VILNIAUS UNIVERSITETAS
MATEMATIKOS IR INFORMATIKOS FAKULTETAS
KOMPIUTERIJOS KATEDRA

Baigiamasis magistro darbas

Trimačių objektų atpažinimas naudojant LiDAR duomenis

Atliko:
Ruslan Karpovič parašas

Vadovas:
dr. Valdas Rapševičius

Vilnius
2017

Turinys

Santrauka

Norint išskirti objektus iš daugybės kitų aplinkoje esančių objektų reikalingi prietaisai, kurie padėtų tai padaryti. LIDAR (Light Detection and Ranging) prietaisas surenka duomenis apie objektus. Darbe bandoma surasti tinkamus metodus, galinčius apdoroti LIDAR duomenis, kurie kuo tiksliau išskirtų objektus ir kurių pagalba būtų galima išskirti žemės paviršiaus taškus, bei metodus, kurie padėtų išskirti cilindrus, kaip medžių kamienus. Atliekama kiekybinė metodų analizė ir metodai lyginami tarpusavyje. Naudojami metodai detaliai aprašomi teorinėje dalyje.

Summary

3D Object Identification by Using LiDAR Data

In today's world we are surrounded by various objects. In order to better understand and explore, we need to move to a virtual environment in order to optimize and research. Manual data analysis requires a lot of time and due to the abundance of data becomes very dif ficult. For this reason, are used computer algorithms, which aim to ensure the rapid and qualitative detection of objects.

In order to stand out against the many other objects in the environment required for devices that would do it. LIDAR (Light Detection and Ranging) device collects data about objects. On the thesis there are attempts to find suitable methods capable of processing LIDAR data. The goal is to precisely distinguish objects and to make it possible to distinguish the Earth's surface points and methods, who help identify the cylinders, like tree trunks. It was made quantitative methods of analysis that are compared between each other. All the used methods are described in details in the theoretical part.

Įvadas

LIDAR (Light Detection And Range) - tai technologija, kuri pradėta plačiai naudoti, norint, gauti ir apdoroti informaciją, susijusią su erdviniu žemės paviršiumi, naudojama kelių, pastatų, miškų, įvairių paviršių tyrimams, kuomet gaunama trijų dimensijų (3D) informacija: siunčiama labai tanki taškinė geometrinė ir radiometrinė informacija.

Norint gauti LIDAR duomenis, lidarai montuojami į lėktuvus, sklandytuvus, laivus, automobilius, traukinius, žemės palydovus, kosmines stotis arba tiesiog naudojamas kaip stacionarus prietaisas. Naudojant lazerinio skenavimo technologiją pasiekiamas didelis greitis bei operatyvumas renkant duomenis. Dėl to galima vienu metu gauti labai daug duomenų. LIDAR metodu renkant informaciją, o vėliau ją susisteminus, galima modeliuoti, identifikuoti, objektus.

Nepaisant to, jog technologija yra labai populiari ir plačiai naudojama, galima įžvelgti ir keletą trūkumų. Kadangi prietaisas gauna labai daug duomenų, reikia ne vieno matematinio metodo jiems susisteminti ir išanalizuoti. Gauti duomenys užima daug vietos.

Sukurti ir naudojami metodai ne visuomet tiksliai klasifikuoja objektus, kartais vieni objektai priskiriami kitiems (pvz.: automobilis gali būti supainiotas su kelio dalimi ir pan.). Tiriant gautus duomenis apie medžius, susiduriama su problema, kuomet sudėtinga išskirti medį, kaip vienetą.

Darbo motyvacija. Analizuojant susijusius darbus buvo pastebėta, jog yra labai mažai darbų, kuriose būtų tiriami stacionaraus prietaiso duomenys.

Darbo tikslai:

- Rasti metodus skirtus žemės paviršiaus taškų išskyrimui;
- Rasti metodus skirtus cilindrų, kaip medžių kamienų taškų išskyrimui;
- Esperimentiškai įgyvendinti ir išbandyti metodus;
- Tarpusavyje palyginti metodus, išskiriančius tuos pačius objektus skirtingais būdais.
- Kiekybiškai palyginti metodus su skirtingais parametrais;

Darbo metu buvo aprašyti skirtingi metodai skirti žemės paviršiaus taškų išskyrimui ir cilindrų, kaip medžių kamienų išskyrimui. Taip pat apžvelgiamas vokselių panaudojimas LIDAR duomenyse. Aprašomas LIDAR veikimo principas ir duomenų formatas. Praktiškai įgyvendinti aprašyti metodai, atlikta jų analizė.

Darbo rezultatai:

- Iš turimų duomenų išskirti žemės paviršiaus taškai;
- Iš turimų duomenų išskirti medžių kamienų taškai;
- Pritaikytas vokselių apibrėžimas duomenų optimizacijai.
- Palyginti du žemės paviršiaus taškų išskyrimo metodai;
- Palyginti du medžių kamienų taškų išskyrimo metodai;
- Išanalizuotas metodų veikimas su skirtingai parametrais;

Tęsiamas mokslo tiriamojo darbo projektas, kurio metu buvo išnagrinėta LIDAR duomenų struktūra, bei aptarti susiję kitų autorių darbai.

Darbo struktūra. Darbas pradedamas susijusių darbų analize, kurioje apžvelgiami kitų autorių metodai. Aprašytas LIDAR prietaiso veikimo principas ir duomenų formatas. Apžvelgta vokselio sąvoka ir jo panaudojimas LIDAR duomenyse. Aprašyti žemės paviršiaus išskyrimo metodai ir cilindrų išskyrimo metodai. Detaliai aprašyta eksperimentinė darbo dalis.

1. Susiję darbai

Padarius susijusių darbų apžvalgą nepavyko rasti daug darbų, kurie būtų susiję su LIDAR duomenų apdorojimu iš stacionaraus prietaiso. Labai daug darbų yra atlikta, naudojant duomenis, gautus iš prietaisų, kurie buvo sumontuoti lėktuvuose, todėl ir algoritmai skirti objektų atpažinimui yra pritaikyti apdoroti duomenis, kurie gauti skenuojant paviršių iš skrydžio.

G. I. Nikolajevič 2011 m. [1] Savo darbo metu patobulino matematinį klasifikavimo metodą, kuris yra pagrįstas nuožulnių paviršių adaptyvia analize.

Levašev S. P. 2015 m [2] klasifikavo objektus, naudodamas Furjė deskriptorius, tarpusavyje lygino objektų kontūrus ir nustatė, kad artumo matmuo gali kisti priklausomai nuo atstumo tarp objekto ir lazerio, todėl būtina įtraukti šiuos pokyčius, pavyzdžiui, viena iš galimybių naudoti Bajeso charakteristikas.

A. A. Tkachev 2014 m. [3] naudojo kombinatorinį algoritmą Space Colonization ir L sistemas, ir nustatė, jog naudojant šį metodą trimačiai landšaftai gaunasi realistiniai, ir nereikalauja didelių skaičiavimo resursų.

Caiyun Zhang , Yuhong Zhou ir Fang Qiu , 2015 m. [4] sukūrė naują algoritmą, kuris iš medžių viršūnių atpažįsta atskirus medžius ir bando jos identifikuoti. Jie susidūrė su problema, jog reikia labai daug resursų, norint apdoroti duomenis. Reikalingas programinis algoritmo optimizavimas.

Qingquan Li, Zhipeng Chen, and Qingwu Hu 2015 m. [5] suformavo metodą, kuris iš 3D elektros stulpo modelio, atpažįsta stulpą LIDAR duomenyse. Atpažinimas ne visuomet pavyksta, dėl to planavo tobulinti ir optimizuoti metodą tolimesniuose darbuose.

Bernhard Höfle, Norbert Pfeifer 2007 m.[6] savo darbe dviem metodais tvarkė duomenis, gautus iš lėktuve sumontuoto prietaiso ("model-driven" ir "data-driven). Naudojant pirmąjį metodą geriausių rezultatų pasiekė, naudojant diapazono-kvadrato priklausomybę.

Peter Axelsson 1998 m.[7], savo darbe paminėjo žemės taškų filtravimo algoritmo principus, kai žemės taškai prijungti prie taškų debesies iš apačios. Paviršius gali svyruoti su tam tikra paklaida. Žemės paviršiaus taškai bus žemiausi taškai, kurie neturi didelio pakilimo nuo savo kaimyninių taškų.

Aušra Kalanaitė 2015 m. [8], savo darbe sprendžia fizinio žemės paviršiaus modeliavimo problemas. Modeliuojant žemės paviršių reikia eliminuoti visus kitus objektus, palikti tik taškus būdingus paviršiui. Sprendžiama problema, kuomet reikia nustatyti staigius paviršiaus pakitimus. Taip pat taiko kaimyninių taškų analizę. Darbas atliekamas su duomenimis gautais iš lėktuvo.

Alireza Asvadi, Cristiano Premebida, Paulo Peixoto, Urbano Nunes 2016 m. [9] savo darbe pritaikė vokselius duomenų apdorojimui ir optimizacijai. Panaudojo išskirtinę kliūčių analizę.

Ruwen Schnabel, Roland Wahl, Reinhard Klein 2006 m. [10] savo darbe aprašė metodus su kuriais galima išskirti pagrindines figūras. Jų metodas remiasi atsitiktine atranka (naudojančią RANSAC algoritmą), atpažįsta plokštumas, sferas, cilindrus ir konusus.

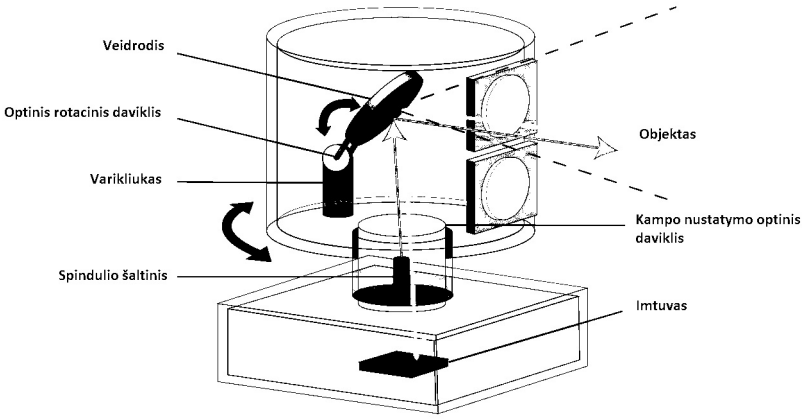
L. Maisonobe 2007 m. [11] aprašė algoritmą, su kurio dvimatėje erdvėje galima nustatyti taškus kurie, priklauso apskritimui.

2. LIDAR

2.1. LIDAR technologija

LIDAR prietaisai gali atpažinti kiekvieno išsiųsto šviesos impulso grįžtančius atspindžius, tuo pačiu metu išmatuojant atstumą iki kliūties, pasitaikiusios šviesos impulsui. LIDAR montuojamas skirtingose mobiliose platformose: lėktuvuose, traukiniuose, automobiliuose ir taip toliau. Taip pat naudojamas stacionarus prietaisas, objektų skenavimui. Mobiliosios sistemos naudojamos didelių teritorijų skenavimui. Tokiu būdu atspindžiai leidžia nustatyti objektų paviršius, plokštumas.

Mokslininkai yra sukūrę daug filtravimo ir klasifikavimo algoritmų, metodų, kurie praktikoje susiduria su problemomis dėl objektų įvairovės. Dėl to dažnai tenka naudoti rankinį koregavimą.



1 pav. LIDAR

LIDAR sudaro trys sistemos: lazerinė atstumo matavimo, globalinės padėties nustatymo ir inertinė navigacinė. Šios sistemos pateikia informaciją, kur lazerio spindulys paliečia paviršių. Taip pat yra naudojamos lazerio spindulio siuntimo ir priėmimo sistemos, kurios paviršiaus kryptimi generuoja optinį impulsą, kuris atsispindėjęs nuo paviršiaus grįžta atgal. Tiksliai matuojamas laikas nuo išsiuntimo iki grįžimo, impulsas keliauja šviesos greičiu. Impulso kelionė yra perskaičiuojama į atstumą, žinant šviesos greitį. Apskaičiavus atstumą, turint spinduliavimo kampą, globalinę padėtį ir orientaciją pagal inertinę sistemą, kiekvienam impulsui atsispindėjusiam nuo paviršiaus gaunamos x, y, z koordinatės.

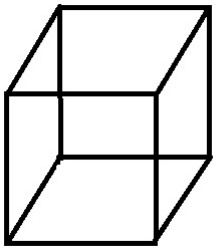
2.2. LIDAR duomenų formatas

Duomenų saugojimui yra naudojamas "LAS" failo formatas [12], jis yra viešas ir naudojamas trijų dimensijų taškų debesies duomenų mainams tarp vartotojų. Pirmiausiai buvo sukurtas LIDAR duomenų formatas mainams, šis formatas palaiko bet kokius trijų dimensijų duomenis x, y, z pavidale. Tai yra dvejetainis failo formatas.

3. Vokselis, jo pritaikymas LIDAR duomenyse

3.1. Vokselis - kas tai

Vokselis yra tūrinis elementas, reprezentuojantis trimatės erdvės elementą, kurį galima atvaizduoti trimatėje erdvėje (2 pav.). Jis yra pikselio, kuris reprezentuoja mažiausią vaizdo elementą dvimatėje erdvėje, analogas.

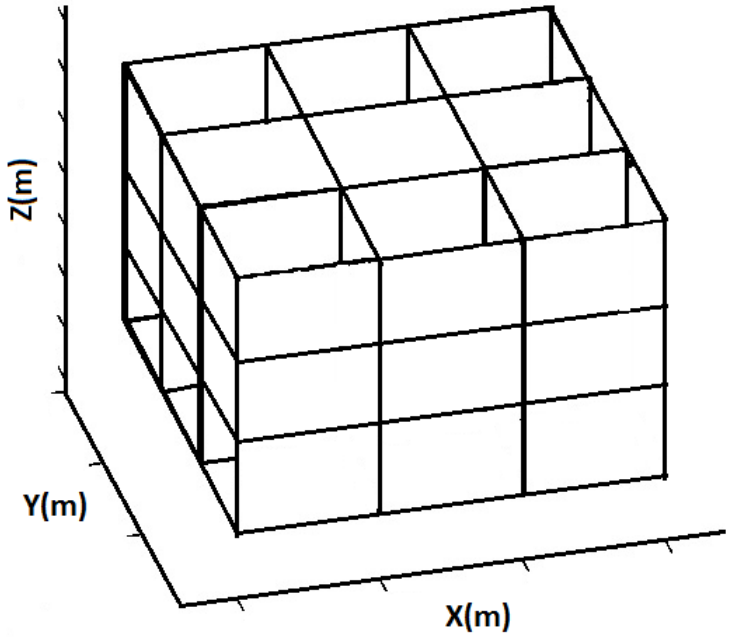


2 pav. Vokselis

Vokseliai dažniausiai naudojami vizualizavimo srityse, kompiuteriniuose žaidimuose, taip pat ten, kur būtina pavaizduoti nepakeistus tūrinius objektus (pvz., tomografijoje ir kitose moksliniuose tyrimuose). Yra žinomas vaizdo tūris, kuris išreiškiamas vokselių raiška (pvz., $n * n * n$ vokselių). Kaip ir su pikseliu, vokselio lokaciją atitinka jo fizinės koordinatės.

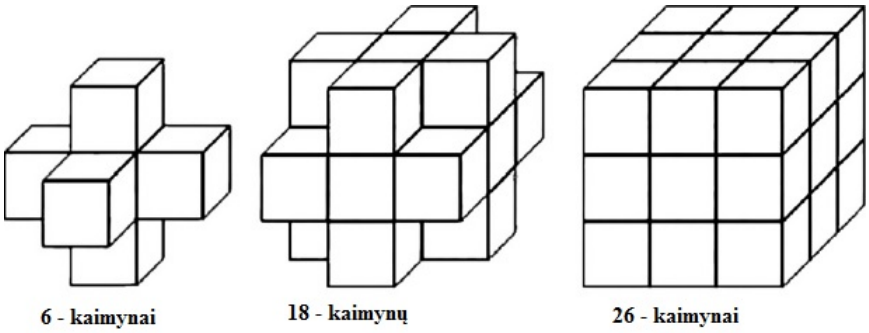
3.2. Vokselis ir LIDAR duomenys

Savo darbe vokselių sąvoka panaudosime kitaip. Turėdami taškų aibę trimatėje erdvėje, tam kad būtų lengviau apdoroti duomenis galime sumažinti jų kiekį, bei apskaičiuoti parametrus. Pasirinkus vokselio matmenis $n * n * n$ metrų (pvz. $0.5 * 0.5 * 0.5$ metrų, tokiu atveju $i = x/0.5$, $j = y/0.5$, $h = z/0.5$ ir paimame tik sveikąją skaičiaus dalį. Taškas su koordinatėmis $(5.9m., 2.9m., 2.8m.)$ priklausys vokseliui $(11, 5, 5)$.), visus taškus suskirstome į vokselius, pagal koordinates, tokiu būdu gaunamas trimatis vokselių su taškais masyvas. Programuojant toks taškų apipavidalinimas, leidžia lengviau iteruoti per taškus, skaičiuoti reikiamus atributus.



3 pav. Vokseliai

Vokselis su kaimynais pavaizduotas (4 pav.), musų atvejų kaimynais skaičiuojami tik tie vokseliai, kurie turi, bent vieną tašką.



4 pav. Vokseliai su kaimynais

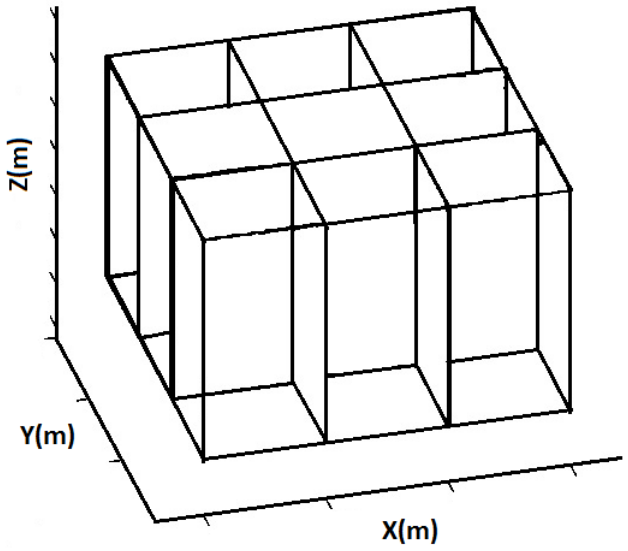
Vienas iš pritaikymo variantų turint taškus trimatėje erdvėje ir jos suskirsčius į vokselius: galime mažinti taškų skaičių paimant tik vieną tašką iš vokselio, tokiu būdu yra optimizuojami skaičiavimai, kurie dėl didelio taškų skaičiaus reikalauja daug resursų. Kitas panaudojimo būdas yra pritaikomas žemės taškų išrinkimui, šiuo atveju reikėtų imti apatinius vokselius, kurie po savimi neturi vokselių su taškais.

4. Žemės paviršiaus taškų išskyrimo metodai

Šiame skyriuje bus aprašyti du metodai žemės taškams išskirti. Tyrime naudojami filtravimo algoritmai, kurie apytiksliai išskiria 98 procentus žemės taškų. Abu metodai veikia su paruoštais duomenimis, kurių pavidalas yra taškų aibė su (x, y, z) koordinatėmis, kurių reikšmės išreikštos metrais. Abu metodai paremti minimumų išskyrimu, tik yra skirtingų implementacijų. Duotame duomenų rinkinyje iš Dekarto koordinačių sistemos reikia klasifikuoti taškus, kurie priklauso žemės paviršiui.

4.1. Taškų išskirstymas į stulpelius

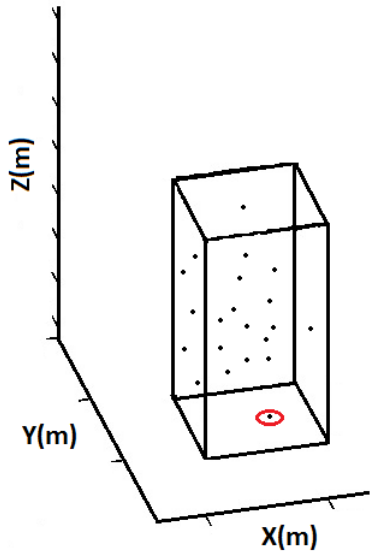
Pirmame etape reikia išskirstyti taškus į stulpelius, kurie identifikuojami pagal (i, j) indeksus. Kaip pavyzdį paimsime stulpelius, kurių pagrindas yra $0.5m.$ x $0.5m.$, tokiu atveju $i = x/0.5$, $j = y/0.5$ ir paimame tik sveikąją skaičiaus dalį. Taškas su koordinatėmis $(5.9m., 2.9m., 2.8m.)$ priklausys stulpeliui $(11, 5)$. Grafiškai pavaizdavome stulpelių modelį (5 pav.).



5 pav. Duomenų išskirtymas į stulpelius

4.2. Žemiausio taško stulpelyje algoritmas

Keisdami stulpelio pagrindo matmenis išrenkame vieną tašką stulpelyje su minimalia z reikšme (6 pav.).

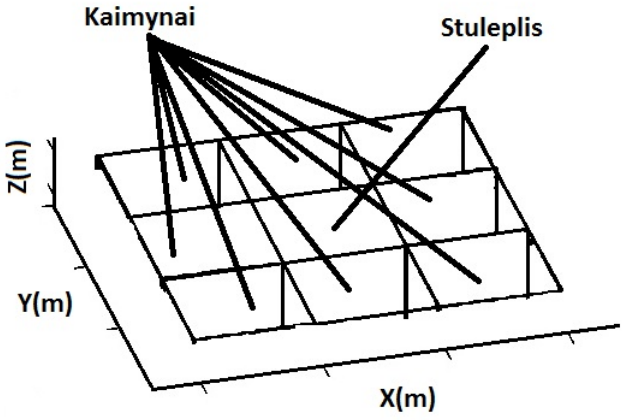


6 pav. Žemiausio taško stulpelyje radimas

Tuo pat metu skaičiuojame visų išrinktų taškų z reikšmių sumą ir kiekį, kad galėtume paskaičiuoti vidutinę z reikšmės vertę. Paskutiniame etape atmetame taškus, kurie smarkiai viršija vidutinę z reikšmę.

4.3. Žemiausio taško kaimyniniuose stulpeliuose algoritmas

Pirmame etape priskiriame visus taškus stulpeliams, kuriems jie priklauso, sekančiame žingsnyje einame per visus stulpelius ir surandame mažiausią z reikšmę iš taškų, kurie priklauso kaimyniniams stulpeliams (7 pav.).



7 pav. Stulpelis ir jo kaimynai

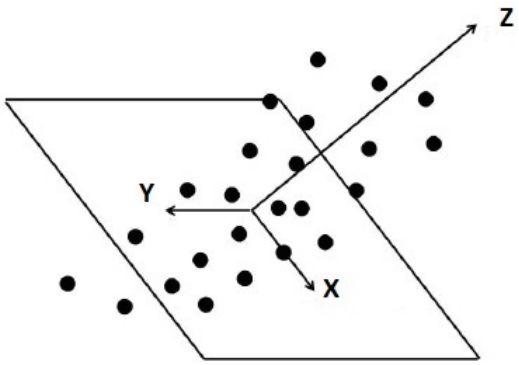
Trečiame žingsnyje išrenkame taškus iš stulpelių, kurių z reikšmė su nustatyta paklaida atitinka minimaliai z reikšmei išrinktai iš kaimyninių stulpelių taškų.

5. Cilindro taškų klasifikavimas

Šiame skyriuje aprašysime metodus, kurių pagalba galime išskirti cilindrus iš taškų debesies. Cilindrai atitinka medžių kamienus, kiekvieno medžio diametras atitinka cilindro diametrą. Aprašysime metodus, kurios naudojant galima išskirti cilindrus.

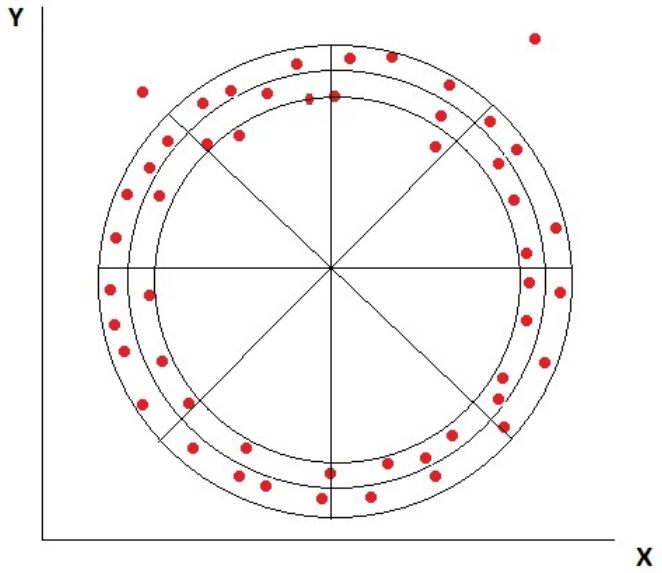
5.1. Apskritimų paieška

Šio metodo idėja yra gauti projekciją taškų plokštumoje, statmenai pagrindui. Pasirinkus tam tikrą žingsnį, kuriuo eitume z ašimi, darome projekcijas: trimatės erdvės taškų į dvimatę. (8 pav.). Tam, kad galėtume įgyvendinti šį metodą projekcijos yra keičiamos taškų sluoksniavimu, kuris detaliau pavaizduotas (14 pav.). Pasirinkus sluoksnio aukščio parametrą, pereiname visus sluoksnius ir suskirstome taškus į juos.



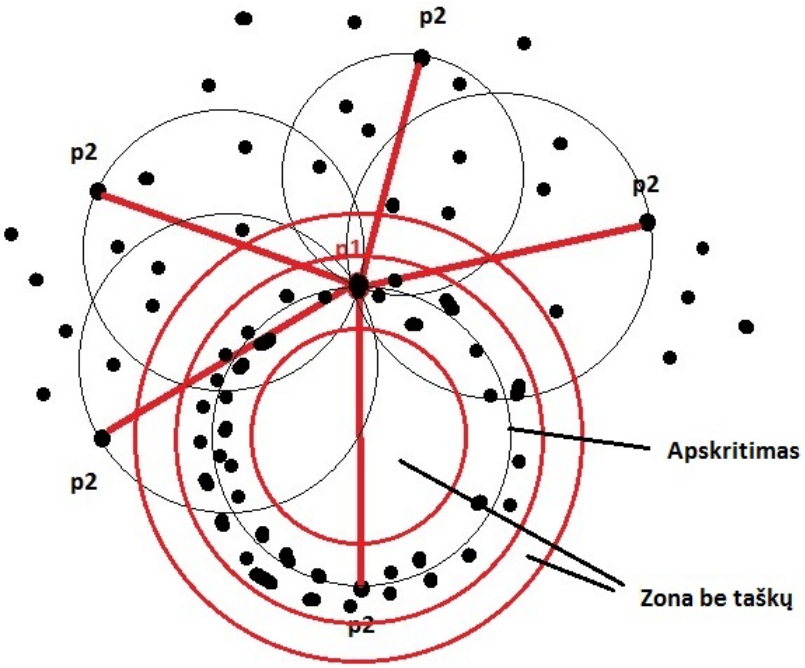
8 pav. 2-D projekcija

Antras žingsnis: pradedame ieškoti apskritimų dvimatėje erdvėje. Pasirinkę apskritimo spindulį, buferinę zoną ir dalių skaičių, kuris privalo turėti taškus, ieškome visų taškų, kurie patenka į apskritimą ir priskiriame kokiai daliai jie priklauso (9 pav).



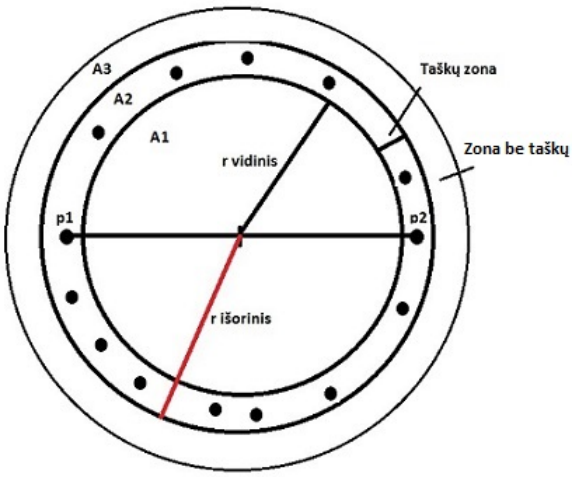
9 pav. Apskritimo paieška taškų aibėje

Pritaikant metodą teko tobulinti, todėl gavome tokį apskritimų paieškos algoritmą, kuris pritaikytas paieškai viename sluoksnyje. Pirmiausia paimami visi sluoksnio taškai, nustatomas spindulio intervalas, kuriame ieškosime apskritimų. Nustatome žingsnį, kuriuo eisime nuo pradinio iki galutinio apskritimo spindulio. Einant per visus taškus, kiekvieną tašką lyginame su visais taškais, kurie patenka į nustatytą spindulį. Tokiu būdu bandome formuoti apskritimą tarp dviejų taškų (10 pav.).



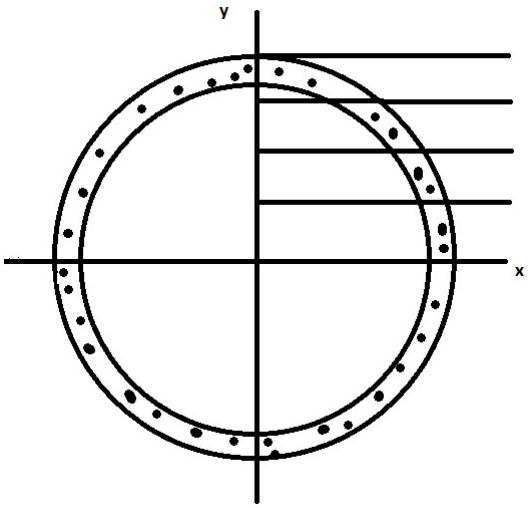
10 pav. Apskritimo paieška taškų aibėje

Apskritimo radimą detaliau pavaizdavome (11 pav.), kaip pavyzdį paimame tašką $p1$, tada ieškome taškų, kurie patenka į nurodytą spindulį ir nėra lygūs taškui $p1$, kaip pavyzdį paimame $p2$. Sekantis žingsnis yra pamatuoti atstumą tarp taško $p1$ ir $p2$, padalinius jį per pusę gaunamas spindulys nuo centro iki taško, atimame nustatytą taškų ribinę zoną ir gauname spindulį apskritimo, kuriame neturi būti taškų. Pridėjus taškų ribinę zoną gaunamas spinduly, kuriame turi būti taškai. Padauginus iš dviejų taškų ribinę zoną, gaunami apskritimo spinduliai be taškų. Pagal šiuos žingsnius turime išrinkti taškus, kurie tenkina apskritimo sąlygas.



11 pav. Apskritimo paieška taškų aibėje

Jei visos apskritimo dalys turi taškų, juos priskiriame apskritimui (12 pav.). Tam, kad būtų paprasčiau patikrinti ar radome apskritimą, tikrinimas vyksta dviem etapais. Pirmiausia apskritimą padaliname į ketvirčius, visi keturi ketvirčiai privalo turėti taškus. Antras etapas, ketvirtį padaliname į n lygių dalių pagal y ašį, kiekviena dalis privalo turėti nors vieną tašką. Tik tuomet, kai yra patenkinamos visos išvardintos sąlygos, priskiriame taškams požymį, kad jie yra apskritimo taškai.

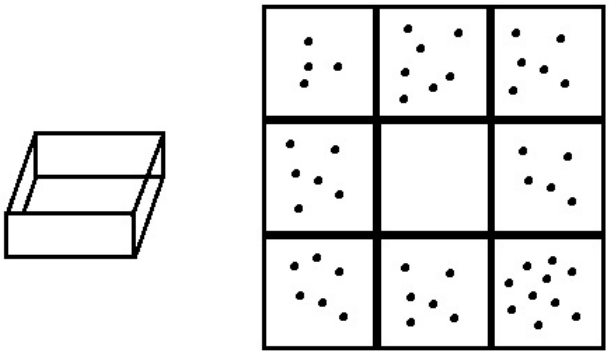


12 pav. Apskritimo indentifikavimas

Tokiu būdu yra pereinami visi sluoksniai ir išrenkami taškai, priklausantys apskritimams, tada sudėjus visus šiuos taškus į bendrą visumą, gaunamas trimatis medžių kamienų - cilindrų vaizdas.

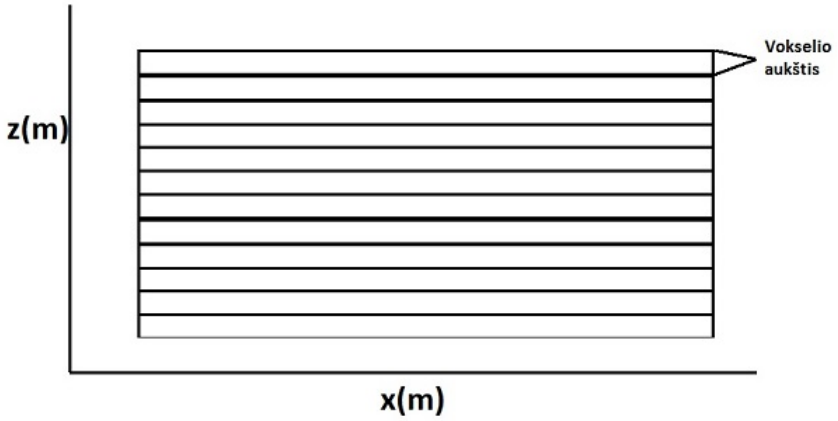
5.2. Cilindrų identifikavimas paremtas vokselio kaimynais

Sekantis metodas, padėsiantis mums identifikuoti cilindrus, bus paremtas vokseliais ir jų kaimynais (13 pav.).



13 pav. Vienas vokselis, identifikuojamas kaip cilindras

Pirmiausia nustatomi pradiniai ir galutiniai vokselių matmenys, parenkamas žingsnis, kuriuo bus didinami vokseliai, tam, kad pasiektume galutinį vokselių dydį. Vokselio aukštis imamas minimalus. Tada judame per visą sluoksnį ir padidiname vokeselio plotį, veiksmą kartojame kol pasiekiame galutinius vokselių matmenis.Tada pereiname į sekantį sluoksnį (14 pav.). Sluoksnio aukštis atitinka vokselio aukštį.



14 pav. Vokselių sluoksniai

Paieška viename sluoksnyje: ieškome tuščių vokselių, kuriuose nėra nei vieno taško, ir skaičiuojame, kiek jis turi kaimyninių vokselių su taškais (13 pav.). Kadangi LIDAR prietaisas skenuoja tik objekto paviršių, medžio kamienas būna tuščiaviduris, todėl radus tuščią vokselį, aplink kurį visi vokseliai pilni taškų galime teigti, kad tai bus medis. Kai analizuojame miško duomenis, taip didindami vokselio matmenis bandome išrinkti skirtingų diametrų kamienus.

6. Papildomi atributai

Norint geriau ištirti tam tikrus objektus, reikalingas papildomų atributų apskaičiavimas kiekvienam taškui, tam, kad būtų galima išskirti tam tikrų objektų priklausomybes nuo jų. Nustačius taško požymį (žemės paviršiaus, medžio kamieno taškas) galima bandyti surasti priklausomybes, turint papildomus atributus.

Žemiau pateikas atributų sąrašas, kuriuos galima apskaičiuoti kiekvienam taškui:

- Stulpelio, kuriame randasi taškas mažiausia z reikšmė;
- Kaimyninių stulpelių mažiausia z reikšmė;
- Stulpelio, kuriame randasi taškas vidutinė z reikšmė;
- Vokselio, kuriame randasi taškas mažiausia z reikšmė;
- Vokselio, kuriame randasi taškas vidutinė z reikšmė;
- Vokselio, kuriame randasi taškas kaimynų skaičius;

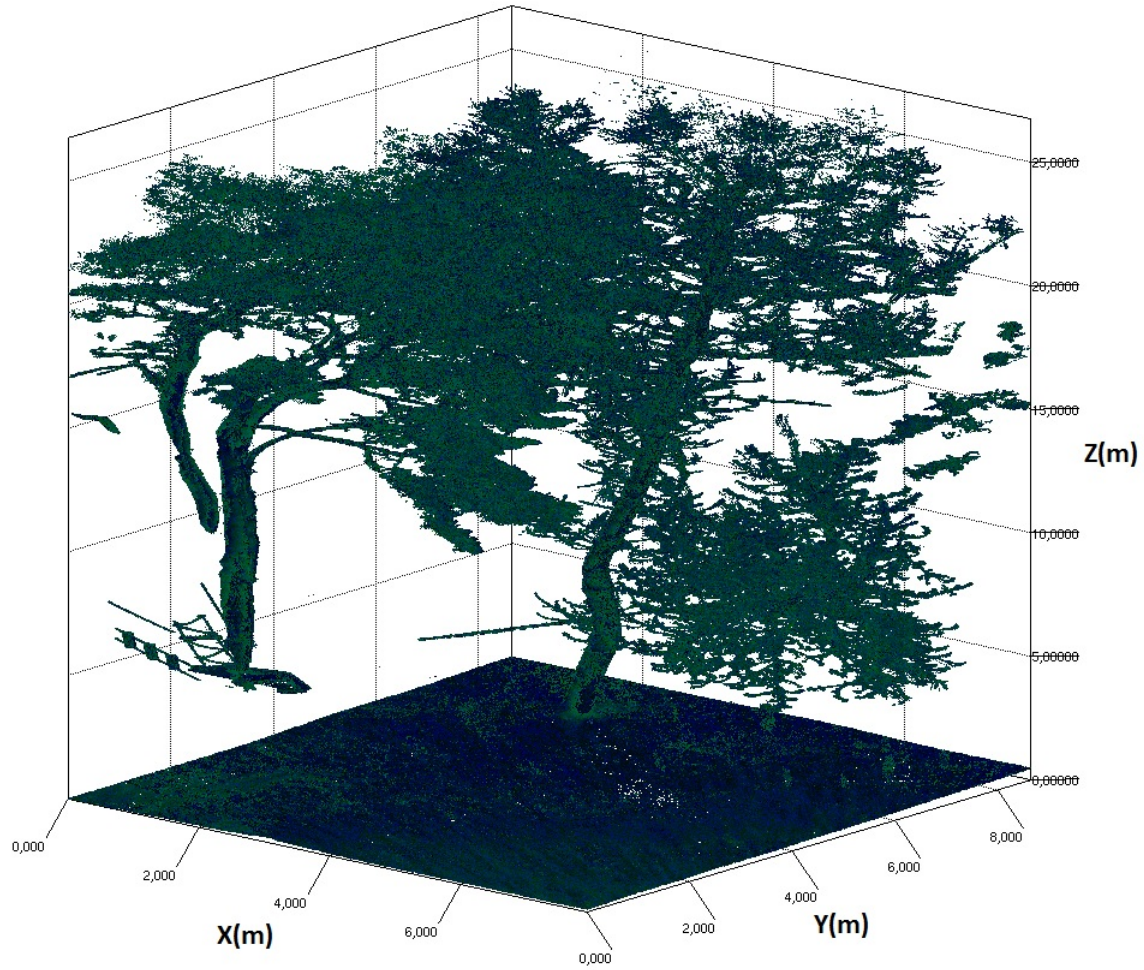
7. Eksperimentinė dalis

7.1. Pradiniai duomenys

Eksperimentiniai duomenys buvo gauti nuskenavus aplinką pasinaudojant stacionariu LIDAR prietaisu. Buvo nuskenuota "UNO PARK" teritorija ir gauta apie 800 milijonų taškų. Aprašytų metodų analizei buvo iškirpti du nedideli plotai: pirmasis su keliais medžiais (15 pav.), antrasis tankesnis su daug medžių (16 pav.) ir sukonvertuoti į tinkamą formatą su koordinatėmis (x, y, z), tam buvo panaudotas įrankis "LAStools" [13] . Pasinaudojus "java" programavimo kalba ir biblioteka "jzy3d" atvaizdavome pradinius duomenis.

7.1.1. Pirmasis taškų rinkinys

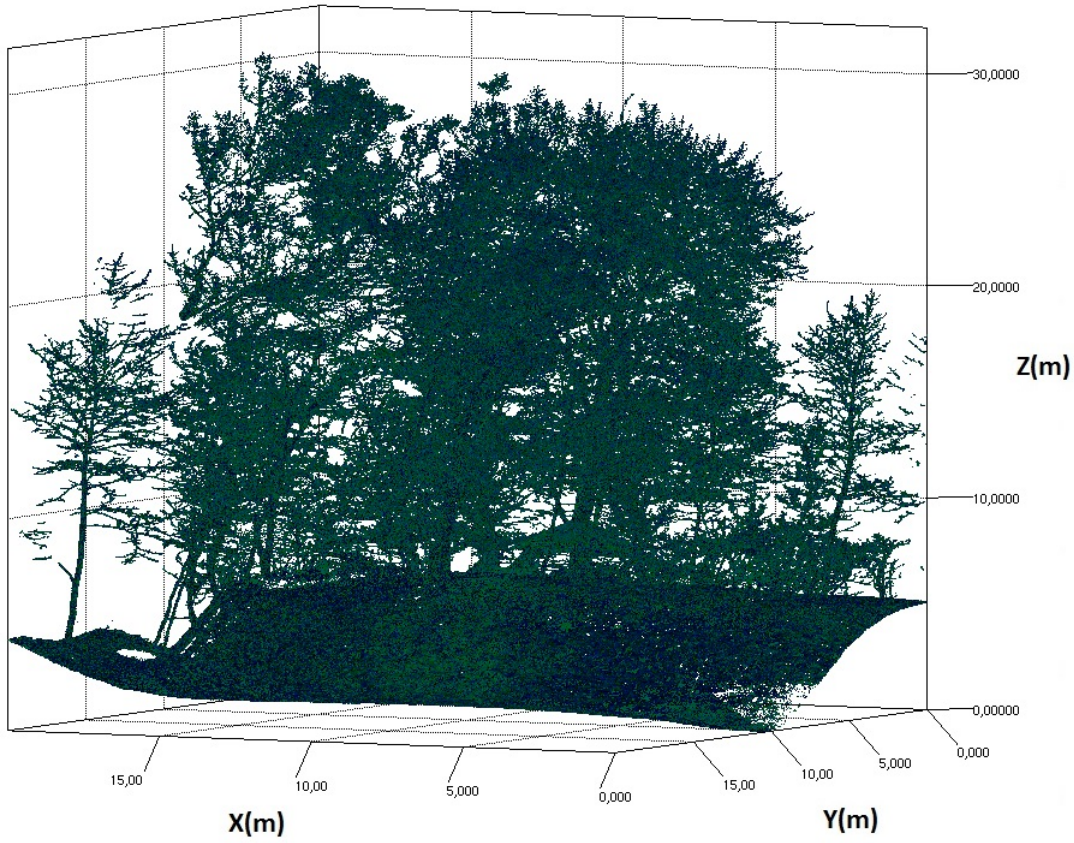
Žemiau pavaizduoti duomenys (15 pav.) paimti iš bendro taškų rinkinio. Buvo iškirptas plotas, kurio matmenys 8 m. x 8 m., jį sudaro 4335041 taškai.



15 pav. Pradiniai duomenys - pirmasis taškų rinkinys.

7.1.2. Antrasis taškų rinkinys

Žemiau pavaizduotas antrasis taškų rinkinys (16 pav.). Jo matmenys yra 20 m. x 20 m., jį sudaro 3510306 tašai.

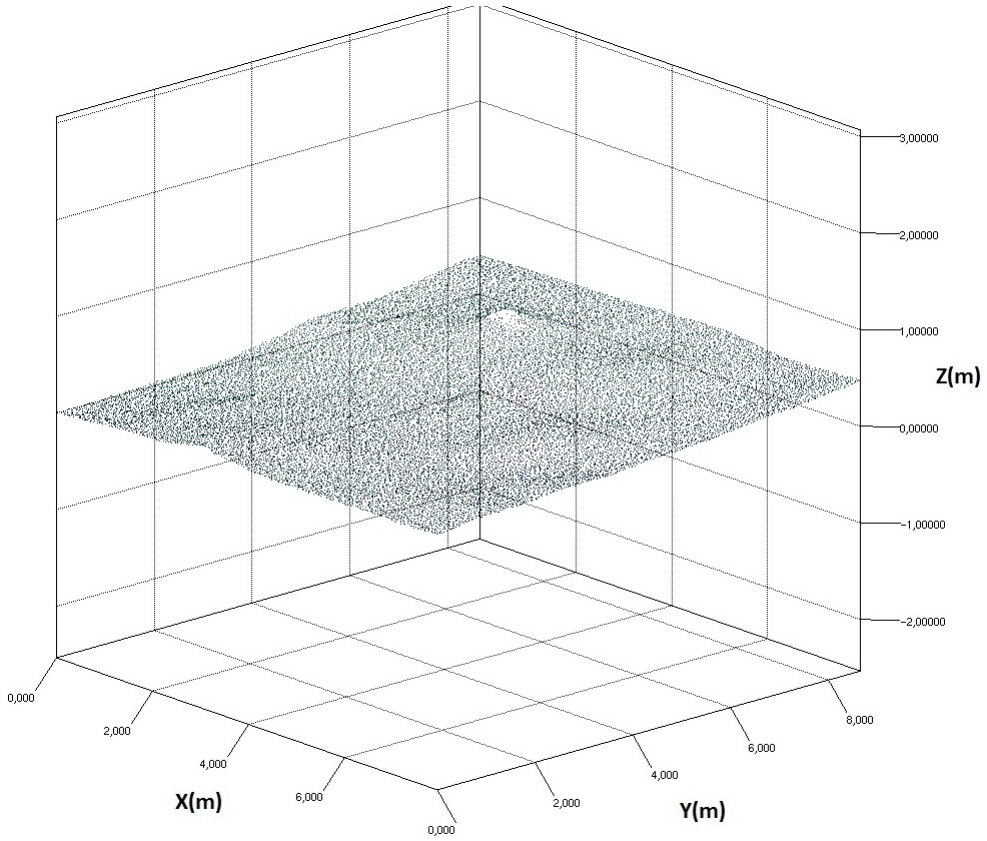


16 pav. Pradiniai duomenys - antrasis taškų rinkinys.

7.2. Žemės paviršiaus taškų išskyrimas

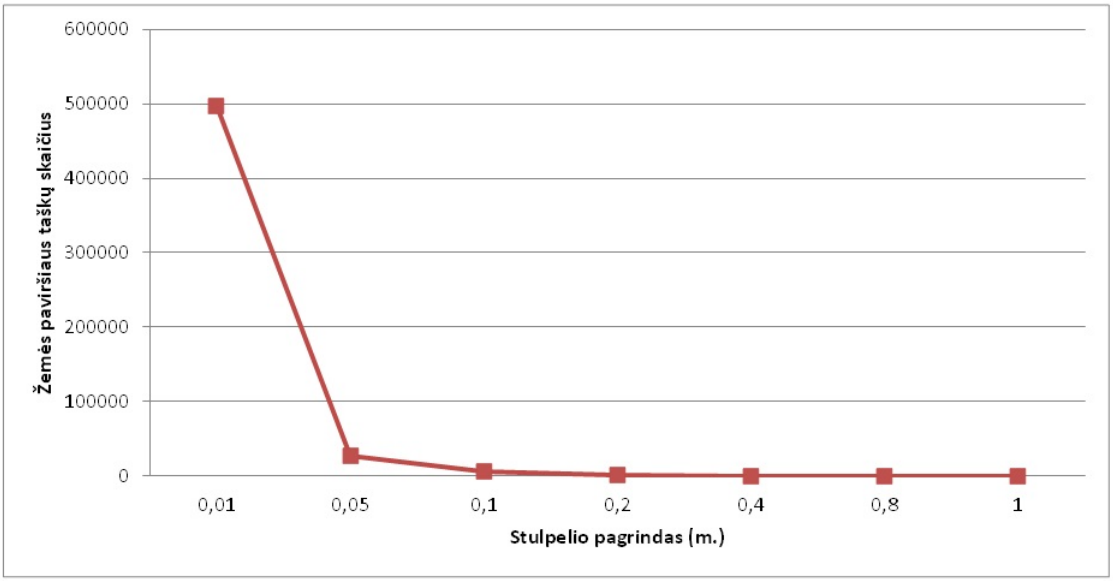
7.2.1. Žemės paviršiaus taškų išskyrimas pagal žemiausio taško stulpelyje algoritmą

Pasinaudojus "java" programavimo kalba įgyvendinome metodą, kuris išrenka žemės paviršiaus taškus remdamasis žemiausio taško stulpelyje algoritmu. Pirmojo taškų rinkinio rezultatas pavaizduotas (17 pav.), iš 4335041 taškų buvo išrinkti 27596 taškai priklausantys žemės paviršiui. Stulpelio pagrindas buvo pasirinktas $0.05m.$ x $0.05m.$. Algoritmo vykdymas užtruko 0,338 sekundės.



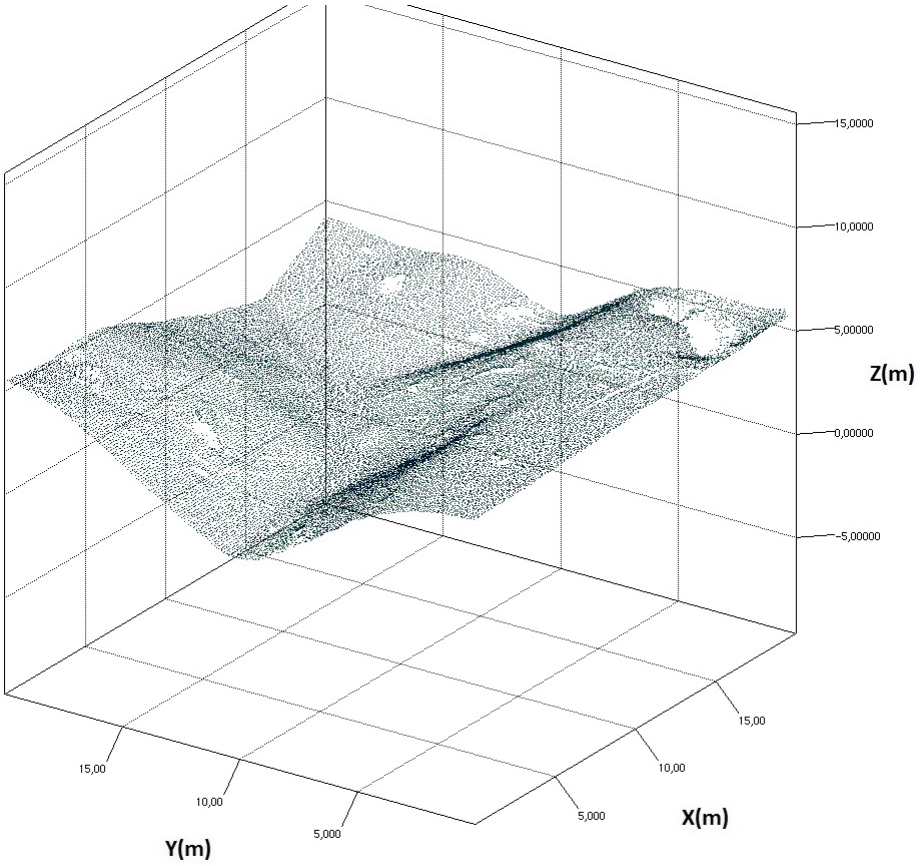
17 pav. Žemės paviršiaus taškai išrinkti iš pirmojo taškų rinkinio, pasinaudojus žemiausio taško stulpelyje algoritmu.

Atlikome analizę, kaip žemės paviršiaus taškų skaičius priklauso nuo stulpelio pagrindo pločio, kai taškai išrenkami naudojant žemiausio taško stulpelyje algoritmą. Gautą rezultatą pavaizdavome (18 pav.) grafike. Stulpelio pagrindo plotis paimtas nuo 0.01 m. iki 1 m. Metodas didžiausią žemės paviršiaus taškų kiekį išrenka su mažiausiu stulpelio pločiu, po to pastebimas staigus išrenkamų taškų kiekio mažėjimas.



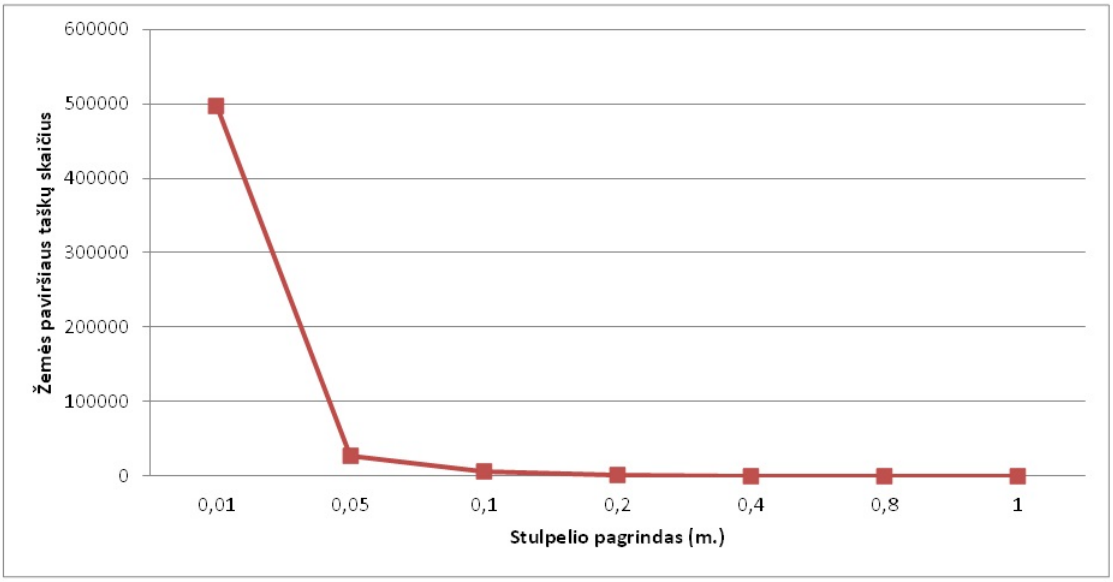
18 pav. Žemės paviršiaus taškų išrinktų iš pirmojo taškų rinkinio, pasinaudojus žemiausio taško stulpelyje algoritmu prilausomybė nuo stulpelio pagrindo pločio.

Tą patį algoritmą pritaikėme antrajam taškų rinkiniui, rezultatas pavaizduotas (19 pav.), iš 3510306 taškų buvo išrinkti 38816 taškai priklausantys žemės paviršiui. Stulpelio pagrindas buvo pasirinktas $0.1m.$ x $0.1m.$. Algoritmo vykdymas užtruko 0,163 sekundės.



19 pav. Žemės paviršiaus taškai išrinkti iš antrojo taškų rinkinio, pasinaudojus žemiausio taško stulpelyje algoritmu.

Analizę, kuomet žemės paviršiaus taškų skaičius priklauso nuo stulpelio pagrindo pločio, kai taškai išrenkami naudojant žemiausio taško stulpelyje algoritmą, padarėmė su taškais iš antrojo duomenų rinkinio. Gautą rezultatą pavaizdavome (20 pav.) grafike.

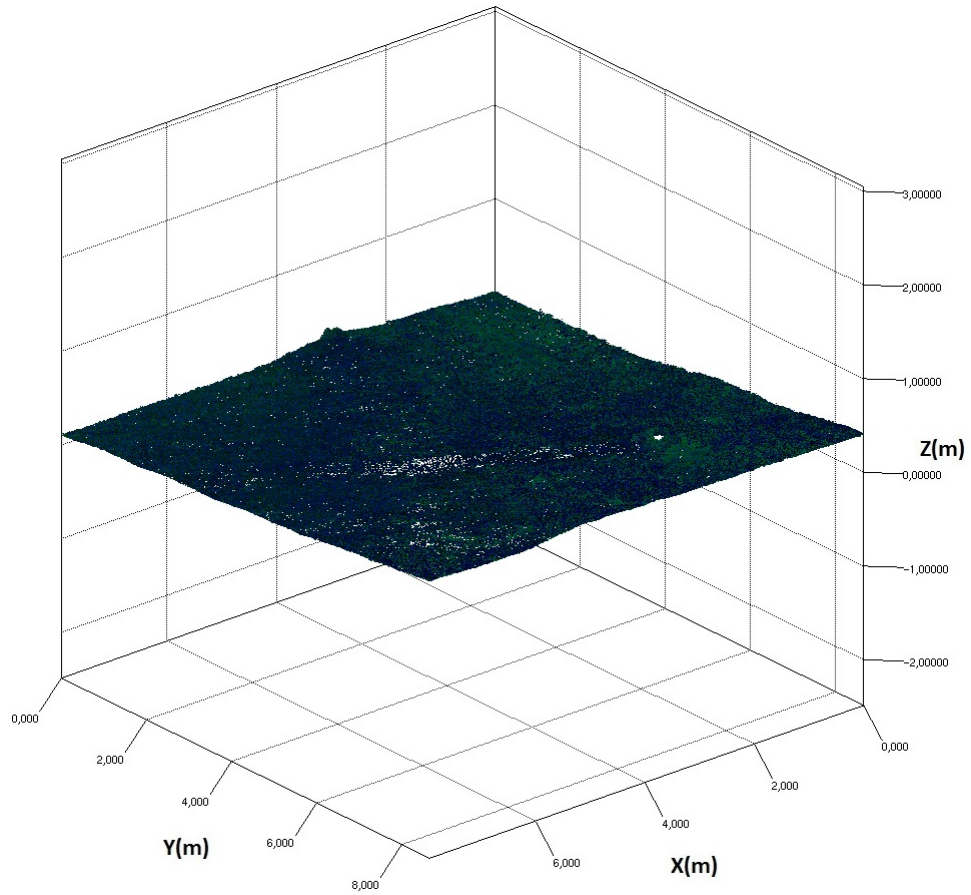


20 pav. Žemės paviršiaus taškų išrinktų iš antrojo taškų rinkinio, pasinaudojus žemiausio taško stulpelyje algoritmu prilausomybė nuo stulpelio pagrindo pločio.

Stulpelio pagrindo plotis paimtas nuo 0.01 m. iki 1 m., didžiausią žemės paviršiaus taškų kiekį išrenka su mažiausiu stulpelio pločiu. Gaunama išvada, jog skirtingi duomenys nedaro įtakos išrenkant žemės paviršiaus taškus. Metodo rezultatas priklauso nuo stulpelio pločio.

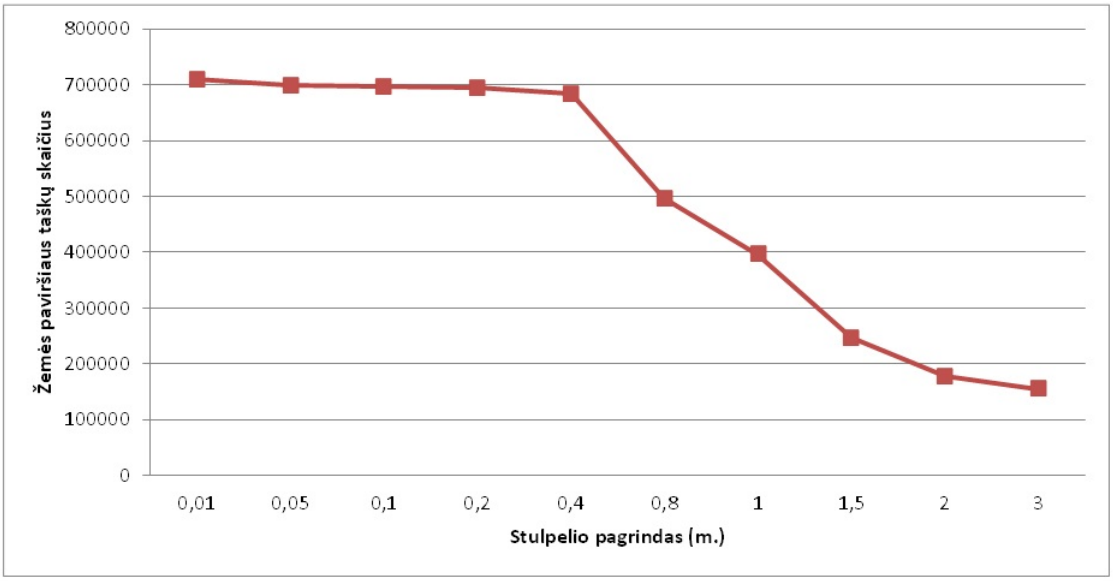
7.2.2. Žemės paviršiaus taškų išskyrimas pagal žemiausio taško kaimyniniuose stulpeliuose algoritmą

Antrajame bandyme pasitelkus "java" programavimo kalbą įgyvendinamas algoritmas išrenkantis žemės paviršiaus taškus pagal žemiausio taško kaimyniniuose stulpeliuose metodą. Rezultatas gautas išrinkus taškus iš pirmojo taškų rinkinio pavaizduotas (21 pav.), iš 4335041 buvo išrinkti 698623 taškai priklausantys žemės paviršiui. Stulpelio pagrindas buvo pasirinktas $0.05m$ x $0.05m$. Algoritmo vykdymas užtruko 1,552 sekundės.



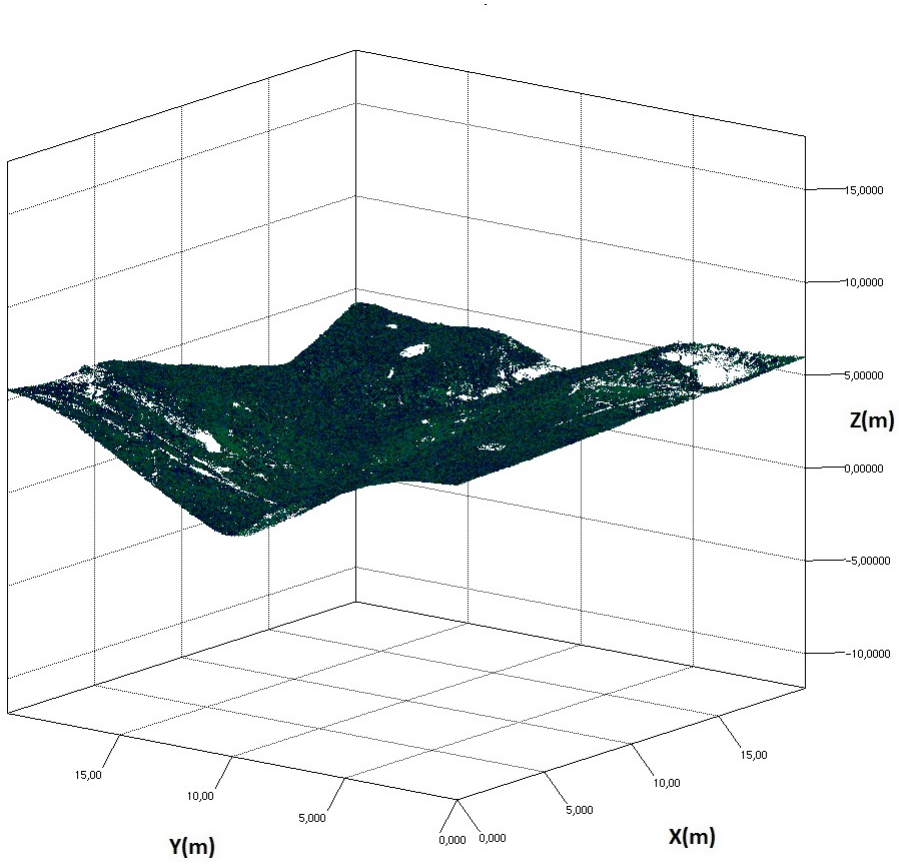
21 pav. Žemės paviršiaus taškai išrinkti iš pirmojo taškų rinkinio, pasinaudojus žemiausio taško kaimyniniuose stulpeliuose algoritmu.

Ištyrėme išrenkamų taškų skaičiaus priklausomybę nuo stulpelio pločio. Gautus rezultatus pavaizdavome (22 pav.)



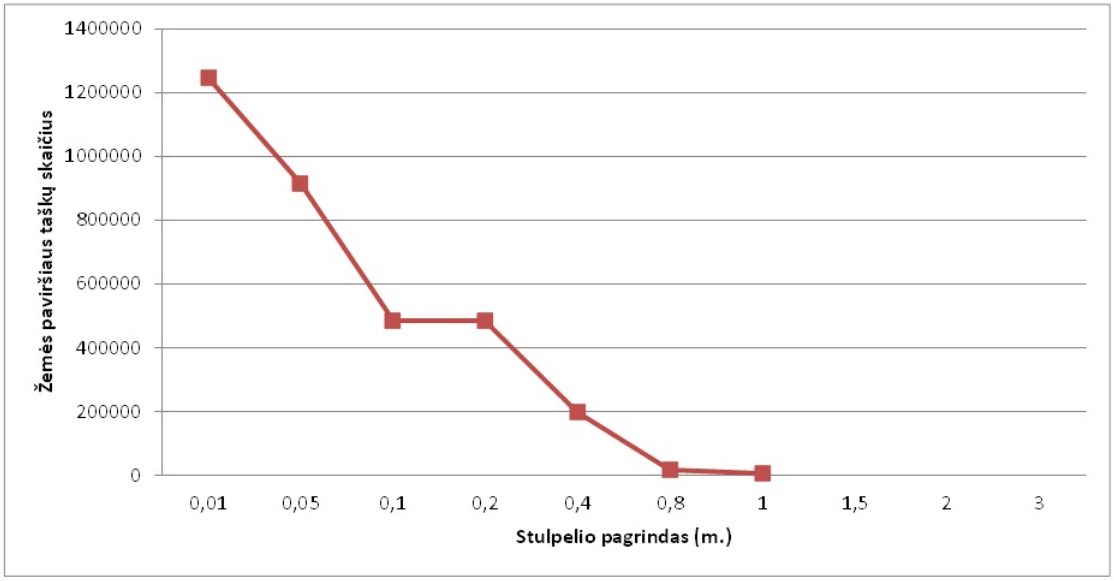
22 pav. Žemės paviršiaus taškų išrinktų iš pirmojo taškų rinkinio, pasinaudojus žemiausio taško kaimyniniuose stulpeliuose algoritmu prilausomybė nuo stulpelio pagrindo pločio.

Rezultatas gautas išrinkus taškus iš antrojo taškų rinkinio pavaizduotas (23 pav.), iš 3510306 buvo išrinkti 992818 taškai priklausantys žemės paviršiui. Stulpelio pagrindas buvo pasirinktas $0.1m$ x $0.1m$. Algoritmo vykdymas užtruko 0,718 sekundės.



23 pav. Žemės paviršiaus taškai išrinkti iš antrojo taškų rinkinio, pasinaudojus žemiausio taško kaimyniniuose stulpeliuose algoritmu.

Įvertinome išrenkamų taškų skaičiaus priklausomybę nuo stulpelio pločio tyrimą. Gautus rezultatus pavaizdavome (24 pav.)



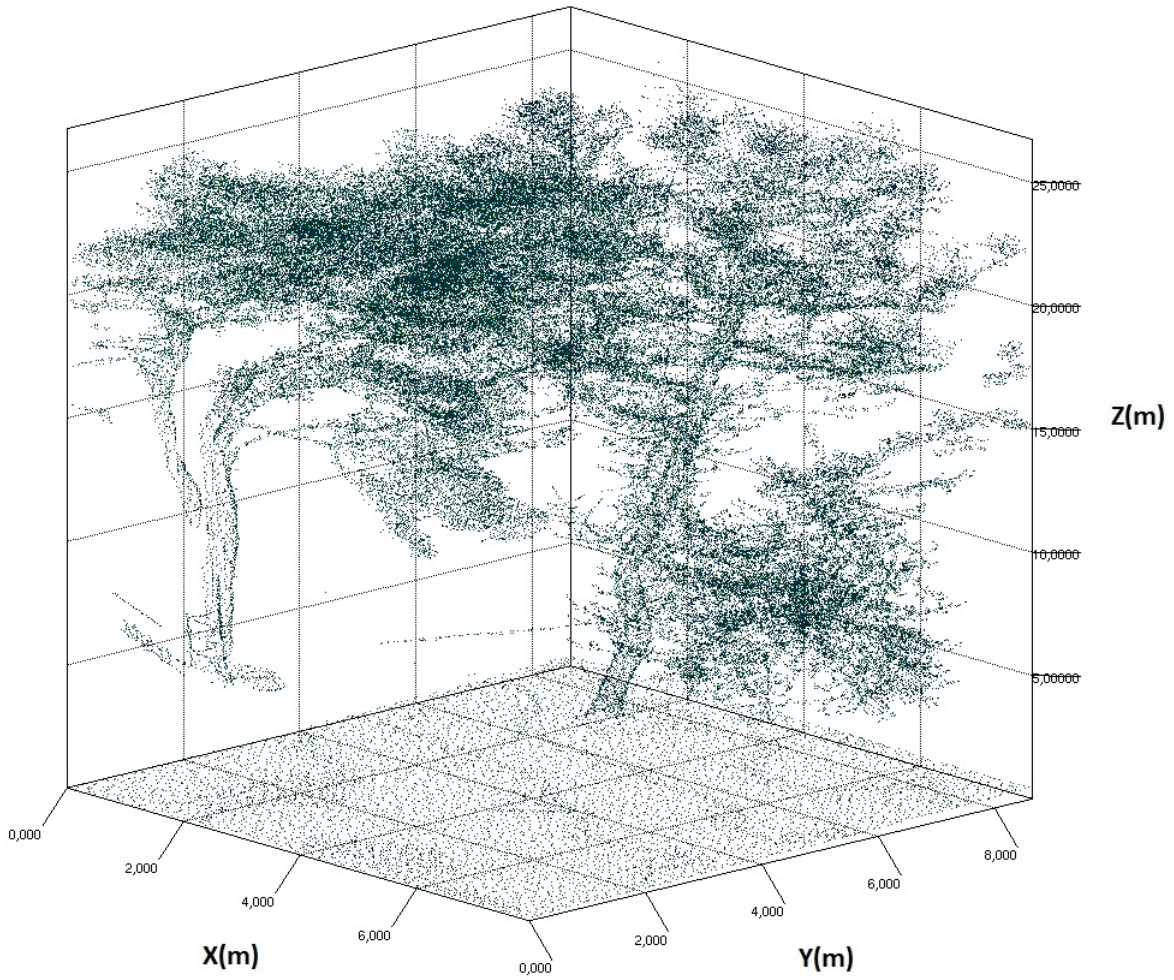
24 pav. Žemės paviršiaus taškų išrinktų iš antrojo taškų rinkinio, pasinaudojus žemiausio taško kaimyniniuose stulpeliuose algoritmu prilausomybė nuo stulpelio pagrindo pločio.

Lyginant abu žemės taškų išskyrimo metodus paaiškėjo, kad pirmasis metodas, kuris išrenka žemiausią tašką iš stulpelio yra gana paprastas, todėl nereikalauja daug resursų, jo vykdymo laikas yra trumpas. Tačiau jis išrenka ne visus žemės paviršiaus taškus. Antrasis metodas, kuris išrenka žemės paviršiaus taškus, atsižvelgiant į kaimyninius stulpelius, yra tikslesnis. Jo pagalba galima išrinkti visus taškus priklausančius žemės paviršiui, bet jo vykdymas užtruko ilgiau. Visi skaičiavimai atlikti nudojantis "Intel Core i7-4510U CPU" procesoriumi su taktiniu dažniu 2.60 GHz ir 8.192 Gb operatyviosios atminties.

7.3. Taškų optimizavimas pasinaudojant vokseliais

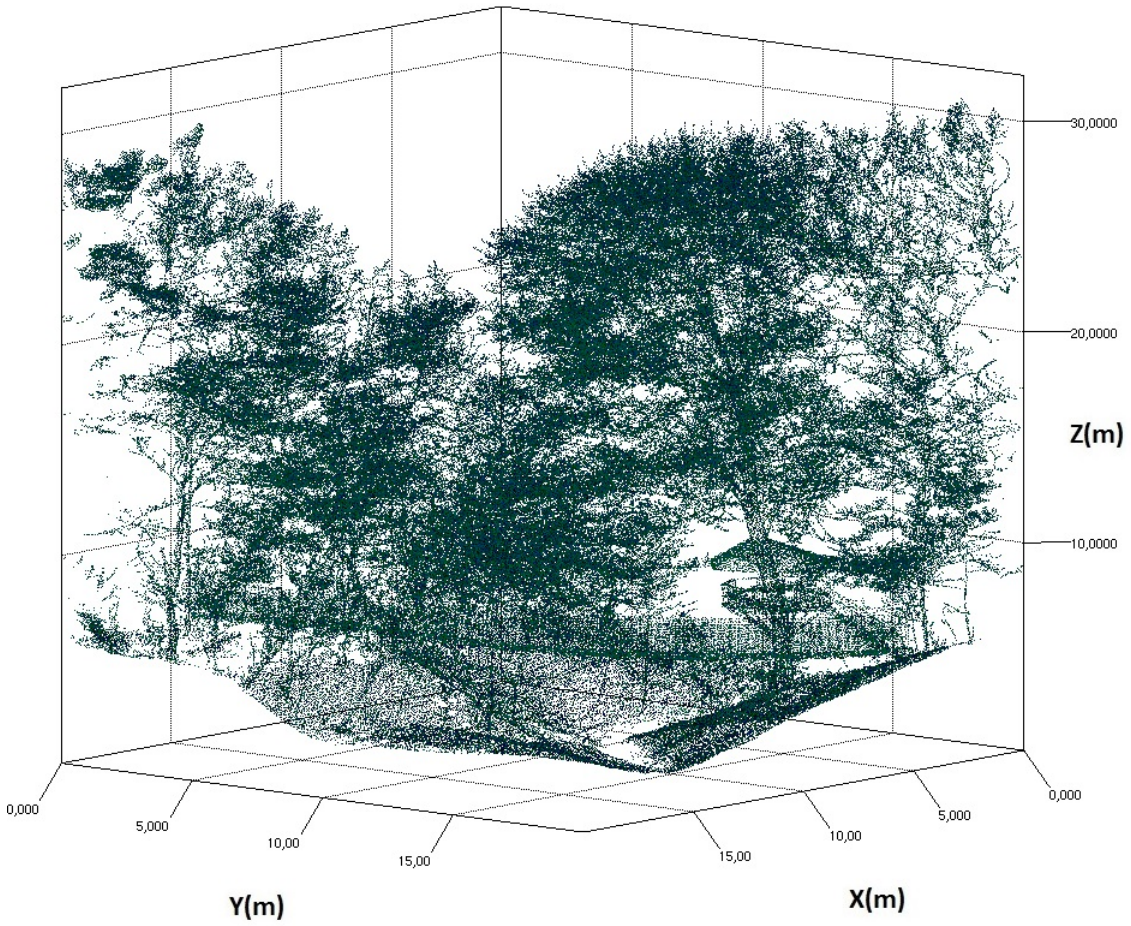
Išdėliojus taškus pagal vokselius atsiranda galimybė sumažinti taškų skaičių neprarandant bendro vaizdo. Nusistačius parametrus, kokių matmenų bus vokselis, judame per visus taškus ir skirstome juos į vokselius pagal taškų koordinates, o galiausiai išrenkame tik vieną tašką iš jo.

Pasinaudojus šia savybe atlikome duomenų optimizaciją pirmajam taškų rinkiniui. Iš 4335041 buvo išrinkti 124308 taškai, vokselio matmenys $0.01m$ x $0.01m$ x $0.01m$, paimta po vieną tašką. Apytiksliai 30 kartų sumažinome taškų kiekį. Rezultatas pavaizduotas (23 pav.).



25 pav. Pradiniai duomenys: pirmasis taškų rinkinys, pritaikius optimizaciją.

Tą patį veiksmą, su tais pačiais parametrais vokselio matmenys $0.01m$ x $0.01m$ x $0.01m$ pritaikėme antrajam taškų rinkiniui, iš 3510306 taškų liko tik 446873 taškai.



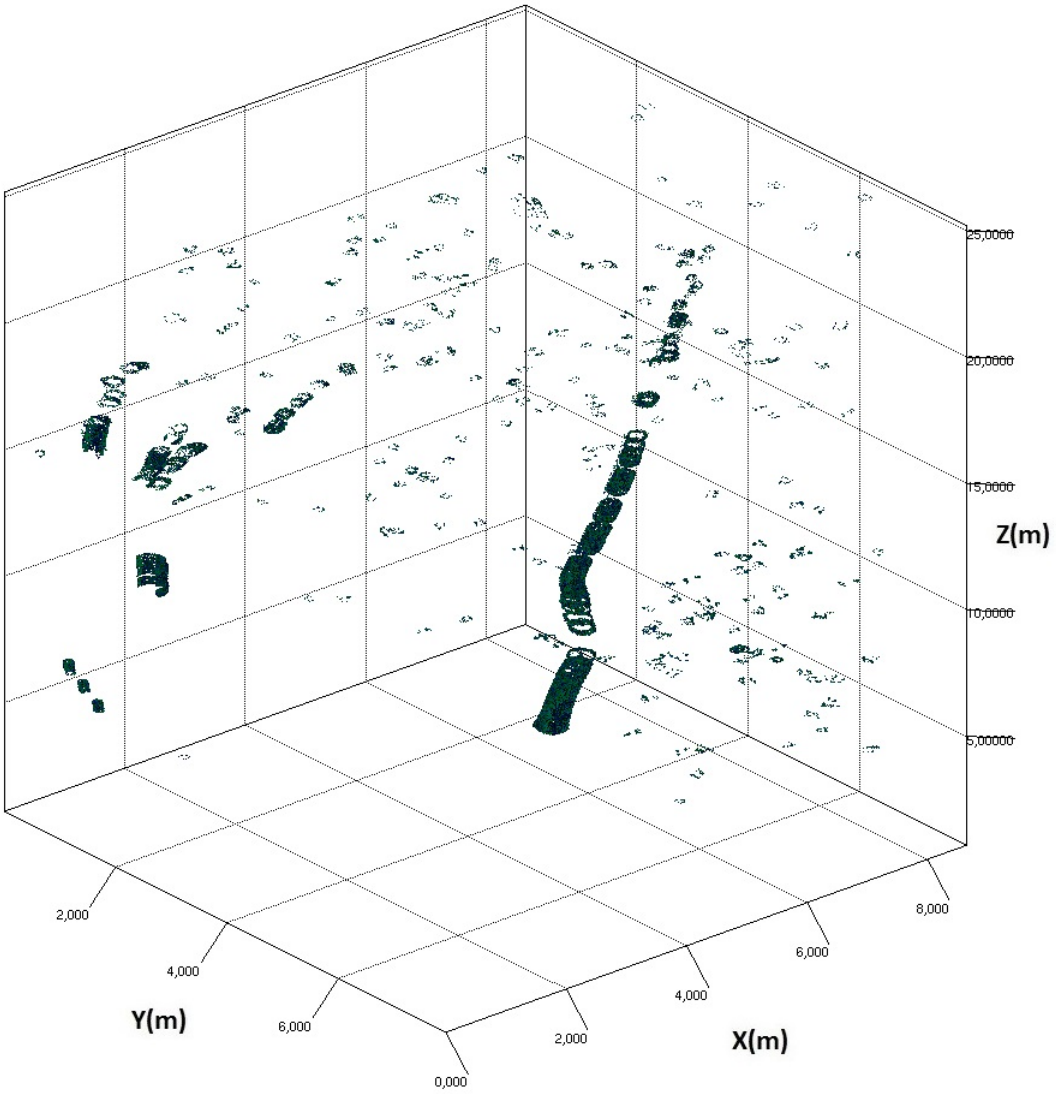
26 pav. Pradiniai duomenys: antrasis taškų rinkinys, pritaikius optimizaciją.

Padarius duomenų optimizaciją pastebima, kad galima mažinti taškų skaičių ir neprarasti bendro vaizdo, todėl norint apdoroti didelį kiekį duomenų galima naudoti tokią optimizaciją.

7.4. Medžio kamieno išskyrimas

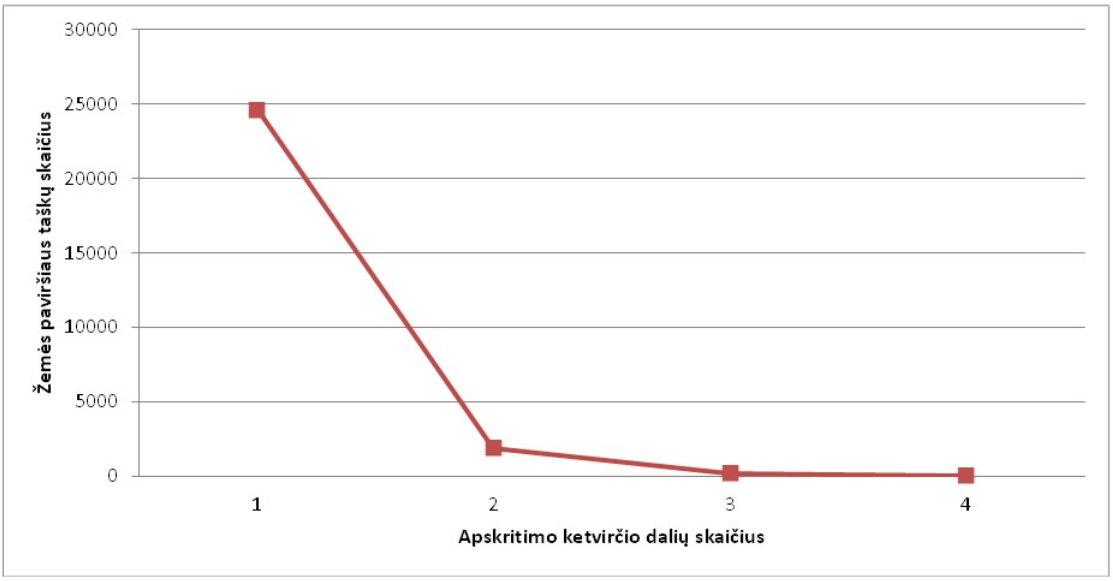
7.4.1. Medžio kamieno išskyrimas naudojant apskritimų paieškos metodą

Pasinaudojus "java" programavimo kalba įgyvendinome metodą, kuris išrenka medžių kamienų taškus, remdamasis apskritimų paieškos metodu. Sluoksnio aukštis buvo pasirinktas $0.1m$.. Maksimalus apskritimo skersmuo $0.5m$.. Apskritimo ketvirtis padalintas į 2 dalis. Rezultatas gautas iš pirmojo taškų rinkinio, pavaizduotas (27 pav.), iš 727548 taškų buvo išrinkti 57832 taškai priklausantys medžiams. Algoritmo vykdymas užtruko 231 sekundę.



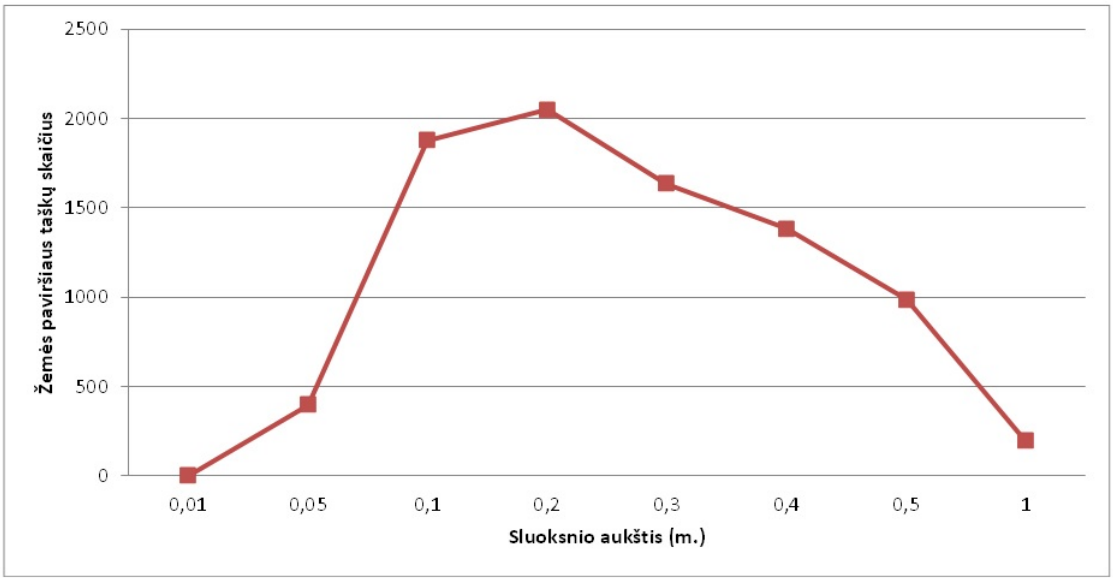
27 pav. Medžių kamienai iš pirmojo taškų rinkinio.

Atlikome tyrimą su optimizuotu pirmuoju duomenų rinkiniu. Bendrų taškų skaičius 132906, tyrėme medžio kamieno išskyrimo metodą, paremtą apskritimų radimu. Randamą medžių kamienų taškų priklausomybę nuo apskritimo ketvirčio dalių skaičiaus pavaizdavome (28 pav.), nuo sluoksnio auksčio (29 pav.) ir nuo maksimalaus apskritimo skersmens (30 pav.).



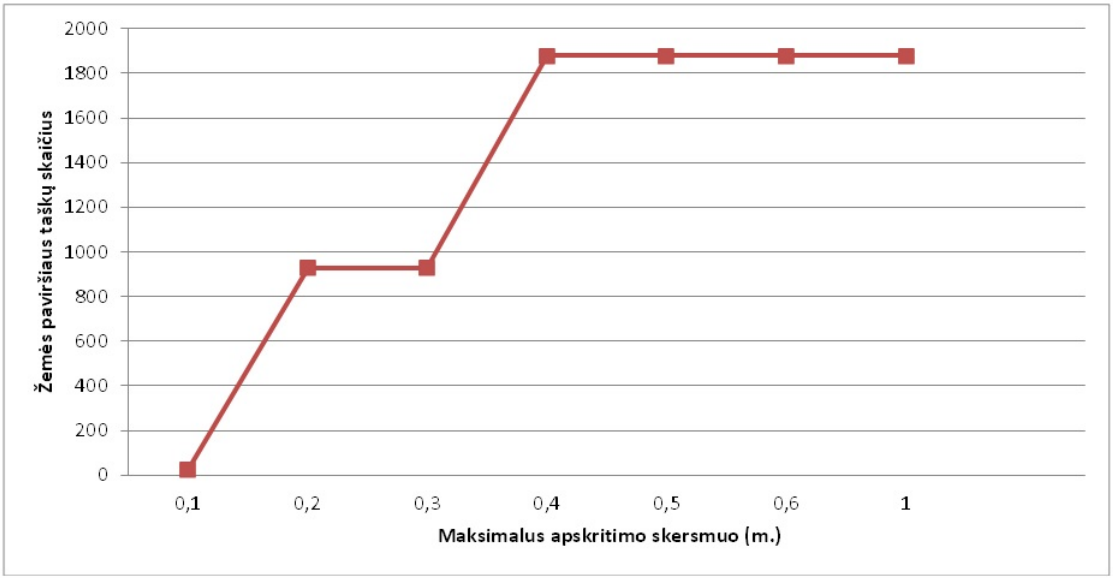
28 pav. Medžių kamienų iš pirmojo optimizuoto taškų rinkinio, išrenkamų taškų priklausomybė nuo apskritimo ketvirčio dalių skaičiaus

Remiantis (28 pav.) grafiku galime pasakyti, kad išrenkamų taškų skaičius priklauso nuo ketvirčio dalių skaičiaus. Tačiau norint gauti kokybišką apskritimą tą skaičių turime didinti, todėl tikrinant tik ketvirčius, kai paimta tik 1 dalis, išrenkama daugiausia taškų. Susiduriama su problema: išrenkami ne tik apskritimai, todėl norint gauti tikslų apskritimą, reikia ketvirtį padalinti bent į dvi dalis.



29 pav. Medžių kamienų iš pirmojo optimizuoto taškų rinkinio, taškų priklausomybė nuo sluoksnio auksčio

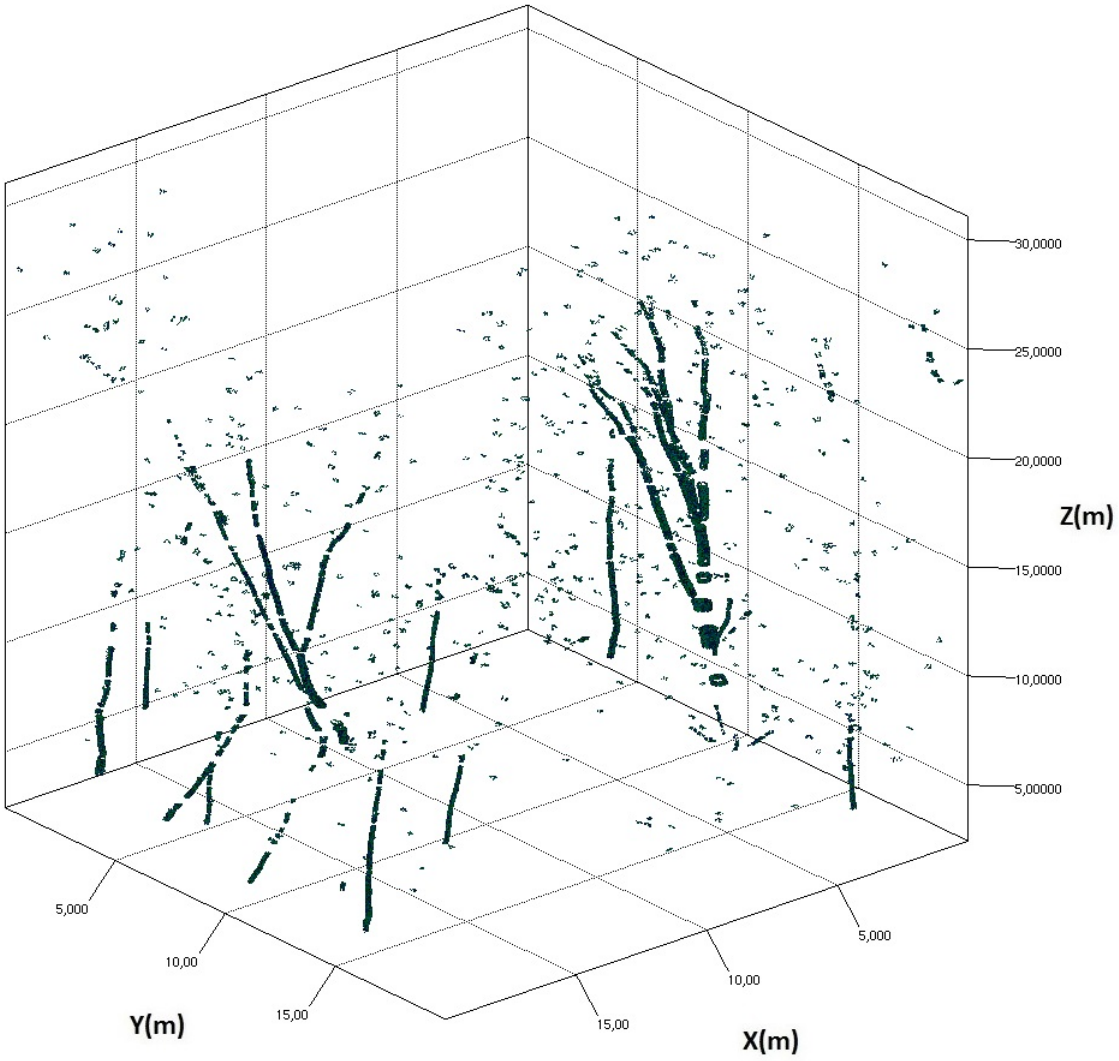
Remiantis (29 pav.) galime teigti, jog medžio kamienas tiksliausiai išrenkamas, kai sluoksnio aukštis yra 0.1 m. - 0.2 m..



30 pav. Medžių kamienų iš pirmojo optimizuoto taškų rinkinio, išrenkamų taškų priklausomybė nuo maksimalaus apskritimo skersmens

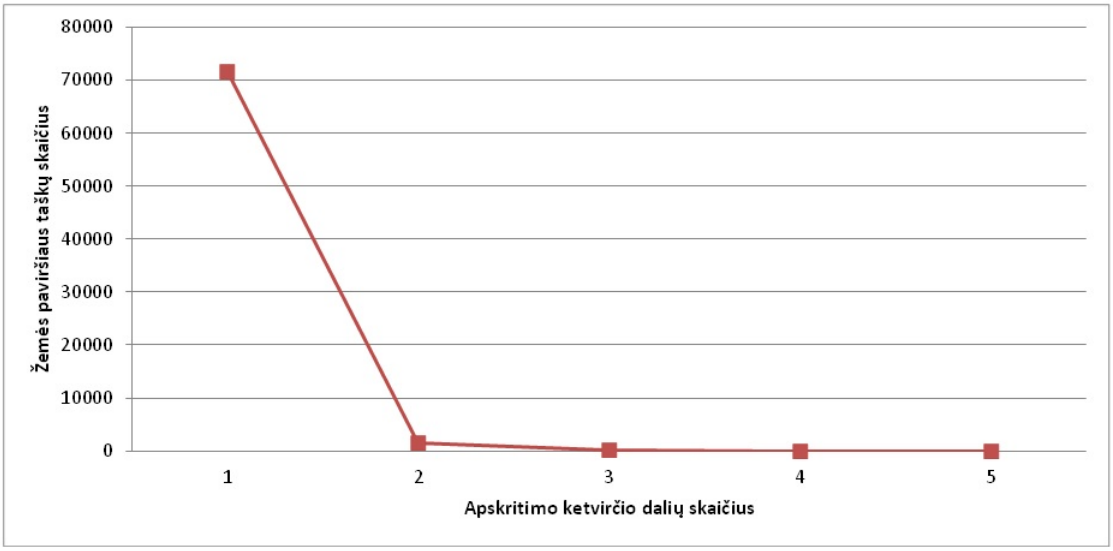
Grafike (30 pav.) pastebime, kad didinat maksimalų apskritimo skersmenį, pasiekiamas maksimalus taškų išrinkimas, ir toliau didnant maksimalų apskritimo skresmenį išrenkamas vienodas taškų skaičius, todėl norint išrinkti kuo daugiau medžio kamieno taškų, maksimalų apskritimo skersmens parametrą reikėtų rinktis ne mažesnį kaip 0.4 m.

Rezultatas gautas iš antrojo taškų rinkinio pavaizduotas (31 pav.), iš 2442581 taškų buvo išrinkti 135835 taškai priklausantys medžiams. Algoritmo vykdymas užtruko 188 sekundžių.

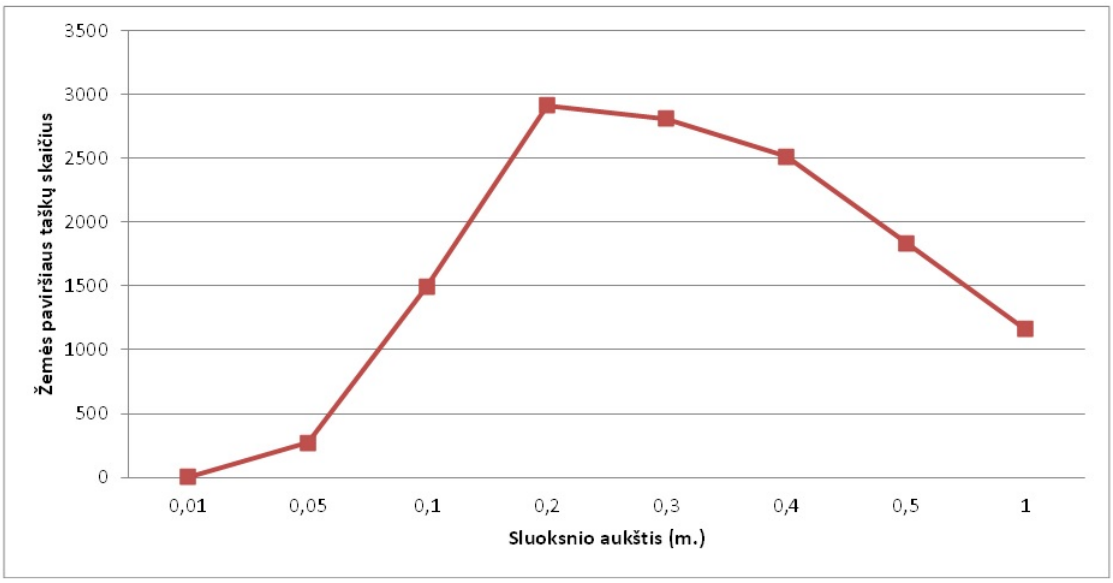


31 pav. Medžių kamienai iš anrojo taškų rinkinio.

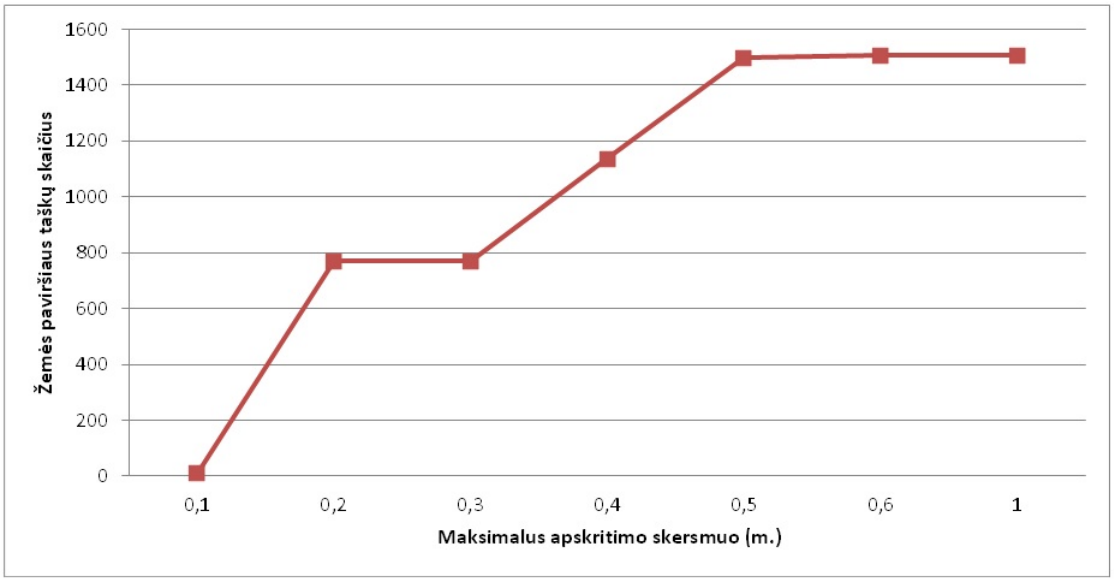
Atlikome tyrimą su optimizuotu antruoju duomenų rinkiniu, bendrų taškų skaičius 480377, tyrėme medžio kamieno išskyrimo metodą, paremtą apskritimų radimu. Randamą medžių kamienų taškų priklausomybę nuo apskritimo ketvirčio dalių skaičiaus pavaizdavome (32pav.), nuo sluoksnio auksčio (33 pav.) ir nuo maksimalaus apskritimo skersmens (34 pav.). Pastebima, jog nepriklausomai nuo duomenų įvairovės grafikų priklausomybių tendencijos nesikeičia.



32 pav. Medžių kamienų iš antrojo optimizuoto taškų rinkinio, taškų priklausomybė nuo apskritimo ketvirčio dalių skaičiaus



33 pav. Medžių kamienų iš antrojo optimizuoto taškų rinkinio, taškų priklausomybė nuo sluoksnio auksčio

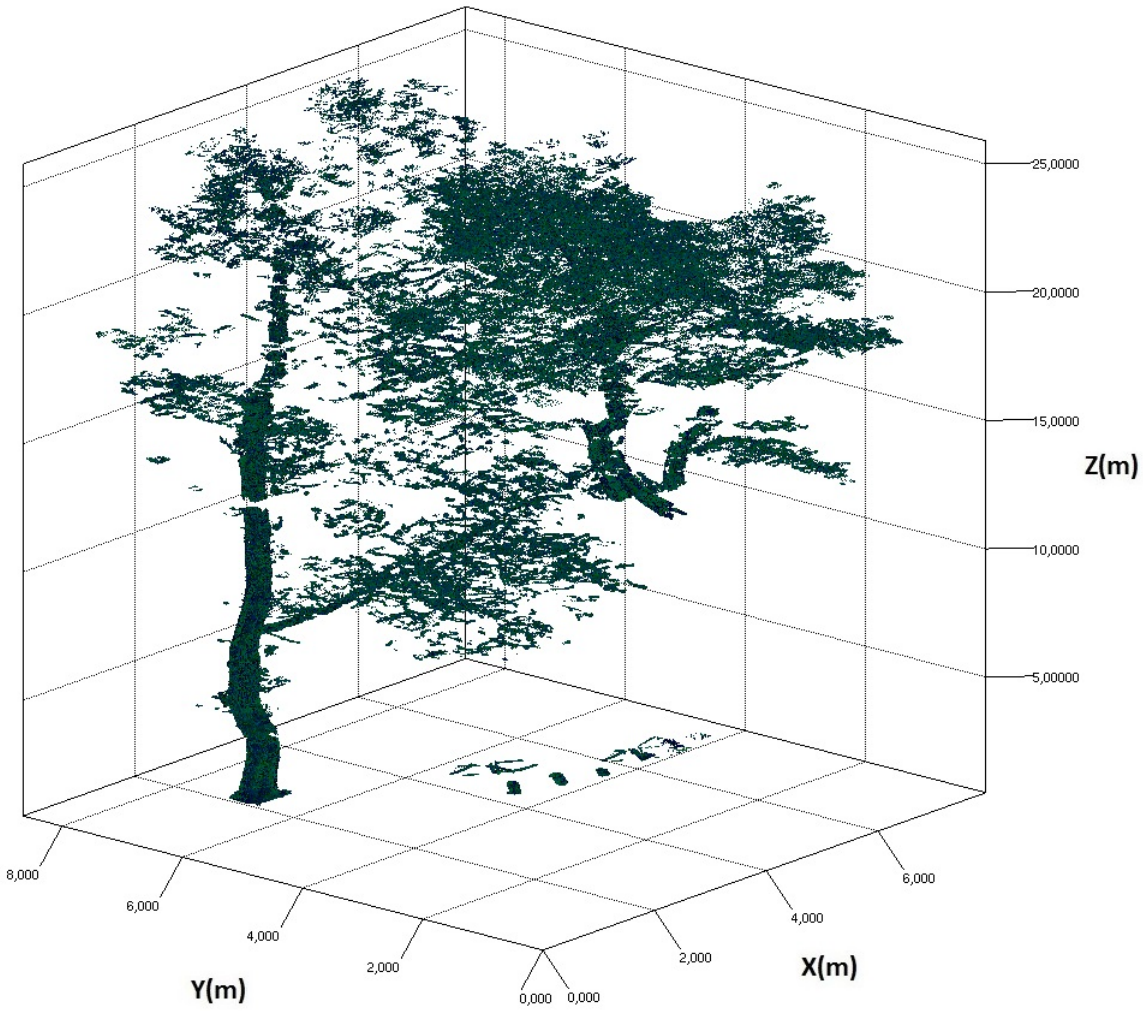


34 pav. Medžių kamienų iš antrojo optimizuoto taškų rinkinio, taškų priklausomybė nuo maksimalaus apskritimo skersmens

7.4.2. Medžio kamieno išskyrimas panaudojant cilindrų identifikavimo metodą, paremtą vokselio kaimynais

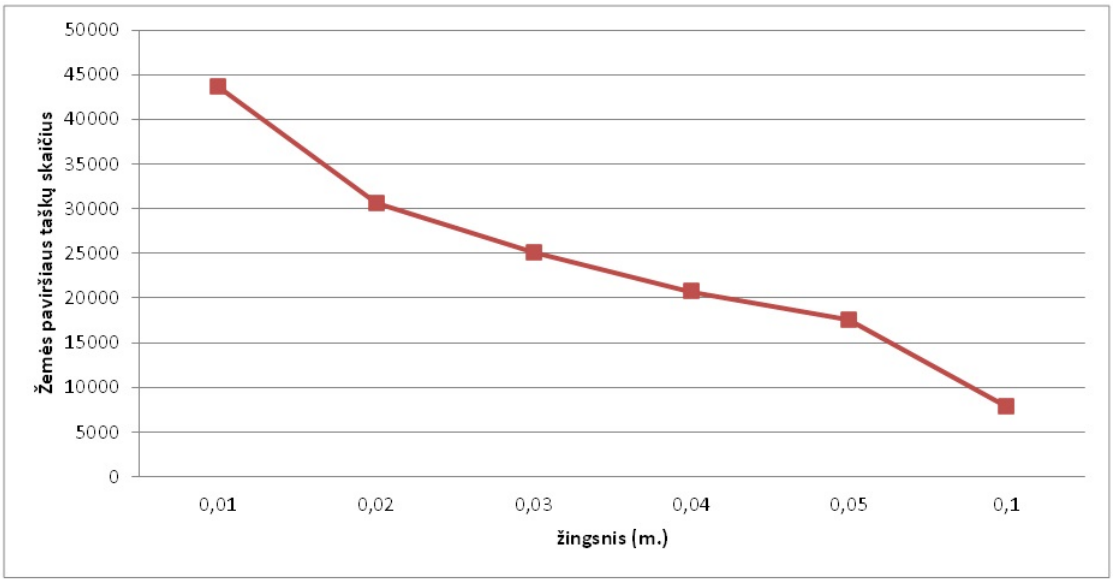
Pasinaudojus "java" programavimo kalba įgyvendinome metodą, kuris išrenka medžių kamienų taškus remdamasis vokselių kaimynais. Vokselių matmenys buvo nuo $0.05m.$ x $0.05m.$ x $0.1m.$ iki $0.3m.$ x $0.3m.$ x $0.1m.$, žingsnis $0.02m.$.

Rezultatas gautas iš pirmojo taškų rinkinio pavaizduotas (35 pav.), iš 3637739 taškų buvo išrinkti 909386 taškai priklausantys medžiams. Algoritmo vykdymas užtruko 10 sekundžių.

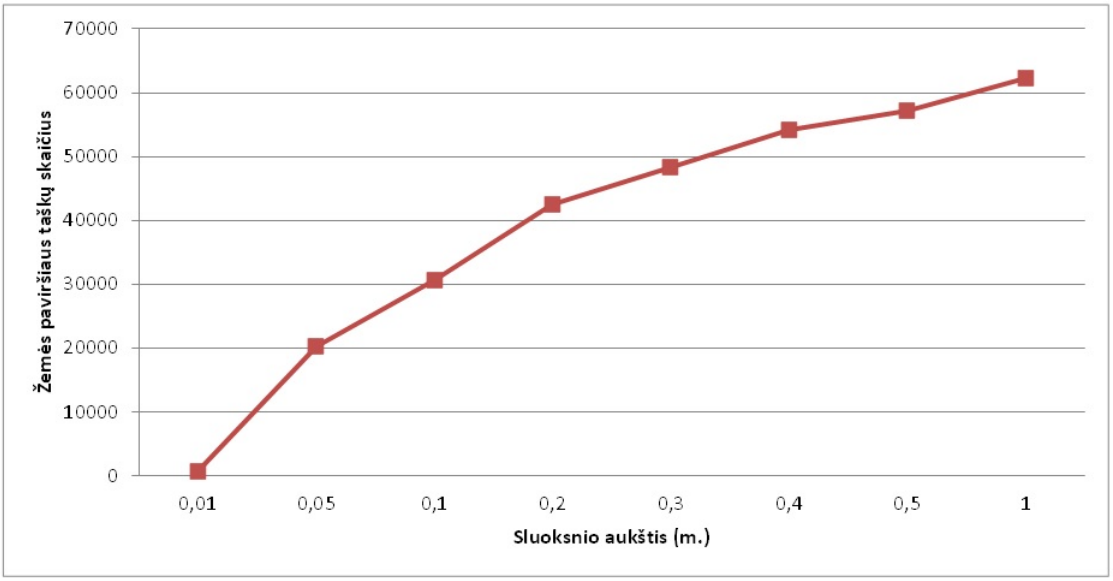


35 pav. Medžių kamienai iš pirmojo taškų rinkinio.

Atlikome tyrimą su optimizuotu pirmųjų duomenų rinkiniu, iš kurio išimti žemės paviršiaus taškai. Bendrų taškų skaičius 124023, tyrėme medžio kamieno išskyrimo metodą, paremtą vokselio kaimynais. Randamų medžių kamienų taškų priklausomybę nuo žingsnio pavaizdavome (36pav.) ir nuo sluoksnio aukščio (37 pav.).

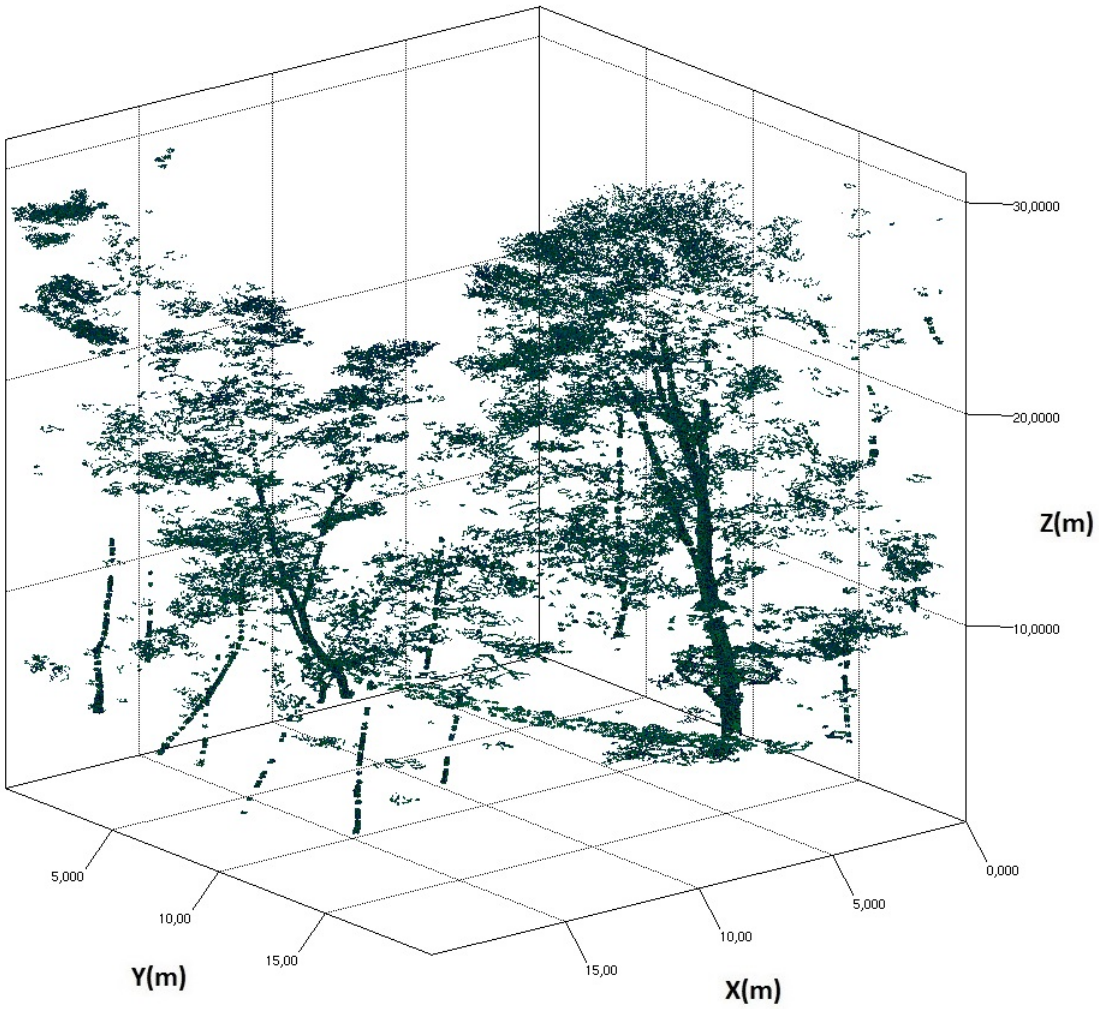


36 pav. Medžių kamienų iš pirmojo optimizuoto taškų rinkinio, taškų priklausomybė nuo žingsnio



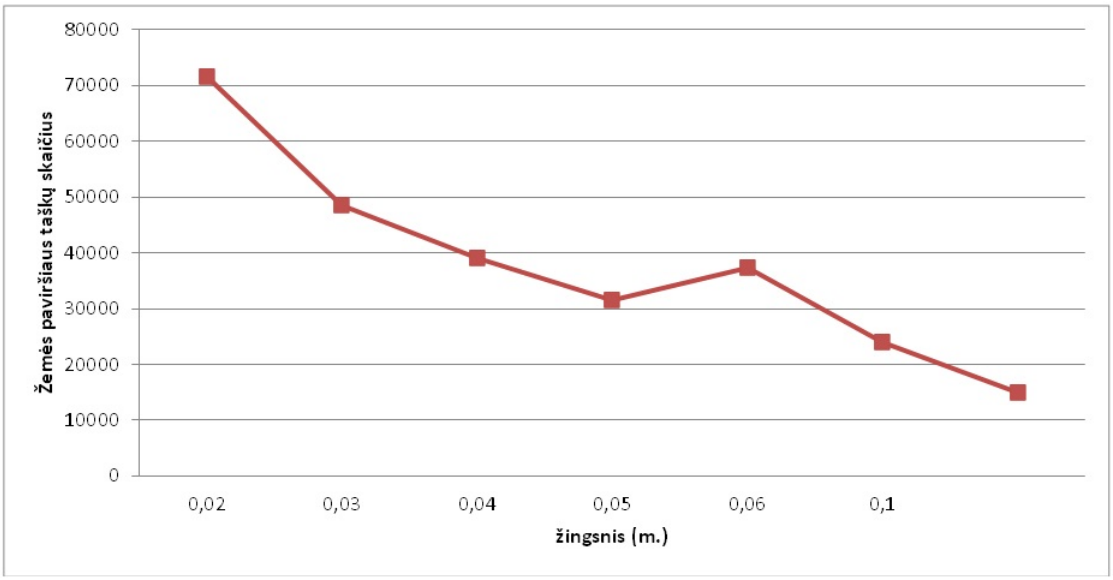
37 pav. Medžių kamienų iš pirmojo optimizuoto taškų rinkinio, taškų priklausomybė nuo sluoksnio auksčio

Rezultatas gautas iš antrojo taškų rinkinio pavaizduotas (38 pav.), iš 2442581 taškų buvo išrinkti 437512 taškai priklausantys medžiams. Algoritmo vykdymas užtruko 11 sekundžių.

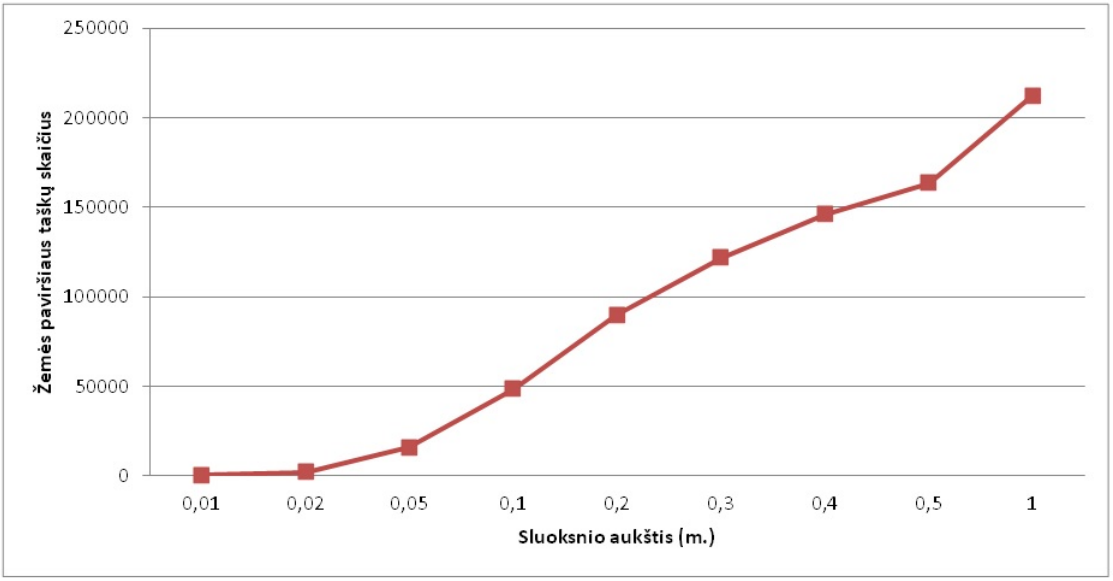


38 pav. Medžių kamienai iš antrojo taškų rinkinio.

Atlikome tyrimą su optimizuotų antroju duomenų rinkiniu, iš kurio išimti žemės paviršiaus taškai. Bendrų taškų skaičius 405046, tyrėme medžio kamieno išskyrimo metodą paremtą vokselio kaimynais. Randamų medžių kamienų taškų priklausomybę nuo žingsnio pavaizdavome (39pav.). Randamų medžių kamienų taškų priklausomybę nuo sluoksnio aukščio pavaizdavome (40 pav.).



39 pav. Medžių kamienų iš antrojo optimizuoto taškų rinkinio, taškų priklausomybė nuo žingsnio



40 pav. Medžių kamienų iš antrojo optimizuoto taškų rinkinio, taškų priklausomybė nuo sluoksnio auksčio

Skaičiavimuose buvo panaudoti duomenys, kuriose išrinkti visi žemės paviršiaus taškai

Išvados ir rekomendacijos

Atlikus baigiamąjį magistro darbą buvo gautos šios išvados:

- Išdėliojus taškus pagal vokselius atsiranda galimybė sumažinti taškų skaičių neprarandant bendro vaizdo.
- Skirtingi duomenys nedaro įtakos išrenkant žemės paviršiaus taškus, metodo išrenkamų taškų skaičius priklauso nuo stulpelio pločio.
- Metodas, paremtas žemiausio taško kaimyniniuose stulpeliuose algoritmu, geba tiksliai ir detaliai išrinkti žemės paviršiaus taškus.
- Ištyrus medžio kamieno taškų išskyrimo priklausomybes, naudojant apskritimų paieškų metodą, nuo: apskritimo ketvirčių dalių skaičiaus, sluoksnio aukščio bei maksimalaus apskritimo skersmens, buvo gauta išvada, jog su skirtingais duomenimis gaunamos tos pačios tendencijos. Naudojantis šiuo metodu pavyko tiksliai išrinkti medžių kamienų taškus.
- Metodas, paremtas vokselio kaimynais, netiskliai išrenka kamienus, jiems priskiria kitus objektus (šakas). Metodas reikalauja tikslių parametrų, tam, kad būtų gauti tikslūs duomenys.

Rekomendacijos ateities darbams: Toliau tęsiant trimačių objektų tyrimo temą, iš LIDAR duomenų, būtų galima ištiriti didesnį duomenų kiekį, tam, kad būtų gauti tikslesni rezultatai. Rekomenduojama išmėginti kuo daugiau ir įvairesnių metodų, kurie galėtų patvirtinti arba paneigti esamų metodų tikslumą.

Gairės

Darbe aprašomi metodai reikalauja daug resursų, todėl ateityje gautus duomenis galima panaudoti algoritmų apmokymui, tokių, kaip atraminių vektorių mašina (SVM), kuris yra vienas tiksliausių ir labiausiai paplitusių metodų, naudojamų klasifikavimui.

Kadangi nebuvo klasifikuoti kiti objektai, tokie kaip: trosai, plokštumos, bokšteliai ir kt., ateities darbuose siūloma išskirti būtent juos.

Literatūros šaltiniai

[1] G. I. Nikolajevich. Development and research methods of treatment and classification of three laser scanner data. Moscow State University of Geodesy and Cartography, 2011. http://www.miigaik.ru/nauka/dissertacionyy_sovet/zasedaniya/20111107163919-7732.doc.

[2] Levashev S.P. Classification of objects by laser scanning data with using the fourier descriptors base on the selected path method. Russia, Taganrog, South Federal University, 2015. http://www.penzgtu.ru/fileadmin/filemounts/science/konf_roganov/robototehnika/%D1%81%D1%82%D1%80._19-25_%D0%9A%D0%9B%D0%90%D0%A1%D0%A1%D0%98%D0%A4%D0%98%D0%9A%D0%90%D0%A6%D0%98%D0%AF_%D0%9E%D0%91%D0%AA%D0%95%D0%9A%D0%A2%D0%9E%D0%92_%D0%9F%D0%9E_%D0%94%D0%90%D0%9D%D0%9D%D0%AB%D0%9C.pdf.

[3] A. A. Tkacheva. Remote sensing for reconstruction of natural landscape scenes. Siberian State Aerospace University named after academician M. F. Reshetnev 31, Krasnoyarsky Rabochy Av., Krasnoyarsk, 660014, Russian Federation, 2014. http://cyberleninka.ru/article/n/ispolzovanie-dannyh-distantsionnogo-zondirovaniya-zemli-pri-trehme

[4] Yuhong Zhou Caiyun Zhang and Fang Qiu. Individual tree segmentation from lidar point clouds for urban forest inventory. Department of Geosciences, Florida Atlantic University, 2015. http://www.mdpi.com/2072-4292/7/6/7892/htm.

[5] Zhipeng Chen Qingquan Li and Qingwu Hu. A model-driven approach for 3d modeling of pylon from airborne lidar data. State Key Laboratory of Information Engineering in Surveying, Mapping and Remote Sensing, Wuhan University, 2015. http://www.mdpi.com/2072-4292/7/9/11501.

[6] Norbert Pfeifer Bernhard Höfle. Correction of laser scanning intensity data: Data and model-driven approaches. Institute of Photogrammetry and Remote Sensing, Vienna University of Technology, 2007. http://www.sciencedirect.com/science/article/pii/S0924271607000603.

[7] Peter Axelsson. Processing of laser scanner data—algorithms and applications. Department of Geodesy and Photogrammetry, Royal Institute of Technology, 1998. https://pdfs.semanticscholar.org/b159/34ef4d6f9a4df9305b054ccfc799b33625b9.pdf.

[8] Aušra Kalantaitė. Lidar matavimų ir taikymų technologijų tobulinimas fiziniam žemės paviršiui modeliuoti. daktaro disertacija. technologijos mokslai, matavimų inžinerija. VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS, 2015. http://gs.elaba.lt/object/elaba:15525154/15525154.pdf.

[9] Paulo Peixoto Urbano Nunes Alireza Asvadi, Cristiano Premebida. 3d lidar-based static and moving obstacle detection in driving environments: An approach based on voxels and multi-region ground planes. Institute of Systems and Robotics, Department of Electrical and Computer Engineering, University of Coimbra - Polo II, 3030-290 Coimbra, Portugal, 2016. http://www.sciencedirect.com/science/article/pii/S0921889016300483.

[10] Reinhard Klein Ruwen Schnabel, Roland Wahl. Shape detection in point clouds. Universität Bonn, Computer Graphics Group, 2006.
http://cg.cs.uni-bonn.de/aigaion2root/attachments/cg-2006-2.pdf.

[11] L. Maisonobe. Finding the circle that best fits a set of points, 2007.
https://www.spaceroots.org/documents/circle/circle-fitting.pdf.

[12] THE IMAGING ASPRS and GEOSPATIAL INFORMATION SOCIETY. Las specification version 1.4 – r13. The American Society for Photogrammetry and Remote Sensing 5410 Grosvenor Lane, Suite 210 Bethesda, Maryland 20814-2160, 2013.
http://www.asprs.org/wp-content/uploads/2010/12/LAS_1_4_r13.pdf.

[13] LAStools, 2016.
http://www.cs.unc.edu/~isenburg/lastools/.

Priedai

A. Kodas, skirtas žemės paviršiaus taškų išskyrimui, panaudojant žemiausio taško stulpelyje algoritmą

Pridedamas esminis programos kodas, kuris buvo realizuotas siekiant išskirti žemės paviršiaus taškus, panaudojant žemiausio taško stulpelyje algoritmą [4.2].

```
1                   Integer x1 = (int) (M / ZINGSNIS);
2                   Integer y1 = x1;
3                   Integer x;
4                   Integer y;
5                   Double[][] arrZ = new Double[x1][y1];
6                   Point[][] arrGrP = new Point[x1][y1];
7
8                   Double sumGrZ = 0.0;
9                   Integer cntGr = 0;
10
11                  for (Point p : points) {
12                          x = (int) (p.getX() / ZINGSNIS);
13                          y = (int) (p.getY() / ZINGSNIS);
14                          if (arrZ[x][y] != null) {
15                                  if (arrZ[x][y] > p.getZ()) {
16                                          arrZ[x][y] = p.getZ();
17                                          p.setGround(true);
18                                          arrGrP[x][y].setGround(false);
19                                          sumGrZ -= arrGrP[x][y].getZ();
20                                          arrGrP[x][y] = p;
21                                          sumGrZ += p.getZ();
22                                  }
23                          } else {
24                                  arrZ[x][y] = p.getZ();
25                                  p.setGround(true);
26                                  arrGrP[x][y] = p;
27                                  sumGrZ += p.getZ();
28                                  cntGr++;
29                          }
30                  }
```

B. Kodas, skirtas žemės paviršiaus taškų išskyrimui, panaudojant žemiausio taško kaimyniniuose stulpeliuose algoritmą

Pridedamas esminis programos kodas, kuris buvo realizuotas siekiant išskirti žemės paviršiaus taškus, panaudojant žemiausio taško kaimyniniuose stulpeliuose algoritmą [4.3].

```
1 Integer x1 = (int) (M / ZINGSNIS);
2 Integer y1 = x1;
3 Integer x;
4 Integer y;
5 Column[][] arrColumn = new Column[x1][y1];
6 for (Point p : points) {
7         x = (int) (p.getX() / ZINGSNIS);
8         y = (int) (p.getY() / ZINGSNIS);
9         if (arrColumn[x][y] == null) {
10                arrColumn[x][y] = new Column();
11                arrColumn[x][y].getPoints().add(p);
12        } else {
13                arrColumn[x][y].getPoints().add(p);
14        }
15 }
16
17 for (x = 0; x < x1; x++) {
18        Integer a = ((x - KAIMYNAI) >= 0) ? x - KAIMYNAI : 0;
19        Integer a2 = ((x + KAIMYNAI) < x1) ? x + KAIMYNAI : x1;
20        for (y = 0; y < y1; y++) {
21                if (arrColumn[x][y] != null) {
22                Integer b = ((y - KAIMYNAI) >= 0) ? y - KAIMYNAI : 0;
23                Integer b2 = ((y + KAIMYNAI) < y1) ? y + KAIMYNAI : y1;
24                for (Integer x2 = a; x2 <= a2; x2++) {
25                for (Integer y2 = b; y2 <= b2; y2++) {
26                if (arrColumn[x2][y2] != null
27                        && !arrColumn[x2][y2].getPoints().isEmpty()
28                && !arrColumn[x2][y2].equals(arrColumn[x][y])) {
29                arrColumn[x][y]
30                        .setKaimynuSk(arrColumn[x][y].getKaimynuSk() +
                              1);
31                for (Point p : arrColumn[x2][y2].getPoints()) {
32                if (arrColumn[x][y].getMinKaimynZ() == null) {
33                arrColumn[x][y].setMinKaimynZ(p.getZ());
34                } else if (arrColumn[x][y].getMinKaimynZ() > p.getZ()) {
35                arrColumn[x][y].setMinKaimynZ(p.getZ());
36                }
37                }
38                }
39                }
40                }
41                }
42        }
43 }
```

C. Kodas, skirtas medžio kamieno išskyrimui, panaudojant apskritimų paieškos metodą

Pridedamas esminis programos kodas, kuris buvo realizuotas siekiant išskirti medžio kamieną, panaudojant apskritimų paieškos metodą [5.1].

```
1                   while (vid_d <= max_vid_d) {
2                   Double isor_d = vid_d + taskuZona;
3                   Points[][] pointCols = SetPointsCols(pointsCloud, isor_d
                        );
4                   Integer x;
5                   Integer y;
6                   Integer maxXY = (int) (M / isor_d);
7                   Integer cnt = 0;
8                   for (Point p1 : pointsCloud) {
9
10                  x = (int) (p1.getX() / isor_d);
11                  y = (int) (p1.getY() / isor_d);
12
13                  Integer x1 = x - 1 > 0 ? x - 1 : 0;
14                  Integer y1 = y - 1 > 0 ? y - 1 : 0;
15
16                  Integer x2 = x + 1 < maxXY ? x + 1 : maxXY;
17                  Integer y2 = y + 1 < maxXY ? y + 1 : maxXY;
18                  List<Point> neighbordPoints = new ArrayList<Point>();
19                  for (int a = x1; a <= x2; a++) {
20                  for (int b = y1; b <= y2; b++) {
21                  if (pointCols[a][b] != null) {
22                  neighbordPoints.addAll(pointCols[a][b].getPoints());
23                  }
24                  }
25                  }
26
27                  for (Point p2 : neighbordPoints) {
28                  if (p1 != p2) {
29                  Double dis = dist(p1.getX(), p1.getY(), p2.getX(), p2.
                        getY());
30                  if (dis <= isor_d && dis > vid_d) {
31                  CyclePoints cyclePoints = new CyclePoints(daliuSkKetv);
32                  Double radius = dis / 2;
33                  cyclePoints.setRadius(radius + taskuZona);
34                  Double midX = mid(p1.getX(), p2.getX());
35                  Double midY = mid(p1.getY(), p2.getY());
36
37                  cyclePoints.setCenterX(midX);
38                  cyclePoints.setCenterY(midY);
39                  cnt = 0;
40                  for (Point p3 : neighbordPoints) {
41                  Double vid_dis = dist(p3.getX(), p3.getY(), midX, midY);
42                  if (vid_dis < (radius - taskuZona)
43                  || (vid_dis > (radius + taskuZona) && vid_dis < (radius
                        + beTaskuZona))) {
44                  cnt++;
45                  break;
```

```
46                  } else {
47                  if (vid_dis >= min_d && vid_dis <= (radius + taskuZona))
                        {
48                  cyclePoints.addPoint(p3);
49                  }
50                  }
51                  }
52                  if (cnt == 0 && cyclePoints.isCycle()) {
53                  for (Point p : cyclePoints.getPoints()) {
54                  if (!pointsList.contains(p)) {
55                  if (!allPoints) {
56                  pointsList.add(p);
57                  } else {
58                  p.setTrunk(true);
59                  }
60                  }
61                  }
62                  }
63
64                  }
65                  }
66                  }
67                  }
68                  vid_d = vid_d + zingsnys;
69                  }
```

D. Kodas, skirtas medžio kamieno išskyrimui, panaudojant cilindrų identifikavimo metodą, paremtą vokselio kaimynais

Pridedamas esminis programos kodas, kuris buvo realizuotas siekiant išskirti medžio kamieną, panaudojant cilindrų identifikavimo metodą, paremtą vokselio kaimynai [5.2].

```
1
2                   while (zingsnis <= TO_ZINGSNIS) {
3                   Points[][] pointCols = SetPointsCols(pointsCloud,
                        zingsnis);
4                   Integer x1 = (int) (M / zingsnis);
5                   Integer y1 = x1;
6                   for (int x = 1; x < x1 - 1; x++) {
7                   for (int y = 1; y < y1 - 1; y++) {
8                   if (pointCols[x][y] == null || pointCols[x][y].getPoints
                        ().size() == 0) {
9                   int cnt = 0;
10                  List<Point> tmpPointsList = new ArrayList<Point>();
11
12                  for (int a = x - 1; a <= x + 1; a++) {
13                  for (int b = y - 1; b <= y + 1; b++) {
14                  if (pointCols[a][b] != null && pointCols[a][b].getPoints
                        ().size() > 0) {
15                  tmpPointsList.addAll(pointCols[a][b].getPoints());
16                  cnt++;
17                  }
18                  }
19                  }
20
21                  if (cnt == UZPILDYTI_APLINK_LANGELIAI) {
22                  for (Point p : tmpPointsList) {
23                  if (!pointsList.contains(p)) {
24                  pointsList.add(p);
25                  }
26                  }
27                  }
28                  }
29                  }
30                  }
31                  zingsnis += ZINGSNIS;
32                  }
```