LAPIN AMK
Lapland University of Applied Sciences

QGIS-lisäosan kehitys Python-ohjelmointikielellä: Kaupunkimallintamisen automatisointi

Issakainen Ilari

Opinnäytetyö
Maanmittaustekniikka
Insinööri (AMK)

2021

LAPIN AMK
Lapland University of Applied Sciences

Opinnäytetyön tiivistelmä

Maanmittaustekniikka
Insinööri (AMK)

Tekijä	Ilari Issakainen	Vuosi	2021
Ohjaaja(t)	Teuvo Heimonen		
Toimeksiantaja			
Työn nimi	QGIS-lisäosan kehitys Python-ohjelmointikielellä: Kaupunkimallintamisen automatisointi		
Sivu- ja liitesivumäärä	52 + 17		

Tämän opinnäytetyön tavoitteena oli ohjelmoida avoimen lähdekoodin paikkatieto-ohjelmaan QGIS:n lisäosa, joka mahdollistaisi helpon ja nopean kaupunkimallintamisen kartalle rajatulta alueelta. Kaupunkimallintaminen on viime vuosina yleistynyt huomattavissa määrin, ja kaupunkimallintamista käytetään yhä useammin kaavoituksen ja kaupunkisuunnittelun perustana. Kaupunkimallintaminen on kuitenkin vielä verrattain työlästä ja vaatii paljon käsin tehtävää työtä. Lähtökohta työn tekemiseen oli tarjota maksuton, nopea ja helppo vaihtoehto kaupunkimallintamiseen.

Työ toteutettiin tutustumalla ensin kaupunkimallinnuksen ohjeisiin, Python-ohjelmointioppaisiin ja QGIS-ohjelman dokumentaatioon. Kerätyn tiedon perusteella lisäosaan suunniteltiin käyttöliittymä ja ohjelmoitavat toiminnallisuudet vaiheittain. Toteutuksessa hyödynnettiin QGIS-ohjelman lisäosaa Plugin Builder, jolla lisäosalle luotiin ohjelmoinnin aloituspiste. Lisäosan käyttöliittymä toteutettiin Qt Designer -ohjelmalla ja lisäosan ohjelmointiin käytettiin IDLE-kehitysympäristöä. Avoimuuden ja maksuttomuuden takaamiseksi teknisessä toteutuksessa mallintamisen lähtöaineistoina käytetään Maanmittauslaitoksen avoimia aineistoja, jotka on peilattu Kapsi Internet käyttäjät ry:n kartat.kapsi.fi-palvelimelle. Kirjallisessa työssä esitellään perusperiaatteet lisäosan suunnittelusta ja toteutuksesta ja työn taustalla vaikuttavasta teoreettisesta perustasta.

Työn tuloksena syntyi QGIS-paikkatieto-ohjelman lisäosa Kaupunkimallintaja ja lisäosan asennus- ja käyttöohje. Ohjelmoitu lisäosa vastaa sille asetettuihin tavoitteisiin ja mahdollistaa kaupunkimallintamisen raskaimpien työvaiheiden automatisoinnin QGIS-ohjelmassa.

Avainsanat	3D-malli, kaupunkimalli, ohjelmointi, QGIS
Muita tietoja	Työhön liittyy ohjelmoitu QGIS-ohjelman lisäosa

LAPIN AMK
Lapland University of Applied Sciences

Abstract of Thesis

Degree Programme in Land Surveying Engineering
Bachelor of Engineering

Author Ilari Issakainen Year 2021
Supervisor Teuvo Heimonen
Commissioned by
Subject of thesis QGIS Plug-in Development with Python Programming Language: Automation of 3D City Modelling
Number of pages 52 + 17

The aim of this thesis was to develop a plug-in for QGIS software that could automate the most time consuming and repetitive tasks of large-scale 3D city modelling. In recent years 3D City modelling has gained popularity amongst urban planning professionals yet the methods for 3D city modelling have required significant amount of manual labour. The objective of developing a QGIS plug-in was to offer an easy-to-use, free, and fast option for 3D city modelling.

The thesis was carried out at first studying the national and international norms of large-scale 3D city modelling, reading several Python programming guides and documentation of QGIS software. Functionalities of the plug-in were designed based on the gathered information. The starting point for programming was made with a plug-in called QGIS Plugin Builder, and the graphical user interface was designed with Qt Designer software. Plug-in was programmed with IDLE development environment. The programmed plug-in uses open data provided by National Land Survey of Finland mirrored into kapsi.kartat.fi server as base material for 3D city modelling.

The result of this thesis study is a QGIS plug-in software called Kaupunkimallintaja and its installation and user manual. Kaupunkimallintaja automates the most repetitive tasks of large-scale 3D City Modeling in QGIS software. The thesis covers the key principles of designing and programming plug-in software and the underlying theoretical basis.

Key words 3D model, 3D city model, programming, QGIS
Special remarks The thesis includes QGIS plug-in software.

SISÄLLYS

KÄYTETYT MERKIT JA LYHENTEET

CityGML	Kaupunkimallinnuksen tiedonsiirtostandardi
DEM	Digital Elevation Model. Korkeusmalli, joka kuvaa maanpinnan muotoa.
DSM	Digital Surface Model. Pintamalli, joka kuvaa ympäristön ylimmän pinnan eli maanpinnan lisäksi muun muassa puiden latvuston ja rakennusten katot.
DTM	Digital Terrain Model. Maastomalli, joka kuvaa maanpinnan muotoa sisältäen maanpinnan taiteviivat ja mahdollisesti ominaistietoa esimerkiksi rinteiden kaltevuuksista tai maanpeitteestä.
GDAL	Geospatial Data Abstraction Library. Komentorivipohjainen avoimen lähdekoodin paikkatieto-ohjelma vektori- ja rasterimuotoisten aineistojen käsittelyyn
LoDn	Level of Detail. Kaupunkimallinnuksen tarkkuustaso, jossa *n* on tarkkuustason numero.
OGC	Open Geospatial Consortium. Kansainvälinen paikkatietoalan avoimia tiedonsiirtostandardeja koordinoiva työryhmä
PDAL	Point Data Abstraction Library. Komentorivipohjainen pistepilvien käsittelyyn tarkoitettu ohjelmisto
QGIS	Avoimen lähdekoodin paikkatieto-ohjelmisto
SIG3D	Special Interest Group SIG3D – kansainvälinen kaupunkimallinnuksen ohjeistusta koordinoiva työryhmä

1 JOHDANTO

Tämän opinnäytetyön tavoitteena on ohjelmoida Python-ohjelmointikielellä QGIS-paikkatieto-ohjelmaan lisäosa, joka mahdollistaa helpon ja nopean kaupunkimallintamisen kartalle rajatulta alueelta Maanmittauslaitoksen avoimien tietoaineistojen pohjalta. Kaupunkimallinnuksella tarkoitetaan kaupunki- tai kuntaympäristön kolmiulotteista mallintamista ja semanttisella kaupunkimallilla sellaista mallia, joka sisältää paikkatiedon lisäksi myös ominaisuustiedot sekä tietojen väliset suhteet. Kaupunkimalleja on alettu luoda niin kaupunkisuunnittelun kuin rakentamisen suunnittelun tarpeesta niiden havainnollisuuden vuoksi. Semanttiset kaupunkimallit myös mahdollistavat uudenlaisten kolmiulotteisten paikkatietoihin pohjautuvien analyysien tekemisen, joista kaupunki- ja rakennussuunnittelu hyötyy. (BuildingSMARTFinland 2016.)

Suomessa kaupunkimallintamisen ohjeistusta on koordinoinut BuildingSMART-Finland-työryhmä, joka on julkaissut Antti Yli-Tainion ja Anssi Savisalon suomennoksen kansainvälisen SIG3D Quality Working Group -työryhmän laatimista kaupunkimallintamisen ohjeista CityGML-formaatissa. Alkuaan SIG3D-työryhmän laatima CityGML-formaatti on kansainvälisen paikkatietoalan avoimia tiedonsiirtostandardeja koordinoivan työryhmän Open Geospatial Consortiumin hyväksymä avoin kaupunkimallinnuksen tiedonsiirtostandardi, jossa on muun muassa kuvattu kaupunkimallinnuksen Level of Detail- eli LoD-tarkkuustasojen sisällöt asteikolla LoD0–4. Ne määrittävät muun muassa sen, kuinka yksityiskohtaisesti rakennusten seinälinjat ja kattogeometriat kuvataan. Vuoden 2020 alussa julkaistiin myös kansallinen julkishallinnon suositus rakennusten ja rakenteiden paikkatietojen mallintamisesta. (SIG3D Quality Working Group 2015, 2; BuildingSMARTFinland 2016; Open Geospatial Consortium 2012, 14–15; Maanmittauslaitos 2020a.)

Tavoitteena on ohjelmoida lisäosa tuottamaan semanttisia kaupunkimalleja LoD1-tarkkuustasolla, jossa rakennukset kuvataan yhtenä laatikkona, jossa ulkopinnat ja lattiat on kuvattu pysty- ja vaakasuorina pintoina kuvion 1 mukaisesti (SIG3D Quality Working Group 2015, 5).

Kuvio 1. Kaupunkimallinnuksen tarkkuustasojen LoD0–3 visuaaliset kuvaukset (SIG3D Quality Working Group, 18)

Työ rajataan LoD1-tasolle, koska Maanmittauslaitos on aloittanut uuden tiheämmän laserkeilausaineiston keräämisen myötä myös kolmiulotteisten rakennustietojen keräämisen. Niiden tarkkuustaso tulee olemaan parempi kuin LoD1. (Marttila 2019, 12–13; Maanmittauslaitos 2021a.) Näin vältytään tekemästä päällekkäistä työtä Maanmittauslaitoksen kanssa. Lisäosan ohjelmointi LoD1-tarkkuustasolle antaa kuitenkin oletettavasti hyvää esitietoa siitä, miten tarkemmat LoD-tarkkuustasot tulisi ohjelmassa myöhemmin toteuttaa. Ohjelma pyritään rakentamaan siten, että tulevaisuudessa ohjelma olisi helposti muokattavissa hakemaan myös Maanmittauslaitoksen julkaisemat kolmiulotteiset rakennustiedot. Yksi tapa kaupunkimallintamiseen on luoda kolmiulotteinen paikkatietoympäristö, jossa maanpinta ja maanpinnan tekstuurit kuvataan korkeusmallilla tai maastomallilla, jonka päälle ortoilmakuva on laskostettu kuvion 2 mukaisesti.

Kuvio 2. Ortoilmakuva venytetty korkeusmallin päälle

Rakennukset ja rakenteet sijoitetaan alareunastaan kiinni korkeusmalliin. LoD1-tarkkuustason mallintamisessa rakennuksen tai rakenteen korkeus voidaan antaa esimerkiksi kohteen ominaisuustietona, joka muutetaan CityGML-formaatin mukaiseksi korkeustiedoksi. Maanmittauslaitoksen avoimet paikkatietoaineistot

sisältävät muun muassa ortoilmakuvia, korkeusmalleja, laserkeilausaineistoja ja maastotietokannan aineistoja (Maanmittauslaitos 2021b). Rakennusten ja rakenteiden sijaintitiedot voidaan hankkia Maastotietokannan aineistoista. Rakennusten korkeudet voidaan laskea puolestaan laserkeilausaineiston tai laserkeilausaineistosta johdetun DSM-pintamallin perusteella ja siirtää rakennusten ominaisuustietoihin. Siten Maanmittauslaitoksen avoimet aineistot soveltuvat hyvin kaupunkimallintamisen pohjaksi ainakin LoD1-tarkkuustasolla.

Tällä hetkellä Maanmittauslaitoksen avoimien tietoaineistojen latauspalvelusta ladataan yksittäisiä tiedostoja ja aineistolajeja karttalehdittäin, jolloin jo pelkästään lähtöaineiston kerääminen on hidasta ja työlästä. Lähtöaineiston keräämisen työläys korostuu erityisesti sellaisilla alueilla, jotka sijaitsevat useamman karttalehden alueella. Ohjelmoitavalla lisäosalla pyritään vähentämään merkittävästi aineiston keräämiseen ja mallintamiseen käytettävää työaikaa ja siten tekemään mallintamisesta helpompaa ja kustannustehokkaampaa.

Avoimen lähdekoodin QGIS-paikkatieto-ohjelman kehittyminen viime vuosina on saanut useat kaupungit kiinnostumaan sen tarjoamista mahdollisuuksista ja QGIS haastaa jo monet kaupalliset vastineet. (Jokela 2017, 26–27.) Vuoden 2021 keväällä QGIS-ohjelmasta julkaistiin versio 3.18. Siihen on integroitu pistepilvien käsittelyyn tarkoitettu komentoriviohjelma PDAL, joka mahdollistaa pistepilviaineistojen käsittelyn. (QGIS 2021a.) Saman version myötä QGIS tukee myös suoraan pistepilviaineistojen kolmiulotteista visualisointia (QGIS 2021a), josta on hyötyä varsinkin kaupunkimallien laaduntarkastuksessa. QGIS on myös muokattavissa tiettyihin tarkoituksiin Python-ohjelmointikielen avulla, jolla ohjelman toiminnot voidaan automatisoida (QGIS 2021b). Näiden ominaisuuksiensa johdosta QGIS-ohjelma valikoitui työn pohjaksi.

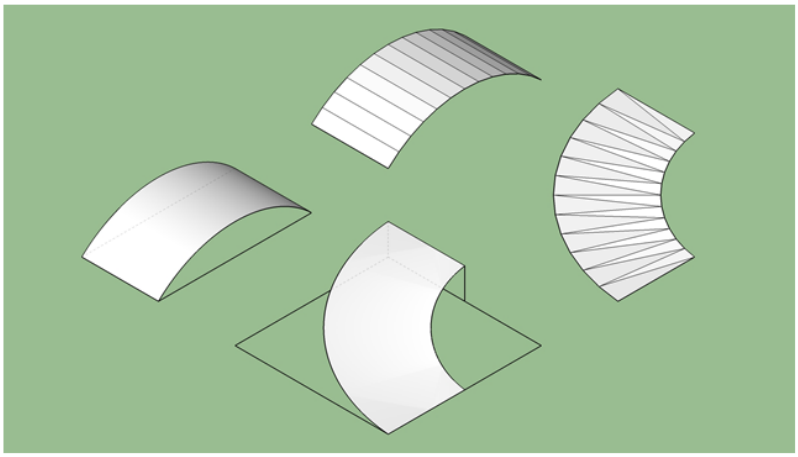
Tässä opinnäytetyössä kerrotaan perustiedot QGIS-lisäosan suunnittelusta, toteutuksesta ja julkaisemisesta. Samalla opinnäytetyö toimii lisäosan teknisenä dokumentaationa. Lisäosan kehitys rajataan QGIS:n Windows-versiolle 3.18 ja siitä ylöspäin. Lisäosan automaattisesti mallinnettavat kohteet rajataan rakennuksiin, pylväisiin johtoineen, mastoihin ja piippuihin LoD1-tarkkuustasolla.

2 TYÖN LÄHTÖKOHDAT

2.1 Mallinnusohjeet ja suositukset

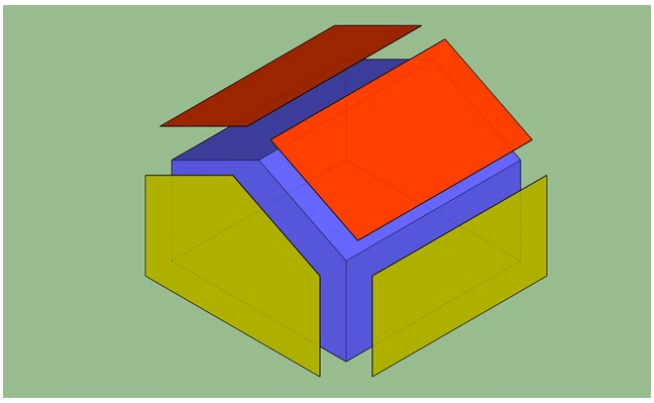
Kaupunkimallintamista koskevissa ohjeissa ja suosituksissa kuvataan, miten kolmiulotteinen malli tulee rakentaa kullakin yksityiskohtaisuuden tasolla. CityGML-tiedonsiirtostandardissa määritellään yksityiskohtaisuuden tasot LoD0-4 ja niihin liittyvät mallinnusohjeet. Suomessa ohjeistusta on tarkennettu julkishallinnon suosituksella 210: Paikkatietojen mallintaminen: rakennukset ja rakennelmat. Siinä kuvataan, mitä kohteita mallinnetaan ja miten kullakin yksityiskohtaisuuden tasolla. Tässä opinnäytetyössä tavoitteena oli ohjelmoida kaupunkimallintamista automatisoiva lisäosa QGIS-ohjelmaan LoD1-tasolla. Tämän vuoksi työssä paneudutaan tarkemmin mallinnusohjeista vain LoD1-tason määrittelyihin. Määrittelyistä keskitytään työn rajauksen mukaisesti vain yleisiin periaatteisiin, rakennuksiin, mastoihin, piippuihin, pylväisiin ja rakennelmiin.

Paikkatietokohde voidaan esittää eri geometrisina muotoina, kuten pisteinä, murtoviivoina, komposiittipintoina, kappaleina tai komposiittikappaleina. Komposiittipinnalla tarkoitetaan pintaa, joka koostuu useasta pienemmästä pinnasta. Esimerkiksi maastossa kaarevat ja kiertyvät pinnat mallinnetaan kolmioista muodostuvina komposiittipintoina Kuvion 3 mukaisesti. Rajaava kaari oikaistaan murtoviivaksi siten, että murtoviivan oikaisusta johtuva sijaintivirhe tasossa on korkeintaan 0,15 metriä. (JHS 210: Liite 5, 3–5.)



Kuvio 3. Kaareva ja kiertyvä kohde mallinnettu komposiittipintoina (JHS 210: Liite 5, 4)

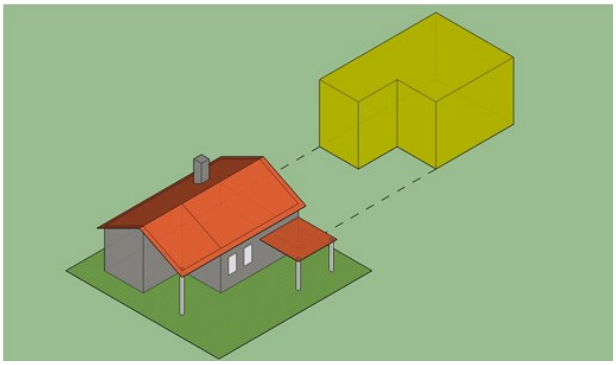
Paikkatietokohde voi olla myös niin sanottu koostekohde, joka koostuu osakohteista. Esimerkiksi rakennus on paikkatietokohde, joka voi koostua eri osakohteista esimerkiksi eri käyttötarkoituksissa olevista tiloista. Osakohteet kuuluvat rakennus-paikkatietokohteeseen, mutta osakohteilla voi olla myös omia toisistaan poikkeavia ominaisuustietoja. Paikkatietokohteella voi olla myös useita eri geometrioita. Esimerkiksi rakennukset mallinnetaan sulkeutuvina kappaleina, jotka muodostuvat useasta sulkeutuvasta pinnasta Kuviossa 4 esitetyllä tavalla. Pinnat puolestaan koostuvat sulkeutuvista murtoviivoista. Semanttisten kaupunkimallien tapauksessa pinnoille tallennetaan ominaisuustiedoksi kyseessä oleva pinta, kuten katto, seinä tai lattia. Myös kohteen avoimet sivut mallinnetaan sulkeutuvina pintoina, joille annetaan ominaisuustiedoiksi *closure_surface*. (JHS 210, 2–3; JHS210: Liite 5, 3–5.)



Kuvio 4. Sulkeutuvan kappaleen muodostuminen sulkeutuvien murtoviivojen muodostamien pintojen perusteella (JHS210: Liite 5, 5)

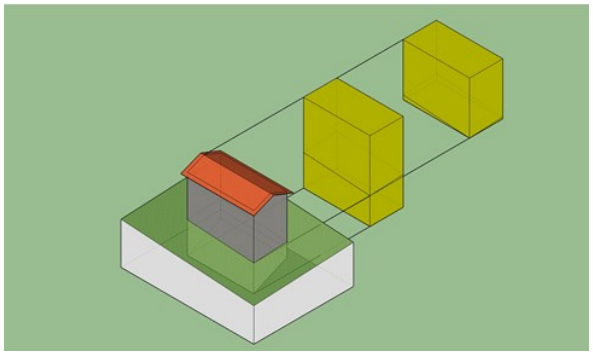
2.1.1 Rakennukset

LoD1-yksityiskohtaisuudentasolla rakennukset mallinnetaan laatikkomaisesti pystysuorina ja vaakasuorinapintoina. Sivupinnat kuvaavat rakennuksen ulointa ulkoseinäpintaa. LoD1-geometriaan ei sisälly rakennuksen varustekohteita kuten katoksia (Kuvio 5). (JHS210: Liite 5, 10, 12.)



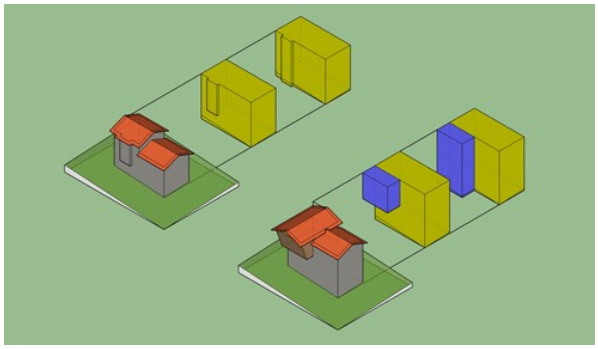
Kuvio 5. Esimerkki rakennuksen mallintamisesta LoD1-yksityiskohtaisuustasolla (JHS210: Liite 5, 12)

Rakennusten alapinnat mallinnetaan vaakatasoisina pintoina ensisijaisesti rakennuksen alimmalle korkeustasolle tai vaihtoehtoisesti alapinta mallinnetaan maanpinnan leikkausviivan alimpaan korkeuteen (Kuvio 6). Maanpinnan leikkausviiva voidaan tallentaa myös ominaisuustiedoiksi nimellä *Terrain Intersection Line*. (JHS210: Liite 5, 10.)



Kuvio 6. Rakennuksen alapinnan vaihtoehtoiset mallintamistavat (JHS210: Liite 5, 13)

Jos alapinta sijaitsee maanpinnan yläpuolella, kuten erkkereissä, alapinta voidaan mallintaa joko todelliseen alimpaan korkeuteen tai maanpinnan leikkausviivan alimpaan korkeuteen (Kuvio 7). Yläpinnat mallinnetaan puolestaan kohteen korkeimpaan kohtaan. (JHS210: Liite 5, 10.)



Kuvio 7. Rakennuksen erkkereiden vaihtoehtoiset mallintamistavat (JHS210: Liite 5, 14)

2.1.2 Mastot, piiput ja pylväät

Mastot, piiput ja pylväät mallinnetaan kohteen ulkopintojen mukaisesti pystysuorina pintoina. Korkeus määräytyy kohteen korkeimman kohdan mukaan. Alareuna voidaan mallintaa kahdella vaihtoehtoisella tavalla, joko kohteen todelliseen alimpaan korkeuteen tai kohteen ja maapinnan leikkausviivan alimpaan korkeuteen. (JHS210: Liite 5, 39–43.)

2.1.3 Rakennelma

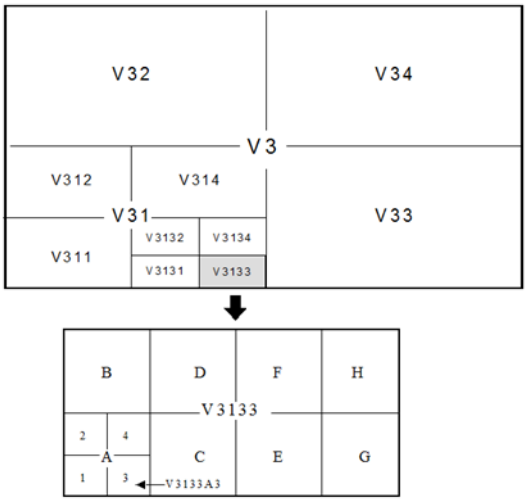
Rakennelmista kolmiulotteisesti mallinnetaan vain mainosrakennelmat, ilmaradat, nosturit, hyppyrimäet, huvipuistolaitteet, muistomerkit, patsaat, taideteokset, suihkulähteet, joukkoliikenteen pysäkkikatokset, katokset, kodat, laavut, kammit, keittokatokset ja maakellarit. Aallonmurtajat mallinnetaan, jos niiden leveys on vähintään viisi metriä. Rautatien asemalaiturit ja rauniot mallinnetaan vain, jos ne muodostavat selkeän sulkeutuvan kappaleen. Yläreuna määräytyy kohteen todellisen korkeimman kohdan mukaisesti. Alareuna voidaan tallentaa joko kohteen todelliseen alimpaan korkeuteen tai kohteen ja maanpinnan leikkausviivan alimpaan korkeuteen. (JHS210: Liite 5, 56–58.)

2.2 Maanmittauslaitoksen avoimet aineistot

2.2.1 Yleistä

Maanmittauslaitos on julkaissut vuodesta 2012 osan tuottamistaan paikkatietoaineistoista kaikille avoimina aineistoina. Aineistot julkaistiin ensin vuosina 2012–2015 Maanmittauslaitoksen avoimien aineistojen lisenssi versio 1.0:n alaisina ja vuodesta 2015 alkaen aineistot on julkaistu CC Nimeä 4.0 -lisenssin alla. Yhteistä molemmille lisensseille on se, että käyttäjä saa jakaa, levittää ja muokata aineistoja vapaasti mainitsemalla kuitenkin alkuperäisen aineiston tuottajan nimen omissa sovelluksissaan tai tuotteissaan. Avoimien aineistojen lisenssiversio 1.0:n alaisiin aineistoihin sovelletaan hankinta-ajankohdan lisenssiä. Maanmittauslaitoksen avoimet aineistot on ladattavissa osoitteesta https://tiedostopalvelu.maanmittauslaitos.fi/tp/kartta. Kaikki työssä tarvittavat aineistot, karttalehtijako, korkeusmallit, ortoilmakuvat, laserkeilausaineisto ja maastotietokanta, kuuluvat avoimiin aineistoihin. Aineistot ovat saatavilla ETRS-TM35FIN-tasokoordinaattijärjestelmässä. (Maanmittauslaitos 2021d.)

Maanmittauslaitos on julkaissut aineistot JHS 197 mukaisessa karttalehtijaossa, jossa 1:200 000 mittakaavassa olevalle karttalehdelle on annettu yksiselitteinen kirjaimesta ja numerosta koostuva tunnus. Kirjain kuvaa karttalehden sijaintia pohjois-eteläsuunnassa ja numero länsi-itäsuunnassa. Kirjaimet aloitetaan etelästä kirjaimesta K kasvaen aakkosjärjestyksessä pohjoiseen. Poikkeuksena on O, jota ei käytetä. Numerot aloitetaan lännestä numerosta 2 kasvaen itään. Suuremmassa mittakaavassa oleville aineistoille saadaan yksiselitteinen karttalehtitunnus jakamalla edellinen karttalehti neljään tai kahdeksaan osaan kuvion 8 mukaisesti. Karttalehtien puolikkaat voidaan vielä erottaa toisistaan lisäämällä karttalehtitunnuksen eteen kirjaimet L tai R, joista L tarkoittaa vasemmanpuoleista ja R oikeanpuoleista. Eri mittakaavojen karttalehtijako on saatavilla Shapefile- ja MapInfo-formaateissa. (JHS 197 1.1 / 9.12.2016; Maanmittauslaitos 2021c.)



Kuvio 8. Karttalehtijaon perusperiaate (JHS197 1.1 / 9.12.2016)

2.2.2 Korkeusmalli ja muut korkeutta kuvaavat mallit

Tässä yhteydessä käytetään Maanmittauslaitoksen kansallisen maastotietokannan korkeusmallien laatukäsikirjan mukaisia määritelmiä, sillä lisäosa käyttää Maanmittauslaitoksen tuottamia aineistoja. Korkeusmallilla DEM tarkoitetaan maanpinnan korkeutta kuvaavaa digitaalista mallia, joka on joko tasavälinen pistehila tai epäsäännöllinen kolmioverkko TIN (Triangulated Irregular Network), joka koostuu x-, y- ja z-koordinaateiltaan tunnettujen pisteiden joukosta ja pisteiden välisistä pinnoista. Maanpinnalla tarkoitetaan paljasta maanpintaa, johon eivät kuulu kasvillisuus eikä maanpinnalla olevat siirtolohkareet tai muut vastaavat irtonaiset kohteet. Maanpintaan ei myöskään kuulu vedenalainen maanpinta, vaan vesialueet on esitetty aineiston keräyksen ajankohdan mukaisena keskivedenpintana. (Sirkiä ym. 2017, 6–9.)

DTM-maastomallilla tarkoitetaan puolestaan maanpinnan korkeutta kuvaavaa digitaalista mallia, joka sisältää myös ominaisuustietoa maanpinnasta, kuten kaltevuuksia ja viettosuuntia. DSM:llä tarkoitetaan ympäristön ylimmän pinnan korkeutta kuvaavaa mallia, joka sisältää kasvillisuutta, puuston latvustoja ja myös ihmisen tekemien rakenteiden korkeuksia, kuten rakennusten kattoja. Maanmittauslaitoksen avoimissa aineistoissa on tarjolla laserkeilausaineistosta johdetut korkeusmallit rasterimuotoisena kahden ja kymmenen metrin pistehiloina GeoTIFF- ja Ascii Grid -formaateissa N2000-korkeusjärjestelmän mukaisina korkeuksina. (Maanmittauslaitos 2021b; Sirkiä ym. 2017, 6–9.)

2.2.3 Ortoilmakuva

Ortoilmakuvalla tarkoitetaan ilmakuvaa, joka on oikaistu alkuperäisestä perspektiivikuvasta karttaprojektioon esimerkiksi korkeus-, maasto- tai pintamallin avulla. Ortoilmakuva vastaa geometrialtaan karttaa. Ortoilmakuvasta on korjattu alkuperäisessä kuvassa esiintyvä mittakaavan vaihtelu, joka aiheutuu kohteiden sijaitessa eri etäisyydellä kuvanottopaikasta. Ortoilmakuva koostuu tyypillisesti useasta ilmakuvasta, josta käytetään myös termiä *ortokuvamosaiikki*. (Haggrén 2002 & Honkavaara 2005, 1–5.)

Ortoilmakuvan oikaisuun käytetyt korkeus- tai maastomallit sisältävät vain maanpinnan korkeustietoja. Ne aiheuttavat siten lopulliseen ortoilmakuvaan näkyvää virhettä siten, että maanpintaa ylempänä olevat rakenteet kuvautuvat alkuperäisen kuvan mukaisessa perspektiivissä. Kyseinen virhe voidaan poistaa käyttämällä orto-oikaisuun korkeusmallin tai maastomallin sijasta pintamallia, joka sisältää ylimmän pinnan korkeudet. Tällöin puhutaan tosiortokuvasta. (Haggrén 2002 & Honkavaara 2005, 1–5, 10.) Maanmittauslaitoksen tuottamat ortoilmakuvat on saatavilla avoimien aineistojen tiedostopalvelusta JPEG2000- eli .jp2-formaatissa (Maanmittauslaitos 2021b).

2.2.4 Maastotietokanta

Maastotietokanta on koko Suomen maastoa, rakenteita, liikenneverkkoa ja maankäyttöä kuvaava vektorimuotoinen aineisto (Maanmittauslaitos 2020b). Aineisto on saatavilla muun muassa Geopackage-, MapInfo- ja Shapefile-formaateissa. Shapefile-formaatissa aineisto on jaoteltu ja nimetty tiedostoihin karttalehdittäin ja teemoittain kunkin geometriatyypin mukaan erikseen taulukossa 1 esitetyllä tavalla (Maanmittauslaitos 2019).

Taulukko 1. Maastotietokannan tiedostojen nimeäminen Shapefile-formaatissa

Lyhenne (x)	Geometria	t_karttalehtitunnus_x
v	viivat	
s	symbolipisteet	
t	tekstit	
p	polygonit (monikulmiot)	
Lyhenne (t)	Teema	
l	Liikenneverkot	
j	Johtoyhteydet	
m	Maasto/1	
n	Maasto/2	
r	Rakennukset	
k	Korkeussuhteet	
s	Suojelukohteet	
e	Erityiskäyttöalueet	
h	Hallinnollinen jaotus	
u	Taajaan rakennettu alue	

Taulukon 1 esimerkissä t on aineiston teeman lyhenne, keskellä oleva kirjainsarja karttalehtitunnus ja x aineiston geometrian lyhenne. Esimerkiksi tiedosto r_L4434R_p.shp sisältää maastotietokannan rakennukset monikulmioina L4434R-karttalehden alueelta. Maaston kohdetyypit ilmaistaan puolestaan tiedoston attribuutteina (Maanmittauslaitos 2019).

2.2.5 Laserkeilausaineisto

Laserkeilauksella tarkoitetaan valon kulkuaikaan tai vaihe-eroon perustuvaa kolmiulotteista mittausmenetelmää, jossa lopputuotteeksi saadaan pistepilvi, useiden x-, y-, z-koordinaateiltaan tunnettujen pisteiden joukko. Valon kulkuaikaan perustuvat keilaimet mittaavat keilaimen lähettämän laserpulssin paluuajan kohteesta, jonka perusteella voidaan laskea pisteen etäisyys keilaimesta. Vaihe-eroon perustuva laserkeilain eroaa pulssilaserkeilaimesta siten, että keilain lähettää kohteeseen jatkuvaa signaalia useammalla eri kantoaallonpituudella, josta lähetetyn ja palanneen signaalin välinen vaihe-ero mitataan etäisyyden laskemiseksi. Kolmiulotteisen pistepilven muodostamiseksi etäisyyden lisäksi täytyy tietää pisteiden ulkoinen orientointi eli etäisyyden mittauksen suunta. Kolmijaloilta

operoitavien laserkeilaimien suunnat saadaan samalla periaatteella kuin takymetrimittauksessa koordinaateiltaan tunnettujen liitospisteiden avulla, jotka sijaitsevat keilausalueella. Laserkeilain voidaan sijoittaa myös liikkuvaan alustaan kuten lentokoneeseen, helikopteriin tai ajoneuvoon. Tällöin koordinaattien ratkaisemiseksi tarvitaan etäisyyden mittauksen lisäksi tietää mittauspaikka ja keilaimen asento. Mittauspaikka voidaan ratkaista esimerkiksi mobiilikeilainjärjestelmään liitetyn GNSS-paikantimen avulla ja asento inertiamittausjärjestelmän perusteella. (Laurila 2012, 269–273; Kukko 2005, 7.)

Maanmittauslaitoksen tuottama aineisto on ilmalaserkeilausaineisto, jota on saatavilla kahdella eri pistetiheydellä. Vuosina 2008–2019 kerätty aineisto on saatavilla avoimena aineistona pistetiheydellä 0,5 pistettä / m². Vuodesta 2020 kerätty aineisto on saatavilla pistetiheydellä 5 pistettä / m², aineiston käyttöön tarvitsee kuitenkin hankkia maksullinen käyttölupa. Vuodesta 2020 kerätty aineisto on kuitenkin saatavilla avoimena aineistona harvennettuna pistetiheyteen 0,5 pistettä / m². (Maanmittauslaitos 2020c.)

Laserkeilausaineisto on saatavilla LAZ-formaatissa luokiteltuna aineistona. Luokittelulla tarkoitetaan pisteille annettua luokkaa, joka kertoo, millä pinnalla tai kohteella piste sijaitsee. Maanmittauslaitoksen aineistossa on yhdistelty LAS 1.2-määrittelyn mukaista luokittelua ja omaa luokittelua. Taulukossa 2 on esitetty Maanmittauslaitoksen laserkeilausaineiston ja LAS1.2 -määrittelyn mukaiset luokat. LAS 1.2 -määrittelyn mukaisiin luokkiin on lisätty tunnus LAS 1.2. (Maanmittauslaitos 2020c.)

Taulukko 2. LAS 1.2 ja Maanmittauslaitoksen pistepilviluokat (Maanmittauslaitos 2020c; ASPRAS 2008, 8)

Luokka	Selite
0	Luotu, mutta luokittelematon (Created, never classified) LAS 1.2
1	Luokittelematon (Unclassified) LAS 1.2
2	Maanpinta (Ground) LAS 1.2
3	Aluskasvillisuus (Low Vegetation) LAS 1.2
4	Keskikorkea kasvillisuus (Medium Vegetation) LAS 1.2
5	Korkea kasvillisuus (High Vegetation) LAS 1.2
6	Rakennus (Building) LAS 1.2
7	Matalat virhepisteet (Low points) LAS 1.2
8	Mallin avainpisteet (Model Key-Points) LAS 1.2
9	Vesi (Water) LAS 1.2
10	Varattu ASPRS määrittelylle (Reserved for ASPRS definition)
11	Varattu ASPRS määrittelylle (Reserved for ASPRS definition)
12	Kahdella lentolinjalla sijaitsevat päällekkäiset pisteet (Overlap points) LAS 1.2
13	Varattu ASPRS määrittelylle (Reserved for ASPRS definition)
14	Varattu ASPRS määrittelylle (Reserved for ASPRS definition)
15	Pilvet tai lentävät kohteet (Air points)
16	Yksittäiset pisteet ilmassa ja maanpinnan alla (Isolated)
17	Keilainhäiriöistä johtuvat pisteet (Fault points)

Maanmittauslaitos tarjoaa aineistoaan automaattisesti ja stereomalliluokiteltuna. Stereomalliluokiteltu aineisto on automaattiluokituksen jälkeen tarkastettua ja luokitukseltaan korjattua aineistoa. (Maanmittauslaitos 2020c.)

2.3 QGIS: avoimen lähdekoodin paikkatieto-ohjelmisto

QGIS on avoimen lähdekoodin graafinen paikkatieto-ohjelmisto, joka on julkaistu GNU General Public License (GPL) -lisenssin alla. Lisenssi mahdollistaa lähdekoodin vapaan tutkimisen ja muokkaamisen, minkä johdosta ohjelma on helposti muokattavissa omiin käyttötarkoituksiin lisäosien avulla. Ohjelma tarjoaa myös valmiin lisäosakirjaston, josta käyttäjän on helppo ladata ja asentaa tarvitsemansa lisäosat. Lisäosat voidaan ohjelmoida joko C++- tai Python-ohjelmointikielellä. Ohjelmaan on myös mahdollista ohjelmoida pienempiä ohjelmia, niin sanottuja skriptejä, jotka esimerkiksi automatisoivat ohjelman toimintoja. Myös yksittäisiä ohjelman komentoja voidaan suorittaa komentorivipohjaisesti Python-konsolin avulla. (QGIS 2021b, 1–3; QGIS 2021c, 3, 1349–1352.)

QGIS on saatavilla Unix-, Windows- ja macOs-järjestelmille (QGIS 2021c, 3). QGIS-ohjelma tarjoaa monipuoliset geoprosessointityökalut paikkatietoanalyysien tekemiseen. Osa toiminnoista on QGIS-ohjelman omaa tuotantoa, mutta ohjelmaan on lisätty myös muiden avoimen lähdekoodin ohjelmien, kuten GDALn, SAGAn ja GrassGISn, tarjoamia prosessointialgoritmeja (QGIS 2021d). Versiosta 3.18 lähtien QGIS ohjelmaan on integroitu komentorivipohjainen pistepilviaineistojen käsittelyyn tarkoitettu PDAL-ohjelma (QGIS 2021a). PDAL-integraation myötä QGIS tukee myös pistepilviaineistojen kolmiulotteista visualisointia (QGIS 2021a).

2.4 Python-ohjelmointikieli

Ohjelmointi voidaan ajatella tietokoneelle annettavina käskyinä, jotka suorittavat jonkin ennalta määrätyn, ohjelmoidun tehtävän. Tietokoneen prosessoreilla on jokaisella oma niin sanottu konekieli, jota se ymmärtää. Konekieli koostuu binäärisistä biteistä eli ykkösistä ja nollista, ja siten se on ihmisille yleensä vaikealukuista. Ohjelmoinnin helpottamiseksi on kehitetty erilaisia helpompilukuisia kieliä, jotka voidaan kääntää tai tulkata konekielen ymmärtämään muotoon. (Zelle 2004, 6–7.)

Ohjelmointikielet jaotellaan pääasiassa käännettäviin ja tulkattaviin kieliin. Käännettävällä kielellä tarkoitetaan kieltä, joka käännetään konekieliseen muotoon ennen ohjelman suorittamista vain kerran. Kerran käännettyä ohjelmaa ei tarvitse enää kääntää, vaan se on valmis suoritettavaksi heti. Tulkattava kieli puolestaan muuttaa ohjelmointikielen rivit konekieliseen muotoon rivi kerrallaan ohjelman suorittamisen aikana. Tulkattavien kielien suorittaminen on siten hitaampaa kuin käännettävän kielen. Toisaalta tulkattavan kielen etuja on sen kehitystyön aikainen nopeampi testaaminen, koska käännettävien kielien kääntäminen on yleensä hidasta. (Zelle 2004, 7–9.)

Python on tulkattava ohjelmointikieli (Zelle 2004, 9), jota on mahdollista laajentaa C- ja C++-kielillä (Python 2021a). Vaihtoehtoisesti Pythonia voidaan käyttää niin sanottuna skriptikielenä C- ja C++-kielillä ohjelmoiduissa ohjelmissa, kuten QGIS:ssa. (QGIS 2021b, 1–3.) Graafiset käyttöliittymät voidaan toteuttaa esimerkiksi C++:lla ohjelmoidulla Qt kirjastolla, jolla myös QGIS:n graafinen käyttöliittymä on toteutettu (QGIS 2021b, 6). Kesäkuussa 2021 Python-ohjelmointikielestä julkaistiin versio 3.9.6 (Python 2021b).

3 LISÄOSAN SUUNNITTELU

3.1 Ohjelmistoversion valinta

Lisäosan kehitys rajataan vain Windows-käyttöjärjestelmälle QGIS:n versioon 3.18 ja siitä ylöspäin.

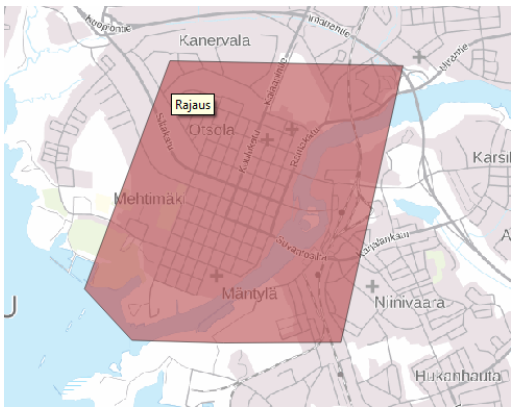
3.2 Mallintamisen lähtöaineiston hankinta

Mallintamisen lähtöaineistona käytetään Maanmittauslaitoksen avoimia aineistoja. Maanmittauslaitos tarjoaa avoimet aineistot hakemistokäyttöliittymänä mutta maksullisena palveluna (Maanmittauslaitos 2021b). Lisäosan suunnittelussa pyrittiin maksuttomuuteen, joten Maanmittauslaitoksen toteuttama palvelu ei tullut kuitenkaan kyseeseen. Ratkaisuksi löytyi Kapsi internet-käyttäjät ry:n ylläpitämä tiedostopalvelin kapsi.kartat.fi. Se peilaa Maanmittauslaitoksen avoimet aineistot omalle ilmaiselle, kaikille avoimelle palvelimelleen kerran vuorokaudessa. (Kapsi Internet-käyttäjät ry 2015.) Tiedostot on saatavilla hakemistokäyttöliittymänä, jossa aineistot on luokiteltu kansioihin aineistolajeittain, aineiston keruuvuosittain, karttalehtitunnuksittain ja tiedostoformaattien mukaan.

Kartat.kapsi.fi-palvelu on toteutettu harrastajavoimin, eikä se siten takaa palvelun olevan aina käytettävissä (Kapsi Internet-käyttäjät ry 2015). Tämä on riski lisäosan toteutuksen kannalta, mutta ohjelman ilmaisuus on tärkeämpi kriteeri.

3.3 Lisäosan tuottaman lopputuloksen määritys

Lisäosan tekemänä lopputuotteena on käyttäjän kartalle rajaaman alueen sisältä ladatut aineistot, joita 3D-kaupunkimallin tekemiseksi tarvitaan. Saman aineistolajin eri karttalehdiltä haetut aineistot tulisi olla yhdistettynä samaan tiedostoon ja rajattuna käyttäjän antaman aluerajauksen mukaisiksi. Kuviossa 9 esitetään esimerkki aluerajauksesta. Kapsin palvelimelta haettavat aineistolajit ovat karttalehtijako, maastotietokanta, korkeusmalli, ortoilmakuva ja laserkeilausaineisto.



Kuvio 9. Esimerkki luotavan kaupunkimallin aluerajauksesta. Sisältää Maanmittauslaitoksen avoimia aineistoja 9/2021.

Maastotietokannan osalta mallinnettavat kohteet rajataan rakennuksiin, mastoihin, piippuihin ja pylväisiin. Näiden kohteiden osalta korkeudet lasketaan laserkeilausaineistosta johdetun pintamallin perusteella. Laserkeilausaineisto jätetään lopulliseen aineistoon myös aineiston visualisoimiseksi ja oikeellisuuden tarkastamiseksi. Laserkeilausaineisto tulisi myös värjätä aineiston visualisoinnin parantamiseksi. Maanmittauslaitoksen avoimien aineistojen lisenssiehtojen täyttämiseksi lisäosan täytyy tehdä myös lokitiedosto, johon kerätään tiedot aineistolajeittain mitä karttalehtiä ja tiedostoja mallin tekemiseksi on tarvittu.

Lopputuloksena syntyy mallintamisen vaihe LoD1 tarkkuustasolla QGIS-ohjemaan, josta käyttäjä pystyy tekemään aineistoon tarkastuksia ja myös korjaamaan automaatiosta mahdollisesti syntyneitä virheitä sekä lisäämään malliin omia aineistojaan. Käyttäjän harkittavaksi jää myös lopullisen julkaisualustan valinta ja kääntäminen Internet-julkaistavaan muotoon. Kuviossa 10 on esitetty luodun mallin kolmiulotteinen kuvaus.

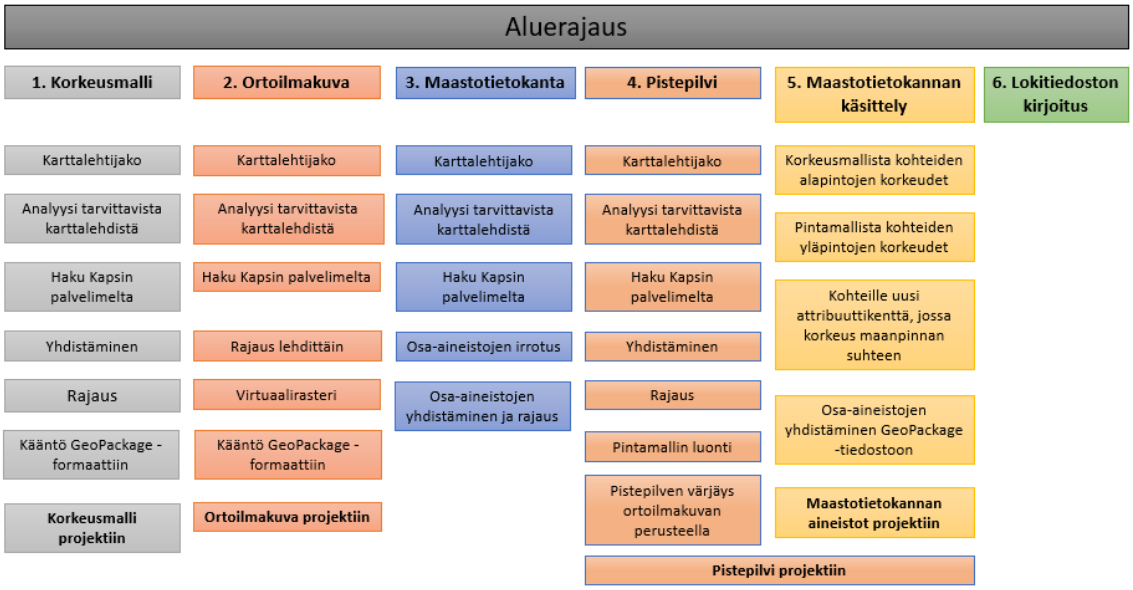
Kuvio 10. Lisäosalla tuotetun kaupunkimallin visuaalinen kuvaus. Sisältää Maanmittauslaitoksen avoimia aineistoja 9/2021.

3.4 Tiedostoformaatit

Maastotietokannassa, ortoilmakuvissa ja korkeusmalleissa käytetään GeoPackage-tietokantaformaattia. Maastotietokannan osa-aineistot yhdistetään samaan GeoPackage-tietokantaan. Pistepilvien osalta käytetään LAZ-formaattia.

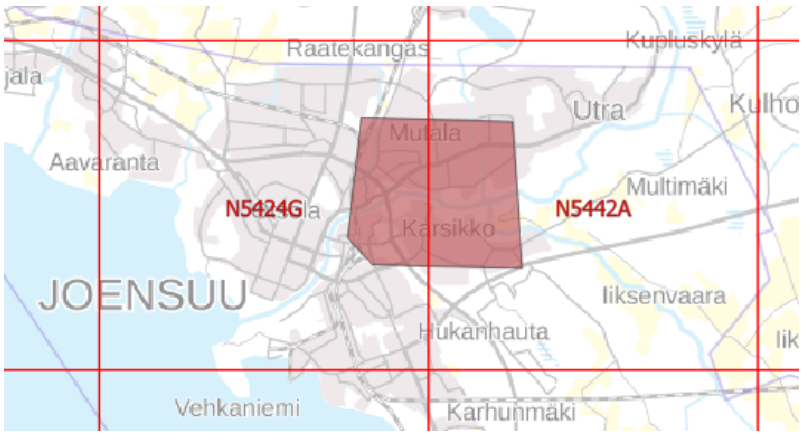
3.5 Ohjelmoitavat toiminnot

Ohjelmoitavat toiminnot jäsennettiin pienemmiksi kokonaisuuksiksi miettimällä, mitä kaikkia vaiheita kaupunkimallin tekemiseksi tarvitaan QGIS-ohjelmalla. Ohjelmoitavat vaiheet jaetaan kuuteen eri päävaiheeseen ja jokaisen vaiheen alle sisältyviin alivaiheisiin (Kuvio 11).



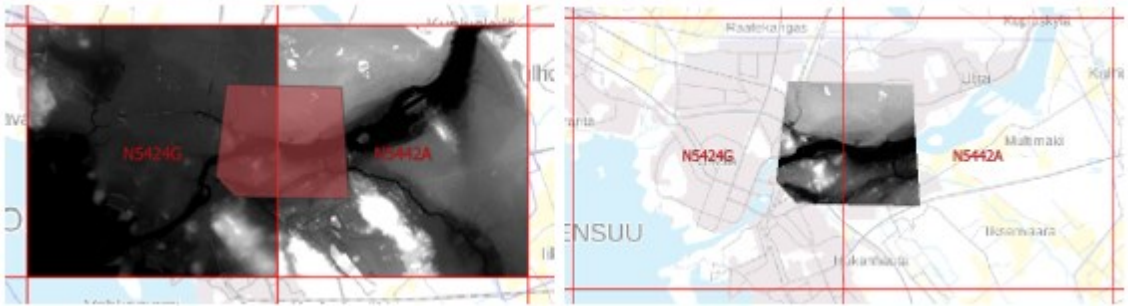
Kuvio 11. Lisäosaan ohjelmoitavien toimintojen vaiheet

Kuvion 11 päävaiheiden kohtiin 1–4 sisältyy analyysi tarvittavista karttalehdistä kussakin aineistolajissa. Ohjelman tulisi selvittää aineistolajeittain, minkä karttalehtien alueella annettu aluerajaus sijaitsee. Kuviossa 12 esitetään utm10 karttalehtijako ja esimerkki aluerajaukseta Maanmittauslaitoksen taustakartta-aineiston kanssa.



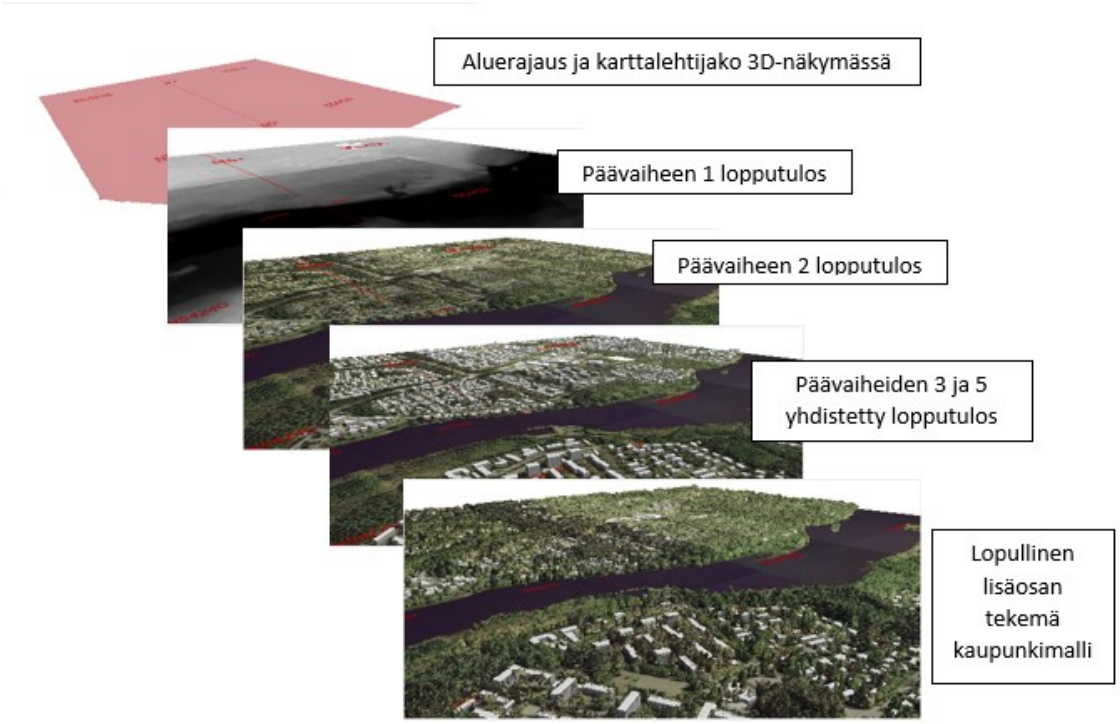
Kuvio 12. Kaupunkimallin aluerajaus ja utm10-karttalehtijako. Sisältää Maanmittauslaitoksen avoimia aineistoja 9/2021.

Kapsin palvelimella utm10-karttalehtijakoa käytetään esimerkiksi korkeusmallien nimeämiseen. Kuvion 12 tapauksessa korkeusmalleista haettaisiin karttalehdet N5424G ja N5442A. Päävaiheisiin 1–4 sisältyy myös aineistojen yhdistäminen ja rajaus annetun aluerajauksen mukaiseksi. Kuviossa 13 on kuvattu haettujen aineistojen yhdistäminen ja rajaus.



Kuvio 13. Kapsin palvelimelta haettujen korkeusmallien yhdistäminen ja rajaus. Sisältää Maanmittauslaitoksen avoimia aineistoja 9/2021.

Kuviossa 13 vasemmalla on päävaiheessa 1 Kapsin palvelimelta haetut korkeusmallit N5424G ja N5442A, joita ei ole vielä rajattu aluerajauksen mukaisesti eikä yhdistetty toisiinsa. Kuviossa 13 oikealla on korkeusmallien yhdistämisen ja rajaamisen tulos. Kuviossa 14 puolestaan esitetään mallintamisen kulku aluerajauksesta lopulliseksi malliksi päävaiheiden lopputulosten kautta.

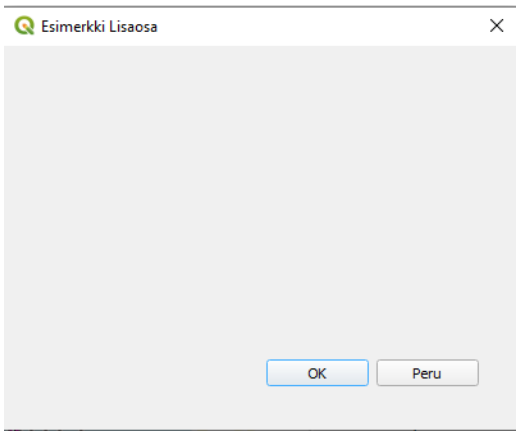


Kuvio 14. Mallintamisen kulku päävaiheiden lopputulosten kautta

4 LISÄOSAN TOTEUTUS

4.1 Aloituspisteen luonti

Lisäosien rakentamiseksi GeoApt LLC on tehnyt QGIS Plugin Builder -nimisen lisäosan, jolla omalle lisäosalleen voi luoda aloituspisteen. Kaupunkimallintajan toteutuksessa käytetään kyseistä lisäosaa ja nojataan Ujaval Gandhin tekemään verkkotutoriaaliin kyseisen lisäohjelman käyttämisestä. Kyseinen verkkotutoriaali on julkaistu osoitteessa https://www.qgistutorials.com/en/docs/3/building_a_python_plugin.html. Lisäosa rakentaa valmiiksi uuden lisäosan valikkojen sijoittelut ja tyhjän graafisen käyttöliittymän (Kuvio 15). Tässä vaiheessa uudelle lisäosalle annettiin myös nimi Kaupunkimallintaja.



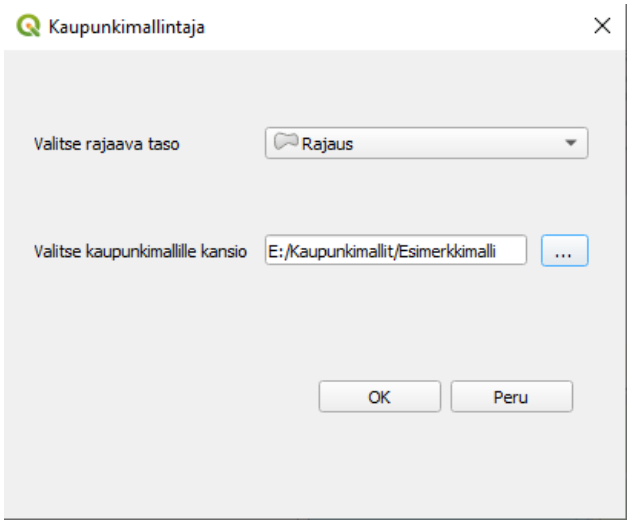
Kuvio 15. QGIS Plugin Builderilla luotu esimerkki lähtötilanteesta

Plugin Builder tuottaa kansion, jossa ovat kaikki lisäosan tiedostot. Muutoksia tehtiin vain lisäosan nimeä kantavaan Kaupunki_mallintaja.py-tiedostoon, Kaupunki_mallintaja_dialog_base.ui-tiedostoon ja metadata.txt-tiedostoon. Ui-tiedostoon luodaan käyttöliittymä Qt designer -ohjelmalla, py-tiedostoon varsinaiset ohjelman suorittamat toiminnot ja metadata.txt-tiedostoon luodaan lisäosan kuvaus, joka näkyy käyttäjälle lisäosien asennusvalikossa.

Toteutuksen dokumentoinnissa keskitytään vain ohjelmoitujen toimintojen kuvauksiin ja perusteluihin sekä algoritmien asetusten vaikutukseen. Toiminnot esitetään lisäosaan suunniteltujen päävaiheiden mukaan jaoteltuna. Koko Kaupunki_mallintaja.py-tiedoston lähdekoodi on tämän työn liitteessä 1. Toiminnot ohjelmoitiin käyttäen Pythonin IDLE-kehitysympäristöä.

4.2 Käyttöliittymä

Käyttöliittymä toteutettiin Qt designer -ohjelmalla Plug-in Builder -lisäosan tuottaman lähtötilanteen pohjalta (Kuvio 16). Käyttöliittymään toteutettiin alasvetovalikko, johon täytetään kaikki QGIS-projektissa olevat monikulmiotasot, joista valittu taso toimii rajaavana alueena kaupunkimallille. Käyttöliittymään toteutettiin lisäksi valikko, josta kaupunkimallille valitaan kansio.



Kuvio 16. Valmiin ohjelman käyttöliittymä

4.3 Automatisoinnin algoritmit

4.3.1 Vaihe 1: Korkeusmallit

Ensimmäisessä vaiheessa ohjelma noutaa Kapsin palvelimelta korkeusmallit karttalehdittäin. Tarvittavien karttalehtien noutamiseksi ohjelma suorittaa ensin QGIS:n algoritmin *joinattributesbylocation* rajaavan tiedoston ja utm10-karttalehtijaon välillä, jossa karttalehtijaon attribuutit yhdistetään rajaavaan tasoon geometrisen sijainnin perusteella. Utm10-karttalehtijakoa ohjelma ei nouda Kapsin palvelimelta, koska Kapsin palvelimella kyseinen tiedosto oli korruptoitunut vielä elokuussa 2021. Utm10-karttalehtijako on ladattu avoimien aineistojen tiedostopalvelusta ja toimitettu käyttäjälle lisäosan mukana. Lisäosa ratkaisee utm10.shp-tiedoston tiedostopolun käyttäjän koneelta automaattisesti. Algoritmin lopputuloksena syntyy uusi väliaikainen tiedosto, jossa on vain rajaavan tason

alueella sijaitsevat karttalehtijakojen monikulmiot. Karttalehtitunnus on tallennettu monikulmioiden attribuutiksi *LEHTITUNNU*. QGIS-algoritmin *joinattributes bylocation* parametrejä voi muuttaa liitteessä 1 olevan lähdekoodin riveiltä 263–269 (Kuvio 17).

```
262    #Karttalehdetattribuutteihin
263    lehtitunnukset = processing.run("native:joinattributesbylocation", {'INPUT':selectedLayer,
264                                                                        'JOIN':karttalehtijako,
265                                                                        'PREDICATE':[1,2,4,5],
266                                                                        'JOIN_FIELDS':['LEHTITUNNU'],
267                                                                        'METHOD':0,'DISCARD_NONMATCHING':False,
268                                                                        'PREFIX':'',
269                                                                        'OUTPUT':'TEMPORARY_OUTPUT'})
```

Kuvio 17. QGIS *joinattributesbylocation* algoritmin parametrit

Predicate-kohdassa määritetään numeroarvoilla kriteerit karttalehtien valintaan:

- Arvolla 1 valitaan karttalehtijaon monikulmio, joka sisältää rajaavan alueen.
- Arvolla 2 valitaan karttalehtijaon monikulmio, joka on yhtenevä rajaavan alueen kanssa.
- Arvolla 4 valitaan karttalehtijaon monikulmio, jonka päällä rajaava monikulmio sijaitsee edes osittain.
- Arvolla 5 valitaan karttalehtijaon monikulmio, joka on kokonaan rajaavan alueen sisällä. (QGIS 2021c, 1013.)

Seuraavassa vaiheessa ohjelma käy läpi saadun tulostiedon *LEHTITUNNU* attribuuttikentän arvot, jotka syötetään ladattavien tiedostojen Internet-osoitteisiin. Esimerkiksi karttalehden N5412A kahden metrin hilakorkeusmallin osoite Kapsin palvelimella on http://kartat.kapsi.fi/files/korkeusmalli/hila_2m/etrs-tm35fin-n2000/N5/N54/N5412A.tif.

Lataaminen käyttäjän koneelle tehdään QGIS:n algoritmilla *filedownloader*. Tässä vaiheessa onnistuneiden latausten Internet-osoitteet siirretään myös listatyyppiseen muuttujaan, josta osoitteet voidaan myöhemmin hakea raportin kirjoittamista varten. Samalla myös koneelle tallennettujen tiedostojen polut kerätään toiseen listatyyppiseen muuttujaan, josta tiedostopolut syötetään aineistojen yhdistämisen algoritmiin seuraavassa vaiheessa.

Noudetut korkeusmallit yhdistetään yhdeksi tiedostoksi QGIS-ohjelmaan integroidun komentoriviohjelma GDAL:n algoritmilla *merge*. Yhdistämisen algoritmin parametrit ovat liitteen 1 riveillä 311–320 (Kuvio 18).

```
311 #Yhdistäminen
312 yhdistettyOutput = processing.run("gdal:merge", {'INPUT':osoitteet,
313                                   'PCT':False,
314                                   'SEPARATE':False,
315                                   'NODATA_INPUT':None,
316                                   'NODATA_OUTPUT':None,
317                                   'OPTIONS':'',
318                                   'EXTRA':'',
319                                   'DATA_TYPE':5,
320                                   'OUTPUT':'TEMPORARY_OUTPUT'})
```

Kuvio 18. Korkeusmallien yhdistämisen algoritmien parametrit ohjelman lähdekoodissa

- *PCT*-kohdassa määrätään, otetaanko tulostiedoston pseudoväritaulu ensimmäisestä tasosta. Arvot annetaan boolean arvoilla eli totena (True) tai epätotena (False).
- *SEPARATE*-kohdassa määrätään, laitetaanko jokainen syötekuva omaan rasterikanavaansa. Arvo annetaan myös boolean arvona.
- *NODATA_INPUT*-kohdassa määrätään syötekuvan pikseliarvo, joka käsitellään tyhjänä arvona.
- *NODATA_OUTPUT*-kohtaan voidaan antaa tulostiedostolle tyhjä arvo.
- *DATA_TYPE*-kohdassa määrätään tulostiedoston datatyyppi, joka oletusarvolla 5 määräytyy tyypiksi Float32.
- *OPTIONS*- ja *EXTRA*-kohtiin voidaan antaa kirjoittamisen lisämääreitä esimerkiksi tulostiedoston pakkausmuodoista.

Yhdistämisen lopputulos kirjoitetaan väliaikaiseen tiedostoon, joka luetaan projektiin mutta ei graafiseen liittymään. Tarkemmin GDAL:n merge-algoritmin lisämääreistä ja parametreistä on QGIS:n ja GDAL:n dokumentaatiossa. (QGIS 2021c, 1223–1224; GDAL 2021a.)

Yhdistämisen jälkeen ohjelma suorittaa väliaikaiseen yhdistettyyn korkeusmallitiedostoon GDAL:n algoritmin *cliprasterbymasklayer*, jolla korkeusmalli rajataan

ohjelman alussa annetun tason mukaiseksi. Algoritmin parametrit ovat liitteen 1 riveillä 320–340 (Kuvio 19).

```
325 clipped = processing.run("gdal:cliprasterbymasklayer", {'INPUT':yhdistettyOutput['OUTPUT'],
326                                                        'MASK':selectedLayer,
327                                                        'SOURCE_CRS':None,
328                                                        'TARGET_CRS':None,
329                                                        'NODATA':0,
330                                                        'ALPHA_BAND':False,
331                                                        'CROP_TO_CUTLINE':True,
332                                                        'KEEP_RESOLUTION':False,
333                                                        'SET_RESOLUTION':False,
334                                                        'X_RESOLUTION':None,
335                                                        'Y_RESOLUTION':None,
336                                                        'MULTITHREADING':False,
337                                                        'OPTIONS':'',
338                                                        'DATA_TYPE':0,
339                                                        'EXTRA':'',
340                                                        'OUTPUT':'TEMPORARY_OUTPUT'})
```

Kuvio 19. Cliprasterbymasklayer-algoritmin parametrit ohjelman lähdekoodissa

Rajaamisen kannalta olennaiset parametrit ovat *CROP_TO_CUTLINE* ja *NODATA*.

- *CROP_TO_CUTLINE*-kohdan True-arvolla korkeusmalli leikataan monikulmion rajausviivan mukaiseksi.
- *NODATA*-kohdassa määrätään arvo, joka tulkitaan ohjelmassa tyhjäksi eli toisin sanoen läpinäkyväksi. (QGIS 2021c, 1213–1214.)

Korkeusmallien käsittelyn viimeisessä vaiheessa saatu rajattu tiedosto käännetään Geopackage-tiedostoformaattiin GDAL:n algoritmilla *translate*. Tässä vaiheessa saatu tulostiedosto tallennetaan jo käyttäjän koneelle ja ladataan projektiin myös graafisesti. *Translate*-algoritmin parametrit alkavat lähdekoodissa (Liite 1) riviltä 342 (Kuvio 20).

```
342 clippedGPKG = processing.run("gdal:translate", {'INPUT':clipped['OUTPUT'],
343                                                 'TARGET_CRS':None,
344                                                 'NODATA':0,
345                                                 'COPY_SUBDATASETS':False,
346                                                 'OPTIONS':'',
347                                                 'EXTRA':'',
348                                                 'DATA_TYPE':0,
349                                                 'OUTPUT':outputdir + '/Korkeusmalli.gpkg'})
```

Kuvio 20. GDAL:n translate algoritmin parametrit lähdekoodissa

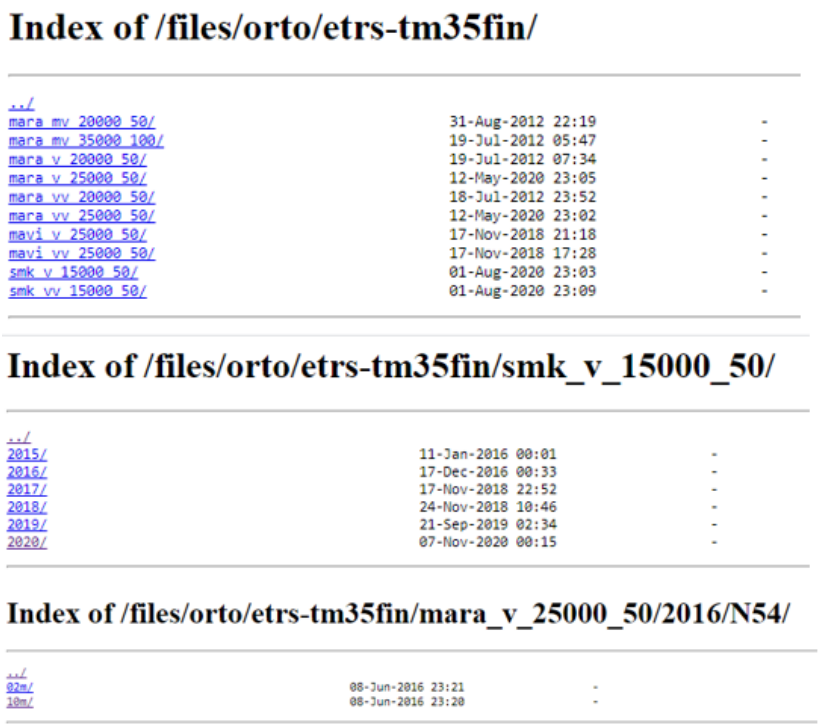
Tarvittaessa aineisto voidaan muuntaa toiseen koordinaattijärjestelmään *TARGET_CRS*-kohdassa viittaamalla haluttuun koordinaattijärjestelmään sen EPSG-koodilla. Arvolla *None* ohjelma tunnistaa syötetason koordinaattijärjestelmän ja

käyttää tätä tietoa myös tulostiedostossa. *DATA_TYPE*-kohdassa arvolla 0 käytetään myös syötetason tietotyyppiä. *OUTPUT*-kohdassa määrätään tulostiedoston sijainti ja nimi. Sijainti määräytyy käyttäjän ohjelman alussa antaman syötteen ja ohjelman käsittelemän monikulmion järjestysnumeron perusteella. Tiedoston formaatti määrätään tiedostonimen päätteen perusteella. Gpkg-päätteellä kirjoitetaan Geopackage-tiedosto. (QGIS 2021c, 1209–1211.)

4.3.2 Vaihe 2: Ortoilmakuvat

Ohjelman vaiheessa 2 noudetaan ortoilmakuvat Kapsin palvelimelta sekä yhdistetään ja rajataan yhdeksi tiedostoksi. Vaiheen suorittamiseksi täytyisi suorittaa analyysi tarvittavista karttalehdistä. Kuitenkin vaiheessa 1 kyseinen analyysi on jo suoritettu, sillä Maanmittauslaitos on julkaissut sekä korkeusmallit että ortoilmakuvat samassa karttalehtijaossa eli utm10:n mukaisina. Ohjelma poimii siten tarvittavat karttalehdet edellisessä vaiheessa tehdyn analyysin perusteella.

Kapsin palvelimella ortoilmakuvat on jaoteltu aineiston keruuvuosittain, aineiston keräysprojekteittain ja pikselikoon mukaan useisiin kansioihin (Kuvio 21). Siten ohjelman on testattava läpi kaikki kansiot ja vuodet oikean tiedoston löytämiseksi.



Kuvio 21. Ortoilmakuvien kansiorakenne Kapsin palvelimella vuonna 2021

Kansioiden läpikäynti on toteutettu lähdekoodissa for silmukan sisällä olevalla try-except-rakenteella ja Pythonin Urllib-moduulin request-funktiolla (Liite 1, rivit 393–431). For-silmukalla käydään läpi annetut vuodet ja try-except-rakenteella annetut osoitteet. Urllib-moduulin request-funktiolla voidaan testata, palautuuko kyselystä tieto tiedoston olemassaolosta (Python 2021c). Jos tieto ei palaudu, ohjelma siirtyy try-except-rakenteen mukaan seuraavaan testattavaan url-osoitteeseen. Jos request-funktio palauttaa tiedon olemassaolosta, siirretään kyseinen url-osoite noudettaviin karttalehtiin.

Ohjelma noutaa lopuksi karttalehdet saaduista url-osoitteista QGIS:n algoritmilla *filedownloader* (Liite 1, rivit 435–436). Koska ortoilmakuvien tiedostokoot ovat paljon suurempia kuin korkeusmallien, suoritetaan rajaaminen karttalehdittäin ennen tiedostojen yhdistämistä toisiinsa. Rajaaminen tehdään GDAL:n:n algoritmilla *cliprasterbymasklayer*, jonka parametrit ovat liitteen 1 riveillä 449–464 (Kuvio 22). Algoritmi on sama, jolla myös korkeusmallit leikattiin aikaisemmassa vaiheessa, mutta parametreissa on muutamia poikkeuksia.

```
449 processing.run("gdal:cliprasterbymasklayer", {'INPUT':uri,
450                                              'MASK':selectedLayer,
451                                              'SOURCE_CRS':None,
452                                              'TARGET_CRS':None,
453                                              'NODATA':0,
454                                              'ALPHA_BAND':False,
455                                              'CROP_TO_CUTLINE':True,
456                                              'KEEP_RESOLUTION':False,
457                                              'SET_RESOLUTION':False,
458                                              'X_RESOLUTION':None,
459                                              'Y_RESOLUTION':None,
460                                              'MULTITHREADING':True,
461                                              'OPTIONS':'COMPRESS=DEFLATE|PREDICTOR=2|ZLEVEL=9|BIGTIFF=YES',
462                                              'DATA_TYPE':0,
463                                              'EXTRA':'',
464                                              'OUTPUT':outputdir + '/ortocrop_' + lehti + '.tif' })
```

Kuvio 22. Ortoilmakuvien leikkaamisen parametrit lähdekoodissa

Suurimpana erona korkeusmallien leikkaamiseen ovat *OPTIONS*-parametrille annetut lisämääreet:

- *COMPRESS*-asetuksella määrätään TIFF-tiedoston kompressio eli pakkaustyyppi, tässä tapauksessa *DEFLATE.*
- *PREDICTOR*-asetuksen arvolla 2 määrätään kuvan pakkaajan tallettamaan vaakasuunnassa vierekkäisten solujen erotuksen arvot varsinaisten solujen RGB arvojen sijasta.
- *ZLEVEL*-asetuksella määrätään pakkauksen suuruus silloin kun käytetään *DEFLATE*-pakkaustyyppiä. Pienemmällä arvolla pakataan

vähemmän ja suuremmalla arvolla enemmän. Arvot voidaan antaa välillä 1–9.

- BIGTIFF-asetuksella määrätään, kirjoitetaanko lopputulos tavallisena vai BIGTIFF-muotoisena TIFF-kuvana. Tavallisen TIFF-kuvan suurin mahdollinen tiedostokoko on 4GB, eli tätä suuremmat tiedostokoot on kirjoitettava BIGTIFF-muotoisen. Arvo *YES* pakottaa aina lopputuloksen BIGTIFF-muotoiseksi, joka on tässä tapauksessa varmin valinta, koska ohjelma ei tarkasta etukäteen ladattujen karttalehtien tiedostokokoja. (GDAL 2021b.)

Karttalehtien leikkaamisen jälkeen lehdet yhdistetään toisiinsa luomalla virtuaalirasteri GDAL:n algoritmilla *buildvirtualraster*. Algoritmi *buildvirtualraster* ei varsinaisesti luo syötetasoista yhtenäistä tiedostoa vaan tiilimosaiikin, jossa yksi karttalehti on yksi tiili. Tällä tavoin laajat alueet toimivat nopeammin, sillä ohjelman tarvitsee hakea muistiin vain se tiili, jota tarkastellaan koko rasteritiedoston sijasta. Algoritmin *buildvirtualraster* parametrit ovat liitteen 1 riveillä 473–481 (Kuvio 23).

```
473 vrt=processing.run("gdal:buildvirtualraster", {'INPUT':cropatut_osoitteet,
474                                               'RESOLUTION':0,
475                                               'SEPARATE':False,
476                                               'PROJ_DIFFERENCE':False,
477                                               'ADD_ALPHA':True,
478                                               'ASSIGN_CRS':None,
479                                               'RESAMPLING':0,
480                                               'SRC_NODATA':'',
481                                               'EXTRA':'',
482                                               'OUTPUT':'TEMPORARY_OUTPUT'})
483
```

Kuvio 23. Buildvirtualraster-algoritmin parametrit lähdekoodissa

Olennaisin parametri on *RESOLUTION*, jolle voidaan antaa kolme eri arvoa:

- Arvolla 0 mosaiikin resoluutio on kaikkien syötetiedostojen resoluutioiden keskiarvo.
- Arvolla 1 mosaiikin resoluutio määritetään syötetiedostojen korkeimman resoluution mukaan.
- Arvolla 2 mosaiikin resoluutio määritetään syötetiedostojen pienimmän resoluution mukaan. (QGIS 2021c, 1225–1226.)

Virtuaalirasteri kirjoitetaan lopulta GeoPackage-formaattiin GDAL:n algoritmilla *translate* (Liite 1, rivit 495–501), jota käytettiin korkeusmalleissa vaiheessa 1. GeoPackage-formaattiin tallentuu virtuaalirasterin tiilet ja tiilien indeksointi. Samassa yhteydessä kirjoitetaan myös väliaikainen TIFF-tiedosto (Liite 1, rivit 485–491), jolla myöhemmin värjätään ladatut pistepilvet PDAL-komentoriviohjelman avulla päävaiheessa 4.

4.3.3 Vaihe 3: Maastotietokanta

Lisäosan kolmannessa vaiheessa projektiin lisätään maastotietokannan aineistot. Tarvittavien karttalehtien hakemiseksi suoritetaan ensin analyysi algoritmilla *joinattributesbylocation* samoin kuin korkeusmallien tapauksessa vaiheessa 1. Algoritmin parametrit ovat liitteen 1 riveillä 538–544 (Kuvio 24). Ainoana erona analyysi tehdään rajaustiedoston ja utm25-karttalehtijaon kanssa, joka on toimitettu myös lisäosan mukana käyttäjälle. Lisäosa ratkaisee kyseessä olevan karttalehtijaon sijainnin käyttäjän koneella automaattisesti.

```
535 MTK_karttalehtijako = resolve('utm25LR.shp')
536
537 #Karttalehdetattribuutteihin
538 MTK_lehtitunnukset = processing.run("native:joinattributesbylocation", {'INPUT':selectedLayer,
539                                                                         'JOIN':MTK_karttalehtijako,
540                                                                         'PREDICATE':[1,2,4,5],
541                                                                         'JOIN_FIELDS':['LEHTITUNNU'],
542                                                                         'METHOD':0,'DISCARD_NONMATCHING':False,
543                                                                         'PREFIX':'',
544                                                                         'OUTPUT':'TEMPORARY_OUTPUT'})
```

Kuvio 24. Algoritmin *joinattributesbylocation* parametrit lisäosan lähdekoodissa

Maastotietokannan osalta käyttäjän koneelle ei ladata koko ZIP-tiedostoa. Projektiin ladataan vain tarvittavat tasot suoraan Kapsin palvelimella sijaitsevasta Shapezip-tiedostosta GDAL:n *virtual file system* -työkalun avulla. GDAL:n virtual file system -työkalut tarjoavat suoran pääsyn http- tai ftp-palvelimella sijaitsevaan tiedostoon lisäämällä URL-osoitteen eteen merkkijononon *vsicurl/* (liite 1, rivi 566). ZIP-tiedoston sisällä olevaan tasoon viitataan puolestaan lisäämällä ZIP-tiedoston URL-osoitteen perään merkkirivi |*layername=* ja yhtäsuuruusmerkin jälkeen tarvittavan tason nimi (Liite 1, rivit 569–600). Maanmittauslaitoksen maastotietokannan tasojen nimeämisen periaate on esitetty aiemmin taulukossa 1. Tässä vaiheessa ohjelma yhdistää ja rajaa eri karttalehtien tiedostot aineistola-

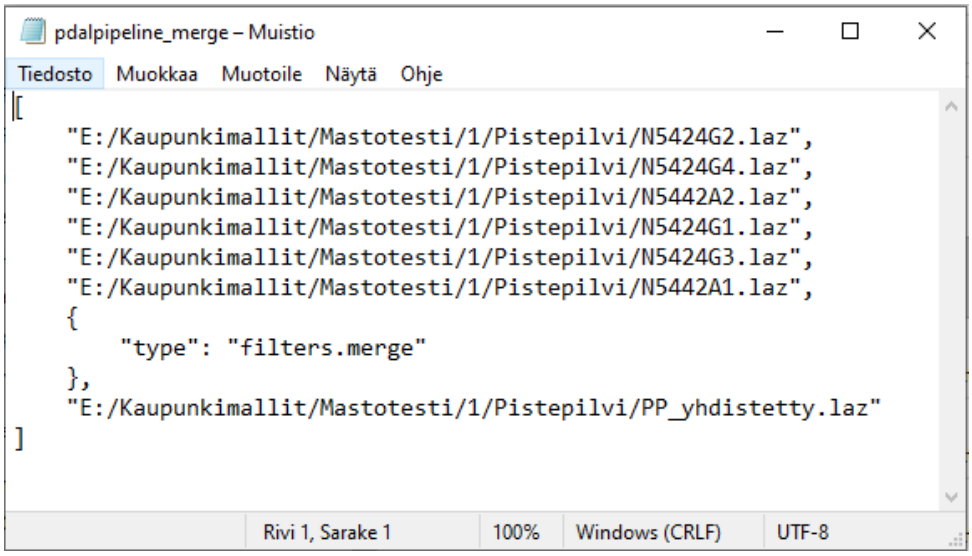
jeittain yhdeksi väliaikaiseksi tiedostoksi QGIS:n omilla algoritmeilla *mergevectorlayer* ja *clip* (Liite 1, rivit 602–695). Maastotietokannan aineistojen muuttamista 3D-aineistoksi käsitellään myöhemmin luvussa 4.2.5.

4.3.4 Vaihe 4: Pistepilvet

Ohjelman suorittamassa päävaiheessa 4 haetaan tarvittavat pistepilvet Kapsin palvelimelta sekä yhdistetään, rajataan ja suodatetaan tietyllä luokituksella olevia pisteitä pois. Lisäksi pistepilvet värjätään aikaisemmin luodun ortoilmakuvan perusteella sekä muodostetaan pistepilvestä pintamalli, jota seuraavassa vaiheessa käytetään maastotietokannan rakennusten korkeuksien määrittämiseen.

Ohjelman alussa suoritetaan analyysi *joinattributesbylocation* utm5-karttalehtijaon ja annetun rajaustiedoston välillä tarvittavien karttalehtien saamiseksi (Liite 1, rivit 717–722). Utm5-karttalehtijako sisältyy lisäosan asennuskansioon, ja lisäosa ratkaisee automaattisesti karttalehtijaon sijainnin käyttäjän koneella. Tarvittavat karttalehdet ladataan käyttäjän koneelle QGIS:n algoritmilla *filedownloader*. Kapsin palvelimella stereomalliluokiteltujen pistepilvien tiedostot on jaettu kansioihin aineiston keruuvuosittain. Ohjelma etsii oikean kansion Urllib-moduulin ja *try-except*-rakenteen avulla vastaavasti kuin ortoilmakuvien tapauksessa vaiheessa 2. Kohdan lähdekoodi on liitteen 1 riveillä 728–761.

Pistepilviaineistojen käsittely poikkeaa hieman muista vaiheista, sillä QGIS-ohjelma ei vielä tarjoa suoraan pistepilviaineistojen käsittelyyn tarkoitettuja algoritmeja. QGIS-ohjelman mukana on kuitenkin asennettu PDAL-niminen komentorivipohjainen pistepilviaineistojen käsittelyyn tarkoitettu ohjelma, jota voidaan käyttää QGIS:n mukana asennetun OsGeoShell-käyttöliittymän kautta. PDAL-ohjelmalla suoritettavat toiminnot voidaan antaa joko komentoriville annettuina käskyinä tai niin sanottuina pipeline-tiedostona (PDAL 2021a; PDAL 2021b). Pipeline-tiedostoon annetaan suoritettavat komennot lineaarisessa järjestyksessä JSON-muotoiltuna tekstitiedostona, jossa käskyt ja arvot annetaan niin sanottuina nimi-arvo pareina (PDAL 2021b). Esimerkki pipeline-tiedostosta on kuviossa 25.



Kuvio 25. Esimerkki PDAL:n pipeline-tiedostosta

Kuviossa 25 tiedostoon on ensin annettu yhdistettävät tiedostot. Jäljempänä *type*-kohdassa määrätään, mitä PDAL:n algoritmia käytetään. Tiedoston toisiksi viimeisellä rivillä annetaan tulostiedoston osoite.

Pistepilvien käsittely on toteutettu siten, että lisäosa luo kustakin käsittelyvaiheesta, esimerkiksi karttalehtien yhdistämisestä, oman pipeline-tiedoston. Pipeline-tiedoston luomisen jälkeen OsGeoShell käynnistyy aliohjelmana Pythonin subprocess-moduulin avulla ja suorittaa PDAL-ohjelman parametreinään aiemmin luodut pipeline-tiedoston mukaiset toiminnot.

Lisäosa yhdistää pistepilvikarttalehdet PDAL:n algoritmilla merge, jonka jälkeen yhdistetty pistepilvi rajataan rajaustiedoston mukaiseksi algoritmilla crop. Pistepilvien yhdistämiseen liittyvä lähdekoodi on liitteen 1 riveillä 775–803 ja pistepilvien rajaamiseen liittyvä lähdekoodi riveillä 805–855. Pistepilviä rajaava monikulmiotaso annetaan pipeline-tiedostoon Well-Known-Text-muotoiltuna koordinaattiluettelona. Koska Maanmittauslaitoksen avoimet aineistot on toimitettu ETRS-TM35FIN-koordinaattijärjestelmässä, ohjelma tekee ensin kopion rajaavasta tasosta ja muuntaa kopioidun tason myös ETRS-TM35FIN-koordinaattijärjestelmään (Liite 1, rivit 810–814). Rajaavan tason monikulmion Well-Known-Text-muotoiltu koordinaattiluettelo haetaan pipeline-tiedostoon tästä muunnetusta tasosta (Liite 1, rivit 816–819 ja 828). Täten rajaustiedoston koordinaattijärjestelmänä voidaan käyttää mitä tahansa koordinaattijärjestelmää, jonka QGIS ohjelma tunnistaa.

Lisäksi rajaamisvaiheessa aineistolle ajetaan PDAL:n algoritmi outlier, joka luokittelee selvästi virheelliset pisteet LAS-luokkaan 7 (PDAL 2021c). Ennen lopullista aineiston kirjoitusta kyseiset virhepisteet suodatetaan pois PDAL:n range-algoritmin avulla. Outlier-algoritmin parametrit ovat lähdekoodissa liitteen 1 riveillä 830–835 ja range-algoritmin parametrit riveillä 836–839 (Kuvio 26).

```
821 #Tee JSON Pipeline: Crop
822
823 filel = open(ppoutputdir + '/pdalpipeline_crop.json','w')
824 filel.write('[\n')
825 filel.write('    "' + ppoutputdir + '/PP_yhdistetty.laz",\n')
826 filel.write('    {\n')
827 filel.write('        "type":"filters.crop",\n')
828 filel.write('        "polygon":' + jsongeometriat +'\n')
829 filel.write('    },\n')
830 filel.write('    {\n')
831 filel.write('        "type":"filters.outlier",\n')
832 filel.write('        "method":"radius",\n')
833 filel.write('        "radius":3.0,\n')
834 filel.write('        "min_k":4\n')
835 filel.write('    },\n')
836 filel.write('    {\n')
837 filel.write('        "type":"filters.range",\n')
838 filel.write('        "limits":"Classification![7:7]"\n')
839 filel.write('    },\n')
840 filel.write('    {\n')
841 filel.write('        "type":"writers.las",\n')
842 filel.write('        "filename":"' + ppoutputdir + '/PP_rajattu.laz"\n')
843 filel.write('    }\n')
844 filel.write(']')
845 filel.close()
```

Kuvio 26. PDAL:n CROP-pipelinetiedoston luominen lähdekoodissa

Kuviossa 26 rivillä 832 kohdassa method määrätään outlier-algoritmi suorittamaan suodatuksen säteen perusteella. Jäljempänä kohdassa *radius* määrätään käytettävä säde ja kohdassa *min_k* lähimpien naapureiden vähimmäismäärä. Tällöin esimerkiksi *radius*-arvolla 3.0 ja *min_k*-arvolla 4 virheellisiksi pisteiksi luokitellaan kolmen metrin säteen sisältä ne pisteet, joilla on vähemmän kuin neljä naapuripistettä. (PDAL 2021c.)

Rajaamisen jälkeen pistepilvestä luodaan GeoTiff-muotoinen pintamalli PDAL:n algoritmilla writers.gdal, jota myöhemmin käytetään rakennusten korkeuksien määrittämiseen. Kyseisen algoritmin parametrit ovat liitteen 1 riveillä 861–874 (Kuvio 27).

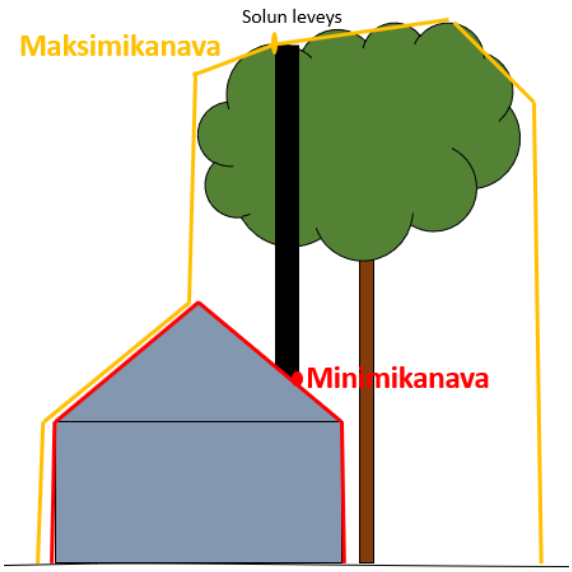
```
#Tee JSON Pipeline: Pointcloud -> DSM

filel = open(ppoutputdir + '/pdalpipeline_dsm.json','w')
filel.write('[\n')
filel.write('    "' + ppoutputdir + '/PP_rajattu.laz",\n')
filel.write('    {\n')
filel.write('        "filename":"' + ppoutputdir + '/PP_rajattu_DSM.tif"
filel.write('        "gdaldriver":"GTiff",\n')
filel.write('        "output_type":"all",\n')
filel.write('        "resolution":"1.0",\n')
filel.write('        "type": "writers.gdal"\n')
filel.write('    }\n')
filel.write(']')
filel.close()

```

Kuvio 27. Pintamallin luomisen parametrit lähdekoodissa

Tärkeimpiä parametrejä ovat output_type ja resolution. Resolution-parametrilla määrätään luotavan pintamallin resoluutio eli rasterin solukoko (PDAL 2021d). Output_type-parametrilla puolestaan määrätään, mitkä kanavat rasteriin luodaan (PDAL 2021d). Arvolla max kirjoitettaisiin puhtaasti yksikanavainen pintamalli, jossa solun z-arvo olisi pistepilven maksimiarvo annetun solukoon sisältä (PDAL 2021d). Arvolla *all* kirjoitetaan myös muun muassa minimi- ja keskiarvokanava (PDAL 2021d). Lisäosa kirjoittaa kaikki kanavat, sillä rakennusten korkeustarkkuuden varmistamiseksi pelkkä maksimikanavan kirjoittaminen ei riitä varsinkaan sellaisissa rakennuksissa, jossa puuston lehvästöt yltävät rakennusten kattojen ylle (Kuvio 28).

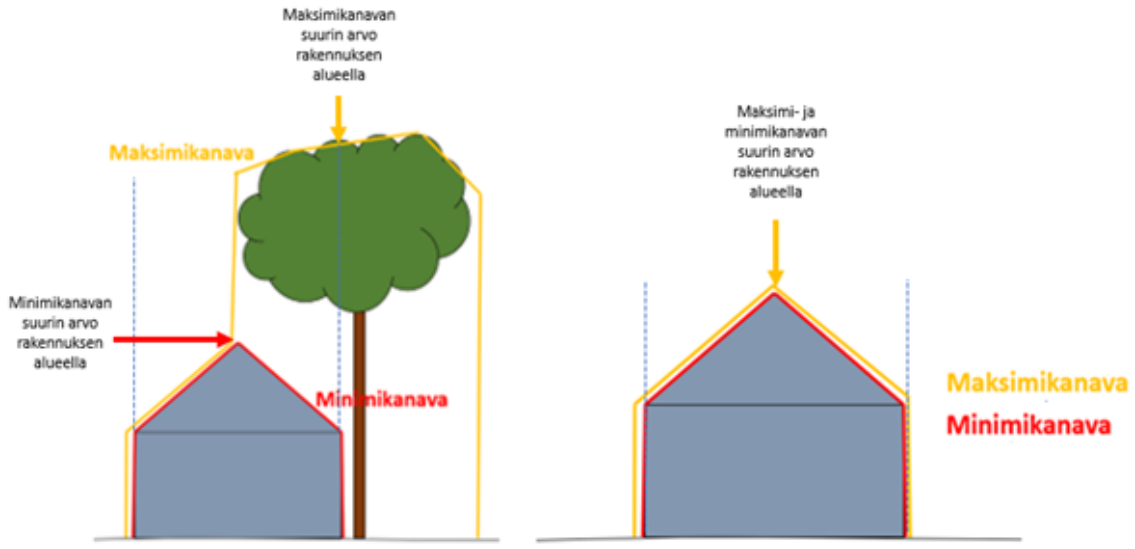


Kuvio 28. Minimi-maksimikanavien tuottamien mallien erot

Kuviossa 28 keltaisella viivalla on kuvattu pintaa, joka tulisi maksimikanavan asetuksella, ja punaisella viivalla kuvattu pintaa, joka tulisi minimikanavan asetuksella silloin kun pistepilvessä on mukana rakennusten ja puuston pisteet.

Kansallisissa ja kansainvälisissä ohjeissa on poikkeavuutta rakennuksen korkeuden määrittämisessä. Kansallisten ohjeiden mukaan rakennusten korkeus tulisi tallentaa rakennuksen korkeimpaan kohtaan, kun taas kansainvälisten ohjeiden mukaan rakennuksen korkeus tulisi tallentaa rakennuksen katon keskikorkeuden mukaan. Rakennusten korkeuksien osalta lisäosa on pyritty ohjelmoimaan kansallisten ohjeiden mukaisesti ja tallentamaan korkeudeksi rakennuksen korkein kohta. Vaikka Maanmittauslaitoksen aineisto on stereomalliluokiteltua aineistoa, aineistossa on paljon pisteitä, jotka ovat väärällä luokituksella, tai niitä ei ole luokiteltu lainkaan. Esimerkiksi rakennusten ja puuston kohdilla luokittelemattomia pisteitä on aineistossa useissa karttalehdissä.

Rakennusten korkeuksien laskemiseksi optimaalisessa tilanteessa pintamalli luotaisiin aineistosta, jossa rakennukset ja puusto olisi luokiteltu kokonaan eri luokkiin ja puuston pisteet suodatettu aineistosta luokituksen mukaan pois. Silloin rakennusten korkeimman kohdan laskemiseksi voitaisiin käyttää rakennuksen muodostaman monikulmion alueen sisältä maksimikanavan suurinta arvoa (Kuvio 29).



Kuvio 29. Vaihtoehdot rakennuksen korkeuden määrittämiseen

Nyt kuitenkin rakennuksia ja puustoa on samalla luokituksella, jolloin kyseistä tapaa käyttämällä rakennuksen korkeudeksi tulisi puuston korkeus. Ongelma voitaisiin ratkaista luokittelemalla aineisto uudestaan PDAL:n avulla, mutta riittävän oikeaan lopputulokseen päästään myös käyttämällä rakennuksen muodostaman monikulmion sisältä rasterin minimikanavan suurinta arvoa ja mahdollisimman pientä resoluutiota (Kuvio 29).

Lopuksi yhdistetty ja rajattu pistepilvitiedosto värjätään yhdistetyn ja rajatun ortoilmakuvan perusteella PDAL:n algoritmilla colorization (Liite1, rivit 896–911). Samassa yhteydessä värjäyksen jälkeen aineisto jaetaan kahteen eri tiedostoon. Toinen tiedosto sisältää kaikkien luokkien pisteet (Liite 1, rivit 912–918) ja toinen tiedosto puuston, siltojen ja mastojen pisteet. Jakaminen tehdään PDAL:n algoritmilla filters.range (Liite 1, rivit 919–932). Range-algoritmissa kasvillisuusluokkiin viitataan luokilla 3, 4 ja 5, siltoihin luokituksella 10 ja mastoihin luokituksella 15 (PDAL 2021e). Tarkempi PDAL:n colorization algoritmin dokumentaatio on PDAL:n verkkosivuilla (PDAL 2021e).

4.3.5 Vaihe 5: Maastotietokannan käsittely

Lisäosan viidennessä vaiheessa Maastotietokannan kohteet käsitellään kolmiulotteisiksi aineistoiksi. Rakennusten kohdalla kolmiulotteinen aineisto toteutetaan ensin laskemalla rakennusten korkeudet maanpinnan suhteen käyttäen apuna korkeusmallia ja muodostettua pintamallia. Saatu korkeus siirretään tasolle *Extrusion*-attribuutiksi, josta tasolle annettu tyylitiedosto hakee rakennuksen korkeuden. Tällä tavoin käyttäjä voi helposti korjata automaation aiheuttamia virheitä muuttamalla vain *Extrusion*-attribuutin arvoja ja tarkastelemalla lopputulosta 3D-näkymässä. Jos taso halutaan kääntää lopulta todelliseksi 3D-geometriaksi, esimerkiksi GML-formaattiin, käyttäjä voi lisäosan suorittamisen jälkeen ajaa tasolle esimerkiksi QGIS:iin integroidun GRASS GIS:n algoritmin v.extrude (GRASS Development Team 2021) ja lisätä saadun tason johonkin lisäosan muodostamaan GeoPackage-tiedostoon. Mastojen, piippujen, pylväiden ja johtojen osalta kolmiulotteinen aineisto toteutetaan myös tyylitiedoston avulla, mutta käyttämällä kullekin kohteelle ennalta määrättyä yhtä korkeutta.

Johtojen, pylväiden ja rakenteiden osalta maaston kohteiden luokittelut on tehty edellä mainittuihin tyylitiedostoihin. Pylväiden tyylitiedostoon kohteista on luokiteltu työn rajauksen mukaisesti suurjännitelinjan pylväät, joiden LUOKKA-attribuutti on maastotietokannan aineistossa 22392 (Maanmittauslaitos 2019). Rakenteiden tyylitiedostoon on luokiteltu mastot, joiden LUOKKA-attribuutti on 44800 (Maanmittauslaitos 2019), sekä savupiiput, joiden LUOKKA-attribuutti on 45300 (Maanmittauslaitos 2019). Lisäksi johtojen tyylitiedostoon on luokiteltu suurjännitelinjat, joiden LUOKKA-attribuutti on 22311 (Maanmittauslaitos 2019).

Rakennusten korkeuksien määrittämisen ensimmäisessä vaiheessa lasketaan korkeusmallin eli maanpinnan ja rakennuksen leikkausviivan alin korkeus käyttämällä algoritmia *zonalstatisticsfb* (Kuvio 30). Kyseisellä algoritmilla voidaan laskea rasteritasosta vyöhykekohtaiset tilastot, jotka tallennetaan tason attribuuttikentiksi. Syötteeksi algoritmille annetaan yhdistetty ja rajattu korkeusmalli ja maastotietokannan rakennusten yhdistetty ja rajattu aineisto.

```
959 zonalDem = processing.run("native:zonalstatisticsfb", {'INPUT':MTK_vlayer_rajattu,
960                                                        'INPUT_RASTER':rajattu,
961                                                        'RASTER_BAND':1,
962                                                        'COLUMN_PREFIX':'zMaa_',
963                                                        'STATISTICS':[5],
964                                                        'OUTPUT':'TEMPORARY_OUTPUT'})
965
966 #Käsittely: maanpinnan korkeus attribuutista z-koordinaateiksi
967
968 setZvalue = processing.run("native:setzvalue", {'INPUT':zonalDem['OUTPUT'],
969                                                 'Z_VALUE':QgsProperty.fromExpression('"zMaa_min"'),
970                                                 'OUTPUT':'TEMPORARY_OUTPUT'})
971
```

Kuvio 30. Vyöhykekohtaisten tilastojen laskenta korkeusmallista ja siirtäminen Z-arvoksi lähdekoodissa

Tärkeimpinä parametreina algoritmin suorittamisessa ovat

- RASTER_BAND, jonka arvolla 1 käytetään rasterin minimikanavaa
- COLUMN_PREFIX, jonka arvolla ’zMaa_’ tilastoattribuuttien etuliitteeksi lisätään zMaa_
- STATISTICS, jonka arvolla 5 lasketaan vyöhykekohtainen minimi. (QGIS 2021c, 922–923.)

Saatu arvo tallennetaan attribuuttitiedon lisäksi monikulmiolle myös geometriseksi Z-koordinaatiksi algoritmilla *setzvalue*, jonka parametrit on esitetty kuvissa 30 riveillä 968–970.

Ennen rakennusten kattojen ylimpien korkeuksien määrittämistä rakennusten monikulmioille ajetaan algoritmi *buffer* (Kuvio 31 ja Liite 1 rivit 975–982), joka tekee rakennusten tasosta kopion ja kopioi rakennusten monikulmioita sisälle päin annetun arvon verran. Toiminnon tarkoituksena on vähentää rakennusten kohdalle osuvien puuston pisteiden määrää ennen kattojen korkeuksien määrittämistä.

```
#Käsittely: Negatiivinen buffer. Pyrkimyksenä pienentää rakennuksen monikulmiota sisälle päin,
#jotta rakennuksen monikulmiolle ei sattuisi puuston pisteitä

Mtk3Dtemp = processing.run("native:buffer", {'INPUT':setZvalue['OUTPUT'],
                                    'DISTANCE':-2.0,
                                    'SEGMENTS':1,
                                    'END_CAP_STYLE':1,
                                    'JOIN_STYLE':1,
                                    'MITER_LIMIT':2,
                                    'DISSOLVE':False,
                                    'OUTPUT':'TEMPORARY_OUTPUT'})
```

Kuvio 31. *Buffer* algoritmi lähdekoodissa

Buffer-algoritmin ajamisen jälkeen rakennusten kattojen korkein kohta määritellään *zonalstatisticsfb*-algoritmin avulla (Kuvio 32 ja Liite 1, rivit 988–994), jonka syötteeksi annetaan pintamalli ja edellä luotu sisennetty rakennusten monikulmiotaso. *Zonalstatisticsfb*-algoritmissa käytetään pintamallin minimikanavaa maksimikanavan sijasta, jonka perustelu on esitetty aiemmin sivuilla 41–42, ja laskettavana statistiikkana pintamallin minimikanavan maksimiarvoa monikulmion sisältä.

```
#Käsittely: Korkeus DSM:stä buffertason rakennuksen katon korkeus-attribuutiksi,
#rasterband 1 on minimikanava, statistics2 on keskiarvo, 6 on maksimi

Mtk3Dtemp = processing.run("native:zonalstatisticsfb", {'INPUT':Mtk3Dtemp['OUTPUT'],
                                            'INPUT_RASTER':dsm,
                                            'RASTER_BAND':1,
                                            'COLUMN_PREFIX':'zKat_',
                                            'STATISTICS':[6],
                                            'OUTPUT':'TEMPORARY_OUTPUT'})
```

Kuvio 32. Rakennusten korkeimman kohdan määrittely *zonalstatisticsfb*-algoritmilla

Laskennan jälkeen tuloksena on sisennetty rakennusten monikulmiotaso, jolla on attribuuttikenttinä *zMaa_min*, jossa on rakennuksen ja maanpinnan leikkausviivan alin korkeusasema, sekä *zKat_max*, jossa on rakennuksen ylin korkeusasema. Rakennusten korkeudet maanpinnan suhteen lasketaan käyttämällä QGIS Expression -dialogia, jossa *zKat_max* kentän arvoista vähennetään *zMaa_min*-kentän arvot. Laskennan tulos tallentuu *Extrusion*-attribuutiksi. Toiminnon lähdekoodi on liitteen 1 riveillä 999–1015. Lopulta saatu *Extrusion*-attribuuttikenttä yhdistetään algoritmilla *joinattributestable* (Liite 1, 1017–1027) siihen tasoon, jossa on rakennusten alkuperäiset muodot ja z-koordinaattina maanpinnan ja rakennuksen leikkausviivan alin korkeusasema.

Lopulta maastotietokannan kaikki osa-aineistot yhdistetään yhdeksi GeoPackage-tietokannaksi algoritmilla *package* (Kuvio 33 ja Liite 1, rivit 1043–1048).

```
1041                #Tehdään Geopackage
1042
1043                outputgeopackage = outputdir + '/Maastotietokanta.gpkg'
1044
1045                processing.run("native:package",{'LAYERS':[MTK3D,MTKjohdot,MTKpylvaat,MTKrakenteet],
1046                                'OUTPUT':outputgeopackage,
1047                                'OVERWRITE':True,
1048                                'SAVE_STYLES':False})
```

Kuvio 33. GeoPackage-tietokannan muodostus lähdekoodissa

Maastotietokannan aineistot lisätään karttaikkunaan muodostetusta GeoPackage-tietokannasta ja kullekin tasolle annetaan omat qml-tyylitiedostonsa (Liite 1, rivit 1050–1078), jotka on toimitettu lisäosan asennuskansion mukana.

4.3.6 Vaihe 6: Lokitiedoston kirjoitus

Lisäosan viimeisessä eli kuudennessa vaiheessa kirjoitetaan lokitiedosto, johon kerätään muodostetut uudet tiedostot sekä alkuperäiset tiedostot, joista uusi aineisto on tehty (Liite 1, rivit 1113–1186). Samassa yhteydessä ohjelma myös poistaa kaikki väliaikaiset tiedostot, joita ohjelma on suorittamisensa aikana muodostanut (Liite 1, rivit 1084–1111). Esimerkki lokitiedostosta on liitteessä 3.

5 LISÄOSAN JULKAISU JA LISENSOINNIT

QGIS:n ylläpitämässä lisäosakirjastossa lisäosien suositellaan toimivan kaikilla QGIS-ohjelman tukemilla käyttöjärjestelmillä eli Windowsilla, MacOS:lla ja Linuxilla. Lisäosan kehitys rajattiin kuitenkin vain Windows-versiolle. Lisäosalle ei myöskään voitu tehdä laajamittaista testaamista, jota kyseinen lisäosakirjasto edellyttäisi. Tämän vuoksi lisäosa julkaistaan vain tämän opinnäytetyön liitteenä (Liite 2).

Lisäosan tekemisessä on käytetty QGIS Plug-in Builder -nimistä lisäosaa, joka on julkaistu GNU General Public License -version 2 alaisena. General Public License on avoin lisenssi, joka mahdollistaa teoksen uudelleen levittämisen ja/tai muokkaamisen (Free Software Foundation 1991), mutta asettaa tiettyjä rajoituksia johdannaisteoksien valmistajille, sillä lisenssiehdot periytyvät johdannaisteokseen (Free Software Foundation 1991). Kuitenkin General Public License mahdollistaa johdannaisteoksen käyttävän myös uudempaa versiota lisenssistä kuin alkuperäinen teos (Free Software Foundation 1991). Koska edellä mainitun QGIS Plug-in Builder lisäosan lisenssiehdot velvoittavat myös luodun lisäosan julkaisua, *Kaupunkimallintaja*-lisäosa julkaistaan lisenssillä General Public License -versio 2.

Lisäosa käyttää hyödykseen myös QGIS:n mukana tullutta PDAL-ohjelmaa, joka on julkaistu kolmen ehdon BSD-lisenssin alla (PDAL 2021f). BSD-lisenssit ovat myös avoimia lisenssejä, joista on useita versioita. Kaikki BSD-lisenssit eivät kuitenkaan ole yhteensopivia GNU General Public License-version 2 kanssa. Yhteensopivuusongelmista johtuen BSD-lisensseistä kehitettiin uusia versioita, kuten kolmen ehdon BSD-lisenssi. Kolmen ehdon BSD-lisenssi ja GNU General Public License versio 2 ovat yhteensopivia lisenssejä. (Free Software Foundation 2021.)

Kaupunkimallintaja-lisäosan asennuskansio sisältää myös Maanmittauslaitoksen avoimista aineistoista karttalehtijaon 06/2021 tiedostoja, jotka on julkaistu Nimeä CC 4.0 lisenssin alaisina. Nimeä CC 4.0 Lisenssi mahdollistaa aineiston kopioinnin, jakamisen ja muuntelun (Creative Commons).

6 ASENNUS- JA KÄYTTÖOHJE

Ohjelmoitu työ rajattiin vain QGIS:n Windows-käyttöjärjestelmän QGIS-versioon 3.18 ja siitä ylöspäin, joten asennus- ja käyttöohje laaditaan myös vain Windows-versiolle. Asennus- ja käyttöohje laadittiin vain suomen kielellä, sillä ohjelman käyttämät aineistot ovat suomalaisia ja siten lisäosakin on suunniteltu suomalaiseen toimintaympäristöön. Asennus- ja käyttöohjeessa käydään lävitse ohjelman asentaminen QGIS-paikkatietosovellukseen sekä opastetaan ohjelman käyttöön. Asennus- ja käyttöohje lisätään lisäosan asennuskansion sisälle.

7 POHDINTA

Opinnäytetyön tavoitteena oli ohjelmoida lisäosa, joka automatisoi kaupunkimallinnusprosessia Maanmittauslaitoksen avoimien aineistojen pohjalta. Tältä osin ohjelma toteuttaa sille asetetut tavoitteet, mutta ohjelmassa ja menetelmässä on myös omat puutteensa. Suurimpana ongelmana toteutuksessa on Maanmittauslaitoksen lähtöaineiston ajantasaisuudesta johtuvat ongelmat varsinkin, jos eri aineistolajit on tuotettu eri aikoina. Esimerkiksi rakennukset voi olla tehty vuonna 2020 ja syötetty maastotietokantaan samana vuonna, mutta pistepilvi, josta muodostetun pintamallin perusteella rakennusten korkeudet tulisi laskea, on tuotettu vuonna 2010. Tällöin rakennusten korkeudet tallentuvat vuoden 2010 tilanteen mukaan virheellisesti. Rakennusten korkeuksien osalta ongelmia aiheuttaa myös se, että Maastotietokannan rakennusten aineistoissa on monesti vain yksi monikulmio rakennusta kohden riippumatta rakennuksen eri kerrosluvuista. Näin rakennukset, joissa on vaihtelevia kerroslukuja, mallintuvat väärin korkeimman kerrosluvun mukaan. Samoin pistepilvet värjäytyvät väärin, jos ortoilmakuva ja pistepilvi on tuotettu eri vuosina. Tämä ongelma on kuitenkin helposti havaittavissa ja siten myös helpohkosti korjattavissa käsityönä. Myös pistepilviaineiston resoluutio 0,5 pistettä /m² on liian harva, jotta aineistosta voitaisiin määrittää korkeuksia mastoille, pylväille tai piipuille. Siten toteutuksessa oli käytettävä ennalta määrättyjä korkeuksia.

Osaltaan riskialtista on myös se, että *Kaupunkimallintaja*-lisäosa nojaa kartat.kapsi.fi-palvelimeen, jonka saatavuutta ja toimivuutta ylläpitäjä ei takaa. Tämän vuoksi lisäosakaan ei toimi, jos kartat.kapsi.fi-palvelun tarjoaja päättää lopettaa palvelimen ylläpidon. Osaltaan viitteitä tästä saatiin myös jo lisäosan testausaikana, jolloin huomattiin, että kartat.kapsi.fi-palvelimella on useissa karttalehdissä ja aineistolajeissa tiedostoja, jotka ovat korruptoituneita. Siten myöskään lisäosa ei pysty takaamaan, että kaupunkimallinnus onnistuisi lisäosalla kaikkialta Suomesta. Lisäosa myös nojaa stereomalliluokiteltuun pistepilviaineistoon, jota Maanmittauslaitos ei ole tuottanut koko Suomen alueelta. Lisäosan toimimattomuus voi johtua myös siitä, ettei stereomalliluokiteltua pistepilviaineistoa halutulta alueelta ole olemassa. Yksi lisäosan jatkokehityskohde olisi käyttää automaattiluokiteltuja pistepilviaineistoja ja luokitella aineistot esimerkiksi PDAL:n avulla lisäosan suorittamisen aikana.

Varsinaisen ohjelmakoodin osalta tärkeimpinä kehityskohteina puolestaan pitäisin koodin uudelleenkäytettävyyden parantamista ja toimintojen luomista funktioihin ja luokkiin. Osaltaan kuitenkin verrattain vähäinen ohjelmointikokemus vaikutti siihen, että ohjelmakoodista tuli melko yksinkertaista ja aloittelijamaista. Aloittelijamaisuudesta johtuen myös virheiden käsittely koodissa on osittain puutteellista. Myöskään ohjelman laajamittaista testaamista ei pystytty toteuttamaan, joten ohjelmassa voi esiintyä suuria virheitä. Ohjelman testaamisen aikana havaittiin kaksi merkittävää ongelmaa:

- skandinaavisen merkistön tai erikoismerkkien käyttäminen tiedostonimissä kaataa ohjelman
- väliaikaisen rajaustason käyttäminen Shapefile-tiedoston sijasta kaataa ohjelman.

Ohjelman käyttöliittymän ja paremman käytettävyyden kannalta käyttöliittymään olisi hyvä lisätä myös käyttäjälle enemmän päätäntävaltaa siitä, mitä aineistolajeja halutaan noudettavan Kapsin palvelimelta ja mitä resoluutiota pintamallin luomisessa käytetään. Toisaalta käyttäjän voitaisiin myös antaa valita oma pintamalliaineisto.

Tämän kaltaisessa työssä haasteita oli myös työn rajaamisessa, sillä yhden henkilön ohjelmointityöksi työ olisi voinut kasvaa kohtuuttoman suureksi, sillä mallinnusohjeiden mukaisia kohteita olisi ollut vielä useita kymmeniä. Kokonaisuutena, puutteistakin huolimatta, työ on onnistunut, sillä se antoi varmuutta Python-ohjelmointikielen käyttöön paikkatietoaineistojen käsittelyssä ja työ osoittaa sen, että kaupunkimallintamisen automatisointi Python-ohjelmointikielellä Maanmittauslaitoksen avoimien aineistojen pohjalta on mahdollista.

LÄHTEET

ASPRS 2008. LAS Specification v.1.2. Viitattu 18.8.2021 https://www.asprs.org/a/society/committees/standards/asprs_las_format_v12.pdf.

BuildingSMARTFinland 2016a. Kaupunkimallinnuksen ohjekirja. Viitattu 6.8.2021 https://buildingsmart.fi/kaupunki/kaupunkimallinnuksen-ohjekirja/.

Creative Commons. Nimeä 4.0 Kansainvälinen (CC BY 4.0). Viitattu 4.10.2021 https://creativecommons.org/licenses/by/4.0/deed.fi.

Free Software Foundation 1991. GNU General Public License Version 2. Viitattu 16.9.2021 https://www.gnu.org/licenses/old-licenses/gpl-2.0.html.

Free Software Foundation 2021. Various Licenses and Comments about Them. Viitattu 4.10 https://www.gnu.org/licenses/license-list.en.html#ModifiedBSD.

GDAL 2021a. GDAL Documentation. Programs. Viitattu 2.9.2021 https://gdal.org/programs/gdal_merge.html.

GDAL 2021b. GDAL Documentation. GTiff – GeoTIFF file format. Viitattu 5.9.2021 https://gdal.org/drivers/raster/gtiff.html#raster-gtiff.

GRASS Development Team 2021. GRASS GIS 7.8.6dev Reference Manual. Viitattu 14.9.2021 https://grass.osgeo.org/grass78/manuals/v.extrude.html.

Haggrén, H. 2002 & Honkavaara, E. 2005. Ortokuvien tuottaminen. Luentomoniste. Viitattu 17.8.2021 https://foto.aalto.fi/opetus/220/luennot/7/L7_2005.pdf.

JHS 197 v1.1 2016. EUREF-FIN -koordinaattijärjestelmät, niihin liittyvät muunnokset ja karttalehtijako. Viitattu 12.8.2021 https://www.suomidigi.fi/sites/default/files/2020-07/JHS197.doc.

JHS 210 v.1.0 2020. Paikkatietojen mallintaminen: rakennukset ja rakennelmat. Viitattu 12.8.2021 https://www.suomidigi.fi/sites/default/files/2020-07/JHS210.doc.

JHS 210 v.1.0 2020. Paikkatietojen mallintaminen: rakennukset ja rakennelmat: Liite 5: Geometrioiden muodostamisohjeet. Viitattu 12.8.2021 https://www.suomidigi.fi/sites/default/files/2020-07/JHS210_liite5.doc.

Jokela, S. 2017. QGIS kiinnostaa kaupunkeja. Positio 1/2017, 26–27. Viitattu 9.8.2021https://www.maanmittauslaitos.fi/sites/maanmittauslaitos.fi/files/attachments/2017/06/Positio_1_2017_0.pdf.

Kapsi Internet-käyttäjät ry 2015. Kartat.kapsi.fi. Viitattu 9.8.2021 https://kartat.kapsi.fi.

Kukko, A. 2005. Laserkeilaimen valinta lähifotogrammetrisiin mittaustehtäviin. Fotogrammetrian erikoistyö. Viitattu 18.8. https://foto.aalto.fi/opetus/290/julkaisut/Antero_Kukko/Laserkeilaimen_valinta_lahifotogrammetrisiin_mittauksiin.pdf.

Laurila, P. 2012. Mittaus- ja kartoitustekniikan perusteet. 4. Uudistettu painos. Rovaniemi: Rovaniemen ammattikorkeakoulu. Sarja D. Nro 3. Viitattu 18.8.2021 http://www.ramk.fi/loader.aspx?id=7fe99c68-3849-4fa8-a563-9327cf51ea79.

Maanmittauslaitos 2019. Maastotietokohteiden luokitus. Viitattu 15.9.2021 https://www.maanmittauslaitos.fi/sites/maanmittauslaitos.fi/files/attachments/2019/12/maastotietokanta_kohdemalli.xlsx.

Maanmittauslaitos 2020a. Rakennusten ja rakennelmien JHS 210 -suositus on julkaistu. Viitattu 9.8.2021 https://www.maanmittauslaitos.fi/ajankohtaista/rakennusten-ja-rakennelmien-jhs-210-suositus-julkaistu.

Maanmittauslaitos 2020b. Maastotietokannan sisältö teemoittain. Viitattu 18.8.2021 https://www.maanmittauslaitos.fi/sites/maanmittauslaitos.fi/files/attachments/2020/06/Maastotietokannan%20sis%C3%A4ll%C3%B6n%20kuvaus%20teemoittain_p%C3%A4ivitys_2020.pdf.

Maanmittauslaitos 2020c. Laserkeilausaineisto 0,5p. Viitattu 18.8.2021 https://www.maanmittauslaitos.fi/kartat-ja-paikkatieto/asiantuntevalle-kayttajalle/tuotekuvaukset/laserkeilausaineisto-05-p.

Maanmittauslaitos 2021a. Testiaineistoa saatavilla Maanmittauslaitoksen tuottamista rakennusten 3D-vektoreista. Viitattu 9.8.2021 https://www.maanmittauslaitos.fi/ajankohtaista/testiaineistoa-saatavilla-maanmittauslaitoksen-tuottamista-rakennusten-3d-vektoreista.

Maanmittauslaitos 2021b. Avoimien aineistojen tietopalvelu. Viitattu 9.8.2021 https://www.maanmittauslaitos.fi/asioi-verkossa/avoimien-aineistojen-tiedostopalvelu#palvelun-sisalto.

Maanmittauslaitos 2021c. Avoimien aineistojen tiedostopalvelu. Viitattu 10.8.2021 https://tiedostopalvelu.maanmittauslaitos.fi/tp/kartta.

Maanmittauslaitos 2021d. Maanmittauslaitoksen avoimen tietoaineiston Nimeä CC 4.0 -lisenssi. Viitattu 11.8.2021 https://www.maanmittauslaitos.fi/avoindata-lisenssi-cc40.

Marttila, E. 2019. Laserkeilauksen uusi aikakausi alkaa. Tietoa Maasta 4/2019, 12–13. Viitattu 6.8.2021 https://www.maanmittauslaitos.fi/sites/maanmittauslaitos.fi/files/attachments/2020/01/Tietoa-Maasta-4-2019.pdf.

Open Geospatial Consortium 2012. CityGML Specification, 14–15. Viitattu 9.8.2021 https://portal.ogc.org/files/?artifact_id=47842.

PDAL 2021a. PDAL documentation. Applications. Viitattu 10.9.2021 https://pdal.io/apps/index.html.

PDAL 2021b. PDAL documentation. Pipeline. Viitattu 10.9.2021 https://pdal.io/apps/pipeline.html.

PDAL 2021c. PDAL documentation. Filters.Outlier Viitattu 11.9.2021 https://pdal.io/stages/filters.outlier.htm.

PDAL 2021d. PDAL documentation. Writers.gdal. Viitattu 13.9.2021 https://pdal.io/stages/writers.gdal.html#radius.

PDAL 2021e. PDAL documentation. Filters.Colorization Viitattu 13.9.2021 https://pdal.io/stages/filters.colorization.html?highlight=colorization.

PDAL 2021f. PDAL License. Viitattu 4.10.2021 https://pdal.io/copyright.html.

Python 2021a. Python 3.9.7 Documentation. Extending Python with C or C++. Viitattu 22.9.2021 https://docs.python.org/3/extending/extending.html.

Python 2021b. Python Changelog 3.9.6. Viitattu 22.9.2021 https://docs.python.org/release/3.9.6/whatsnew/changelog.html#changelog.

Python 2021c. Python documentation 3.9.7. HOWTO Fetch Internet Resources Using The urllib Package. Viitattu 5.9.2021 https://docs.python.org/3/howto/urllib2.html.

QGIS 2021a. Changelog for QGIS 3.18. Viitattu 9.8.2021 https://qgis.org/fi/site/forusers/visualchangelog318/index.html#changelog-for-qgis-3-18.

QGIS 2021b. PyQGIS testing developer cookbook. Viitattu 9.8.2021 https://docs.qgis.org/testing/pdf/en/QGIS-testing-PyQGISDeveloperCookbook-en.pdf.

QGIS 2021c. QGIS User guide. Viitattu 2.9.2021 https://docs.qgis.org/testing/pdf/en/QGIS-testing-DesktopUserGuide-en.pdf.

QGIS 2021d. QGIS Documentation. Configuring external applications. Viitattu 24.8.2021 https://docs.qgis.org/testing/en/docs/user_manual/processing/3rdParty.html.

SIG3D Quality Working Group 2015. Opas 3D-esineiden mallintamiseen, Osa 2: Rakennusten mallintaminen (LoD1, LoD2 ja LoD3). Viitattu 6.8.2021 https://buildingsmart.fi/wp-content/uploads/2016/11/VER_0.99_-_201311_SIG3D_Modeling_Guide_for_3D_Objects_Part_2.pdf.

Sirkiä, O., Ahonen, T., Ilves, R., Pyysalo, U., Ahokas, E & Oksanen, J. 2017. Kansallisen maastotietokannan laatumalli. Korkeusmallit. V.1.3. Viitattu 16.8.2021 https://www.maanmittauslaitos.fi/sites/maanmittauslaitos.fi/files/attachments/2017/05/KMTK_korkeusmallit_laatukasikirja_2017-01-02.pdf.

Zelle, J. 2004. Python programming: An introduction to computer science. Oregon: Franklin. Beedle & Associates Inc.

LIITTEET

Liite 1. Kaupunki_mallintaja.py-lähdekoodi

Liite 2. Ohjelman asennuskansio

Liite 3. Esimerkki lokitiedostosta

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 Kaupunkimallintaja
                                 A QGIS plugin
 Lisäosa hakee rajatun alueen perusteella Kapsi ry:n palvelimelta Maanmittauslaitoksen avoimia aineistoja
 (CC Nimeä 4.0 Lisenssi https://www.maanmittauslaitos.fi/avoindata-lisenssi-cc40) (Korkeusmalli,
 Ortoilmakuva, Maastotietokanta & Pistepilvi) ja muodostaa haetuista aineistoista kaupunkimallin.
 Generated by Plugin Builder: http://g-sherman.github.io/Qgis-Plugin-Builder/
                             -------------------
        begin                : 2021-05-09
        git sha              : $Format:%H$
        copyright            : (C) 2021 by Ilari Issakainen
        email                :
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
"""
from qgis.PyQt.QtCore import QSettings, QTranslator, QCoreApplication, QVariant
from qgis.PyQt.QtGui import QIcon
from qgis.PyQt.QtWidgets import QAction,QFileDialog
from qgis.core import
QgsProject,Qgis,QgsMapLayerProxyModel,QgsRasterLayer,QgsVectorLayer,QgsApplication,QgsProperty,QgsField,QgsExp
ression,QgsExpressionContext,QgsExpressionContextUtils,QgsCoordinateReferenceSystem
from qgis.utils import iface
from qgis.core.additions.edit import edit

# Initialize Qt resources from file resources.py
from .resources import *
# Import the code for the dialog
from .Kaupunki_mallintaja_dialog import KaupunkimallintajaDialog
import os.path
import os
import processing
import urllib
import json
import subprocess
import time

def resolve(name, basepath=None):                                   #This
    if not basepath:                                                #function is
      basepath = os.path.dirname(os.path.realpath(__file__))        #adapted from
    return os.path.join(basepath, name)
    #https://gis.stackexchange.com/questions/130027/getting-a-plugin-path-using-python-in-qgis

class Kaupunkimallintaja:
    """QGIS Plugin Implementation."""

    def __init__(self, iface):
        """Constructor.

        :param iface: An interface instance that will be passed to this class
            which provides the hook by which you can manipulate the QGIS
            application at run time.
        :type iface: QgsInterface
        """
        # Save reference to the QGIS interface
        self.iface = iface
        # initialize plugin directory
        self.plugin_dir = os.path.dirname(__file__)
        # initialize locale
        locale = QSettings().value('locale/userLocale')[0:2]
        locale_path = os.path.join(
            self.plugin_dir,
            'i18n',
            'Kaupunkimallintaja_{}.qm'.format(locale))

        if os.path.exists(locale_path):
            self.translator = QTranslator()
            self.translator.load(locale_path)
            QCoreApplication.installTranslator(self.translator)

        # Declare instance attributes
        self.actions = []
        self.menu = self.tr(u'&Kaupunkimallintaja')

        # Check if plugin was started the first time in current QGIS session
        # Must be set in initGui() to survive plugin reloads
        self.first_start = None

    # noinspection PyMethodMayBeStatic
    def tr(self, message):
        """Get the translation for a string using Qt translation API.
```

```
        We implement this ourselves since we do not inherit QObject.

        :param message: String for translation.
        :type message: str, QString

        :returns: Translated version of message.
        :rtype: QString
        """
        # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
        return QCoreApplication.translate('Kaupunkimallintaja', message)


    def add_action(
        self,
        icon_path,
        text,
        callback,
        enabled_flag=True,
        add_to_menu=True,
        add_to_toolbar=True,
        status_tip=None,
        whats_this=None,
        parent=None):
        """Add a toolbar icon to the toolbar.

        :param icon_path: Path to the icon for this action. Can be a resource
            path (e.g. ':/plugins/foo/bar.png') or a normal file system path.
        :type icon_path: str

        :param text: Text that should be shown in menu items for this action.
        :type text: str

        :param callback: Function to be called when the action is triggered.
        :type callback: function

        :param enabled_flag: A flag indicating if the action should be enabled
            by default. Defaults to True.
        :type enabled_flag: bool

        :param add_to_menu: Flag indicating whether the action should also
            be added to the menu. Defaults to True.
        :type add_to_menu: bool

        :param add_to_toolbar: Flag indicating whether the action should also
            be added to the toolbar. Defaults to True.
        :type add_to_toolbar: bool

        :param status_tip: Optional text to show in a popup when mouse pointer
            hovers over the action.
        :type status_tip: str

        :param parent: Parent widget for the new action. Defaults None.
        :type parent: QWidget

        :param whats_this: Optional text to show in the status bar when the
            mouse pointer hovers over the action.

        :returns: The action that was created. Note that the action is also
            added to self.actions list.
        :rtype: QAction
        """

        icon = QIcon(icon_path)
        action = QAction(icon, text, parent)
        action.triggered.connect(callback)
        action.setEnabled(enabled_flag)

        if status_tip is not None:
            action.setStatusTip(status_tip)

        if whats_this is not None:
            action.setWhatsThis(whats_this)

        if add_to_toolbar:
            # Adds plugin icon to Plugins toolbar
            self.iface.addToolBarIcon(action)

        if add_to_menu:
            self.iface.addPluginToMenu(
                self.menu,
                action)

        self.actions.append(action)

        return action

    def initGui(self):
        """Create the menu entries and toolbar icons inside the QGIS GUI."""

```

```
        icon_path = ':/plugins/Kaupunki mallintaja/icon.png'
        self.add_action(
            icon_path,
            text=self.tr(u'Kaupunkimallintaja'),
            callback=self.run,
            parent=self.iface.mainWindow())

        # will be set False in run()
        self.first_start = True


    def unload(self):
        """Removes the plugin menu item and icon from QGIS GUI."""
        for action in self.actions:
            self.iface.removePluginMenu(
                self.tr(u'&Kaupunkimallintaja'),
                action)
            self.iface.removeToolBarIcon(action)


    def select_output_dir(self):
      outputdir = str(QFileDialog.getExistingDirectory(
        self.dlg, "Select output directory"))
      self.dlg.lineEdit.setText(outputdir)


    def run(self):
        """Run method that performs all the real work"""

        # Create the dialog with elements (after translation) and keep reference
        # Only create GUI ONCE in callback, so that it will only load when the plugin is started
        if self.first_start == True:
            self.first_start = False
            self.dlg = KaupunkimallintajaDialog()
            self.dlg.pushButton.clicked.connect(self.select_output_dir)

        self.dlg.mMapLayerComboBox.setFilters(QgsMapLayerProxyModel.PolygonLayer)


        # show the dialog
        self.dlg.show()
        # Run the dialog event loop
        result = self.dlg.exec_()
        # See if OK was pressed
        if result:
            # Do something useful here - delete the line containing pass and
            # substitute with your code.
            outputdirbase = self.dlg.lineEdit.text()
            #outputdirname = os.path.basename(outputdir)


            def rmvLyr(lyrname):                                        #This fuction is
                qinst = QgsProject.instance()                           #adapted from

                qinst.removeMapLayer(qinst.mapLayersByName(lyrname)[0].id())#https://gis.stackexchange.com/questions/306615/pyqgis-remove-layer-from-legend

            #INPUTS: Valittu taso ja pluginin kansiossa oleva karttalehtijako.
            #Tehdään myös eri kansiot per monikulmio, joka mahdollistaa laajemman alueen
            #mallintamisen osissa

            selectedLayer = self.dlg.mMapLayerComboBox.currentLayer()

            featIDs=[]
            for feature in selectedLayer.getFeatures():
                featIDs.append(feature.id())

            for fts in featIDs:
                outputdir = outputdirbase+ '/' + str(fts + 1)
                os.mkdir(outputdir)

                outputdirname = os.path.basename(outputdir)

                ppoutputdir = outputdir + '/Pistepilvi'
                os.mkdir(ppoutputdir)

                selectedLayer.setSubsetString("fid=" + str(fts))
                iface.mapCanvas().refresh()

                karttalehtijako=resolve('utm10.shp')

                ##############################
                ######   PROSESSOINTI   #######
                ##############################

                ################################################
                ### VAIHE 1: KORKEUSMALLIEN HAKU JA PROSESSOINTI

                #Karttalehdetattribuutteihin
                lehtitunnukset = processing.run("native:joinattributesbylocation", {'INPUT':selectedLayer,
```

```
                                                    'JOIN':karttalehtijako,
                                                    'PREDICATE':[1,2,4,5],
                                                    'JOIN FIELDS':['LEHTITUNNU'],
                                                    'METHOD':0,'DISCARD_NONMATCHING':False,
                                                    'PREFIX':'',
                                                    'OUTPUT':'TEMPORARY_OUTPUT'})

                #Tee lista karttalehtien osoitteista ja nouda karttalehdet kapsin palvelimelta
                vlayer = lehtitunnukset['OUTPUT']

                osoitteet=[]
                karttalehdet=[]
                raportinOsoitteet=[]
                for feature in vlayer.getFeatures():
                    lehti1 = feature["LEHTITUNNU"]
                    l1 = lehti1[0:2] + '/'
                    le1=lehti1[0:3] + '/'
                    uri1 ='http://kartat.kapsi.fi/files/korkeusmalli/hila_2m/etrs-tm35fin-n2000/' +l1 + le1
                    + lehti1 + '.tif'

                    #Testataan löytyykö tiedosto palvelimelta
                    if not urllib.request.urlopen (uri1).code == 200:

                        self.iface.messageBar().pushMessage('Error','Haku kapsin palvelimelta ei onnistunut!
                        Näyttää siltä, että haluttua tasoa ' + uri1 + ' ei löytynyt.', level=Qgis.Critical,
                        duration=20)

                        raise NameError('Tiedostoa ei löydy')

                    else:

                        processing.run("native:filedownloader", {'URL':uri1,
                                                                 'OUTPUT':outputdir + '/' + lehti1 + '.tif' })

                        #Testataan onko tiedosto viallinen tai 0 kilotavua
                        rlayer = QgsRasterLayer(outputdir + '/' + lehti1 + '.tif',lehti1,'gdal')
                        if not rlayer.isValid():
                            self.iface.messageBar().pushMessage('Error','Haku kapsin palvelimelta ei
                            onnistunut! Näyttää siltä, että haluttu taso ' + uri1 + ' oli viallinen.',
                            level=Qgis.Critical, duration=20)

                            raise NameError('Haluttu taso oli viallinen')

                        #Muutoin hyväksytään ja jatketaan
                        else:

                            QgsProject.instance().addMapLayer(rlayer)
                            QgsProject.instance().removeMapLayer(rlayer.id())
                            osoitteet.append(outputdir + '/' + lehti1 + '.tif')
                            raportinOsoitteet.append(uri1)
                            karttalehdet.append(lehti1)

                #Yhdistäminen
                yhdistettyOutput = processing.run("gdal:merge", {'INPUT':osoitteet,
                                                'PCT':False,
                                                'SEPARATE':False,
                                                'NODATA_INPUT':None,
                                                'NODATA_OUTPUT':None,
                                                'OPTIONS':'',
                                                'EXTRA':'',
                                                'DATA_TYPE':5,
                                                'OUTPUT':'TEMPORARY_OUTPUT'})


                #leikkaus
                clipped = processing.run("gdal:cliprasterbymasklayer", {'INPUT':yhdistettyOutput['OUTPUT'],
                                                        'MASK':selectedLayer,
                                                        'SOURCE_CRS':None,
                                                        'TARGET_CRS':None,
                                                        'NODATA':0,
                                                        'ALPHA_BAND':False,
                                                        'CROP_TO_CUTLINE':True,
                                                        'KEEP_RESOLUTION':False,
                                                        'SET_RESOLUTION':False,
                                                        'X_RESOLUTION':None,
                                                        'Y_RESOLUTION':None,
                                                        'MULTITHREADING':False,
                                                        'OPTIONS':'',
                                                        'DATA_TYPE':0,
                                                        'EXTRA':'',
                                                        'OUTPUT':'TEMPORARY_OUTPUT'})

                clippedGPKG = processing.run("gdal:translate", {'INPUT':clipped['OUTPUT'],
                                                    'TARGET_CRS':None,
                                                    'NODATA':0,
                                                    'COPY_SUBDATASETS':False,
                                                    'OPTIONS':'',
                                                    'EXTRA':'',
                                                    'DATA_TYPE':0,
```

```
                                                'OUTPUT':outputdir + '/Korkeusmalli.gpkg'})

                rajattu=iface.addRasterLayer(clippedGPKG['OUTPUT'], outputdirname + ' Korkeusmalli','gdal')

                rasterDataProvider = rajattu.dataProvider()

                rasterDataProvider.setNoDataValue(1, 0) #eka viittaa kaistanumeroon

                rajattu.triggerRepaint()

                #Poistetaan ylimääräiset tiedostot koneelta

                for item in osoitteet:
                    if os.path.isfile(item):
                        try:
                            os.remove(item)
                        except:
                            pass



                ################################################
                ### VAIHE 2: ORTOILMAKUVIEN HAKU JA PROSESSOINTI

                #Haetaan tarvittavat lehtitunnukset

                orto_karttalehdet =[]

                for feature in vlayer.getFeatures():
                    lehti1 = feature["LEHTITUNNU"]

                    orto_karttalehdet.append(lehti1)

                Orto_raportinOsoitteet=[]
                koneen_osoitteet=[]
                cropatut_osoitteet=[]

                #Testataan Kapsin palvelimelta mistä kansiosta tiedostot löytyy

                for lehti in orto_karttalehdet:
                    karttalehtikansio = '/' + lehti[0:3]

                    #Kapsin palvelimella jaotellaan tiedostot vuosiluvun mukaan kansioihin. #Haetaan
                    vuosilta 2006-2039.

                    vuodet = []
                    for i in range(2006,2039):
                        vuodet.append(i)

                    for item in vuodet:
                        item_str=str(item)

                    #Kapsin palvelimella jaotellaan myös tiedostot kansioihin, SMK, MARA ja MAVI, testataan
                    mistä löytyy tiedosto

                        #SMK:T
                        try:
                            if urllib.request.urlopen
                            ('http://kartat.kapsi.fi/files/orto/etrs-tm35fin/smk_v_15000_50/' + item_str +
                            karttalehtikansio + '/02m/1/' + lehti + '.jp2').code == 200:
                                oikea_osoite =
                                ('http://kartat.kapsi.fi/files/orto/etrs-tm35fin/smk_v_15000_50/' + item_str
                                + karttalehtikansio + '/02m/1/' + lehti + '.jp2')
                        except:
                            try:
                                if urllib.request.urlopen
                                ('http://kartat.kapsi.fi/files/orto/etrs-tm35fin/smk_v_15000_50/' + item_str
                                + karttalehtikansio + '/10m/1/' + lehti + '.jp2').code == 200:
                                    oikea_osoite =
                                    ('http://kartat.kapsi.fi/files/orto/etrs-tm35fin/smk_v_15000_50/' +
                                    item_str + karttalehtikansio + '/10m/1/' + lehti + '.jp2')
                            except:

                                #MARAT
                                try:
                                    if urllib.request.urlopen
                                    ('http://kartat.kapsi.fi/files/orto/etrs-tm35fin/mara_v_25000_50/' +
                                    item_str + karttalehtikansio + '/02m/1/' + lehti + '.jp2').code == 200:
                                        oikea_osoite =
                                        ('http://kartat.kapsi.fi/files/orto/etrs-tm35fin/mara_v_25000_50/' +
                                        item_str + karttalehtikansio + '/02m/1/' + lehti + '.jp2')
                                except:
                                    try:
                                        if urllib.request.urlopen
                                        ('http://kartat.kapsi.fi/files/orto/etrs-tm35fin/mara_v_25000_50/' +
                                        item_str + karttalehtikansio + '/10m/1/' + lehti + '.jp2').code ==
                                        200:
                                            oikea_osoite =
                                            ('http://kartat.kapsi.fi/files/orto/etrs-tm35fin/mara_v_25000_50/'
                                             + item_str + karttalehtikansio + '/10m/1/' + lehti + '.jp2')
```

```
                                except:

                                #MAVIT
                                    try:
                                        if urllib.request.urlopen
                                        ('http://kartat.kapsi.fi/files/orto/etrs-tm35fin/mavi_v_25000_50/'
                                         + item_str + karttalehtikansio + '/02m/1/' + lehti +
                                        '.jp2').code == 200:
                                            oikea_osoite =
                                            ('http://kartat.kapsi.fi/files/orto/etrs-tm35fin/mavi_v_25000_
                                            50/' + item_str + karttalehtikansio + '/02m/1/' + lehti +
                                            '.jp2')
                                    except:
                                        try:
                                            if urllib.request.urlopen
                                            ('http://kartat.kapsi.fi/files/orto/etrs-tm35fin/mavi_v_25000_
                                            50/' + item_str + karttalehtikansio + '/10m/1/' + lehti +
                                            '.jp2').code == 200:
                                                oikea_osoite =
                                                ('http://kartat.kapsi.fi/files/orto/etrs-tm35fin/mavi_v_25
                                                000_50/' + item_str + karttalehtikansio + '/10m/1/' +
                                                lehti + '.jp2')
                                        except:
                                            continue

                            #Kun on saatu oikea osoite, ladataan tiedosto

                            processing.run("native:filedownloader", {'URL':oikea_osoite,
                                                                     'OUTPUT':outputdir + '/' + lehti + '.jp2' })

                            #Kerätään koneenosoitteet raporttia ja ja käsittelyä varten

                            uri= outputdir + '/' + lehti + '.jp2'
                            koneen_osoitteet.append(uri)

                            #Kerätään oikeat palvelimen osoitteet raporttia varten

                            Orto_raportinOsoitteet.append(oikea_osoite)

                            #Clipataan jokainen tiedosto erikseen isojen tiedostokokojen takia

                            processing.run("gdal:cliprasterbymasklayer", {'INPUT':uri,
                                                                          'MASK':selectedLayer,
                                                                          'SOURCE_CRS':None,
                                                                          'TARGET_CRS':None,
                                                                          'NODATA':0,
                                                                          'ALPHA_BAND':False,
                                                                          'CROP_TO_CUTLINE':True,
                                                                          'KEEP_RESOLUTION':False,
                                                                          'SET_RESOLUTION':False,
                                                                          'X_RESOLUTION':None,
                                                                          'Y_RESOLUTION':None,
                                                                          'MULTITHREADING':True,

                                                                          'OPTIONS':'COMPRESS=DEFLATE|PREDICTOR=2|ZL
                                                                          EVEL=9|BIGTIFF=YES',
                                                                          'DATA_TYPE':0,
                                                                          'EXTRA':'',
                                                                          'OUTPUT':outputdir + '/ortocrop_' +
                                                                          lehti + '.tif' })

                            #Kerätään cropatut koneen osoitteet virtuaalirasterin muodostusta varten
                            #(Virtuaalirasteri ei lue koko tiedostoja ensin koneen muistiin toisin kuin merge ->
                            nopeampi käsittely)

                            cropatut_osoitteet.append(outputdir + '/ortocrop_' + lehti + '.tif')

                    #Muodostetaan virtuaalirasteri

                    vrt=processing.run("gdal:buildvirtualraster", {'INPUT':cropatut_osoitteet,
                                                                   'RESOLUTION':0,
                                                                   'SEPARATE':False,
                                                                   'PROJ_DIFFERENCE':False,
                                                                   'ADD_ALPHA':True,
                                                                   'ASSIGN_CRS':None,
                                                                   'RESAMPLING':0,
                                                                   'SRC_NODATA':'',

                                                                   'EXTRA':'',

                                                                   'OUTPUT':'TEMPORARY_OUTPUT'})

                    #tehdään väliaikainen tiffi pistepilvien värjäystä varten
                    resultortotif = processing.run("gdal:translate", {'INPUT':vrt['OUTPUT'],
                                                                      'TARGET_CRS':None,
                                                                      'NODATA':0,
                                                                      'COPY_SUBDATASETS':False,
                                                                      'OPTIONS':'',
                                                                      'EXTRA':'',
                                                                      'DATA_TYPE':0,
```

```
                                        'OUTPUT':outputdir + '/' + outputdirname + '
                                        Ortoilmakuva.tif'})

            #tehdään varsinainen orto gpkg tiedostoksi
            resultortogpkg = processing.run("gdal:translate", {'INPUT':vrt['OUTPUT'],
                                        'TARGET_CRS':None,
                                        'NODATA':0,
                                        'COPY_SUBDATASETS':False,
                                        'OPTIONS':'',
                                        'EXTRA':'',
                                        'DATA_TYPE':0,
                                        'OUTPUT':outputdir + '/Ortoilmakuva.gpkg'})

            #Luetaan varsinainen yhdistetty ja rajattu orto kartalle

            ortogpkg = iface.addRasterLayer(resultortogpkg['OUTPUT'],outputdirname + '
            Ortoilmakuva','gdal')

            #Additional nodata value 0, koska gpkg ei tue suoraan nodata valueta

            rasterDataProvider = ortogpkg.dataProvider()

            rasterDataProvider.setNoDataValue(1, 0) #eka viittaa kaistan numeroon

            ortogpkg.triggerRepaint()

            #Poistetaan väliaikaset tiedostot koneelta

            for item in koneen_osoitteet:
                if os.path.isfile(item):
                    try:
                        os.remove(item)
                    except:
                        pass

            for item in cropatut_osoitteet:
                if os.path.isfile(item):
                    try:
                        os.remove(item)
                    except:
                        pass

            ################################################
            ### VAIHE 3: MAASTOTIETOKANNAN HAKU JA PROSESSOINTI

            MTK_karttalehtijako = resolve('utm25LR.shp')

            #Karttalehdetattribuutteihin
            MTK_lehtitunnukset = processing.run("native:joinattributesbylocation", {'INPUT':selectedLayer,
                                                    'JOIN':MTK_karttalehtijako,
                                                    'PREDICATE':[1,2,4,5],
                                                    'JOIN_FIELDS':['LEHTITUNNU'],
                                                    'METHOD':0,'DISCARD_NONMATCHING':False,
                                                    'PREFIX':'',
                                                    'OUTPUT':'TEMPORARY_OUTPUT'})

            #Tee lista karttalehtien osoitteista ja nouda karttalehdet kapsin palvelimelta
            MTK_vlayer = MTK_lehtitunnukset['OUTPUT']

            MTK_tiedostot=[]
            MTK_karttalehdet=[]
            MTK_raportinOsoitteet=[]

            MTKjohto_tiedostot=[]
            MTKjohto_karttalehdet=[]

            MTKpylvaat_tiedostot = []
            MTKpylvaat_karttalehdet = []

            MTKrakenteet_tiedostot = []
            MTKrakenteet_karttalehdet = []

            for feature in MTK_vlayer.getFeatures():
                lehti1 = feature["LEHTITUNNU"]
                l1 = lehti1[0:2] + '/'
                le1=lehti1[0:3] + '/'
                zip_base ='/vsicurl/http://kartat.kapsi.fi/files/maastotietokanta/kaikki/etrs89/shp/'
                +l1 + le1 + lehti1 + '.shp.zip'
                MTK_raportinOsoite='http://kartat.kapsi.fi/files/maastotietokanta/kaikki/etrs89/shp/'
                +l1 + le1 + lehti1 + '.shp.zip'

                #Rakennukset
                tiedostonimi = ('r_' + lehti1 + '_p')
                shp_path = zip_base +'|layername=' + tiedostonimi
                v1 = iface.addVectorLayer(shp_path,tiedostonimi,'ogr')

                MTK_tiedostot.append(shp_path)
                MTK_karttalehdet.append(tiedostonimi)
                MTK_raportinOsoitteet.append(MTK_raportinOsoite)
```

```
578                     #Johdot
579                     tiedostonimiJohdot = ('j ' + lehti1 + ' v')
580                     shp_path = zip_base +'|layername=' + tiedostonimiJohdot
581                     v1 = iface.addVectorLayer(shp_path,tiedostonimiJohdot,'ogr')
582
583                     MTKjohto_tiedostot.append(shp_path)
584                     MTK_karttalehdet.append(tiedostonimiJohdot)
585
586                     #Pylväät
587                     tiedostonimiPylvaat = ('j_' + lehti1 + '_s')
588                     shp_path = zip_base + '|layername=' + tiedostonimiPylvaat
589                     v1 = iface.addVectorLayer(shp_path,tiedostonimiPylvaat,'ogr')
590
591                     MTKpylvaat_tiedostot.append(shp_path)
592                     MTK_karttalehdet.append(tiedostonimiPylvaat)
593
594                     #Rakenteet
595                     tiedostonimiRakenteet = ('r_' + lehti1 + '_s')
596                     shp_path = zip_base + '|layername=' + tiedostonimiRakenteet
597                     v1 = iface.addVectorLayer(shp_path,tiedostonimiRakenteet,'ogr')
598
599                     MTKrakenteet_tiedostot.append(shp_path)
600                     MTK_karttalehdet.append(tiedostonimiRakenteet)
601
602                 ###### Rakennukset ######
603
604                 #Yhdistäminen MTK
605
606                 MTK_yhdistetty = processing.run("native:mergevectorlayers", {'LAYERS':MTK_tiedostot,
607                                                                              'CRS':QgsCoordinateReferenceSyste
                                                                             m('EPSG:3067'),
608                                                                              'OUTPUT':'TEMPORARY_OUTPUT'})
609
610                 MTK_vlayer_yhd = MTK_yhdistetty['OUTPUT']
611
612                 #Rajaaminen
613
614                 MTK_rajattu = processing.run("native:clip", {'INPUT':MTK_vlayer_yhd,
615                                              'OVERLAY':selectedLayer,
616                                              'OUTPUT': 'TEMPORARY_OUTPUT'})
617
618                 #Väliaikainen output, koska tämä menee vielä 3D-Maastotietokannan käsittelyyn.
619                 #Ei siis lopullinen tiedosto vielä.
620
621                 MTK_vlayer_rajattu = MTK_rajattu['OUTPUT']
622
623                 ##### Johdot #####
624
625                 #Yhdistäminen
626
627                 MTKjohto_yhdistetty = processing.run("native:mergevectorlayers", {'LAYERS':MTKjohto_tiedostot,
628                                                                              'CRS':QgsCoordinateReferenceSyste
                                                                             m('EPSG:3067'),
629                                                                              'OUTPUT':'TEMPORARY_OUTPUT'})
630
631                 MTKjohto_vlayer_yhd = MTKjohto_yhdistetty['OUTPUT']
632
633                 #Rajaaminen
634
635                 MTKjohto_rajattu = processing.run("native:clip", {'INPUT':MTKjohto_vlayer_yhd,
636                                              'OVERLAY':selectedLayer,
637                                              'OUTPUT': 'TEMPORARY_OUTPUT'})
638
639
640                 MTKjohto_rajattu = processing.run("native:savefeatures", {'INPUT':
                    MTKjohto_rajattu['OUTPUT'],
641                                                                           'LAYER_NAME': 'Johdot',
642                                                                           'OUTPUT': 'TEMPORARY_OUTPUT'})
643
644                 #Rajatut johdot projektiin
645                 MTKjohdot = QgsVectorLayer(MTKjohto_rajattu['OUTPUT'],'Johdot','ogr')
646
647                 ##### Pylväät #####
648
649                 #Yhdistäminen
650
651                 MTKpylvaat_yhdistetty = processing.run("native:mergevectorlayers",
                    {'LAYERS':MTKpylvaat_tiedostot,
652                                                                                   'CRS':QgsCoordinateReferen
                                                                                  ceSystem('EPSG:3067'),
653                                                                                   'OUTPUT':'TEMPORARY_OUTPUT
                                                                                  '})
654
655                 MTKpylvaat_vlayer_yhd = MTKpylvaat_yhdistetty['OUTPUT']
656
657                 #Rajaaminen
```

```
658
659                 MTKpylvaat rajattu = processing.run("native:clip", {'INPUT':MTKpylvaat vlayer yhd,
660                                                                     'OVERLAY':selectedLayer,
661                                                                     'OUTPUT': 'TEMPORARY_OUTPUT'})
662
663
664                 MTKpylvaat_rajattu = processing.run("native:savefeatures", {'INPUT':
                    MTKpylvaat_rajattu['OUTPUT'],
665                                                                         'LAYER_NAME': 'Pylvaat',
666                                                                         'OUTPUT': 'TEMPORARY_OUTPUT'})
667
668                 #Rajatut johdot projektiin
669                 MTKpylvaat = QgsVectorLayer(MTKpylvaat_rajattu['OUTPUT'],'Pylvaat','ogr')
670
671
672                 ##### Rakenteet #####
673
674
675                 #Yhdistäminen
676
677                 MTKrakenteet_yhdistetty = processing.run("native:mergevectorlayers",
                    {'LAYERS':MTKrakenteet_tiedostot,
678
                                                                                    'CRS':QgsCoordinateRe
                                                                                    ferenceSystem('EPSG:3
                                                                                    067'),
679
                                                                                    'OUTPUT':'TEMPORARY_O
                                                                                    UTPUT'})
680
681                 MTKrakenteet_vlayer_yhd = MTKrakenteet_yhdistetty['OUTPUT']
682
683                 #Rajaaminen
684
685                 MTKrakenteet_rajattu = processing.run("native:clip", {'INPUT':MTKrakenteet_vlayer_yhd,
686                                                                       'OVERLAY':selectedLayer,
687                                                                       'OUTPUT': 'TEMPORARY_OUTPUT'})
688
689
690                 MTKrakenteet_rajattu = processing.run("native:savefeatures", {'INPUT':
                    MTKrakenteet_rajattu['OUTPUT'],
691                                                                           'LAYER_NAME': 'Rakenteet',
692                                                                           'OUTPUT':
                                                                           'TEMPORARY_OUTPUT'})
693
694                 #Rajatut rakenteet projektiin
695                 MTKrakenteet = QgsVectorLayer(MTKrakenteet_rajattu['OUTPUT'],'Rakenteet','ogr')
696
697
698
699                 #Poistetaan ylimääräiset tasot
700
701                 for lyr in MTK_karttalehdet:
702                     rmvLyr(lyr)
703
704                 ##############################################
705                 ### VAIHE 4: PISTEPILVIEN HAKU JA PROSESSOINTI
706
707                 #Tehdään pistepilvien karttalehdille oma lista
708
709                 pp_karttalehdet =[]
710
711                 #Ratkaistaan pluginin mukana tulleen utm5.shp tiedoston osoite.
712                 #Tiedosto sisältää karttalehtijaon, josta saadaan tarvittavat karttalehdet
713
714                 ppkarttalehtijako=resolve('utm5.shp')
715
716                 #Karttalehdetattribuutteihin
717                 pcl = processing.run("native:joinattributesbylocation", {'INPUT':selectedLayer,
718                                                                          'JOIN':ppkarttalehtijako,
719                                                                          'PREDICATE':[1,2,4,5],
720                                                                          'JOIN_FIELDS':['LEHTITUNNU'],
721                                                                          'METHOD':0,'DISCARD_NONMATCHING':False,
722                                                                          'PREFIX':'',
723                                                                          'OUTPUT':'TEMPORARY_OUTPUT'})
724
725                 #Haetaan tarvittavat karttalehdet attribuuteista ja haetaan kapsin palvelimelta
726                 pclvLayer=pcl['OUTPUT']
727
728                 for feature in pclvLayer.getFeatures():
729                     pp_lehti = feature["LEHTITUNNU"]
730                     pp_karttalehdet.append(pp_lehti)
731
732                 #tehdään eri listat raporttia ja käsittelyjä varten
733
734                 pp_raportinOsoitteet=[]
735                 pptiedostot=[]
736                 ppKoneenOsoitteet=[]
737
738                 #Testataan mistä kansiosta pistepilvet löytyvät Kapsin palvelimelta
```

```
739
740                 for lehti in pp karttalehdet:
741                     pp karttalehtikansio = '/' + lehti[0:4]
742
743                     #Kapsin palvelimella tiedostot jaetaan vuosittain kansioihin, käydään läpi vuodet
                        2006-2039
744
745                     vuodet = []
746                     for i in range(2006,2036):
747                         vuodet.append(i)
748
749                     for item in vuodet:
750                         item_str=str(item)
751
752                         try:
753                             if urllib.request.urlopen
                              ('http://kartat.kapsi.fi/files/laser/etrs-tm35fin-n2000/mara_2m/' + item_str +
                              pp_karttalehtikansio + '/1/' + lehti + '.laz').code == 200:
754                                 pp_oikea_osoite =
                                  ('http://kartat.kapsi.fi/files/laser/etrs-tm35fin-n2000/mara_2m/' + item_str
                                  + pp_karttalehtikansio + '/1/' + lehti + '.laz')
755                         except:
756                             pass
757
758                     #Kun saadaan oikea osoite, ladataan se
759
760                     processing.run("native:filedownloader", {'URL':pp_oikea_osoite,
761                                                              'OUTPUT':ppoutputdir + '/' + lehti + '.laz' })
762
763                     #Tämä menee JSON tiedostoon pistepilven käsittelyyn
764                     pptiedosto='    "'+ ppoutputdir +'/' + lehti + '.laz",\n'
765                     pptiedostot.append(pptiedosto)
766
767                     #Tämä kerätään väliaikaisten tiedostojen poistoa varten
768                     ppKoneenOsoite= ppoutputdir + '/' + lehti + '.laz'
769                     ppKoneenOsoitteet.append(ppKoneenOsoite)
770
771                     #Tämä kerätään raporttia varten
772                     pp_raportinOsoitteet.append(pp_oikea_osoite)
773
774
775                 #Käsittely: Tee PDAL -pipeline merge
776
777                 path1 = os.path.dirname(QgsApplication.prefixPath())
778                 path2 = os.path.dirname(path1)
779                 OsGeo4Wpath = path2 + '/OSGeo4W.bat'
780
781                 file1 = open(ppoutputdir + '/pdalpipeline_merge.json','w')
782                 file1.write('[\n')
783                 file1.writelines(pptiedostot)
784                 file1.write('    {\n')
785                 file1.write('        "type": "filters.merge"\n')
786                 file1.write('    },\n')
787                 file1.write('    "' +ppoutputdir+ '/PP_yhdistetty.laz"\n')
788                 file1.write(']')
789                 file1.close()
790
791                 #Suorita PDAL -pipeline:merge
792
793                 my_call = [OsGeo4Wpath,                     #Inspired
794                     'pdal',                                 #
795                     'pipeline',                             #
796                     ppoutputdir+'/pdalpipeline_merge.json', #
797                     '--verbose',                            #by
798                     '4'                                     #
799                 ]                                           #
800                                                             #
801                 subprocess.call(my_call)
                    #https://gis.stackexchange.com/questions/274828/use-osgeo4w-shell-command-from-python-2-7-scri
                    pt
802
803                 time.sleep(5)
804
805                 #########
806                 #Leikkaus
807
808                 #Muuttaa geometriat WKT-tyypiksi ja tekee listan json muotoilulla
809
810                 selectedLayerOld = processing.run("native:reprojectlayer", {'INPUT': selectedLayer,
811                                                                             'TARGET_CRS':
                                                                              QgsCoordinateReferenceSystem('EPSG
                                                                              :3067'),
812                                                                             'OUTPUT': 'TEMPORARY_OUTPUT'})
813
814                 selectedLayerNew = selectedLayerOld['OUTPUT']
815
816                 geometriat=[]
817                 for feat in selectedLayerNew.getFeatures():
818                     geometriat.append(feat.geometry().asWkt())
819                 jsongeometriat = (json.dumps(geometriat))
```

```
820
821                 #Tee JSON Pipeline: Crop
822
823                 file1 = open(ppoutputdir + '/pdalpipeline_crop.json','w')
824                 file1.write('[\n')
825                 file1.write('    "' + ppoutputdir + '/PP_yhdistetty.laz",\n')
826                 file1.write('    {\n')
827                 file1.write('        "type":"filters.crop",\n')
828                 file1.write('        "polygon":' + jsongeometriat +'\n')
829                 file1.write('    },\n')
830                 file1.write('    {\n')
831                 file1.write('        "type":"filters.outlier",\n')
832                 file1.write('        "method":"radius",\n')
833                 file1.write('        "radius":3.0,\n')
834                 file1.write('        "min_k":4\n')
835                 file1.write('    },\n')
836                 file1.write('    {\n')
837                 file1.write('        "type":"filters.range",\n')
838                 file1.write('        "limits":"Classification![7:7]"\n')
839                 file1.write('    },\n')
840                 file1.write('    {\n')
841                 file1.write('        "type":"writers.las",\n')
842                 file1.write('        "filename":"' + ppoutputdir + '/PP_rajattu.laz"\n')
843                 file1.write('    }\n')
844                 file1.write(']')
845                 file1.close()
846
847                 #Suorita JSON Pipeline: Crop
848
849                 my_call = [OsGeo4Wpath,                     #Inspired
850                     'pdal',                                 #
851                     'pipeline',                             #
852                     ppoutputdir+'/pdalpipeline_crop.json',  #
853                     '--verbose',                            #by
854                     '4'                                     #
855                 ]                                           #
856                                                             #
857                 subprocess.call(my_call)
                    #https://gis.stackexchange.com/questions/274828/use-osgeo4w-shell-command-from-python-2-7-scri
                    pt
858
859                 time.sleep(5)
860
861                 #Tee JSON Pipeline: Pointcloud -> DSM
862
863                 file1 = open(ppoutputdir + '/pdalpipeline_dsm.json','w')
864                 file1.write('[\n')
865                 file1.write('    "' + ppoutputdir + '/PP_rajattu.laz",\n')
866                 file1.write('    {\n')
867                 file1.write('        "filename":"' + ppoutputdir + '/PP_rajattu_DSM.tif",\n')
868                 file1.write('        "gdaldriver":"GTiff",\n')
869                 file1.write('        "output_type":"all",\n')
870                 file1.write('        "resolution":"1.0",\n')
871                 file1.write('        "type": "writers.gdal"\n')
872                 file1.write('    }\n')
873                 file1.write(']')
874                 file1.close()
875
876                 #Suorita JSON Pipeline: DSM
877
878                 my_call = [OsGeo4Wpath,                   #Inspired
879                     'pdal',                               #
880                     'pipeline',                           #
881                     ppoutputdir+'/pdalpipeline_dsm.json', #
882                     '--verbose',                          #by
883                     '4'                                   #
884                 ]                                         #
885                                                           #
886                 subprocess.call(my_call)
                    #https://gis.stackexchange.com/questions/274828/use-osgeo4w-shell-command-from-python-2-7-scri
                    pt
887
888                 #Odota hetki ja tuo tehty DSM projektiin
889
890                 time.sleep(5)
891
892                 dsm = QgsRasterLayer(ppoutputdir + '/PP_rajattu_DSM.tif',outputdirname + ' DSM','gdal')
893
894                 #Tee JSON Pipeline: Pistepilven värjäys ja puuston ja siltojen erilleen otto
895
896                 file1 = open(ppoutputdir + '/pdalpipeline_colorize.json','w')
897                 file1.write('{\n')
898                 file1.write('    "pipeline": [\n')
899                 file1.write('        {\n')
900                 file1.write('            "filename":"' + ppoutputdir + '/PP_rajattu.laz",\n')
901                 file1.write('            "type":"readers.las",\n')
902                 file1.write('            "spatialreference":"EPSG:3067"\n')
903                 file1.write('        },\n')
904                 file1.write('        {\n')
905                 file1.write('            "type": "filters.colorization",\n')
```

```
                file1.write('            "raster": "' + outputdir + '/' + outputdirname + '
Ortoilmakuva.tif"\n')
                file1.write('        },\n')
                file1.write('    {\n')
                file1.write('            "type": "filters.range",\n')
                file1.write('            "limits": "Red[1:]"\n')
                file1.write('        },\n')
                file1.write('        {\n')
                file1.write('            "type": "writers.las",\n')
                file1.write('            "compression": "true",\n')
                file1.write('            "minor_version": "2",\n')
                file1.write('            "dataformat_id": "3",\n')
                file1.write('            "filename":"' + ppoutputdir + '/PP_varjatty.laz"\n')
                file1.write('        },\n')
                file1.write('        {\n')
                file1.write('            "type":"filters.range",\n')
                file1.write('
                "limits":"Classification[3:5],Classification[10:10],Classification[15:15]"\n')
                file1.write('        },\n')
                file1.write('        {\n')
                file1.write('            "type":"writers.las",\n')
                file1.write('            "compression": "true",\n')
                file1.write('            "minor_version": "2",\n')
                file1.write('            "dataformat_id": "3",\n')
                file1.write('            "filename":"' + ppoutputdir + '/PP_varjatty_puusto.laz"\n')
                file1.write('        }\n')
                file1.write('    ]\n')
                file1.write('}')
                file1.close()

                #Suorita JSON Pipeline: Pistepilven värjäys

                my_call = [OsGeo4Wpath,                                  #Inspired
                    'pdal',                                              #
                    'pipeline',                                          #
                    ppoutputdir+'/pdalpipeline_colorize.json',           #
                    '--verbose',                                         #by
                    '3'                                                  #
                ]                                                        #
                                                                         #
                subprocess.call(my_call)
                #https://gis.stackexchange.com/questions/274828/use-osgeo4w-shell-command-from-python-2-7-scri
                pt

                time.sleep(5)

                #Värjätty pistepilvi kartalle

                iface.addPointCloudLayer(ppoutputdir + '/PP_varjatty_puusto.laz',outputdirname + '
                Pistepilvi puusto','pdal')
                iface.addPointCloudLayer(ppoutputdir + '/PP_varjatty.laz',outputdirname + ' Pistepilvi
                kaikki','pdal')


                ##############################################################################
                ### VAIHE 5: Maastotietokannan käsittely DSM:n perusteella 3D-Maastotietokannaksi

                #Käsittely: maanpinnan korkeus rakennuksen attribuutiksi, statistics [5] on minimi

                zonalDem = processing.run("native:zonalstatisticsfb", {'INPUT':MTK_vlayer_rajattu,
                                                        'INPUT_RASTER':rajattu,
                                                        'RASTER_BAND':1,
                                                        'COLUMN_PREFIX':'zMaa_',
                                                        'STATISTICS':[5],
                                                        'OUTPUT':'TEMPORARY_OUTPUT'})

                #Käsittely: maanpinnan korkeus attribuutista z-koordinaateiksi

                setZvalue = processing.run("native:setzvalue", {'INPUT':zonalDem['OUTPUT'],
                                                'Z_VALUE':QgsProperty.fromExpression('"zMaa_min"'),
                                                'OUTPUT':'TEMPORARY_OUTPUT'})

                #Käsittely: Negatiivinen buffer. Pyrkimyksenä pienentää rakennuksen monikulmiota sisälle päin,
                #jotta rakennuksen monikulmiolle ei sattuisi puuston pisteitä

                Mtk3Dtemp = processing.run("native:buffer", {'INPUT':setZvalue['OUTPUT'],
                                                'DISTANCE':-2.0,
                                                'SEGMENTS':1,
                                                'END_CAP_STYLE':1,
                                                'JOIN_STYLE':1,
                                                'MITER_LIMIT':2,
                                                'DISSOLVE':False,
                                                'OUTPUT':'TEMPORARY_OUTPUT'})


                #Käsittely: Korkeus DSM:stä buffertason rakennuksen katon korkeus-attribuutiksi,
                #rasterband 1 on minimikanava, statistics2 on keskiarvo, 6 on maksimi

                Mtk3Dtemp = processing.run("native:zonalstatisticsfb", {'INPUT':Mtk3Dtemp['OUTPUT'],
                                                        'INPUT_RASTER':dsm,
```

```
                                                    'RASTER BAND':1,
                                                    'COLUMN PREFIX':'zKat ',
                                                    'STATISTICS':[6],
                                                    'OUTPUT':'TEMPORARY_OUTPUT'})

            #Tuodaan tehty taso Mtk3Dtemp muuttujaan

            Mtk3Dtemp = Mtk3Dtemp['OUTPUT']

            #Lisätään Mtk3Dtemp tasolle Extrusion kenttä

            pr = Mtk3Dtemp.dataProvider()                                          #Inspired
            pr.addAttributes([QgsField("Extrusion", QVariant.Double,"double",11,3)])    #by
            Mtk3Dtemp.updateFields()
            #https://anitagraser.com/pyqgis-101-introduction-to-qgis-python-programming-for-non-programmer
            s/pyqgis-101-using-expressions-to-compute-new-field-values/

            #Lasketaan extrusion kentälle uudet arvot käyttämällä QGIS-Expressionia, jos dsm laskennan
            statistiikka muuttuu, muuta zKat_ etuliitteen jälkeinen statistiikan nimi

            expression1 = QgsExpression('if("zKat_max" - "zMaa_min" <0,0,round("zKat_max" -
            "zMaa_min",3))') #Inspired
            context =
            QgsExpressionContext()                                                 #

            context.appendScopes(QgsExpressionContextUtils.globalProjectLayerScopes(Mtk3Dtemp))
               #

#by
            with
            edit(Mtk3Dtemp):                                                       #
                for f in
                Mtk3Dtemp.getFeatures():                                           #

                    context.setFeature(f)
                       #
                    f['Extrusion'] =
                    expression1.evaluate(context)                                  #

                    Mtk3Dtemp.updateFeature(f)

                    #https://anitagraser.com/pyqgis-101-introduction-to-qgis-python-programming-for-non-pr
                    ogrammers/pyqgis-101-using-expressions-to-compute-new-field-values/

            #Join Attributes alkuperäiseen MTK -geometriaan, jossa maan Z-koordinaatit

            MTK3D = processing.run("native:joinattributestable", {'INPUT':setZvalue['OUTPUT'],
                                                    'FIELD':'KOHDEOSO',
                                                    'INPUT_2':Mtk3Dtemp,
                                                    'FIELD_2':'KOHDEOSO',
                                                    'FIELDS_TO_COPY':['Extrusion'],
                                                    'METHOD':1,
                                                    'DISCARD_NONMATCHING':False,
                                                    'PREFIX':'',
                                                    'OUTPUT':'TEMPORARY_OUTPUT'})

            MTK3D = MTK3D['OUTPUT']

            #Tallenetaan tason nimi geopaketin tallennnusta varten

            MTK3D = processing.run("native:savefeatures", {'INPUT': MTK3D,
                                                    'LAYER_NAME': 'Rakennukset',
                                                    'OUTPUT': 'TEMPORARY_OUTPUT'})

            #3D rakennukset projektiin
            MTK3D = QgsVectorLayer(MTK3D['OUTPUT'],'Rakennukset','ogr')


            #Tehdään Geopackage

            outputgeopackage = outputdir + '/Maastotietokanta.gpkg'

            processing.run("native:package",{'LAYERS':[MTK3D,MTKjohdot,MTKpylvaat,MTKrakenteet],
                            'OUTPUT':outputgeopackage,
                            'OVERWRITE':True,
                            'SAVE_STYLES':False})

            #Johdot, Pylväät ja rakennukset geopaketista kartalle

            johdot3d = QgsVectorLayer(outputgeopackage + '|layername=Johdot',outputdirname + '
            Johdot','ogr')
            if johdot3d.isValid():
                QgsProject.instance().addMapLayer(johdot3d)

                #Ladataan tyylitiedosto
                johdot3d.loadNamedStyle(resolve('johdot_tyyli.qml'))
                johdot3d.triggerRepaint()

            pylvaat3d = QgsVectorLayer(outputgeopackage + '|layername=Pylvaat',outputdirname + '
            Pylvaat','ogr')
```

```
1061                    if pylvaat3d.isValid():
1062                        QgsProject.instance().addMapLayer(pylvaat3d)
1063
1064                        #Ladataan tyylitiedosto
1065                        pylvaat3d.loadNamedStyle(resolve('pylvaat_tyyli.qml'))
1066                        pylvaat3d.triggerRepaint()
1067
1068                    MTK3D = QgsVectorLayer(outputgeopackage + '|layername=Rakennukset',outputdirname + '
                        Rakennukset','ogr')
1069                    if MTK3D.isValid():
1070                        QgsProject.instance().addMapLayer(MTK3D)
1071
1072                        #Ladataan tyylitiedosto
1073                        MTK3D.loadNamedStyle(resolve('tyyli.qml'))
1074                        MTK3D.triggerRepaint()
1075
1076                    rakenteet3d = QgsVectorLayer(outputgeopackage + '|layername=Rakenteet',outputdirname + '
                        Rakenteet','ogr')
1077                    if rakenteet3d.isValid():
1078                        QgsProject.instance().addMapLayer(rakenteet3d)
1079
1080                        #Ladataan tyylitiedosto
1081                        rakenteet3d.loadNamedStyle(resolve('rakenteet_tyyli.qml'))
1082                        rakenteet3d.triggerRepaint()
1083
1084                    ##############################################################################
1085                    ### VAIHE 6: Ylimääräisten tiedostojen poisto
1086
1087                    file1 = ppoutputdir + '/pdalpipeline_colorize.json'
1088                    file2 = ppoutputdir + '/pdalpipeline_crop.json'
1089                    file3 = ppoutputdir + '/pdalpipeline_dsm.json'
1090                    file4 = ppoutputdir + '/pdalpipeline_merge.json'
1091                    file5 = ppoutputdir + '/PP_rajattu.laz'
1092                    file6 = ppoutputdir + '/PP_yhdistetty.laz'
1093                    file7 = outputdir + '/' + outputdirname + ' Ortoilmakuva.tif'
1094
1095                    poistettavat=[file1,file2,file3,file4,file5,file6,file7]
1096
1097                    for item in poistettavat:
1098                        if os.path.isfile(item):
1099                            try:
1100                                os.remove(item)
1101                            except:
1102                                pass
1103
1104                    #Poistetaan ylimääräiset pistepilvikarttalehdet
1105
1106                    for item in ppKoneenOsoitteet:
1107                        if os.path.isfile(item):
1108                            try:
1109                                os.remove(item)
1110                            except:
1111                                pass
1112
1113                    ##############################################################################
1114                    ### VAIHE 6: Raportin kirjoitus
1115
1116                    file1 = open(outputdir + '/Raportti.txt','w', encoding='iso-8859-1')
1117
1118                    file1.write('####### - KAUPUNKIMALLINTAJA - ####### \n \n')
1119                    file1.write('###################################### \n \n')
1120                    file1.write('Kaupunkimallintaja muodosti tiedostot: \n \n')
1121
1122                    file1.write('Korkeusmalli: \n \n')
1123                    if os.path.isfile(outputdir + '/Korkeusmalli.gpkg'):
1124                        file1.write(outputdir + '/Korkeusmalli.gpkg \n \n')
1125
1126                    file1.write('Ortoilmakuva: \n \n')
1127                    if os.path.isfile(outputdir + '/Ortoilmakuva.gpkg'):
1128                        file1.write(outputdir + '/Ortoilmakuva.gpkg \n \n')
1129
1130                    file1.write('Maastotietokannan aineistot: \n \n')
1131                    if os.path.isfile(outputdir + '/Maastotietokanta.gpkg'):
1132                        file1.write(outputdir + '/Maastotietokanta.gpkg \n \n')
1133
1134                    file1.write('LAZ - pistepilvet: \n \n')
1135                    if os.path.isfile(ppoutputdir + '/PP_varjatty.laz'):
1136                        file1.write(ppoutputdir + '/PP_varjatty.laz \n')
1137
1138                    if os.path.isfile(ppoutputdir + '/PP_varjatty_puusto.laz'):
1139                        file1.write(ppoutputdir + '/PP_varjatty_puusto.laz \n \n')
1140
1141                    file1.write('DSM eli Pintamalli: \n \n')
1142
1143                    if os.path.isfile(ppoutputdir + '/PP_rajattu_DSM.tif'):
1144                        file1.write(ppoutputdir + '/PP_rajattu_DSM.tif \n \n')
1145
1146                    file1.write('Entwine - pistepilvet: \n \n')
1147                    if os.path.isdir(ppoutputdir + '/ept_PP_varjatty_puusto'):
1148                        file1.write(ppoutputdir + '/ept_PP_varjatty_puusto \n')
```

```
1149
1150                    #if os.path.isdir(ppoutputdir + '/ept PP varjatty'):
1151                    file1.write(ppoutputdir + '/ept PP varjatty \n \n')
1152
1153                    file1.write('\n')
1154                    file1.write('################################################## \n \n')
1155                    file1.write('Käytettiin Kapsin palvelimelta haettuja tiedostoja:\n')
1156                    file1.write('\n')
1157                    file1.write('Korkeusmallit:\n')
1158                    file1.write('\n')
1159
1160                    for item in raportinOsoitteet:
1161                        file1.write(item + '\n')
1162
1163                    file1.write('\n')
1164                    file1.write('Maastotietokanta:\n')
1165                    file1.write('\n')
1166
1167                    for item in MTK_raportinOsoitteet:
1168                        file1.write(item + '\n')
1169
1170                    file1.write('\n')
1171                    file1.write('Ortoilmakuvat:\n')
1172                    file1.write('\n')
1173
1174                    for item in Orto_raportinOsoitteet:
1175                        file1.write(item +'\n')
1176
1177                    file1.write('\n')
1178                    file1.write('Pistepilvet:\n')
1179                    file1.write('\n')
1180
1181                    for item in pp_raportinOsoitteet:
1182                        file1.write(item +'\n')
1183
1184                    file1.write('\n')
1185
1186                    file1.close()
1187
1188                #Palautetaan filtterit. Jokaisesta monikulmiosta tehdään oma mallinsa omaan kansioon
                    filtteröimällä valitun tason attribuutit kukin kerrallaan.
1189                #Monikulmioiden läpikäynnin jälkeen filtterit palautetaan niin, että rajaustiedostossa näkyy
                    kaikki monikulmiot.
1190
1191                selectedLayer.setSubsetString('')
1192
1193                iface.mapCanvas().refresh()
1194
1195                #Onnistumisen ilmoitus
1196
1197                self.iface.messageBar().pushMessage(
1198                   "Haku kapsin palvelimelta onnistui! ",
1199                  level=Qgis.Success, duration=20)
1200
1201
1202
```

####### - KAUPUNKIMALLINTAJA - #######

######################################

Kaupunkimallintaja muodosti tiedostot:

Korkeusmalli:

E:/Kaupunkimallit/Iisalmi/1/Korkeusmalli.gpkg

Ortoilmakuva:

E:/Kaupunkimallit/Iisalmi/1/Ortoilmakuva.gpkg

Maastotietokannan aineistot:

LAZ - pistepilvet:

E:/Kaupunkimallit/Iisalmi/1/Pistepilvi/PP_varjatty.laz
E:/Kaupunkimallit/Iisalmi/1/Pistepilvi/PP_varjatty_puusto.laz

DSM eli Pintamalli:

E:/Kaupunkimallit/Iisalmi/1/Pistepilvi/PP_rajattu_DSM.tif

Entwine - pistepilvet:

E:/Kaupunkimallit/Iisalmi/1/Pistepilvi/ept_PP_varjatty_puusto
E:/Kaupunkimallit/Iisalmi/1/Pistepilvi/ept_PP_varjatty

##

Käytettiin Kapsin palvelimelta haettuja tiedostoja:

Korkeusmallit:

http://kartat.kapsi.fi/files/korkeusmalli/hila_2m/etrs-tm35fin-n2000/P5/P52/P5222D.tif

Maastotietokanta:

http://kartat.kapsi.fi/files/maastotietokanta/kaikki/etrs89/shp/P5/P52/P5222L.shp.zip

Ortoilmakuvat:

http://kartat.kapsi.fi/files/orto/etrs-tm35fin/smk_v_15000_50/2020/P52/02m/1/P5222D.jp2

Pistepilvet:

http://kartat.kapsi.fi/files/laser/etrs-tm35fin-n2000/mara_2m/2012/P522/1/P5222D2.laz

http://kartat.kapsi.fi/files/laser/etrs-tm35fin-n2000/mara_2m/2012/P522/1/P5222D4.laz
http://kartat.kapsi.fi/files/laser/etrs-tm35fin-n2000/mara_2m/2012/P522/1/P5222D1.laz
http://kartat.kapsi.fi/files/laser/etrs-tm35fin-n2000/mara_2m/2012/P522/1/P5222D3.laz