

# Tiralabran viikkoraportti #1

## Kuvan pakaus

16.5.2013

Mikko Kangasmäki

Toistaiseksi en ole kirjoittanut yhtään riviä valmista koodia, vaan selvitellyt bmp-tiedostojen muotoa ja kuvankäsittelyä javalla.

Aluksi käytin huomattavasti aikaa kuvan lukemiseen bittitasolla ImageInputStream-luokan avulla. Yritin päästää käsiksi pikselidataan, ja kopioida kuvan toiseen tiedostoon sitä kautta. Kun osaisin tehdä sen, voisim käsittellä dataa siinä välissä. Lopulta onnistuin tässä, mutta se oli työlästä sekä minulle, että koneelle, jolta kuvan kopiointi toiseen tiedostoon kesti n. 50 sekuntia.

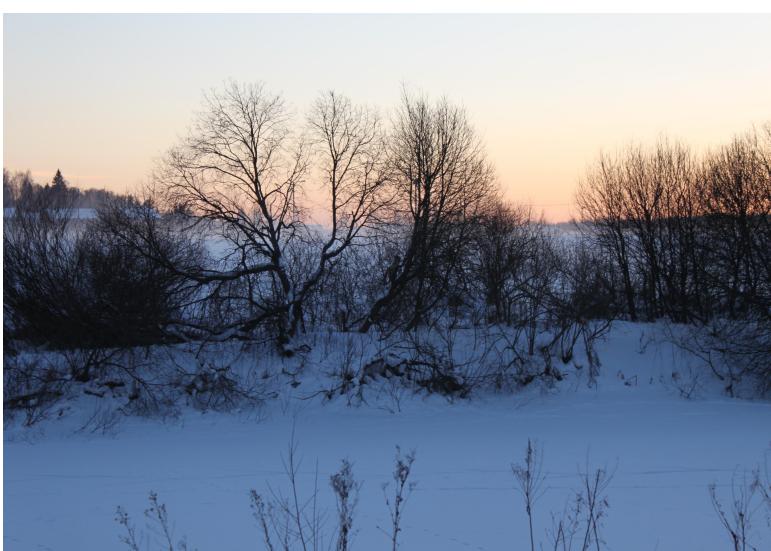
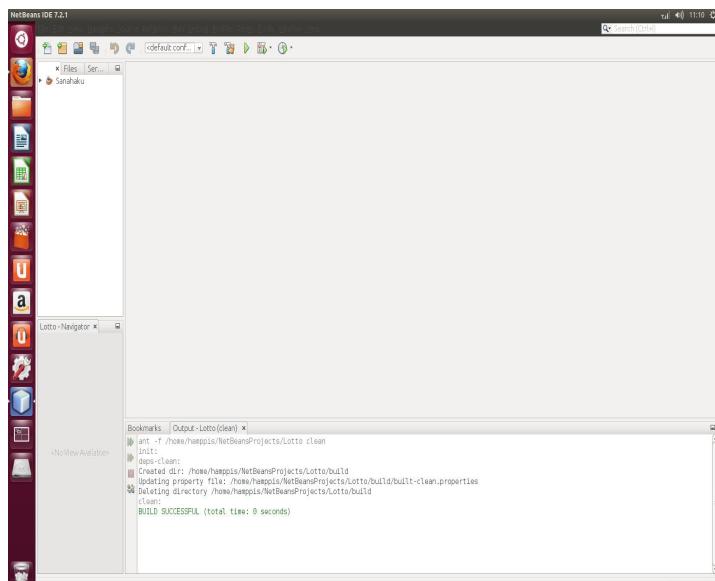
Vaiva ei kuitenkaan ollut hukkaan heitettyä, sillä sen avulla opin ymmärtämään paremmin bmp-tiedostomuotoa ja vastaan tulleet ilmiöt antoivat ajattelemisen aihetta koko projektista.

Minulla oli, ja on vieläkin, tiettyjä epäilyjksiä aalokkeiden käytön mielekkyydestä, sillä lopputuloksesta saattaa helposti tulla triviaali tai liian vaikea, eli totetutan vain jonkin muun kirjoittaman algoritmin tai yritän itse tehdä jotain, missä en onnistu. Kuvan pikselidataa käsitellessä aloin miettiä joitakin yksinkertaisempia pakkaamismuotoja: esimerkiksi, että ilmoitettaisiin väreille alue, joka väritetään, tai sitten unohdetaan vähän esiintyviä värisävyjä.

Laskin koneella kultakin kuvan pikseliriviltä siinä esiintyvien punaisen sävyjen määrän. Ensimmäinen testikuva on screenshot netbeansista. Erilaisia punaisen sävyjä esiintyy sen erään satunnaisen rivin 1680 pikselissä tällaisella jakaumalla:

Luvut siis kertovat, että eniten esiintynyt punainen sävyä oli testirivillä 1410 pikselissä. Toiseksi esiintyneintä 205 pikselissä jne. Yhteensä rivillä siis esiintyy vain 48 kaikista 256 punaisen sävystä. Screenshotista tällaista yhdenmukaisuutta voi tietenkin odottaakin, mutta toisessa testikuvassani, joka on talvimaisema, on myös aika vähän eri sävyjä (leveys 3456 pikseliä):

215 131 107 102 90 85 85 84 77 65 55 53 30 30 29 28 28 27 25 25 24 24 24 23 23 23 22 22 22 21  
21 21 21 20 20 20 19 19 19 19 19 19 17 17 17 17 17 17 17 17 17 17 17 17 17 16 16 16 16 16 16 16  
16 16 16 15 15 15 15 14 14 14 14 14 14 14 14 14 14 14 14 14 14 13 13 13 13 13 13 13 13 13 13  
13 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 11 11 11 11 11 11 11 11 11 11 11 11  
11 11 11 11 10 10 10 10 10 10 10 10 10 10 9 9 9 9 9 8 8 8 8 8 8 7 7 7 7 7 7 7 7 7 7 7 6 6  
6 6 6 6 5 5 5 5 5 5 4 4 4 3 3 2 2 2 2 2 1 1 1 1 1 1 1 1



Näyttäisi järkevältä tiivistää tiedosta unohtamalla värisävyt, joita esiintyy vain pienessä määrässä pikseleitä. Esimerkiksi jättämällä pois kaikki sävyt, joita on alle kymmenessä pikselissä, saadaan sävyjen määrä pienentämään 140:een. Vielä parempi olisi, jos sävyn poisjättäminen voitaisiin ratkaista lähekkäin olevien pikselien määrellä mieluummin kuin niiden kokonaismäärellä.

Tässä vaiheessa kuulin BufferedImage-luokasta, joka olikin huomattavasti nopeampi ja helppokäytöisempi kuin aiemmat työkaluni.

Aloin kokeilumielessä toteuttaa ohjelmaa, joka lukee riviltä värisävyn (vaikkapa punainen) taulukkoon `int[] värit = new int[256];`. Taulukon arvo `värit[n]` kertoo, kuinka monessa pikselissä rivillä esiintyy punaisen sävyä `n`.

Seuraavaksi ohjelma kirjoittaa kohdekuvan pikseleihin värit esiintymismäärien mukaisessa järjestyksessä suurimmasta pienimpään. Voidaan asettaa jokin minimimääriä, jota vähemmän esiintyneitä sävyjä ei kirjoiteta ollenkaan, ja näin säästetään tilaa, ainakin periaatteessa.

Toistaiseksi ohjelma ei ole vielä toiminut aivan toivotulla tavalla:



Seuraavaksi ajattelin:

1. palauttaa mieleeni aalokkeet
2. selvittää, miten niitä voisi käyttää oikean maailman sovelluksissa
3. selvittää tarkkaan, miten BufferedImage-värit on koodattu (tästä minulla on hajua)
4. kirjoittaa lopullisesta ohjelmasta osat, joissa luetaan kuva levyltä ja kirjoitetaan se levylle.