VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
KOMPIUTERIJOS KATEDRA

Baigiamasis magistro darbas

Medicininių 3D vaizdų analizė

Atliko:
Jevgenij Teodorovič parašas

Vadovas:
dr. Valdas Rapševičius

Vilnius
2017

Turinys

Santrauka

Medicininių vaizdų kiekis per pastaruosius kelis metus padvigubėjo. Rankinė tokių duomenų analizė yra sudetinga ir užima daug laiko. Vaizdų analizės kompiuteriniai algoritmai skirti padėti medicininiam personalui greitai bei kokybiškai apdoroti tyrimų vaizdus. Šiame darbe bus plačiai aptarta kompiuterinės tomgofrafijos tyrimo esmė, susipažinta su DICOM formato ypatumais. Bus aptartas vaizdų binarizavimo procesas pasinaudojus slenksčio dydžio sąvoka. Bus aptarti keli slenksčio dydžio pasirinkimo algoritmai, pateiktas jų matematinis modelis. Praktinėje darbo dalyje pasinaudojus vieno tyrimo vaizdus bus segmentuojami kaulai bei atliekama jų 3D vizualizacija.

Vėliau pasinaudojus gautais rezultatais bus segmentuojama plaučių sritis. Plaučių segmentavimo dalyje bus aptarti kompiuterinės regos metodai bei algoritmai, plačiai aprašomas jų veikimo principas bei matematinis apibrėžimas. Vėliau pasinaudojus kaulų segmentavimo dalyje gautais rezultatais bus segmentuojami plaučiai. Tam bus Java kalba sukurta sistema su realizuotais algoritmais. Iš segmentuotų plaučių srities taškų aibės bus konstruojamas dengiantysis paviršius pasinaudojus judančių kubų algoritmu. Galų gale bus papildomai įgyvendinta vartotojo sąsaja su 3D vizualizavimo galimybe.

Summary

Analysis of 3D Medical Images

Nowadays medicine generates twice as much medical images as it generated several years ago. Manual analysis and segmentation of this images is a very di cult task which is time consuming and also has quite high error rate. Computer-aided diagnosis by using computer algorithms tries to help medical personnel to make image analysis fast and in a high quality. In this thesis computer tomography process will be deeply discussed. Some principles will be discussed about how CT works and what are main parameters which influences medical images. Format DICOM will be also discussed to show how does are medical images stored and transported via the web. Image binarization techniques will be presented. Several algorithms to calculate threshold values will be discussed along with their mathematical explanation. In practical part of this thesis these methods will be implemented and tested with real CT images. Then best method will be chosen to make whole chest bones segmentation. After final segmentation bones will be visualized in 3D.

In the second part of this thesis solution for lung segmentation will be presented. Firstly, algorithms for image processing will be discussed. Image labeling techniques, mathematical morphology, background removal techniques will be outlined. After that method to segment lung will be presented.

After successful lung segmentation marching cubes algorithm for surface reconstruction will be introduced. Finally 3D visualization solution will be presented. We will briefly cover WebGL and Three.js library usage.

Practical part of lung segmentation and reconstruction is also presented in this thesis. Implementation consists of three parts: lung points segmentation, surface reconstruction and visualization. First two parts are implemented using Java programming language, third part is implemented in Java Script and HTML.

Įvadas

Kompiuterinės tomografijos vaizdai vieni svarbiausių klinikinių tyrimų instrumentų. Tokių vaizdų analizavimas, interpretavimas naudojant kompiuterines sistemas labai svarbus planuojant chirurgines operacijas, gydymo planus. Kompiuterinės tomografijos vaizdai susideda iš sekos 2D vaizdų (skilčių) su fiksuotu atstumu tarp jų. Šis atstumas yra žymiai didesnis už pikselio dydį 2D vaizde. Analizės tikslas šias vaizdų sekas interpretuoti ir modeliuoti kaip 3D objektus.

Interpretuojant medicininius vaizdus susiduriama su keliomis pagrindinėmis problemomis, viena jų yra medicinos etikos problema. Interpretavimo rezultatuose neturi pasirodyti klaidingų artefaktų, kurie bylotų apie lygos (pvz. vėžio, tuberkuliozės) buvimą. Tokie rezultatai pakenks tiek pacientams, kurie klaidingai bus informuoti apie neesamos lygos buvimą, tiek medicinos specialistui, kuris gaiš laiką bandydamas atrasti neesamos lygos būvimo priežastis ir gydymo būdus. Kita problema su kuria tenka susidurti yra praktinis vaizdų apdorojimo greitis. KT vaizdų kiekvienais metais vis daugėja, tad jų apdorojimo greitis neturėtų trukti labai ilgai.

Kompiuterinės tomografijos vaizdų analizėje vienas svarbiausių žingsnių yra organų bei audinių išskyrimas. Šis procesas yra sudėtingas, kadangi kompiuterinės tomografijos tyrimo metu gaunami vaizdai dažniausiai turi smulkius triukšmus bei trūkumus. Žmogaus organai bei audiniai tyrimo rezultatuose turi skirtingus ryškumo lygius. Ryškumo lygis priklauso nuo audinio tankio, struktūros bei sudėties, tad dauguma organų turi panašią struktūrą, ko pasekoje, tyrimuose turi panašų ryškumo lygį.

Medicininių vaizdų analizės uždavinį galime padalinti į keturis esminius žingsnius:

1. Duomenų (vaizdų) gavimas. Šis žingsnis numato duomenų gavimą medicinos tyrimuose naudojamais įrankiais, pavyzdžiui, kompiuteriniu tomografu. Šio žingsnio metu gauti pirminiai duomenys yra rekonstruojami specializuotų programų pagalba, kurių išėjimo rezultatas yra sluoksniuoti dviejų dimensijų vaizdai.

2. Vaizdų apdorojimas. Šiame žingsnyje bandoma apdoroti gautus vaizdus išskiriant dominančias struktūras, objektus.

3. Paviršiaus sukonstravimas. Apdorojimo metu gautos struktūros yra taškų aibės. Norint gauti paviršiaus modelį, būtina atlikti papildomą paviršiaus sukonstravimo žingsnį.

4. Atvaizdavimas. Sukonstravus paviršių galutinis tikslas yra atvaizduoti jį vartotojui (medikui).

Pakankamai dažnai šios srities darbuose bandoma realizuoti tik antrąjį žingsnį. Šio baigiamojo darbo pagrindinis tikslas, atlikti trijų paskutinių žingsnių analizę, bei pateikti jų realizaciją. Darbe bus sukurtas sprendimas, kuris sugeba iš pateiktu vaizdų išskirti plaučių sritį, sukonstruoti jos paviršių bei patogiai pateikti vartotojui.

Šis darbas susideda iš penkių pagrindinių dalių:

1. Darbo pradžioje bus aprašoma probleminė sritis. Bus trumpai aptarta žmogaus kūno anatomija, vėliau bus pateikta pagrindinė informacija apie kompiuterinės tomografijos veikimo principus.

2. Antrojoje dalyje bus aprašomas medicininių duomenų saugojimo formatas DICOM. Bus aprašyta jo struktūra, išskirti pagrindiniai elementai.

3. Kaulų segmentavimo dalyje bus aptartas krūtinės ląstos kaulų segmentavimo procesas. Bus pasiūlytas metodas kaulams išskirti, taip pat aptarti gauti rezultatai bei algoritmų įgyvendinimas.

4. Plaučių ertmės segmentavimo dalyje bus pasiūlytas sprendimas segmentuoti plaučių taškus. Iš pradžių bus aptarti kompiuterinės regos metodai bei algoritmai, vėliau bus kuriamas automatinis sprendimas segmentacijai atlikti.

5. Rekonstravimo bei vizualizavimo dalyje bus pasiūlyti algoritmai išskirtiems plaučių ertmės taškams rekonstruoti. Bus aptartas paviršiaus rekonstravimo metodas. Vizualizavimo dalyje bus pateiktas sprendimas vizualizuoti šiame darbe gautus rezultatus.

Šis darbas yra mokslo tiriamojo darbo tęsinys. Kaulų segmentavimo dalis yra paimta iš mokslo tiriamojo darbo.

1. Kompiuterinė tomografija

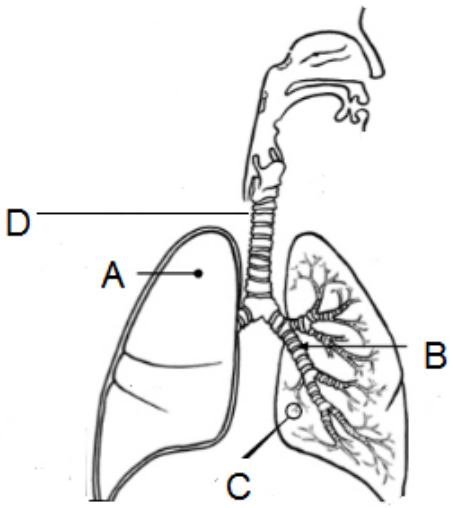
Šioje darbo dalyje aptarsime problemos sritį. Iš pradžių bus pateikta informacija apie žmogaus kūno anatomiją, akcentuojant žmogaus plaučius. Šio skyriaus pabaigoje bus aptarti medicininiai diagnostikos metodai skirti diagnozuoti plaučių susirgimus, išskiriant, bei detaliai aprašant, kompiuterinės tomografijos tyrimą.

1.1. Plaučių anatomija

Plaučiai didžiausias žmogaus organas krūtinės ląstoje. Šis organas atsakingas už dujų apykaitą organizme, kuri yra gyvybiškai svarbi tiek žmogui tiek kitiem kvėpuojantiems organizmams. Plaučiai yra porinis organas susidedantis iš kairiojo ir dešinėjo plaučio. Dešinysis plautis yra didesnis, nes širdis, esanti kairiojoje krūtinės ląstos dalyje, užima didesnę jos dalį. To pasekoje dešinysis plautis sudarytas iš trijų skilčių, o kairysis iš dviejų. Kiekvienas plautis apgaubtas pleura, kuri saugo plaučius kvėpavimo metu.

Plaučių pradžia dažnai vadinama trachėja. Ji sujungia gerklės žiedinę kremzlę su bronchais. Bronchai savo ruoštu išsišakoja besileisdami gilyn į plaučius, suformuodami medžio struktūros darinius. Bronchų pabaigoje yra alveolės. Alveolės tai smulkiausios puslėlės, kuriose vyksta dujų apykaita.

1 pav. galime matyti plaučių sandarą.

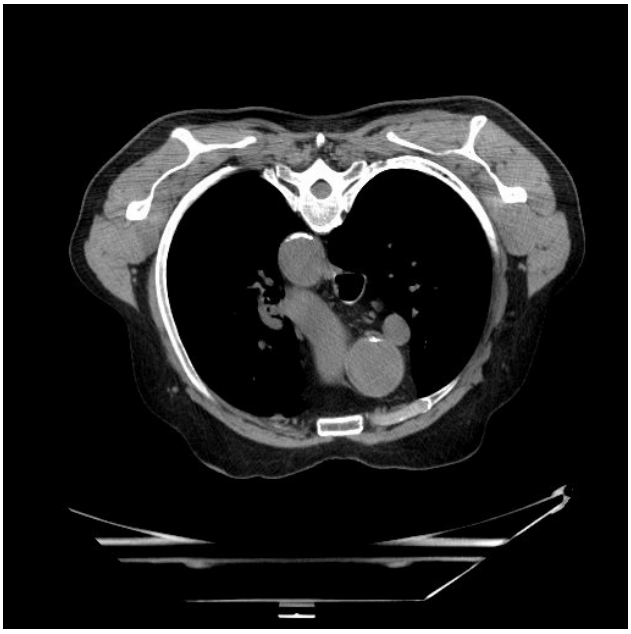


1 pav. Plaučių ertmės sandara. Kur A - plautis (dešinysis), B - kairysis bronchas, C - alveolė, D - trachėja.

1.2. Kompiuterinė tomografija

Praeito amžiaus pabaigoje medicininiai tyrimo metodai išgyveno drastišką kitimą ir tobulėjimą. Pradėjus medicinoje taikyti rentgeno spinduliuotę pirmieji tyrimai buvo rentgenogramų analizavimas. Rentgeno spinduliai skruosdami žmogaus kūną patenka ant foto juostos palikdami informaciją apie žmogaus organus. Žmogaus kūnas nėra homogeniškas, o susideda iš organų, kaulų, minkštųjų audinių, kurie savo ruoštu turi skirtingas savybes (skirtingas tankis, sudėtis). Šie skirtumai lemia rentgeno spindulių sugėrimą, kuris savo ruoštu yra atvaizduojamas rengenogramuose. Šešėliai, šviesesnės sritys rentgenogramuose leidžia analizuoti žmogaus sveikatos būklę,

ligos progresavimą. Pagrindiniai rentgenogramų trukumai jų mažas kontrastiškumas bei neturėjimas informacijos apie objektų gyli kūne.



2 pav. Kompiuterinės tomografijos tyrimo vieno sluoksnio vaizdas.

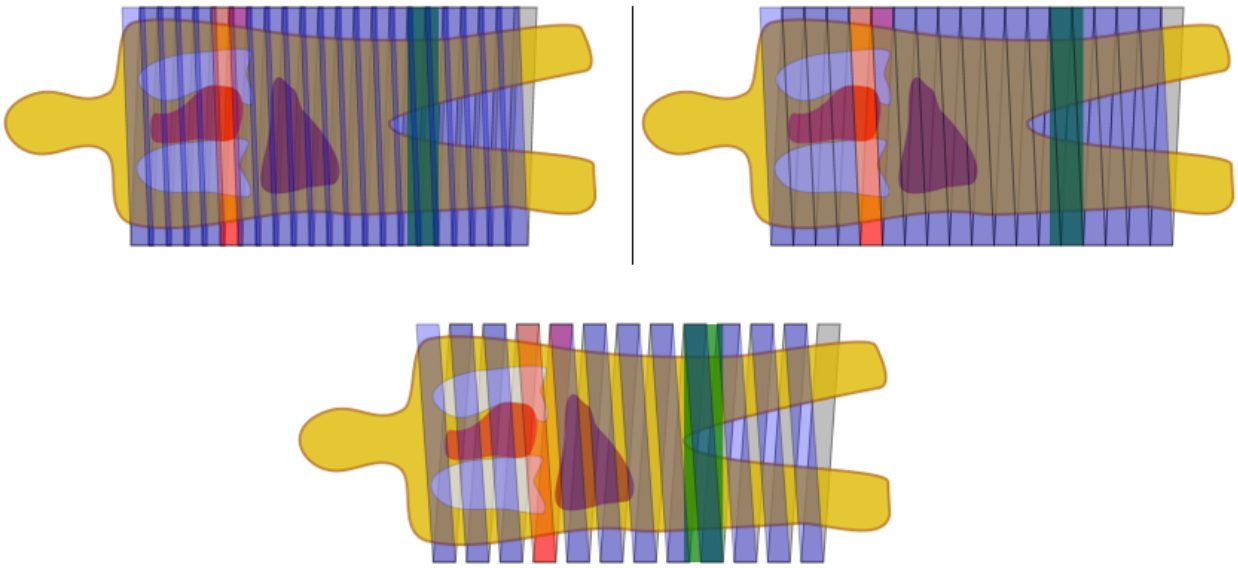
Tobulėjant kompiuterinei technikai tapo įmanoma panaudoti skaičiavimo resursus atliekant medicininius tyrimus. Kompiuterinis tomografas taip pat remiasi rentgeno spinduliuote. Šiuo tyrimo metu skirtingai nei rentgeno tyrime atliekama daug žmogaus rentgenogramų apšvitinant kūną vis kitu kampu. Tyrimo metu kompiuterinis tomografas sukasi aplink tiriamą objektą ir su kiekvienu posūkiu atlieka vis naują rentgenogramą. Atlikus pilną apsisukimą aplink objektą kompiuteris sugeneruoja objekto "pjuvio" vaizdą. Tada kompiuterinis tomografas paslenkia stalą ant kurio guli tiriamas objektas tam tikru reguliuojamu atstumu ir kartoja apsisukimo bei skenavimo žingsnius. Šis būdas vadinamas skenavimas seka, tuo tarpu metodas kai stalas slenka pastoviai besisukant kompiuteriniam tomografui vadinamas spiraliniu skanavimu. Atlikus kompiuterinės tomografijos tyrimą gaunami tam tikro skaičiaus kūno "pjūviu" dviejų dimensijų vaizdai.

Atliekant spiralinę kompiuterinę tomografija būtina atlikti tyrimo parametrų nustatymą. Vienas pagrindinių kompiuterinės tomografijos tyrimų parametrų - nuožulnumas (angl. pitch) 1.1.

$$P = \frac{S_T}{C} \tag{1.1}$$

Kur $S(T)$ stalo poslinkis per viena apsisukimą, C - kolimacija, nusakanti lygiagretaus spinduliuotės pluošto plotį. P - nuožulnumas, koeficientas nusakantis skanavimo padengimo laipsnį.

Pastebėkime, kad nuožulnumui esant 1, stalas slenka per vieną tomografo apsisukimą atitinkantį tomografo skleidžiamam spinduliuotės pluošto pločiui. Tokiu atveju, skanavimas nepalieka tarpų, tuo tarpu kai nuožulnumas mažesnis už 1, spinduliuotės pluošto plotis yra didesnis už stalo poslinkio atstumą. Šiuo atveju vyksta sluoksnių persidengimas. Kai nuožulnumas viršija 1, tada stalo poslinkio atstumas viršija spinduliuotės pluošto plotį ir turime skanavimą su tarpais. 3 pav. galime matyti skanavimo iliustracijas su skirtingais nuožulnumo parametrais. Tyrimuose norint gauti aukštos kokybės skenavimo rezultatus galima pasirinkti nuožulnumo parametrą lygiu 1 arba mažiau. Tokiu atveju, spinduliuotės pluoštas apriepia visą kūno plotą. Pagrindinis tokio skanavimo trūkumas yra ilga skanavimo trukmė, kuri savo ruoštu skleidžia didelę apšvitos dozę.

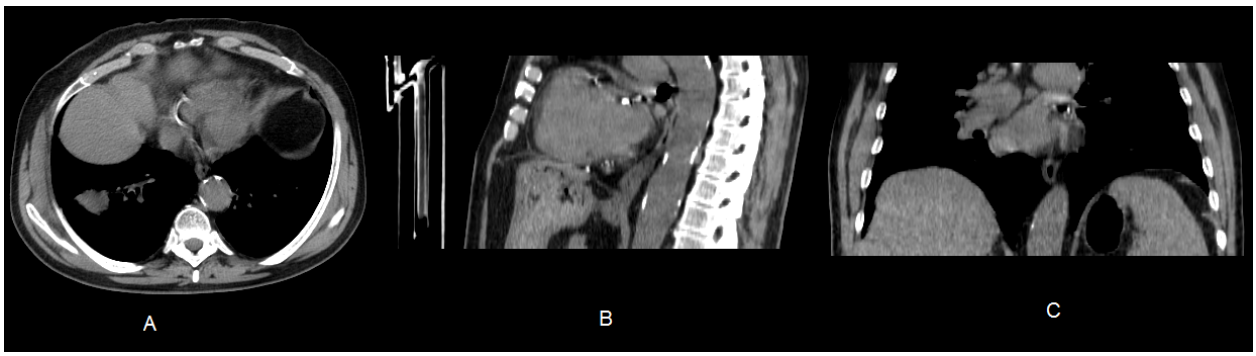


3 pav. Spiralinio skanavimo iliustracija. Spinduliuotės pluošto kelias pavaizduotas mėlyna spalva, vienas apsisukimas vaizduojamas raudona spalva. Pjūvio plotis pavaizduotas žalia spalva. Iliustracija kairėje nuožulnumo dydis 0.8, iliustracija dešinėje nuožulnumo dydis 1, iliustracija apačioje nuožulnumo dydis 1.5.

Kompiuterinės tomografijos tyrimo rezultatai yra sluoksniuoti dviejų dimensijų vaizdai. Kiekviename vaizde galime matyti šviesesnius bei tamsesnius objektus, sritis. Spalvos intensyvumas matuojamas Haunsfildo (angl. Hounsfield) vienetais (HV). Žemiau galime matyti įvairių medžiagų ir audinių charakteristikas kompiuterinės tomografijos tyrimo vaizduose.

Medžiaga	HV
Oras	-1000
Plaučių ertmė	-600 iki -400
Riebalai	-100 iki -50
Vanduo	0
Kraujas	+30 iki +45
Minkštieji audiniai	+100 iki +300
Kaulai	Nuo +400

Kompiuterinės tomografijos tyrimo vaizdus galima atvaizduoti trijuose skirtinguose pjūviuose: Aksialinis pjūvis, Sagitalinis pjūvis, Koronarinis pjūvis. Pjūvio pasirinkimas priklauso nuo to, kokią informaciją norima išgauti, pavyzdžiui plaučių tyrimuose dažniausiai naudojamas aksialinis pjūvis, tuo tarpu stuburui ištirti naudojamas sagitalinis pjūvis. Kompiuterinėse tomografijose vaizduojamus pjūvius galime matyti 4 pav..

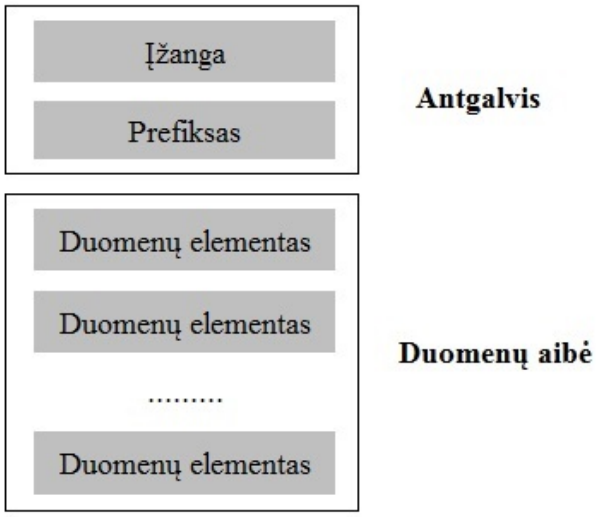


4 pav. Kompiuterinės tomografijos tyrimo vaizdas skirtingose pjūviuose. Kairėje aksialinis pjūvis, viduryje sagitalinis, dešinėje koronarinis.

2. DICOM formatas

Medicininių skaitmeninių vaizdų šaltinių, tokių kaip kompiuterinė tomografija, atsiradimas iškėlė naujų iššūkiu duomenų apdorojimo srityje. Tuo metu dauguma medicininių prietaisų gamintojų naudojo savotiškus, neuniversalius duomenų saugojimo formatus, kas savo ruoštu apsunkino duomenų dalijimąsi tarp skirtingų gamintojų įrenginių. Tuo tarpu medicininių duomenų kiekis sparčiai augo ir tai iškėlė būtinumą sukurti bendrą ir standartizuotą šių duomenų saugojimo bei dalijimosi metodą. Šiam tikslui buvo suformuotas komitetas iš Amerikos radiologijos koledžo (ACR) bei JAV Nacionalinės elektros instaliacijos medžiagų gamintojų asociacijos (NEMA) specialistų. Kelių metų bėgyje komitetas išleido pirmą standartizuoto formato versiją "ACR-NEMA Standards Publication No. 300-198" [1]. Laikui bėgant šis formatas buvo tobulinamas ir galutinė jo versija buvo išleista 1993 pavadinimu "Digital Imaging and Communications in Medicine" kurio abreviatūra yra DICOM.

DICOM failo struktūrą galime stebėti 5 pav..



5 pav. DICOM failo struktūra

Failo antgalvis susideda iš 128 baitų ilgio įžangos bei 4 baitų ilgio prefikso, kurio reikšmė yra "DICM". Tiek įžanga, tiek prefiksas skirti taikomosioms programos iš anksto validuoti, ar failas yra tinkamos struktūros. Duomenų aibė sudaryta iš duomenų elementų sekos. Kiekvienas duomenų elementas sudarytas iš minimum trijų laukų: žymės numerio atributo, reikšmės ilgio bei pačios reikšmės. Žymės numeris privalo atitikti DICOM specifikacijoje numatytas reikšmes. Keli pagrindiniai žymių atributai naudojami kompiuterinės tomografijos vaizdų apibrėžimui galime matyti lentelėje apačioje.

Žymė	Žymės aprašymas
(0010,0020)	Paciento indetifikavimo numeris
(0018,0050)	Vaizdo pjūvio storis
(0018,9307)	Kolimacija
(0018,9309)	Stalo greitis
(0020,0011)	Vaizdo numeris serijoje
(0028,0010)	Eilučių skaičius vaizde
(0028,0011)	Stuleplių skaičius vaizde
(0028,1052)	Tiesinis tranformacijos koeficientas
(7FE0,0010)	Vaizdo duomenys

Šie atributai būtini tolimesnei vaizdų analizei taikomosiose programose. Atributai aprašantys atstumą tarp taškų vaizde bei atstumą tarp dviejų vaizdų pjūvių būtini rekonstruoti vaizdą trijose dimensijose. Tiesinės transformacijos koeficientas reikalingas konvertuoti taškų reikšmes vaizde į HV skalės vienetus. Atributai taipogi aprašo kokio tipo yra pats vaizdas, koks bitų skaičius yra viename taške, kokio tipo sanglauda buvo panaudota glaudžiant vaizdą ir t.t..

3. Susijusių darbų analizė

Viena pagrindinių šiuolaikinės medicininių vaizdų analizes užduočių yra vaizdų segmentavimas. Vaizdų segmentavimas apibrėžiamas kaip vaizdo padalijimas į tarpusavio nesikertančius regionus, kur kiekvienas regionas turi tam tikras charakteristikas, pavyzdžiui, spalvos intensyvumas. Jeigu vaizdo regionas apibrėžiamas R, tada segmentavimo uždavinį galime traktuoti kaip radimą visų sričių $S_k \subset R$, kurių sąjunga ir būtų visas vaizdo regionas R. Apibrėžiame:

$$R = \bigcup_{n=1}^{N} S_n \tag{3.1}$$

kur $S_i \bigcap S_j = \emptyset$ kai $i \neq j$, N bendras sričių skaičius .

Vieni populiariausių bei plačiai paplitusių segmentavimo algoritmų yra slenksčio parinkimo algoritmai. Nisar Memon savo darbe [2] išbandė skirtingus slenksčio parinkimo algoritmus segmentuoti plaučių sritį. Darbe autoriui pavyko gauti gana tikslius rezultatus su tam tikrais duomenimis, bet pagrindinis darbo trukūmas, kad pasiūlytas sprendimas netinka visiems kompiuterinės tomografijos vaizdų atvejams.

Jiahui Wang [3] savo darbe pamėgino išskirti plaučių sritį kompiuterinės tomografijos vaizduose. Darbe buvo taikomos binarizavimo bei kraštų atpažinimo algoritmai. C. Karthikeyan, B. Ramadoss ir S. Baskar savo darbe [4] atliko plaučių srities segmentavimą naudojant morfologines operacijas bei dirbtinius neuroninius tinklus.

Vėliau Boykovce ir Jolly [5] pasiūlė metodą segmentuoti plaučių audinius. Autoriams pavyko gauti gana tikslius rezultatus, tačiau jų sprendimas reikalavo vartotojo įsikišimo. Vartotojas turėjo rankiniu būdu identifikuoti vidinius plaučių taškus vaizde, bei taškus esančius už jų. Identifikavus šiuos taškus, naudojant grafo pjūvio (angl. graph cut) algoritmą, buvo bandoma rasti optimaliausią pjūvį atskirti foną ir plaučių taškus.

4. Kaulų segmentavimas

Šiame skyrelyje bus aptartas kaulų segmentavimo procesas. Šis procesas skirtas apdoroti kompiuterinės tomografijos vaizdus tokiu būdu, kad rezultate būtų išskirti kauliniai audiniai. Segmentavimui atlikti bus naudojami binarizavimo (angl. Binarization) metodai. Pradžioje bus aptarti bendri šio metodo aspektai, o vėliau konkrečių algoritmų veikimo principai. Skyriaus pabaigoje bus atlikti segmentavimo eksperimentai naudojant turimus kompiuterinės tomografijos duomenis.

4.1. Binarizavimas

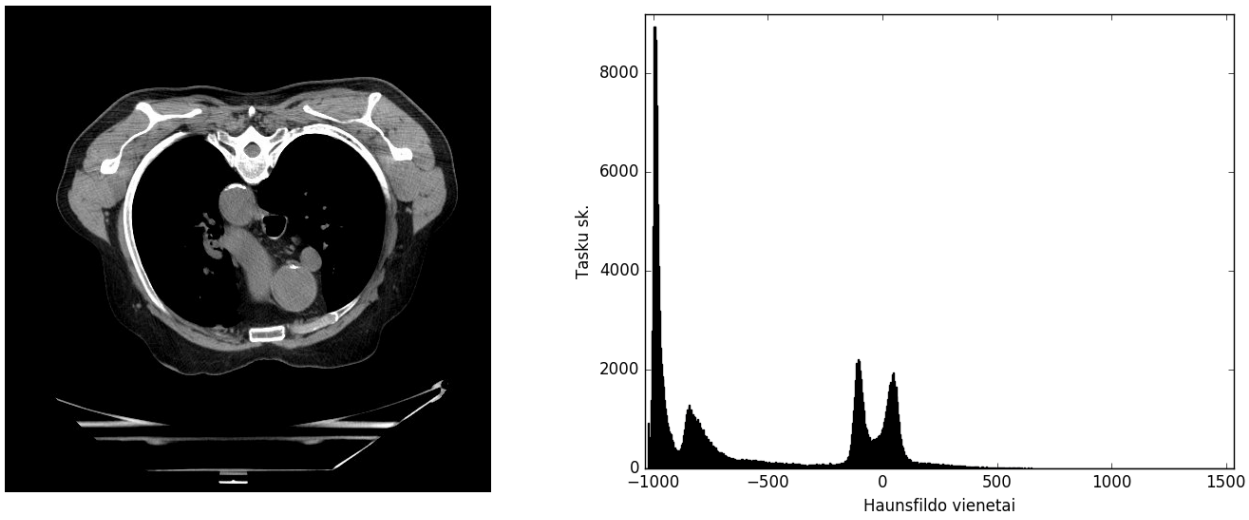
Binarizavimo metodas intuityviai suprantamas, tačiau, turi pakankamai platų panaudojimą vaizdų apdorojimo algoritmuose. Metodo esmė parinkti slenksčio reikšmę ir jos pagalba transformuoti vaizdą į dviejetainį (angl. binary) pavidalą. Binarizavimo procesą galime apibrėžti formulę 4.1.

$$g(x,y) = \begin{cases} 1, & \text{kai } f(x,y) >= T \\ 0, & \text{kitais atvejais} \end{cases} \tag{4.1}$$

Čia x, y yra taško koordinatės, T parinktas slenksčio dydis, $f(x,y)$ taško (x,y) reikšmė, $g(x,y)$ nauja taško (x,y) reikšmė.

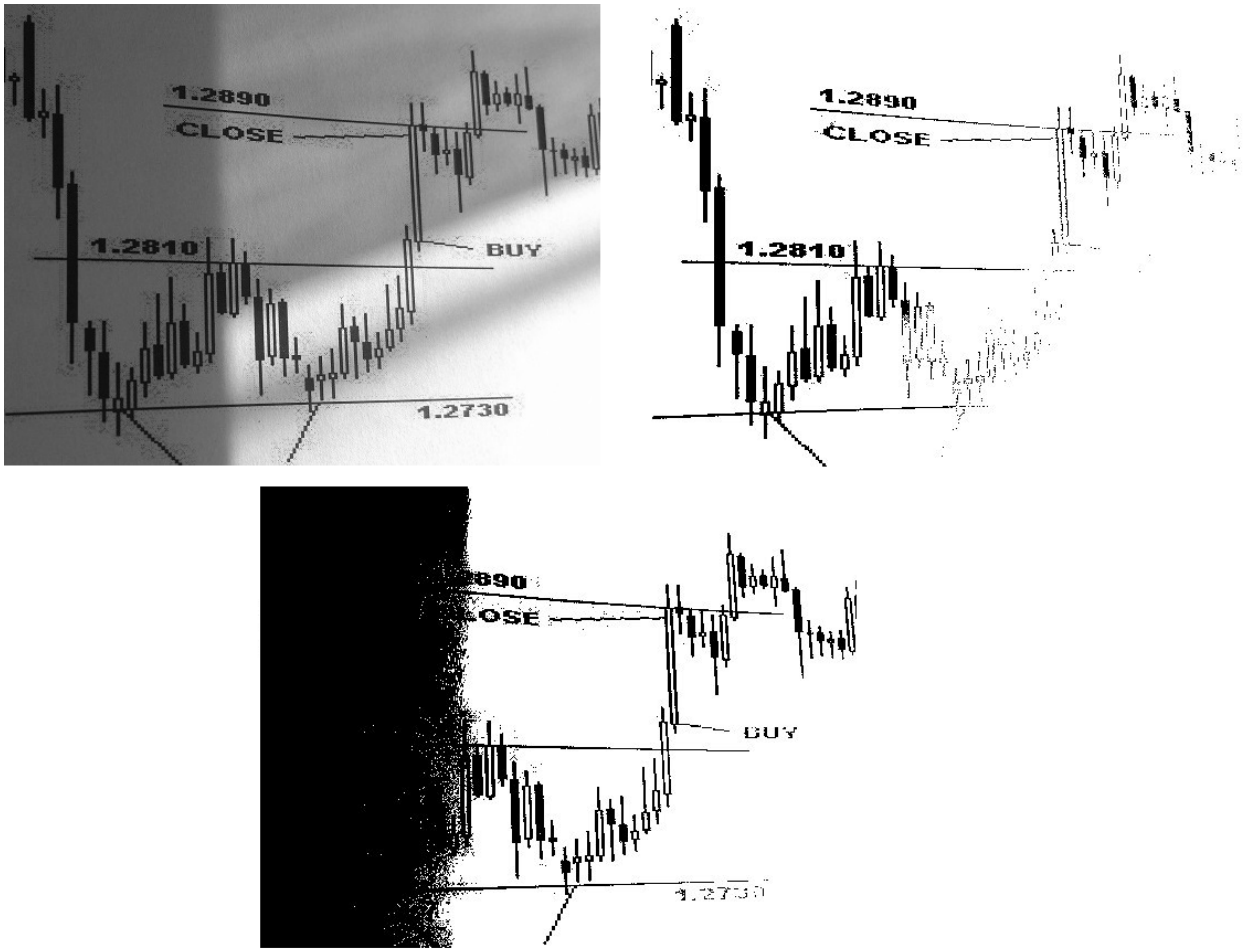
Paprasčiausias būdas parinkti slenkstį yra rankinis slenksčio dydžio parinkimas. Šis būdas yra tiek paprasčiausias supratimo prasme tiek greičiausias skaičiavimo atžvilgiu. Vis dėl to galime įžvelgti akivaizdų šio metodo trūkumą, būtinas vartotojo įsikišimas kadangi pats vartotojas nustato kokią slenksčio reikšmę naudoti.

Vieni dažniausiai naudojamų kompiuterinių algoritmų yra algoritmai skirti automatiškai parinkti optimalų slenksčio dydį be vartotojo įsikišimo. Tam dažnai yra atliekama vaizdų histogramų analizę. Vaizdo histograma tai tokia histograma, kurioje grafiškai vaizduojamas spalvų pasiskirstymas skaitmeniniame vaizde. 6 pav. galime matyti kompiuterinės tomografijos tyrimo vieno sluoksnio vaizdą bei jo histogramą. Histogramoje matyti pikai kurie atitinka įvairius organus bei audinius.



6 pav. Kompiuterinės tomografijos tyrimo vieno sluoksnio vaizdas kairėje. Šio vaizdo atspalvių histograma dešinėje.

Rankiniu būdu parinktas arba kompiuterinio algoritmo parinktas optimalus slenkstis yra globalus ir statinis, tai reiškią, kad kiekviename vaizdo regione naudojamas tas pats slenksčio dydis. Tai pasunkina atpažinimą kai skirtingi vaizdo regionai turi skirtingą apšvietimą. Paveikslelyję 7 galime matyti, kad skirtingose vaizdo regionuose vyrauja skirtingas apšvietimas. Tokiu atveju tikslus slenksčio parinkimas tampa beprasmiu, kadangi per mažas jo dydis nesugeba aptikti apšviestos dalies objektų, tuo tarpu per didelis jo dydis klaidingai atpažįsta neapšviestą vaizdo dalį kaip objektą tuo užgoždamas mus dominančius kitus objektus. Šiai problemai spręsti sukurti adaptyvus slenksčio parinkimo algoritmai, kurie apskaičiuoja slenksčio dydį įvertinant kaimyninių taškų vertes.



7 pav. Viršuje kairėje originalus vaizdas, viršuje dešnėje apdorotas vaizdas naudojant slenksčio dydį 50, apačioje centre apdorotas vaizdas naudojant slenksčio dydį 100.

Kaulų segmentavimą atliksim kiekviename sluoksnyje atskirai kadangi atstumai tarp sluoksnių kompiuterinės tomografijos tyrimuose gali siekti 3-5mm, tuo tarpu atstumai sluoksnio viduje sąntikinai maži iki 1mm.

4.2. Otsu algoritmas

Otsu algoritmas vienas populiariausių optimalaus slenksčio parinkimo algoritmų. Šio algoritmo pagalba apskaičiuojama slenksčio reikšmė stengiantis minimizuoti vidutinę segmentacijos paklaidą, t.y. vidutinę paklaidą priskiriant taškus į mus dominančią taškų klasę ir į fono taškų klasę. Vaizdo taškų reikšmes galime laikyti atsitiktinėmis reikšmėmis, o vaizdo histogramą kaip šių

taškų pasiskirstimo tankį. Jeigu tankis iš anksto yra žinomas galime parinkti optimalų slenksčio dydį.

Tarkime, kad L yra maksimalus šviesumo lygis vaizde. Tada galime apibrėžti normalizuotąją histogramą:

$$p_i = \frac{n_i}{N} \tag{4.2}$$

Čia N bendras taškų skaičius vaizde, n_i taškų skaičius su reikšme i, kur $i = 0..L$.

Algoritmas leis klasifikuoti taškus į dvi klases K_0 ir K_1. Galime apibrėžti tikimybes kiekvienai klasei:

$$P_{K_0} = \sum_{0}^{t} p_i \qquad P_{K_1} = \sum_{t+1}^{L} p_i \tag{4.3}$$

Papildomai apibrėžkime šių klasių vidurkius:

$$\mu_{K_0} = \frac{\sum_{i=0}^{t} ip_i}{P_{K_0}} \qquad \mu_{K_1} = \frac{\sum_{i=t+1}^{L} ip_i}{P_{K_1}} \tag{4.4}$$

Apibrėžkime tarpklasinę dispersiją:

$$\sigma_2^B = P_{K_0}(\mu_{K_0} - \mu_{K_1})^2 + P_{K_1}(\mu_{K_0} - \mu_{K_1})^2 \tag{4.5}$$

Tokiu atveju maksimizuodami šią tarpklasinę dispersiją galime apskaičiuoti optimalų slenksčio dydį t^*:

$$t^* = arg(\max_{0<t<L}(\sigma_2^B(t))) \tag{4.6}$$

Įvertinant visas formules galime apibrėžti algoritmo žingsnius:

1. Apskaičiuojame normaliąją histogramą, apibrežiame $\sigma_M = 0$ ir $t^* = 0$.
2. Su kiekvienu galimu $t = 0..L$:
   (a) Apskaičiuojame $\sigma_2^B(t)$.
   (b) Jei apskaičiuotas $\sigma_2^B(t)$ didesnis nei esamas σ_M priskiriam $\sigma_2^B(t)$ ir fiksuojame $t^* = t$.
3. Galutinis likęs t^* ir bus optimalus slenksčio dydis.

Atkreipkime dėmesį, kad šio metodo limitacija ta, kad vaizdas turi susidėti iš dviejų klasių taškų. Kompiuterinės tomografijos vaizduose klasių daugiau, tad šiuo atveju šis algoritmas veiks neteisingai. Šiai problemai išspręsti galime atlikti išankstinį vaizdų apdorojimą, tiksliau pašalinti taškus kurie kardinaliai skiriasi nuo mus dominančių kaulų taškų. Tam atfiltruosim vaizdus su filtru HU < 0, kadangi viskas iki 0 yra oras bei riebalai. Tokiu atveju gausim vaizdą, kurio histogramoje liks tik du pikai.

4.3. Niblako algoritmas

Niblako algoritmas remiasi lokalaus slenksčio parinkimo sąvoka. Šiuo atveju binarizavimo procesą apibrežiame pakoregavę 4.1 formulę sekančiu būdu:

$$g(x,y) = \begin{cases} 1, & \text{kai } f(x,y) >= T(x,y) \\ 0, & \text{kitais atvejais} \end{cases} \tag{4.7}$$

Kaip matome vietoje globalios slenksčio reikšmės T dabar naudosime $T(x,y)$ reikšmę kuri priklauso nuo taškų (x,y). Šią lokalią slenksčio reikšmę mes apskaičiuosime kiekvienam taškui atskirai. Pastebėkime, kad šis slenksčio parinkimo būdas reikalauja didesnių skaičiavimo resursų, kas savo ruoštu sulėtins apskaičiavimo spartą.

Niblako algoritmas apskaičiuoja lokalią $T(x,y)$ slenksčio reikšmę taške (x,y) vertinant kaimyninių taškų reikšmes. Kaimynai nustatomi naudojant $w * w$ matmenų langą. Apibrėžkime $T(x,y)$:

$$T(x,y) = m(x,y) + k * \sigma_(x,y) \tag{4.8}$$

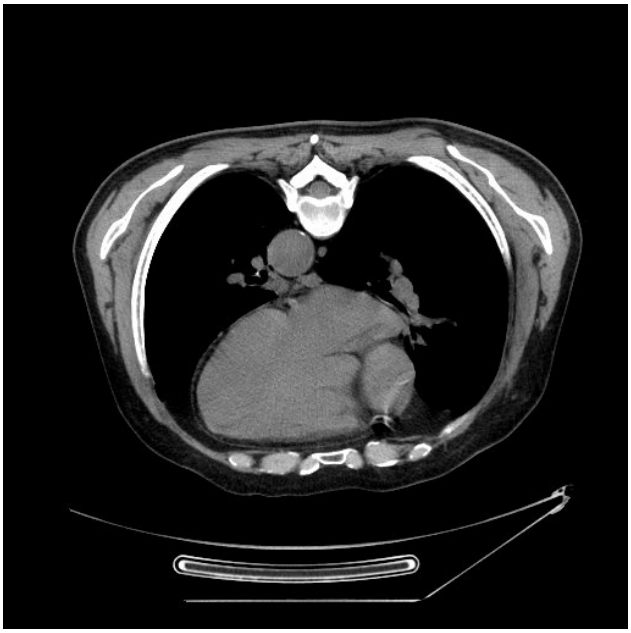
Kur $m(x,y)$ vidurkis, o $\sigma_(x,y)$ standartinis nuokrypis visų kaimyninių taškų reikšmių, k numatytoji algoritmo konstanta, bendru atveju lygi 0.2.

Galime apibrėžti algoritmo žingsnius:

1. Su kiekvienu (x,y) vaizde:
   (a) Įvertinus visų kaimynų lange $w * w$ reikšmes apskaičiuojami $m(x,y)$ ir $\sigma_(x,y)$.
   (b) Apskaičiuojama lokali slenksčio reikšmė $T(x,y)$.

4.4. Eksperimentai

Šiame skyriuje bus atliekami aprašytų algoritmų vertinimai. Šiam etapui buvo pasirinktas kompiuterinės tomografijos vaizdas kuris pavaizduotas 8 pav..



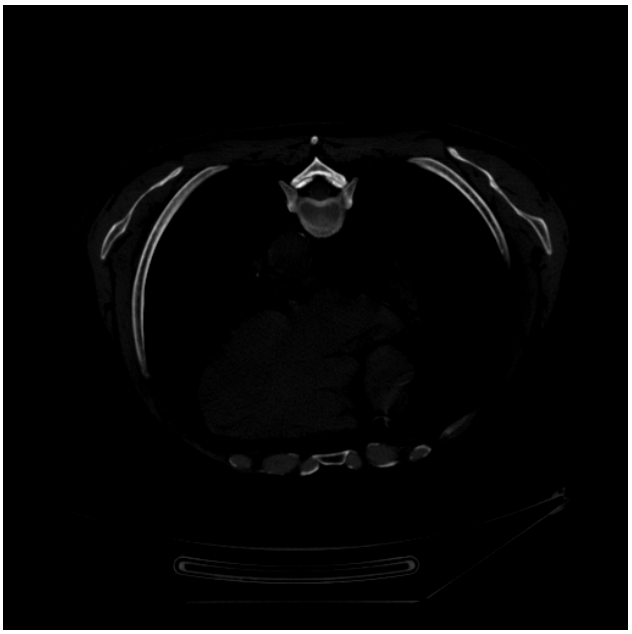
8 pav. Tyrimui naudotas kompiuterinės tomografijos sluoksnio vaizdas.

Kadangi Otsu algoritmas reikalauja, kad vaizdo histograma susidėtų iš dviejų spalvinių pikų mes papildomai apdorosim vaizdą. Nagrinėjamas vaizdas turi atspalvių spektrą nuo -1000HU iki 1369HU. Žinodami, kad kaulai turi aukštas HU reikšmes, mes atfiltruojame vaizdą ir paliekame visus taškus kurių reikšmė > 0HU. Visiems kitiems taškams < 0HU priskiriam 0HU reikšmę. Papildomai vaizdas buvo transformuotas į 256 pilkumo atspalvių vaizdą, kad pagerinti greitaveiką bei gauti galimybę išsaugoti jį kaip 8bit vaizdą. Transformavimo procesą galime apibrėžti taip:

$$g(x,y) = \frac{255 * f(x,y)}{fMax} \tag{4.9}$$

Čia g(x,y) nauja taško (x,y) reikšmė, f(x,y) pradinė taško (x,y) reikšmė, o fMax maksimali visų taškų reikšmė vaizde. Atkrepikime dėmesį į tai, kad čia pateikta supaprastinta transformacijos formulė, kadangi minimali taško reikšmė vaizde po pirminės transformacijos yra 0, jos galime netraukti į formulę. Vaizdą po transformacijos galime matyti 9 pav..

Otsu algoritmo įgyvendinimui buvo panaudotas laisvai prieinamas CV2 programinis įrankis Python platformoje.

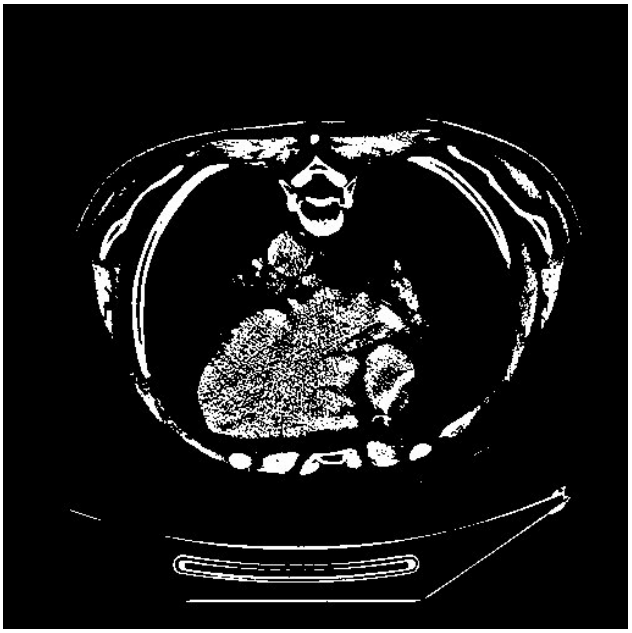


9 pav. Atfiltruotas kompiuterinės tomografijos vaizdas paliekant tik > 0HU taškus.

Atlikus Otsu algoritmo binarizavimo procesą gavome binarizuotą vaizdą kurį galime matyti 10 pav.. Matome, kad ryškūs kaulai atsiskyrė tvarkingai. Papildomai buvo segmentuotos išorinės detalės, kurios nėra kaulų dalys.

10 pav. Otsu algoritmo binarizuotas vaizdas.

Antras algorimtas kuris buvo ištirtas, tai Niblako algoritmas. Niblako algoritmo tyrimui buvo panaudotas ImageJ, lasivai prieinamas įrankis, kuris turi įgyvendintą Niblako algoritmą.



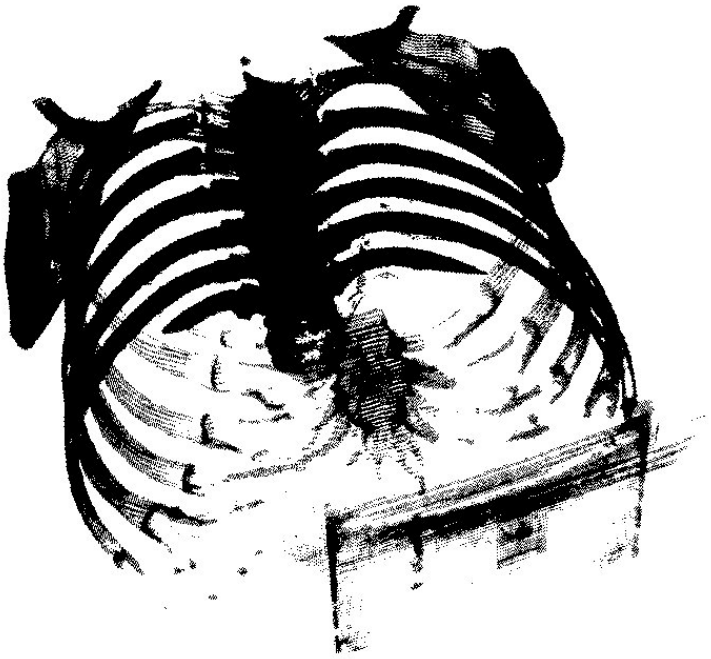
11 pav. Niblako algoritmo binarizuotas vaizdas.

Atlikus Niblako algoritmo binarizavimo procesą gavome binarizuotą vaizdą kurį galime matyti 11 pav.. Matome, kad praktiškai visos smulkios detalės buvo išryškintos, todėl nebeįmanoma atskirti kaulinių audinių. Rezultatas logiškas, nes Niblako algoritmas yra lokalaus slenksčio parinkimo algoritmo tipas.

4.5. 3D vizualizavimas

3D vizualizavimo procesui atlikti naudosime Python programavimo kalbą bei grafinį OpenGL variklį. Sukonstruosime trimatį masyvą taškų kurie sudarys krūtinės ląstos kaulų struktūrą. Iš-

rinkę Otsu binarizavimo algoritmą jį galime taikyti visiems kompiuterinės tomografijos tyrimo sluoksniams. Atlikus šį procesą gausime visų sluoksnių binarizuotus vaizdus, o tai atitinką dvimatį masyvą su galimomis reikšmėmis 1 ir 0. Atlikus visų sluoksnių masyvų perranką išrinksime tuos taškus kurie turi reikšmę 1 ir tokius taškus dėsime į konstruojamą trimatį masyvą 3D vizualizavimui su taškais x ir y. Trečioji masyvo reikšmė bus sluoksnio eilės numeris. Surinkus visus taškus į trimatį masyvą papildomai būtina įvertinti atstumus tarp taškų. Šiuos atstumus gausime iš DICOM failuose nurodytų atitinkamų laukelių reikšmių.



12 pav. Rekonstruotas 3D krūtinės ląstos vaizdas.

5. Plaučių segmentavimo metodai

Šiame skyrelyje bus aprašomas plaučių segmentavimo procesas. Bus aptarti pagrindiniai algoritmai, jų ypatumai. Veliau bus pateiktas jų realizavimo principas, bei gauti rezultatai. Šiame etape bus panaudoti rezultatai gauti 4 skyriuje. Rekonstruotų kaulų taškai bus eliminuoti iš bendros tyrimo vaizdų taškų aibės. Tai pagreitins bei supaprastins galutinio sprendimo veikimą.

Vaizdų segmentavimo procesui atlikti egzistuoja daug skirtingų būdų ir algoritmų. Vieni algoritmai grąžina segmentuotų objektų ribas, tokių algoritmų pavyzdys galėtų būti kraštų radimo (angl. edge detection) algoritmai. Kitų algoritmų rezultatas gali būti taškų aibė, kurioje kiekvienas taškas gali priklausyti skirtingiems segmentams (grupėms) su apskaičiuotais priklausomybės koeficientais kiekvienai grupei. Šiame darbe bus realizuotas pilnai automatinis sprendimas, kuris segmentuoja plaučių regioną kompiuterinės tomografijos tyrimų vaizduose.

5.1. Sujungtų komponenčių algoritmas

Sujungtų komponenčių (angl. image labeling) yra rekursyvus algoritmas, kuris sujungia vaizde esančius taškus į didesnes grupes. Grupavimui atlikti, iš pradžių nustatomos sąlygos kurias turi tenkinti taškai, kad juos galima būtų priskirti prie konkrečios grupės. Tokių sąlygų pavyzdys vaizdų segmentavime dažnai yra taško intensyvumo panašumas į grupės vidurkį. Tokią sąlygą galime apibrėžti sekančiai:

$$|g(t) - mean(G)| < \varepsilon \tag{5.1}$$

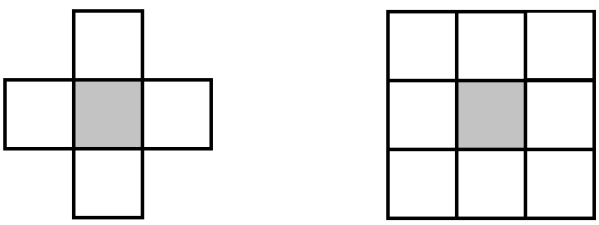
Kur $g(t)$ taško t intensyvumas, $mean(G)$ vidutinis grupės G intensyvumas, ε iš anksto parinkta konstanta. Jei taškas t tenkina šią sąlygą, jį priskiriam grupei G.

Taipogi dažnai galime sutikti sąlygą kai lyginamos kaimyninių taškų vertės. Tokią sąlygą matome 5.2.

$$|g(t) - g(t_k)| < \varepsilon \tag{5.2}$$

Kur $g(t)$ taško t intensyvumas, kuris jau priskirtas grupei G, $g(t_k)$ kaimyninio taško intensyvumas, ε iš anksto parinkta konstanta. Jei taškas t_k tenkina šią sąlygą, jį priskiriam grupei G.

Algoritmo pradžioje yra parenkamas pradinis taškas vaizde. Kadangi šis taškas yra pradinis, automatiškai jis yra priskiriamas į naująją grupę. Paskui yra nusprendžiama kaip bus ieškoma kitų taškų. 2D atvejuose dažniausiai yra ieškoma 4-kaimynų arba 8-kaimynų aplinkose. 4-kaimynų atvejyje ieškomi taškai virš pradinio taško, apačioje, kairėje ir dešinėje. 8-kaimynų atveju papildomai ieškomi taškai besiliečiantys kampuose. Kitaip tariant, 4-kaimynų atveju ieškomi kaimynai turintys bendrą kraštinę su pradiniu tašku, o 8-kaimynų atveju turinčių bendrą viršūnę.

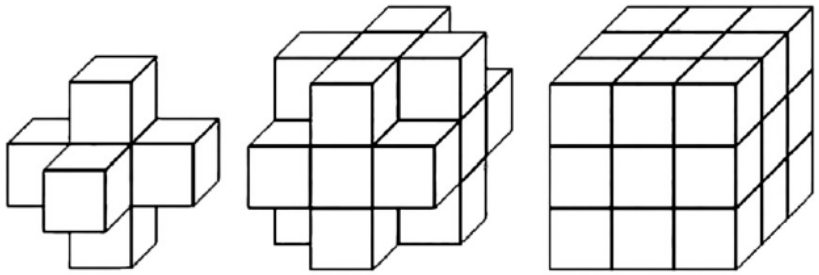


13 pav. Kairėje 4-kaimynų atvejis, dešinėje 8-kaimynų.

Norėdami taikyti šį algoritmą 3D vaizdų sekoms apdoroti, privalome apibrėžti kaimyninių taškų sritis 3D aplinkoje. Galime išskirti tris atvejus:

1. 6-kaimynų aplinka. Ši aplinka panaši 2D atveju aprašytai 4-kaimynų sričiai, papildomai pridedant trečiąją dimensiją. Ieškomi kaimynai turintys bendrą sieną.
2. 18-kaimynų aplinka. Ieškoma kaimynų turinčių bendrą kraštinę.
3. 26-kaimybų aplinka. Ieškome visų kitų taškų turinčių bent vieną bendrą viršūnę.

Šie atvejai iliustruoti 14 pav. .

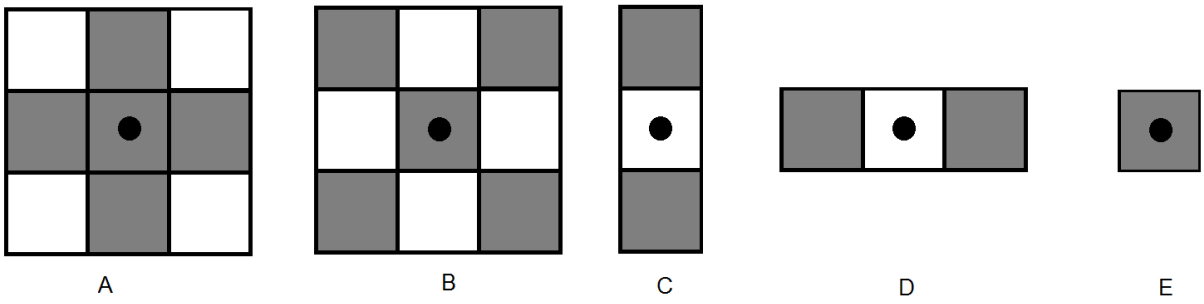


14 pav. Kairėje 6-kaimynų atvejis, viduryje 18-kaimynų, dešinėje 26-kaimynų.

5.2. Matematinės morfologijos operacijos

Morfologija moksline prasme traktuojama kaip mokslas apie formas ir struktūras. Vaizdų apdorojime morfologija siejama su geometrinių struktūrų analize, kurios būdingos vaizdams. Matematinė morfologija buvo pasiūlyta Matheron ir Serra, kurie darbe [1] tyrė binarinius vaizdus. Jų pasiūlytas sprendimas šiais laikais plačiai naudojamas vaizdų apdorojimo algoritmuose. Morfologinis filtravimas dažnai naudojamas artefaktų, esančių vaizduose, segmentavimui, kraštų atpažinimui, triukšmo šalinimui ir t.t. . Šiame skyrelyje bus aprašytos esminės matematinės morfologijos operacijos bei jų taikymo būdai vaizdų apdorojime.

Matematinė morfologija grįsta aibių teorija ir originale buvo taikoma binariniams vaizdams. Vaizdai traktuojami kaip taškų aibės su kuriomis yra atliekamos tam tikros operacijos su struktūriniu elementu. Struktūrinis elementas (literatūroje taip pat vadinamas struktūrizavimo matrica) yra pakankamai mažas, lyginant su pačiu vaizdu, elementas sudarytas iš užpildytų ir neužpildytų taškų. Struktūrinio elemento pavyzdžius galime matyti 15 pav.. Atkreipkime dėmesį į tai, kad 15 pav. struktūrinis elementas E neturi praktinės prasmės, tačiau jį patogu naudoti dėstant matematinės morfologijos operacijų veikimo principus.



15 pav. Struktūrinių elementų pavyzdžiai, kur tamsūs kvadratai atitinka elemento taškus priklausančius jam, balti atitinka nepriklausančius. Taškas su apskritimu viduje žymi elemento pradžios tašką.

Matematinės morfologijos esminė operacija yra struktūrinio elemento pritaikymas visose vaizdo taškuose. Pritaikant struktūrinį elementą kiekvienam taškui yra atliekamos aibių jungimo bei sankirtos operacijos. Labai svarbu tinkamai pasirinkti struktūrinio elemento pavidalą, kadangi jis tiesiogiai lemia atliktų operacijų rezultatą.

Dažniausiai yra išskiriamos keturios pagrindinės matematinės morfologijos operacijos: išplėtimo, ėsdinimo, atidarymo bei uždarymo operacijos.

5.2.1. Morfologinis ėsdinimas

Viena paprasčiausių matematinės morfologijos operacijų yra ėsdinimo operacija. Tarkime, kad turime taškų aibę I (binarinis vaizdas arba jo dalis) ir struktūrinį elementą S. Kai struktūrinis elementas yra pritaikomas konkrečiam taškui (x, y) mes tai žymėsime $S_{(x,y)}$. Tada ėsdinimo operacijos pritaikymo aibei I su struktūriniu elementu S rezultatas yra taškų aibė iš I, kuriuose transformuotas struktūrinis elementas S yra aibės I poaibis. Apibrėžiame:

$$I \ominus S = \left\{ (x, y) : S_(x, y) \subset I \right\} \tag{5.3}$$

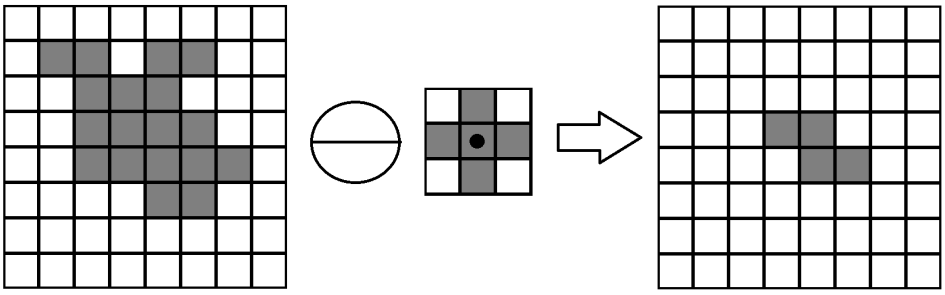
Kur I vaizdo taškų aibė, S struktūrinis elementas, $S_{(x,y)}$ transformuotas S elementas taške (x, y) 5.4.

$$S_{(x,y)} = \{ s + (x, y) : s \in S \} \tag{5.4}$$

Intuityviai apibrėžkime ėsdinimo operaciją:

1. Kiekviename vaizdo taške (i, j) "įstatome" struktūrinį elementą S.
2. Jeigu visi struktūrinio elemento S taškai patenka į vaizdo srities taškų aibę tada traktuojame, kad tas taškas (i, j) patenka į rezultato aibę.

Šios operacijos veikimo pavyzdį galime matyti 16 pav..



16 pav. Kairėje pradinis vaizdas, centre struktūrinis elementas su pradžios tašku viduryje, dešinėje ėsdinimo operacijos rezultatas.

Pastebėkime, kadangi šita operacija reikalauja, kad visi struktūrinio elemento taškai patektų į vaizdo aibę. Tai yra griežtas reikalavimas, todėl ėsdinimo operacijos rezultato taškų aibė yra mažesnė už pradinio vaizdo taškų aibę. Dažniausiai prarandami taškai aibės kraštuose ir taškai šalia skylių, kadangi jų aplinkose yra trūkstamų taškų. Atkreipkime dėmesį, kad jeigu struktūrinis elementas būtų sudarytas tik iš vieno užpildyto taško (15 pav.elementas E), tai tokiu atveju ėsdinimo operacijos rezultato taškų aibė būtų lygi pradinio vaizdo taškų aibei.

5.2.2. Morfologinis išplėtimas

Kita dažnai naudojama operacija yra morfologinio išplėtimo operacija. Išplėtimo operacijos pritaikymo aibei I su struktūriniu elementu S rezultatas yra visų transformuotų struktūrinių elementų sąjunga visuose taškuose iš I kuriuose transformuotas struktūrinis elementas S kertasi su I. Apibrėžiame:

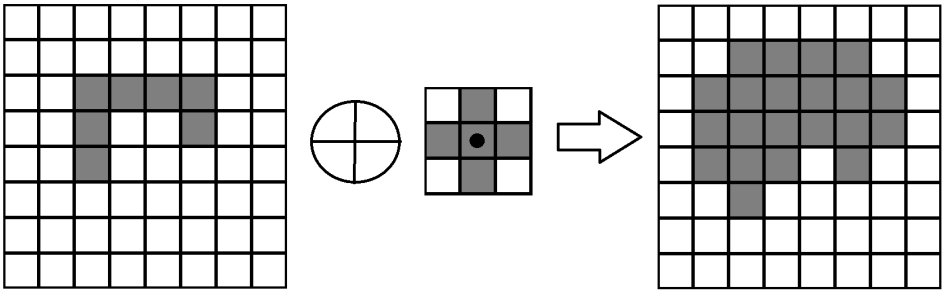
$$I \oplus S = \left\{ (x,y) : S_(x,y) \cap I \neq \varnothing \right\} \tag{5.5}$$

Kur I vaizdo taškų aibė, S struktūrinis elements, $S_{(x,y)}$ transformuotas S elementas taške (x,y) 5.4.

Pabandykime paaiškinti 5.5 paprasčiau:

1. Kiekviename vaizdo taške (i,j) "įstatome" struktūrinį elementą S.
2. Jeigu struktūrinio elemento S pradžios taškas priklauso I tada traktuojame, kad S taškai patenka į rezultatų aibę.

Šios operacijos veikimo pavyzdį galime matyti 17 pav..



17 pav. Kairėje pradinis vaizdas, centre struktūrinis elementas su pradžios tašku viduryje, dešinėje išplėtimo operacijos rezultatas.

Matome, kad skirtingai nei ėsdinimo operacija, išplėtimo operacija padidiną vaizdo taškų aibę. Nesunku pastebėti kodėl, nes išplėtimo operacijai nekeliamas reikalavimas, kad visi struktūrinio elemento taškai patektų į vaizdo srities aibę. Atvirkščiai, jei struktūrinio elemento pradžios taškas patenka į vaizdo srities aibę, tai ir visi kiti taškai pateks į rezultato aibę.

5.2.3. Morfologinis atidarymas

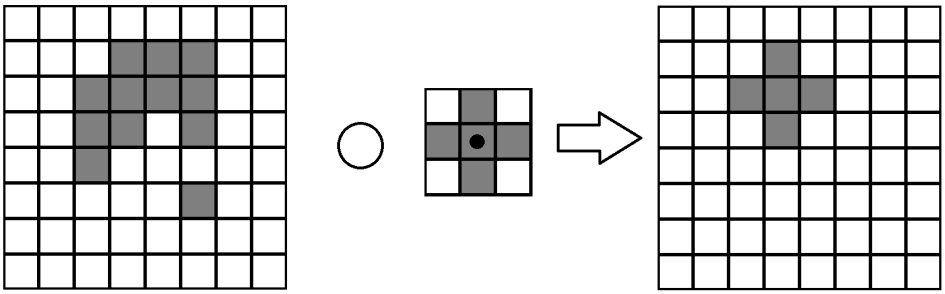
Morfologinio atidarymo operacija apibrėžiama kaip dviejų jau aptartų operacijų sąjunga. Tiksliau tariant morfologinis atidarymas yra ėsdinimo operacijos rezultatas papildomai pritaikius išplėtimo operaciją. Svarbu paminėti, kad atliekant ėsdinimo bei išplėtimo operacijas, būtina naudoti tokį pat struktūrinį elementą. Apibrėžkime atidarymo operaciją sekančiai:

$$I \circ S = (I \ominus S) \oplus S \tag{5.6}$$

Kur I vaizdo taškų aibė, S struktūrinis elementas.

Nesunku nuspėti, kad šios operacijos rezultatas bus panašus į anksčiau aptartų ėsdinimo bei išplėtimo operacijų rezultatus. Prisiminkime, kad ėsdinimo operacija sutraukia vaizdą, tuo tarpu išplėtimo operacija jį išplečia. Tada atliekant morfologinio atidarymo operaciją vaizdas iš pradžių bus sutrauktas, o po to išplėstas. Galime pastebėti, kad vaizde esantys pakankamai maži objektai po ėsdinimo operacijos bus eliminuoti, o po to sekanti išplėtimo operacija jau nebegalės atkurti pašalintų objektų. Šios savybės dėka atidarymo operacija yra naudinga triukšmo šalinimo problemoms spręsti.

18 pav. pateiktas šios operacijos veikimo pavyzdys.



18 pav. Kairėje pradinis vaizdas, centre struktūrinis elementas su pradžios tašku viduryje, dešinėje atidarymo operacijos rezultatas.

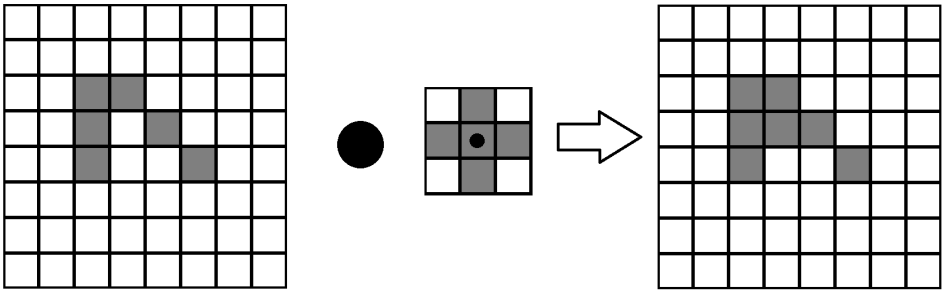
5.2.4. Morfologinis uždarymas

Morfologinio uždarymo operacija panaši į morfologinio atidarymo operaciją. Vienintelis skirtumas tame, kad skirtingai nuo atidarymo operacijos, uždarymo operacijos metu iš pradžių yra atliekamas morfologinis išplėtimas, o po to su gauta taškų aibe atliekamas ėsdinimas. Kaip ir morfologinio atidarymo atveju, svarbu, kad tiek išplėtimo tiek ėsdinimo operacijos būtų atliekamos su tuo pačiu struktūriniu elementu. Uždarymo operacija apibrėžiama sekančiai:

$$I \bullet S = (I \oplus S) \ominus S \tag{5.7}$$

Kur I vaizdo taškų aibė, S struktūrinis elementas.

Morfologinio uždarymo veikimo pavyzdys pateiktas pav. .



19 pav. Kairėje pradinis vaizdas, centre struktūrinis elementas su pradžios tašku viduryje, dešinėje uždarymo operacijos rezultatas.

Stebėdami 19 pav. gautus rezultatus matome, kad gautas rezultatas panašus į pradinį vaizdą. Tik vienas papildomas taškas atsirado po uždarymo operacijos. Įvertinę uždarymo operacijos veikimo principą, suprantame, kad iš pradžių vaizdas buvo išplėstas, paskui sutrauktas. Pradiniame vaizde galime matyti "įlankos" formos taškų išsidėstymą. Atliekant išplėtimo operaciją įlanka buvo "užlieta" iš trijų pusių ir susiformavo pakankamai didelis taškų plotas, kurio ėsdinimo operacija nebesugebėjo sutraukti. Ši savybė naudinga vaizdų apdorojimo sprendimuose, kadangi padeda apjungti skyles, įlankas bei siaurus tarpeklius vaizduose.

5.3. Plaučių segmentavimo sprendimas

5.3.1. DICOM duomenų skaitymas

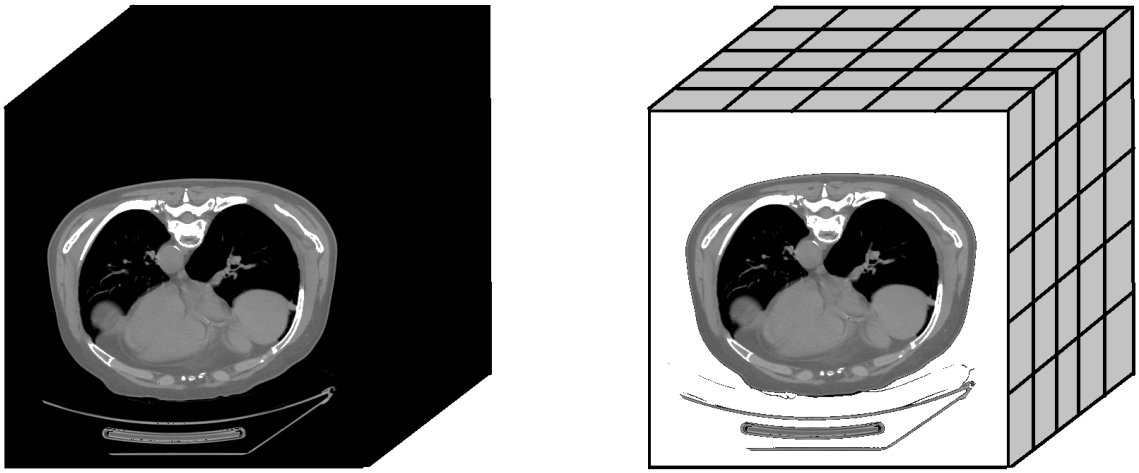
Prieš atliekant plaučių segmentaciją būtina atlikti visų vieno tyrimo nuotraukų perkėlimą į sistemą. Tam buvo įgyvendinta atskira Java klasė skirta duomenų skaitymui bei jų išsaugojimui duomenų struktūrose. Algoritmo įgyvendinimo esmė yra DICOM vaizdų skaitymas bei vaizdų taškų išsaugojimas trimačiame masyve. Kadangi visi vieno tyrimo vaizdai turi vienodą rezoliuciją mes galime apibrėžti du trimačio masyvo dydžius, vadinsime masyvo pločiu ir ilgiu. Trečiasis masyvo dydis lygus tyrimo nuotraukų skaičiui, ji vadinsime masyvo aukščiu. Algoritmo pseudo kodą galime matyti žemiau:

```
tyrimoSluoksniuTaskuMasyvas = new Integer[
    tyrimoSluoksniuSkaicius][rezoliucijaX][rezoliucijaY]
for sluoksnis in visiTyrimoSluoksniai do
begin
sluoksnioSekosNr = gautiSluoksnioSekosNr(sluoksnis)
tyrimoSluoksniuTaskuMasyvas[sluoksnioSekosNr] =
    gautiSluoksnioTaskus(sluoksnis)
end;
```

5.3.2. Išorės taškų šalinimas

Pirmasis sprendimo žingsnis yra išorės taškų šalinimo procedūra. Tai sumažins likusių taškų aibės dydį, kas savo ruoštu leis mums koncentruotis ties vidiniais žmogaus kūno taškais. Išorės

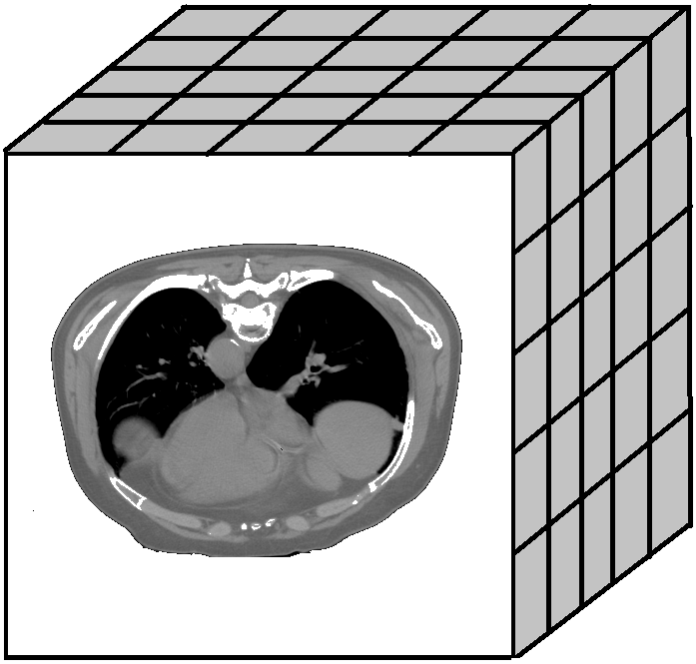
taškų šalinimui bus naudojamas užpylimo algoritmas. Patogumo dėlei pradinis taškas bus parenkamas viršutiniame kairiajame kampe, kadangi jis visada yra išorės taškas. Kadangi išorės HU įvertis artimas -1000, žmogaus odos įvertis artimas -200. Pasirenkame ribinį dydį -200 ir vykdome užpylimo algoritmą. Šio žingsnio rezultatą galime matyti 20 pav. .



20 pav. Kairėje sluoksnių pjūvio vaizdas eliminavus kaulų taškus, dešinėje vaizdas pašalinus fono taškus.

5.3.3. Pašalinių artefaktų taškų šalinimas

Pašalinus išorės taškus matome, kad vaizde liko žmogaus kūnas bei kiti artefaktai (kompiuterinio tomografo dalys, stalo paviršius). Būtina pašalinti taškus, kurie nepriklauso žmogaus kūnui. Šiai užduočiai atlikti pasinaudosime sujungtų komponenčių algoritmu, kuris savo ruoštu išrinks visus vaizde esančius sujungtus objektus. Turėdami šiuos objektus mes paliksime didžiausią objektą, nes žmogaus kūnas užima didžiausią vaizdo dalį. Naudosime supaprastintą sujungtų komponenčių algoritmą, t.y. trauksime į grupes visus kaimyninius taškus. Pašalinus fono taškus, tarp išorės artefaktų ir žmogaus kūno nebeliko taškų, atitinkamai nebeliko ir galimų kaimynų. Atlikus šį žingsnį pasiliekame didžiausią, pagal taškų kiekį, grupę sekantiems sprendimo žingsniams. Šio žingsnio rezultatą galime matyti 21 pav. .



21 pav. Vaizdas eliminavus pašalinius artefaktus.

5.3.4. Plaučių srities taškų radimas

Turėdami vaizde tik žmogaus kūno taškus, galime pradėti plaučių srities taškų segmentavimą. Tam pasinaudosime sujungtų komponenčių algoritmu. Atstumo funkciją apibrėšime kaip verčių skirtumas tarp taškų įvertinant atstumą tarp jų. Segmentuojant bus naudojamas 14-kaimynų aplinkos atvejis, 14 pav. antrasis atvejis.

$$|g(t) - g(t_k)| < \varepsilon * s(t, t_k) \tag{5.8}$$

Kur $g(t)$ taško t intensyvumas, $g(t_k)$ kaimyninio taško intensyvumas, ε iš anksto parinkta konstanta, $s(t, t_k)$ atstumas tarp taškų t ir t_k (5.9).

$$s(t, t_k) = \sqrt{\sum_{i=0}^{2} (t_i - t_{k_i})^2} \tag{5.9}$$

Čia t_i taško t koordinatė dimensijoje i, t_{k_i} taško t_k koordinatė dimensijoje i.

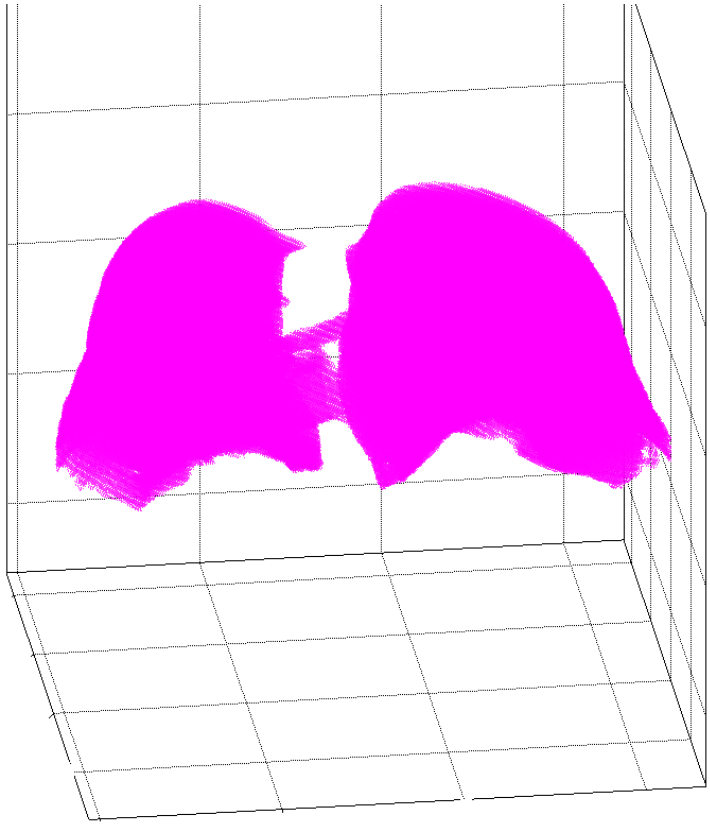
Jei taškas t_k tenkina (5.8) sąlygą priskiriame jį taško t grupei.

Išrinkus visus sujungtus objektus, juos galime išrūšiuoti mažėjimo ar didėjo tvarka pagal bendrą taškų skaičių grupėse. Taipogi turint taškus ir jų įverčius galime apskaičiuoti kiekvienos grupės taškų verčių vidurkį. Žemiau pateikta lentelė, kurioje galime matyti šiame etape išskirtus sujungtus objektus. Lentelėje rodomos tik didžiausios grupės, tiksliau tik tos grupės, kuriuose taškų skaičius didesnis nei 10000 taškų.

Taškų skaičius grupėje	Vidutinė grupės taškų vertė
676784	-69.8726
52599	42.0270
151021	48.0663
239637	33.3973
1477268	-756.9783

Žvelgdami į lentelę galime pastebėti, kad išsiskiria viena grupė, kurios vidutinė grupės taškų vertė yra stipriai neigiama, apytiksliai -757. Taipogi pastebime, kad ši grupė yra didžiausia. Žinodami, kad plaučiai yra vienas didžiausių žmogaus organų bei tai, kad plaučių sugerties koeficientas yra neigiamas, galime teigti, kad būtent grupė kurios vidutinė vertė mažiausia ir yra plaučių ertmė. Prisiminkime, kad neigiamą taškų reikšmę taipogi turi išorės taškai nepriklausantys žmogaus kūnui. Bet kadangi išorės taškai jau buvo eliminuoti pirmajame sprendimo žingsnyje, čia jie nebepatenka.

Palikę vaizde tik plaučių ertmės taškus, gausime segmentuotą plaučių sritį. Šio žingsnio rezultatą matome 22 pav..



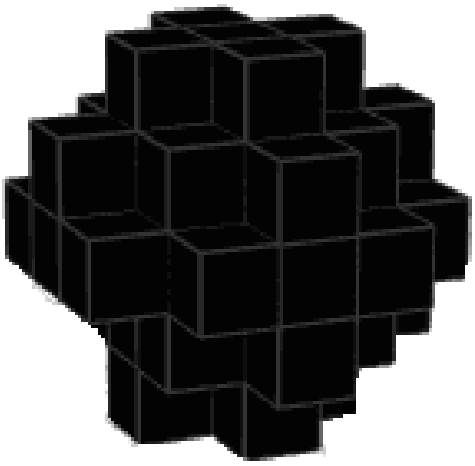
22 pav. Segmentuota plaučių taškų aibė.

5.3.5. Galutinis glotninimas, skylių šalinimas

Atlikus plaučių srities taškų segmentaciją, būtina atlikti galutinį glotninimą, pašalinti smulkias skyles, bei įtrūkimus. Šiai užduočiai atlikti pasinaudosime matematinės morfologijos operacijomis. Skyrelyje 5.2 aptarėme pagrindinius matematinės morfologijos principus, bei dažniausiai

naudojamas operacijas. Prisiminkime, kad morfologinio uždarymo operacija tinka siaurų tarpeklių, įtrūkimų bei smulkių skylių šalinimui. Tuo tarpu morfologinio atidarymo operacija padeda pašalinti smulkius objektus, darinius. Taipogi jos pagalba glotninami kraštai.

Realizuojamam sprendimui buvo pasirinktos matematinės morfologijos atidarymo bei uždarymo operacijos. Šios operacijos bus atliekamos trijose dimensijose naudojant struktūrinį elementą taipogi sudarytą iš trijų dimensijų taškų. Buvo pasirinktas 5x5 išmatavimų struktūrinis elementas, kurio užpildyti elementai suformuoja sferos pavidalo formą. Šio struktūrinio elemento pradžios taškas sferos centras. Šį elementą galime matyti 23 pav. .

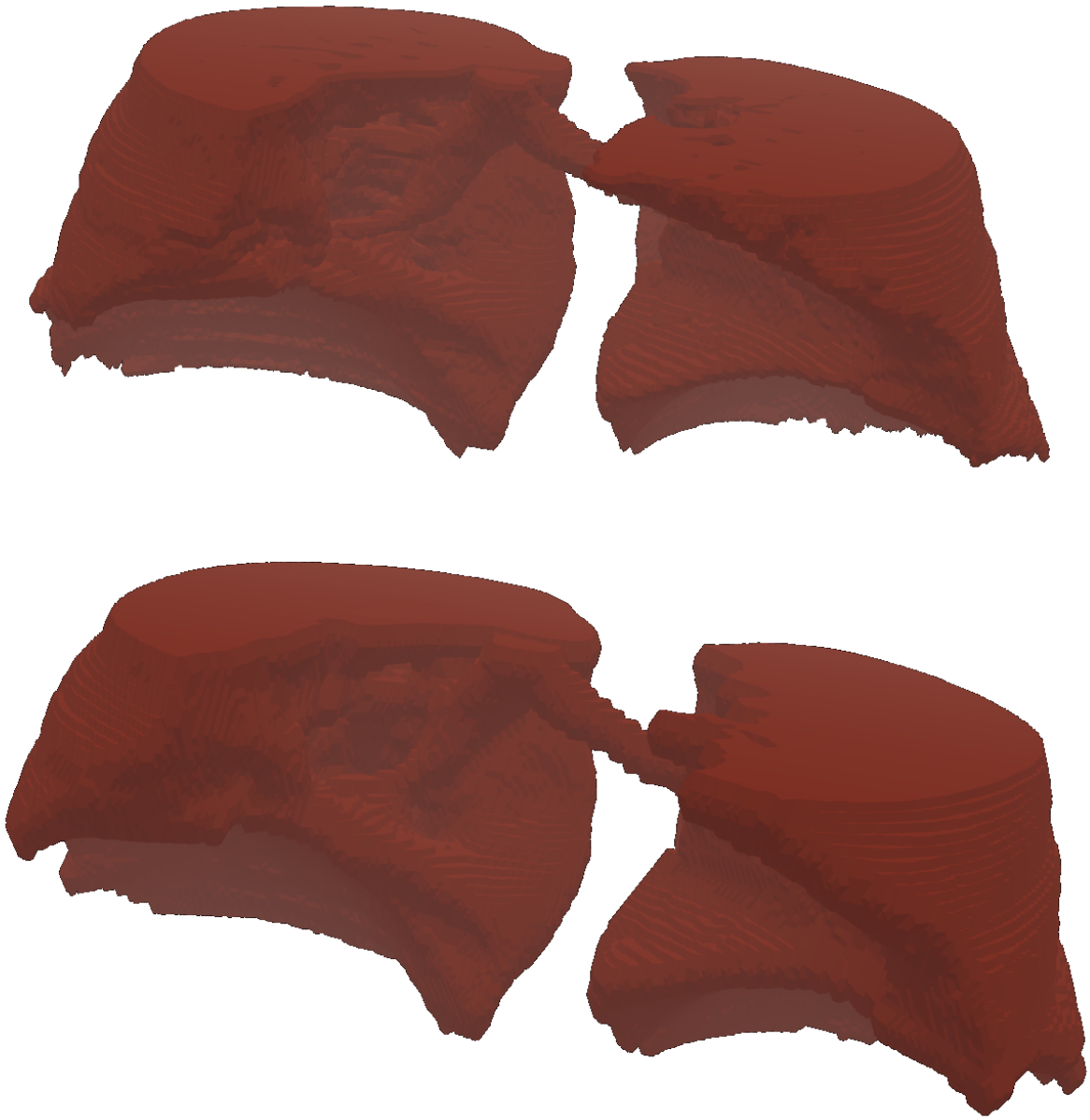


23 pav. Struktūrinis elementas sferos pavidalu.

Matematinės morfologjios operacijos buvo įgyvendintos Java programavimo aplinkoje kaip pagalbinis modulis pagrindinėje sprendimo sistemoje. Sukurtas sprendimas leidžia atlikti ėsdinimo, išplėtimo, atidarymo bei uždarymo operacijas. Visos šios operacijos prieinamos tiek 2D tiek 3D atvejuose. Taipogi buvo sukurta patogi aplikacijų programavimo sąsaja (angl. API) leidžianti panaudoti realizuotą sprendimą kitose sistemose:

```
MorphologyOperation.dilate(taskuAibe, StructureElement.
    get3dRBall5x5());
MorphologyOperation.erode(taskuAibe, StructureElement.
    get3dCross3x3());
MorphologyOperation.open(taskuAibe, StructureElement.
    get3dRectangle3x3());
MorphologyOperation.close(taskuAibe, StructureElement.
    get3dCross3x3());
```

Iš pradžių atlikus morfologinio uždarymo operaciją, o paskui gautiems duomenims atlikus morfologinio atidarymo operaciją, gauname galutinę segmentuotų plaučių sritį. 24 pav. galime matyti vaizdą prieš atliekant morfologines operacijas ir po jų. Patogumo bei aiškumo dėlei iliustracijoje vaizduojamas jau galutinis vaizdas sukonstravus paviršių. Tai leis aiškiau pamatyti ir įvertinti galutinį rezultatą.



24 pav. Viršuje plaučių sritis prieš morfologinio uždarymo bei atidarymo operacijas, apačioje ta pati plaučių sritis bet jau atlikus uždarymo bei atidarymo operacijas.

6. Paviršiaus sukonstravimas ir 3D vizualizavimas

Šiame skyrelyje bus aprašomas segmentuotų plaučių taškų 3D modeliavimas bei jo atvaizdavimas. 5 skyriuje gauti taškai bus apjungti sukonstravus dengiantįjį paviršių. Sukonstruotas paviršius vėliau bus atvaizduotas vartotojui, pasitelkus 3D vaizdavimo algoritmus.

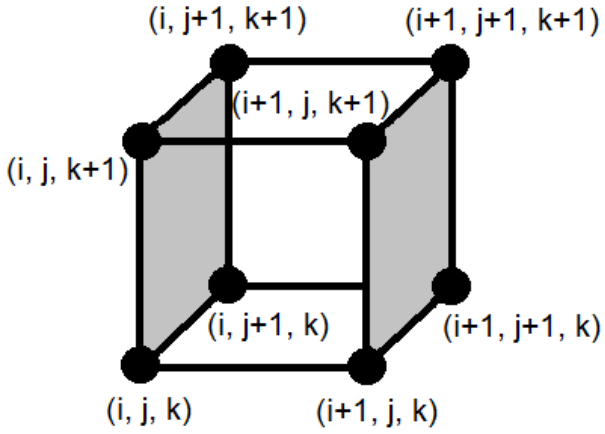
6.1. Dengiančiojo paviršiaus sukonstravimas

Ankstesniuose skyriuose gauti rezultatai yra taškų aibės. Taškų aibė nėra patogus įrankis medikams bandant suprasti plaučių struktūrą, darinių dydžius ir t.t. . Norėdami išspręsti šį trūkumą, būtina sugeneruoti dengiantįjį paviršių.

6.1.1. Judančių kubų algoritmas

Spręsdami dengiančiojo paviršiaus sukonstravimo uždavinį pasinaudosime "judančių kubų" (angl. Marching cubes) algoritmu. Šis algoritmas yra vienas populiariausių algoritmų skirtas generuoti dengiančiuosius paviršius iš turimos taškų aibės. Šis metodas buvo pasiūlytas Viljamo Lorenso ir Harvei Klaino [6] 1987 metais. Autoriai savo darbe aprašė bei pritaikė šitą algoritmą rezonanso tyrimo turiniams duomenų rinkiniams.

Judančių kubų algoritmas interpretuoja skaliarinį duomenų rinkinį kaip kubų aibę sudarančią visą duomenų tūrį. Kiekvienas kubas sudarytas iš 8 viršūnių bei 12 kraštinių. Toks kubas dažnai yra vadinamas vokseliumi, tokio kubo pavyzdį galime matyti 25 pav. . Algoritmo vykdymo metu "judama" nuo vieno vokselio prie kito tikrinant kiekvieno vokselio viršūnių reikšmes. Priklausant nuo vokselio viršūnių reikšmių algoritmas keičią kubą atitinkamais poligonų rinkiniais. Suma visų poligonų iš visų kubų ir bus aproksimuotas paviršius.



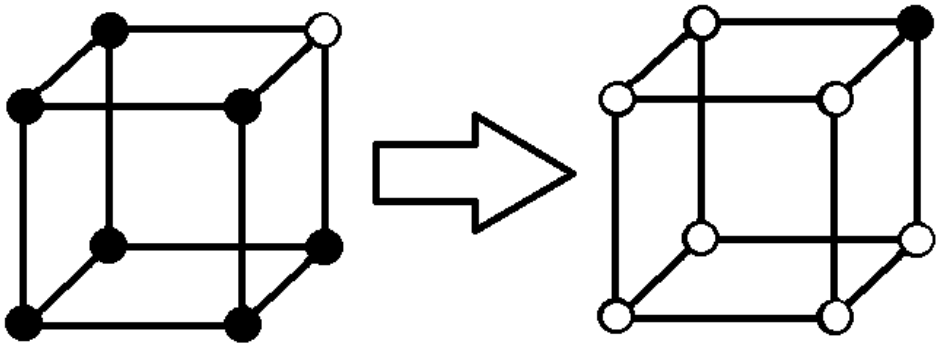
25 pav. Vokslio pavyzdys.

Kaip jau minėta anksčiau, vokselis sudarytas iš 8 viršūnių. Kiekviena viršūnė gali priklausyti tiek objekto vidui tiek išorei. Galime nesunkiai apskaičiuoti visų kubų variacijų skaičių:

$$P = a^n = 2^8 = 256 \tag{6.1}$$

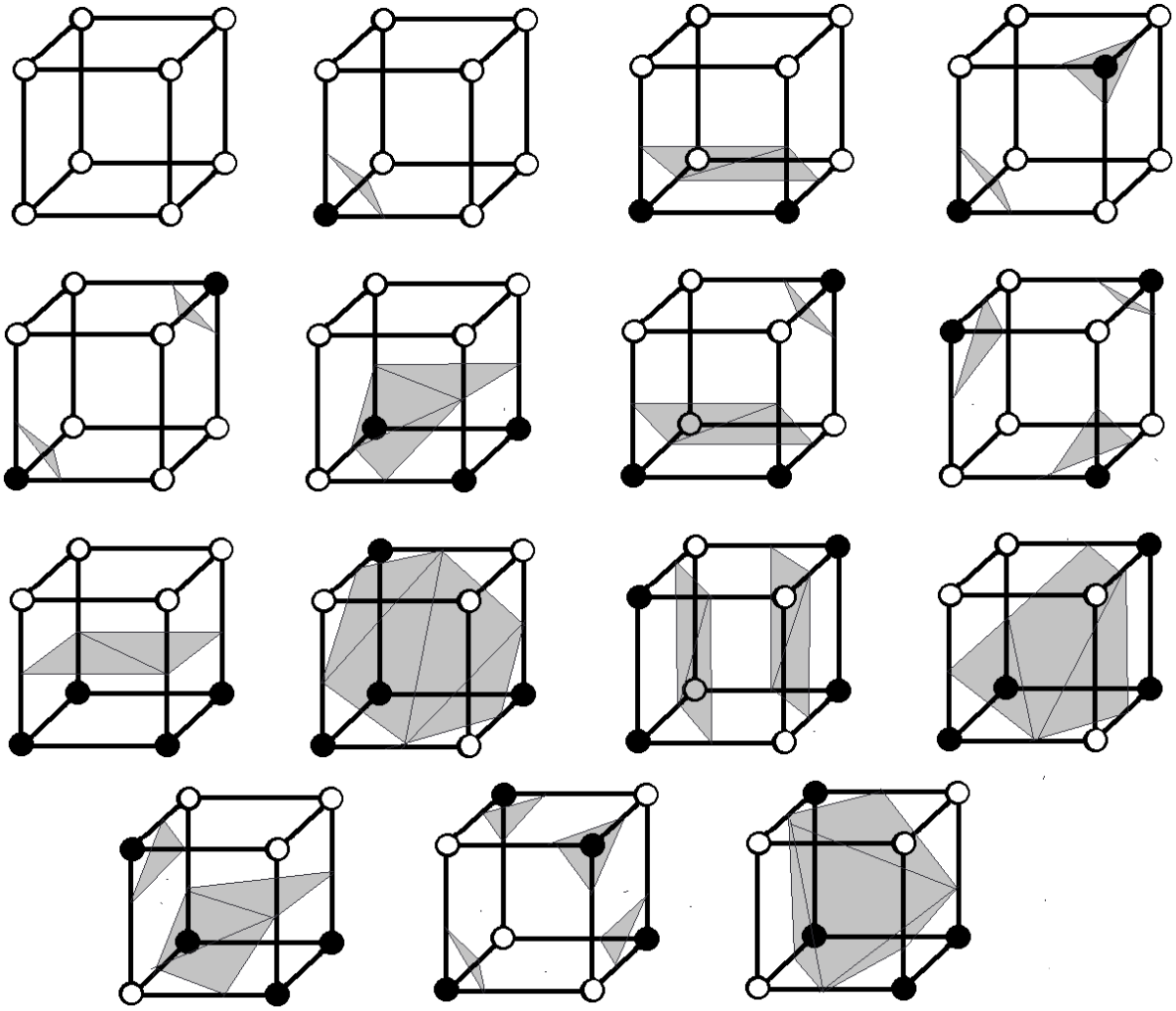
Čia P visų galimų variacijų skaičius, a - viršūnės galimų reikšmių aibės dydis, n - bendras viršūnių skaičius.

Galime pastebėti, kad tarp visų 256 variantų galime atrasti pasikartojančių bei transformuojamų atvejų. Tokio pasikartojimo pavyzdį galime matyti 26 pav. .



26 pav. Variacijų simetriškumas. Kairėje taškų aplinka su vienu tuščiu tašku viršuje, dešinėje taškų su vienu užpildytu taškų viršuje

Pasinaudoję simetriškumo savybe, besukant kubą palei tris pagrindines jo ašis, bei papildomai įvetinę atvejus kai variacijos gautos atspindintint kubo viršūnes (26 pav.), galime išskirti tik 15 unikalių kombinacijų 27 pav. .



27 pav. 15 unikalių poligonų kombinacijų.

Jundančių kubų algoritmą buvo nuspręsta įgyvendinti Java programavimo aplinkoje. Šio algoritmo įgyvendinimas buvo kurtas kaip pagalbinis modulis pagrindinėje sprendimo sistemoje. Šis modulis yra atskiras, standžiai nesusijęs su pagrindine sprendimo sistema. Tai leidžia paviršiaus sukonstravimo procesą atsieti nuo plaučių taškų radimo proceso, tai savo ruožtu leis tobulinti du skirtingus modulius nepriklausomai vienas nuo kito. Šio modulio API panaudojimą galime matyti žemiau:

```
1        int[][][] plauciuSritiesTaskuMasyvas = gautiTaskuMasyva();
2        String stlDuomenuFailoPavadinimas = "C:/Temp/plauciai.stl";
3        MarchingCubes.make(plauciuSritiesTaskuMasyvas,
             stlDuomenuFailoPavadinimas);
```

Paviršiaus sukonstravimo algoritmo pagrindinis uždavinys yra apskaičiuoti paviršiaus aproksimavimo poligonus. Kadangi poligonai sudaryti iš trikampių, svarbu yra apskaičiuoti visas trikampių viršūnes bei jų normales. Atlikus visus šiuos apskaičiavimus būtina juos išsaugoti tolimesniai jų analizei.

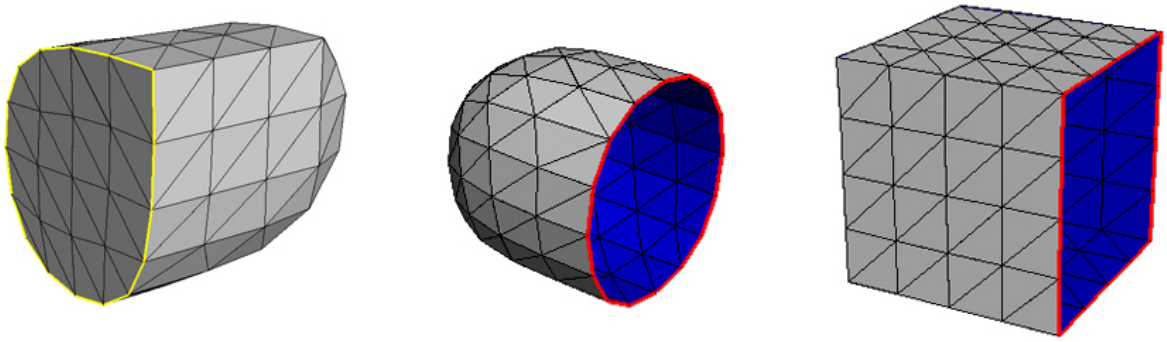
6.1.2. STL formatas

Sukonstravus dengiantįjį paviršių būtina jį išsaugoti, kadangi jis yra galutinis segmentacijos rezultatas. Išsaugotas paviršius gali būti perkeltas į kitas aplinkas vizualizacijai arba tolesniai analizei.

Paviršiui saugoti dažnai yra naudojamas STL [7] (angl. StereoLithography) formatas. Šis formatas plačiai naudojamas 3D duomenų saugojimui ir perdavimui tarp įvairių sistemų, pavyzdžiui 3D spausdintuvai, 3D apdorojimo programos. Šis formatas aprašo objekto geometriją, tiksliau jis saugo rekonstruoto paviršiaus struktūrą, trikampius. Žemiau pateiktas šio formato failo pavyzdys:

STL failo elementas	Paaiškinimas
solid	Paviršiaus aprašymo pradžios elementas.
facet normal 0.66666 0.66669 0.33333	Pirmojo trikampio aprašymo pradžia bei jo normalių reikšmės.
outer loop	Trikampio viršūnių aprašymo pradžia.
vertex 329 -2 153	Pirmoji trikampio viršūnė.
vertex 328 -2 154	Antroji trikampio viršūnė.
vertex 329 0 154	Trečioji trikampio viršūnė.
endloop	Trikampio viršūnių aprašymo pabaiga.
endfacet	Pirmojo trikampio aprašymo pabaiga.
facet normal 0 0 0	Antrojo trikampio aprašymo pradžia bei jo normalės.
...	...
endsolid	Paviršiaus aprašymo pabaiga.

Galime pastebėti, kad šis formatas yra gana paprastas, bet tai neapriboja jo galimybių. Šio formato pagalba galima aprašyti praktiškai bet kokią geometriją, neapsiribojant viršūnių bei kraštinių skaičiumi. Kelių objektų pavyzdžius galime matyti 28 pav.



28 pav. Skirtingų geometrijų pavyzdys STL formatu

6.2. 3D vizualizavimas

Sukonstravus dengiantįjį paviršių bei išsaugojus jį STL formatu, užbaigėme atkūrimo procesą. Toliau būtina realizuoti vartotojui patogią aplinką atkurtiems plaučiams peržiūrėti. Šiai užduočiai atlikti bus naudojami 3D vizualizavimo įrankiai.

6.2.1. WebGL

Pastaraisiais metais pasiektas ženklus progresas įgyvendinant vaizdo apdorojimo metodus, naudojant vaizdo plokštės galimybes, interneto naršyklėse. Atsiradus HTML5 standartui tapo įmanoma apdoroti bei atvaizduoti sudėtingus 3D objektus realiame laike interneto naršyklėje. Šiame darbe gautus rezultatus buvo nuspręsta atvaizduoti internetiniame puslapyje, kadangi šis būdas nereikalauja jokių papildomų kompiuterinės įrangos programų bei sistemų. Pakanka turėti šiuolaikinę interneto naršyklę palaikančią HTML5 standartą ir WebGL specifikaciją. Lentelėje žemiau galime matyti populiariausias interneto naršykles bei jų minimalias versijas palaikančias HTML5 ir WebGL.

Naršyklė	Versija nuo kurios palaikomas WebGL
Firefox	31+
Chrome	30+
Internet Explorer	11+
Opera	20+

WebGL [8] yra interneto standartas skirtas įgalinti primityvios (angl. low-level) geometrijos vaizdavimą. Šis standartas remtas OpenGL ES 2.0 standartu, kuris plačiai naudojamas šiuolaikinėje kompiuterinėje grafikoje kuriant žaidimus, grafinius įrankius ir kitus vaizdo plokštės resursų reikalaujančius sprendimus. WebGL leidžia pasinaudoti praktiškai visomis vaizdo plokštės teikiamomis galimybėmis.

6.2.2. Three.js

Three.js [9] yra atvira JavaScript programavimo kalbos biblioteka paremta WebGL technologija. Ši biblioteka supaprastina 3D vaizdavimo programavimo procesą, kadangi nebereikia tiesiogiai dirbti su primityvia geometrija. Biblioteka suteikiant patogią aplikacijų programavimo sąsają (angl. API). Žemiau pateikti pagrindiniai šios bibliotekos komponentai.

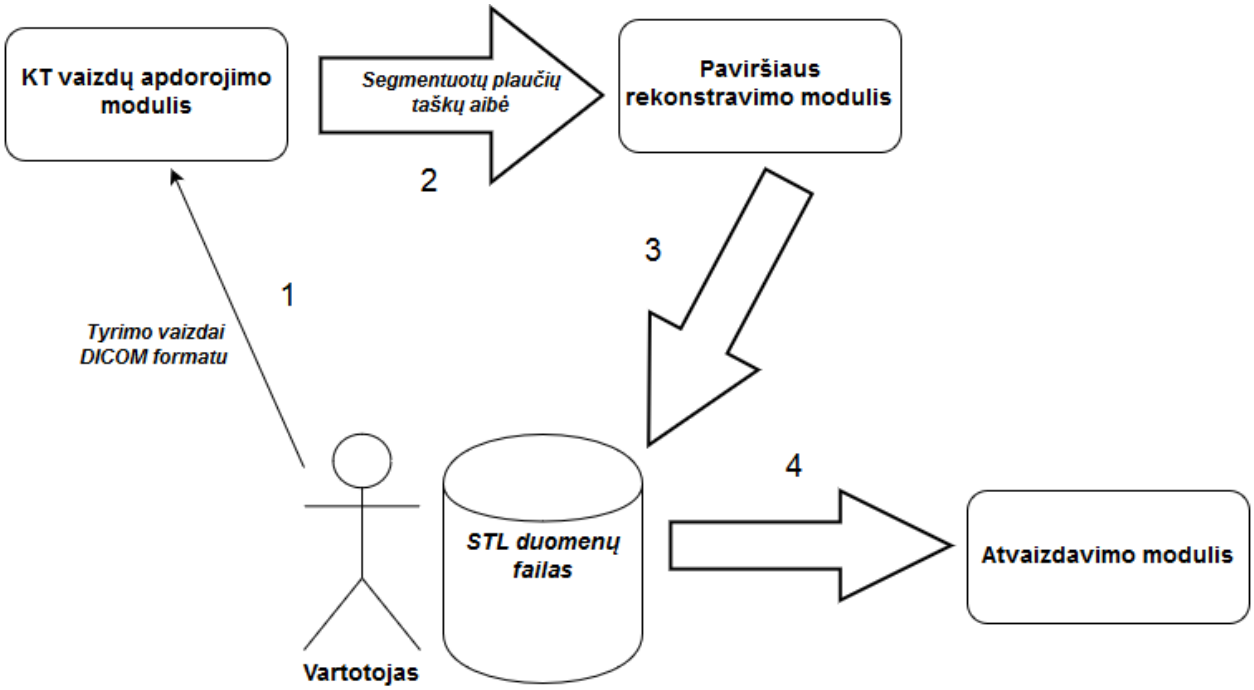
Komponentas	Aprašymas
THREE.Renderer	Pagrindinis komponentas kuris perteikia vaizdą.
THREE.Scene	Komponentas realiu laiku laikantis objektus kuriuos norima atvaizduoti.
THREE.Camera	Virtuali kamera, tiksliau taškas iš kurio bus rodomas vaizdas vartotojui.
THREE.Object3D	Esminė objekto abstrakcija.
THREE.Light	Apšvietimo modelio komponentas.

7. Sprendimo realizacija

Galutinis įgyvendintas sprendimas susideda iš 3 modulių:

1. Kompiuterinės tomografijos vaizdų apdorojimo modulis skirtas DICOM duomenų skaitymui, vaizdų apdorojimui bei plaučių segmentavimui.
2. Paviršiaus sukonstravimo modulis skirtas paviršiui rekonstruoti bei jį išsaugoti.
3. Atvaizdavimo modulis skirtas atkurto paviršiaus atvaizdavimui.

Šių modulių tarpusavio sąveika pateikta 29 pav..



29 pav. Sistemos modulių sąveika

Vartotojas tiesiogiai gali sąveikauti su KT vaizdų apdorojimo bei atvaizdavimo moduliais.

7.1. Vaizdų apdorojimo bei paviršiaus rekonstravimo moduliai

Vaizdų apdorojimo modulis bei paviršiau rekonstravimo modulis realizuotas Java programavimo kalba. Pasirinkta Java programavimo kalba yra viena populiariausių programavimo kalbų kuriant IT sprendimus. Ši kalba yra nuolat tobulinama bei turi gana platų įrankių pasirinkimą. Taipogi specialistų, išmanančių šią kalbą, Lietuvoje ir pasaulyje sparčiai daugėja. Įvertinus šiuos privalumus bei mąstant apie galimą šio darbo gamybinę realizaciją, buvo nuspręsta pasirinkti būtent Java programavimo kalbą.

Vaizdų apdorojimo modulis yra pradinis modulis, kuriame prasideda segmentavimo procesas. Šio modulio įeinamasis parametras yra kompiuterinės tomografijos tyrimų vaizdai, o išeinamasis segmentuotų plaučių taškų masyvas. Rašant programos kodą visi algoritmai buvo realizuoti nuo nulio, tačiau, DICOM formato duomenų skaitymui buvo pasitelkta ImageJ DICOM biblioteka [10].

Žemiau abstrakčiai pateikti šio modulio esminiai žingsniai:

1. DICOM formato duomenų skaitymas bei išsaugojimas programos veikimo atmintyje.

2. Plaučių taškų segmentavimas.

3. Segmentuotų plaučių taškų perdavimas paviršiaus rekonstravimo moduliui.

Paviršiaus rekonstravimo modulis atsakingas už segmentuotų taškų rasterizavimą. Įeinamasis šio modulio parametras segmentuotų taškų aibė, o modulio veikimo rezultatas, STL formato rinkmena apibrėžianti rekonstruotą paviršių.

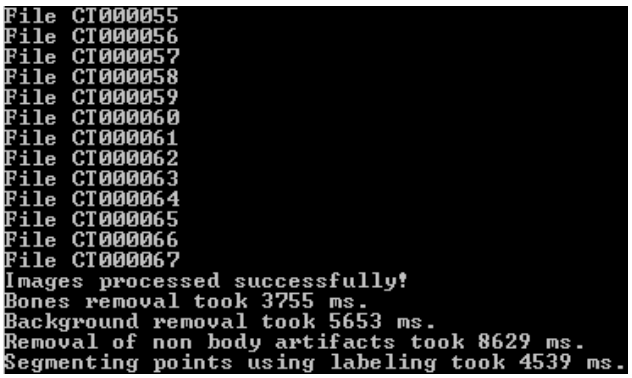
Apibrėžkime esminius šio modulio žingsnius:

1. Segmentuotų plaučių taškų aibės perranka, vokslelių konstravimas.

2. Paviršiaus trikampių apskaičiavimas, normalių apskaičiavimas.

3. Gautų trikampių bei jų normalių išsaugojimas struktūritizuotame STL formato rinkmenoje.

Šių modulių realizuotos programos veikimą galime stebėti 30 pav..



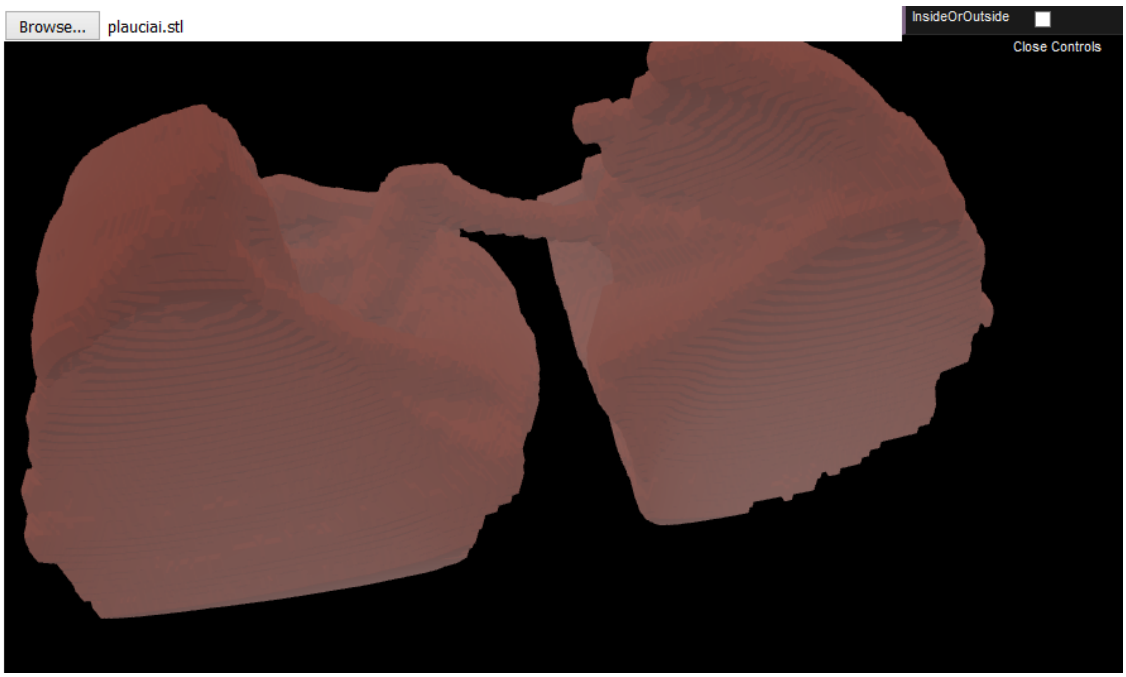
30 pav. Vaizdų apdorojimo bei paviršiaus rekonstravimo modulių programos veikimas.

7.2. Atvaizdavimo modulis

Atvaizdavimo moduliui vartotojas pateikia rekonstruoto paviršiaus duomenų failą, kurį kaip rezultatą grąžina paviršiaus rekonstravimo modulis. Atvaizdavimo modulis realizuotas JavaScript programavimo aplinkoje. Šis modulis veikia interneto naršyklėje, tad vartotojo aplinkai nekeliami jokie papildomi reikalavimai. Tačiau, svarbu, kad naršyklė palaikytų WebGL standartą. Šiais laikas praktiškai visos populiariausios interneto naršyklės jį palaiko.

Įgyvendinant atvaizdavimo modulio sprendimą buvo pasitelkta Three.js, JQuery.js, STLLoader.js bibliotekų pagalba. Šios bibliotekos palengvina programavimo procesą dirbant su 3D WebGL varikliu.

Atvaizdavimo modulio vartotojo aplinka pavaizduota 31 pav. .



31 pav. Vartotojui prienama programos aplinka

Vartotojui prieinamos sekančios valdymo operacijos:

1. Vartotojas turi galimybę pasirinkti rekonstruotų plaučių paviršiaus STL failą.
2. Paviršiaus rekonstruotas modelis kompiuterio ekrane gali būti priartinamas arba atitolinamas.
3. Rekonstruotas paviršius kompiuterio ekrane gali būti stumdomas, vartomas, sukiojamas kompiuterinės pėlės pagalba.
4. Taip pat yra galimybė vaizduoti išorines arba vidines paviršiaus dalis.

Išvados ir rekomendacijos

Šiame darbe buvo aptarti bendri kompiuterinės tomografijos veikimo principai. Išsiaiškinta ir aprašyta DICOM formato specifikacija bei formato ypatumai. Plačiai aptarta vaizdų binarizavimo metodika, aprašyta šio metodo specifika, pasiūlyti keli binarizavimo algoritmai.

Analizuojant susijusius darbus buvo nustatyta, kad esami algoritmai bei baigtiniai sprendimai remiasi segmentavimu su būtinu vartotojo įsikišimu. Vartotojas būtinas slenksčio pasirinkimo metu. Darbe buvo pasiūlyta naudoti automatinio segmentavimo algoritmą apskaičiuojant slenksčio dydį automatiškai. Rezultate buvo gautas visiškai automatizuotas metodas, kuris teisingai segmentuoja kaulų audinius be vartotojo įsikišimo, be to sugeba automatiškai apskaičiuoti taškų poziciją erdvėje ir pakankamai gerai atvaizduoti krūtinės ląstos kaulų sistemos taškus 3D.

Vėliau darbe pasiūlytas sprendimas segmentuoti plaučių srities taškus. Gautas sprendimas remiasi kompiuterinės regos algoritmais ir yra visiškai automatinis, nereikalaujantis vartotojo įsikišimo. Gautus duomenis buvo nuspręsta vektorizuoti tam panaudojant paviršiaus rekonstravimo algoritmą. Šio algoritmo pagalba buvo gautas dengiantysis paviršius, kuris gali būti atvaizduojamas vartotojui.

Praktinėje darbo dalyje buvo sukurta kompiuterinė sistema susidedanti iš 3 modulių:

- Modulis atsakingas už plaučių taškų išskiriamą bei segmentaciją.
- Modulis skirtas dengiančiąjam paviršiui sukonstruoti.
- Vartotojo sąsajos modulis skirtas atvaizduoti plaučių sritį ir leisti vartotojui realiu laiku apžiūrėti gautą rezultatą.

Ateities tyrimų planas

Šiame darbe buvo pasiūlytas sprendimas apimantis plaučių taškų segmentavimą, jų paviršiaus rekonstravimą, bei jo atvaizdavimą vartotojui patogioje sąsajoje. Kiekvienas iš šių etapų gali būti tobulinamas. Išskirkime pagrindines ateitės tyrimų kryptis:

1. Šiame darbe aptarti segmentavimo algoritmai ateityje gali būtų tobulinami. Pavyzdžiui atsiradus poreikiui segmentuotį ortakių sistemą, sprendimas turi būti patobulintas.

2. Paviršiaus rekonstravimo sprendimas gali būti taipogi patobulintas. Šiais laikas egzistuoja daug paviršiaus rekonstravimo algoritmų bei metodų. Esant poreikiui tikslesniam paviršiaus rekonstravimui būtina pasiūlytą sprendimą patobulinti, pridedant galimybę rekonstruoti paviršius kitais metodais.

3. Vartotojo sąsaja taipogi privalo būti patobulinta norint perkelti pasiūlyta sprendimą į gamybinę eksploataciją.

Literatūros šaltiniai

[1] Maguire Jr Baxter BS, Hitchner LE. A standard format for digital image exchange. american institute of physicists in medicine (aapm) report number 10, 1982.

[2] Nisar Ahmed Memoni. Segmentation of lungs from ct scan images for early diagnosis of lung cancer, 2006. World Academy of Science, Engineering and Technology.

[3] B. Ramadoss ir S. Baskar C. Karthikeyan. Segmentation algorithm for ct images using morphological operation and artificial neural network, 2012. International Journal of Signal Processing, Image Processing and Pattern Recognition Vol. 5, No. 2.

[4] Feng Li ir Qiang Li Jiahui Wang. Automated segmentation of lungs with severe interstitial lung disease in ct, 2009. Med Phys. 2009 Oct; 36(10): 4592–4599.

[5] Yuri Y. Boykov Marie-Pierre Jolly. Interactive graph cuts for optimal boundary region segmentation of objects in n-d images, 2001. Proceedings of “Internation Conference on Computer Vision" vol.I, p.105.

[6] William E. Lorensen Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm, 1987. Computer Graphics, Volume 21.

[7] Inc. 3D Systems. Stereolithography interface specification, 1988.

[8] Dean Jackson. Webgl specification. https://www.khronos.org/registry/webgl/specs/1.0/.

[9] Jos Dirksen. Three.js essentials.

[10] ImageJ. Imagej dicom plugin api specification, 2008. https://imagej.nih.gov/ij/developer/api/ij/plugin/DICOM.html.

8. Priedai

Žemiau pateikiama dalis(maža) esminio sistemos programinio kodo.

1 išeities kodas. Matematinės morfologijos operacijų implementacija

```
1
2 /**
3 * Created by Jevgenij Teodorvic on 12/08/2016.
4 */
5 public class StructureElement {
6
7         public static final String NON_APPLICABLE = "NON_APPLICABLE";
8         public static final String DID_NOT_MATCH = "DID_NOT_MATCH";
9         public static final String MATCH = "MATCH";
10
11        /*
12        ....
13        ....
14        */
15
16        private int[][] matrix2d;
17        private int[][][] matrix3d;
18
19        private final int side;
20
21        /*
22        ...
23        */
24
25        public static StructureElement get3dRBall5x5() {
26                return new StructureElement(BALL_R3_3D);
27        }
28
29        /*
30        ...
31        */
32
33        public Map<String, Integer> get3d(int[][][] val, int x, int y,
              int z) {
34                if (this.matrix3d == null)
35                        throw new UnsupportedOperationException("Chosen
                              structure element is not applicable for 3D
                              operations!" +
36                        "Ensure that you are using correct structure
                              element");
37
38                Map<String, Integer> results = new HashMap<>();
39
40                int na = 0;
41                int match = 0;
42                int no_match = 0;
43                for (int lx = (-1) * side; lx <= side; lx++)
44                for (int ly = (-1) * side; ly <= side; ly++)
45                for (int lz = (-1) * side; lz <= side; lz++){
46                        if (!ensurePosition(val, x + lx, y + ly, z + lz)
```

```
                                ) {
47                                  if (matrix3d[lx + side][ly + side][lz +
                                        side] == 0)
48                                  na++;
49                                  else
50                                  no_match++;
51                                  continue;
52                              }
53
54                              if (matrix3d[lx + side][ly + side][lz + side] ==
                                    0)
55                                  na++;
56                              else if (val[x + lx][y + ly][z + lz] == 1)
57                                  match++;
58                              else
59                                  no_match++;
60                      }
61
62                      results.put(NON_APPLICABLE, na);
63                      results.put(MATCH, match);
64                      results.put(DID_NOT_MATCH, no_match);
65                      return results;
66              }
67
68              private boolean ensurePosition(int[][][] val, int i, int j, int
                    z) {
69                      if (i < 0 || i > val.length - 1)
70                              return false;
71                      if (j < 0 || j > val[0].length - 1)
72                              return false;
73                      if (z < 0 || z > val[0][0].length - 1)
74                              return false;
75
76                      return true;
77              }
78
79 }
80
81
82 /**
83 * Created by Jevgenij Teodorvic on 12/08/2016.
84 */
85 public final class MorphologyOperation {
86
87          /*
88          ...
89          */
90
91          public static int[][][] erode(int[][][] data, StructureElement
                structureElement) {
92                  return doOperation(data, structureElement, (s) -> s.get(
                        StructureElement.DID_NOT_MATCH) == 0);
93          }
94
95          public static int[][][] dilate(int[][][] data, StructureElement
```

```
            structureElement) {
96              return doOperation(data, structureElement, (s) -> s.get(
                    StructureElement.MATCH) > 0);
97          }
98
99          public static int[][][] open(int[][][] data, StructureElement
                structureElement) {
100             return dilate(erode(data, structureElement),
                    structureElement);
101         }
102
103         public static int[][][] close(int[][][] data, StructureElement
                structureElement) {
104             return erode(dilate(data, structureElement),
                    structureElement);
105         }
106
107
108         /**
109         * Method which makes actual operations.
110         * @param data - Image data
111         * @param structureElement - Structure element
112         * @param criteria - Criteria for accepting points
113         * @return Image data after morphological operations.
114         */
115         private static int[][][] doOperation(int[][][] data,
                StructureElement structureElement, MorphologyCriteria
                criteria) {
116             int x = data.length;
117             int y = data[0].length;
118             int z = data[0][0].length;
119             int[][][] res = new int[x][y][z];
120
121             for (int i = 0; i < x; i++)
122                 for (int j = 0; j < y; j++)
123                     for (int k = 0; k < z; k++)
124                         if (criteria.acceptPoint(
                                structureElement.get3d(data,
                                i, j, k)))
125                             res[i][j][k] = 1;
126
127                 return res;
128         }
129
130         /**
131         *
132         */
133
134 }
```

2 išeities kodas. Sujungtų komponenčių algoritmo branduolys

```
1
2 /**
3 * Created by Jevgenij Teodorvic on 10/1/2016.
```

```
4 */
5 public abstract class EightWayFloodFill<T> {
6
7         public T[][][] floodFill(T[][][] points) {
8         boolean[][][] painted = new boolean[DEEP()][WIDTH()][HEIGHT()];
9
10        floodFillCore(points, painted, null, Utils.getCoord(0,0,0));
11
12        doWhatYouNeedWithPoints(points, painted);
13
14        return points;
15        }
16
17        protected void floodFillCore(T[][][] points, boolean[][][]
              painted, List<Integer> labelList, Integer seedPoint) {
18        java.util.Queue<Integer> queue = new LinkedList<>();
19        queue.add(seedPoint);
20
21        while (!queue.isEmpty()) {
22        Integer p = queue.remove();
23        int z = Utils.getZ(p);
24        int x = Utils.getX(p);
25        int y = Utils.getY(p);
26
27        if (skipPoint(points, painted, z, x, y, null))
28                continue;
29
30        //Only for labeling, in background it is impossible to have null
31        Object ourPoint = points[z][x][y];
32        if (ourPoint == null)
33                continue;
34
35        painted[z][x][y] = true;
36        for (int k = -1; k <= 1; k++)
37        for (int i = -1; i <= 1; i++)
38        for (int j = -1; j <= 1; j++) {
39                if (Math.abs(k) + Math.abs(i) + Math.abs(j) != 1)
40                        continue;
41
42                int newK = z + k;
43                int newI = x + i;
44                int newJ = y + j;
45                if (newK >= 0 && newK < DEEP() && newI >= 0 && newI <
                      WIDTH() && newJ >= 0 && newJ < HEIGHT())
46                        if (!skipAddingPoint(points, painted, newK, newI
                              , newJ, ourPoint)) {
47                                Integer coord = Utils.getCoord(newI,
                                      newJ, newK);
48
49                                queue.add(coord);
50                                // Only for labeling
51                                if (labelList != null) {
52                                        labelList.add(coord);
53                                        painted[newK][newI][newJ] = true
                                              ;
```

```
54                                      }
55                                  }
56          }
57          }
58          }
59
60 }
```

3 išeities kodas. Judančių kubų algoritmo branduolys

```
 1
 2 /**
 3 * Created by Jevgenij Teodorvic on 10/1/2016.
 4 */
 5
 6 public static Map<String, List<int[]>> m(int[] dimensions, int[][][]
       points) {
 7          int[] cubes = new int[8];
 8          int[] edges = new int[12];
 9          List<int[]> vertices = new ArrayList<>();
10          List<int[]> faces = new ArrayList<>();
11
12          Map<String, List<int[]>> results = new HashMap<>();
13          results.put("positions", vertices);
14          results.put("cells", faces);
15
16          int[] x = new int[]{0,0,0};
17
18          for ( x[2]=0; x[2] < dimensions[0]; x[2]++)
19          for( x[1]=0; x[1] < dimensions[1]; x[1]++)
20          for(x[0]=0; x[0] < dimensions[2]; x[0]++) {
21                  int cube_index = 0;
22
23                  for(int i=0; i<=7; ++i) {
24                          int[] v = cubeVerts[i];
25                          int s = points[x[0]+v[0]][x[1]+v[1]][x[2]+v[2]];
26                          cubes[i] = s;
27                          cube_index |= (s > 0) ? 1 << i : 0;
28                  }
29
30                  int edgem = edgeTable[cube_index];
31                  if(edgem == 0)
32                          continue;
33
34
35                  for(int i=0; i<=11; ++i) {
36                          if((edgem & (1<<i)) == 0)
37                                  continue;
38                          edges[i] = vertices.size();
39
40                          int[] normal = new int[] {0,0,0};
41                          int[] e = edgeIndex[i];
42
43                          int[] p0 = cubeVerts[e[0]];
44                          int[] p1 = cubeVerts[e[1]];
```

```
45                          int a = cubes[e[0]];
46                          int b = cubes[e[1]];
47                          int coef = a - b;
48                          int t = 0;
49                          if(Math.abs(coef) > 1e-6)
50                                  t = a / coef;
51
52                          for(int j=0; j<=2; ++j) {
53                                  normal[j] = (x[j] + p0[j]) + t * (p1[j]
                                        - p0[j]);
54                                  if (j==2) normal[j]*=2;
55                          }
56                          vertices.add(normal);
57                  }
58
59                  int f[] = triTable[cube_index];
60                  for(int i=0; i<f.length; i += 3)
61                          faces.add(new int[] {edges[f[i]], edges[f[i+1]],
                                edges[f[i+2]]});
62
63          }
64
65          return results;
66  }
```

4 išeities kodas. DICOM duomenų skaitymo branduolys

```
1
2 /**
3 * Created by Jevgenij Teodorvic on 9/15/2016.
4 */
5 public static Integer[][][] getPoints(String pathToDicomFolder) throws
      IOException {
6     File folder = new File(pathToDicomFolder);
7     File[] listOfFiles = folder.listFiles();
8     List<int[][]> allFilesPoints = new ArrayList<>();
9
10    int i = 0;
11    for (; i < listOfFiles.length; i++) {
12        System.out.println("File " + listOfFiles[i].getName());
13        allFilesPoints.add(getImagePoints(listOfFiles[i].getPath()));
14        }
15
16        DEEP = i;
17        Integer[][][] allPoints = null;
18    for (int[][] filePoints : allFilesPoints) {
19        if (allPoints == null) {
20                WIDTH = filePoints.length;
21                HEIGHT = filePoints[0].length;
22                allPoints = new Integer[DEEP][WIDTH][HEIGHT];
23        }
24        int z = filePoints[0][0];
25                filePoints[0][0] = -1000;
26
27        for (int a = 0; a < filePoints.length; a++)
```

```
28                  for (int b = 0; b < filePoints[0].length; b++)
29          allPoints[i - z][a][b] = filePoints[a][b];
30      }
31
32          return allPoints;
33  }
```

5 išeities kodas. Atvaizdavimo JavaScript kodo branduolys

```
 1
 2          var X = 1200, Y = 1200;
 3
 4          var container3d;
 5          var camera, cameraTarget, scene, renderer, lungs, cameraControls
                ;
 6          var clearColor = 0xcccccc;
 7          var lungColor = 0xff5534;
 8          var lungMaterialInside = new THREE.MeshPhongMaterial({
 9                  color: lungColor, emissive: lungColor, emissiveIntensity
                        : 0.2,
10                  specular: 0xFFFFFF, shininess: 10,
11                  shading: THREE.SmoothShading,
12                  side: THREE.FrontSide
13          });
14          var lungMaterialOut = new THREE.MeshPhongMaterial({
15                  color: lungColor, emissive: lungColor, emissiveIntensity
                        : 0.2,
16                  specular: 0xFFFFFF, shininess: 10,
17                  shading: THREE.SmoothShading,
18                  side: THREE.BackSide
19          });
20          var gui = new dat.GUI();
21
22          gui.add({InsideOrOutside: false}, 'InsideOrOutside').
                onFinishChange(function(value) {
23                  lungs.material = value ? lungMaterialInside:
                        lungMaterialOut;
24          });
25
26          $('input[type=file]').change(function () {
27                  initialize($(this).val());
28                  animation();
29          });
30
31
32
33          function initialize(path) {
34                  container3d = document.getElementById('container3d');
35                  document.body.appendChild(container3d);
36
37                  scene = new THREE.Scene();
38                  scene.fog = new THREE.Fog(clearColor, 0.1, 8);
39
40                  camera = new THREE.PerspectiveCamera(45, X / Y, 0.1,
                        20000);
```

```
41                  camera.position.set(3, 0.8, 3);
42                  scene.add(camera);
43
44                  new THREE.STLLoader().load(path, function (geometry) {
45                          lungs = new THREE.Mesh(geometry, lungMaterialOut
                                ;
46                          lungs.position.set(0, 0, 0);
47                          lungs.rotation.set(-Math.PI / 2, 0, 0);
48                          lungs.scale.set(0.005, 0.005, 0.005);
49                          scene.add(lungs);
50                  } );
51
52                  scene.add(new THREE.HemisphereLight(0x443333, 0x111122,
                        1));
53
54                  renderer = new THREE.WebGLRenderer({antialias: true});
55                  renderer.setSize(X, Y);
56                  container3d.appendChild(renderer.domElement);
57
58                  cameraControls = new THREE.TrackballControls(camera,
                        renderer.domElement);
59                  cameraControls.target.set( 0, 0, 0 );
60          }
61
62          function animation() {
63                  requestAnimationFrame(animation);
64                  renderer.render(scene, camera);
65                  cameraControls.update();
66          }
```