TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Tarkvarateaduse instituut

Jakob Pindis 143066IAPB

LASERSKANNERIGA KOGUTUD 3D PUNKTIPILVE TÖÖTLEMINE JA VISUALISEERIMINE

Bakalaureusetöö

Juhendaja: Martin Rebane
MSc

Tallinn 2018

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Jakob Pindis

09.05.2018

Annotatsioon

Käesoleva töö eesmärgiks oli luua veebirakendus, mis võimaldab töödelda 3D laserskanneri poolt genereeritud toorandmeid, neid salvestada ja visualiseerida seadistustele vastavalt. Töödeldud andmetega on Teede Tehnokeskusel võimalik teostada rakenduse väliselt täiendavaid arvutusi. Veebirakenduse visuaalne osa annab selge ülevaate tee üldisest seisukorrast.

Rakenduse loomiseks tuli kõigepealt tutvuda toorandmetega, mõelda läbi kuidas näeb välja kogu töötluse protsess rakenduse siseselt ning kuidas siduda see veebirakenduseks mis on võimeline töötlema ja visualiseerima suurt hulka andmeid. Esimesena loodi lahendus toorandmete töötlemiseks ja salvestamiseks. Teiseks loodi veebirakendus töödeldud andmete visualiseerimiseks ja seadistuste valimiseks. Loodud veebirakenduse osa ühendati andmetöötluse osaga.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 33 leheküljel, 6 peatükki, 11 joonist, 4 tabelit.

Abstract

Point cloud processing and visualization based on 3D laser scanner data

The purpose of this thesis was to create a web application to process, save and display road data generated by 3D laser scanner according to user defined settings. Processed data is easily usable for additional calculations later for Teede Tehnokeskus. Visualization part of application gives clear overview of road conditions.

At first, raw data needed to be examined and thought through whole internal process order of application. Original raw data was too dense and not optimal for external calculations and tools use. Moreover, data generated by scanner software was split into multiple files without clear connection between them and contained some defective data. Teede Tehnokeskus also noted that some crucial data should be added to final file which is not possible to get from scanner files directly to help simplify later calculations for them. Steps included in the process were following: conversion of GPS heights to EH2000 heights via Maa-amet, calculation of relative heights, choosing profile step range and removing defective profiles, smoothing and compressing of road points, interpolating and connecting GPS data with road points, generating GeoJSON list.

When the processing part was working, visualization and user control over certain settings needed to be added. Scanner provided software gave some visual info about the data but Teede Tehnokeskus needed broader options to inspect the road data. Therefore, some available options for data visualization were tested and most suitable ones were picked. After that, visual part was implemented which gave quick overview of road conditions as 2D plot or as points generated on satellite map. Finally, web application was connected with processing part and some user control settings were added that would give control over data processor settings.

The thesis is in Estonian and contains 33 pages of text, 6 chapters, 11 figures, 4 tables.

Lühendite ja mõistete sõnastik

Back-end	Serveripoolne osa arhitektuurist, mis tegeleb äriloogika ja andmetega
Crossfall	Tee kaldenurk mõlemal teerajal, oluline tee ohutuse tagamiseks
Endpoint	Aadress, mille kaudu on võimalik veebiteenusega suhelda
Front-end	Kliendipoolne osa arhitektuurist, mis suhtleb serveriga
GeoJSON	JSON formaat, mis mõeldud eelkõige geograafiliste andmete edastamiseks
heap[1]	Mäluosa, kus mälublokid eraldatakse objektidele ja prügikoristusel vabastatakse
HTTP	Protokoll teabe edastamiseks arvutivõrkudes
HTTP GET	HTTP idempotentne päringu liik, REST puhul kasutatakse andmete küsimiseks
HTTP PUT	HTTP idempotentne päringu liik, REST puhul kasutatakse andmete lisamiseks või muutmiseks
JAR	Formaat Java klasside ja teiste ressursside koondamiseks ühte faili
Java *Garbage Collector*[2]	Taustal töötav protsess, mis tuvastab, millistele objektidele enam ei viidata ning kustutab need *heap* mälust
JSON	Formaat andmete edastamiseks, mis sisaldab teksti kujul liste ning atribuut-väärtus paare
Punktide värvi interpoleerimine	Meetod vahepealsete väärtuste (värvide) leidmiseks etteantud väärtuste alusel
Punktipilv	Ruumiliselt laiali olevate punktide kogum, milles igale punktile on määratud x, y, z koordinaadid
REST	REST (*Representational State Transfer*) on tarkvaraarhitektuuri stiil
Swap space	Operatiivmälu virtuaalne jätk andmekandjal
Tee profiil	Teepinna ristläbilõige, sisaldab algselt 540 punkti
Xmx	Terminali kaudu Java programmile eraldatav maksimaalne *heap* mälu

[1] https://docs.oracle.com/cd/E13150_01/jrockit_jvm/jrockit/jrdocs/refman/optionX.html

[2] http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Sisukord

Jooniste loetelu

Tabelite loetelu

1 Sissejuhatus

Teede Tehnokeskus mõõdistab teid LIDAR tehnoloogial põhineval 3D laserskanneriga. Laserskanneriga andmete kogumine on nüüdsel ajal populaarsust koguv meetod teede olukorra hindamiseks [1]. Töö keerukus seisnes sobivate viiside ning algoritmide leidmisel 3D punktipilve andmete eeltöötlemises, nende kokku ühendamises ja veebirakenduses kuvamises [2]. Praeguseni ei olnud Teede Tehnokeskusel lihtsat võimalust, kogutud andmete põhjal, saada ülevaadet teede üldisest seisukorrast. Kasutusel olev tarkvara ViaPPS Desktop võimaldab näha ühte sõiduraja teelõiku 3D ruumis, mis põhineb kaamera pildil ning ei anna head tagasisidet tee olukorrast. Lisaks kuvab ViaPPS Desktop tee läbilõike profiili. Probleemiks oli, et see ei andnud kiiret ülevaadet pikemast teelõigust ning toorandmeid oli keeruline kasutada täiendavateks arvutusteks suure mahu ja toorandmetes kasutatava vormingu tõttu [3].

Töö lõppeesmärgiks oli luua tarkvara, mille abil saab selge visuaalse ülevaate skaneeritud tee seisukorrast ja võimaldab salvestada töödeldud andmed, mida saab hiljem kasutada täiendavataks arvutusteks. Töö raames seadis autor endale kaks alameesmärki. Esiteks luua serveripoolne lahendus andmete töötluseks ja salvestamiseks. Teiseks luua veebirakendus, mis suhtleb serveriga REST päringute kaudu ning võimaldab töödeldud andmeid visualiseerida ja valida seadistusi töötlusprotsessi tarvis.

Esimese eesmärgi saavutamiseks uuriti võimalike lahendusi ning pandi paika sammud, mis tuleb läbida, et jõuda algsetest toorandmetest töödeldud andmeteni. Kõigi sammude juures oli vaja luua või otsida olemasolev algoritmiline lahendus ja teha kindlaks, et iga samm ei oleks liiga aeganõudev ning annaks soovitud tulemuse. Lisaks tuli jälgida mälukasutust andemete töötlusel. Töötluse protsess hõlmas endas järgmisi samme: EH2000 kõrguste konverteerimist Maa-ameti mudeli abil, relatiivsete kõrguste leidmist, võimalust jätta alles profiile teatud sammu kaupa, defektsete profiilide eemaldamist, andmete silumist Savitzky–Golay filtri abil, andmete hõrendamist Douglas-Peucker'i algoritmi põhiselt, töödeldud andmete interpoleerimist ning ühendamist GPS koordinaatidega, LEST koordinaatide leidmist, GeoJSON'i genereerimist.

Teise eesmärgi saavutamiseks loodi veebirakendus, mis muudab andmetöötluse protsessi käivitamise kasutajale mugavamaks ja võimaldab andmeid visualiseerida. Veebirakendus annab võimaluse kasutajal seadistada töötlusprogrammi teatud parameetreid ning valida, mida tehakse saadud andmetega peale töötluse lõppu. Andmete visualiseerimiseks tuli testimise teel leida sobiv viis suure koguse punktide kuvamiseks.

Loodud rakenduse kood on üleval GitLabi repositooriumis[1].

[1] https://gitlab.cs.ttu.ee/Jakob.Pindis/RoadProject.git

2 Analüüs

Selles peatükis kirjeldatakse veebirakenduse funktsionaalseid ja mittefunktsionaalseid nõudeid, tuuakse välja sarnased lahendused ning selgitatakse töö planeerimist.

2.1 Funktsionaalsed nõuded

Funktsionaalseid nõudeid vaadeldakse Teede Tehnokeskuse töötaja seisukohast. Kasutaja saab teostada järgmisi toiminguid:

- anda ette asukoht, kust faile soovitakse esitada ja salvestada
- valida punktipilve ja GPS fail mida töödelda
- valida hõrendamise algoritmi tolerantsi väärtust
- valida mitmes iga profiil jäetakse alles ehk profiili sammu
- valida defektse profiili tuvastuse väärtust
- valida, kas töödeldud fail salvestada failina
- valida, kas töödeldud andmetest kuvatakse 2D diagramm vaadet
- valida, kas töödeldud andmetest kuvatakse kaardivaadet
- klikkida kaardivaatel punktil, mis kuvab selle relatiivse kõrguse, profiili kalde (*crossfall*) ja punkti geograafilised koordinaadid
- valida, kas soovitakse jätta eelneval kaardivaatel olevad punktid meelde, võimaldab vaadata kahte kihti korraga, näiteks kahte teerada kõrvuti
- valida, kas kasutatakse Maa-ameti EH2000 GPS punktide absoluutsete kõrguste konverteerimist

2.2 Mittefunktsionaalsed nõuded

Mittefunktsionaalsete nõuete alla käivad inglisekeelne kasutajaliides ja lähtekood. Rakendus töötab töö kirjutamise ajal uusima Google Chrome, Firefox brauseritega ja Java 8-ga.

Rakendus peab suutma töödelda vähemalt ühe kilomeetriseid lõike rakenduse algseadistustega arvutis, kus on 8GB mälu. Teised rakendused olid samal ajal suletud.

2.3 Tehnoloogia valik

Antud ülesande lahendamiseks valis autor Java[1] programmeerimiskeele peamiselt isikliku eelistuse ja juhendaja soovituse tõttu. Projekti sõltuvuste haldamiseks ja lõpliku JAR'i ehitamiseks kasutati Maven'it[2]. Loodud rakendus koosneb kahest osast:

- serverirakendus – järgib REST põhimõtteid, kasutatakse Spring Boot[3] teeki, visualiseerimiseks JFreeChart[4]
- kasutajapoolne rakendus – kasutatakse HTML'i, CSS'i, JavaScript'i, Bootsrap[5] ja jQuery[6] teeki, visualiseerimiseks Mapbox API [4]

Mapbox osutus valituks, kuna paralleelselt nii Mapbox API kui Google Maps API [5] katsetamisega leiti, et Mapbox on töö jaoks otstarbekam valik. Mapbox suudab suurema hulga punktidega oluliselt efektiivsemalt käituda. Näiteks Google Maps'i testides ainult 30 000 punktiga tekkisid juba probleemid, Mapbox'i jaoks 200 000 punkti kuvamine probleemiks ei olnud. 200 000 punkti kuvamine võttis umbes sama palju või isegi vähem mälu kui Google Maps 20 000 punktiga.

2.3.1 Mapbox'i ja Google Maps'i võimalused

Mõned tähelepanekud täiendavalt mis mõlema API proovimisel tekkisid olid järgmised. Mapbox'i satelliidi vaade lubab suumida kaarti oluliselt lähemale kui Google Maps ning

[1] https://java.com/en/
[2] https://maven.apache.org/
[3] https://projects.spring.io/spring-boot/
[4] http://www.jfree.org/jfreechart/
[5] https://getbootstrap.com/
[6] https://jquery.com/

kaardile kuvatavad punktid paistsid kaugemalt vaadates paremini eristatavad. Google Maps'il puudub kaardil punktide värvi interpoleerimise võimalus, Mapbox'il on see olemas[1].
Google Maps'ile on võimalik kuvada *Fusion Table* kihiti, millel on värvide interpoleerimise võimalusele sarnane valik nimega gradient[2]. *Fusion Table* leevendaks suure hulga punktide töötlust kasutaja arvutis. *Fusion Table* lahendusel on kaks suurt miinust. Esiteks, nõuaks see andmete üleslaadimist ja teiseks piirang, et kuvatakse üleslaetud failist ainult esimest 100 000 rida[3].

Satelliidi vaade osutus valituks, kuna tavavaates muutuvad tee laiused olenevalt suumi astmest ning ei vastanud tegelikkusele. Google Maps kaardil tavavaatega maksimaalse suumi astmega oli tee laius 2.5m, kaustades selleks sisseehitatud *Measure distance* tööriista. Tegelikult peab kaherajalise teel laius olema umbes 8 meetrit. Sama süsteem oli ka Mapbox'i tavavaates.

2.3.2 Mapox'i ja Google Maps'i mälukasutuse võrdlus

Mapbox'i ja Google Maps'i kuvatavate punktide ja mälukasutuse võrdlust teostati järgnevalt. Mõlemal juhul genereeriti objektid, mille geograafilised koordinaadid erinesid üksteisest nii laius ja pikkuskraadi suhtes 0.000001 võrra. Nii Mapbox'ile kui Google Maps'ile genereeritud punktid olid samad geograafilised koordinaadid.

Google Maps'iga genereeriti igale koordinaadile objekt *front-end*'is. Mapbox'i puhul otse *front-end*'is genereerida objekte ei saanud, vaid samade koordinaatidega punktid tuli *front-end*'is sisse lugeda GeoJSON failist. Mapbox'i testimiseks kasutati lokaalset Node.js http-serverit[4]. See oli kiire viis Mapox'ile ette anda genereeritud testimise punktid ning võimaldas hoida Google Maps'i ja Mapbox'i võrdluse võimalikult sarnasena.

Mälukasutust kontrolliti Google Chrome sisseehitatud *Task Manager*'iga. See võimaldas vaadata iga brauseri saki mälukasutust eraldi. Iga testi mõõtmiseks tehti eelnevalt Google Chrome brauseris *Empty Cache and Hard Reload*, mis tagaks, et iga kord oleks vahemälu

[1] https://www.mapbox.com/mapbox-gl-js/style-spec#expressions-interpolate
[2] https://developers.google.com/fusiontables/docs/v2/reference/style
[3] https://developers.google.com/maps/documentation/javascript/fusiontableslayer
[4] https://www.npmjs.com/package/http-server

tühjendatud ning ei ole midagi alles eelmisest mõõtmisest. Ajamõõtmist ei peetud oluliseks, kuna tabelis kontrollitud punktide arvu juures oli ajaline erinevus minimaalne, võttes arvesse kogu programmi tööaega. Testimise hetkel oli Google Chrome brauseri versiooniks 65.0.3325.181. Operatsioonisüsteemiks Ubuntu 16.04 põhinev Linux. Kõik peale brauseri ühe saki, *Task Manager*'i, ja lokaalse Node.js serveri oli testimise ajal suletud.

Nagu näha Tabel 1 põhjal, kasutab Google Maps natuke vähem mälu kui on vaja kuvada väike kogus punkte ehk 1000 punkti. Töö jaoks on olulisem pigem teine äärmus, kus kuvatakse palju punkte. Võrreldes 20 000 ja 200 000 punktide ridasid, kasutab Mapbox umbes 10 korda vähem mälu. Üheks Google Chrome miinuseks Firefox'i ees, mis testimise käigus selgus, et Google Chrome brauser üritab hoiduda palju mälu nõudvate tegevuste eest, mis üldjuhul on hea, kuid see piirab lõputöö eesmärgi saavutamist. Näiteks anti teade *Paused before potential out-of-memory crash* Google Maps'iga 30 000 punkti korral kui oli veel 3GB vaba mälu. Sama juhtus Mapbox'i 300 000 punkti kuvamisega. Kui avati Google Chrome konsool ja vajutati sealt nuppu mis jätkas peatatud lõimede tööd, siis tegelikult kuvati kõik 300 000 punkti Mapbox'i kaardil, ilma, et oleks kasutatud *swap* mälu.

Tabel 1. Mapbox'i ja Google Maps'i mälukasutuse võrdlus Google Chrome brauseris.

Kuvatav punktide arv	**Mälukasutus Mapbox'iga**	**Mälukasutus Google Maps'iga**
1000	0.21 GB	0.19 GB
5000	0.25 GB	0.65 GB
10 000	0.28 GB	1.2 GB
15 000	0.3 GB	1.8 GB
20 000	0.36 GB	2.4 GB
30 000	0.46 GB	Chrome teade: *Paused before potential out-of-memory crash*
100 000	1.2 GB	-
200 000	2.2 GB	-
300 000	Chrome teade: *Paused before potential out-of-memory crash*	-

Kuna leiti, et vaba operatiivmälu oli veel piisavalt, otsustati testida 300 000 punkti kuvamist Firefox brauseriga. Testimise hetkel Firefox'i versiooniks oli 59.0.2. Firefox suutis kuvada 300 000 punkti korraga. Firefox'il taolist kontrolli peal ei olnud mis mälukasutust üritas piirata. Firefox brauseril sissehitatud *Task Manager*'i ei olnud mille järgi jälgida brauseri saki mälukasutust. Testiti, mis oleks 8GB mäluga arvutis maksimaalne piir, kus *swap* mälu veel kasutusele ei võeta. Jälgiti operatsioonisüsteemi *Task Manager*'i kaudu *swap* mälu kasutust. Niikaua kui operatsioonisüsteemi *swap* püsis 0, laaditi kõik operatiivmällu. Operatiivmälu piiriks oli umbes 450 000 punkti korraga kuvamine. Suudeti kuvada ka 600 000 punkti korraga, kuid siis oli näha, et osa andmetest läks *swap* mällu.

2.4 Sarnased lahendused

Eestis teadaolevat sarnast lahendust ei pakuta. Välismaalt otsides on olemas sarnaseid lahendusi nagu ViaPPS pakub ehk skanneri ja tarkvara kombinatsioon, mille põhjal kogu süsteemi kasutatakse. Järgnevalt on välja toodud mõned ligilähedased valikud:

- Lehmann+Partner[1] - Saksamaal tegutsev ettevõte. Pakuvad teenusena teeprofiili mõõtmisi koos geograafiliste koordinaatidega ning andmete salvestamist andmebaasi.

- Topcon[2] - Rahvusvaheliselt tegutsev Jaapanist pärit ettevõte, pakuvad sarnast laserskannerit integreeritud tarkvaraga.

- Faro[3] - Rahvusvaheliselt tegutsev USA'st pärit ettevõte. Väidetavalt üks kompaktsemaid ja soodsamaid lahendusi[4]. Laserskanner koos integreeritud tarkvaraga.

[1] http://www.lehmann-partner.de/company/company?L=6

[2] https://www.topconpositioning.com/paving-milling-and-compacting/milling/rd-m1-scanner#panel-product-info

[3] https://www.faro.com/download-center/download-centre/04ref201-665-en---road-scanner-c-tech-sheet.pdf

[4] https://www.faro.com/en-gb/products/construction-bim-cim/road-scanner-c/

- MNG[1] - Austraalias tegutsev ettevõte. Pakub mitte ainult tee kaardistamist, vaid näiteks raudteede olukorra hindamist teenusena. Kasutavad integreeritud tarkvara.

On olemas ka ainult tarkvaralisi lahendusi[2], mis ei ole kindla skanneriga seotud. Need võimaldavad importida punktipilve faili programmi. Taoliste tarkvarade peaeesmärk ei ole tee kvaliteedi hindamine, vaid punktipilve visualiseerimine üldisemalt.

2.5 Töö planeerimine

Töö ja regulaarsed kohtumised lõputöö tarvis algasid oktoobris 2017. Alustuseks uuriti „*Toolset for data-based evaluation and visualisation of road surface condition*“ magistritöös [6] sealset läbiviidud protsessi, et saada idee, mis suunas peaks liikuma. Realiseeriti viidatud töös relatiivsete kõrguste leidmise osa Java programmeerimiskeeles.

Kõige intensiivsem arendusfaas kestis veebruari algusest kuni märtsi lõpuni. Sel perioodil toimus iganädalane kohtumine kliendiga kus vaadati üle mis vahepeal tehtud sai ning mida tuleks järgmine nädal lisada või parandada. Leiti lahendused andmetest defektide eemaldamiseks, silumiseks, hõrendamiseks, GPS koordinaatidega ühendamiseks ja salvestamiseks. Seejärel loodi veebirakendus kus katsetati nii Google Maps’i kui Mapbox’i visualiseerimist kaardil, valiti Mapbox. Täpsematest põhjustest räägiti punktis 2.3. Peale kaardipakkuja valiku kindlakstegemist, ühendati andmetöötluse ja visualiseerimise osad kokku üheks veebirakenduseks.

Kohtumiste käigus täpsustati tarkvara lisavõimalusi ning realiseeriti täiendavalt funktsionaalsust. Lisati EH2000 kõrguse arvutamine, profiili sammu valimine, LEST koordinaatide arvutamine ja lisamine lõppfaili, JFreeChart 2D diagrammi võimalus, kahe kihi kuvamise võimalus kaardivaates ning et kasutajal oleks valikuvõimalus mida soovitakse töödeldud andmetega teha.

[1] https://www.mngsurvey.com.au/services/laser-scanning
[2] https://en.wikipedia.org/wiki/List_of_programs_for_point_cloud_processing

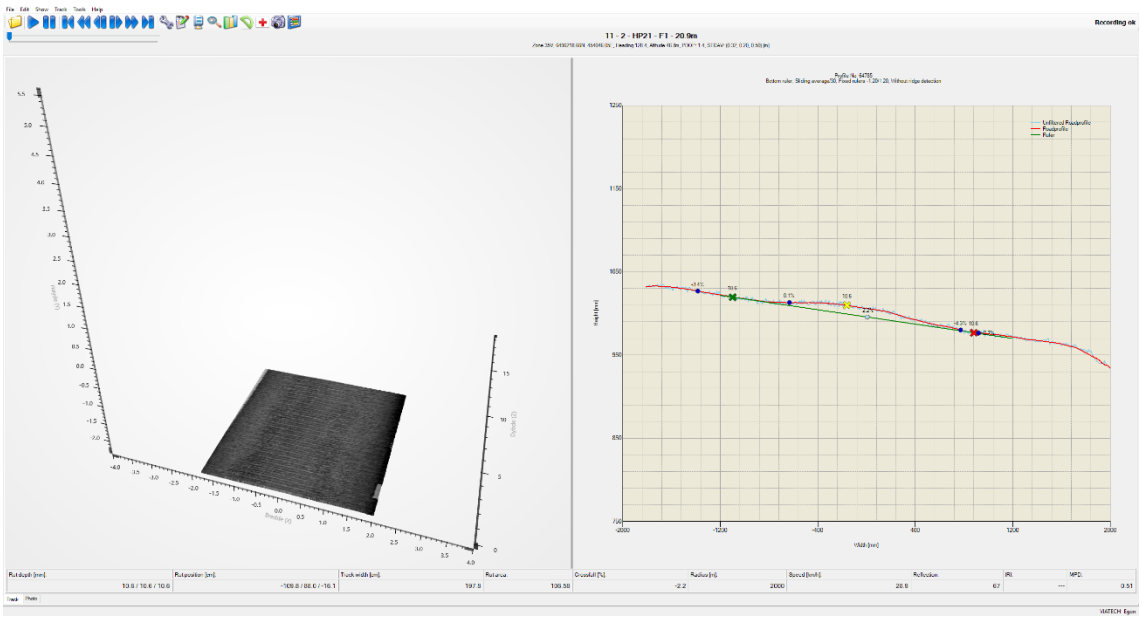
3 Skanneri toorandmed ja töölauarakendus

Toorandmetena on kaks tekstifaili millest töötluse tulemusena saab kokku ühe töödeldud tekstifaili. Esimene fail on punktipilve fail. Teine fail on GPS fail. Nimetatud toorandmetega failid genereeritakse kasutatava 3D laserskanneri ViaPPS Desktop tarkvara abil.

3.1 Töölauarakendus

Laserskanneriga on kaasas töölauarakendus ViaPPS Desktop. Näide rakenduse vaatest on toodud Joonis 1 abil. Töö kirjutamise ajal oli tarkvara versiooniks v8.9.0.0.

ViaPPS töölauarakenduse vasakus pooles kuvatakse 3D ruumis skanneri kaameraga filmitud teepinda, mis näitab teed visuaalselt, aga ei too välja olulist infot tee seisukorrast (x y z telgede kasutus on teistsuguses järjestuses kui toorandmete failides). Paremal pool on tee läbilõige, mis näitab küll tee kallet ja roopaid, aga seda ainult ühe profiilina. Läbilõiget kuvatakse kohast kus vasakpoolse diagrammi Z telg on 0.



Joonis 1. ViaPPS Desktop 10 meetrise lõigu vaade.

3.2 Punktipilve faili andmed

Punktipilve fail sisaldab järgmiste pealkirjadega veerge: *X-Value*, *Y-Value*, *Z-Value*, *Intensity*. *X-Value* on piki teed, *Y-Value* teega risti nihe tee keskelt, *Z-Value* on kõrgus ja *Intensity* on tagasipeegelduse intensiivsus. Kuigi lõputöö eesmärgi jaoks ei ole vaja kasutada intensiivsust, siis Teede Tehnokeskuse soovil loetakse see programmi sisse, et lisada lõppfaili. Intensiivsust loodab Teede Tehnokeskus tulevikus hakata realiseerima näiteks teemärgistuste kvaliteedi hindamise jaoks. Lõik punktipilve failist on näidatud Tabel 2 abil.

Tabel 2. Lõik punktipilve failist.

X-Value	**Y-Value**	**Z-Value**	**Intensity**
20.8802998618507	-1811.58377072057	1034.67967220595	155
20.8802998618507	-1801.73187812613	1034.44368775139	158
20.8802998618507	-1790.74075286302	1034.18817412877	164
20.8802998618507	-1780.53951904777	1033.92582325786	168
20.8802998618507	-1772.64850395409	1033.67841812124	166
20.8802998618507	-1762.47659083659	1033.35714148491	164
20.8802998618507	-1753.07866559773	1033.11340463438	164
20.8802998618507	-1743.31395748811	1032.89814432568	164
20.8802998618507	-1733.56544730188	1032.65844572598	162
20.8802998618507	-1723.83204443107	1032.426941707	159
20.8802998618507	-1713.36451840754	1032.22501467144	160
20.8802998618507	-1705.15539469225	1032.02977698607	161
20.8802998618507	-1695.09349184248	1031.82953320005	162
20.8802998618507	-1686.1552745606	1031.78272785357	162
20.8802998618507	-1676.12230251011	1031.69788501343	166
20.8802998618507	-1667.20829095722	1031.68324693236	168
20.8802998618507	-1659.03796230356	1031.65915388458	167
20.8802998618507	-1651.60340240719	1031.62366521921	164

Failis olevad koordinaadid x, y, z on lokaalsed ehk ei ole otseselt seotud geograafiliste koordinaatide süsteemiga. *X-Value* näitab kaugust mõõtmispunkti algusest. *Y-value*

näitab nihet tee keskpunktist risti suunas, jääb vahemikku -2000mm kuni 2000mm. *Z-Value* on kõrgus millimeetrites, mille kohta Teede Tehnokeskuse inimesed ei osanud täpselt öelda miks seda just 1000 ligidalt mõõtma hakatakse, kuid see ei sega suhtelise kõrguse leidmist. Arvati, et laser on kalibreeritud 1000 millimeetrile kui nullpunktina. Kõik neli veergu loetakse programmi sisse. Üks rida failis vastab ühele punktile. Üks profiil ehk tee ristlõige vastab 540 punktile. Täpsem näide tee profiilist on toodud Lisas 1.

Varasemalt pani Teede Tehnokeskuse töötajal punktipilve faile käsitsi kokku, kuna tarkvara ega skanner seda loetaval kujul ei genereerinud. Skanner genereerib hulga binaarfaile, mida skanneri tarkvara oskab lugeda. Selleks, et saada loetaval kujul punktipilve andmed, tuli kasutada töölauarakenduse operatiivmällu kopeerimise võimalust. See võimaldas kopeerida lõigu andmed kõigepealt arvuti operatiivmällu ja seejärel oli võimalik faili ümber kopeerida. Kuna kasutajaliides polnud kõige mugavam, toimus kopeerimine tihti ülekattega. See tekitas dubleeritud andmeid. Selleks tekitati töö käigus lahendus, mis eemaldas faili sisselugemisel dubleeritud punktid.

Üsna lõputöö praktilise osa lõpus sai Teede Tehnokeskus ViaPPS poolse tarkvarauuenduse, mis lasi punktipilve faile genereerida samamoodi nagu GPS faile. Genereeritakse terve lõigu fail ilma käsitsi kopeerimisteta. Seega otsustati duplikaatide eemaldamise osa projektist eemaldada, kuna duplikaatide kontrollimine võttis teatud osa mälu ja aega. Sellest, kuidas ülekattega tekkinud duplikaatidest projekti algusepoole vabaneti, räägitakse täpsemalt Lisas 2.

3.3 GPS faili andmed

GPS fail sisaldab 56 veergu, sellest programmi loetakse 5. Need 5 veergu on: *meter*, *height*, *latitude*, *longitude* ja *heading*. Üks rida failis on mõõdetud ligikaudu iga 10-30cm sammu tagant ning kujutab punkti tee keskel. Selles failis olevad geograafilised koordinaadid *latitude* ja *longitude* tuleb ühendada punktipilve lokaalsete punktidega. Lõik GPS failist ainult vajaminevate veergudega on näidatud Tabel 3 abil.

Tabel 3. Lõik GPS failist vajaminevate veergudega.

Meter [m]	**GPS Height [m]**	**Latitude [°]**	**Longitude [°]**	**Heading [deg]**
20.77	49.16	58.6039168937500	26.2091680375000	128.390000
20.95	49.16	58.6039160187500	26.2091701500000	128.390000
21.12	49.16	58.6039151437500	26.2091722312500	128.370000
21.28	49.16	58.6039142625000	26.2091743125000	128.360000
21.44	49.16	58.6039133875000	26.2091763937500	128.350000
21.60	49.15	58.6039125125000	26.2091784750000	128.340000
21.76	49.15	58.6039116625000	26.2091805625000	128.320000
21.93	49.15	58.6039107875000	26.2091826500000	128.310000
22.09	49.15	58.6039099125000	26.2091847312500	128.310000
22.26	49.15	58.6039090375000	26.2091868187500	128.300000
22.40	49.15	58.6039081562500	26.2091889125000	128.290000
22.55	49.14	58.6039073250000	26.2091909187500	128.270000
22.70	49.14	58.6039064437500	26.2091930125000	128.270000
22.85	49.14	58.6039055562500	26.2091951125000	128.280000
23.01	49.14	58.6039046750000	26.2091972187500	128.280000
23.16	49.14	58.6039037875000	26.2091993312500	128.290000
23.31	49.14	58.6039029500000	26.2092013687500	128.310000
23.47	49.13	58.6039020562500	26.2092034875000	128.320000

Võti punktipilve ja GPS punkti ühendamiseks on *X-Value* punktipilve failist ja *meter* GPS failist – mõlemad näitavad kaugust mõõtmise alguspunktist meetrites. Küll aga ei saa neid ühendada rea- ega profiilipõhiselt, kuna samm GPS failis ja punktipilve failis ei ühti. GPS faili samm varieerub rohkem ning on tavaliselt suurem kui punktipilve samm. GPS faili sammu erinevuse üheks põhjuseks on liikumiskiiruse muutus. Kuna GPS fail annab iga sammu kohta ainult ühe punkti ja sammud ei ühti, tuleb leida lahendus kõigepealt tee keskmise punkti ühendamiseks piki teed. Järgmisena kasutatakse leitud keskmist punkti, ülejäänud profiili punktide leidmiseks, mis on teega risti. Algselt on ühes profiilis 540 punkti, aga peale hõrendamist vähem. Sellest, kuidas punktide ühendamine käib räägitakse põhjalikumalt punktis 4.6.

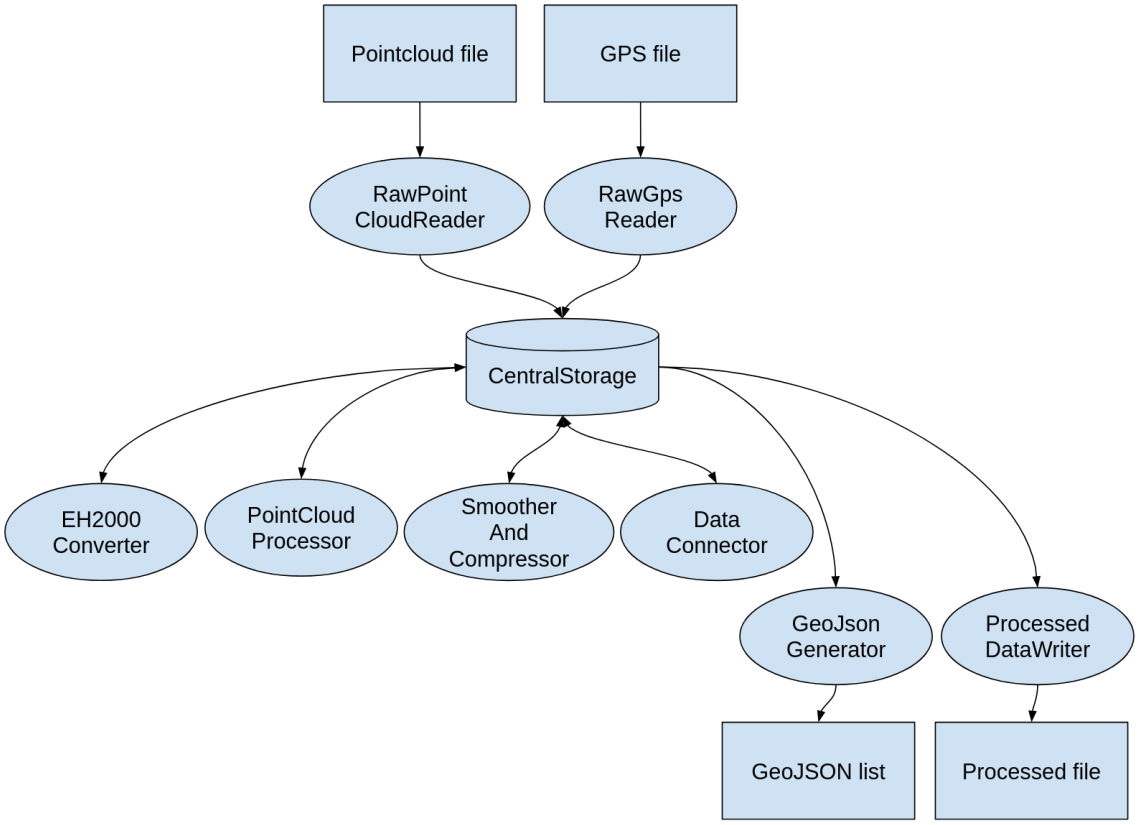
Peale *latitude*, *longitude* ja *meter* veerge, loetakse veel sisse *GPS height* ja *heading*. *GPS Height* on absoluutne kõrgus, see loetakse sisse, kuna Teede Tehnokeskus soovis, et see veerg oleks lõppfailis olemas mis tuleb abiks arvutustel. *Heading'*ut on vaja teada kui hakatakse arvutama teega risti olevaid geograafilisi punkti koordinaate, et teada mis suunas tee kulgeb ja oleks võimalik leida teega risti suund. See võimaldab leida profiili geograafilised punktid. *Heading* veergu lõppfaili ei lisata.

Ühe kilomeetrise GPS faili suurus on ligikaudu 1.2MB. Erinevalt punktipilve failist, on see oluliselt väiksem kui genereeritav punktipilve fail sama pika lõigu jaoks, kuna GPS failis on iga sammu kohta antud üks punkt, aga punktipilve failis 540 punkti.

4 Andmetöötlus

Andmetöötlus sisaldab järgmisi etappe. Etapid on alampunktidena toodud välja samas järjekorras nagu neid tegelikult programmi siseselt käivitatakse. Kohati on võetud eeskuju Potteri [6] magistritööst kus tehti kindlaks, et läbitav protsess annab tulemuse.

Joonis 2 abil on välja toodud andmetöötluse peamised komponendid. Ristkülikud kujutavad sisend- ja väljundandmeid, ovaalid Java objekte samade nimedega nagu lähtekoodis neid kasutati. *CentralStorage* on samuti Java objekt, selle eesmärk oli hoida kõige värskemaid andmeid peale iga töötluse vahesammu täitmist.



Joonis 2. Andmetöötluse üldskeem.

4.1 EH2000 kõrguse Maa-ameti kaudu konverteerimine

Kui punktipilve ja GPS faili punktid on programmi sisse loetud, saab alustada töötlemisega. Alustatakse EH2000 kõrguste konverteerimisega, kuna GPS punkte

eraldisesivalt kogu protsessi käigus ei muudeta, vaid hiljem liidetakse kokku punktipilve punktidega, võib need kohe esimese sammuna ära konverteerida. Enne alampunkti 4.6 GPS punkte rohkem ei puututa.

Hiljuti võeti Eestis kasutusele uus kõrguse mõõtmise süsteem, lühendiga EH2000, mis kasutab Kroonilinna nulli asemel Amsterdami nulli [7]. Leiti, et see funktsionaalsus peaks programmis sees olema. EH2000 kõrguse arvutamise kohta avalikku ressurssi ei olnud mille põhjal saaks lahenduse ehitada. Selleks oli programmi kirjutamise hetkel olemas võimalus saata ühekaupa päringuid üksikute punktidena Maa-ametisse mis tagastas EH2000 kõrgusega punkti. See ei sobinud lahenduseks, kuna punkte oli piisavalt suur kogus, mis põhjustas serveri ülekoormuse paljude üksikute päringute tõttu. Teine võimalus oli taotleda EH2000 kõrguse mudel ja see kohandada oma programmile vastavaks. Autori ettepanekul, Teede Tehnokeskuse ja Maa-ameti koostöö tulemusena tekitati kolmanda võimalusena veebiteenus, kuhu saab kõik GPS punktid korraga üles laadida ühe päringuga. Üleslaetud punktid töödeldakse Maa-ametis sama mudeli põhjal, mida oli võimalik taotleda ning saadetakse EH2000 kõrgustena tagasi. See lahendus saigi kasutusse võetud.

Kui üldjuhul on andmemahud suured, siis Maa-ametisse saatmisel see probleemiks ei osutunud. Maa ametisse saadetakse ainult GPS punktid, mida on üks iga paarikümne sentimeetri kohta. Konteksti mõttes võib tuua siin näite: 1km GPS fail on algselt 1.2MB ning sisaldab 56 veergu. Maa-ametisse saadeti siit 3 veergu: *latitude*, *longitude* ja *GPS height*. Testifaili suuruseks jäi pärast üleliigsete veergude eemaldamist 90KB. Lisaks on kasutajal võimalus EH2000 konverteerimine vahele jätta, kui konverteerimist ei nähta vajalikuks või peaks esinema võrguprobleem.

4.2 Profiili sammu valimine ja relatiivse kõrguse leidmine

Kasutajal on võimalik veebirakenduse kaudu valida, mitme sammu kaupa profiile alles jäetakse. Teostatakse moodularvutus, et leida kas profiil jääb alles või mitte. Sammu pikemaks valimisel kui 1 on teatud kitsaskoht visualiseerimise suhtes. Kui lõigus on palju defektseid profiile ja valitakse, et jätta alles näiteks iga 5. profiil, aga 5. ja 10. profiil on defektiga, siis need ikkagi eemaldatakse ja jääb visualiseerides sisse suurem vahe ning hõredam esitlus võimendab puuduvaid profiile.

Järgmiseks on vaja leida punktide relatiivsed kõrgused. Selleks kasutati eelnevalt Potteri [6] leitud lahendust MATLAB'ist. Kuna Teede Tehnokeskus soovib MATLAB'i vältida MATLAB'i litsentsi tõttu, realiseeriti relatiivsete kõrguse osa Javas. Relatiivseid kõrguseid on vaja, kuna ilma nendeta on keeruline tekitada visuaalset ettekujutust. Algses punktipilve failis on kõrguse nullpunktiks võetud arv, mis on 1000 ligidal ning see on antud ilma teekallet arvesse võtmata.

Potteri järgi [6] näeb relatiivse kõrguse leidmine välja järgmiselt. Leitakse profiili algusest ja lõpust 5% punktide ulatuses kaks mediaani. Need kaks mediaani tagavad, et joon mis leitakse ei põhineks ainult kahel äärmisel punktil. Ainult äärmised punktid võivad olla mürast häiritud. Võetakse mediaan 5% ulatuses mõlemast profiili otsast mis oluliselt vähendab müra probleemi.

$$y = kx + b \tag{1}$$

$$z = ky + b \tag{2}$$

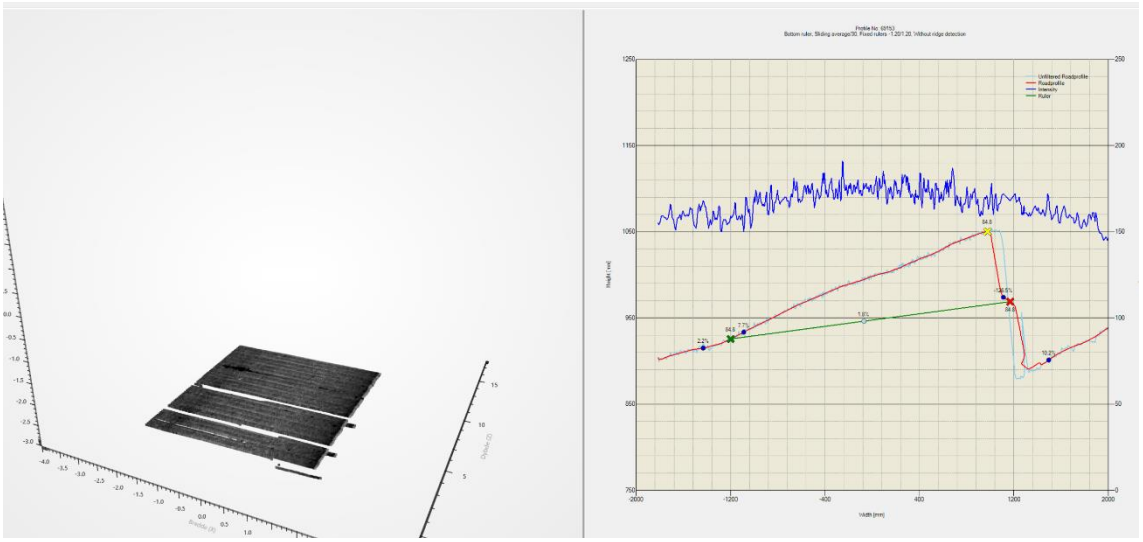
$$b = z - ky \tag{3}$$

Kõrguse joon leitakse sirge võrrandiga (1). Kuna tervikuna lahendatakse ülesannet 3D vaates, aga joont vaadatakse 2D vaates, siis õigete telgede nimetustega näeks sirge joone valem välja nagu valem (2). *Y* väärtuseks tegelikult on *z* ja *x* väärtuseks *y*. Kõrguste otspunktideks on mediaani abil leitud kõrgused. Tõus *k* leitakse mediaanide kõrguste vahe jagamisel tee laiusega. Leitud tõusu *k* korrutades läbi 100-ga saadakse ühtlasi protsendilisel kujul teeraja kalle ehk *crossfall*. *Crossfall* on oluline tee kvaliteedi ja ohutuse tagamiseks [8]. Väärtus *b* leitakse esimeseks punktiks joonel valemist (3). *Z* väärtusena kasutatakse alguse mediaani ja *y* väärtustena kasutatakse kõige esimest profiili punkti väärtust – see paneb paika joone esimese punkti mille abil leitakse ka järgmised joone punktid. Lõpuks leitakse punktipilve failis oleva *z* väärtuse ja joone *z* väärtuse vahe mis ongi relatiivne kõrgus.

4.3 Defektsete profiilide eemaldamine

Üsna alguses peale relatiivsete kõrguste leidmist selgus, et teatud profiilidega on kõrvuti olevate punktide kõrguste vahed ebareaalselt suured. Kõrvuti olevate punktide vahekaugus profiilis jääb keskmiselt 7-8mm juurde. Joonis 3 abil on näha, et umbes 30cm nihkega profiilis oli tee kõrguste vahe 16cm. Tuli lähemalt uurida mis seda põhjustab.

Selgus, et suured kõrguste vahed tekivad kohtades, kus on teel teravam pragu või auk. Probleemi sai kindlaks teha, kuna Teede Tehnokeskus lisaks pildistab ja mõõdab kogu skaneeritud teekonda. Kui vaadata profiile milles on kõrvuti olevate punktide vahed väga suured ja samal kaugusel pildistatud kohta kus profiil reaalselt asub, selgus, et tegu on ebatasasustega teel. Need ebatasasused liigutavad mõõtebussi praost või august läbi sõites piisavalt kiiresti, et ajada skanneri mõõdikud korraks valeks ning registreeritakse valed andmed. Otsustati, et kõige mõistlikum on defektsed profiilid lõppfailist ning visualiseerimisest välja jätta.



Joonis 3. Näide defektsest profiilist (punane joon).

Profiil ehk 540 punkti märgitakse defektseks ja eemaldatakse kui kogu profiil ei jää nõuete piiridesse. Nõudeid on kaks. Esiteks, profiili keskmine kõrgus peab olema suurem kui 980. Valiti väärtus 980, kuna see tundus Teede Tehnokeskuse töötaja kogemuse põhjal mõistlik piir ja ka visuaalselt testides pidas see paika. Teiseks nõudeks oli, et profiilis kõrvuti olevate punktide kõrguste vahe ei tohi olla suurem 2-st millimeetrist. Tavaliselt defektsed profiilid sisaldavad kõrvuti punkte, kus kõrguste vahed muutuvad äkiliselt. Viimast tingimust saab läbi veebirakenduse seadistada, kuna mõningatel lõikudel võib 2 millimeetrit olla liiga väike vahemik.

4.4 Silumine profiilipõhiselt

Enne andmete hõrendamist teostatakse andmete silumine. See tagab andmete efektiivsema hõrendamise. See tuli katsetuste tulemusena ka välja. Kui lülitada silumine välja enne hõrendamist, genereeriti töödeldud fail suurema mahuga.

Silumise eesmärgiks on andmeid ühtlustada eemaldades müra, kuid samal ajal jätta alles võimalikult palju algset infot. Andmete profiilide kaupa silumiseks kasutatakse Savitzky-Golay filtrit[1] samade väärtustega mis Potteri töö [6] järeldusena olid optimaalsed ehk akent suurusega 50 ja 11. astme polünoomi. Aken suurusega 50 tähendab, et polünoomi leidmiseks kasutatakse 25 punkti taha ja 25 punkti ette jäävaid punkte praegusest asukohast.

4.5 Hõrendamine profiilipõhiselt

Andmeid pärast defektsete profiilide eemaldamist on endiselt väga palju ning neid tuleks vähendada. Selleks kasutatakse Douglas-Peucker'i algoritmi [9]. Hõrendamist teostatakse nagu relatiivsete kõrguste ja silumise protsessi ehk iga profiili töödeldakse eraldi.

Douglas-Peucker'i algoritm eemaldab punktid, kus ei ole suurt kõrvalekallet. Alles jäetakse punktid, mis jäävad tolerantsi piirangust välja. Mida suurem on tolerants, seda vähem punkte tolerantsi piirangust välja läheb ja alles jäetakse [10]. Hõrendamise tolerantsi saab kasutaja ise veebirakenduse kaudu muuta.

4.6 Töödeldud andmete kokku ühendamine

Pärast eelnevaid protsesse, ühendatakse kokku töödeldud punktipilve punktid ja GPS punktid. Ühendamiseks kasutatakse alusena GPS punkte ning vaadatakse järjest läbi punktipilve punkte.

4.6.1 Punktide ühendamise kriteeriumid

Ühendatakse punktid, kus GPS faili *meter* veeru väärtus jääb nõutud kauguse piiridesse punktipilve *X-Value* veeru väärtusega. Efektiivsust silmas pidades ei hakata kumbagi listi algusest uuesti vaatama. Väliselt itereeritakse GPS punkte ja sisemiselt punktipilve punkte. Punktipilve asukohta peetakse alati meeles, et ka järgmise punkti ühendamisel saaks jätkata samas kohast + 1. Punktipilve väärtus peab alati olema suurem GPS väärtusest ja peab jääma nõutud piiridesse ehk vahe ei tohi olla liiga suur. Vastasel juhul

[1] https://github.com/swallez/savitzky-golay-filter/blob/master/src/mr/go/sgfilter/SGFilter.java

peatatakse sisemise listi itereerimine koheselt ning valitakse järgmine GPS punkt mis on lähemal punktipilve punktile. Optimaalseks vaheks osutus 0.4 meetrit, millega saab enamus punkte kaetud.

Sellise lahenduse mõte on esiteks, kuna sammud on erineva pikkusega punktipilve ja GPS failis, kasutatakse vahemiku piiri, mille sees olevad punktid ühendatakse. Teiseks, punktipilve ja GPS faili lõigud ei pea olema sama pikkusega, oluline on, et need mingil vahemikul ühtiksid. See tähendab, et kui on ühtivad failid, siis alati on GPS failis olemas punkt mis on väiksem kui punktipilve punkt ja vahe on nõutud vahemikus, isegi kui failid pole sama pikkusega.

4.6.2 Punkti interpoleerimine piki teed

Selguse huvides kasutatakse valemis (4) ja (5) pikkuskraade asemel *x* ja laiuskraadide asemel *y*. *X1*, *y1* on interpoleeritud punktile eelnev ja *x2*, *y2* järgnev punkt. *X*, *y* on otsitava interpoleeritud punkti geograafilised koordinaadid.

$$x = x_1 * \frac{protsent1}{100} + x_2 * \frac{protsent2}{100} \quad (4)$$

$$y = y_1 + (x - x_1)\frac{y_2 - y_1}{x_2 - x_1} \quad (5)$$

Kui sobiv vahemik *X-Value* ja *GPS meter* veeru alusel on leitud mida ühendada, interpoleeritakse punkt eelmise ja järgmise GPS punkti vahele. Leitakse geograafilised pikkus- ja laiuskraadid. Interpoleerimisega saadakse punkti täpsem asukoht piki teed ning täpselt tee keskele kus veel ei arvestata *Y-Value* risti telje nihet.

Kõigepealt leitakse pikkuskraadid. Piki teed interpoleerimiseks leitakse kauguste vahed nii eelmisesse kui ka järgmisesse punkti punktipilve *X-value* ja *GPS meter* põhjal. Leitakse kumb vahe on väiksem, see omab suuremat protsendilist kaalu. Valemis (1) *protsent1* ja *protsent2* leitakse mõlema kauguste summana punkti, mida soovitakse interpoleerida. Need kaugused liidetakse ja summast leitakse mõlema kauguse protsendiline osakaal ning kontrollitakse, et see kaugus mis on väiksem, oleks lõpuks suurema protsendilise väärtusega, kuna väärtus, mis oli interpoleeritavale punktile lähemal mõjutab rohkem saadavat väärtust. Lõpuks kasutatakse valemit (4) kus leitakse interpoleeritava punkti *x* ehk pikkuskraad.

Kui piki teed koordinaat *x* on valemi (4) põhjal käes, tuleb leida ka teise telje geograafiline koordinaat. Selleks kasutatakse lineaarse interpoleerimise valemit kahe punkti põhjal [11]. Lineaarse interpoleerimise valemis (5) *x*, *y* on otsitavad, kus *x* on juba leitud. Leitakse valemi (5) abil interpoleeritava punkti *y* ehk laiuskraad.

4.6.3 Tõeliste koordinaatide leidmine

Kui kahe punkti vahele on punkt tee keskele interpoleeritud, tuleb leida tõelised pikkus ja laiuskraadid kus võetakse arvesse punktipilve failist y telge ehk kui suur on nihe tee keskelt risti suunas. Selleks, et saada punkt ka geograafiliselt teega risti, kasutatakse GPS failist veergu *heading*, mis ütleb, mis suunas tee kulgeb. *Heading* + 90 kraadi annab teega risti suuna. Punktipilve failist *Y-Value* veergu arvesse võttes, mis on nihe millimeetrites, saab leida tõelise geograafilise koordinaadi.

$$trueX = x + vahe * (cos(90 + heading)) \quad (6)$$

$$trueY = y + vahe * (sin(90 + heading)) \quad (7)$$

Detailsemalt näeb see välja nii, et saada teada tõelised koordinaadid, leitakse kõigepealt ilma nurka arvesse võtmata nihkega geograafilised koordinaadid. Selleks kasutati valemit mis leiti lehelt „Stack Overflow“[1]. Kui uued geograafilised koordinaadid nihkega saadi teada mõlema telje jaoks, lahutati see algsetest koordinaatidest, et teada koordinaatide vahe – see vahe on valemites (6) ja (7) toodud välja muutujaga *vahe*. Viimaseks sammuks leitakse valemite (6) ja (7) abil muutujad *trueX* ja *trueY* ehk tõeline pikkus- ja laiuskraad interpoleeritavale punktile. Valemis liidetakse algsele koordinaadile juurde koordinaatide vahe mis on läbi korrutatud *heading* + 90 kraadi[2] millest võeti kas *sin* või *cos*, olenevalt kumba koordinaati parajasti leiti.

[1] https://stackoverflow.com/a/7478827/8707346

[2] https://se.mathworks.com/matlabcentral/answers/123482-calculation-the-coordinate-of-a-point#answer_130916

4.7 LEST koordinaatide ja absoluutse GPS kõrguse leidmine

Peale punkti interpoleerimist, leitakse igale punktile LEST ristkoordinaatide süsteemis x ja y koordinaat, mida Teede Tehnokeskus soovib kasutada erinevate arvutuste tarbeks. Selleks kasutati Maa-ameti poolset PHP koodi[1], mis konverteeriti ümber Javasse.

GPS failiga on kaasas absoluutne kõrgus, see on olemas ainult ühele punktile tee keskel ehk üks kõrgus profiili kohta. Et kõik ühe profiili punktid ei oleks lõppfailis sama kõrgusega, liidetakse GPS kõrgusele juurde kõrgus punktipilve failist ehk *Z-Value*. *Z-Value* kõrgusest lahutati 1000, et eemaldada skanneri poolt lisatav 1000 alampiir ja jagati 1000-ga, kuna GPS kõrgus on meetrites, aga lokaalne kõrgus millimeetrites. Võib tunduda, et lihtsam oleks liita GPS kõrgusele juurde iga punkti uus leitud relatiivne kõrgus, kuid relatiivne kõrgus on leitud arvestades tee kaldenurka. Absoluutse kõrguse juures tuleks leida kõrgus ilma kaldenurka arvestamata ning sellepärast seda lahendust ei kasutatud.

4.8 Faili salvestamine ja GeoJSON'i genereerimine

Protsessi lõpus salvestatakse töödeldud andmed faili kui kasutaja seda soovib. Andmete salvestamisel ja GeoJSON'i genereerimisel kasutatakse Java NumberFormat funktsionaalsust, mille abil seatakse täisarvu ja murdarvu eraldajaks alati punkt, olenemata kasutaja arvutis seadistatud regioonist. Lisaks seatakse NumberFormat'i abil maksimaalne murdarvu kohtade limiit. Faili salvestamisel aitab see omakorda faili suurust vähendada, kuna mõnes veerus, näiteks *X-Value*, *Y-Value*, *Z-Value*, võib algselt olla kuni 13 kohta pärast punkti, jäetakse alles ainult 3. Koordinaatide puhul jäetakse alles kuni 8 kohta peale punkti, et tagada koordinaatide piisav täpsus. Lõik lõppfailist on näidatud Tabel 4 abil.

Toore näitefaili suurus oli 220MB ja GPS faili suurus 1.2MB. Lõppfail suurus algseadistustega tekitati 16.2MB. Lõppfailis andmete eraldajaks valiti koma märk ja failivorminuks .txt, kuna lõppfaili soovitakse sisse lugeda Teede Tehnokeskuses AutoCAD tarkvaraga ja see oli nendepoolne soovitud seadistus.

[1] https://www.maaamet.ee/rr/geo-lest/files/geo-lest_function_php.txt.

Kõige viimase sammuna saadetakse alati *back-end*'ist *front-end*'i tagasi GeoJSON list. Seda on vaja, kuna Mapbox oskab lugeda GeoJSON objekte, neid filtreerida ja kuvada. Kui kasutaja ei soovi kaardile punkte kuvada, saadetakse tagasi tühi GeoJSON list.

Tabel 4. Lõik lõppfailist (height(GPS) EH2000 kõrgustena).

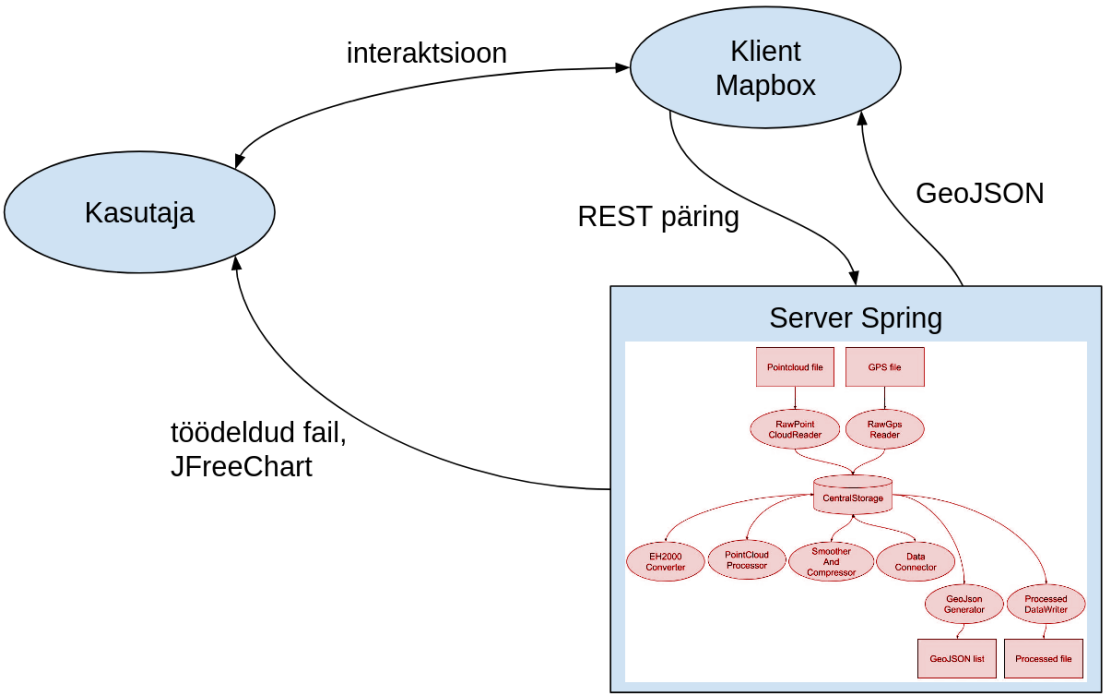
xValue	yValue	zValue	intensity	relativeHeight	crossfall	longitude	latitude	height(GPS)	xLEST	yLEST
20.88	-1811.584	1034.68	155	1.662	-2.165	26.20918412	58.60393107	30.355	6498114.203	628396.805
20.88	-1772.649	1033.678	166	2.785	-2.165	26.2091837	58.6039308	30.354	6498114.172	628396.782
20.88	-1753.079	1033.113	164	2.876	-2.165	26.20918349	58.60393066	30.353	6498114.156	628396.77
20.88	-1705.155	1032.03	161	2.51	-2.165	26.20918298	58.60393032	30.352	6498114.118	628396.742
20.88	-1458.176	1029.071	174	4.915	-2.165	26.20918033	58.60392858	30.349	6498113.919	628396.594
20.88	-1238.288	1023.043	181	3.58	-2.165	26.20917797	58.60392703	30.343	6498113.742	628396.463
20.88	-1082.74	1019.264	173	3.266	-2.165	26.2091763	58.60392594	30.339	6498113.617	628396.37
20.88	-948.609	1016.343	176	3.217	-2.165	26.20917486	58.60392499	30.336	6498113.509	628396.29
20.88	-901.486	1015.621	172	3.508	-2.165	26.20917436	58.60392466	30.336	6498113.471	628396.262
20.88	-847.257	1014.388	169	3.426	-2.165	26.20917378	58.60392428	30.334	6498113.427	628396.229
20.88	-826.421	1013.957	175	3.533	-2.165	26.20917355	58.60392413	30.334	6498113.41	628396.217
20.88	-374.748	1012.811	173	12.106	-2.165	26.20916871	58.60392095	30.333	6498113.047	628395.947
20.88	-285.214	1012.002	177	13.199	-2.165	26.20916775	58.60392032	30.332	6498112.975	628395.894
20.88	-120.768	1008.751	177	13.655	-2.165	26.20916598	58.60391916	30.329	6498112.842	628395.795
20.88	-44.971	1006.927	177	13.338	-2.165	26.20916517	58.60391862	30.327	6498112.781	628395.75
20.88	-10.023	1006.176	175	13.429	-2.165	26.2091648	58.60391838	30.326	6498112.753	628395.729
20.88	36.517	1005.119	177	13.341	-2.165	26.2091643	58.60391805	30.325	6498112.716	628395.701
20.88	83.247	1004.258	179	13.5	-2.165	26.2091638	58.60391772	30.324	6498112.678	628395.673

5 Veebirakendus ja andmete visualiseerimine

Töödeldavate andmete valimise, seadistuste muutmise ja kuvamise jaoks loodi veebirakendus.

5.1 Rakenduse ülesehitus

Rakendus sarnaneb klient-server arhitektuurile, mida illustreerib Joonis 4. Kui klient soovib saada töödeldud andmeid, saadetakse serverile korraldus andmete töötlemiseks, mille server täidab. Rakenduse käivitamisel ja brauseris sisestades http://localhost:9000 avaneb algselt tühi Mapbox'i kaart, millel kasutatakse Bootsrap *Modal* elementi, seadistusmenüü kuvamiseks.

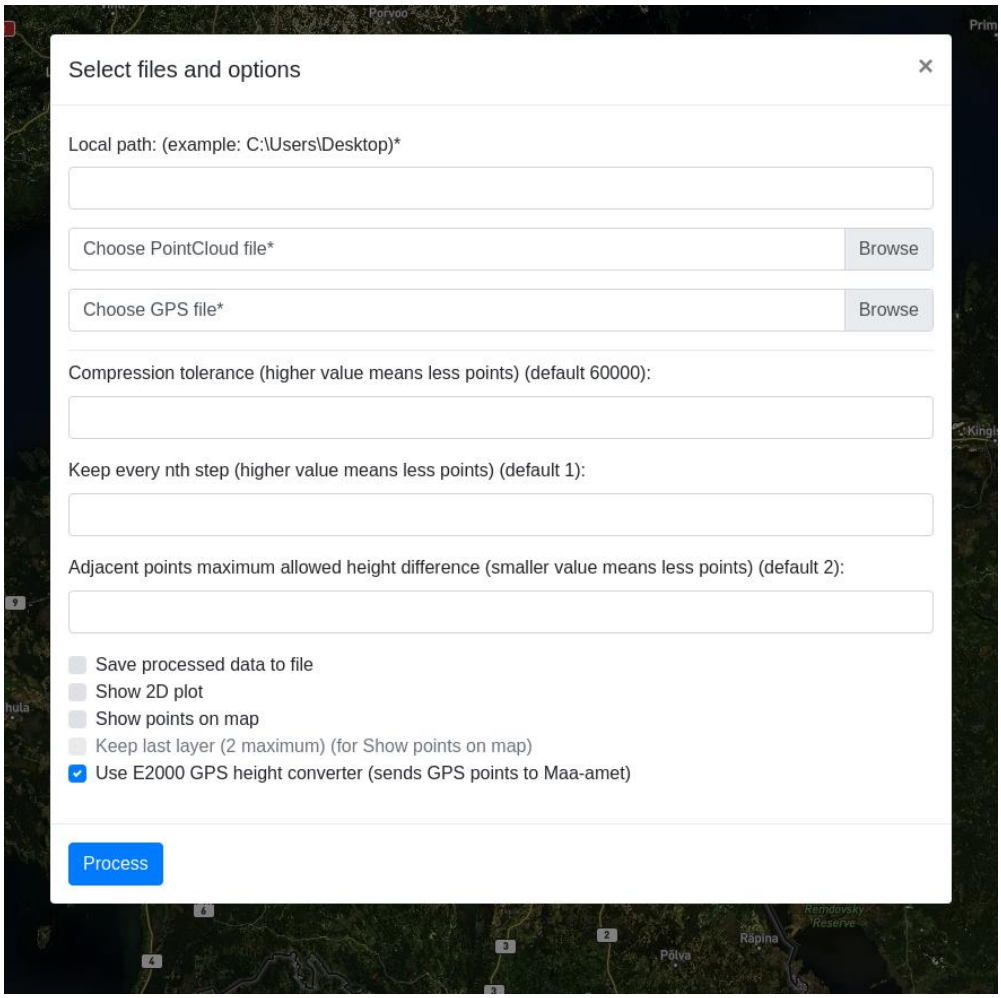


Joonis 4. Süsteemi arhitektuur.

5.1.1 Rakenduse menüü valikud ja päringute järjekord

Kasutajal on kohustuslik täita minimaalselt kolm esimest välja failide ja seadistuste valimise menüüst, need on märgitud tärniga, ja valida vähemalt üks tegevuse märkeruut

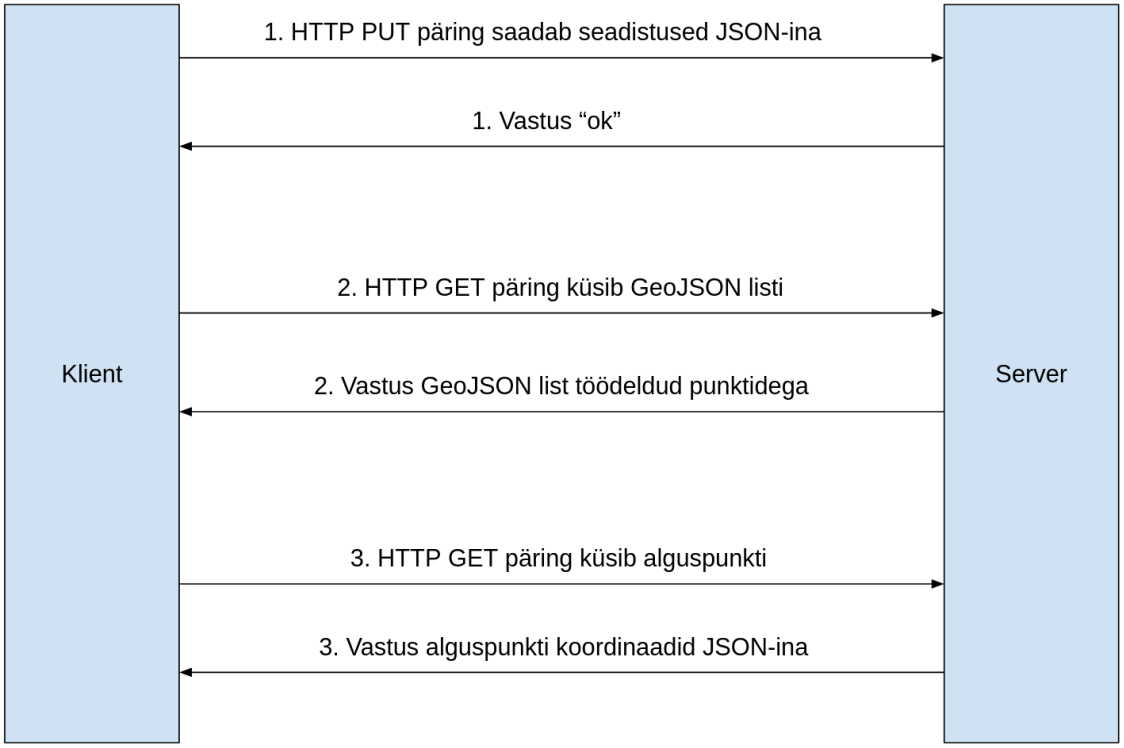
(erandiks on EH2000 arvutamine). Ülejäänud väljad võib soovi korral jätta tühjaks ning sel juhul ei kirjutata algväärtusi serveris üle. Ekraanipilt, kuidas failide ja seadistuste valimise menüü välja näeb on toodud Joonis 5 abil.



Joonis 5. Näide failide ja seadistuste valimise menüüst.

Kogu protsessi jooksul toimub kolm päringut serverile nagu näidatud Joonis 6 abil. Esimese HTTP PUT päringuga saadetakse serverile kõik kasutaja poolt valitud parameetrid. Kui klient saab vastuse, et server sai parameetrid kätte, toimub Mapbox'i GET päring. Mapbox lubab omaltpoolt saata ainult GET päringut, mis ootab vastuseks GeoJSON listi. Kaks esimest päringut tehakse eraldi, kuna kasutaja poolt sisestatud parameetreid on mugavam saata PUT päringu kehas, JSON formaadis, mida *back-end*'is saab otse Java objektiks teisendada ja vastupidi. Lisaks läheb see paremini kokku REST arhitektuuriga [12] kus andmete küsimiseks kasutatakse HTTP GET ja muutmiseks HTTP PUT päringut. Kasutatakse PUT päringut POST asemel, kuna tegu on idempotentse päringuga, mis tähendab, et mitu korda sama päringut saates, andmed ei muutu.

Kolmas ja viimane GET päring saadetakse kliendipoolt välja kui Mapbox'i API kaudu saadakse info, et kaart on laetud. Sel juhul sai Mapbox GeoJSON listi vastuseks ja punktid on kaardile lisatud. Kui kasutaja ei valinud *Show points on map*, siis saadeti tagasi tühi GeoJSON list. Viimase päringu idee on, et saada teada kuna töötlus on lõppenud ja *Show points on map* valiku puhul oleks teada, kuhu kaart sisse suumida. Päringuga küsitakse, mis on töödeldud punktide listi esimese punkti geograafilised koordinaadid. Kui kaardile punkte ei soovita kuvada, siis kaarti kuhugi sisse ei suumita ning kasutatakse algseadistatud vaadet ehk kuvatakse tervet Eesti kaarti.



Joonis 6. HTTP Päringute ja vastuste diagramm.

5.1.2 JFreeChart'i kuvamine ja faili salvestamine

Nii klient kui server mõlemad töötavad kasutaja arvutis. Kui kasutaja saadab päringu serverile omaltpoolt valitud parameetritega, teostab server saadud parameetrite põhjal andmete töötluse ja saadab vastuse. Erandiks on tavalise klient-server arhitektuurist JFreeChart diagrammi kuvamine ja faili salvestamine.

Vastava parameetri saatmisel, käivitatakse JFreeChart otse serveris. Kui soovitakse faili salvestada, siis on alati salvestamise koht teada ning kuna nii klient ja server töötavad samas arvutis, salvestatakse töödeldud andmed otse kasutaja valitud kohta tekstifailina.

5.1.3 Rakenduse tagasiside

Kasutajal on soovi korral võimalik näha terminalist serveri logist täiendavat infot. Näiteks allesjäänud punktide arv peale töötlust ja töötluseks kulunud aeg. Lõik serveri logist on näidatud Joonis 7 abil.

```
Path: /home/xps/Desktop
PointCloud file: Fy11_f1_20170609_1km.txt, GPS file: GPS_Fy11_felt1___20170609_1km.txt
Tolerance: 60000.0, Step: 1, Adjacent points dif: 2.0
SaveToFile: true, show2D: true, showMap: true, EH2000:true
Started uploading GPS points to Maa-amet
Received new GPS heights from Maa-amet
Defective profile points removed: 17280
Number of points smoothing and compression removed: 3926529
Final points: 178071
Process time: 20851 ms
StartPoint: 58.603931071810855 26.209184117937937
```

Joonis 7. Rakenduse terminali logi 1 kilomeetrise lõigu töötlemisel.

Lisaks terminali tagasisidele antakse tagasisidet programmi hetkeseisust menüü avamise nupu värvi muutmisega. Kui menüü avamise nupp on roheline, siis töötlust ei toimu ning kõik töötab nagu peab. Kui nupp on kollane, siis töötluse käigus läks midagi valesti, näiteks anti valed sisendfailid. Punane nupp tähendab, et töötlus parajasti käib.

5.1.4 Pikemate lõikude töötlus

Kui soovitakse töödelda suuri lõike, näiteks 1 kilomeeter ja pikemaid, tuleb Java JAR'i käivitamisel anda kaasa lisaparameeter -Xmx mille lõppu lisatakse näiteks gigabaitides arv (näide: java -Xmx4G -jar RoadProject-1.0.jar). Xmx määrab ära, mis on maksimaalne operatiivmälu limiit, mida Java saab kasutada. 8GB puhul oli selleks algseadistuses piiranguks umbes 2GB. Kui kasutajal on palju vaba mälu, saab Xmx väärtust oluliselt suurendada ja töödelda oluliselt pikemaid lõike. Iga kord kui programm käivitatakse, näidatakse hetke Xmx väärtust. Xmx väärtuse kuvamise näide terminalist on toodud Joonis 8 abil.

```
Maximum possible RAM amount to Allocate for Java (XMX): 2067 MB
Open URL in browser: http://localhost:9000
```

Joonis 8. Java Xmx vaikimisi väärtus veebirakenduse käivitamisel 8GB mäluga.

Spring Boot serveripoolses koodis on meetodiks, kus toimub töötlus *getGeoJson()*. Kuna andmeid on palju ja serverit ei taaskäivitata peale iga töötluse käsu täitmist, siis leiti, et mõttekas on objektid, mis võivad sisaldada palju andmeid deklareerida ja initsialiseerida töötlust käivitatava meetodi sees. Idee seisneb selles, et peale igakordset meetodi väljakutset, peaks meetodi lõppu jõudes saama Java *Garbage Collector* vihje, et objektid

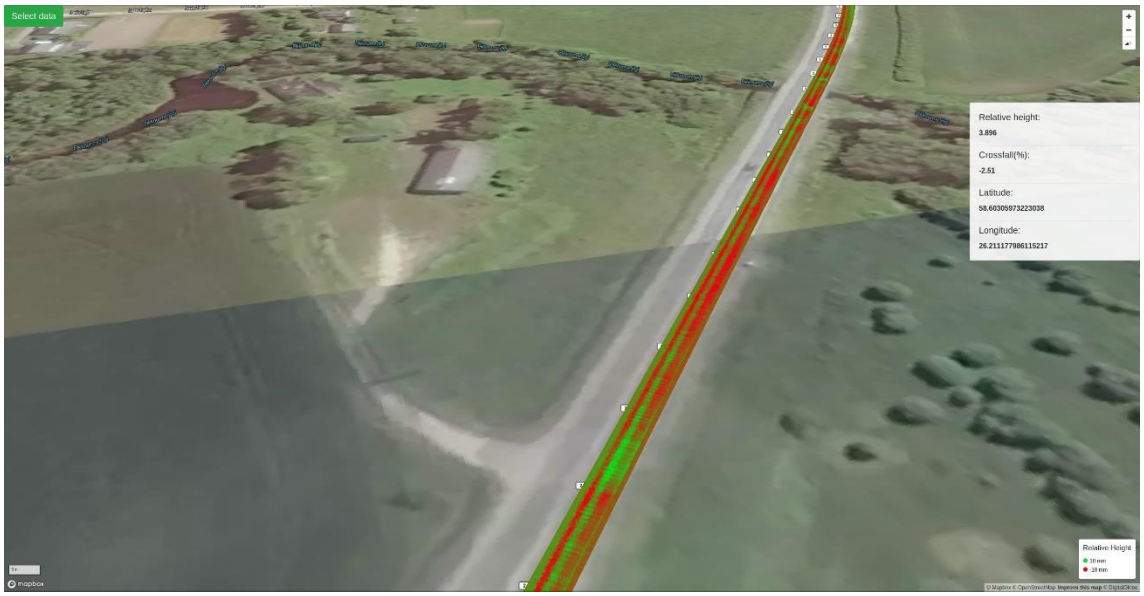
mis tekitati meetodi eluea jooksul meetodi sees, võib meetodi lõppedes kustutada. Java mälukasutust kontrolliti VisualVM tööriistaga[1]. Näide Java *heap* kasutusest on toodud Lisas 3.

5.2 Mapbox kaardipõhine visualiseerimine

Mapbox on OpenStreetMap[2] andmetepõhine kaardikihi pakkuja mis kasutab Leaflet[3] JavaScript teeki kaardi manipuleerimiseks. Eesmärgiks oli luua lahendus, kus oleks võimalik kaardile kuvada töödeldud faili punktid koos kõrgust eristavate värvidega. Selleks tehakse *front-end*'is Mapbox API kaudu päring Spring Boot'i, kus teostatakse peatükis 4 läbiviidavad protsessid ning saadakse tagasi GeoJSON list, mille objektide põhjal teostatakse visualiseerimine. Saadud list sisaldab GeoJSON objekte, kus igal objektil on antud *latitude*, *longitude*, *crossfall* ja *relativeHeight*.

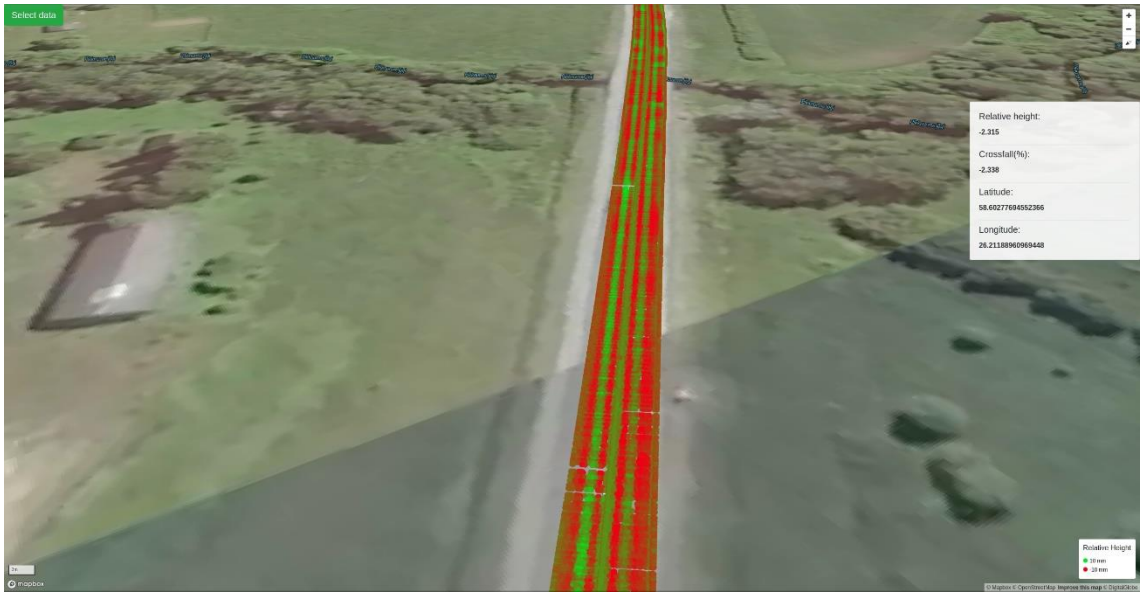
Mapbox API abil on võimalus värve interpoleerida GeoJSON objekti parameetri järgi, värvi parameetriks valiti iga objekti relatiivne kõrgus. Värvi interpoleerimise vahemikuks valiti -10mm kuni 10mm, kuna see katab suurema osa punktidest ja annab parema visuaalse ettekujutuse. Katsetati ka -20mm ja 20mm vahemikku ja kõigi punktide maksimum ja miinimum väärtust. Nendel juhtudel ei olnud värvid piisavalt erksad, sest maksimumi ja miinimumi piirile küünivad vähesed punktid. Näide Mapbox'i visuaalist on näidatud Joonis 9 abil.

[1] https://visualvm.github.io/
[2] https://www.openstreetmap.org
[3] http://leafletjs.com/



Joonis 9. Näide Mapbox'i visuaalist ühe kihiga.

Kaardivaates ilma *Keep last layer* valikuta kuvatakse ühte kihiti korraga. Iga uue töötluse tulemusena eemaldatakse vana ning lisatakse uus kiht uute punktidega. Kui valida menüüst märkeruut *Keep last layer*, hoitakse meeles eelmist kihti, millele saab juurde lisada uue kihi uute andmetega. Kui teostatakse sama protsessi uuesti, kustutatakse kõige vanem kiht ning näidatakse kahte uuemat. Kui lülitada *Keep last layer* välja, kustutatakse mõlemad kihid ning lisatakse uuesti ainult üks kiht kõige uuemate andmetega. Lõik Mapbox'i visuaalist kahe kihiga on näidatud Joonis 10 abil.



Joonis 10. Näide Mapbox'i visuaalist kahe kihiga.

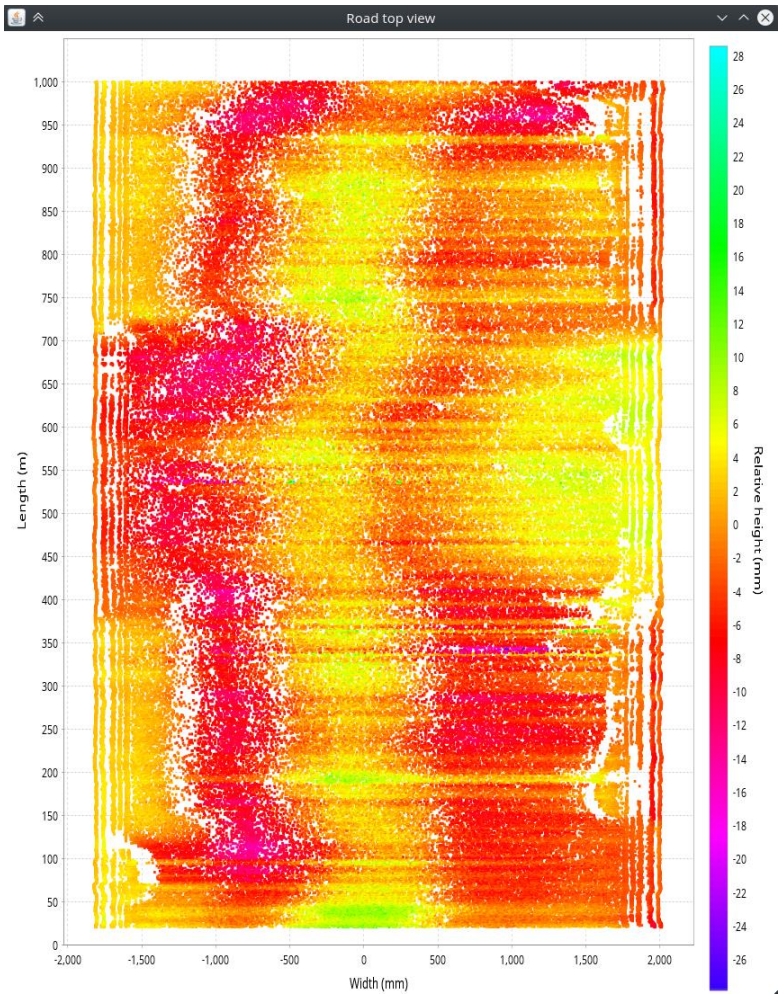
5.3 JFreeChart 2D diagrammipõhine visualiseerimine

Kuna Teede Tehnokeskusega arutati ka 2D diagrammipõhisest lahendusest, sai selleks valitud JFreeChart diagramm. See annab olukorra tee seisust pealtvaates. JFreeChart on teek Javas, mis võimaldab luua erinevaid tüüpi diagramme. Visualiseerimiseks kasutatakse töödeldud punktide *xValue*, *yValue* ja *relativeHeight* andmeid.

JFreeChart'i alternatiivne lahendus sai tehtud lisaks Mapbox'i kõrvale, kuna Mapbox annab küll visuaalselt ruumis parema ettekujutuse, kus teatud kõrgem või madalam koht asub, aga kasutab rohkem mälu kui lihtne 2D diagramm. Näiteks 10 kilomeetriste lõikude kuvamisel, hoitakse oluliselt mälu kokku kui vaadata ainult 2D diagrammi otse serverist. 2D diagrammi aluseks kasutati lahendust, mis leiti lehelt „Stack overflow"[1] ning modifitseeriti töö jaoks sobivaks.

2D diagrammi puhul on skaala dünaamiline. Skaala otspunktideks on relatiivsete kõrguste minimaalne ja maksimaalne väärtus lõigu kohta mida töödeldakse. Siin oli oluline kasutada minimaalselt ja maksimaalset väärust skaala vahemiku määramiseks, vastasel juhul hakkaksid värvid korduma mis läksid manuaalselt seatud skaala piiridest välja. Näide JFreeChart 2D diagrammist on näidatud Joonis 11 abil.

[1] https://stackoverflow.com/a/37235165/8707346



Joonis 11. Näide JFreeChart 2D visuaalist.

6 Kokkuvõte

Töö lõppeesmärgiks oli luua tarkvara, mille abil saab selge visuaalse ülevaate skaneeritud tee seisukorrast ja mis võimaldab salvestada töödeldud andmed. Töödeldud andmeid on Teede Tehnokeskusel hiljem võimalik kasutada täiendavataks arvutusteks. Antud töös seati kaks alameesmärki. Esiteks luua süsteem, mis võimaldab toorandmeid töödelda ja töödeldud andmeid salvestada. Teiseks luua veebirakendus, mis võimaldab kasutajal valida töötlusprotsessi teatud parameetreid, valida kuidas käitutakse töödeldud andmetega ning visualiseerida töödeldud andmeid.

Esimese eesmärgi täitmiseks implementeeriti serveripoolne lahendus andmete töötlemiseks ning salvestamiseks. Mõeldi läbi töötlusprotsessi sisemiste alamosade täitmise järjekord. Sammude realiseerimiseks tuli leida või kirjutada algoritmilised lahendused. Lisaks tuli kontrollida, et kogu protsess annaks tulemuse, aga samas ei jääks liiga aeglaseks. Esimeseks sammuks oli GPS kõrguste EH2000 kujule konverteerimine läbi Maa-ameti *endpoint*'i. Järgmisena leiti relatiivsed kõrgused, kus kasutati Potteri [6] varasemalt teostatud MATLAB'i lahendust, mis konverteeriti ümber Javasse. Lisati võimalus jätta alles profiile teatud sammuga. Eemaldati defektsed profiilid, kasutades selleks profiili keskmist kõrgust ning kõrvuti olevate punktide kõrguste vahe. Teostati andmete silumine Savitzky-Golay filtri abil. Teostati andmete hõrendamine Douglas-Peucker'i algoritmi abil. Eelviimaseks sammuks oli punktipilve ja GPS punktide pärast eelnevaid töötlusprotsesse kokku ühendamine ja LEST koordinaatide leidmine. Suurimaks probleemiks oli kahte eri tüüpi skanneri andmete ühendamine, mis täpselt omavahel ei ühildunud. Probleemi lahenduseks osutus sobiva kaugusevahemiku seadmine ja interpoleerimise põhjal punktide ühendamine. LEST koordinaatide leidmine teostati Maa-ameti PHP koodi põhiselt mis konverteeriti ümber Javasse. Viimaseks sammuks oli töödeldud punktide GeoJSON objektideks genereerimine ja vajadusel lõppfaili salvestamine.

Teise eesmärgi saavutamiseks kirjutati klient-server arhitektuurile sarnanev veebirakendus. Loodud veebirakendus ühendati varem ehitatud serveripoolse

töötlusprotsessi osaga. Kontrolliti, et peale iga töötlust ja visualiseerimist normaliseeruks mälukasutus ehk eelmised töödeldud punktid Javas ja kui sooviti kuvada punkte kaardile, siis kihid Mapbox'i kaardilt oleks mälust kustutatud. Veebirakenduses implementeeriti seadistusvõimalused toorandmete töötluseks, valikuvõimalused töödeldud andmetele ja punktide visualiseerimiseks. Suurimaks probleemiks oli, kuidas suurt hulka punkte on võimalik visualiseerida veebirakenduses. Probleemi lahenduseks osutus testimise tulemusena Mapbox API abil kaardile punktide kuvamine ja alternatiivina JFreeChart diagramm tee pealtvaatest.

Töö käigus valminud rakenduse viimane versioon anti kliendile üle aprilli alguses. Teede Tehnokeskuse töötaja sõnul leiab rakendus kasutust eelkõige mõõdetud andmetega tutvumiseks ja nüüd on neil olemas rakendus, millele saab tulevikus, soovi korral, lisada täiendusi ning lisafunktsionaalsust. Käesolev bakalaureusetöö teostati TTÜ Tarkvarateaduse instituudi Andmeteaduse töörühma ja Teede Tehnokeskuse vahelise koostöö raames.

Kasutatud kirjandus

[1] M. Soilán, B. Riveiro, J. Martínez-Sánchez, ja P. Arias, „Segmentation and classification of road markings using MLS data“, *ISPRS J. Photogramm. Remote Sens.*, kd 123, lk 94–103, jaan 2017.

[2] H. Guan, J. Li, Y. Yu, M. Chapman, ja C. Wang, „Automated Road Information Extraction From Mobile Laser Scanning Data“, *IEEE Trans. Intell. Transp. Syst.*, kd 16, nr 1, lk 194–205, veebr 2015.

[3] „ViaPPS - ViaTech AS“. [WWW] http://www.viatech.no/products.aspx?lang=en&id=6. (18.12.2017).

[4] „Mapbox GL JS API“, *Mapbox*. [WWW] https://www.mapbox.com/api. (01.03.2018).

[5] „Code Samples | Google Maps JavaScript API“, *Google Developers*. [WWW] https://developers.google.com/maps/documentation/javascript/examples/. (01.03.2018).

[6] E. Potter, „Toolset for data-based evaluation and visualisation of road surface condition“, Tallinn University of Technology, 2017.

[7] „Lühiülevaade: Eesti hakkab kõrgusi arvutama Kroonlinna nulli asemel Amsterdami nulli järgi | Maa-amet“. [WWW] https://www.maaamet.ee/et/uudised/luhiulevaade-eesti-hakkab-korgusi-arvutama-kroonlinna-nulli-asemel-amsterdami-nulli-jargi. (05.03.2018).

[8] Pennsylvania State University, „Crown_and_Cross_Slope.pdf“. [WWW] https://www.dirtandgravel.psu.edu/sites/default/files/General%20Resources/Technical%20Bulletins/IB_Crown_and_Cross_Slope.pdf. (20.02.2018).

[9] H. Goebl, „simplify-java: Simplification of a 2D-polyline or a 3D-polyline“, 28-veebr-2018. [WWW] https://github.com/hgoebl/simplify-java. (10.02.2018).

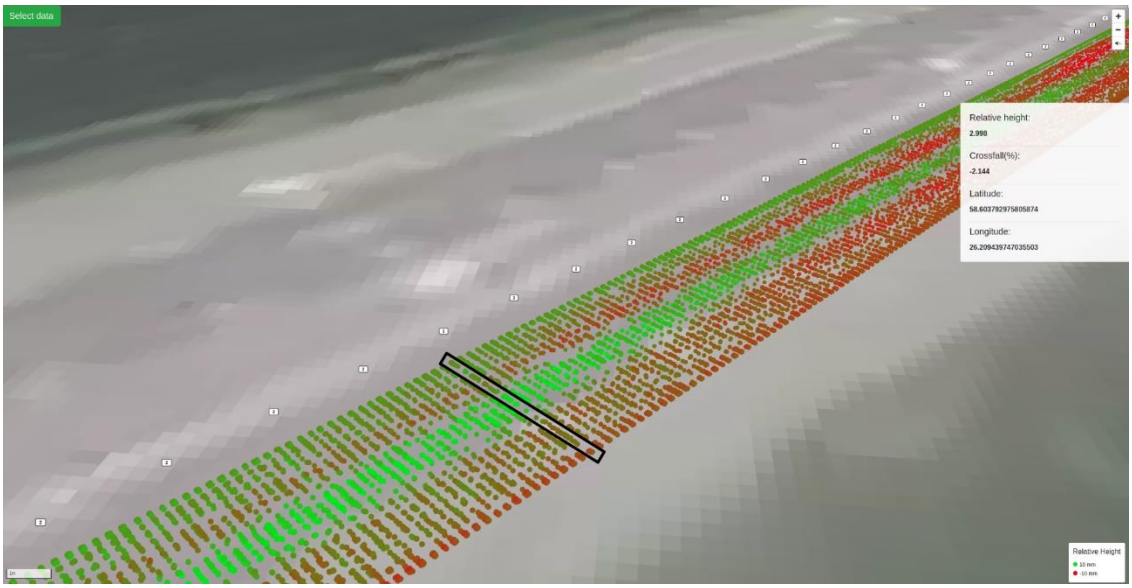
[10] D. Warren, „Line simplification“. [WWW] http://web.pdx.edu/~jduh/courses/geog475f09/Students/W6_Line%20simplification%20presentation.pdf. (11.02.2018).

[11] „Linear interpolation“. [WWW] http://www.eng.fsu.edu/~dommelen/courses/eml3100/aids/intpol/index.html. (14.02.2018).

[12] „Understanding : REST“. [WWW] https://spring.io/understanding/REST. (20.03.2018).

Lisa 1 – Mapbox ja tee profiilid

Järgnevalt on toodud joonis Mapbox'i visualiseerimisest, maksimaalse suumi astmega, kus on võimalik eristada tee profiile üksteisest. Musta kontuuriga on märgistatud üks tee profiil. Kõik tee profiilid sisaldasid algselt 540 punkti, kuid pärast töötlust vähem. Alltoodud pildil, kus kasutati algseadistusi, on peale töötlust alles 20-30 punkti profiili kohta.



Mapbox’i visuaal maksimaalse suumiga

Lisa 2 – Duplikaatide eemaldamine punktipilvest

Enne tarkvarauuendust ja ligipääsu ViaPPS juhendile saamist, toimus punktipilve faili kokkupanek Teede Tehnokeskuses ViaPPS Desktop töölauarakenduses liugur mehhanismi abil. See võimaldas liikuda ühest kohast teise teelõigu ulatuses. Tervet faili korraga ei olnud võimalik kopeerida failide suurte mahtude tõttu. Teelõike sai kopeerida operatiivmällu lõikude kaupa.

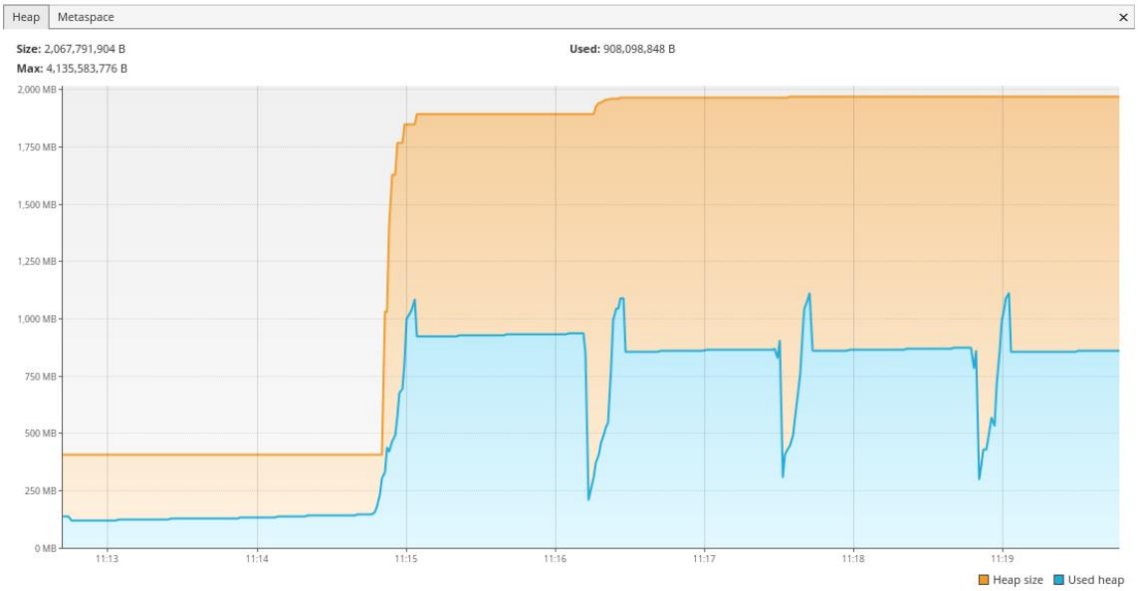
Ühe kilomeetrise punktipilve faili suurus on ligikaudu 220MB. Korraga ei olnud võimalik keskmise arvutiga, millel on 8GB operatiivmälu, kopeerida näiteks kogu 10 kilomeetrist teelõiku. Optimaalne aja ja mälukasutuse kopeerimise suurus jäi 1-2 kilomeetriste lõikude vahele. Kuigi ühekilomeetrise faili suurus on 220MB, siis töölauarakendusest ühekilomeetrise lõigu kopeerimine ja soovitud faili kleepimine võttis oluliselt rohkem mälu, kui oli faili suurus. Näiteks 200 meetrise lõigu kopeerimisel oli maksimaalne mälukasutus ligikaudu 0.5GB. Seega tuli teha mitu kopeerimist. Kuna liugur oli üsna ebatäpne lõigul asukoha valiku suhtes, tekitati punktipilve failid ülekattega, et ei jääks ükski punkt osade kaupa kopeerides vahele.

Et jätta välja duplikaadid ning säilitada järjekord, kasutati selleks andmestruktuuri *LinkedHashSet*'i ning üritati lisada objekte, mille *hashCode* ja *equals* meetod olid üle kirjutatud. Üle kirjutatud meetodid tagasid, et *hash* arvutati objekti x, y, z väärtuse põhjal ning *equals* võrdlus käis kõige kolme parameetriga iga uue objekti lisamisel *LinkedHashSet*'i. *LinkedHashSet* automaatselt kontrollib *hashCode* ja *equals* meetodite abil, kas sama objekti juba ei ole *Set*'i lisatud.

Üsnagi lõputöö lõpus sai Teede Tehnokeskus kätte ViaPPS Desktop tarkvarauuenduse, mis võimaldas faile genereerida ilma igasuguse vahepealse kopeerimiseta. Lisaks selgus juhendist, et tegelikult on olemas funktsionaalsus, mis võimaldab hüpata täpselt kindlasse kohta teelõigul kui vajutada klahvi F3. Selle abil on võimalik vältida ebatäpse liuguri kasutamist täielikult. Seega otsustati, kuna duplikaate enam ei teki, pole mõtet kontrolli rakendusse sisse jätta. Duplikaatide kontrolli eemaldamisega vähendati natukene programmi mälukasutust ja töötlemisele kuluvat aega.

Lisa 3 – Java mälukasutus VisualVM tööriistaga

Alloleval joonisel on näidatud Java *heap* kasutus kus korrati samade toorfailide töötlust neli korda, ilma rakendust taaskäivitamata. Nagu näha, siis peale iga protsessi lõppu on kasutatav *heap* mälukogus peaaegu sama, mis oligi soovitud tulemus. Erandiks on esimese protsessi lõpp, kus *heap* kasutus on natuke kõrgem kui hilisematel töötlustel.



VisualVM *heap* mälukasutus