jamk.fi

Metsän kehitysluokkien tunnistus lento-konekeilausaineistojen perusteella Python-ohjelmointina

Hannu Sirén

Opinnäytetyö
Elokuu 2015

Ohjelmistotekniikan koulutusohjelma
Tekniikan ja liikenteen ala

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

KUVAILULEHTI

Tekijä(t) Hannu Sirén	Julkaisun laji Opinnäytetyö	Päivämäärä 16.8.2015
	Sivumäärä 44	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty (X)

Työn nimi
Metsän kehitysluokkien tunnistus lentokonekeilausaineistojen perusteella Python-ohjelmointina

Koulutusohjelma
Ohjelmistotekniikan koulutusohjelma

Työn ohjaaja(t)
Kalevi Pietikäinen, Matti Mieskolainen

Toimeksiantaja(t)
Suomen metsäkeskus

Tiivistelmä
Opinnäytetyön aiheena on suunnitella ja toteuttaa uusia ominaisuuksia, metsän kehitysluokkien tunnistukseen tehtyyn ohjelmistosovellukseen. Aihe on osa laajempaa ”Metsävaramittaus kunnostamattomissa metsissä” projektia.

Opinnäytetyö käsittelee lentokonekeilausaineistoissa käytettyä LAS-formaatin luku- ja kirjoitusprosessin integrointia aiemmin tehtyyn ohjelmistosovellukseen, joka on tehty Python-ohjelmistokielellä. Työssä käsitellään myös luku- ja kirjoitusprosessien keskusmuistinhallintaa. Lisäksi työ esittelee sekä Matplotlib-moduulilla aineistorajojen graafista esittämistä xml-tiedostosta, että aineiston esittämisen periaatteita ArcGIS-työkalulla.

Tuloksena suunniteltiin LAS-tiedostolle lukija, joka pystyy lukemaan minkä tahansa kokoisen tiedoston. Toinen tulos oli xml-tiedostosta Matplotlib-moduulia apuna käyttävä kuviorajojen piirto-ohjelmakoodi. Kolmas tulos oli selvitys ASCII- ja LAS- koodien luku- ja kirjoitusnopeuden eroista. Lisäksi perehdyttiin aineiston siirtämiseen ArcGIS-työkaluun.

Avainsanat (asiasanat)
LAS, LAZ, Lentokonekeilausaineisto, Matplotlib, ArcGIS, Python, Metsän kehitysluokat

Muut tiedot

JYVÄSKYLÄN AMMATTIKORKEAKOULU
JAMK UNIVERSITY OF APPLIED SCIENCES

DESCRIPTION

Author(s)	Type of publication	Date
Hannu Sirén	Bachelor´s Thesis	16.8.2015
	Pages 44	Language Finnish
		Permission for web publication (X)

Title
Forest growth categorization from laser point data using Python programming language.

Degree Programme
Degree Programme in Software Engineering

Tutor(s)
Kalevi Pietikäinen, Matti Mieskolainen

Assigned by
Suomen metsäkeskus

Abstract

The subject of the thesis was the design and implementation of new features to a software the purpose of which is to identify categorized growth in forests. The thesis is part of a larger project called Forest reserve measurement in unmend forests.

The thesis includes design and implementation of a reader and writer software for files in LAS format, which originate from point data collected by scanning an area using an airplane equipped with special scanning instruments. The implementation of the identification software has been written in Python software language. The application handles the RAM consumption as part of the design.

The thesis illustrates a graphical presentation of specified areas using the Matplotlib module. The areas are stored in an XML file. The thesis compares read and write speeds in both ASCII and LAS standard. By handling the RAM consumption the design is able to load LAS file in all possible sizes. Additionally, the basics of presenting the categorized forest growth in ArcGIS software have been documented.

Keywords
LAS, LAZ, Laser point data, Matplotlib, ArcGIS, Python, Categorized forest growth

Miscellaneous

Sisällysluettelo

Kuviot

Taulukot

1 Työn lähtökohdat

1.1 Tehtävän kuvaus

Opinnäytetyön tehtävänä oli suunnitella ja toteuttaa uusia ominaisuuksia, metsän kehitysluokkien tunnistukseen tehtyyn ohjelmistosovellukseen. Työssä tutustuttiin lentokonekeilausaineistoissa käytettyyn tiedostostandardiin ja sen lukemiseen sekä kirjoittamiseen. Lisäksi työssä suunniteltiin kehitysluokkien graafista esittämistä sovelluksen käyttäjälle.

Opinnäytetyön oli osa laajempaa ”Metsävaramittaus kunnostamattomissa metsissä” – projektia.

”Metsävaramittaus kunnostamattomissa metsissä” on Manner-Suomen maaseudun kehittämisohjelman (2007–2013) rahoittama kehittämishanke (2011–2014), jonka tarkoituksena on ollut metsävaratietojen mittaamisen ja hyödyntämisen kehittäminen erityisesti kunnostamattomissa metsissä energiapuukorjuun tarpeisiin. Projektin taustalla on vuonna 2010 Jyväskylän ammattikorkeakoulussa (JAMK) toteutettu esiselvitystyö, josta selkeästi ilmeni, että maastossa tehtävää koealamittausta tulee kehittää kaukokartoitusmenetelmien tueksi ja niitä täydentämään ja että tarvitaan riittävän tarkkaa, mutta kustannustehokasta tietoa käytännön metsänhoitotöiden suunnittelua ja toteuttamista varten. (Paananen, Pellinen, Pietikäinen, Sauranen, Savolainen, Viitala, Weckström & Väänänen, 2014, 11)

Opinnäytetyö tehtiin täysin ohjelmistosuunnittelun ja -tekniikan näkökulmasta, eikä siinä vaadittu esimerkiksi metsätalousosaamista.

1.2 Metsän kehitysluokka

Käsitteellä metsän kehitysluokka halutaan ilmaista kyseisellä mittaushetkellä vallinnutta metsänhoito- ja hakkuutarvetta. Kehitysluokkien tunnistussovellus määrittelee kehitysluokan puuston korkeuden mukaan. Lisäksi luokitukseen vaikuttaa puuston tiheys.

Ohjelmistotyö kohdistui erityisesti kolmeen kehitysluokkaan: pieni taimikko T1, varttunut taimikko T2 ja nuori kasvatusmetsikkö 02. Pienen taimikon keskipituus on enintään 1,3 metriä. Varttunut taimikko on keskipituudeltaan vähintään 1,3 metriä, männyllä ja kuusella alle 7 metriä sekä koivulla alle 9 metriä. Nuoressa kasvatusmetsikössä havupuut ovat vähintään 7 metriä korkeita ja koivikko vähintään 9 metriä (Puuston kehitysluokat, N.d.).

1.3 Toimeksiantaja

Tunnistussovelluksen kehitystyön asiakkaana on ollut Suomen metsäkeskus. Maa- ja metsätalousministeriön ohjaama metsäkeskus kerää ja jakaa tietoa Suomen metsistä sekä valvoo metsälainsäädän-

nön noudattamista. Metsäkeskus on asiantuntijatalo joka on erikoistunut metsä-, luonto-osaamiseen sekä metsäalan elinkeinojen tuntemukseen (Metsäkeskus - metsäalan asiantuntijatalo, N.d.).

1.4 Lentokonekeilausaineisto

Aineisto

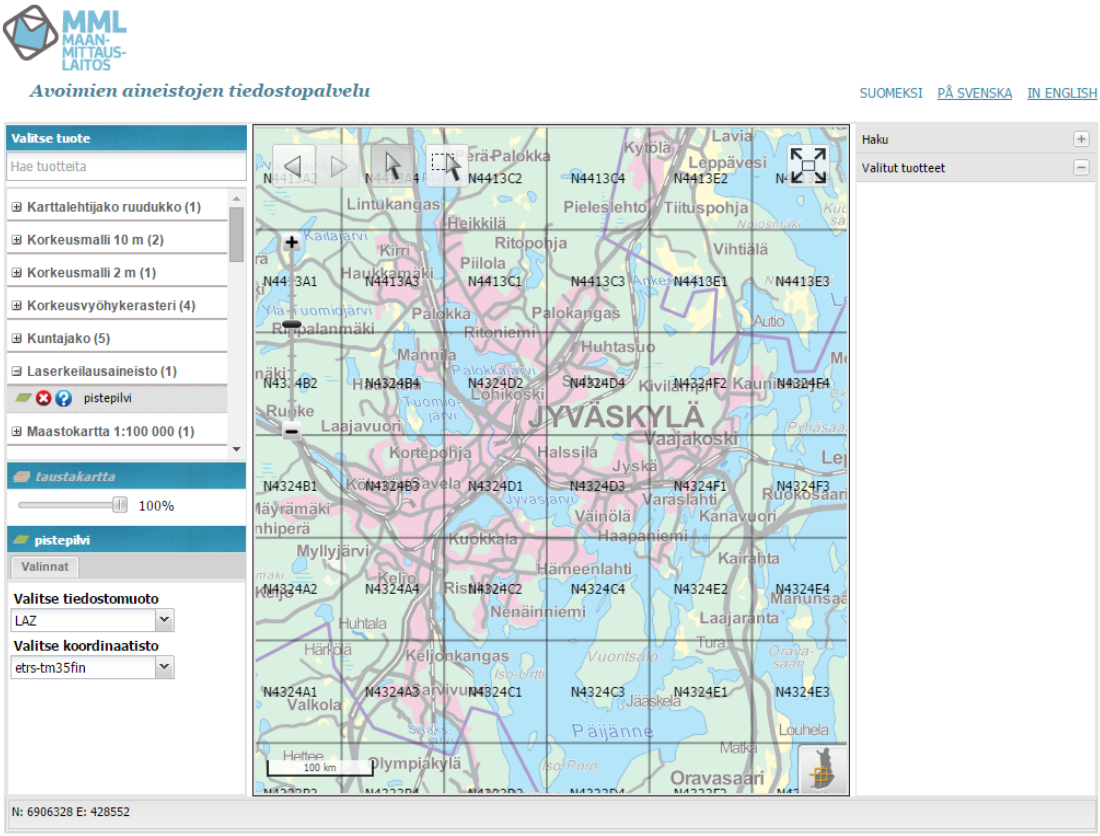
Lentokonekeilausaineisto koostuu kolmiulotteisen avaruuden pisteistä. Pisteet on kerätty lentokoneeseen asennetulla laserkeilaimella, josta ne on tallennettu tiedostoiksi.

Maanmittauslaitos avasi maastotietoaineistot 1.5.2012 julkiseen käyttöön. Tarkoituksena on viranomaistiedon antaminen kaikille siitä kiinnostuneilla pääministeri Jyrki Kataisen hallituksen tavoitteen mukaisesti. (Maastotietoaineistot koko kansan käyttöön, 2012). Aineisto on vapaasti ladattavissa Maanmittauslaitoksen tiedostopalvelusta.

Maanmittauslaitoksen tiedostopalvelu

Tiedostopalvelu on tietokoneen selaimella käytettävä palvelu Maanmittauslaitoksen verkkosivuilla. Palvelun käyttöön käyttäjä tarvitsee toimivan sähköpostiosoitteen.

Tiedostopalvelusta on saatavissa mm. laserkeilausaineistoa, maastokarttarastereita, peruskarttarastereita, maastotietokantoja, maastokarttoja, maastokarttarastereita, ortoilmakuvia, korkeusmalleja, korkeusvyöhykerastereita, vinovalovarjorastereita, taustakarttasarjoja, yleiskarttoja, yleiskarttarastereita, nimistöjä ja kuntajakoja (Avoimien aineistojen tiedostopalvelu, N.d.). Kuvankaappaus tiedostopalvelun käyttöliittymästä on esitetty kuviossa 1.



Kuvio 1. Maanmittauslaitoksen tiedostopalvelu

Metsän kehitysluokkien tunnistussovelluksessa käytettiin maanmittauslaitoksen keräämää laserkeilausaineistoa. Aineisto on ladattavissa tiedostopalvelusta pakatussa muodossa.

1.5 Laserkeilausaineisto

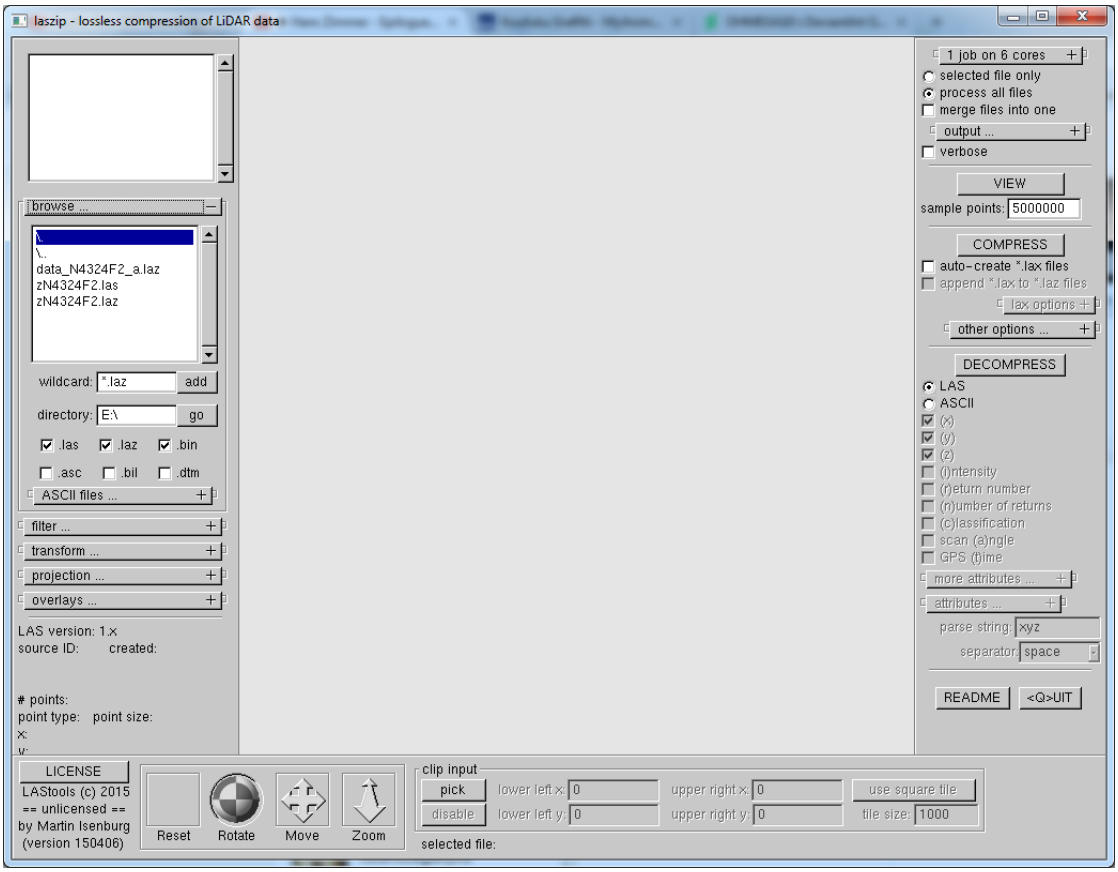
Laserkeilausaineistoa kerätään lentokoneeseen asennetulla laserkeilaimella. Laserkeilain lähettää laserpulsseja, jotka osuttuaan fyysiseen objektiin heijastuvat takaisin lentokoneeseen asennettuun vastaanottimeen (Laserkeilaustekniikka, N.d.). Kun tiedetään lentokoneen koordinaatit, korkeus, kallistuskulma sekä tarkka aikaväli laserpulssin lähetyksen ja vastaanottimeen heijastumisen välillä, voidaan laskea pulssin osuman X-, Y- ja Z-koordinaatit.

Kun lentokoneeseen asennetulla laserkeilaimella haravoidaan haluttu metsäalue, saadaan kerättyä alueen laserkeilausaineisto. Aineisto sisältää kaikki kohteet, joista laserpulssi heijastui takaisin vastaanottimeen. Kun alue on keilattu, aineisto luokitellaan, pakataan ja siirretään tiedostopalveluun ladattavaksi.

Sovelluksen onkin pystyttävä erottamaan mahdolliset lintu-, maa- sekä vesistöosumat puusto-osumista.

Tyypillisesti laserkeilausaineistojen tiedostopääte on las, kuitenkin Maanmittauslaitoksen tiedostopalvelusta tulleet tiedostot ovat laz-tiedostopäätteisiä. Laz-tiedostossa pistejoukko on pakattu murto-osaan alkuperäisestä koostaan. Algoritmin kuvaus löytyy artikkelista http://www.lastools.org/download/laszip.pdf.

Maanmittauslaitos suosittelee käyttämään tiedostojen purkuun sovellusta Laszip (Avoimien aineistojen tiedostopalvelu). Kyseiseen sovellukseen tiedostopalvelusta löytyy linkki, josta työkalun voi ladata. Kuviossa 2 on kuvakaappaus sovelluksen graafisesta käyttöliittymästä.



Kuvio 2. Kuvankaappaus Laszip työkalusta

2 Perusteet

2.1 Binääritiedosto

Aineisto on koottu LAS-standardin mukaisesti binääritiedostoon. Seuraavassa lyhyesti binääritiedostojen rakenteesta.

Pienin mahdollinen tietomäärä yksittäisessä tiedostossa on yksi tavu, joka koostuu kahdeksasta bitistä, ottamatta huomioon tyhjää tiedostoa, jossa ei ole ainuttakaan tavua. Jokainen tavun kahdeksasta bitistä sisältää arvon nolla tai yksi. Yhdessä tavussa on siis 256 ainutlaatuista yhdistelmää.

Tyypillinen ASCII-koodattu tiedosto, jota pystyy käsittelemään perinteisillä tekstinkäsittelytyökaluilla, ja jota kutsutaan puhekielessä tekstitiedostoksi, koostuu myös tavuista. ASCII-koodatun tiedoston lukija käyttää ASCII-taulukkoa hyväkseen ja muuttaa tavun numeroarvon sitä vastaavaksi kirjaimeksi ASCII-standardin mukaisesti. Esimerkkinä char-tietotyypin tavu 0100 0001, joka vastaa kymmenjärjestelmän arvoa 65, on ASCII-koodattuna kirjain A (ASCII Table and Description, N.d.).

Riippuen tietotyypistä yksi tavun biteistä voi myös ilmaista, että kyseessä on negatiivinen arvo. Esimerkkinä tietotyypit signed char ja unsigned char; molemmat ovat yksitavuisia tietotyyppejä, mutta tyypin signed char kymmenjärjestelmän arvoalue on välillä -128 ja 127, kun taas unsigned char on välillä 0–255. Lisäksi on tietotyyppejä, jotka koostuvat useammasta kuin yhdestä tavusta (ks. taulukko 1).

Taulukko 1. Tietotyypit ja niiden tavukoko

Tietotyypit

Tyyppi	Koko
Char	1 tavu
Signed char	1 tavu
Unsigned char	1 tavu
Short	2 tavua
Unsigned short	2 tavua
Long	4 tavua
Unsigned long	4 tavua
Long long	8 tavua
Unsigned long long	8 tavua
Float	4 tavua
Double	8 tavua
String	1 tavu per merkki, Ascii koodattuna

Binääritiedostoissa on kuitenkin käytännön ongelmia: tiedostoa ei pysty helposti avaamaan ja muokkaamaan tyypillisillä tekstinkäsittelyohjelmilla, koska lukijan pitää tietää tiedoston rakenne. Esimerkkinä ongelmasta olisi tiedosto, jossa on yhteensä 2 tavua. Mikäli lukija ei tunne tiedoston rakennetta, se ei voi tietää tavujen tietotyyppiä, ja täten ei pysty tulkitsemaan tavujen arvoa oikein.

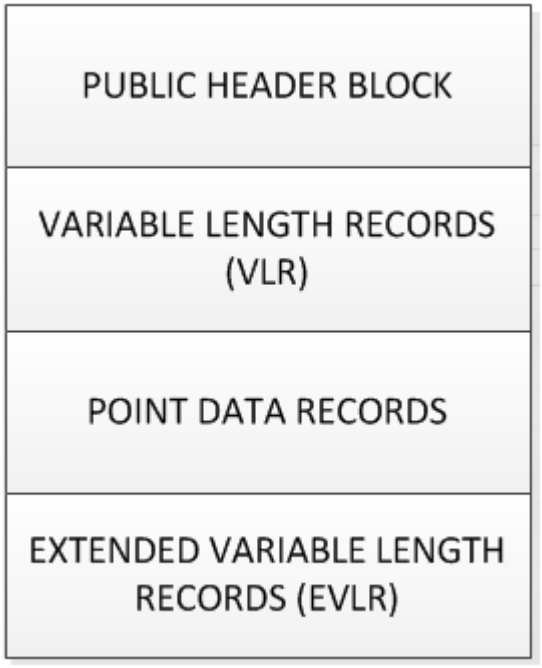
Binääritiedostolukusovellusten pitää tietää myös tiedoston koko, mikäli tiedosto koostuu lohkoista, joiden määrä voi vaihdella. Tämän ongelman ratkaisemiseksi binääritiedostot tyypillisesti sisältävät otsikko-osion. Tämä osio on tyypillisesti myös binäärillä kirjoitettu, joten otsikon rakenteen pitää ilmetä standardin dokumentaatiosta.

Mainittakoon vielä, että koska ASCII-koodausstandardi käyttää vain yhtä tavua jokaiselle merkille, ei siihen mahdu kaikki maailman kielten merkit. Tämän ongelman korjaamiseen on tullut monia koodausstandardeja, jotka ovat joko yksitavuisia tai monitavuisia. Näistä standardeista yksi laajimmista ja suosituimmista on Utf-8.

2.2 LAS-formaatti

LAS-tiedosto, standardin version 1.4 mukaan, koostuu tavuista, jotka on kyseisen standardin mukaisesti kirjoitettu. LAS on julkinen tiedostotyyppi, joka on tarkoitettu kolmiulotteisessa koordinaatistossa olevien pisteiden sijaintitiedon varastointiin (LASer (LAS) File Format Exchange Activities). Standardin käyttämä tavujärjestys on “little endian” (Las specification version 1.4 - R13, 2013, 3).

LAS-tiedosto koostuu 2 – 4 eri lohkosta. Lohkojen nimet standardin mukaisessa järjestyksessä ovat: julkinen otsikkolohko, muuttuvapituiset tietueet, pistetietotallenteet ja laajennetut muuttuvapituiset tietueet. Näistä neljästä osiosta pakollisia ovat julkinen otsikkolohko ja pistetietotallenteet, jotka sisältävät itse pisteosumat. Alkuperäiset englanninkieliset nimet lohkoille standardin mukaisessa järjestyksessä on esitetty kuviossa 3. Solujen sisällä suluissa on nimien lyhennykset.



Kuvio 3. Las 1.4 standardin mukainen tiedostorakenne englanniksi

Julkinen otsikkolohko

Tiedoston otsikkolohko sisältää tiedoston metatiedon, jota sovellus käyttää tiedoston tulkitsemiseen (Binary file, N.d.). Lohko on pakollinen jokaisen LAS-standardin mukaisen tiedoston alussa.

Tiedoston otsikko on pakollinen jokaisen LAS-standardin mukaisen tiedoston alussa. Mitään otsikon sisältämää arvoa ei voi jättää kirjoittamatta. Mikäli osioihin ei ole syöttää arvoja, jokainen osio on alustettava nollalla (Las specification version 1.4 - R13, 2013, 5).

Taulukossa 2 esitetään standardin version 1.4 otsikkolohko, joka on täsmälleen 375 tavua pitkä. Vaikka projektin tunnistekentät eivät ole pakollisia, on nekin alustettava 0-arvoilla, mikäli ne jätetään käyttämättä (Mts. 5 - 6).

Taulukko 2. Julkinen otsikkolohkon rakenne

Osio	Tyyppi	Koko	Pakollinen	Versio
File Signature ("LASF")	char[4]	4 tavua	Kyllä	1.2
File Source ID	unsigned short	2 tavua	Kyllä	1.2
Global Encoding	unsigned short	2 tavua	Kyllä	1.2
Project ID - GUID data 1	unsigned long	4 tavua	Ei	1.2
Project ID - GUID data 2	unsigned short	2 tavua	Ei	1.2
Project ID - GUID data 3	unsigned short	2 tavua	Ei	1.2
Project ID - GUID data 4	unsigned char[8]	8 tavua	Ei	1.2
Version Major	unsigned char	1 tavu	Kyllä	1.2
Version Minor	unsigned char	1 tavu	Kyllä	1.2
System Identifier	char[32]	32 tavua	Kyllä	1.2
Generating Software	char[32]	32 tavua	Kyllä	1.2
File Creation Day of Year	unsigned short	2 tavua	Kyllä	1.2
File Creation Year	unsigned short	2 tavua	Kyllä	1.2
Header Size	unsigned short	2 tavua	Kyllä	1.2
Offset to point data	unsigned long	4 tavua	Kyllä	1.2
Number of Variable Length Records	unsigned long	4 tavua	Kyllä	1.2
Point Data Record Format	unsigned char	1 tavu	Kyllä	1.2
Point Data Record Length	unsigned short	2 tavua	Kyllä	1.2
Legacy Number of point records	unsigned long	4 tavua	Kyllä	1.2
Legacy Number of point by return	unsigned long [5]	20 tavua	Kyllä	1.2
X scale factor	double	8 tavua	Kyllä	1.2
Y scale factor	double	8 tavua	Kyllä	1.2
Z scale factor	double	8 tavua	Kyllä	1.2
X offset	double	8 tavua	Kyllä	1.2
Y offset	double	8 tavua	Kyllä	1.2
Z offset	double	8 tavua	Kyllä	1.2
Max X	double	8 tavua	Kyllä	1.2
Min X	double	8 tavua	Kyllä	1.2
Max Y	double	8 tavua	Kyllä	1.2
Min Y	double	8 tavua	Kyllä	1.2
Max Z	double	8 tavua	Kyllä	1.2
Min Z	double	8 tavua	Kyllä	1.2
Start of Waveform Data Packet Record	unsigned long long	8 tavua	Kyllä	1.3
Start of first Extended Variable Length Record	unsigned long long	8 tavua	Kyllä	1.4
Number of Extended Variable Length Records	unsigned long	4 tavua	Kyllä	1.4
Number of point records	unsigned long long	8 tavua	Kyllä	1.4
Number of points by return	unsigned long long [15]	120 tavua	Kyllä	1.4

Eroavaisuutena aiempiin standardin versioihin on otsikon pituus. Standardi 1.3 on kooltaan 235 tavua pitkä, kaikki aiemmat arvot ovat kooltaan, tyypiltään ja sijainniltaan identtisiä (Las specification version 1.3 - R11, 2010, 5).

Standardia 1.3 aiempi, standardi 1.2, on 227 tavua pitkä (Las specification version 1.2.2008. 3). Tämäkin arvot ovat kooltaan, tyypiltään ja sijainniltaan identtisiä.

Muuttuvapituiset tietueet

Julkisen otsikkolohkon jälkeen seuraa otsikossa määrätty määrä muuttuvapituisia tietueita (Las specification version 1.4 - R13. 8). On siis myös mahdollista, ettei kyseisiä osioita ole tiedostossa ainuttakaan. Taulukossa 3 on muuttuvapituisten tietueiden rakenne tarkennettu.

Taulukko 3. Muuttuvapituisten tietueiden rakenne

Osio	Tyyppi	Koko	Pakollinen
Reserved	unsigned short	2 tavua	Ei
User ID	char[16]	16 tavua	Kyllä
Record ID	unsigned short	2 tavua	Kyllä
Record Length After Header	unsigned short	2 tavua	Kyllä
Description	char[32]	32 tavua	Ei

Jokainen lohko on vähintään 54 tavua pitkä ja koostuu viidestä muuttujasta. Huomioitavaa on, että tietuetunniste-muuttuja on riippuvainen käyttäjätunniste-muuttujasta. Käyttäjätunniste-muuttujalla on siis käytössä 0 – 65536 tietue tunnistetta. Kaikki käyttämättömät osat on alustettava arvolla nolla.

Pistetietotallenteet

Pistetietotallenteet sisältävät itse pisteosumat, joiden määrä on määritelty otsikkolohkossa. Ennen LAS-standardin versiota 1.4 pistetietotallenteita saattoi olla enintään hieman alle 4,3 miljardia. (Mts. 7). Uusimmassa formaatissa 1.4 pistetietotallenteita voi enintään olla noin $1.8 * 10^{19}$.

Ensimmäinen pistetietotallenne lohko alkaa julkisen otsikkolohkon määräämästä siirtymä pistetietoihin -arvosta. Arvo on tavumäärä tiedoston alusta lohkon ensimmäiseen tavuun (Mts. 9). Kyseisellä lohkolla on 11 virallista formaattia. Kehitysluokkien tunnistussovellus on käyttänyt formaattia yksi (ks. taulukko 4).

Taulukko 4. Pistetietotallenteiden ensimmäisen formaatin rakenne

Osio	Tyyppi	Koko	Pakollinen
X	long	4 tavua	Kyllä
Y	long	4 tavua	Kyllä
Z	long	4 tavua	Kyllä
Intensity	unsigned short	2 tavua	Ei
Return number	3 bittiä (bitit 0 - 2)	3 bittiä	Kyllä
Number of Returns (given pulse)	3 bittiä (bitit 3 - 5)	3 bittiä	Kyllä
Scan Direction Flag	1 bitti (bitti 6)	1 bitti	Kyllä
Edge of Flight Line	1 bitti (bitti 7)	1 bitti	Kyllä
Classification	unsigned char	1 tavu	Kyllä
Scan Angle Rank (-90 to +90) - Left side	char	1 tavu	Kyllä
User Data	unsigned char	1 tavu	Ei
Point Source ID	unsigned short	2 tavua	Kyllä
GPS Time	double	8 tavua	Kyllä

Laajennetut muuttuvapituiset tietueet

Laajennettujen muuttuvapituisten tietueiden määrä on määrätty myös otsikkolohkossa. Tallenteet sisältävät raa´an aaltoliikkeen amplitudiarvot, joita tarvitaan pistetietotallenneformaattien neljä, viisi, yhdeksän ja kymmenen kanssa (Mts. 27). Rakenteeltaan kyseinen lohko on identtinen muuttuvapituisten tietueiden kanssa (ks. taulukko 3).

2.3 Python

Perusteet

Python on laajasti käytetty, korkeantason, tulkattu ohjelmointikieli. Kieli pyrkii pitämään koodin luettavana ja lyhyenä. Python tukee monia ohjelmointiparadigmoja kuten proseduraalista, oliopohjaista ja funktionaalista. Python-tulkkeja on monille käyttöjärjestelmille, kuten Windows, Mac OS X ja useimmille Linux-jakeluille (Python (programming language), N.d.).

Pythonin uusin vakaa versio kirjoitushetkellä on 3.4.2 (Python, N.d,). Myös vanhempaa, 2.7-haaraa jatkokehitetään. Kehitysluokkien tunnistussovellus on tehty käyttäen Python-versiota 3.2.

Moduulit

Pythonin ohjelmakirjastoja kutsutaan moduuleiksi. Kun käyttäjä haluaa Python-sovellukseen toiminnallisuuksia muiden Python-tiedostojen luokista tai funktioista, tulee käyttää import-avainsanaa. Avainsanaa seuraa moduulin tiedostonimi ilman tiedostopäätettä. Näin koodiin tulee tiedostonimeä muuttuja, josta pääsee käsittelemään tiedoston luokkia ja funktioita. Muuttuja on kuin luokan objekti ja tiedoston sisällä olevat funktiot ovat kuin sen metodeja ja luokat luokkia.

Mikäli käyttäjä haluaa kaikki toisen tiedoston funktiot ja luokat kooditiedostoonsa ilman erillistä muuttujaa, tulee käyttää avainsanaa from ja tätä seuraa ”import *”.

```
# imports everything from module
import python_module
from my_first_module import *

# imports specific classes from module
from my_second_module import my_first_class, my_second_class
from my_third_module import my_finest_class as fine_class
```

Käyttäjä voi myös ladata vain tiettyjä luokkia tai funktioita käyttäen avainsanoja from ja import joita seuraa halutut ominaisuudet. Haluttuja ominaisuuksia voi uudelleen nimetä käyttämällä as-avainsanaa.

Eroavaisuudet ohjelmoinnissa C-kieleen

Pythonia ensi kertaa lukiessa huomio kiinnittyy sulkeiden ja puolipisteiden puuttumiseen. Python-kielessä koodilohkot erotetaan joko yhtenäisillä välilyönneillä tai sisennyksillä. Rivinvaihto korvaa puolipisteet yksittäisten funktioiden lopetuksessa.

Pythonissa on dynaaminen tyypitys eli tulkki ajonaikaisesti määrää muuttujille tyypin annetun arvon perusteella. Tyypityksestä on esimerkki:

```
int c_integer = 10;
char[] c_char_array = "Hello World";

python_integer = 10
python_string = "Hello World"
```

Pythonissa on automaattinen roskienkeruu, joka huolehtii kaikkien muuttujien vapauttamisesta muistista. Pythonissa ei myöskään erikseen ohjata muuttujien muistiosoitteita osoittimin tai viittauksilla.

Myös main-metodissa on selkeä ero. Pythonissa ei ole pakollista tehdä main-funktiota lainkaan, ja sen alustaminenkin on selkeästi erilaista kuin C:ssä. Mikäli main puuttuu, lähtee tulkki suorittamaan ohjelmaa rivi kerrallaan ylhäältä alaspäin ensimmäisestä sisennyksestä.

Main ei ole oma funktionsa, vaan ehtolause. Jos ehtolause on olemassa, alkaa ohjelma siitä ja päättyy ehdon loppumiseen.

```python
if __name__ == "__main__":
    print("Hello World!")
```

Muuttujien parametrit

Äkkiseltään voi vaikuttaa siltä, että Python käyttää metodien ja funktioiden parametreihin muuttujien osoittimia. Tämä ei kuitenkaan pidä paikkaansa. Ohjelmointikielissä, kuten C:ssä ja C++:ssa, on mahdollista, että parametreissa muuttuja on arvona tai osoittimena. Pythonissa ei ole kumpaakaan, vaan parametreissa on muuttujan olio. Tämä saattaa aiheuttaa käyttäjille ongelmia.

```cpp
#include <iostream>
using namespace std;

struct ExampleStruct
{
    int integerValue;
};

void myFunction(ExampleStruct obj)
{
    obj.integerValue = 2;
}

void printStruct(ExampleStruct obj)
{
    cout << obj.integerValue << endl;
}

int main()
{
   ExampleStruct myStruct;
   myStruct.integerValue = 1;
   myFunction(myStruct);
   printStruct(myStruct);
   return 0;
}
```

```cpp
#include <iostream>
using namespace std;

struct ExampleStruct
{
    int integerValue;
};

void myFunction(ExampleStruct* obj)
{
    obj->integerValue = 2;
}

void printStruct(ExampleStruct* obj)
{
    cout << obj->integerValue << endl;
}

int main()
{
   ExampleStruct myStruct;
   myStruct.integerValue = 1;
   myFunction(&myStruct);
   printStruct(&myStruct);
   return 0;
}
```

Edellisillä riveillä on esimerkit C++-koodeista joista muuttujista välitetään joko arvo tai osoitin. Vasemmanpuoleisessa myFunction-funktio saa parametrinsa arvona. Vaikka tällöin funktio muuttaakin luokan oliota, muutokset eivät säily funktion ulkopuolella. Oikealla taas on esimerkki, jossa parametri on osoitin, jolloin muutokset säilyvät.

Pythonissa ei pysty määräämään erikseen, onko parametreissa muuttujan osoitin. Pythonissa metodin parametrin voi välittää kahdella eri tavalla riippuen onko kyseessä muuttumaton tietotyyppi vai ei (Evaluation strategy, N.d.). Mikäli kyseessä on muuttumaton tietotyyppi, muuttujan sisältö on käytännössä arvo.

```
class ExampleClass:
    integer_value = None

def my_function(obj):
    obj.integer_value = 2

def print_object(obj):
    print(obj.integer_value)

if __name__ == "__main__":
    my_object = ExampleClass()
    my_object.integer_value = 1
    my_function(my_object)
    print_object(my_object)
```

Edellisessä esimerkissä luokan olion integerValue-arvoa muutetaan funktiossa. Sovellus toimii kuten voi olettaa ja tulostaa numeron kaksi. Seuraavassa koodiesimerkissä funktion olio ei enää viittaa alkuperäiseen muuttujaan. Joten vaikka olion arvoja muutetaan funktiossa, muutokset eivät säily funktion ulkopuolella.

```
class ExampleClass:
    integer_value = None

def my_function(obj):
    obj = ExampleClass()
    obj.integer_value = 2

def print_object(obj):
    print(obj.integer_value)

if __name__ == "__main__":
    my_object = ExampleClass()
    my_object.integer_value = 1
    my_function(my_object)
    print_object(my_object)
```

Edellinen esimerkkisovellus tulostaa arvon yksi. Tämän parametrissa olio ominaisuuden vuoksi on Python-ohjelmointikielen parissa oltava tarkkana muuttujien parametrien kanssa. Ongelma tulee vastaan erityisen helposti seuraavassa koodiesimerkissä:

```
def my_function(list_object):
    list_object = [0,1,2]

if __name__ == "__main__":
    my_list = [0,1]
    my_function(my_list)
    print(my_list)
```

Edellinen esimerkkisovellus tulostaa kaksisoluisen listan arvoina nolla ja yksi. Parametrien käytön minimoinnilla voidaankin välttää useimmat virhemahdollisuudet.

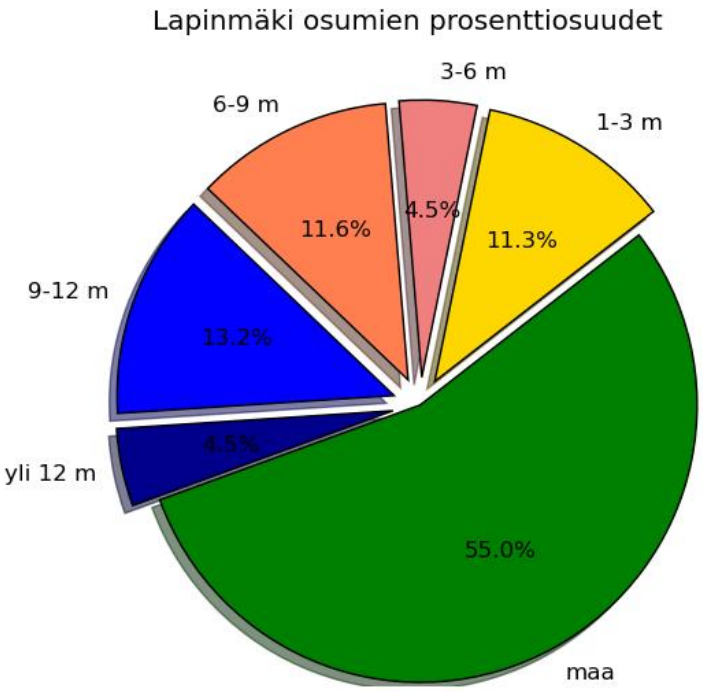
Kehitysluokkien tunnistussovelluksen käyttämä Matplotlib-moduuli

Kehitysluokkien tunnistukseen tehty sovellus käyttää graafisessa piirrossa hyväkseen Matplolib-moduulia. Kyseinen moduuli on kaksiulotteisten kuvien piirtämiseen erikoistunut ohjelmistokirjasto, joka tuottaa ja piirtää julkaisukelpoisia taulukoita ja kuvia (matplotlib, N.d.).

Sovellus tuottaa kuvia, joilla havainnoidaan eri kehitysluokkien sijainteja ja kokoja. Seuraavassa esimerkki kuvion piirto-ohjelmakoodista:

```
import matplotlib.pyplot as pl

hit_ground = []
hit_1_3 = []
hit_3_6 = []
hit_6_9 = []
hit_9_12 = []
hit_over_12 = []

# hit_grounds = something...
# etc

pl.title('Lapinmäki osumien prosenttiosuudet\n\n')
labels = 'maa', '1-3 m', '3-6 m', '6-9 m', '9-12 m', 'yli 12 m'
pieces = [hit_ground, hit_1_3, hit_3_6, hit_6_9, hit_9_12, hit_over_12]
colors = ['green', 'gold', 'lightcoral', 'coral', 'blue', 'darkblue']
explode = (0, 0.1, 0.1, 0.1, 0.1, 0.1)
pl.pie(pieces, explode=explode, labels=labels, colors=colors,
    autopct='%1.1f%%', shadow=True, startangle=200)
pl.axis('equal')

pl.show()
```

Edellinen ohjelmakoodi piirtää piirakkadiagrammin (ks. kuvio 4). Syöttämällä korkeuden perusteella jaetut osumapistelistat algoritmille, tuloksena on esimerkiksi seuraavanlainen kuvio:



Kuvio 4. Aiemman ohjelmakoodin piirtämä kuvio, esimerkkitaulukkoarvoilla

Kyseinen kuvio (ks. Kuvio 4) voidaan myös tallentaa suoraan tiedostoon lisäämällä algoritmiin seuraava ohjelmankoodi:

```
pl.savefig('Lapinmaki_osumien_prosenttiosuudet.eps')
```

3 Työn suunnitelma

Keväällä 2014 olin mukana Kalevi Pietikäisen vetämässä työryhmässä jatkokehittämässä metsän kehitysluokkien tunnistukseen käytettävää sovellusta. Sovelluksen tarkoitus oli lukea Maanmittauslaitokselta saatuja laserkeilausaineistoja ja luoda niistä kuva, jossa näkyvät alueen metsän kehitysluokat.

Sovellus ei ollut kuitenkaan käyttäjäystävällinen. Ensinnäkin sovellus pystyi lukemaan vain ja ainoastaan ASCII-koodattuja laserkeilausaineistoja. Käyttäjän piti siis ladattuaan aineiston itse muuntaa se ASCII-koodatuksi tiedostoksi. Tämä on täysin turhaa työtä käyttäjälle. ASCII-koodatut aineistot ovat myös huomattavasti suurempia kuin alkuperäiset tiedostot, joten ne vievät myös turhaa kovalevytilaa.

Sovellus pystyi myös kirjoittamaan käsiteltyjä ASCII-koodattuja laserkeilausaineistoja. Käsitellyt aineistot koostuivat ASCII-koodutuista pistetietotallenteista. Tietyt pistetietotallenteiden -muuttujat olivat ylikirjoitettu ja tarkoittivat nyt kehitysluokkia. Täten käyttäjä pystyi ottamaan tiedostoista koordinaatit ja kehitysluokat ja siirtämään ne ArcGIS-työkaluun graafista esitystä varten. Tämä on kuitenkin hyvin epäkäytännöllinen tapa esittää aineistoa.

Käytössä oli myös Pythonin Matplotlib-moduuli, joka myös tuotti graafisia esityksiä aineistosta. Kirjasto on kuitenkin varsin yksinkertainen verrattuna ArcGIS-työkaluun ja työkalu oli valmiiksi asiakkaalle tuttu.

Sovelluksessa oli siis käyttäjän kannalta huomattavia puutteita, jotka hidastivat sekä loivat turhaa monimutkaisuutta sovelluksen käyttöön. Työssä selvitettiin miten nämä puutteet voidaan korjata.

Työssä tulisi selvittää, miten voitaisiin vähentää käyttäjän tekemää työmäärää ja täten yksinkertaistaa sovelluksen käyttöä. Ensimmäiseksi olisi selvitettävä, voiko aineiston ladata sovellukseen suoraan LAS-standardin mukaisena tiedostona ja selvittää, miten tämä vaikuttaisi lukunopeuksiin. Ennakko-oletus oli että laserkeilausaineisto käännettynä ASCII-muotoiseksi ei ole tehokkain tapa.

Käsiteltyään aineiston sovelluksen pitäisi pystyä kirjoittamaan kehitysluokallinen aineisto niin, että sen siirtäminen ArcGIS-työkaluun olisi mahdollisimman yksinkertaista. Työssä tulisi selvittää, miten aineisto saataisiin yksinkertaisesti ArcGIS-työkaluun. Työhön myös pyydettiin ominaisuus esittää halutun metsäalueen rajat graafisesti Matplotlib-moduulia käyttäen. Alue luettiin xml-tiedostosta.

Työ rajattiin Linux-jakelun Ubuntu 12.07 -käyttöjärjestelmäympäristöön, käyttäen Pythonin versiota 3.2.

4 Toteutus

4.1 LAS- ja ASCII-formaattien väliset luku- ja kirjoituserot

Pohdinta

Ennakko-oletuksena oli, että kiintolevylle kirjoittaminen on hitaampaa kuin sieltä lukeminen. Selvittääkseen kuinka suuri ero näissä on, pitää suorittaa ajanotto sekä lukemisesta että kirjoittamisesta, niin LAS- kuin ASCII-koodattuihin tiedostoihin. Ajanotto pitää toteuttaa samassa ympäristössä ja samanlaisessa sovelluksessa. Testauksessa on käytetty HDD-kovalevyä.

Vertaus

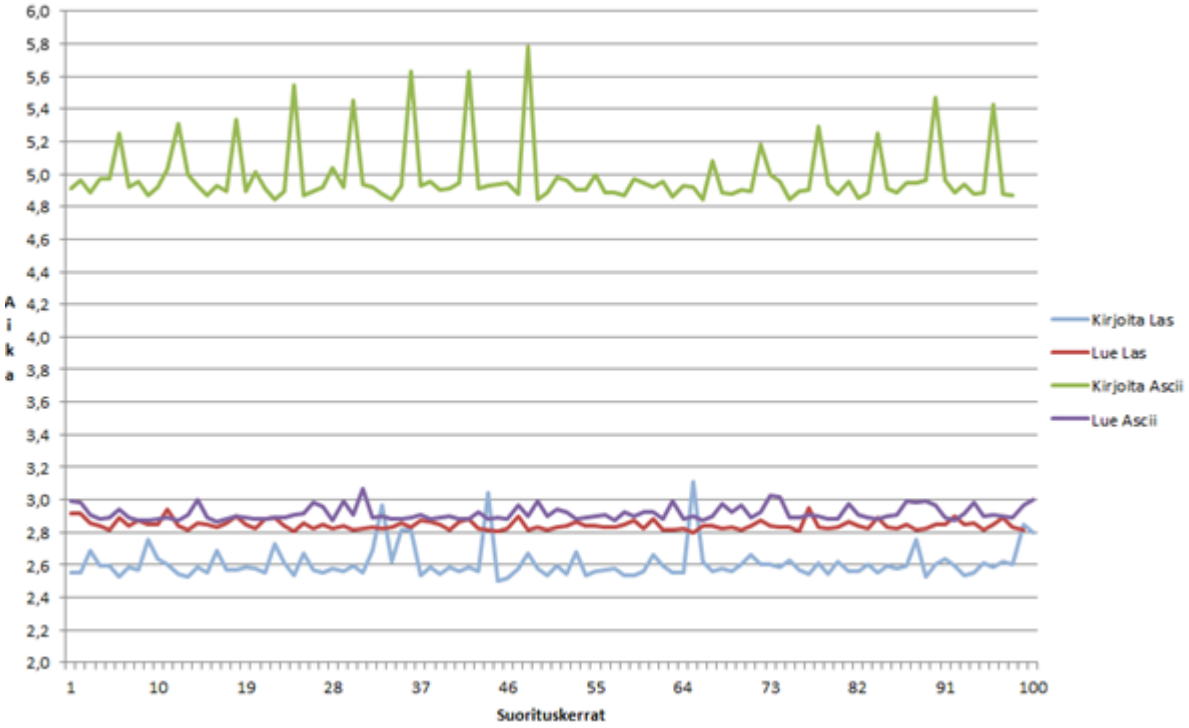
Testaus koostui kahdesta eri sovelluksesta. Ensimmäinen sovellus, jonka käyttäjä itse käynnisti, lähettää käyttöjärjestelmälle käskyn käynnistää uuden prosessin, mikä käynnistää toisen sovelluksen. Toinen sovellus oli varsinainen tiedostojen lukija ja kirjoittaja. Prosessi käynnisti samalla Ubuntu-käyttöjärjestelmään asennetun Time-sovelluksen joka otti aikaa, kuinka kauan prosessin käynnistämä sovellus oli päällä. Käyttäjän käynnistämä sovellus toisti tätä prosessien käynnistämistä sata kertaa niin lukemiselle kuin kirjoittamiselle.

Jälkimmäinen sovellus, jonka ensimmäinen sovellus oli käynnistänyt, saa käynnistyessään avainsanan, jonka perusteella se joko kirjoitti tai luki, määräten myös oliko tiedostomuoto ASCII vai LAS. Sovellus käytti Pythonin Timeit-moduulia, jolla pystyy ottamaan mikrosekuntien tarkkuudella aikaeron kahden hetken väliltä. Time ja Timeit erona on, että Timeit laskee vain kirjoitusajan tai lukuajan, kun taas Time-ohjelma laskee koko prosessin keston. Tästä syystä Time-ohjelman ajat ovat suurempia.

```
398218 678762 132 16 2 1 3 1 23 20933176.6295
```

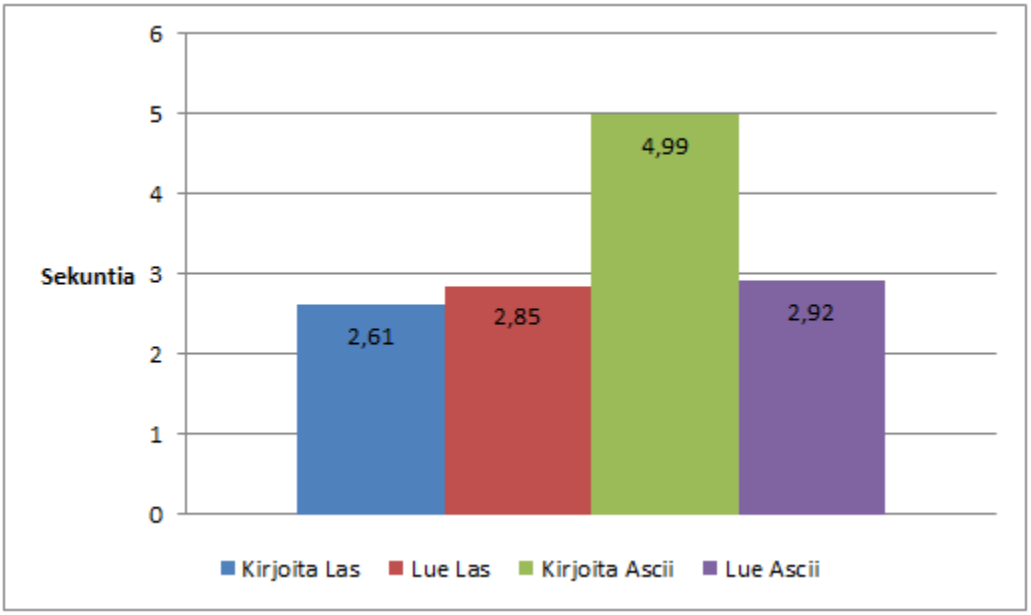
Edellisellä rivillä on rivi luetusta tiedostosta. Luettu tiedosto sisälsi miljoona samaa riviä, jotka ovat laserkeilauspisteitä. Tiedoston pisteiden arvot eivät muuttuneet. Sovelluksen piti kestää vähintään muutaman sekunnin, jotta sovelluksesta riippumattomat hidasteet eivät sotkisi tilastoja. Yksi näistä hidasteita voisi olla taustalla suoritettava prosessi johon käyttöjärjestelmä varaa resursseja. Itse testiä myös toistettiin useita kertoja, jotteivät satunnaismuuttujat sotkisi tuloksia.

Kuvio 5 koostuu 100 toistosta, jossa luku- ja kirjoitusajat on otettu ylös käyttäen Timeit-moduulia. Kyseisestä kuvioista voi verrata formaattien välistä luku- ja kirjoitusnopeutta.

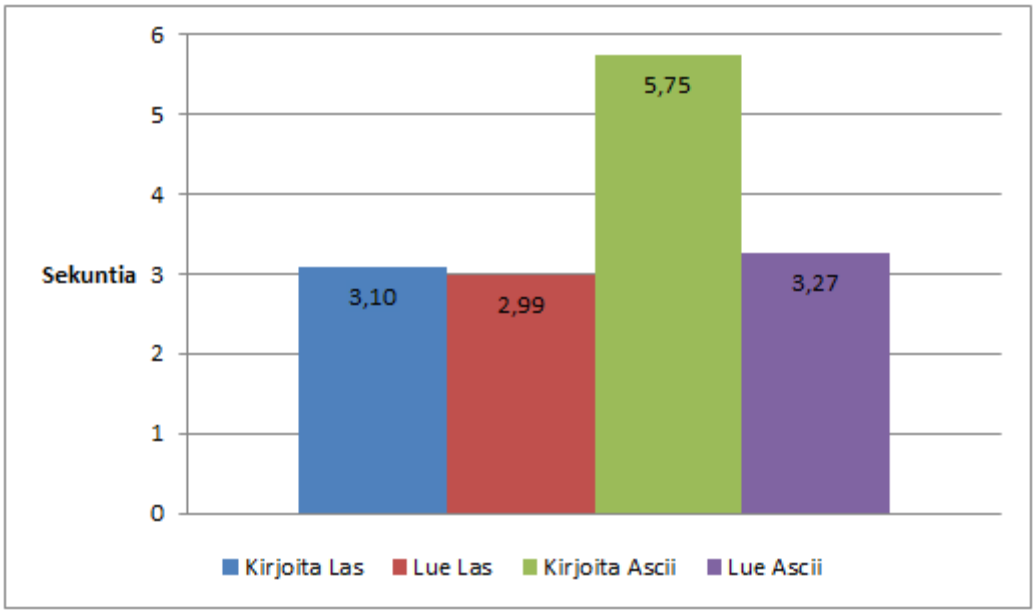


Kuvio 5. Toistokerran kesto Timeit-moduulin ilmoittamana

Kuvioissa 6 ja 7 oli otettu keskiarvot kustakin luku- ja kirjoitusprosessista. Kuvioista voi nähdä että Time-sovelluksen ottamat ajat ovat pidempiä. Syy tähän on että Time-sovellus ottaa myös ohjelma-prosessin käynnistämisen ja sammuttamisen huomioon, kun Timeit ottaa pelkästään kirjoitus- ja/tai lukuprosessin huomioon.



Kuvio 6. Timeit-moduulin ilmoittamien tuloksien keskiarvot



Kuvio 7. Time-ohjelman ilmoittamien tuloksien keskiarvot

Vertailussa käytetty sovellus

Vertailussa ollut sovellus oli rakenteeltaan hyvin yksinkertainen. Sovellus otti käynnistysparametrina arvon jonka perusteella sitten käynnistettiin joko luku tai kirjoitus sekä pääteltiin formaatti joko binäärinä tai ASCII-koodattuna.

Binäärin kirjoituksessa käytettiin Pythonin Struct-moduulia. Tämä moduuli sisältää metodit arvojen kirjoittamiseen binääriksi sekä binäärin kääntämiseksi takaisin arvoiksi alkuperäisine tyyppeineen. Seuraavassa on esimerkki moduulin Pack-metodista, joka ottaa muuttujat parametrikseen ja muuntaa ne binääriksi:

```
with open ('filename.bin', 'wb') as f:
    st = struct.pack('ii', x,y)
    f.write(st)
```

Seuraavassa on esimerkki Struct-moduulin Unpack-metodista, joka taas ottaa binääritaulukon parametrikseen ja palauttaa listan oikein tyypitettyjä muuttujia:

```
with open ('filename.bin', 'rb') as f:
    binary_data = f.read(8)
    value = struct.unpack('ii', binary_data[:8])
```

Johtopäätös

LAS-tiedoston lukeminen ja erityisesti kirjoittaminen ovat nopeampia. Tähän on kaksi tärkeää syytä. Ensimmäiseksi LAS-tiedosto on kooltaan huomattavasti pienempi kuin ASCII-koodattu tiedosto. ASCII-tiedosto on kooltaan 46 miljoonaa tavua, kun LAS-tiedosto on 28 miljoonaa tavua. Syy tähän kokoeroon johtuu miten arvot esitetään eri tiedostoformaateissa, mistä on selostettu kappaleessa Binääritiedosto.

Toinen tärkeä syy aikaeroon on, että sovellus joutuu muuttamaan tyyppejä ASCII-tiedoston käsittelyssä. Kun ASCII-tiedosto luetaan, siitä tulee merkkijono. Kuitenkin jos arvoa halutaan käsitellä, on merkkijono muutettava numeroarvoksi. Sama toistuu tiedostoon kirjoitettaessa: numeroarvot on muutettava takaisin merkkijonoksi.

Huomioitavaa vertailussa oli, ettei siinä ollut binääritiedostoille tyypillistä otsikkolohkoa, tämä ei kuitenkaan 28 miljoonan tavun tiedoston kokoa juurikaan kasvattaisi, koska otsikko on kooltaan alle 400 tavua.

4.2 Lukijan suunnittelu LAS-formaatille

Johdanto

LAS-formaatti koostuu kahdesta neljään rakenteeltaan erilaisesta osasta, joista kaikki paitsi otsikkolohko voi toistua useasti. Toistojen määrä on määrätty julkisessa otsikkolohkossa.

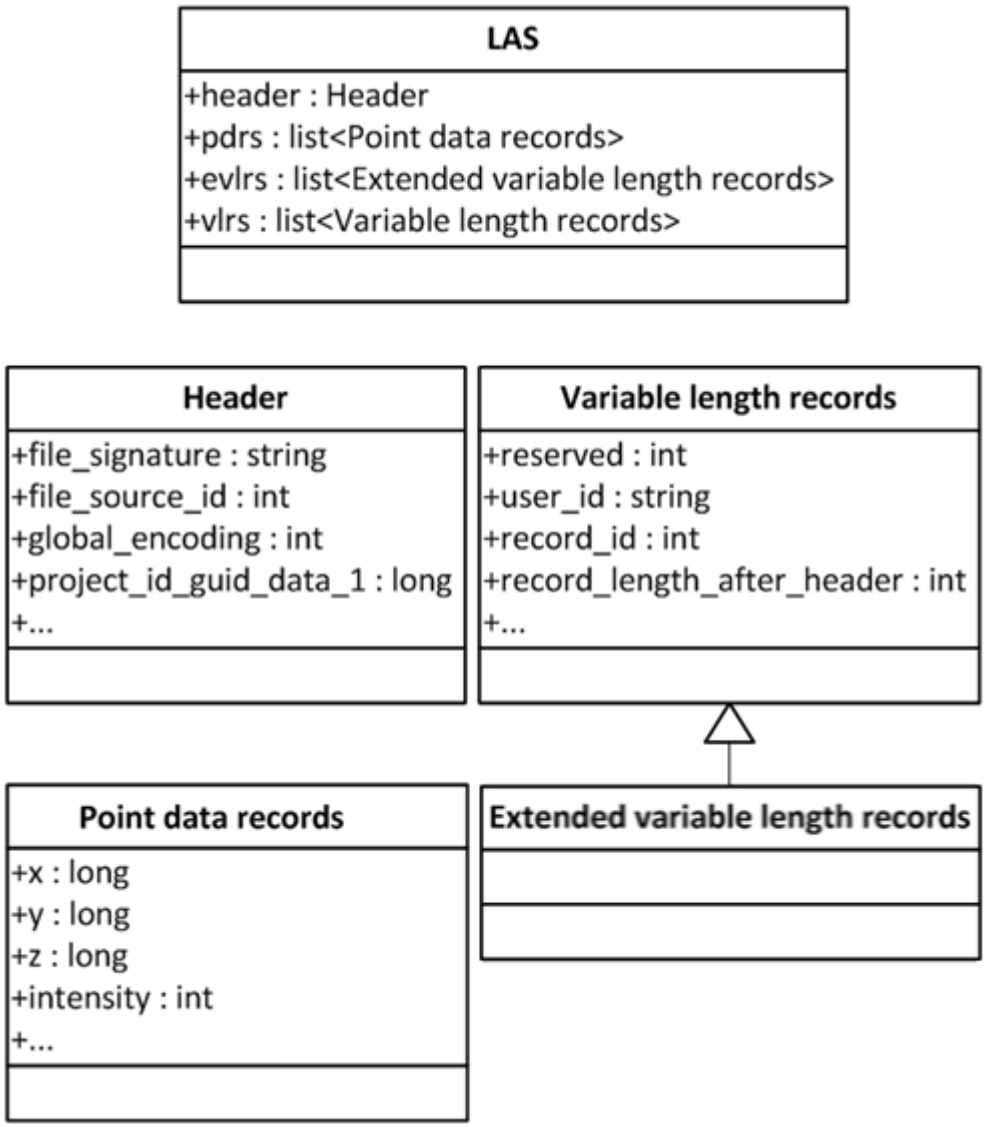
Lukijan tulisi olla mahdollisimman yksinkertainen ylläpidettävyyden ja testattavuuden vuoksi. Lukijan pitäisi pystyä käsittelemään kaikkia LAS-standardin mukaisesti tehtyjä tiedostoja. Työssä voisi myös suunnitella tiedoston lukijan, joka pystyisi käsittelemään vielä julkaisemattomia LAS-standardin versioita mahdollisimman vähällä muokkaamisella.

Perusrakenne

Käyttäen oliopohjaista rakennetta rakenne koostuisi yhdestä pääluokasta. Tämän pääluokan sisällä olisi neljä luokan oliomuuttujaa. Ensimmäisenä julkiselle otsikkolohkolle oma muuttujansa, joka olisi ainoa yksittäinen muuttuja, kaikkien muiden luokkien oliot olisivat listoissa. Toisena tulisi lista muuttuvapituinen tietue -olioista, kolmantena lista pistetietotallenneolioista ja viimeisenä lista laajennetuista muuttuvapituinen tietue -olioista.

Laajennettujen muuttuvapituisten tietueiden rakenne on identtinen muuttuvapituisten tietueiden rakenteen kanssa. Sillä ei tarvitse olla omaa luokkaa, mutta koodin luettavuuden vuoksi olisi hyvä,

että se määriteltäisiin omaksi luokaksi, joka periytyy muuttuvapituisten tietueiden luokasta. Kuviossa 8 on esimerkki oliorakenteesta.



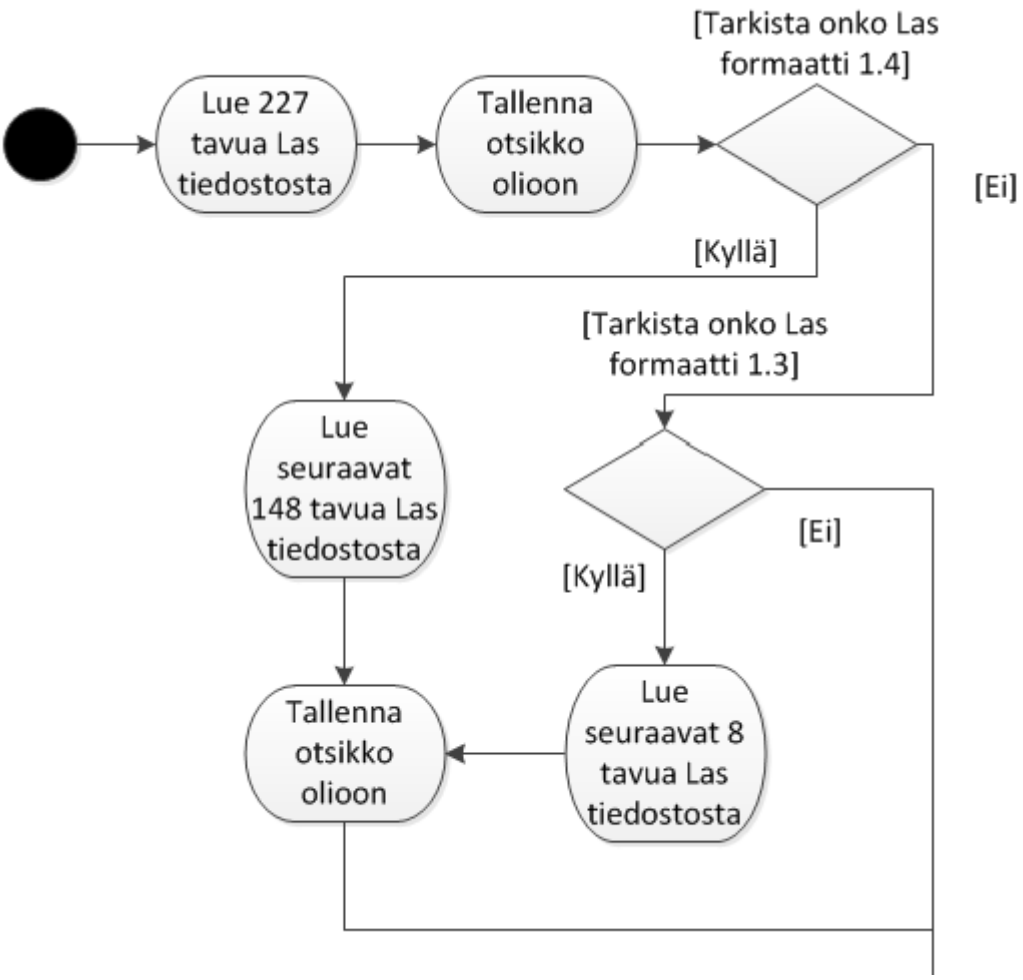
Kuvio 8. Uml-kaavio oliorakenteista

Jokainen pääluokan muuttujana oleva luokan olio sisältää LAS-formaatin mukaiset muuttujat. Piste-tietotallenteet osiolla on kymmenen erilaista virallista formaattia, joista luokan pitäisi tukea jokaista (Mts. 12 - 20).

4.3 Lukijan toteutus LAS-formaatille

Aktiviteettikaavio

Kun sovellus pidetään mahdollisimman yksinkertaisena ja toiminta rajoitetaan ainoastaan LAS-formaatin tiedostoille, saadaan kuviossa 9 esitetty lukija.

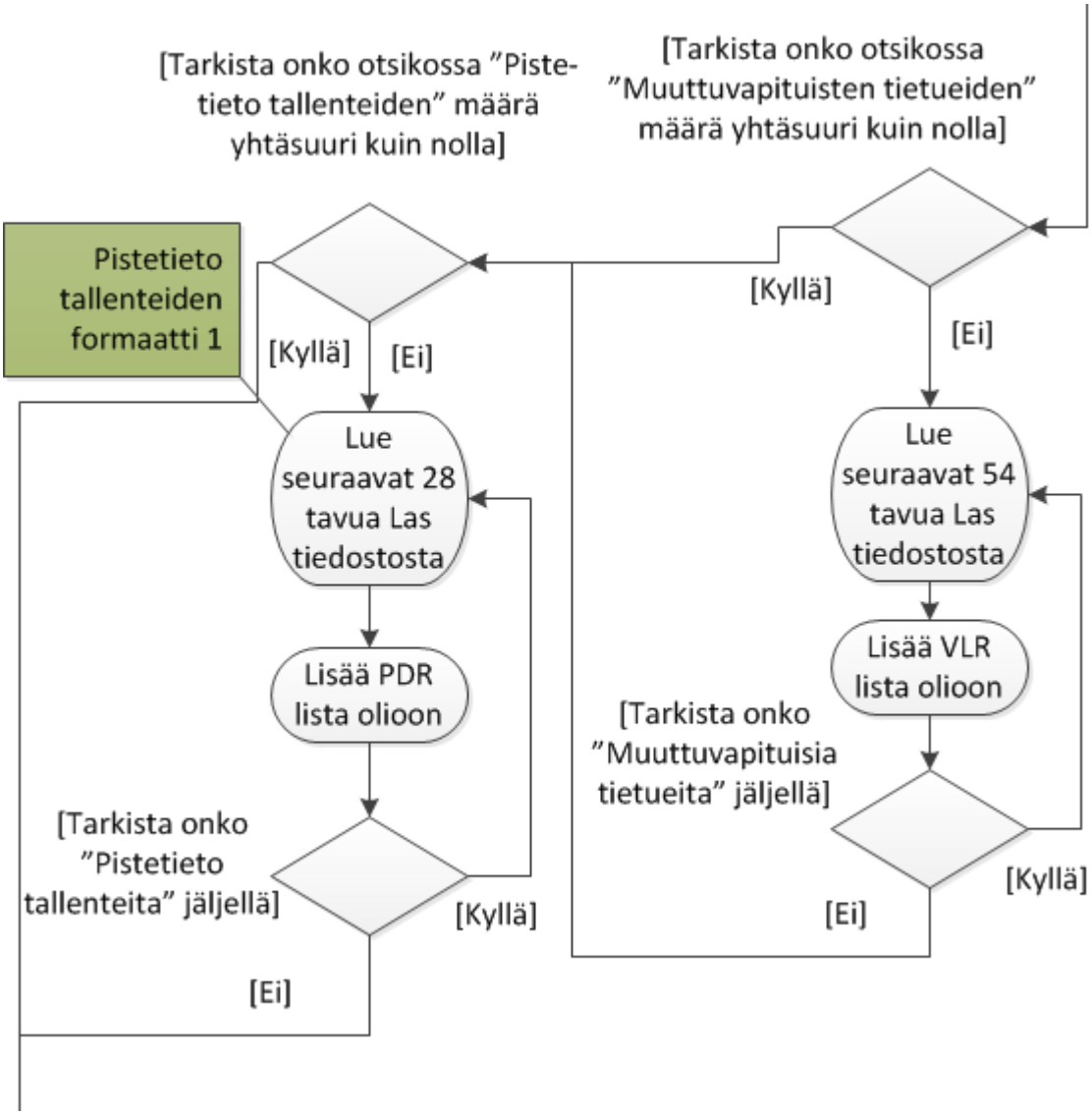


Kuvio 9. Julkisen otsikkolohkon lukukaavio

Sovellus lukee LAS-tiedostosta ensimmäiset 227 tavua eli juuri LAS 1.2-formaatin otsikon. Tästä juuri luetusta otsikosta tarkistetaan, onko tiedostossa käytetty uudempaa formaattia kuin 1.2 ja mikäli on, myös loput otsikosta luetaan.

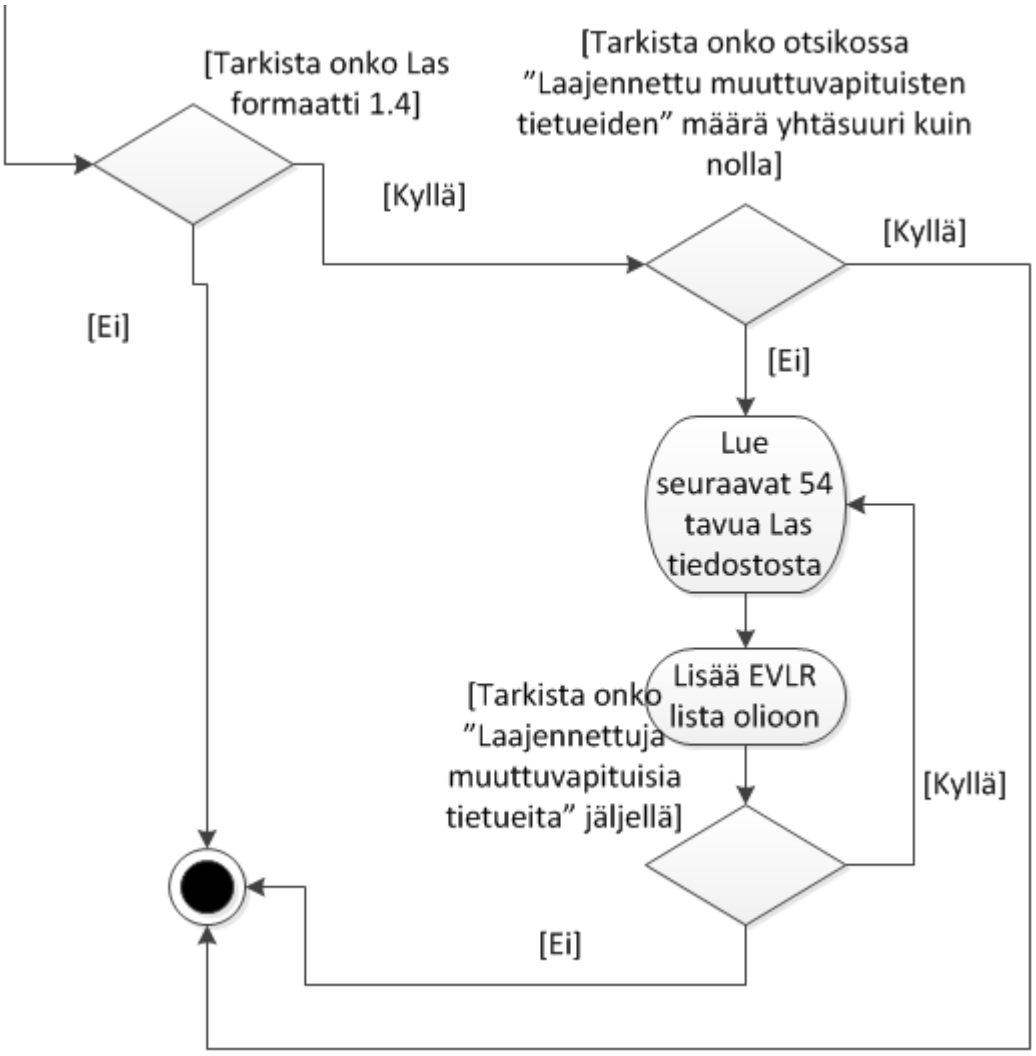
Kun otsikko on kokonaisuudessaan luettu, tarkistetaan otsikosta muuttuvapituisten tietue lohkojen määrä (ks. kuvio 10). Mikäli määrä on suurempi kuin nolla, ohjelma alkaa lukea 54 tavun lohkoja, kunnes kaikki muuttuvapituiset tietueet on luettu. Tämän jälkeen tarkistetaan otsikosta pistetietotallennelohkojen määrä, ja samoin kuin edellisessä, mikäli niiden määrä on suurempi kuin nolla, lue-

taan nekin läpi. Kuviosta tulee huomioida, että 28 tavun pistetietotallenteet pitää paikkansa vain pistetietotallenneformaatti yhdelle.



Kuvio 10. Muuttuvapituisten tietueiden lohkojen sekä pistetietotallenteiden lukukaavio

Lopuksi tarkistetaan, onko tiedosto formaatissa 1.4 ja mikäli on, siitä tarkistetaan, onko laajennettuja muuttuvapituisia tietue lohkoja olemassa. Mikäli se löytyy, niin luetaan nekin läpi. Kuviossa 11 esimerkki.



Kuvio 11. Laajennettujen muuttuvapituisten tietueiden lukukaavio

Toteutus

Kuten jo aiemmin esitetyssä luokkarakenteessa käy ilmi (ks. Kuvio 8), sisältää LAS-luokka neljä muuttujaa. Nämä, otsikkoa lukuun ottamatta, ovat listoja. LAS-luokalla on myös metodeja, tärkeimpinä tiedoston latausmetodi josta koodiesimerkki on esitetty seuraavassa kuviossa:

```
class Las():
    def __init__(self):
        self.header = None
        self.variable_length_records = list()
        self.point_data_records = list()
        self.extended_variable_lenght_records = list()

    def load(self, file_name):
        converted_data = list()
        with open (file_name, 'rb') as f:
            binary_data = f.read(227)
            # the '<' character stands for 'little-endian'
            converted_data += struct.unpack('<cccc', binary_data[:4])
            converted_data += struct.unpack('<HH', binary_data[4:8])

            # ... Rest of the header ...

            converted_data += struct.unpack('<ddd', binary_data[203:227])
            self.header = Header(converted_data)

            # ... Rest of the method ...
```

Edellisessä koodiesimerkissä oleva LAS-luokan load-metodi hakee ja lukee tiedoston polusta muuttujan file_name arvon perusteella. Aktiviteettikaavion mukaisesti (ks. kuvio 9) aloitetaan lukemalla 227 tavua, tämän jälkeen jokainen lohko tallennetaan listaan converted_data. Kun kaikki 227 tavua on luettu, luodaan Header-olio, johon tämä lista syötetään parametriksi. Vastaava toteutus on muuttuvapituisilla tietueilla, pistetietotallenteilla ja laajennetuilla muuttuvapituisilla tietueilla. Seuraavassa esimerkissä on nähtävissä miten muuttuvapituisten tietueiden olion konstruktori rakentaa listasta muuttujansa.

```
class VariableLengthRecord():
    def __init__(self, parameter_list):
        self.reserved = parameter_list[0]
        self.user_id = parameter_list[1]
        self.record_id = parameter_list[2]
        self.record_length_after_header = parameter_list[3]
        self.description = parameter_list[4]
```

Muut vaihtoehdot

Sen sijaan, että toteuttaa itse LAS-tiedoston lukijan, on todennäköisesti tehokkaampaa käyttää jo valmiina olevia kirjastoja. Toteutuksessa ei täten tarvitse keksiä pyörää uudestaan. Internetissä on tarjolla useita erilaisia kirjastoja, jotka on suunniteltu LAS-tiedoston lukuun. Tarjolla olevista sovelluksista tulee selvittää niiden käytön lisenssiehdot, ja hakea näistä asiakkaalle mahdollisesti sopiva.

LAS-tiedoston lukijoita on tarjolla useilla eri ohjelmointikielillä. Täten käytännön kysymys nouseekin, onko Pythonin käyttö lopullisessa toteutuksessa järkevää. Python on tulkattu ohjelmointikieli mikä tarkoittaa, että lähdekoodia ei käännetä konekielelle ennen ajon aloittamista, vaan lähdekoodi syötetään käynnissä olevalle prosessille, jota kutsutaan tulkiksi. Tämä tulkin kautta ajaminen tekee kielestä huomattavasti hitaampaa kuin ei-tulkattu ohjelmointikieli.

Lopullisessa toteutuksessa on syytä selvittää, kuinka tärkeää loppukäyttäjälle on ohjelmiston nopeus. Mikäli ohjelmisto ajetaan muutaman kerran vuodessa palvelinympäristössä, nopeus tuskin on ensimmäinen kriteeri. Mikäli kuitenkin nopeus on tärkeää, olisi syytä katsoa käännettyjä ohjelmointikieliä kuten C ja C++.

Yhteenveto

Toteuttamalla LAS-lukijan itse ovat lisenssiehdot tietysti itse valittavissa, ja aktiviteettikaaviot soveltuvat myös muihin ohjelmointikieliin, mikäli kieltä halutaan vaihtaa. Kaavioiden mukainen sovelluksen yksinkertainen rakenne tekee siitä helposti ylläpidettävän, mikäli tulevaisuudessa tulisi uusia LAS-formaatteja.

4.4 Lukija tuleville taaksepäin yhteensopiville LAS-formaateille

Xml-tiedosto

Edellisessä kappaleessa LAS-tiedoston lohkojen arvot oli jaettu ennalta määrättyihin muuttujiin. Tämä kuitenkin aiheuttaa ongelman, jos LAS-standardi muuttuu. Lisäksi pistetietotallenteet-lohkolla on kymmenen erilaista virallista formaattia, jotka toisin kuin otsikko, eivät ole vain laajennettuja versioita edellisestä. Näillä jokaisella formaatilla pitäisi olla oma luokkansa, koska funktion ylilataus ei Pythonissa ole mahdollista. Jotta rakenteen muuttumisen aiheuttaman ongelman voi välttää joutumatta muokkaamaan lähdekoodia, muuttujat on kerrottava ohjelman ulkopuolelta.

Tiedoston rakenne on mahdollista lukea toisesta tiedostosta. Tähän xml-tyyppinen tiedosto soveltuu hyvin, sillä toisin kuin binääritiedostoa, on sitä helppo muokata perustekstinkäsittelytyökaluilla. Seuraavassa on esimerkki xml-tiedostosta jossa on kerrottu binääritiedoston rakenne:

```
<?xml version="1.0" encoding="UTF-8"?>
<las>
    <settings>
        <endian>little-endian</endian>
    </settings>
    <header name="Public Header Block">
        <variation format="1.0">
            <variable name="File Signature">
                <type>char</type>
                <count>4</count>
            </variable>
            <variable name="File Source ID">
                <type>unsigned short</type>
                <count>1</count>
            </variable>
                <!-- Rest of the Public Header Block -->
        </variation>
        <variation format="1.1>
            <!-- Identical with format 1.0 -->
        </variation>
            <!-- Rest of the Public Header Block variations-->
    </header>
    <block name="Variable Length Records">
        <variation format="default">
            <variable name="Reserved">
                <type>unsigned short</type>
                <count>1</count>
            </variable>
            <variable name="User ID">
                <type>char</type>
                <count>16</count>
            </variable>
                <!-- Rest of the Variable Length Records -->
        </variation>
    </block>
    <!-- Rest of the las structure -->
</las>
```

On turvallista olettaa, että otsikkoja on vain ja ainoastaan yksi, ja vaikka se on erikokoinen eri versioissa, aiempien versioiden muuttujia ei ole muutettu, ainoastaan uusia arvoja on lisätty. Täten ensimmäisen formaatin otsikkoon voidaan lisätä listamaisesti uudempien formaattien muuttujia. Muissa lohkoissa näin ei voi olettaa. Lohkojen arvot voivat vaihdella suuresti riippuen formaatista. Siksi kaikki muut osiot ovat block-elementin sisällä, tämän sisällä taas variation-elementti jolla format-attribuutti.

Seuraavassa esimerkissä näkyy kuinka pistetietotallenteella on muuttujia joiden koko on yksittäisiä bittejä:

```
<variable name="Return Number">
    <type>bit</type>
    <count>3</count>
</variable>
```

Struct-moduulin muunnosarvot

Sovellus käyttäisi struct-moduulia binääriksi ja binääristä kääntäessä. Kun LAS-tiedoston rakenne on kerrottu xml-tiedostossa, sovelluksen pitää kääntää rakenne struct-moduulille sopivaksi. Käännössä voitaisiin käyttää hyväksi toista xml-tiedostoa, jossa rakenne xml-tiedoston arvot muutetaan moduulille sopiviksi:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <endian>
        <type>
            <name>little-endian</name>
            <dec>60</dec> <!-- < -->
        </type>
        <type>
            <name>big-endian</name>
            <dec>62</dec> <!-- > -->
        </type>
        <type>
            <name>native</name>
            <dec>64</dec> <!-- @ -->
        </type>
        <type>
            <name>network</name>
            <dec>33</dec> <!-- ! -->
        </type>
    </endian>
    <format>
        <name>char</name>
        <code>c</code>
        <size>1</size>
    </format>
    <format>
        <name>signed char</name>
        <code>b</code>
        <size>1</size>
    </format>
    <!-- Rest of the file -->
```

Koska pack- ja unpack-metodit käyttävät kuvion kommenteissa näkyviä erikoismerkkejä tyypitystä varten, tuottaa se ongelmia xml-tiedoston kanssa, koska ne ovat elementtien avaus- ja sulkumerkke-

jä. Tämän ongelman ohittamiseksi arvot otetaan kymmenjärjestelmämuodossa, jonka ohjelma kääntää ASCII-taulukon mukaisesti erikoismerkiksi.

Muille tyypeille, kuten char, joka on yksittäinen merkki, on moduulissa varattu kirjain c, jonka tavukoko on aina yksi.

Xml-luku

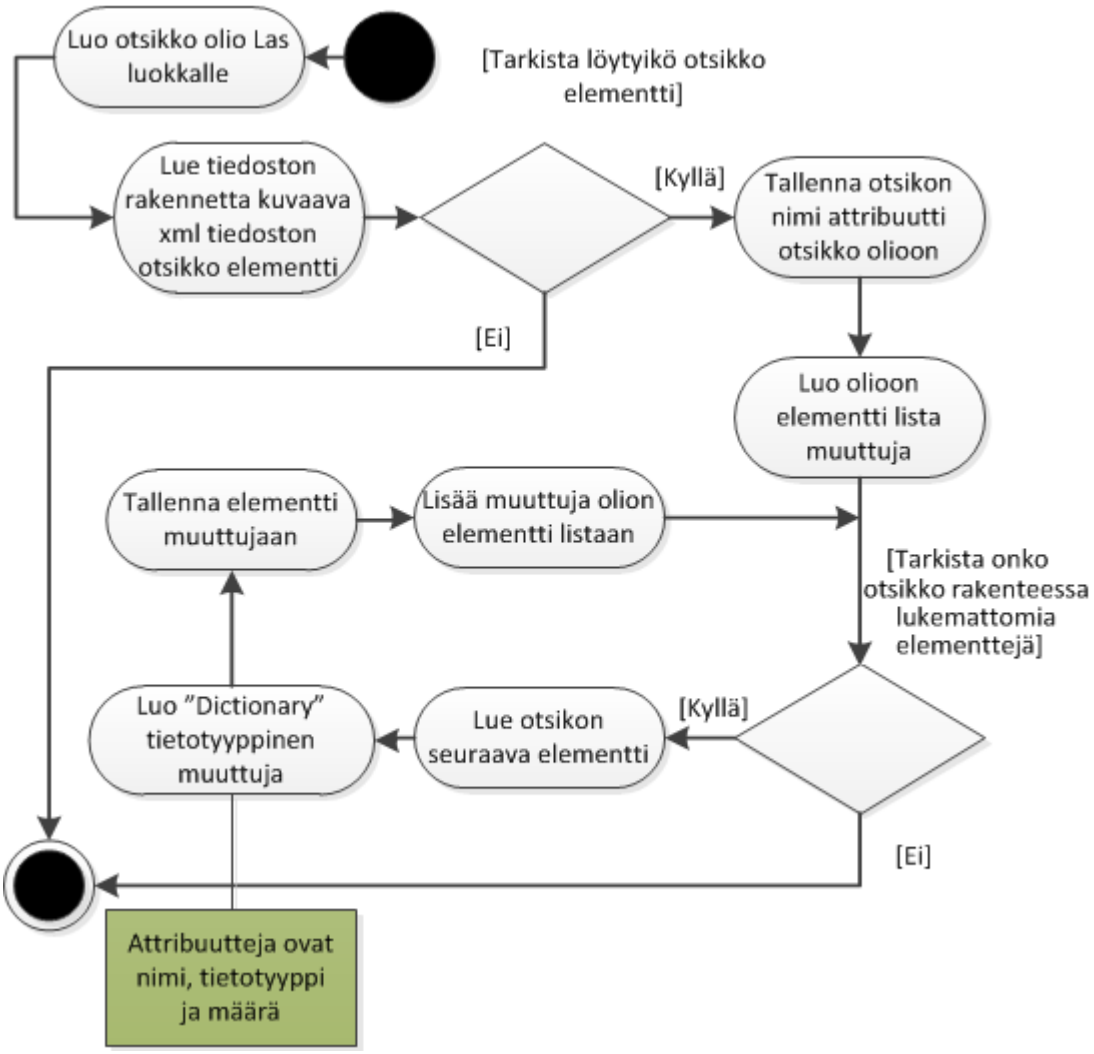
Pythonissa eräs xml-tiedoston lukemiseen ja kirjoittamiseen tarkoitettu moduuli on The ElementTree XML API. Kuviossa esimerkki moduulin käytöstä:

```
import xml.etree.ElementTree as ET

tree = ET.parse('las_structure.xml')
root = tree.getroot()
print('Endian is set as:' + root[0][0].text)
```

Edellinen koodiesimerkinsovellus lukee aiemmin esitetyn rakennetta kuvaavan xml-tiedoston, ja tulostaa juuritason ensimmäisen elementin, ja tämän ensimmäisen elementin sisällön. Tämä xml-tiedostossa arvoltaan “little-endian”.

LAS-tiedoston rakenne koottu xml-tiedostosta

Ohjelma hakisi xml-tiedostosta elementin header eli otsikon, tästä luotaisiin muuttuja LAS-luokan olioon. Olettaen että otsikkoelementti löytyi, sovellus loisi sille listan, jonne tallennettaisiin kaikki otsikkoelementin sisäiset elementit. Sovellus kävisi läpi luetun xml-tiedoston otsikon rakenteen, tallentaen sen sisällön luetussa järjestyksessä dictionary-tietotyyppeihin (ks. kuvio 12). Jokainen tietotyyppi lisättäisiin olion elementtilistaan.



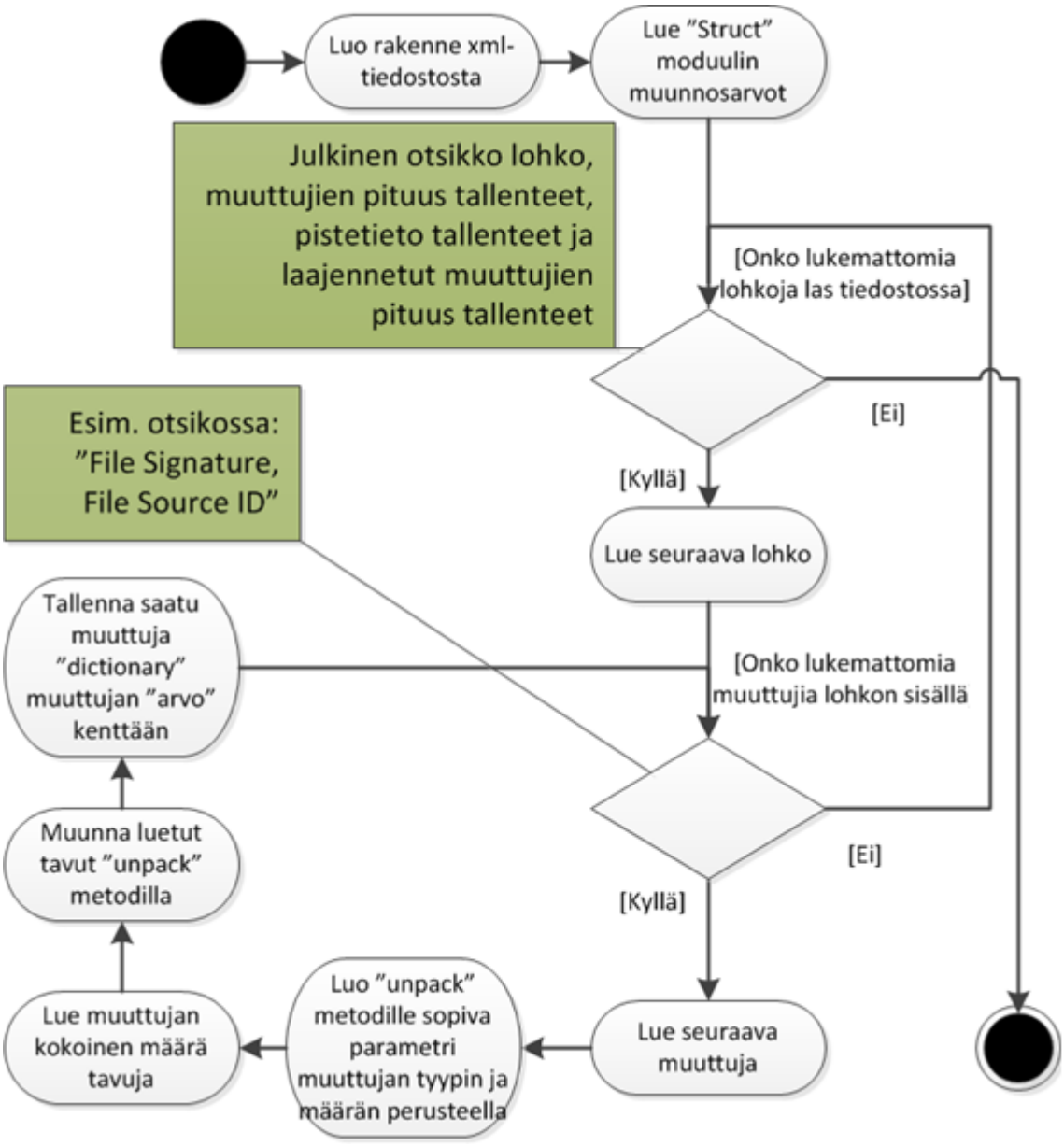
Kuvio 12. LAS-tiedoston otsikon luvun aktiviteettikaavio, käyttäen xml-tiedostosta saatua rakennetta

Tietotyyppi dictionary soveltuu tähän tarkoitukseen hyvin, koska siihen on yksinkertaista lisätä ja lukea arvoja. Toisin kuin perinteisessä taulukossa, dictionaryn taulukon avaimet voi nimetä vapaasti, kunhan ei ole kahta samannimistä avainta. Seuraavassa esimerkki dictionary-tietotyypistä:

```
#Element variables read from xml
element_name = 'example'
element_type = 'char'
element_count = 1
#Creates new dictionary data structure
element_dictionary = {'name': element_name, 'type': element_type,'count' : element_count, 'value': None}

# Would print 'example' to console
print(element_dictionary['name'])
```

Käyttäen samaa periaatetta kuin otsikossa voidaan lukea xml-tiedostosta kaikki muutkin LAS-tiedoston lohkot. Ainoa eroavaisuus on, että muita lohkoja voi olla useita, kun taas otsikkolohko on aina yksin. Tämä ei kuitenkaan ole ongelma, vaan LAS-luokan olioon luodaan listoja, jotka sisältävät

lohkojen omat oliot nimen perusteella järjestettynä. Listat voivat olla myös dictionary-tietotyypin sisällä. Kuviossa 13 on LAS-tiedoston lukemista osoittava kaavio.



Kuvio 13. Koko LAS-tiedoston luku xml-tiedostosta aktiviteettikaavio

Ensimmäinen vaihe on aiemmin esitetty tiedoston rakenteen luonti xml-tiedostosta. Toisena luetaan struct-moduulille tarvittavat muunnosarvot, jotka muuntavat tavut sovelluksen ymmärtämään muotoon. Näiden jälkeen ohjelma toistaa lukuprosessia kunnes rakenne on käyty täysin loppuun.

Yhteenveto

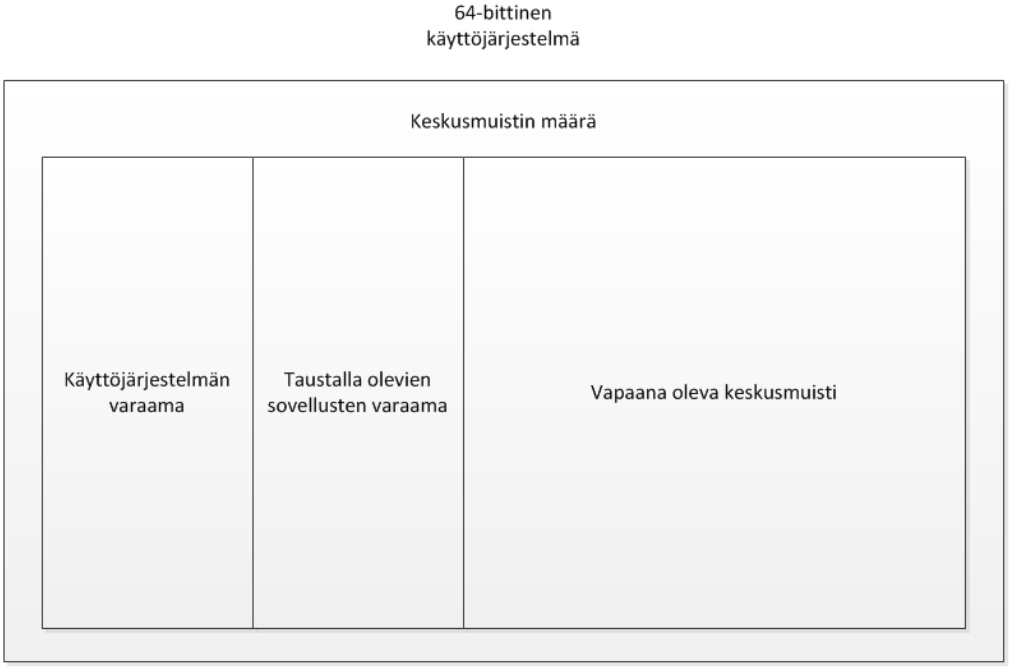
Pythonin kaltaisessa tulkatussa ohjelmointikielessä, jossa lähdekoodi ei ole käännettyä ja on jokaisen helposti luettavissa, ei kyseiselle rakenteelle ole käytännön hyötyä. Tämä rakenne lisää ohjelmaan turhaa monimutkaisuutta ja vain heikentää suoritustehoa. Tämän lisäksi ohjelmaa pitää joka tapauksessa muokata, mikäli LAS-tiedostorakenne muuttuu tulevaisuudessa, ja uusia arvoja pitäisi pystyä käsittelemään. Kyseinen rakenne siis vain siirtää tätä ongelmaa.

4.5 Muistinhallinta

Pohdinta

LAS-tiedosto, varsinkin standardin 1.4 mukainen tiedosto, voi olla erittäin suurikokoinen (Las specification version 1.4 - R13, 8). Jotta sovellus toimisi mahdollisimman monissa eri tietokoneissa, on keskusmuistin rajallinen määrä otettava huomioon. On siis vältettävä tilannetta jossa sovelluksen muistinvaraus ylittäisi sovelluksen käytössä olevaa keskusmuistin määrän. Aineisto on siis ladattava osissa. Helpoiten tämä onnistuu kun jakaa LAS-tiedostot ennalta määrätyn kokoisiksi.

Sovelluksella ei ikinä ole koko keskusmuistia yksin käytettävissä. Käyttöjärjestelmä itsessään vie jo osan. Myös taustalla olevat sovellukset vievät muistia (ks. kuvio 14). Mitä jää jäljelle, on sovelluksella käytettävissä, olettaen että halutaan välttää keskusmuistin korvaamista kiintolevyllä sen täyttyessä.



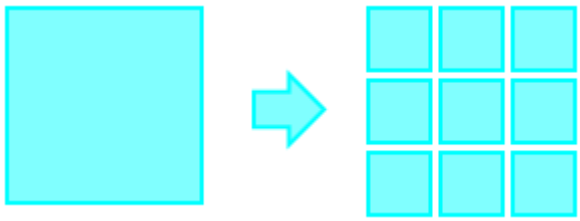
Kuvio 14. Sovelluksen käytössä oleva keskusmuistin määrä

Mainittakoon myös että, käyttöjärjestelmän arkkitehtuureista X86 eli 32-bittisessä käyttöjärjestelmässä on omat rajoituksensa keskusmuistin suhteen. Tietyissä koneissa käyttöjärjestelmästä ja komponenteista riippuen keskusmuistin raja oli noin kolmessa gigatavussa (3 GB Barrier, N.d.).

Alueen rajaus

Maanmittauslaitoksen LAS-tiedostot ovat kooltaan yli 800 megatavua. Mikäli tiedosto koodataan ASCII-muotoon, tiedoston kooksi tulee hieman vajaa kaksi gigatavua. Ottaen huomioon, että kun tiedosto on ladattu keskusmuistiin, sovellus tarvitsee silti vielä pitää muistissa suorituksen aikaisia muuttujia.

Muodostamalla yksittäisestä LAS-tiedostosta helpommin käsiteltäviä ja ennalta määrätyn kokoisia pieniä tiedostoja yksinkertaistaa ohjelmaa, koska voimme luottaa että tiedosto mahtuu keskusmuistiin (ks. kuvio 15). On myös eduksi järjestää pienet tiedostot koordinaattien perusteella, jotta tietyn alueen paikantaminen on nopeaa.



Kuvio 15. Tiedoston jako

Tiedostojen hallintaan soveltuu xml-tiedosto, jossa on kerrottu mistä kukin tiedosto on otettu. Xml-tiedosto voisi myös kertoa tiedoston alueen rajojen koordinaatit. Kun käyttäjä haluaisi tiettyjen koordinaattien pisteosumat, sovellus kävisi läpi xml-tiedoston ja palauttaisi alueen sisällä olevat LAS-tiedostot käsiteltäviksi. Etuna tässä olisi, että mikäli haluttu alue on hyvin pieni, turhan aineiston käsittely olisi vähäistä verrattuna siihen, että koko LAS-tiedosto käsiteltäisiin aina.

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
    <cell>
        <base name="N4324D1.las">
            <latitude min="6900024" max="6903000"/>
            <longitude min="433992" max="496968"/>
        </base>
        <subcell name="N4324D1_0.las">
            <latitude min="6902009" max="6903000"/>
            <longitude min="433992" max="454984"/>
        </subcell>
        <subcell name="N4324D1_1.las">
        <!-- Rest of the file -->
```

Edellisillä riveillä on esimerkki, miltä xml-tiedosto voisi näyttää. Tiedostossa olisi base-elementti, joka olisi alkuperäinen LAS-tiedosto. Tällä elementillä olisi kaksi omaa elementtiä, joissa olisi alueen leveyspiirin ja pituuspiirin rajat. Elementin base lisäksi olisi useita subcell-elementtejä, jotka olisi tehty jakamalla alkuperäinen tiedosto osiin. Näilläkin olisi leveyspiirin ja pituuspiirin rajat elementteinä.

4.6 Metsäalueiden graafinen esitys

Metsäkeskus on myös rajannut eri omistuksessa olevat metsäalueiden rajat xml-tiedostoon. Tiedostossa on järjestyksessä pistesarja koordinaatteja, jotka yhdistettäessä muodostavat kyseisen alueen rajauksen. Seuraavassa esimerkki otos xml-tiedostosta:

```
<gml:polygonProperty>
  <gml:Polygon srsName="EPSG:3067">
    <gml:exterior>
      <gml:LinearRing>
        <gml:coordinates>350632.8964,6923825.5653 350916.8832,6924047.7001
      </gml:LinearRing>
    </gml:exterior>
  </gml:Polygon>
</gml:polygonProperty>
```

Osana tätä opinnäytetyötä kirjoitettiin työryhmälle xml-tiedostoja lukeva sovellus, joka palauttaa leveys- ja korkeuskoordinaatit omina listoinaan. Ohjelma hakee valitun stand number -alueen koordinaatit ja palauttaa ne kahtena listana.

```
import xml.etree.ElementTree as ET
def get_coordinates_from_xml(file_path, stand_number):
    data = []
    x = []
    y = []
    st = "{http://standardit.tapio.fi/schemas/forestData/stand/2010/08/31}"
    gdt = "{http://standardit.tapio.fi/schemas/forestData/"
    gdt += "common/geometricDataTypes/2010/08/31}"
    gml = "{http://www.opengis.net/gml}"
    findall_string = st+"Stands/"+st+"Stand/"+st+'StandBasicData'
    find_string = gdt+'PolygonGeometry/'+gml+'polygonProperty/'
    find_string += gml+'Polygon/'+gml+'exterior/'
    find_string += gml+'LinearRing/'+gml+'coordinates'
    tree = ET.parse(file_path)
    root = tree.getroot()

    for e in root.findall(findall_string):
        if int(e.find(st+ 'StandNumber').text) == stand_number:
            data = e.find(find_string).text
            data = data.split(' ')
    for i in data:
        tmp = i.split(',')
        x.append(float(tmp[0]))
        y.append(float(tmp[1]))
    return x, y

x, y = get_coordinates_from_xml('K249A558MS56.xml', 79)
```

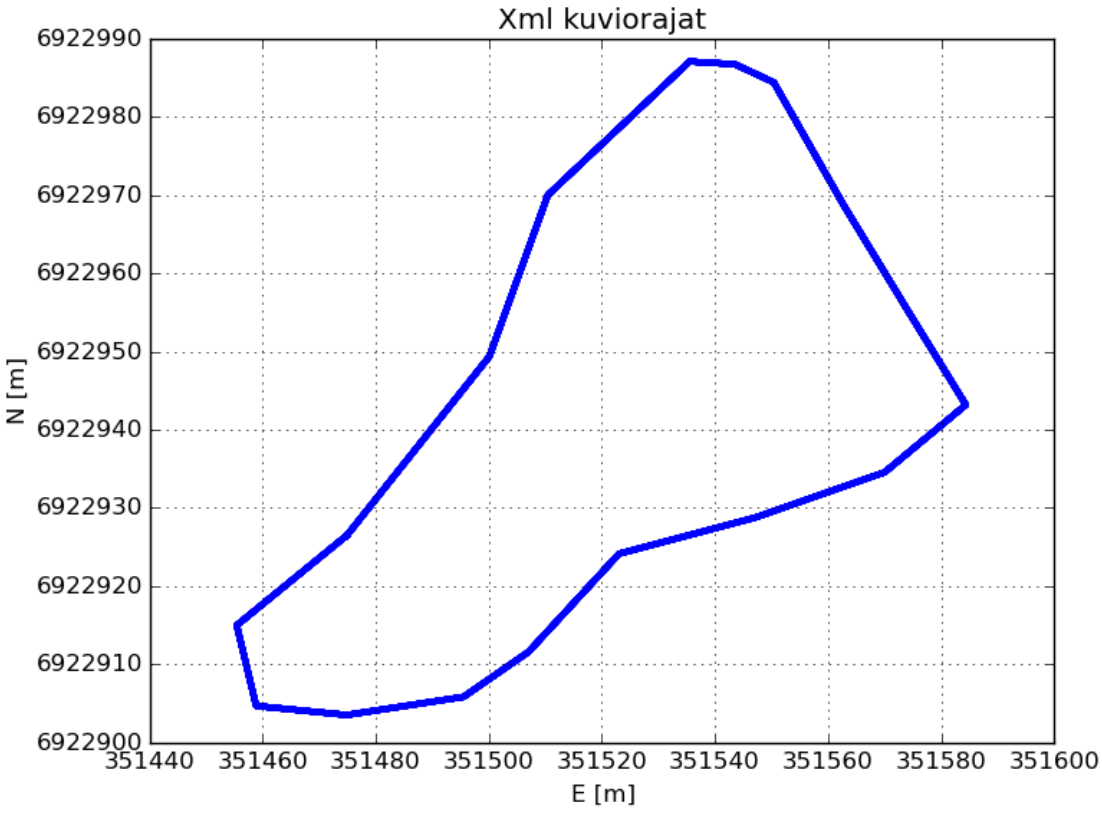
Edellisen koodiesimerkin xml-elementtien hakumuuttuja find_string on jaettu monelle riville luettavuuden vuoksi. Huomioitavaa tässä on, että funktio palauttaa kaksi muuttujaa kerralla. Kumpikin lista sisältää koordinaatit yksinkertaisen tarkkuuden liukulukuna eli float-tietotyyppinä. Esimerkkitulostukseen on lisätty X: ja Y: -merkit luettavuuden vuoksi:

```
X: 351781.2044  Y: 6923039.7453
X: 351777.7659  Y: 6923051.9428
X: 351772.538   Y: 6923068.1295
X: 351769.083   Y: 6923080.327
X: 351777.4113  Y: 6923086.9857
X: 351786.9682  Y: 6923099.5031
X: 351789.7883  Y: 6923116.4397
X: 351786.4323  Y: 6923130.6468
X: 351780.083   Y: 6923144.8739
X: 351777.8401  Y: 6923161.0306
```

Kun koordinaatit on haettu, ja halutaan esittää alueen koordinaatit graafisesti, onnistuu sekin Matplotlib-moduulin avulla. Edellisen koodiesimerkin sovelluksen palauttamat listat syötetään seuraavalle Matplotlib-moduulia hyödyntävälle sovellukselle:

```
import matplotlib.pyplot as plt
import matplotlib.ticker as tic

fig, ax = plt.subplots()
formatter = tic.ScalarFormatter(useOffset=False)
ax.xaxis.set_major_formatter(formatter)
ax.yaxis.set_major_formatter(formatter)

x_list,y_list = get_coordinates_from_xml('file.xml', area_id)
for k in range(len(x_list)):
    plt.plot(x_list, y_list, '-', color='blue',  markersize=5.0, linewidth=3)

plt.grid(True)
plt.title("Xml kuviorajat")
plt.ylabel('N [m]')
plt.xlabel('E [m]')

plt.show()
```

Edellisessä koodiesimerkissä esitetty sovellus piirtää viivat peräkkäisten pisteiden välille, muodostaen täten alueen muodon. Pysty- ja vaaka-akseleilla on koordinaatit. Kuviossa 16 on esimerkki alueen rajauksen graafisesta esityksestä.



Kuvio 16. Kuvion koordinaatit

Liitteessä 1 on esimerkki opinnäytetyön tuotoksesta metsävaraprojektiin. Tuotoksessa näkyvät kuviorajat ja metsänhoidon kehitysluokat *Paananen, Pellinen, Pietikäinen, Sauranen, Savolainen, Viitala, Weckström & Väänänen. 2014, 46).*

4.7 Käsitellyn aineiston esittäminen ArcGIS-työkalulla

Koska Metsäkeskus käytti ArcGIS-työkalua aineiston analysointiin ja graafiseen tutkimiseen, olisi sovelluksen pystyttävä siirtämään aineisto kyseiselle työkalulle. Siirto tapahtuisi tehokkaimmin binääriformaatissa, tämän pienemmän tiedostokoon vuoksi verrattuna ASCII-koodattuun tiedostoon.

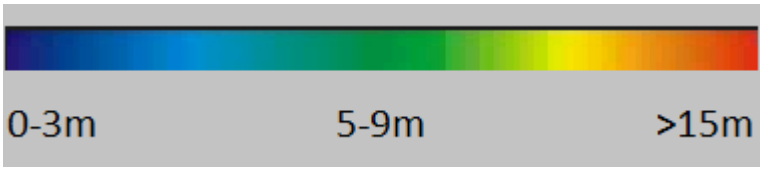
ArcGIS-työkalu pystyy lukemaan suoraan LAS-formaattia; työkalu tukee LAS-versioita 1.0, 1.1, 1.2 ja 1.3. Näiden lisäksi työkalu tukee pistetietotallenneformaatteja 0 – 5 (Storing lidar data, N.d.).

Pistetietotallenteiden formaatit 2 ja 3 soveltuvat kuvion esittämiseen, koska kyseiset formaatit sisältävät kahden tavun muuttujat punaiselle, vihreälle ja siniselle värille (ks. taulukko 5). Värien lisäksi tarvitaan vain koordinaatit, jotka kummassakin formaatissa on. Käyttäjille kerrottaisiin, mitä kukin väri tarkoittaa (ks. kuvio 17).

Taulukko 5. Pistetietotallenneformaatti kaksi

Point Data Record Format 2

Osio	Tyyppi	Koko	Pakollinen
X	long	4 tavua	Kyllä
Y	long	4 tavua	Kyllä
Z	long	4 tavua	Kyllä
Intensity	unsigned short	2 tavua	Ei
Return number	3 bittiä (bitit 0 - 2)	3 bittiä	Kyllä
Number of Returns (given pulse)	3 bittiä (bitit 3 - 5)	3 bittiä	Kyllä
Scan Direction Flag	1 bitti (bitti 6)	1 bitti	Kyllä
Edge of Flight Line	1 bitti (bitti 7)	1 bitti	Kyllä
Classification	unsigned char	1 tavu	Kyllä
Scan Angle Rank (-90 to +90) - Left side	char	1 tavu	Kyllä
User Data	unsigned char	1 tavu	Ei
Point Source ID	unsigned short	2 tavua	Kyllä
Red	unsigned short	2 tavu	Kyllä
Green	unsigned short	2 tavua	Kyllä
Blue	unsigned short	2 tavua	Kyllä



Kuvio 17. Esimerkki pituusarvojen värikoodeista

Käsiteltyään aineiston sovellus siis loisi uuden LAS-tiedoston, jossa pistetietotallenteet olisivat joko formaatissa kaksi tai kolme. Tämä LAS-tiedosto lähetettäisiin ArcGIS-työkaluun, joka esittäisi sen graafisesti. ArcGIS täyttäisi alueen erivärisillä pisteillä, joista pystyisi ihmissilmällä nopeasti selaamaan missä on metsänhoitotarvetta ja missä ei. Sovelluksen tekijän tulisi ottaa huomioon, että käyttäjä voi haluta vaihtaa väriarvoja esimerkiksi värisokeuden takia.

5 Jatkokehitys

Heti sovelluksen teon alussa olisi pitänyt päästä lähemmäs ArcGIS-yhteensopivuutta. Nykyinen sovellus käytti Matplotlib-moduulia kaavioiden piirtämiseen, kun taas Metsäkeskus käytti ArcGIS:a aineiston katsomiseen. ArcGIS on huomattavasti laajempi ohjelmisto kuin mitä Matplotlib ja koska tämä oli jo asiakkaan ennestään tuntema sovellus, tulisi sitä käyttää datan esittämisessä.

Projektin kanssa pitäisi lähteä aivan perusteista liikkeelle ja samalla voisi vaihtaa toiseen ohjelmointikieleen, jossa on vahva tyypitys. Vahvalla tyypityksellä voidaan aina varmasti sanoa minkä tyyppinen kukin arvo on. Pythonin dynaamisella tyypityksellä tämä on mahdollista, mutta vaatii tyyppimuunnoksia. Tämän lisäksi saataisiin todennäköisesti laskentanopeutta lisättyä. Oletus on että C++ sovellus olisi tehokkaampi kuin Python sovellus. Alun perin Python ohjelmointikieli valittiin, koska ArcGIS työkalussa on olemassa Python ohjelmointirajapinta (ArcGIS Resources, N.d). Tätä ominaisuutta ei kuitenkaan ole koskaan käytetty hyväksi. Vahvalla tyypityksellä tai ilman, ohjelmointikieltä valittaessa pitää selvittää, minkälaiseen käyttöjärjestelmäympäristöön sovellus menisi.

Tämän hetkinen sovellus ei ole helposti ylläpidettävä koodin vaikealukuisuuden vuoksi. Ohjelmistosovellukseen tarvitaan selkeä rakenne, tähän voisi sopia esimerkiksi malli-näkymä-käsittelijä -arkkitehtuuri. Huonon ylläpidon lisäksi nykyisessä sovelluksessa ei ole käytännössä minkäänlaista käyttöliittymää. Ohjelmointikielen vaihtoa harkittaessa on hyvä selata käytössä olevia graafisen käyttöliittymän kirjastoja, kuten esimerkiksi .NET-sovelluskehyksen wpf-kirjasto. Käytännössä nämä tarkoittaisivat, että projekti lähtisi nollasta liikkeelle, mutta nykyisestä projektista tulisi ottaa mallinnusalgoritmit ja siirtää ne uuteen ympäristöön.

Tulisi myös tutkia tarjolla olevia LAS-tiedoston lukijoita ja kirjoittajia. Olisi hyvä jos sovellus pystyisi myös sekä lukemaan ja kirjoittamaan LAZ-formaattia. LAS/LAZ-tiedoston kirjoitus pitää tukea pistetietotallenteiden formaatteja kaksi ja kolme, jotta värit voidaan syöttää tiedoston mukana suoraan ArcGIS-työkaluun.

Testauksessa olisi tarvittu ArcGIS-lisenssiä. Työryhmässä työskennellessä tätä ei ollut. Vähäisen testauksen vuoksi jälkeenpäin kävi ilmi, että ArcGIS työkalulle mieluisin formaatti on Shape. Tämä kävi ilmi kun tiedostoja siirrettiin ArcGIS-työkaluun jatkokäsiteltäväksi rasteripintoina. LAS-formaatista voidaan siirtää tietoa Shape-formaattiin. Shape on laajempi ja voi sisältää vektoritietoa attribuutteineen. Täten sitä voidaan käsitellä rastereina eli ruutupohjaisena.

6 Pohdinta

Työtä rajasivat työryhmän tekemä sovellus, joka oli osittain toimiva. Jotta tämä työ toisi uutta, piti työ rajata työryhmän sovelluksen ympäriltä, että se sopisi työryhmän sovellukseen. Tavoitteina oli, että LAS-tiedosto saadaan luettua, käsiteltyä ja kirjoitettua ArcGIS-yhteensopivana LAS-formaattina. Koska käsittely oli jo tehty nykyisessä sovelluksessa, keskityin lukemiseen ja kirjoittamiseen. Tämä teki valtavan rajoituksen tekemiseen, ja näin jälkeenpäin voi todeta että se oli virhe.

Tuloksena suunniteltiin LAS-tiedostolle lukija, joka pystyisi lukemaan minkä tahansa kokoisen tiedoston. Toinen tulos oli xml-tiedostosta Matplotlib-moduulia apuna käyttävä kuviorajojen piirto-ohjelmakoodi. Kolmas tulos oli selvitys ASCII- ja LAS- koodien luku- ja kirjoitusnopeuden eroista. Lisäksi perehdyttiin aineiston siirtämiseen ArcGIS-työkaluun.

Vielä mahdollisesti tulossa oleville LAS-formaateille lukijan suunnittelu oli täysin turhaa työtä. Suunnitteluvaiheessa idea vaikutti lupaavalta, mutta käytännön toteutuksen jälkeen tuli selväksi, että kyseinen toteutus vain siirtää ongelman eteenpäin. Vaikka lukija saisikin tiedoston luettua, niin käsittelijän pitäisi myös tietää, mitä lukija on lukenut. Samalla toteutus tekee lukemisesta vain hitaampaa.

Projektin ongelma on, että nykyinen sovellus on erittäin hankalasti ylläpidettävä sen rakenteen puutteen takia. Sen sijaan, että parannettaisiin luettavuutta koodia muokkaamalla, voitaisiin aloittaa projekti alusta. Tällä hetkellä Matplotlib-moduulin ainoa käyttötarkoitus on ohjelmistotestauksessa, koska työryhmälle ei ole ArcGIS-lisenssiä. Koodikieli voisi olla C++:aan tai C#:iin, riippuen mitä käyttöjärjestelmää Metsäkeskus käyttää palvelimissaan. Tulisi myös etsiä internetistä mahdollista kriteerit täyttävää valmista LAS-tiedoston luku- ja kirjoituskirjastoa.

Nykyisellään ainoastaan kuviorajojen piirto-ohjelmaa xml-tiedostosta käytetään sovelluksessa. Muut opinnäytetyössä esitetyt ratkaisut tulisi alusta aloitettavaan sovellukseen.

Lähteet

3 GB Barrier, Wikipedia artikkeli. N.d. Viitattu 1.8.2015.
https://en.wikipedia.org/wiki/3_GB_barrier

ArcGIS Resources. N.d. Viitattu 16.8.2015
http://resources.arcgis.com/en/communities/python/

ASCII Table and Description, N.d. Viitattu 28.1.2015.
http://www.ASCIItable.com/

Avoimien aineistojen tiedostopalvelu, Maanmittauslaitos. N.d. Viitattu 28.1.2015.
http://www.maanmittauslaitos.fi/aineistot-palvelut/latauspalvelut/avoimien-aineistojen-tiedostopalvelu

Binary file – Wikipedia artikkeli. N.d. Viitattu 8.8.2015
https://en.wikipedia.org/wiki/Binary_file

Evaluation strategy – Wikipedia artikkeli. N.d. Viitattu 1.8.2015.
https://en.wikipedia.org/wiki/Evaluation_strategy#Call_by_sharing

Las specification version 1.2, julkaistu 2.8.2008. Viitattu 25.1.2015.
http://www.asprs.org/a/society/committees/standards/asprs_las_format_v12.pdf

Las specification version 1.3 - R11, The American Society for Photogrammetry & Remote Sensing, julkaissut 24.10.2010. Viitattu 25.1.2015.
http://www.asprs.org/a/society/committees/standards/LAS_1_3_r11.pdf

Las specification version 1.4 - R13. The American Society for Photogrammetry & Remote Sensing, julkaissut 15.6.2013. Viitattu 24.1.2015.
http://www.asprs.org/a/society/committees/standards/LAS_1_4_r13.pdf

LASer (LAS) File Format Exchange Activities, N.d. Viitattu 24.1.2015.
http://www.asprs.org/Committee-General/LASer-LAS-File-Format-Exchange-Activities.html

Laserkeilaustekniikka, Maanmittauslaitos. N.d. Viitattu 28.1.2015.
http://www.maanmittauslaitos.fi/kartat/laserkeilausaineistot/laserkeilaustekniikka

Maastotietoaineistot koko kansan käyttöön, Maanmittauslaitos julkaissut kolumnin 19.1.2012. Viitattu 28.1.2015.
http://www.maanmittauslaitos.fi/tiedotteet/2012/01/maastotietoaineistot-koko-kansan-kayttoon

Matplotlib, Matplotlib kotisivu. N.d. Viitattu 30.7.2015.
http://matplotlib.org/

Metsäkeskus - metsäalan asiantuntijatalo. N.d. Viitattu 28.1.2015.
http://www.metsakeskus.fi/metsakeskus#.VMkwmmjkfzk

Paananen, R., Pellinen, R., Pietikäinen, K., Sauranen, T., Savolainen, V., Viitala, R., Weckström, P. & Väänänen, O. 2014. Uusia menetelmiä metsävaratietojen hyödyntämiseen.
Julkasun pysyvä osoite: http://urn.fi/URN:ISBN:978-951-830-336-0

Puuston kehitysluokat – Suomen metsäurheiluliitto r.y. N.d. Viitattu 8.8.2015.
http://www.helsinki.fi/hyytiala/smul/saannot/dokumentit/saannot_kehitysluokat_watermark.pdf

Python, Python kotisivu. N.d. Viitattu 28.1.2015.
https://www.python.org/

Python (programming language) - Wikipedia artikkeli. N.d. Viitattu 28.1.2015.
http://en.wikipedia.org/wiki/Python_%28programming_language%29

Storing lidar data - ArcGIS help 10.1, ArcGIS ohjesivu. N.d. Viitattu 13.4.2015.
http://resources.arcgis.com/en/help/main/10.1/index.html#//015w00000054000000

Liitteet

Liite 1. Lapinmäen esimerkkipiirto

Lapinmäen metsätilan kuviorajat sekä taimikoiden ja nuorien metsienhoitotarpeet vuonna 2013.