, možemo izračunati linearnu kombinaciju originalnih vrijednosti piksela za određivanje pozicije graničnih točaka.

Marching cubes algoritam

Marching squares algoritam može biti direktno proširen na volumetrijske slike. Slika se sastoji od voksela sa binarnim vrijednostima. Ćelije su kocke čije vrhove čine centri osam susjednih voksela, po četiri na dva uzastopna poprečna presjeka (Slika 4.3a).



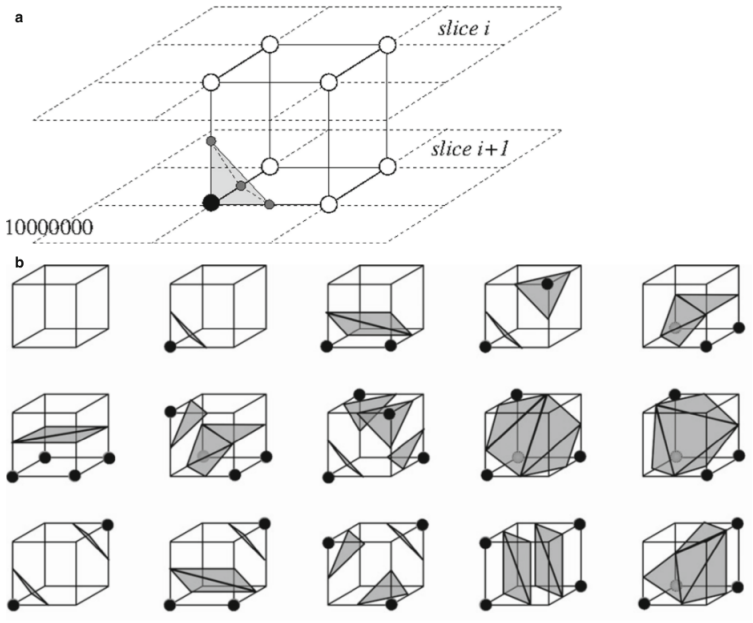
Slika 4.2: Koordinate vrhova kocke

Kada je svih osam vrijednosti vrhova ćelije crno, volumen ćelije je dio objekta. Analogno, kada je vrijednost svih vrhova bijelo, tada je ćelija dio pozadine. Kada je jedna ili više vrijednosti vrhova ćelije različito, ćelija je dio objekta i dio pozadine. U tom slučaju, dio granice objekta mora biti unutar ćelije kako bi odvojio objekt od pozadine. Granice su sada površine umjesto dužina. Na primjer, pretpostavimo da je donji lijevi prednji vrh crni, a preostali su bijeli (Slika 4.3a). Crni vrh je dio tri susjedne strane kocke. Svaka strana je kvadrat s jednim crnim vrhom i tri bijela. Dužina razdvaja crni vrh od bijelih na svakoj strani - tri dužine formiraju trokut koji razdvaja donji crni vrh od preostalih sedam bijelih.

Kocka ima 8 vrhova od kojih svaki vrh može imati dva stanja, 1 ili 0. Binarne vrijednosti vrhova tvore osmeroznamenkasti binarni indeks (Slika 4.4) te postoji $2^8 = 256$ mogućih konfiguracija kocke. Numeriranjem tih 256 slučajeva kreiramo tablicu presjeka koja sadrži bridove presjeka za svaki od 256 slučajeva. Rotacijom, simetrijom i komplementom možemo reducirati broj konfiguracija kocke na 15 (Slika 4.3b) [6].

Nakon što algoritam, ovisno o stanju vrhova kocke, odredi indeks, koristi ga kao pokazivač na tablicu slučajeva. Određivanjem slučaja iz tablice poznato je koje bridove kocke površina presijeca. Kako bi se odredila točka presijecanja brida x_p, koristi se linearna interpolacija s obzirom na gustoću v u dva susjedna vrha (s koordinatama npr. $[i, j, k]$ i $[i + 1, j, k]$ - slika 4.2) prema formuli (za ravninu u smjeru osi x):

$$x_p = i + \frac{T - v[i]}{v[i + 1] - v[i]} \tag{4.1}$$

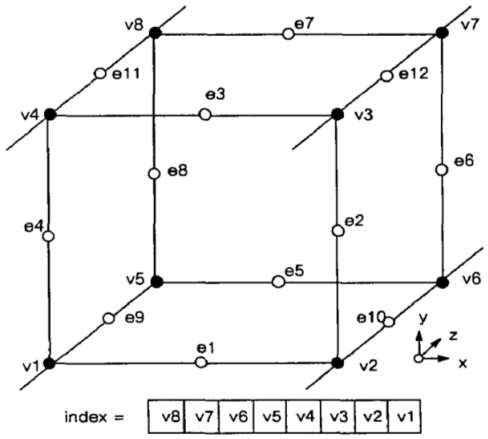


Slika 4.3: *Marching cubes*

Posljedica navedene linearne interpolacije je postavljanje točke presjeka bliže onom vrhu u kojem je gustoća bliža gustoći izopovršine (ulazna varijabla T algoritma). Interpolacije višeg stupnja pokazuju malo poboljšanje u točnosti jer algoritam u jednom koraku stvara jedan, do maksimalno četiri trokuta pa se koristi linearna interpolacija. [11]. Također, ovisnost interpolacije o gustoćama u dva susjedna vrha osigurava neprekinutost modelirane površine jer ne ovisi o koraku algoritma.

Posljednji korak *marching cubes* algoritma je računanje normala za sve vrhove trokuta koji u konačnici određuju željenu površinu. Algoritmi za prikaz modela u prostoru (engl. Rendering algorithms) koriste izračunate normale za stvaranje Gouraud-osjenčanih slika [11]. Za izračun normala, prvo se računa gradijent u dva susjedna vrha kocke kao prostorna derivacija funkcije gustoće (formula 4.2). Na površinama konstantne gustoće gradijent nema komponentu tangencijalnu na površinu. Dapače, ako se radi o prijelazu između objekta i pozadine, gradijent ima samo komponentu u smjeru normale na površinu. Zaključno, moguće je odrediti normalu na površinu jednake gustoće na mjestu granice objekta i pozadine jer je na tom području iznos gradijenta različit od nule i u smjeru normale.

$$\bar{g}(x, y, z) = \nabla \bar{f}(x, y, z) \tag{4.2}$$



Slika 4.4: Osmeroznamenkasti binarni indeks i oznake stanja vrhova

Gradijent u vrhu kocke se računa centralnom diferencijom (razlikom, u slučaju jedinične duljine brida kocke) po koordinatnim osima prema formulama:

$$G_x(i, j, k) = \frac{D(i+1, j, k) - D(i-1, j, k)}{\Delta x}, \tag{4.3}$$

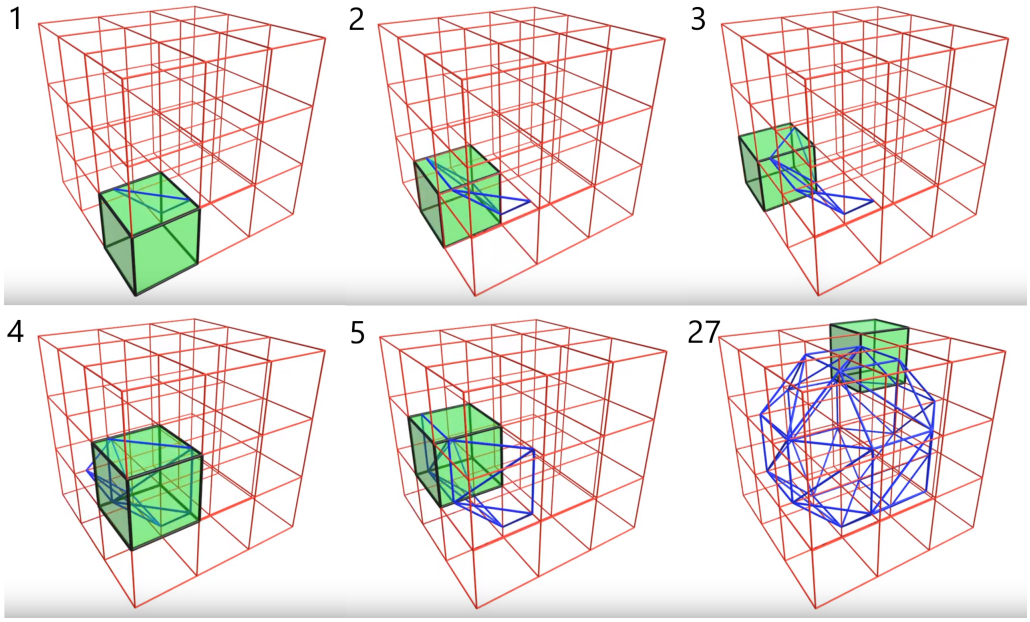
$$G_y(i, j, k) = \frac{D(i, j+1, k) - D(i, j-1, k)}{\Delta y}, \tag{4.4}$$

$$G_z(i, j, k) = \frac{D(i, j, k+1) - D(i, j, k-1)}{\Delta z}, \tag{4.5}$$

gdje je $D(i, j, k)$ gustoća u pikselu (i, j) u presjeku k (engl. slice), a $\Delta x, \Delta y, \Delta z$ su duljine bridova kocke. Važno je primijetiti da je za računanje gradijenata u svim vrhovima kocke potrebno imati učitana četiri poprečna presjeka istovremeno u memoriji jer je potrebno promatrati i piksele izvan kocke (ili ih se može estimirati). Linearnom interpolacijom (zbrajanjem vektora) gradijenata dva susjedna vrha kocke u točki presjecanja računa se gradijent u vrhu trokuta. Dijeljenjem dobivenog vektora gradijenta sa njegovim iznosom dobiva se normala u vrhu trokuta kojeg rendering algoritam uzima u obzir.

Ukratko, *marching cubes* algoritam stvara površinu iz trodimenzionalnog skupa podataka na sljedeći način (Slika 4.5):

1. Učita četiri uzastopna poprečna presjeka u memoriju.
2. Pretraži dva poprečna presjeka i kreira kocku od četiri susjedna piksela na prvom i četiri susjedna piksela na drugom poprečnom presjeku.
3. Izračuna indeks kocke uspoređujući gustoću svakog vrha s gustoćom izopovršine.
4. Pomoću indeksa, potraži popis bridova u unaprijed izračunatoj tablici presjeka.
5. Pronađe presjeke brida i površine pomoću linearne interpolacije, koristeći gustoće u krajnjim točkama brida.
6. Računa normalu u svakom vrhu kocke koristeći centralnu diferenciju. Interpolira normalu za svaki vrh trokuta.
7. Vraća vrhove trokuta i njihove normale.



Slika 4.5: Slikoviti prikaz *marching cubes* algoritma[1]

[1]Izvor slike 4.5: `https://www.youtube.com/watch?v=B_xk71YopsA/`

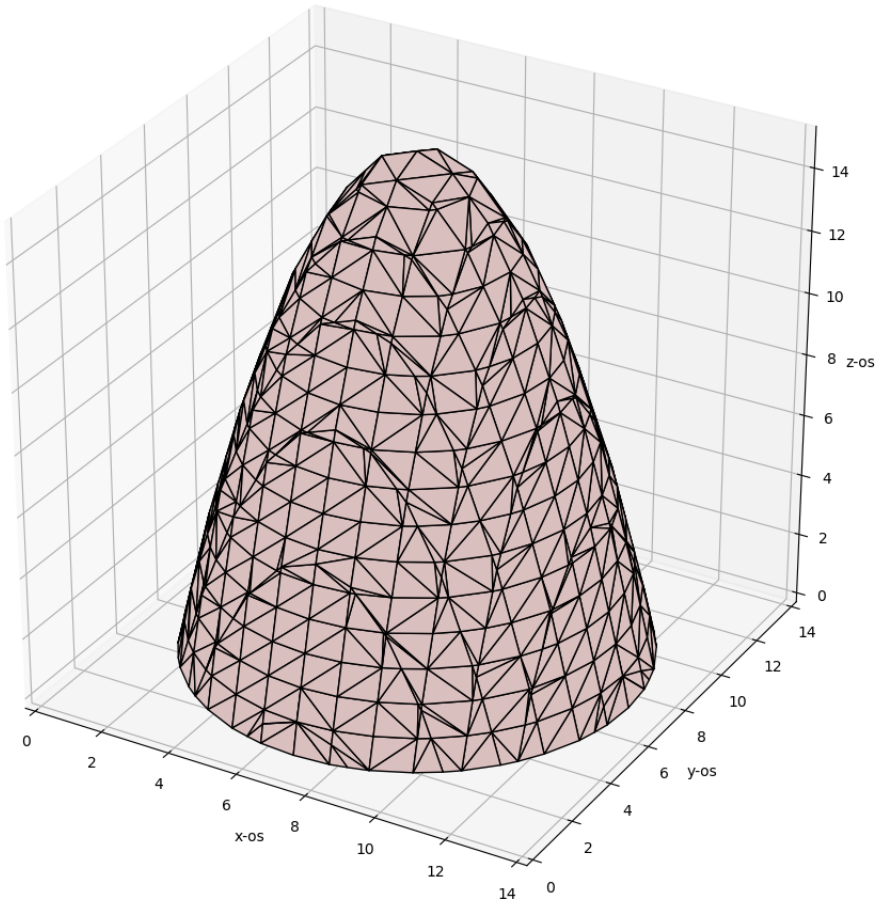
4.2 Primjena algoritma u Pythonu

U ovom dijelu poglavlja dan je jednostavan primjer primjene *marching cubes* algoritma u programskom jeziku Python. Na primjeru paraboloida, koji je u ovom slučaju idealizirani prikaz zuba, *marching cubes* algoritam stvara mrežu trokuta.

U kodu je implementirana funkcija koja vraća paraboloid, a zatim je pozvan klasični *marching cubes* algoritam koji je opisan u poglavlju 4. Implementacija *marching cubes classic* algoritma je dana u dodatku B i može se pronaći na internetu [10].

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d.art3d import Poly3DCollection

from skimage import measure

def paraboloid(a, b, c, spacing=(1., 1., 1.), levelset=False):

    if (a <= 0) or (b <= 0) or (c <= 0):
        raise ValueError('Parameters a, b, and c must all be > 0')

    offset = np.r_[1, 1, 1] * np.r_[spacing]

    # Calculate limits, and ensure output volume is odd & symmetric
    low = np.ceil((- np.r_[a, b, c] - offset))
    high = np.floor((np.r_[a, b, c] + offset + 1))

    for dim in range(3):
        if (high[dim] - low[dim]) % 2 == 0:
            low[dim] -= 1
        num = np.arange(low[dim], high[dim], spacing[dim])
        if 0 not in num:
            low[dim] -= np.max(num[num < 0])

    # Generate (anisotropic) spatial grid
    x, y, z = np.mgrid[low[0]:high[0]:spacing[0],
                       low[1]:high[1]:spacing[1],
                       low[2]:high[2]:spacing[2]]

    if not levelset:
        arr = ((x / float(a)) ** 2 +
               (y / float(b)) ** 2 +
               (z / float(c)) ) <= 0
    else:
        arr = ((x / float(a)) ** 2 +
               (y / float(b)) ** 2 +
```

```
                (z / float(c)) )

    return arr


parab_base = paraboloid(6, 6, 15, levelset=True)


# Use marching cubes to obtain the surface mesh of paraboloid
verts, faces = measure.marching_cubes_classic(parab_base, 0)

# Display resulting triangular mesh using Matplotlib
fig = plt.figure(figsize=(10, 10))
ax = fig.add_subplot(111, projection='3d')

# Fancy indexing: `verts[faces]` to generate a collection of triangles
mesh = Poly3DCollection(verts[faces])
mesh.set_edgecolor('k')
mesh.set_facecolor('#DABFBF')
ax.add_collection3d(mesh)

ax.set_xlabel("x-os")
ax.set_ylabel("y-os")
ax.set_zlabel("z-os")

ax.set_xlim(0, 14)
ax.set_ylim(0, 14)
ax.set_zlim(0, 15)

plt.tight_layout()
plt.show()
```



Slika 4.6: Primjena marching cubes algoritma na paraboloidu

Bibliografija

[1] Francois Budin, *ModelToModelDistance*, 2015, https://www.slicer.org/wiki/Documentation/Nightly/Extensions/ModelToModelDistance, posjećena 2019-09-08.

[2] BWH i 3D Slicer contributors, *3D Slicer*, 2019, https://www.slicer.org/, posjećena 2019-09-06.

[3] Ennex Corporation., *The STL Format*, 2019, http://www.fabbers.com/tech/STL_Format, posjećena 2019-08-17.

[4] Kalpathy Cramer J. Finet J. Fillion Robin J C. Pujol S. Bauer C. Jennings D. Fennessy F. Sonka M. Buatti J. Aylward S. Miller J.V. Pieper S. Kikins R. Fedorov A., Beichel R., *3D Slicer as an Image Computing Platform for the Quantitative Imaging Network*, Magnetic Resonance Imaging **9** (2012), br. 30, 1323–1341.

[5] Stančić H. Hofmann G., *Arhiviranje medicinskih slika u digitalnome obliku*, Physiotherapia Croatica **1** (2016), br. 14, 94–98.

[6] L. Joskowicz, *Modeling and Simulation*, Intraoperative Imaging and Image-Guided Therapy (2014), 49–61.

[7] J. Leder, *Analiza naprezanja u jednokorijenskom zubu kod djelovanja ortodontskih sila*, Magistarska radnja, Sveučilište u Zagrebu, Fakultet strojarstva i brodogradnje, Zagreb, Hrvatska, 2008.

[8] D. Martinović, *CBCT u ortodonciji*, Magistarska radnja, Sveučilište u Zagrebu, Stomatološki fakultet, Zagreb, Hrvatska, 2016.

[9] Strugačevac P., *Teorijska osnova imaging CT tehnike*, Klinička bolnica Osijek, 1999.

[10] scikit image, *_marching_cubes_classic.py*, 2019, https://github.com/scikit-image/scikit-image/blob/master/skimage/measure/_marching_cubes_classic.py#L7, posjećena 2019-09-06.

[11] Harvey E. Cline William E. Lorensen, *Marching Cubes: a High Resolution 3D Surface Construction Algorithm*, Computer Graphics **21** (1987), br. 4, 163–169.

[12] Petrinec B. Ćupurdija A., *Kompjutorizirana tomografija - CT*, Matematičko-fizički list **2** (2017-2018), br. LXVIII, 80–86.

[13] Zdravstvena škola Split, *CT u radiologiji*, 2014, http://ss-zdravstvena-st.skole.hr/upload/ss-zdravstvena-st/images/static3/1762/attachment/CT_u_radiologiji.pptx, posjećena 2019-08-11.

Sažetak

U ovom radu analiziran je rad sustava 3D Slicer na primjeru rekonstrukcije zuba dobivenog segmentacijom CBCT snimke čeljusti. Dobiveni su CAD modeli dentina i cakline lijevog centralnog sjekutića, te CAD model njihove dodirne površine koja je važna prilikom 3D ispisa, ali i kod simulacija zbog postavljanja fizikalnih uvjeta pod kojima se ostvaruje kontakt. Opisan je *marching cubes* algoritam koji kreira mrežu trokuta izopovršina iz trodimenzionalnog diskretnog skalarnog polja i jedan je od temeljnih algoritama za rekonstrukciju površine.

Summary

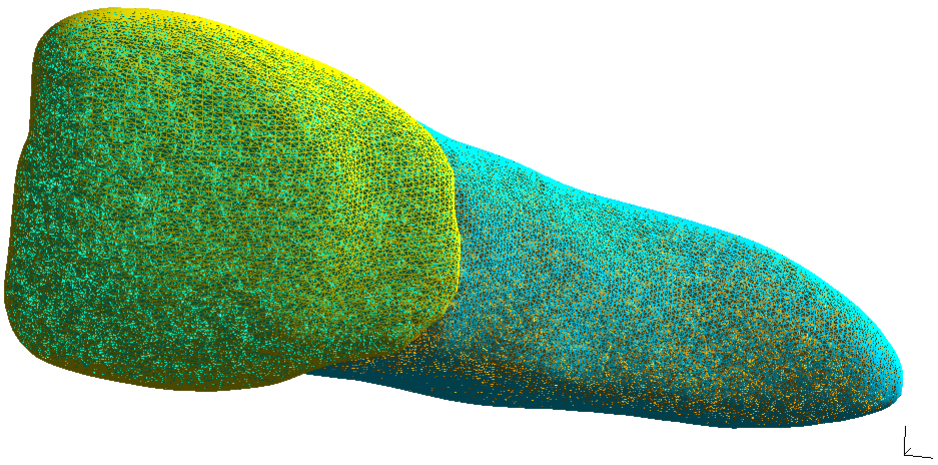
This thesis analyzes the work of 3D Slicer on the example of tooth reconstruction obtained by segmentation of CBCT jaw scans. CAD models of dentin and enamel of the left central incisor were obtained, as well as a CAD model of their contact surface, which is important for 3D printing, but also for simulations because of the physical conditions under which contact is made. The *marching cubes* algorithm that creates triangle surface mesh from a three-dimensional discrete scalar field is described and is one of the fundamental algorithms for surface reconstruction.

Životopis

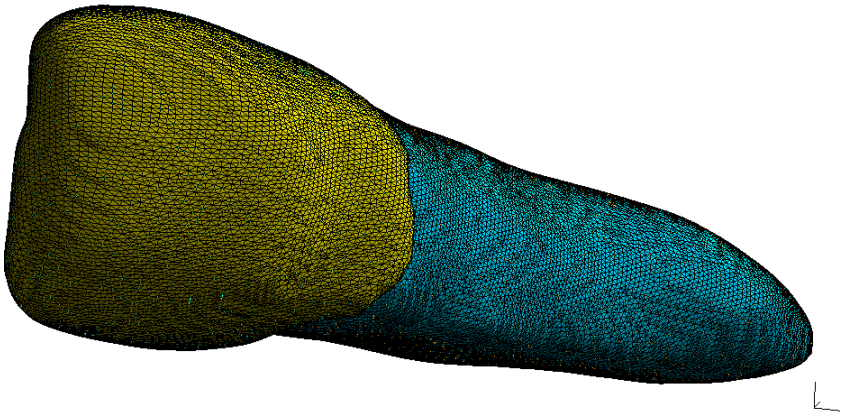
Rođena sam 04. rujna 1992. u Zagrebu. Školovanje započinjem u Osnovnoj školi Ivana Gorana Kovačića u Zagrebu nakon koje upisujem XV. gimnaziju (MIOC). Matematički odsjek Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu upisujem 2011. godine. Tijekom studiranja sam bila demonstrator iz kolegija "Elementarna geometrija" i "Kombinatorna i diskretna matematika". Diplomski studij "Računarstvo i matematika" upisujem 2016. godine. Tijekom studiranja sam radila u Zagrebačkoj banci u odjelu informatike.

Dodatak A

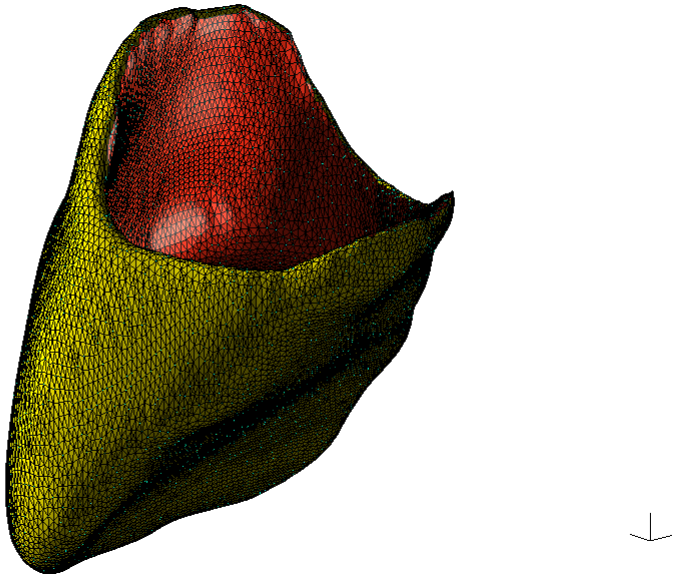
U program Gmsh učitane su STL datoteke dobivene u 3D Sliceru. Na tim podacima je napravljena 3D mreža, označeni su rubovi i kontaktna površina između cakline i dentina. Na novodobivenoj 3D mreži zuba možemo rješavati različite rubne zadaće. Upravo zbog toga, površinu cakline, tj. dentina, označavamo da bi na tom dijelu mreže mogli zadati pripadne rubne uvjete.



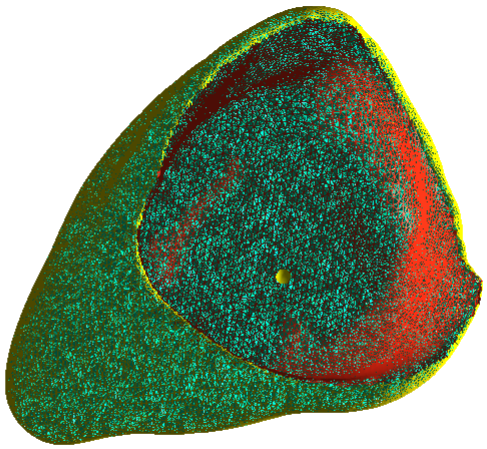
Slika 4.7: 3D mreža zuba



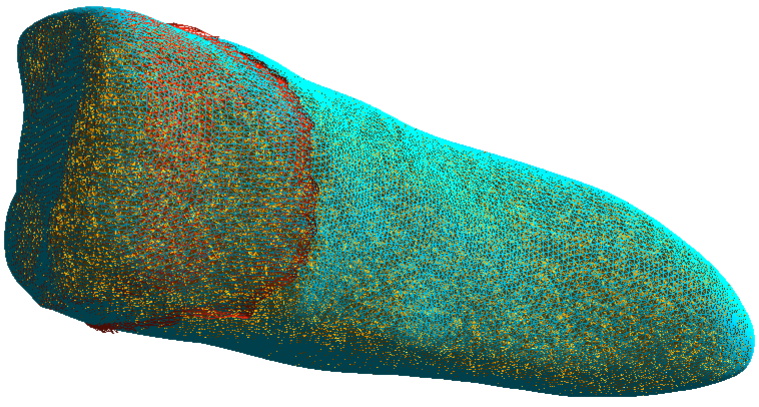
Slika 4.8: 3D mreža zuba



Slika 4.9: Crvenom bojom je označena dodirna površina unutar cakline



Slika 4.10: Caklina



Slika 4.11: Dentin

Dodatak B

Implementacija *marching cubes* algoritma u programskom jeziku Python [10].

```
import numpy as np
import scipy.ndimage as ndi
from .._shared.utils import warn
from . import _marching_cubes_classic_cy


def marching_cubes_classic(volume, level=None, spacing=(1., 1., 1.),
                           gradient_direction='descent'):
    """

    volume : (M, N, P) array of doubles
        Input data volume to find isosurfaces. Will be cast to
        `np.float64`.
    level : float
        Contour value to search for isosurfaces in `volume`. If not
        given or None, the average of the min and max of vol is used.
    spacing : length-3 tuple of floats
        Voxel spacing in spatial dimensions corresponding to numpy array
        indexing dimensions (M, N, P) as in `volume`.
    gradient_direction : string
        Controls if the mesh was generated from an isosurface with
        gradient descent toward objects of interest (the default),
        or the opposite.
        The two options are:
        * descent : Object was greater than exterior
        * ascent : Exterior was greater than object
    Returns
    -------
    verts : (V, 3) array
        Spatial coordinates for V unique mesh vertices. Coordinate
        order matches input `volume` (M, N, P).
    faces : (F, 3) array
        Define triangular faces via referencing vertex indices
```

```
        from ``verts``. This algorithm specifically outputs triangles,
        so each face has exactly three indices.

    """
    # Check inputs and ensure `volume` is C-contiguous
    #for memory views
    if volume.ndim != 3:
        raise ValueError("Input volume must have 3 dimensions.")
    if level is None:
        level = 0.5 * (volume.min() + volume.max())
    else:
        level = float(level)
        if level < volume.min() or level > volume.max():
            raise ValueError("Surface level must be within
                             volume data range.")
    if len(spacing) != 3:
        raise ValueError("`spacing` must consist of three floats.")

    volume = np.array(volume, dtype=np.float64, order="C")

    # Extract raw triangles using marching cubes in Cython
    #   Returns a list of length-3 lists, each sub-list containing
    #three tuples. The tuples hold (x, y, z) coordinates for triangle
    #vertices. Note: this algorithm is fast, but returns degenerate
    #"triangles" which have repeated vertices - and equivalent
    #vertices are redundantly placed in every triangle they
    #connect with.
    raw_faces = _marching_cubes_classic_cy.iterate_and_store_3d(volume,
                                                        float(level))

    # Find and collect unique vertices, storing triangle verts as indices.
    # Returns a true mesh with no degenerate faces.
    verts,faces = _marching_cubes_classic_cy.unpack_unique_verts(raw_faces)

    verts = np.asarray(verts)
    faces = np.asarray(faces)

    # Fancy indexing to define two vector arrays from triangle vertices
    faces = _correct_mesh_orientation(volume, verts[faces], faces,spacing,
                                      gradient_direction)

    # Adjust for non-isotropic spacing in `verts` at time of return
    return verts * np.r_[spacing], faces


def mesh_surface_area(verts, faces):
    """
```

```
    Compute surface area, given vertices & triangular faces
    Parameters
    ----------
    verts : (V, 3) array of floats
        Array containing (x, y, z) coordinates for V unique mesh vertices.
    faces : (F, 3) array of ints
        List of length-3 lists of integers, referencing vertex
        coordinates as provided in `verts`
    Returns
    -------
    area : float
        Surface area of mesh. Units now [coordinate units] ** 2.

    """
    # Fancy indexing to define two vector arrays from triangle vertices
    actual_verts = verts[faces]
    a = actual_verts[:, 0, :] - actual_verts[:, 1, :]
    b = actual_verts[:, 0, :] - actual_verts[:, 2, :]
    del actual_verts

    # Area of triangle in 3D = 1/2 * Euclidean norm of cross product
    return ((np.cross(a, b) ** 2).sum(axis=1) ** 0.5).sum() / 2.


def _correct_mesh_orientation(volume, actual_verts, faces,
                              spacing=(1., 1., 1.),
                              gradient_direction='descent'):
    """
    Correct orientations of mesh faces.
    Parameters
    ----------
    volume : (M, N, P) array of doubles
        Input data volume to find isosurfaces. Will be cast to
        `np.float64`.
    actual_verts : (F, 3, 3) array of floats
        Array with (face, vertex, coords) index coordinates.
    faces : (F, 3) array of ints
        List of length-3 lists of integers, referencing vertex
        coordinates as provided in `verts`.
    spacing : length-3 tuple of floats
        Voxel spacing in spatial dimensions corresponding to
        numpy array indexing dimensions (M, N, P) as in `volume`.
    gradient_direction : string
        Controls if the mesh was generated from an isosurface
        with gradient descent toward objects of interest (the default),
        or the opposite. The two options are:
        * descent : Object was greater than exterior
```

```
    * ascent : Exterior was greater than object
Returns
-------
faces_corrected (F, 3) array of ints
    Corrected list of faces referencing vertex coordinates in
    `verts`.

"""
# Calculate gradient of `volume`, then interpolate to vertices
#in `verts`
grad_x, grad_y, grad_z = np.gradient(volume)

a = actual_verts[:, 0, :] - actual_verts[:, 1, :]
b = actual_verts[:, 0, :] - actual_verts[:, 2, :]

# Find triangle centroids
centroids = (actual_verts.sum(axis=1) / 3.).T

del actual_verts

# Interpolate face centroids into each gradient axis
grad_centroids_x = ndi.map_coordinates(grad_x, centroids)
grad_centroids_y = ndi.map_coordinates(grad_y, centroids)
grad_centroids_z = ndi.map_coordinates(grad_z, centroids)

# Combine and normalize interpolated gradients
grad_centroids = np.c_[grad_centroids_x, grad_centroids_y,
                       grad_centroids_z]
grad_centroids = (grad_centroids /
                  (np.sum(grad_centroids ** 2,
                          axis=1) ** 0.5)[:, np.newaxis])

# Find normal vectors for each face via cross product
crosses = np.cross(a, b)
crosses = crosses/(np.sum(crosses**2,axis=1)**(0.5))[:,np.newaxis]

# Take dot product
dotproducts = (grad_centroids * crosses).sum(axis=1)

# Find mis-oriented faces
if 'descent' in gradient_direction:
    # Faces with incorrect orientations have dot product < 0
    indices = (dotproducts < 0).nonzero()[0]
elif 'ascent' in gradient_direction:
    # Faces with incorrection orientation have dot product > 0
    indices = (dotproducts > 0).nonzero()[0]
else:
```

```
            raise ValueError("Incorrect input %s in 'gradient_direction', see"
                             "docstring." % (gradient_direction))

        # Correct orientation and return, without modifying original data
        faces_corrected = faces.copy()
        faces_corrected[indices] = faces_corrected[indices, ::-1]

    return faces_corrected
```