Spektrikuvaformaatit

Standardin määrittelemisen alkuvaiheet

Tuija Jetsu

7.9.2004

Joensuun yliopisto
Tietojenkäsittelytieteen laitos
Pro gradu -tutkielma

Tiivistelmä

Perinteiset digitaalikuvien esitysmuodot eivät pysty takaamaan väri-informaation säilymistä samanlaisena eri näyttölaitteissa ja erilaisissa valaistuksissa. Perinteisten esitysmuotojen lisäksi värikuva voidaan tallentaa myös spektrikuvana. Tällöin kuvan esitys ei ole riippuvainen mistään tietystä värikoordinaatistosta, valaistuksesta tai tarkastelijasta, vaan spektreistä voidaan aina laskea halutunlainen värikoordinaattiesitys tarkasteluympäristön ominaisuudet huomioiden. Spektrikuvia voidaan hyödyntää esimerkiksi lääketieteen, tähtitieteen, maatalouden ja kaukokartoituksen sovelluksissa. Eräs merkittävä spektrikuvien sovelluskohde on myös värintoiston parantaminen tietotekniikan eri osa-alueilla kuten näyttölaitteissa ja paperitulosteissa.

Vaikka spektrikuvia tutkitaan useissa eri paikoissa, on kuvien vaihto tutkimusryhmien välillä ollut vähäistä. Yksi merkittävä syy tähän lienee yhteisen tallennusformaatin puuttuminen. Kansainvälisen valaistuskomitean (CIE, Commission Internationale de l'Eclairage) 8. jaosto on keskittynyt kuvateknologiaan liittyviin asioihin. Jaoston alaisuudessa toimii myös useita teknisiä komiteoita, joista yksi, TC8-07, on keskittynyt spektrikuviin liittyviin asioihin. Parhaillaan kyseinen komitea työskentelee tavoitteenaan määritellä standardi spektrikuvien tiedostoformaatiksi.

Tässä tutkielmassa tutustutaan aluksi standardien kehittämiseen yleisellä tasolla. Sen lisäksi tutkitaan erilaisia kuvaformaatteja ja perhedytään asioihin, joita tulee ottaa huomioon formaattien suunnittelussa. Osa tämän tutkielman kuvaformaateista on alun perin luotu perinteisten digitaalikuvien tallentamiseen ja osa on suoraan kehitetty spektrikuvia varten. Formaatit ovat CIE:n TC8-07-työryhmän jäsenten esille tuomia. Tutkielmassa pohditaan myös, mitä ominaisuuksia spektrikuvasta tulisi tallentaa tiedostoon. Tutkielmassa esiteltyjä kuvaformaatteja vertaillaan spektrikuvan perusominaisuuksien osalta. Perusominaisuuksiltaan eri kuvaformaatit ovat pitkälti samanlaisia, suurin ero eri formaattien välillä ovat sovelluskohtaiset ominaisuudet. Juuri nämä sovelluskohtaiset ominaisuudet ovatkin myös standardoinnin hankalin asia, sillä jokaisen kuvaformaatin kaikkien sovelluskohtaisten ominaisuuksien huomioiminen standardissa on käytännössä mahdotonta.

ACM-luokat (ACM Computing Classification System, 1998 version): H.3.2, I.4.9, I.4.10

Avainsanat: spektrikuva, standardi, kuvatiedosto, kuvaformaatti

Sisältö

Liitteet

Liite 1: CIE:n TC8-07-työryhmän taulukko kuvaformaattien ominaisuuksista.

Liite 2: MUSP-tiedostoformaatin luku- ja kirjoitusrutiinien lähdekoodi.

Liite 3: Natural Vision -tiedostoformaatin lukurutiinin lähdekoodi.

Liite 4: Binaaritiedostoformaatin luku- ja kirjoitusrutiinien lähdekoodi.

1 Johdanto

Tekniikan kehittyessä on noussut esiin tarve esittää kuvien väri-informaatiota entistä tarkemmin muun muassa tietokoneen näytöllä, television ruudulla ja painetuissa tuotteissa. Suurimmassa osassa kuvaformaatteja kuvan arvot esitetään koordinaatteina jossakin kolmikomponenttisessa väriavaruudessa. Väriavaruuksia on olemassa useita erilaisia. Jokin tietty kolmikomponenttinen väriavaruus toimii optimaalisesti vain tietyissä olosuhteissa, esimerkiksi tietyllä tavalla säädettyä näyttölaitetta käytettäessä. Tästä seuraa, että perinteiset digitaalikuvien esitysmuodot eivät pysty takaamaan väri-informaation säilymistä samanlaisena eri näyttölaitteissa ja erilaisissa valaistuksissa. Kolmikomponenttisten esitysten lisäksi värikuva voidaan tallentaa myös spektrikuvana. Tällöin kuvan esitys ei ole riippuvainen mistään tietystä värikoordinaatistosta, valaistuksesta tai tarkastelijasta, vaan spektreistä voidaan aina laskea halutunlainen värikoordinaattiesitys. Värikoordinaattiesitystä laskettaessa voidaan huomioida myös tarkasteluympäristön ominaisuudet.

Spektrikuvia käytetään moniin erilaisiin tarkoituksiin. Niitä voidaan hyödyntää esimerkiksi lääketieteen, tähtitieteen, maatalouden ja kaukokartoituksen sovelluksissa. Eräs merkittävä spektrikuvien sovelluskohde on myös värintoiston parantaminen tietotekniikan eri osa-alueilla kuten näyttölaitteissa ja paperitulosteissa. Vaikka tarkasteltavat ominaisuudet vaihtelevat sovelluskohteittain, kaikissa sovelluksissa pyritään spektrimittauksia käyttäen analysoimaan tarkasti kohteen ominaisuuksia.

Erilaisia spektrikuvien tallennusformaatteja on olemassa lähes yhtä monta kuin spektrikuvia tutkivia ryhmiäkin. Vaikka spektrikuvia tutkitaan useissa eri paikoissa, on kuvien vaihto tutkimusryhmien välillä ollut melko vähäistä. Yksi merkittävä tekijä vaihdon vähäisyyteen lienee yhteisen tallennusformaatin puuttuminen. Kun jokainen ryhmä käyttää omaa formaattiaan spektridatan tallentamiseen, uuden formaatin kohdalla on aina ensimmäiseksi selvitettävä, mitä kyseinen formaatti pitää sisällään. Lisäksi on selvitettävä, kuinka uudessa formaatissa olevasta datasta saadaan rekonstruoitua spektrikuva haluttuun muotoon. Vasta formaatin tulkitsemisen jälkeen päästään käsiksi tutkimuksen kohteena olevaan spektridataan.

Kansainvälisen valaistuskomitean (CIE, Commission Internationale de l'Eclairage) 8. jaosto on keskittynyt kuvateknologiaan liittyviin asioihin. Jaoston tarkoituksena on tutkia laaja-alaisesti kuvankäsittelyn menetelmiä sekä valmistella alalla tarvittavia kä-

sikirjoja ja standardeja. Jaoston alaisuudessa toimii myös useita teknisiä komiteoita, joista yksi, TC8-07, on keskittynyt spektrikuviin liittyviin asioihin. Parhaillaan kyseinen komitea työskentelee tavoitteenaan määritellä standardi spektrikuvien tiedostoformaatiksi. TC8-07:n tekemää työtä ja komitean jäsenten käymää sähköpostikeskustelua on käytetty yhtenä tämän tutkielman lähteistä.

Tässä tutkielmassa muodostetaan aluksi yleiskuva standardien ja kuvaformaattien määrittelemisestä. Tutkielman painopiste on kuitenkin spektrikuvaformaattien tarkastelussa ja niitä varten tarkoitetun mahdollisen standardin ominaisuuksien kartoittamisessa. Luvussa 2 tuodaan esille standardeihin ja kuvaformaatteihin liittyviä asioita yleisellä tasolla. Luku 3 sisältää yksityiskohtaisempaa tietoa spektrikuvien käyttötarkoituksista ja ominaisuuksista, jotka ovat tarpeellisia spektrikuvia tulkittaessa. Spektrikuvaformaateiksi soveltuvia tiedostoformaatin määrittelyjä tarkastellaan yleisellä tasolla luvussa 4, minkä lisäksi formaattien mukaisia tiedostoja tutkitaan käytännössä luvussa 5. Luvussa 6 vertaillaan tutkielmassa esiteltyjä formaatteja. Luku 7 sisältää yhteenvedon ja pohdintaa tutkielmassa käsitellyistä asioista, muun muassa pohditaan tutkielmassa esiteltyjen formaattien soveltuvuutta yleiseksi standardiksi.

2 Yleistä standardeista ja kuvaformaateista sekä niiden suunnittelusta

Kuvadatan tallentamiseen on olemassa lukuisia erilaisia formaattivaihtoehtoja käyttötarkoituksesta ja laitteistosta riippuen. Jo formaatin suunnitteluvaiheessa on otettava huomioon sekä käyttötarkoituksen että laitteiston formaatille asettamat vaatimukset. Lisäksi on syytä miettiä, millainen formaatin on oltava, jotta sillä olisi mahdollisuuksia levitä laajalle ja saavuttaa yleisen standardin asema. Tässä luvussa käsitellään standardin suunnittelemiseen liittyviä asioita sekä formaattien suunnittelua yleisellä tasolla. Yleiskatsauksen jälkeen perehdytään tarkemmin erityyppisten kuvaformaattien ominaisuuksiin.

2.1 Standardien kehittäminen

Lyhyesti sanottuna standardi on toistuvaan tapaukseen tarkoitettu yhdenmukainen ratkaisu (SFS, 2004). Tarkemmin määriteltynä standardi on standardoinnista huolehtivan viranomaisen, järjestön tai muun tunnustetun elimen hyväksymä kirjallinen julkaisu, joka on kaikkien saatavilla. Standardi voi olla voimassa kansainvälisesti, alueellisesti tai kansallisesti, riippuen siitä, millä tasolla se on hyväksytty. Standardit valmistellaan yhteistyössä avoimissa työryhmissä ja niiden valmistelussa pyritään yhteisymmärrykseen. Aina ei päästä täydelliseen yksimielisyyteen, mutta tavoitteena on aina sellainen yhteisymmärrys, jossa mikään tärkeä eturyhmä ei ole oleellisissa asioissa pysyvästi eri mieltä. Standardit on tarkoitettu yleiseen ja toistuvaan käyttöön, niiden käyttö tosin on vapaaehtoista.

Standardien tavoitteena on ollut alun perin yhteensopivuuden turvaaminen, mikä on edelleenkin standardoinnissa tärkeää. Standardeilla pyritään myös turvallisuuden takaamiseen sekä viestinnän helpottamiseen. Lisäksi standardeissa voidaan määritellä laatuvaatimuksia tuotteille ja niiden avulla pystytään varmistamaan puolueettomien ja vertailukelpoisten tulosten saaminen. Myös lainsäädäntö yksinkertaistuu, kun säädöksiin kirjataan vain olennaiset vaatimukset ja yksityiskohtien osalta viitataan standardeihin.

Vaikka standardi olisi kuinka hyvin määritelty ja helposti kaikkien saatavilla, ei tämä vielä takaa sitä, että standardia todella käytettäisiin. Hyvän standardin on oltava myös

toimiva ja hyödyllinen, eli siitä on saatava jotain selkeää hyötyä työskentelyyn. Wilson (1991) toteaakin, että teollisuuden standardit syntyvät, kun joku henkilö tai jokin ryhmä kehittää ja julkaisee toimivan ratkaisun tiettyyn laajan kohderyhmän ongelmaan. Jos esitetty ratkaisu saavuttaa kohderyhmänsä ja se otetaan yksimielisesti käyttöön kaikkialla, muodostuu ns. de facto -standardi. Tällaisessa tapauksessa ei yleensä ole enää tarvetta standardin virallistamiselle, sillä laajalti käytössä olevaa määrittelyä on lähes mahdotonta ruveta muuttamaan. Virallisten standardien kehitys toimii samalla periaatteella kuin teollisuuden standardien kehitys. Virallinen prosessi voi tosin kestää paljon pidempään, koska se noudattaa yleensä jotakin virallista ja dokumentoitua kaavaa. Virallisten standardien kehitykseen kuuluu myös paljon aikaavievää julkista tarkastelua.

Standardia kehitettäessä on otettava huomioon useita seikkoja, jotka vaikuttavat standardin kehitykseen, toteutukseen sekä standardin peruskäyttäjiin. Tällaisia ovat muun muassa standardin vaikutusalue, kohderyhmä sekä testaus ja vahvistaminen.

Standardin vaikutusalueen määrittäminen voi olla hankalaa. Aluksi asiasta kiinnostuneet ihmiset tekevät standardin vaikutusalueesta alustavan määrittelyn. Alustava määrittely tulee kuitenkin todennäköisesti muuttumaan työskentelyn edetessä. Jos määrittelyssä aiotaan käsitellä standardin ydinalue täydellisesti, tarvitaan todennäköisesti myös laajennuksia ympäröiville alueille. Vaikutusaluetta mietittäessä tulee myös ottaa huomioon standardin mahdolliset tulevaisuuden tarpeet. Jos kehitetään tilapäisratkaisuksi standardi, joka täyttää vain silloisen hetken vaatimukset, saatetaan joutua tilanteeseen, jossa standardin jatkokehitys ei olekaan enää mahdollista.

Standardin määrittelyyn vaikuttaa myös se, millainen standardin kohderyhmä tulee olemaan. Jos standardi on tarkoitettu pääasiassa ihmisten tulkittavaksi, ei määrittelyn tarvitse olla yhtä tarkka kuin sellaisissa tapauksissa, joissa standardi määrittelee jonkin tietokoneella käsiteltävän asian. Tietokonetta varten jokainen pienikin yksityiskohta täytyy määritellä hyvin tarkasti. Ihminen pystyy yleensä ymmärtämään saman asian paljon yksinkertaisemmilla määrittelyillä. Standardin sisältämien määrittelyjen on oltava kuitenkin riittävän tarkkoja ja yksikäsitteisiä kohderyhmän tarpeisiin. Mitä monimutkaisemmasta järjestelmästä on kyse, sitä monimutkaisempia määrittelyjä yleensä tarvitaan. Laaja-alaista standardia määriteltäessä voidaan tarvita paljon asiantuntijoita. Tästä taas voi seurata, että standardin määrittelyn johdonmukaisuus ja yhtenäisyys kärsii. Suuremman ihmisjoukon ollessa mukana määrittelyn tekemisessä määrittelyn koko voi kasvaa, jolloin sen hallinta ja muutosten tekeminen hankaloituu.

2.2 Formaattien suunnittelun näkökulmia

Dataformaatin voidaan ajatella olevan eräänlainen kielen määritelmä, jossa kerrotaan, missä muodossa informaatio tulee tallentaa ja missä muodossa sitä välitetään eteenpäin (Brown & Shepherd, 1995). Dataformaateilla ja yleiskäyttöisillä kielillä on monia yhteisiä ominaisuuksia. Molemmissa tapauksissa täytyy määritellä syntaksi, jonka mukaan kielioppi muodostuu. Näiden kielioppisääntöjen perusteella formaatti tai kieli voidaan tulkita oikealla tavalla. Sekä kieltä että dataformaattia täytyy lukea, jotta saadaan haluttu tieto poimittua käyttöön. Kummassakin, sekä dataformaatin että kielen tapauksessa, käsiteltävän tiedon on oltava sellaista, että tieto on mahdollista tulkita vain yhdellä tavalla. Lisäksi tiedon on oltava käyttökelpoista ja täydellistä siten, ettei mitään datan tai kielen tulkitsemiseen tarvittavaa tietoa puutu.

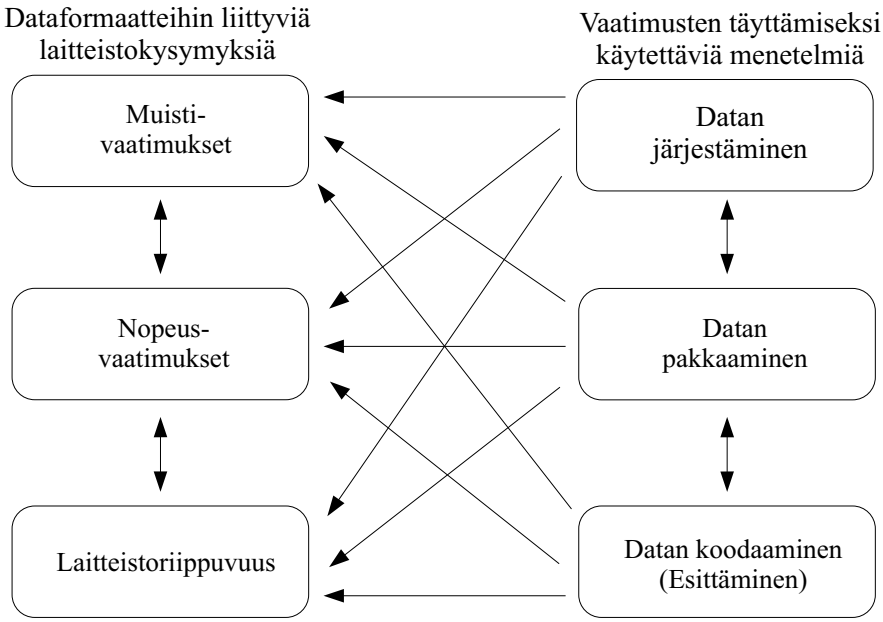
MacLennanin (1983) esittämiä hyvän kielensuunnittelun periaatteita voidaan hyvin soveltaa myös dataformaattien suunnittelussa (Brown & Shepherd, 1995). Periaatteita sovellettaessa voidaan kuitenkin joutua tilanteeseen, joissa kahden periaatteen noudattaminen johtaisi ristiriitaiseen tilanteeseen. Suunnittelun periaatteet voivat siis joiltakin osin olla toistensa vastakohtia, joten kaikki niistä eivät pysty olemaan voimassa yhtä aikaa. Suunnittelija saattaa esimerkiksi joutua tinkimään formaatin yksinkertaisuudesta saadakseen kaiken oleellisen tiedon tallennettua formaattiin. Vastaavasti formaatin siirrettävyys eri laitealustojen välillä voi kärsiä, jos toteutuksen on oltava yksinkertainen ja tehokas. Mitään periaatteista ei kuitenkaan kannata jättää suunnittelussa kokonaan huomioimatta, vaikka tietyt periaatteet saattavatkin tuntua hyvin vastakohtaisilta. Jotkin seuravista Brownin ja Shepherdin (1995) käsittelemistä periaatteista ovat tärkeitä vain tietyillä formaatin suunnitteluun liittyvillä alueilla, toiset taas liittyvät olennaisesti kaikkiin suunnittelun osa-alueisiin.

Itse datan sekä sen järjestyksen, esitysmuodon ja pakkauksen määritteleminen tulee tehdä johdonmukaisesti. Formaatti on pyrittävä suunnittelemaan siten, että toteutus käyttää laitteistoa tehokkaasti. Laitteistoriippuvaisten ominaisuuksien toteuttamista tulee kuitenkin välttää mahdollisimman pitkälle, jotta formaatin siirrettävyys ei kärsisi. Formaattia täytyy pystyä myös tarvittaessa laajentamaan ja/tai muuttamaan ja datasta on pyrittävä tekemään mahdollisimman modulaarista. Lisäksi datalle tehtävät virhetarkastukset kannattaa toteuttaa formaatissa useammalla kuin yhdellä tasolla. Jos virhettä ei huomata yhtä menetelmää käyttämällä, se todennäköisesti voidaan havaita jollakin toisella tavalla. Vähimmäisvaatimuksena voidaan pitää sitä, että yksittäiset virheelli-

set datan arvot eivät saa aiheuttaa koko datajoukon muuttumista lukukelvottomaksi. Formaatin määrittelyn on siis oltava luotettava.

Formaatin määrittelyn on oltava sellainen, että tiedostoon pystytään tallentamaan kaikki tarpeellinen informaatio. Toisaalta taas datan määrittely on pyrittävä saamaan mahdollisimman yksinkertaiseksi. Lisäksi tarpeettomia rajoituksia esimerkiksi datan arvojen tarkkuudelle tai lukumäärälle on syytä välttää. Formaatti on suunniteltava siten, että dataa voidaan käyttää yhä uudelleen ja uudelleen ilman, että sen tarkkuus kärsii. Dataa täytyy pystyä käyttämään muissakin kuin siinä sovelluksessa, johon se on alun perin tarkoitettu. Formaatin rakenne on pyrittävä määrittelemään siten, että tietojen tallennusjärjestys vastaa järjestystä, jossa tietoja käytetään. Tällöin tiedostosta pystytään saamaan talteen yhdellä lukukerralla kaikki tarpeellinen informaatio. Määrittelyssä kannattaa myös ottaa huomioon, että jos jokin dataosio ei liity millään tavalla mihinkään toiseen osioon, kyseinen dataosio tulee tallentaa myös formaatissa itsenäisesti.

Dataformaatin suunnittelussa on myös huomioitava kohdelaitteiston muistin, nopeuden ja muiden komponenttien asettamat vaatimukset. Laitteiston asettamien vaatimusten täyttämiseksi dataformaatteihin on toteutettu muun muassa erilaisia datan tallennusjärjestyksiä, pakkaustapoja ja esitystapoja. Kuvassa 1 esitetään laitteiston ja formaatin ominaisuuksien välisiä riippuvuussuhteita. Kuten kuvassa esitetyistä riippuvuussuhteista näkyy, tietynlaisen menetelmän käyttäminen tai laitteiston ominaisuuden muuttaminen vaikuttaa myös muihin dataformaatin ominaisuuksiin. Esimerkiksi käytettävissä olevan muistin määrä vaikuttaa laitteiston tiedonkäsittelynopeuteen. Datan järjestys tiedoston sisällä voi vaikuttaa pakkaustehokkuuteen. Tehokkaasti pakatun datan lukeminen taas voi olla hidasta ja muistia vievää, koska data joudutaan purkamaan ennen kuin se saadaan käyttöön. Toisaalta pakattu data vie vähemmän levytilaa kuin pakkaamaton data, vaikka onkin hitaampaa lukea. Tietylle laitteistolle optimoitua formaattia pystytään käsittelemään nopeasti kyseisellä laitteistolla, mutta ei välttämättä läheskään yhtä nopeasti muilla. Datan järjestys vaikuttaa myös lukunopeuteen: jos data on tallennettu sovelluksen kannalta oikeaan järjestykseen, niin tarvittavan tiedon esillesaamiseen riittää yksi tiedoston lukukerta.



Kuva 1: Laitteiston ja formaatin ominaisuuksien välisiä riippuvuuksia (Brown & Shepherd, 1995).

2.3 Kuvaformaattien ominaisuuksia

Brownin ja Shepherdin (1995) mukaan digitaalisen kuvan voidaan ajatella olevan joukko diskreettejä näytteitä, jotka jakautuvat tarkastelun kohteena olevalle alueelle. Kuvassa jokainen näyte edustaa väriä tai valon intensiteettiä. Kuvatiedostojen sisältämä data voidaan jakaa karkeasti kolmeen eri luokkaan: *graafinen data*, *metadata* ja *ei-graafinen data*. Lisäksi näiden kolmen luokan sisällä datalla voi olla erilaisia esitysmuotoja.

Kuvan muodostamiseen tarvittava graafinen data voidaan esittää usealla eri tavalla. Jos kuva esitetään *rasteridata*muodossa, silloin kuva koostuu joukosta diskreettejä näytteitä. Näytteet joko muodostavat suoraan kuvan tai niistä voidaan tuottaa kuva. Rasteridata voi olla kaksi- tai kolmiuloitteista, tieteellisissä sovelluksissa datan ulotteisuus voi olla suurempi. *Geometrisen datan* tapauksessa kuvan komponentit kuvataan matemaattista tai geometrista mallia käyttäen joko kaksi-, kolmi- tai useampiuloitteisessa tilassa. Geometriset esitykset, toisin kuin rasteriesitykset, ovat jatkuvia eli kuvan näytteiden välillä ei ole aukkoja. Geometrisissa esityksissä on kaksi vallitsevaa lähestymistapaa, joista toisessa objekti esitetään tallentamalla sen reunat. Toisessa lähes-

tymistavassa määritellään joukko kiinteitä perusmuotoja ja esitetään monimutkaisemmat objektit näiden perusmuotojen kombinaatioina, leikkauksina ja erotuksina. Suurin osa käytössä olevista datan geometriseen esitykseen perustuvista formaateista noudattaa ensin mainittua lähestymistapaa. *Latentti kuvadata* on ei-graafista dataa, jota ei ole sellaisenaan tarkoitettu esitettäväksi kuvana. Laajasti ajatellen mikä tahansa data on latenttia kuvadataa. Kaikentyyppiselle datalle vain ei ole olemassa määrittelyä, jonka mukaisesti datasta saataisiin muodostettua kuva. Yksi esimerkki latentista kuvadatasta ovat laskentataulukon luvut, joista voidaan muodostaa graafisia kaavioita.

Harvat kuvatiedostot sisältävät pelkkää graafista dataa, vaan yleensä tiedostossa on myös jotain oheisdataa eli metadataa. Metadata on graafista dataa kuvaavaa dataa, joka voi olla tyypiltään useammanlaista. *Tulkintadata* kertoo, kuinka tiedostoa täytyy käsitellä ja tulkita. Joissakin formaateissa tulkintadataa tarvitaan enemmän kuin toisissa. Jotkut formaatit voivat olla rakenteeltaan hyvinkin tarkkaan ennalta määriteltyjä, jolloin tiedostoon tallennettavan tulkintadatan määrä on hyvin vähäinen. *Attribuuttidata* määrittelee kuvan objektien ulkoasun, kuten esimerkiksi viivan paksuuden ja värin geometrisen datan tapauksessa. Metadata voi sisältää myös sellaista kuvaan liittyvää dataa, jota ei suoranaisesti tarvita kuvan esittämiseen. Tällaista dataa voi olla esimerkiksi tieto kuvan luontipäivämäärästä, tekijästä tai tekijänoikeuksista. Tällaista dataa kutsutaan *dokumentaatiodataksi*.

Graafisen datan ja metadatan lisäksi kuvatiedosto voi sisältää myös ei-graafista dataa. Ei-graafinen data voi liittyä jollain tavalla tiedoston graafisiin elementteihin, jolloin datan tyyppi on yleensä hyvin riippuvainen käytettävästä sovelluksesta. Ei-graafinen data voi olla tyypiltään myös sellaista, että se ei tiedoston liity graafisiin elementteihin muuten kun käsitteellisellä tasolla. Tällöin graafisen ja ei-graafisen datan välillä ei ole olemassa mitään eksplisiittisiä linkkejä tai suhteita.

2.4 Kuvaformaattien fyysinen rakenne

Suurin osa kuvaformaateista rakentuu siten, että tiedoston alussa on erillinen otsikko-osio, johon on tallennettu kuvatiedoston tulkitsemiseen tarvittavaa metatietoa. Otsikko-osio voi sisältää esimerkiksi tiedon tiedoston bittijärjestyksestä, pikselien järjestyksestä, kuvan koosta ja siitä, onko kuvaa pakattu jollain algoritmilla. On myös tärkeää, että tiedoston tyyppi määritellään heti tiedoston alussa otsikko-osiossa. Tiedostoa

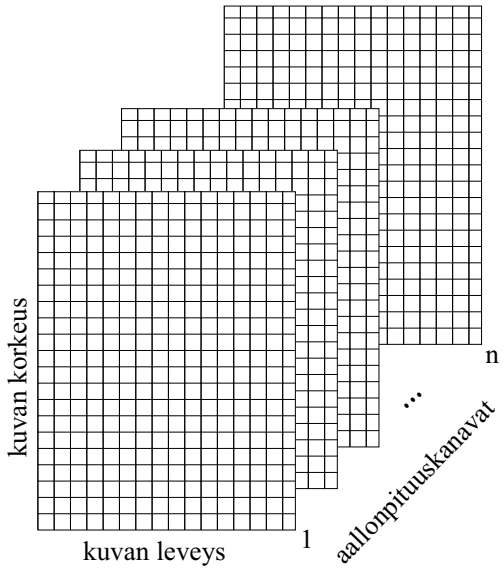
käsiteltäessä ei nimittäin voida luottaa siihen, että tiedoston tyyppi voidaan päätellä tiedostopäätteestä.

Itse kuvadata on tallennettuna tiedostoon kunkin formaatin määrittelemällä tavalla. Useimmissa tapauksissa kuvadata sijaitsee tiedostossa otsikko-osion jälkeen joko pakattuna tai pakkaamattomana. Kuvadataa käsiteltäessä voidaan otsikko-osiosta löytyvän metatiedon lisäksi tarvita myös formaatin määrittelystä löytyvää tietoa, jota ei ole tallennettu tiedostoon.

Riippuen kuvan koosta sekä siitä, millä tarkkuudella kuvadata tallennetaan, kuvatiedosto voi viedä ilman pakkausta hyvinkin paljon tilaa. Erityisesti spektrikuvat, jotka voivat koostua kymmenistä tai jopa sadoista kanavista, kuluttavat levytilaa helposti useita satoja megatavuja. Jos lukuisia kuvatiedostoja aiotaan arkistoida samaan paikkaan tai niitä halutaan siirtää verkossa paikasta toiseen, kuvan koko on pyrittävä saamaan mahdollisimman pieneksi. On olemassa erilaisia pakkausalgoritmeja, häviöllisiä ja häviöttömiä, joilla kuvatiedostot saadaan tallennettua huomattavasti raakadataa pienempään tilaan. Sopivan algoritmin valinta riippuu täysin kuvan käyttötarkoituksesta. Joka tapauksessa kuvan pakkaaminen ja samanaikaisesti kuvanlaadun säilyttäminen ovat asioita, joihin on oleellista kiinnittää huomiota mitä tahansa kuvaa tallenettaessa. Usein kuvia tallennettaessa joudutaan tekemään jonkinasteinen kompromissi kuvan laadun ja kuvatiedoston viemän tilan välillä.

3 Spektrikuvien tarve

Spektrikuva on kuva, jonka jokaisessa kuvapisteessä on spektri. Kuvassa 2 on esitetty pelkistetty malli spektrikuvasta. Spektrejä käyttämällä saadaan tarkasteltavasta kohteesta talteen tarkka väriesitys, joka ei riipu valaistuksesta. Nykyisillä mittalaitteilla päästään hyvin tarkkaan värinmääritykseen, jolloin spektrikuva voi koostua jopa useista sadoista eri aallonpituuksilta mitatuista komponenteista. Väriesityksen tarkkuus on siis aivan eri luokkaa kuin perinteisiä kolmekomponenttisia esityksiä käytettäessä. Usein spektrejä tarkastellaan näkyvän valon aallonpituuksilla välillä 400–700 nanometriä, mutta on olemassa myös sovelluksia, joissa laajennetaan tarkasteltavaa aluetta näkyvän valon alueen ulkopuolelle infrapuna- tai ultraviolettiaallonpituuksille.



Kuva 2: Yksinkertaistettu malli spektrikuvasta, jonka jokainen pikseli sisältää *n*-komponenttisen spektrin.

3.1 Spektrikuvien käyttökohteita

Spektrikuvista tarkasteltavat ominaisuudet vaihtelevat sovelluskohteittain. Kaikille sovelluksille kuitenkin on yhteistä se, että spektrimittauksilla pyritään analysoimaan tarkasti kohteen ominaisuuksia. Seuraavassa käydään läpi esimerkkejä erilaisista spektrikuvien sovelluskohteista.

Omenoiden pinnan vikojen ja pilaantumisen tunnistaminen (Mehl *& al.*, 2004). Tarkasteltavaksi valittiin pilaantumattomia omenia satunnaisotannalla. Pilaantumattomien omenien lisäksi tutkittiin selvästi kolhuisia ja pilaantuneita omenia. Mittaukset suoritettiin aallonpituusalueella 430–900 nm kuvien spatiaaliresoluution ollessa 0.5–1.0 mm. Pilaantuneiden kohtien määrittämisessä käytettiin reflektanssispektrejä. Pilaantuneita kohtia etsittiin yksittäisiä aallonpituuskanavia (542, 682 ja 752 nm) avuksi käyttäen. Tiettyä yksittäistä kanavaa tarkastelemalla ei kuitenkaan pystytty suoraan erottamaan pilaantuneita omenia pilaantumattomista, sillä erilaiset vauriot erottuivat omenissa eri aallonpituuksilla. Yksittäisten kanavien tarkastelun lisäksi spektrikuvia tutkittiin myös pääkomponenttianalyysia ja toisen asteen differenssimenetelmiä käyttäen.

Perunaruton aiheuttaman stressin tunnistaminen tomaateista (Zhang *& al.*, 2003). Kasvukauden aikana mitattiin pellolla kasveista yksittäisiä spektrejä. Näistä mittauksista saatujen tulosten perusteella tulkittiin AVIRIS(airborne visible infrared imaging spectrometer)-mittauksista saatua spektrikuvadataa. Mittaukset tehtiin aallonpituusalueella 400–2500 nm ja AVIRIS-kuvien spatiaaliresoluutio oli 4 m. Mittauksissa havaittiin, että lähi-infrapuna-alue (NIR, 700–1300 nm) paljasti kasvien sairaudet näkyvän valon aallonpituusaluetta paremmin. NIR-alueella erot spektreissä terveiden ja sairaiden kasvien välillä olivat 10 %:n luokkaa, näkyvän valon alueella vain 1,19 %. Sairaat kasvit voidaan erottaa terveistä spektrikuvantamisen avulla, sillä jos kasvi on saanut tartunnan, taudin aiheuttama fysiologinen reaktio saa aikaan muutoksia kasvista mitattujen spektrien reflektanssiarvoissa. Sairaassa kasvissa lehtivihreän määrä vähenee ja kasvin sisäinen rakenne muuttuu.

Taideteosten ja historiallisten esineiden analyysi (Balas *& al.*, 2003). Fyysisten näytteiden ottaminen vahingoittaa taide-esineitä. Lisäksi jostain tietystä teoksen kohdasta otettu näyte edustaa vain kyseistä pistettä eikä välttämättä anna oikeaa kuvaa koko tarkasteltavasta alueesta. Spektrikuvantaminen ei vaurioita kohdetta ja menetelmää voidaan käyttää siellä, missä tarkasteltavat kohteet sijaitsevat. Lisäksi spektrikuva voidaan ottaa koko tarkasteltavasta kohteesta, jolloin teoksesta pystytään tutkimaan suurempia alueita kerralla. Vaikka spektreistä ei pystyttäisi suoraviivaisesti päättelemään, mistä materiaalista on kyse, kohteen lisäksi voidaan mitata referenssinäytteet tunnetuista materiaaleista ja verrata kohteen spektrejä referenssispektreihin. Spektrikuvantamista voidaan hyödyntää esimerkiksi maalipigmenttien tunnistamisessa ja päällekirjoitettujen käsikirjoitusten paljastamisessa.

Sähköinen kaupankäynti (Tsumura *& al.*, 2002). Sähköisessä kaupankäynnissä tavoitteena on, että tuotteen värit näyttävät samanlaisilta tietokoneen näytöllä ja koti- tai toimistovalaistuksessa. Toisin sanoen, asiakkaan olosuhteissa tuotteen tulisi näyttää samanväriseltä kuin tietokoneen näytöllä. Tarkkaan väriesitykseen päästään spektreihin pohjautuvia menetelmiä käyttäen. Tsumura *& al.* ovat kehittäneet menetelmän, jossa kameralla tietyssä valaistuksessa otetut monikanavakuvat muunnetaan reflektanssispektrikuviksi etukäteen mitattujen näytespektrien ominaisuuksien perusteella. Spektrikuvista voidaan tuottaa kullekin näyttö- ja tulostuslaitteelle oikeellinen väriesitys siten, että esitystä muodostettaessa huomioidaan valaistus ja laitteen ominaisuudet.

Tekstiiliteollisuuden sovelluskohteet (Herzog & Hill, 2003). Kankaiden ulkonäkö ei määräydy pelkästään värien perusteella, vaan myös pinnan rakenne vaikuttaa siihen, miltä kangas näyttää. Spektrikuvaan tallentuu tieto pinnan heijastuksesta, minkä perusteella voidaan tutkia värin lisäksi myös pinnan rakennetta. Tekstiiliteollisuudessa spektrikuvien tarjoamia mahdollisuuksia voidaan soveltaa muun muassa värintoiston parantamisessa laadunvalvontaa ja vaatemallien tulostamista varten. Spektrikuvia käytettäessä voidaan mahdollistaa myös värien toistuminen oikein, kun suunnitteluluonnoksia vaihdetaan sähköisessä muodossa eri suunnittelijoiden kesken.

Lääketieteen sovelluskohteet (Okuyama *& al.*, 2003). Ihon ominaisuuksien tarkastelu ja erityisesti ihon väri on tärkeässä osassa useiden tautitilojen diagnoosissa ja lääketieteellisten kuvien tietokoneanalyysissa. Spektrikuvantamisen avulla saadaan potilaan tilan diagnosointia varten muodostettua kuvia, joista eliminoidaan valonlähteen ja sen aiheuttamien varjojen vaikutus ihon väriin. Matemaattisia menetelmiä käyttäen 4-kanavakameralla otetut kuvat saadaan muunnettua reflektanssispektrikuviksi, joiden arvot ovat välillä 400–700 nm. Reflektanssispektreistä taas saadaan pääkomponenttianalyysia ja käänteistä Monte Carlo-simulaatiota käyttäen ihon pigmentaatioita mallintavat kuvat, joita voidaan käyttää potilaan tilan tarkkailemiseen.

Tähtitieteen sovelluskohteet (Semeter *& al.*, 2001). Spektrikuvantamista voidaan hyödyntää esimerkiksi revontulia tutkittaessa. Koska revontulet vaihtelevat hyvin nopeasti, täytyy spatiaali- ja spektrisuunnan mittaukset saada tehtyä samanaikaisesti. Mittauksia ei siis voida tehdä käyttäen monokromaattista kuvauslaitetta ja filtterikiekkoa, koska tällöin eri aallonpituuksilta saatavat tulokset eivät kuvaa samaa tilannetta. Vastaavasti jos spektrejä mitattaisiin spektrografilla yhdestä pisteestä kerrallaan, koko kuva ei olisi samasta tilanteesta, koska revontulet ehtivät muuttua kuvauksen aikana. Revontulien tutkimiseen onkin kehitetty järjestelmä, joka tekee mittaukset yhtä aikaa neljältä

eri aallonpituudelta (427,8 nm, 630 nm, 732,5 nm ja 844,6 nm). Järjestelmä pystyy ottamaan kuvia ionosfäärin E-kerroksesta eli noin 105–160 km korkeudesta käyttäen 1 km:n spatiaaliresoluutiota.

3.2 Spektrikuvan muodostaminen

Spektrikuvassa jokainen pikseli sisältää spektrin $\mathbf{s}(\lambda)$. Spektri voidaan esittää kaavan 1 mukaisesti $n \times 1$-vektorina, jonka jokainen komponentti kuvaa spektrin arvoa tietyllä aallonpituudella. Aallonpituuksia on yhteensä n kappaletta, joista ensimmäinen aallonpituus on λ_1 ja viimeinen λ_n.

$$\mathbf{s}(\lambda) = (s(\lambda_1)\ s(\lambda_2)\ \ldots\ s(\lambda_n))^T \tag{1}$$

Spektrikuvan tulkitsemiseen ei riitä pelkkä spektridata, vaan tarvitaan myös muuta kuvaan liittyvää oheistietoa. Kuvasta on tiedettävä, millä aallonpituusvälillä sen arvot on mitattu. Tarpeelliset arvot ovat ensimmäinen aallonpituus λ_1 ja viimeinen aallonpituus λ_n. Näiden lisäksi on tiedettävä n eli aallonpituuksien lukumäärä, jonka avulla voidaan laskea, kuinka monen yksikön välein mittaukset on suoritettu. Aallonpituuksien lukumäärän sijasta kuvaan voidaan tallentaa suoraan tieto siitä, kuinka monen yksikön välein spektrin arvoja on mitattu.

Kuvan tulkitsemisen kannalta on myös oleellista tietää, minkätyyppisiä spektrejä kuvan arvot esittävät. Spektriä, josta valonlähteen vaikutus on eliminoitu, kutsutaan heijastavan kohteen tapauksessa *reflektanssispektriksi* ja läpäisevän kohteen tapauksessa *transmittanssispektriksi*. Molemmissa tapauksissa spektriä, jossa valonlähteen vaikutus näkyy, kutsutaan *radianssispektriksi*. Valonlähteen vaikutus pystytään eliminoimaan siten, että mitataan aluksi referenssivalkoisen spektri $\mathbf{s}_{ref}(\lambda)$, jolla radianssispektridata $\mathbf{s}_{rad}(\lambda)$ jaetaan (kaava 2).

$$\mathbf{s}_{refl}(\lambda) = \frac{\mathbf{s}_{rad}(\lambda)}{\mathbf{s}_{ref}(\lambda)} \tag{2}$$

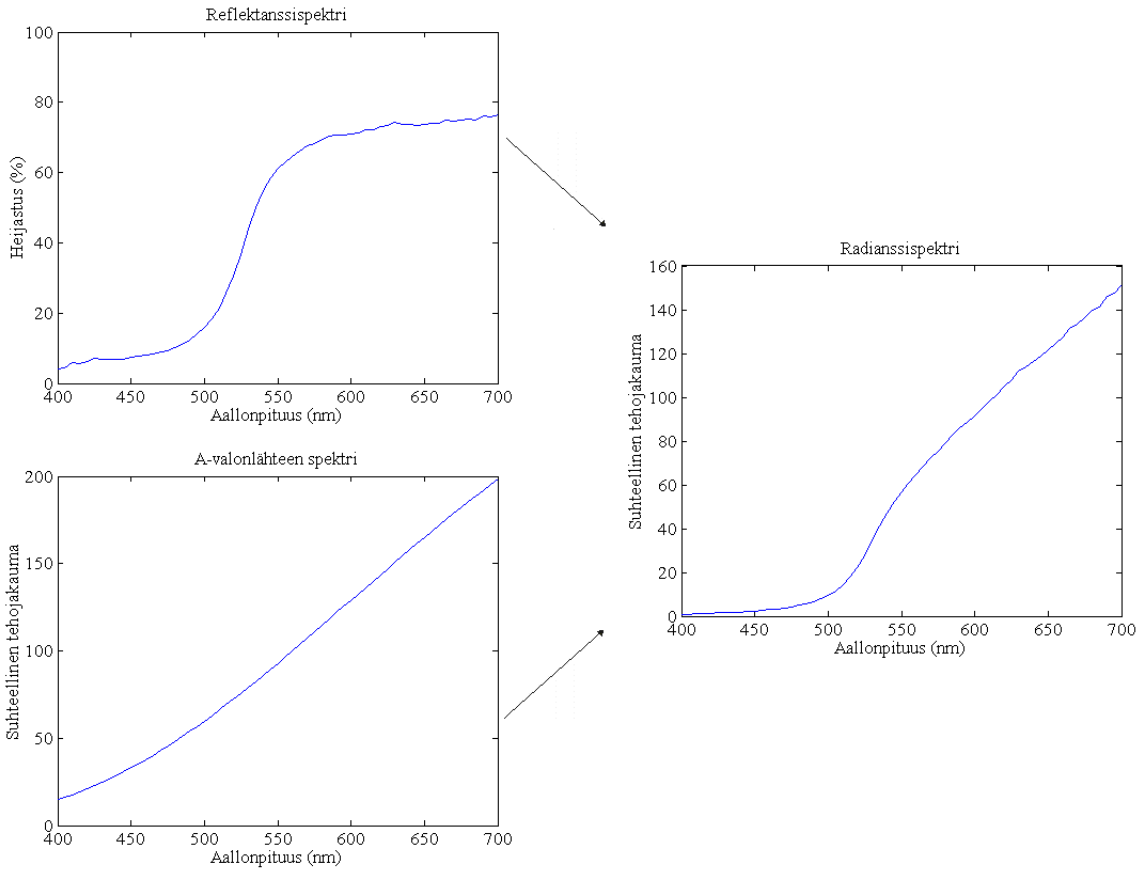
Tarkassa spektrikuvauksessa myös käytetyn mittalaitteen ilmaisimen aiheuttama virhe tulisi korjata. Tämä voidaan tehdä vähentämällä mitatuista spektreistä spektri $\mathbf{s}_{black}(\lambda)$, joka on mitattu, kun ilmaisimeen ei tule ollenkaan valoa. Tällöin reflektanssispektri

saadaan kaavasta

$$\mathbf{s}_{refl}(\lambda) = \frac{\mathbf{s}_{rad}(\lambda) - \mathbf{s}_{black}(\lambda)}{\mathbf{s}_{ref}(\lambda) - \mathbf{s}_{black}(\lambda)} \tag{3}$$

Valonlähteen vaikutus spektriin saadaan selville, kun kerrotaan reflektanssi- tai transmittanssispektri pisteittäin valonlähteen spektrillä (kaava 4). Kuvassa 3 on esimerkki siitä, miten standardi A-valonlähde (hehkulamppu) vaikuttaa mitattuun reflektanssispektriin.

$$\mathbf{s}_{rad}(\lambda) = \mathbf{s}_{ref}(\lambda) \cdot \mathbf{s}_{refl}(\lambda) \tag{4}$$



Kuva 3: Radianssispektri saadaan, kun kerrotaan reflektanssispektri valonlähteen spektrillä.

Yksinkertaisimmillaan spektridata voidaan tallentaa ilman pakkausta suoraan kuvamatriisina. Harvat tallennusmuodot tukevat kolmiulotteisten matriisien tallentamista, joten yleensä spektrikuva on kätevintä esittää kaavan 5 mukaisesti kaksiuloitteisena kuvamatriisina **S**. Kuvamatriisissa spektrien lukumäärä on $m = x * y$ (x on kuvan leveys ja y kuvan korkeus) sekä aallonpituuskanavien lukumäärä n.

$$\mathbf{S} = \begin{pmatrix} s_1(\lambda_1) & \cdots & s_m(\lambda_1) \\ \vdots & \ddots & \vdots \\ s_1(\lambda_n) & \cdots & s_m(\lambda_n) \end{pmatrix} \tag{5}$$

Spektridata voi olla tiedostossa esitettynä jonkin matemaattisen mallin avulla tai pakattuna häviöttömällä tai häviöllisellä algoritmilla. Tällaisissa tapauksissa kuvaformaatin määrittelyn täytyy sisältää tieto siitä, kuinka spektridata saadaan rekonstuoitua tiedoston sisällöstä. Esimerkiksi pääkomponenttianalyysia (PCA, Principal Component Analysis) (Oja, 1983) käyttämällä spektrikuvaa pystytään pakkaamaan ja pakkauksesta huolimatta säilyttämään spektridata lähes virheettömänä. Pääkomponenttianalyysia käytettäessä spektrikuva voidaan esittää kerroinmatriisin ja kantavektorien avulla. Pakkauksen aiheuttaman virheen suuruus riippuu käytettävien kantavektorien lukumäärästä: mitä enemmän suurimpia ominaisarvoja vastaavia vektoreita kantavektorijoukkoon valitaan, sitä vähemmän virhettä syntyy alkuperäisten ja rekonstruoitujen spektrien välille.

Pääkomponenttianalyysissa alkuperäisestä spektrikuvamatriisista **S**, jonka aallonpituuksien lukumäärä on n ja spektrien lukumäärä m, lasketaan ominaisarvojen ja -vektorien määrittämistä varten korrelaatiomatriisi **R**, jonka koko on $n \times n$.

$$\mathbf{R} = \frac{1}{m}\mathbf{S}\mathbf{S}^T \tag{6}$$

Matriisilla **R** on n kappaletta ominaisarvoja ja -vektoreita. Ominaisvektorit Φ ovat yhtälön

$$\mathbf{R}\Phi = \sigma\Phi \tag{7}$$

ratkaisuja. Kyseisessä yhtälössä σ on diagonaalimatriisi, joka sisältää matriisin **R** ominaisarvot.

Ominaisvektoreista valitaan kantavektoreiksi $\mathbf{b}_i(\lambda)$ p kappaletta vektoreita, jotka vastaavat p:tä suurinta ominaisarvoa. Jokaisessa kantavektorissa $\mathbf{b}_i(\lambda)$ on n komponenttia kuten spektrissäkin. Kantavektorit voidaan esittää yhtenä matriisina kaavan 8 mukaisesti.

$$\mathbf{B} = \begin{pmatrix} b_1(\lambda_1) & \cdots & b_1(\lambda_n) \\ \vdots & \ddots & \vdots \\ b_m(\lambda_1) & \cdots & b_m(\lambda_n) \end{pmatrix} \tag{8}$$

Kun lasketaan kantavektorien ja spektrikuvamatriisin sisätulo

$$\mathbf{P} = \mathbf{B}^T\mathbf{S}, \tag{9}$$

saadaan sisätulomatriisi $\mathbf{P}$, jonka koko on $p \times m$. Kantavektorien lukumäärästä riippuen, kantavektorit ja sisätulomatriisi yhteensä vievät yleensä huomattavasti vähemmän tilaa kuin alkuperäinen spektrikuvamatriisi. Spektrikuva voidaan rekonstruoida kantavektorien ja sisätulomatriisin avulla seuraavasti:

$$\tilde{\mathbf{S}} = \mathbf{B}\mathbf{B}^T\mathbf{S} = \mathbf{B}\mathbf{P} \tag{10}$$

Spektrikuva voidaan muodostaa myös käyttäen kameran vasteita ja rekonstruointimatriisia (Hardeberg, 2001). Rekonstruointimatriisin muodostamiseen on olemassa lukuisia eri menetelmiä. Spektrien rekonstruointi tehdään tässä tapauksessa hyvin samankaltaisesti kuin pääkomponenttianalyysia käytettäessä. Sen sijaan, että kerrottaisiin ominaisvektorimatriisi ja kerroinvektori keskenään, kerrotaankin $n \times k$-rekonstruontimatriisi $\mathbf{Q}$ ja kameralta saadut k vastetta sisältävä vektori $\mathbf{c}_k$ (kaava 11).

$$\tilde{\mathbf{s}}(\lambda) = \mathbf{Q}\mathbf{c}_k \tag{11}$$

Nähdään siis, että monipuolisessa kuvaformaatissa tulee kuvan tallentamisen lisäksi olla mahdollisuus tallentaa kattavasti kuvantamiseen liittyvää tietoa.

3.3 Spektrikuvaformaatin ominaisuuksia

CIE:n tekninen komitea TC8-07 on keskittynyt spektrikuviin liittyviin asioihin, ja parhaillaan kyseinen komitea työskentelee tavoitteenaan määritellä standardi tiedostoformaatti spektrikuvien tallentamista varten. Spektrikuvien käyttäjäjoukko on laaja ja käyttötarkoitukset hyvin moninaisia, joten standardia määriteltäessä joudutaan miettimään monista eri näkökulmista, mitä ominaisuuksia standardiformaattiin loppujen lopuksi sisällytetään. Kuten luvussa 2.1 tuli ilmi, standardin määrittelystä on tehtävä tarkka ja yksikäsitteinen ja sen on oltava myös laajennettavissa myöhemmin, jos tarve vaatii. Standardin vaikutusalueen ja kaikkien tarpeellisten ominaisuuksien määrittäminen on tässäkin tapauksessa hankala tehtävä ja kaikenlaisten spektrikuvien tallentamiseen sopivan standardiformaatin kehittäminen on aikaavievä ja työläs projekti.

CIE:n työryhmän keskusteluissa (CIE, 2003-2004) (P. Herzog, 14.1.2004) onkin ehdotettu, että formaatin kehittäminen jaettaisiin useampaan osaan ja työskentely aloitettaisiin määrittelemällä yksinkertainen formaatti, joka sisältää vain sen verran informaatiota kuin tarvitaan kuvan lukemiseen tiedostosta sekä sen esittämiseen oikein. Seuraavassa vaiheessa formaattiin määriteltäisiin lisää asioita, joita tarvitaan edistyneemmissä kuvankäsittelyn sovelluksissa, kuten esimerkiksi häviöllinen ja häviötön pakkaus. Kussakin vaiheessa määriteltävien asioiden lisäksi formaattiin voitaisiin sisällyttää myös lisätieto-osio, johon kaikki formaatin määrittelyn ulkopuolinen kuvaan liittyvä tieto pystyttäisiin tallentamaan.

Spektrikuvaformaatin suunnittelussa on siis otettava huomioon eri sovelluskohteiden tarpeita. Kuvaan liittyvien lisätietojen tallennukselle on erilaisia vaatimuksia käyttötarkoituksen mukaan. Teollisissa sovelluksissa spektrikuvien on oltava helppokäyttöisiä, kun taas arkistointitarkoituksia varten täytyy kuviin liittyvä informaatio pystyä tallentamaan yksityiskohtaisesti. Eri sovelluskohteiden tarpeiden lisäksi suunnittelussa on kiinnitettävä huomiota formaatin helppokäyttöisyyteen ja selkeyteen. Formaatin yhtenäisyyden kannalta paras ratkaisu olisi tallentaa kaikki spektrikuvaan liittyvä tieto yhteen tiedostoon. Spektridatan ja siihen liittyvän informaation lisäksi kuvatiedosto voi myös sisältää muuta oheistietoa, esimerkiksi tietoja mittalaitteistosta tai mittausympäristöstä, esimerkiksi ilmaisimen pimeävirta $\mathbf{s}_{black}(\lambda)$ ja valonlähteen spektri $\mathbf{s}_{ref}(\lambda)$.

Tarkkojen mittaustulosten tallentamiseksi spektrikuvatiedoston koon on voitava olla suuri, ainakin 2^{64} tavua. Lisäksi spektrikuvien arvojen tallennustarkkuuden on oltava hyvä sekä koko kuvaa ajatellen (mahdollisuus tallentaa useita satoja, jopa tuhansia

aallonpituuskanavia) että yksittäisen arvon kohdallakin (mahdollisuus tallentaa arvot 32-bittisinä, käyttötarkoituksesta riippuen oltava mahdollista valita myös esim. 16- tai 8-bittinen esitys). Formaatissa olisi hyvä olla pakkaamattoman datan tallentamisen lisäksi mahdollisuus myös datan häviöttömään ja häviölliseen pakkaukseen.

Osa spektrikuvaformaatin ominaisuuksista, kuten tiedoston bitti- ja tavujärjestys, lukuarvojen talletustarkkuus, tiedoston kokorajoitteet, kuvan spatiaalidimensiot ja kuvan resoluutio, ovat minkä tahansa kuvatiedoston perusominaisuuksia. Spektrikuvaan liittyviä, formaattiin tarvittavia erityisominaisuuksia ovat muun muassa kuvan aallonpituusarvot λ, aallonpituuskanavien lukumäärä n (spektrinen dimensio), spektrien tyyppi (reflektanssi/transmittanssi/radianssi) sekä spektridatan esitysmuoto. Jos kuva on esitetty jonkin matemaattisen mallin avulla tai pakattu joko häviötöntä tai häviöllistä pakkausta käyttäen, niin formaatin määrittelyssä on tultava ilmi, kuinka tiedoston sisällöstä pystytään rekonstruoimaan spektrikuva.

Edellä mainittujen yksityiskohtien lisäksi formaattiin voitaneen toteuttaa tarvittaessa myös monimutkaisempia ominaisuuksia. Tässä luvussa lueteltuja ominaisuuksia voidaan kuitenkin pitää sellaisina, jotka ovat oleellisia spektrikuvan tulkinnan kannalta ja joiden avulla saadaan esitettyä spektrikuvan tulkitsemiseen tarvittavat perustiedot.

Yhteenvetona voidaan todeta, että minimitiedot, jotka spektrikuvan tallentamiseen tarvitaan, ovat kuvamatriisi M, aallonpituusalueen rajat λ_1 ja λ_n sekä aallonpituuskomponenttien määrä n.

4 Spektrikuvaformaattiehdokkaita

Useat tutkimusryhmät ovat kehittäneet omiin tarpeisiinsa soveltuvia spektrikuvaformaatteja. Täysin uusien formaattien kehittämisen lisäksi on myös joitakin valmiita kuvaformaatteja muokattu siten, että ne soveltuvat spektrikuvien tallentamiseen. Eri tutkimusaloilla tarvitaan itse spektridatan lisäksi hyvinkin vaihtelevaa tietoa esimerkiksi kuvauslaitteistosta, värisovitusfunktioista tai otetun kuvan maantieteellisestä sijainnista. Tässä luvussa esitellään joitakin olemassaolevia spektrikuvaformaatteja (MUSP, luku 4.1 ja Natural Vision, luku 4.2) sekä kuvaformaatteja, joiden käyttäminen spektrikuvien tallentamiseen on mahdollista joko suoraan tai jonkinlaisten lisämäärittelyjen jälkeen (JPEG2000, luku 4.3, TIFF / GeoTIFF, luku 4.4 ja HDF5, luku 4.5). Vertailun vuoksi luvussa 4.6 esitellään yksinkertainen binaariformaatti spektrikuvan tallentamista varten. Tässä tutkielmassa käsiteltävät formaatit ovat luvun 4.6 binaariformaattia lukuunottamatta sellaisia, jotka on tuotu esille CIE:n 8. jaoston spektrikuvatyöryhmän (TC8-07) keskusteluissa (CIE, 2003-2004).

4.1 MUSP

MUSP Multispectral Image File Format (MUSP, 2003) on Color AIXperts GmbH -nimisen saksalaisen yrityksen kehittämä spektrikuvaformaatti, joka on käytössä teollisissa sovelluksissa.

MUSP-formaatti on suhteellisen yksinkertainen, sillä siinä on määritelty, että kaikki spektrikuvaan liittyvä tieto on tallennettuna yhteen tiedostoon. Tiedosto muodostuu vakiomittaisesta otsikko-osiosta, tunnistekenttätaulukosta ja näiden jälkeisestä dataosiosta, josta löytyvät tunnistekenttätaulukossa määritellyt tiedot (taulukko 1). MUSP-määrittelyn mukaisen tiedoston arvot on tallennettu merkitsevin tavu ensin (ns. big endian -muoto) ja tiedoston maksimikoko on 2^{64} tavua. Tiedostoon tallennettavien spektrien rekonstruointiin käytettävien arvojen tarkkuutta ei ole kiinnitetty, vaan voidaan määritellä tapauskohtaisesti, kuinka monesta tavusta ja bitistä yksi arvo koostuu.

MUSP-formaatissa spektridata jaetaan kahteen osaan: rekonstruointimatriisiin ja spektrikehyksiin. Jokainen spektrikehys voi sisältää eri tavoin tallennettua dataa, sillä tiedot kehyksen arvojen bittimäärästä sekä arvoille käytetystä pakkauksesta tallennetaan kunkin kehyksen yhteyteen. MUSP-formaatissa voi käyttää kehysten arvoille

häviötöntä (ZIP) tai häviöllistä (JPEG) pakkausta. Tiedoston sisällöstä saadaan siis rekonstruoitua alkuperäinen spektridata, kun kerrotaan spektrikehysten arvot rekonstruointimatriisilla. Rekonstruointimatriisi ja spektrikehykset voivat sisältää esimerkiksi pääkomponenttianalyysista saatua dataa, jolloin rekonstruointimatriisi muodostuu kantavektoreista (matriisi **B** kaavassa 8) ja jokainen spektrikehys sisältää yhden sisätulomatriisin **P** (9) sarakkeista. Tiedostoon voidaan tallentaa myös kameralta saatua dataa siten, että rekonstruointimatriisi on matriisi **Q** (11) ja spektrikehykset sisältävät kameralta filttereitä käyttäen saadut vasteet eli vektorin $\mathbf{c_k}$ komponentit (11).

Kuvan tulkitsemiseksi oikein tarvitaan myös tieto siitä, minkäkokoisesta spektrikuvasta on kyse ja millä aallonpituusvälillä kuvan arvot sijaitsevat. Spektridatan spatiaalidimensiot sekä -resoluutiot tallennetaan MUSP-formaatissa tiedoston otsikko-osioon. Aallonpituusarvot ja spektrisuunnan dimensio tallennetaan rekonstruointimatriisin yhteyteen omiin tunnistekenttiinsä.

Taulukko 1: MUSP-tiedoston rakenne.

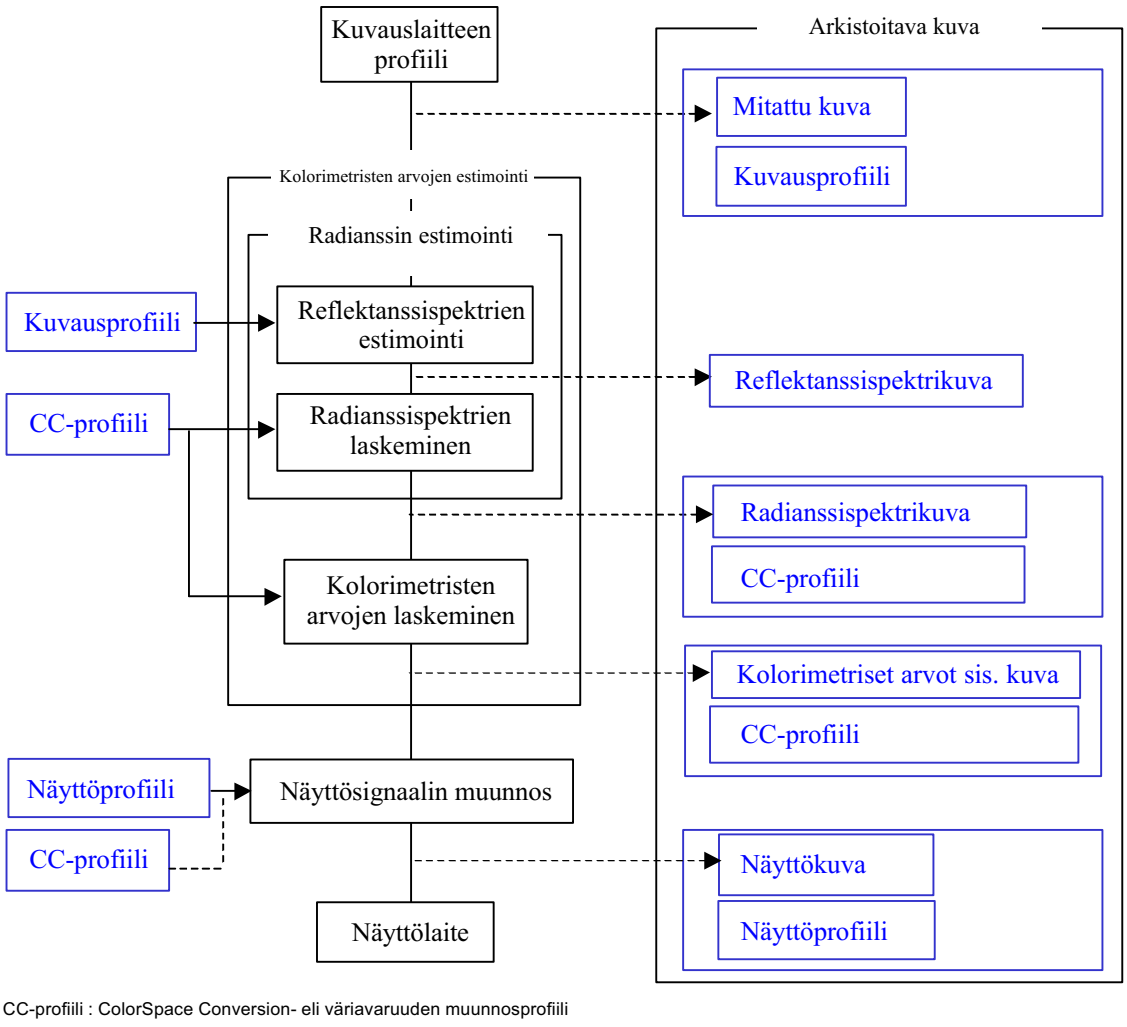
Tiedoston osio		**Tavua**
Otsikko-osio		64
Tunnistekenttätaulukko (T = kenttien lkm)		$T * 20$
	Kentän nimi	4
	Kentän alkukohta	8
	Kentän pituus	8
	…	…
Dataosio sisältää kaikki tunnistekenttätaulukossa määritellyt kentät (T kpl), esim. rekonstruointimatriisin ja spektrikehykset		
Tunnistekenttä 1		
Tunnistekenttä 2		
…		

4.2 Natural Vision

Natural Vision -tiedostoformaatin määrittely (Natural Vision, 2003) on osa spektrien kuvantamiseen tarkoitettua tiedostoformaattia, jota käytetään Natural Vision -projektissa. Natural Vision -projektin on perustanut vuonna 1999 Japanin teletekniikan kehitysjärjestö (Telecommunications Advancement Organization of Japan, TAO – nykyisin kansallinen informaatio- ja viestintätekniikan tutkimuslaitos, National Institute of Information and Communications Technology eli NICT). Projektin tarkoituksena on mahdollistaa luonnollinen värintoisto suurella tarkkuudella näköön liittyvissä teletekniikan järjestelmissä. Tiedostoformaatti on kehitetty siksi, että pystyttäisiin vaihtamaan kuvadataa spektrikuvia tutkivien ja kehittävien tahojen välillä.

Natural Vision -formaatin nykyisessä versiossa (2.0s) on määritelty pelkästään kuviin liittyvä väriprofiilidata. Määritellyt väriprofiilit voidaan liittää mihin tahansa kuvaformaattiin, jossa monikanavainen data on määritelty sopivalla tavalla. Natural Visionin profiilimäärittelyt perustuvat ICC:n (International Color Consortium) tekemiin väriprofiilimäärittelyihin (ICC, 2004). Natural Vision -formaatissa ei käytetä ICC:n määrittelyjä sellaisenaan, koska ne eivät suoraan sovellu spektrikuvantamisen tarkoituksiin. Kuvatiedostoon tallennettu data ei kuitenkaan Natural Vision -formaatissa ole välttämättä suoraan spektridataa, vaan kuvatiedosto voi sisältää esimerkiksi mittaus- tai näyttölaitteelta saatua dataa, joka voidaan muuntaa spektridataksi kuvaan liitetyn väriprofiilin avulla. Jokaiseen kuvaan liittyy siis tietty väriprofiili, joka sisältää datan spektrisiä tai väriominaisuuksia. Kuvassa 4 näkyy, kuinka eri profiileja käytetään värintoistoprosessin eri vaiheissa.

Koska Natural Vision -formaatin määrittely sisältää tällä hetkellä vain väriprofiilin osuuden, monet tallennettavan spektrikuvan ominaisuudet riippuvat myös siitä kuvaformaatista, jota käytetään kuvadatan tallentamiseen. Esimerkiksi tallennettavien arvojen tavu- ja/tai bittijärjestys ja tarkkuus, tiedoston ja kuvan maksimikoko sekä kuvan pakkaaminen ovat kaikki kuvatiedoston ominaisuuksia. Natural Vision -formaatista julkaistiin 10.5.2004 TC8-07-työryhmän testikäyttöön versio, joka sisältää profiilidatan lisäksi myös kuvadataa. Kyseisen version tiedostorakenne on esitetty taulukossa 2.



Kuva 4: Natural Vision -formaatin väriprofiilien merkitys värintoistoprosessissa (Natural Vision, 2003).

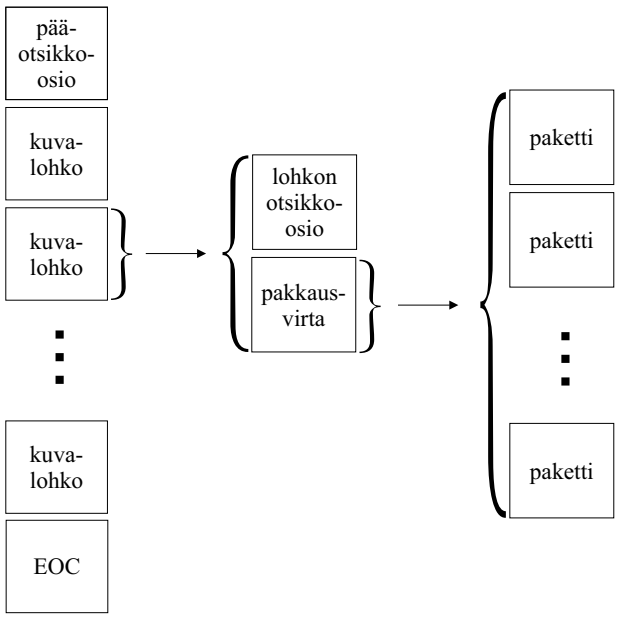
Taulukko 2: Natural Vision -tiedoston rakenne.

Tiedoston osio		**Tavua**
Otsikko-osio		48
Tunnistekenttätaulukko		64
	kenttien lukumäärä	4
	vhdr-kenttä (viittaa kuvadatan otsikkoon)	12
	chdr-kenttä (viittaa väriprofiilidatan otsikkoon)	12
	mvi-kenttä (viittaa kuvadatan tietoihin)	12
	mci-kenttä (viittaa väriprofiilidatan tietoihin)	12
	cpr-kenttä (viittaa tekijänoikeustietoihin)	12
Tunnistekenttädata		1032
	kuvadatan otsikko	140
	väriprofiilidatan otsikko	88
	kuvadatan tiedot	272
	väriprofiilidatan tiedot	272
	tekijänoikeustiedot	260
Kuvadata		
Väriprofiilidata		

4.3 JPEG2000

JPEG2000 (Taubman & Marcellin, 2002) on viimeisin Joint Photographic Experts Groupin (JPEG) kehittämä kansainvälinen standardi. JPEG2000-standardin osassa 1 on kuvanpakkauksen lisäksi määritelty myös yksinkertainen tiedostoformaatti, JP2, johon voidaan tallentaa kuvasta JPEG2000-koodivirran lisäksi myös muuta tarvittavaa tietoa. Yksinkertaisimmillaan JPEG2000-koodivirta (kuva 5) koostuu otsikko-osiosta ja sen jälkeisistä kuvalohkoista. Jokaisella kuvalohkolla on oma otsikko-osionsa, jota seuraa sarjasta paketteja muodostuva pakkausvirta. Kuvalohkojen otsikko-osioissa on tieto siitä, kuinka pakkausvirran sisältämät paketit saadaan purettua.

JPEG2000:ssa koodivirran arvot tallennettu merkitsevin tavu ensin (big endian -muoto) ja maksimimäärä bittejä yhtä pikseliä kohden on 38. Tiedoston maksimikoko on 2^{64} tavua. Jotta spektrikuvia voitaisiin tallentaa JP2-formaatissa, tallentamista varten olisi määriteltävä, kuinka aallonpituusarvot, spektrin tyyppi sekä rekonstruointitiedot talletettaisiin tiedoston metadataan. JPEG2000-kuvan dimensiot ja resoluutiot tallennetaan tiedoston otsikko-osioon, ja spektrikuvan kanavat voitaisiin tallentaa tiedostoon sellaisenaan. JPEG2000-formaatissa voidaan käyttää häviötöntä tai häviöllistä pakkausta.



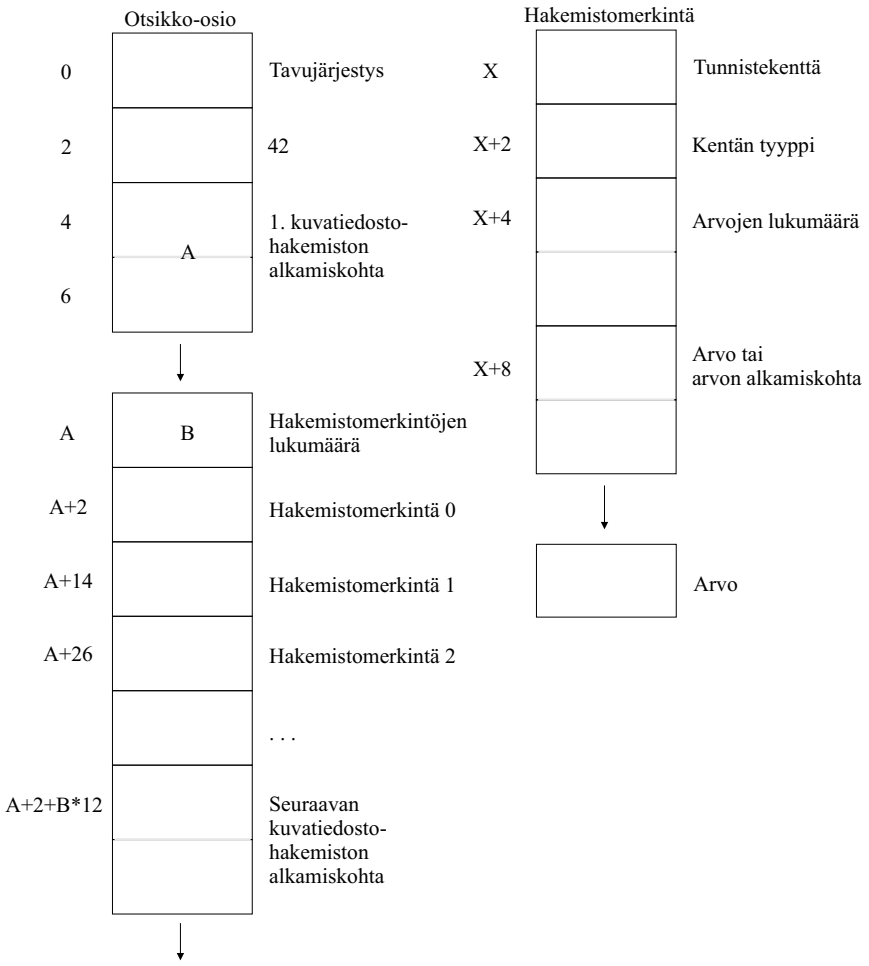
Kuva 5: JPEG2000-koodivirran rakenne (Taubman & Marcellin, 2002).

4.4 TIFF / GeoTIFF

TIFF (Tagged Image File Format) (Adobe, 1992; Brown & Shepherd, 1995) on tunnistekenttiä käyttäen toteutettu tiedostoformaatti, joka on tarkoitettu rasterikuvien tallentamiseen ja vaihtamiseen eri tahojen välillä. TIFF-kuvien alkuperäinen käyttötarkoitus oli tietokonepohjaisissa julkaisujärjestelmissä, mutta nyttemmin TIFFiä käytetään myös video- ja faksisovelluksissa, lääketieteellisessä ja satelliittikuvantamisessa sekä dokumenttien tallentamisessa. TIFF-formaattiin on tehty laajennus, GeoTIFF (Ritter & Ruth, 1995), jossa määritellään lisätunnistekenttiä maantieteellisen datan tallentamista varten.

TIFF-tiedosto alkaa kahdeksan tavun kokoisella otsikko-osiolla, jossa on kuvaan liittyvän tiedon lisäksi myös osoitin kuvatiedostohakemistoon (image file directory, IFD). Kuvatiedostohakemisto sisältää informaatiota kuvasta ja myös osoittimet itse kuvadataan. Hakemisto rakentuu yksittäisistä hakemistomerkinnöistä (directory entry). TIFF-tiedoston rakenne on esitetty kuvassa 6. GeoTIFF-tiedosto on rakenteeltaan kuten TIFF-tiedosto. Siinä on tavallisten TIFF-tunnistekenttien lisäksi metatunnistekenttiä, joihin tallennetaan maantieteellistä dataa.

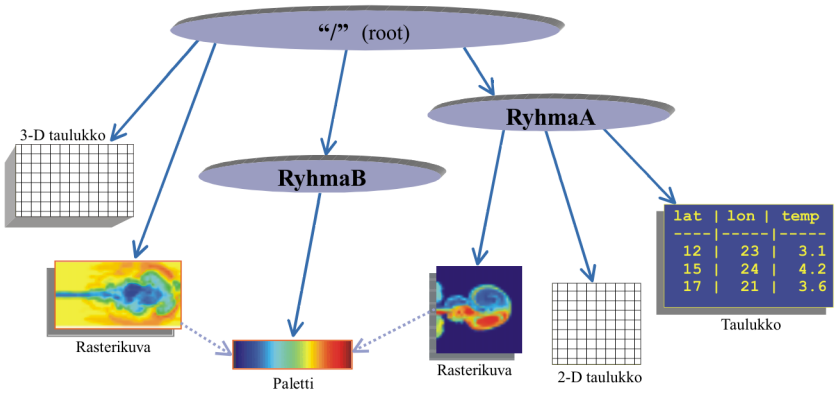
TIFF-tiedoston tavujärjestyksen voi käyttäjä määritellä itse otsikko-osion tunnisteella. TIFF-tiedostossa lukujen maksimitarkkuus on 16 bittiä/pikseli ja tiedoston koon yläraja on 2^{32} tavua. Spektridatan kanavat pystytään tallentamaan tiedostoon sellaisenaan, kun tiettyjen tunnistekenttien (SamplesPerPixel ja BytesPerPixel) arvot määritellään vastaamaan spektrikuvan ulottuvuuksia (Wilkie *& al.*, 1999). Jos TIFF-tiedostoon halutaan tallentaa spektridataa, täytyy määritellä uusi arvo tunnistekentälle, joka kertoo tiedoston sisältämän spektridatan tyypin. Lisäksi on määriteltävä, kuinka spektridatan tyyppi ja aallonpituusarvot tallennetaan tiedostoon. Jos spektridata halutaan esittää jonkin rekonstruointimatriisin avulla (esim. matriisi **B** – kaava 8 tai matriisi **Q** – kaava 11), on tällöin määriteltävä myös sen tallentaminen. TIFF-formaatissa voidaan käyttää sekä häviötöntä että häviöllistä (JPEG) pakkausta.



Kuva 6: TIFF-formaatin tiedostorakenne (Adobe, 1992).

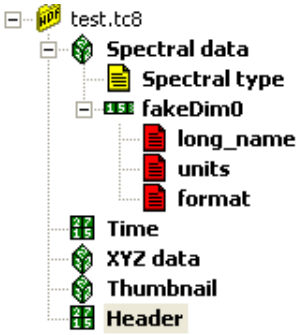
4.5 HDF / HDF5

HDF5 (HDF5, 2003) on tiedostoformaatti, joka on suunniteltu tieteellisen datan tallentamiseen. HDF5-tyyppiseen tiedostoon voidaan tallentaa lähes millaista tietoa vain. HDF5-tiedosto koostuu kahdenlaisista objekteista: datakokoelmista ja ryhmistä. Näihin molempiin voi liittyä attribuuttilista, joka sisältää lisätietoa kyseisestä objektista. Esimerkki HDF5-tiedoston sisällöstä näkyy kuvassa 7. HDF5-formaatti ei aseta tiedoston koolle mitään rajoituksia, ainoat rajoitukset ovat käyttöjärjestelmästä johtuvia. HDF5-formaatille ei ole määritelty standardia, missä muodossa spektrikuvat pitäisi tallentaa. Käytännössä HDF5-tiedostoon voidaan tallentaa spektrikuvasta mitä tahansa tietoa kuinka paljon tahansa.



Kuva 7: Esimerkki HDF5-formaattiin tallennetusta datasta (HDF5, 2003).

HDF5-formaatin edeltäjä, HDF (HDF, 2004), on toteutettu samalla periaatteella kuin HDF5. Se on kuitenkin rajoittuneempi kuin HDF5: esimerkiksi HDF-tiedosto voi sisältää enintään 20000 objektia ja HDF-tiedoston maksimikoko on 2 gigatavua. Linköpingin yliopiston tutkijat ovat tehneet Matlabille yksinkertaisen toteutuksen (Bui & Lenz, 2004), jolla voidaan tallentaa spektrikuvia HDF-muotoon. Kyseisessä toteutuksessa spektrikuva tallennetaan reflektanssispektreinä. Spektridatan lisäksi tiedostoon tallennetaan kuvan aallonpituusarvot, tristimulusarvot sekä tiedoston luontipäivä ja esikatselukuva spektrikuvasta RGB-muodossa.



Kuva 8: Linköpingin esimerkki-HDF-tiedoston rakenne (HDF, 2004).

4.6 Binaariformaatti

Yksinkertaisimmillaan spektrikuva voidaan tallentaa tiedostoon pakkaamattomassa muodossa raakana binaaridatana.

Kuvan tulkitsemiseen tarvitaan tieto kuvan dimensioista sekä aallonpituusarvoista. Nämä tallennetaan tässä esiteltävän tiedostoformaatin otsikko-osioon, kuvan dimensiot etumerkittöminä 32-bittisinä kokonaislukuina ja aallonpituusarvot 32-bittisinä liukulukuina. Kuvan arvot tallennetaan tiedostoon reflektanssispektreinä 32-bittisinä liukulukuina. Formaatissa arvot on skaalattu välille 0..1, missä 1 tarkoittaa maksimiheijastusta. Tiedoston arvot on tallennettu vähiten merkitsevä tavu ensin (little endian-muoto). Tiedoston tunnisteeksi on määritelty 3-kirjaiminen merkkijono SPB (SPectral Binary file). Tiedoston kolme ensimmäistä tavua sisältävät kyseisen tunnisteen siten, että kussakin tavu sisältää yhden char-muodossa tallennetun kirjaimen. Binaariformaatissa tallennetun spektrikuvatiedoston rakenne näkyy taulukossa 3.

Taulukko 3: Binaaritiedoston rakenne.

Tiedoston osio		**Tavua**
Tiedoston tunniste ’SPB’		3
Otsikko-osio		24
	Kuvan leveys (x)	4
	Kuvan korkeus (y)	4
	Kuvan spektrikanavien lukumäärä (n)	4
	Ensimmäinen aallonpituus	4
	Aallonpituuden resoluutio	4
	Viimeinen aallonpituus	4
Kuvadata		$x*y*n*4$

5 Formaattien tarkastelua käytännössä

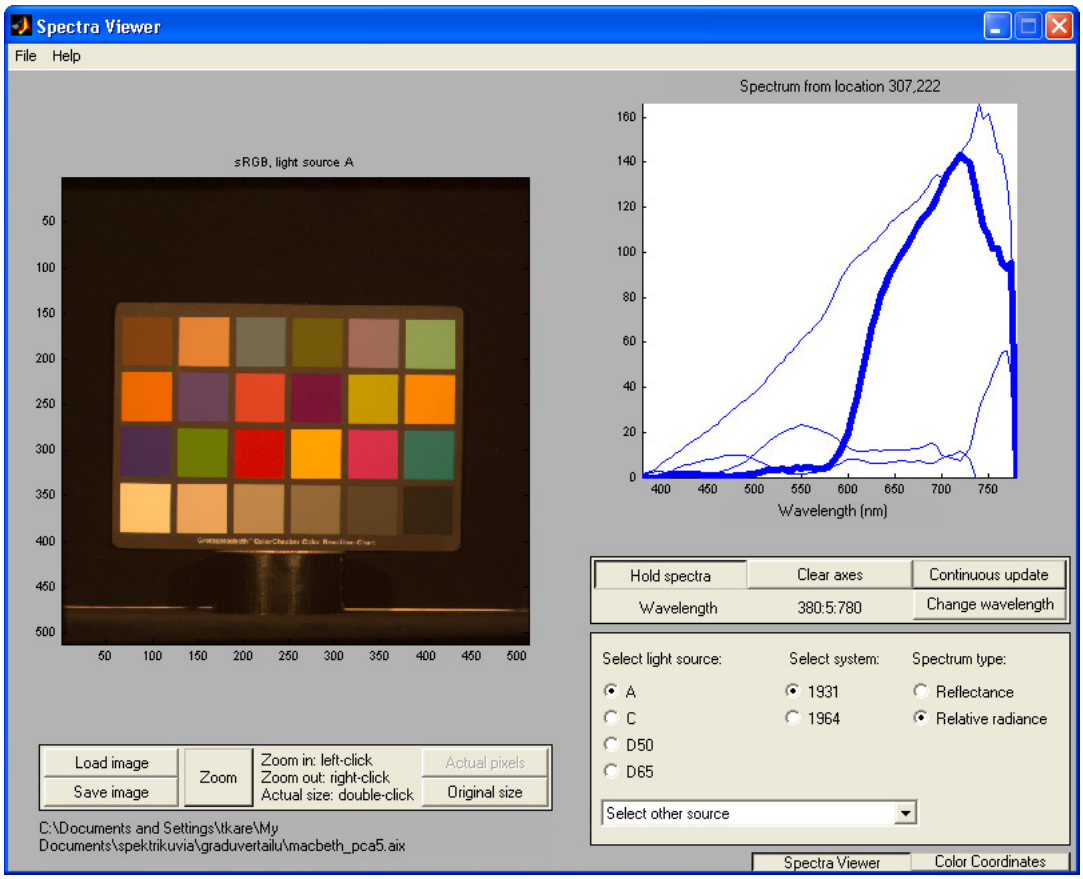
Luvussa 4 esiteltyjä kuvaformaatteja tarkasteltiin myös käytännössä. Käytännön tarkastelussa tutkittiin formaattien dokumentaatioiden selkeyttä ja mahdollisuuksia toteuttaa tarvittavat tiedoston luku- ja kirjoitusrutiinit saatavilla olleen dokumentaation perusteella. Lisäksi testattiin kuvaformaattien lukemista ja kirjoittamista. Tutkielman tekovaiheessa ei ollut saatavilla JPEG- ja TIFF-formaateissa olevia spektrikuvia, joten näiden formaattien kohdalla käytännön testausta spektrikuvaformaattien kannalta ei ollut mahdollista toteuttaa. Muiden formaattien kohdalla tarkasteltiin formaattien kehittäjiltä saatuja esimerkkitiedostoja sekä tiedostojen luku- ja kirjoitusrutiineja. Kuvien käsittelemisessä hyödynnettiin mahdollisimman paljon valmiita tiedostonkäsittelyrutiineja.

Tiedostojen testaaminen toteutettiin Matlabilla Spectra Viewer -nimistä graafista käyttöliittymää (Kareinen, 2003) hyödyntäen. Olemassa ollutta käyttöliittymää päivitettiin siten, että siihen lisättiin tiedostonkäsittelyrutiinit MUSP-, Natural Vision-, HDF- ja binaariformaateille. Spectra Viewer -käyttöliittymään luetaan spektrikuva, josta lasketaan RGB-esitys halutun valonlähteen alla. Käyttöliittymässä spektrikuvia käsitellään kolmiulotteisina, reflektanssispektrejä sisältävinä matriiseina. Spektrien lisäksi käyttöliittymässä tarvitaan tieto siitä, millä aallonpituusvälillä kuvan spektrit ovat. Käyttöliittymässä voidaan tarkastella spektrikuvasta yksittäisiä reflektanssi- ja radianssispektrejä yksi tai useampi kerrallaan (kuva 9). Spektrien lisäksi voidaan tarkastella kuvan pisteiden värikoordinaattiesityksiä (kuva 10).

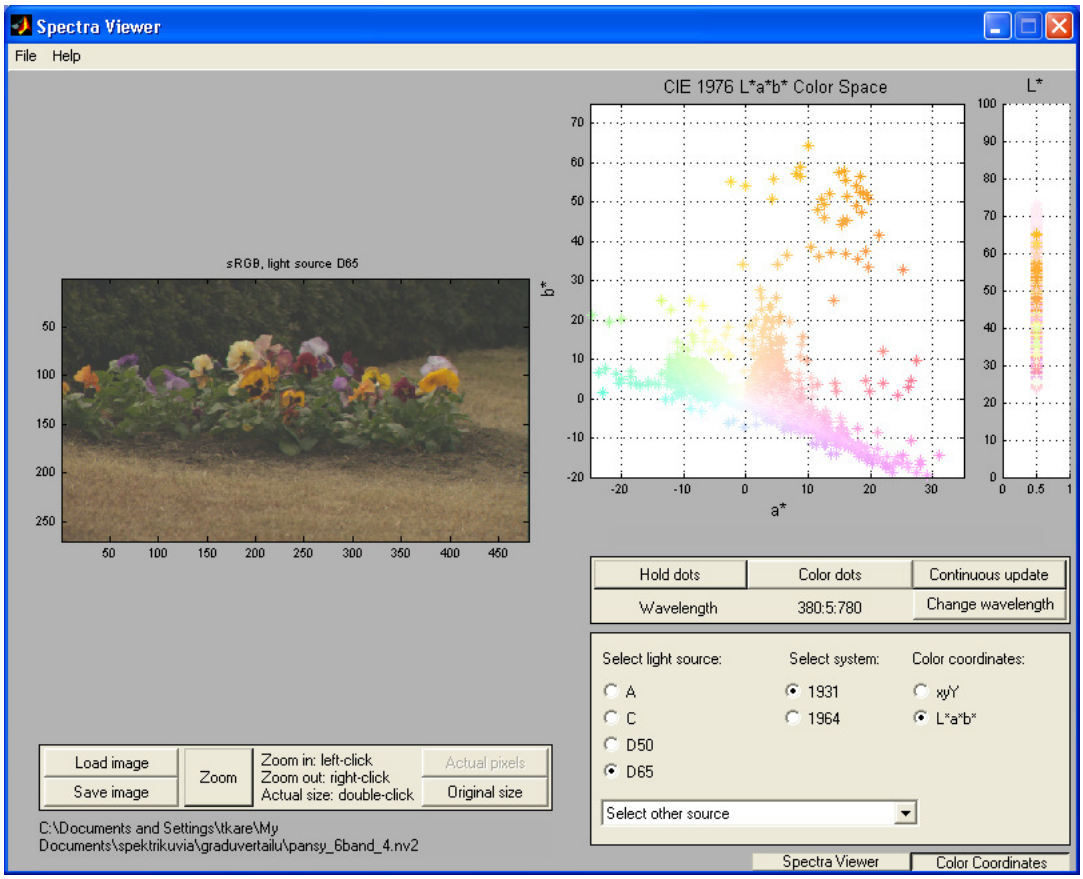
Formaattien käytännön testaus suoritettiin Windows XP -käyttöjärjestelmällä varustetulla PC:llä, jossa oli 2,4 GHz:n Pentium 4 -prosessori ja 512 Mt keskusmuistia. Matlabista käytössä oli versio 6.5.1.

5.1 MUSP

MUSP-tiedoston rakenne selvisi melko helposti dokumentaatiota lukemalla ja tiedoston sisällön lukeva ohjelmakin oli suhteellisen yksinkertainen toteuttaa. Dokumentaation versiosta 1.3.1 puuttuivat muutamien tunnistekenttien merkitysten selostukset, mikä häiritsi hieman dokumentaation kunnollista ymmärtämistä. Dokumentaatiossa ei ollut selvitetty esimerkiksi otsikko-osiossa sijaitsevan datan tyypin osoittavan kentän ei-



Kuva 9: MUSP-formaattiin tallennettu kuva GretagMacbethin ColorChecker-laudasta ladattuna Spectra Viewer -käyttöliittymään. Tarkasteltavana ovat kuvan pisteiden radianssispektriesitykset A-valonlähteen alla.



Kuva 10: Natural Vision -formaatissa oleva orvokkikuva ladattuna Spectra Viewer -käyttöliittymään. Tarkasteltavana ovat kuvan pisteiden CIELAB-värikoordinaattiesitykset.

kä spektrikehysosion pakkauksen laadun määrittelevän kentän eri arvojen merkityksiä. Spektrikehysten datan arvojen tulkitseminen oli myös hieman ongelmallista. Kehysten sisältämät arvot olivat 16-bittisiä lukuja, joiden muotoa ei dokumentaatiossa ollut selvitetty riittävästi. Luvuista ei kerrottu, ovatko ne kokonaislukuja vai desimaalilukuja eikä myöskään sitä, millä välillä kyseiset arvot on esitetty. Kyselemällä formaatin tekijöiltä epäselviin kohtiin kuitenkin nopeasti lisäinformaatiota, ja formaatin dokumentaation päivitetyssä versiossa 1.4 edellä mainitut asiat onkin selitetty tarkemmin.

MUSP-formaatissa olevan tiedoston lukurutiinin toteuttaminen onnistui suoraviivaisesti, sillä kaikki tieto oli tiedostossa peräkkäin tunnistekenttien osoittamassa järjestyksessä. Toteutettu lukurutiini palauttaa parametreinaan muun muassa rekonstruointimatriisin, spektrikehykset sekä kuvan dimensiot ja aallonpituusarvot. Näistä tiedoista saadaan muodostettua spektrikuva, kun kerrotaan spektrikehysdata rekonstruointimatriisilla ja järjestetään kertomisen tuloksena saatu data kuvan dimensioiden mukaisesti oikean kokoiseksi matriisiksi. Lukurutiini huolehtii lukuarvojen muuntamisesta oikeaan muotoon, jos käytössä oleva tietokone ei käsittele lukuja big endian -muodossa.

Algoritmi MUSP-formaatissa olevan tiedoston lukemiseen:

```
avataan tiedosto lukemista varten

luetaan tiedoston otsikko-osion tiedot, mm. tunnistekenttien lukumäärä T
luetaan tiedoston tunnistekenttätaulusta ensimmäisen kentän tiedot
siirrytään käsittelemään ensimmäistä tunnistekenttää
käsitellään kaikki T tunnistekenttää järjestyksessä

suljetaan tiedosto
```

Kirjoitusrutiinia varten spektrikuvasta laskettiin Matlabilla pääkomponenttianalyysia käyttäen käyttäjän valitsema määrä kantavektoreita rekonstruointimatriisiksi (matriisi **B** kaavassa 8) ja sisätulokuvia (matriisi **P** kaavassa 9) spektrikehyksiksi. Tiedoston kirjoitusrutiinin toteuttaminen ei ollut lukurutiinin toteuttamista hankalampaa.

Algoritmi tiedoston kirjoittamiseen MUSP-formaatissa:

```
avataan tiedosto kirjoittamista varten

kirjoitetaan tiedostoon otsikko-osion tiedot
kirjoitetaan tiedostoon tunnistekenttien tiedot tunnistekenttätauluun
    ensin rekonstruointimatriisin tiedot
    seuraavaksi spektrikehysten tiedot
kirjoitetaan tiedostoon tunnistekenttien tiedot
     tunnistekenttätaulun mukaisessa järjestyksessä

suljetaan tiedosto
```

MUSP-formaatissa olevan tiedoston lukemiseen ja kirjoittamiseen käytettyjen rutiinien lähdekoodit löytyvät liitteestä 2. MUSP-formaatin esimerkkikuvat olivat valitettavasti melko suuria, parhaimmillaan yli tuhat pikseliä suuntaansa ja aallonpituusalueena 400–700 nm 5 nm:n välein, joten koneen muistikapasiteetti täyttyi esimerkkikuvia tarkasteltaessa nopeasti. Pienempien kuvien lukeminen tallentaminen MUSP-formaattiin onnistui vaivattomasti, samoin MUSP-formaatissa tallennettujen pienempien kuvien lukeminen uudelleen käyttöliittymään.

5.2 Natural Vision

Natural Vision näytti ensivaikutelman perusteella hyvin laajaöta formaatilta. Dokumentaatiosta oli hankala saada selville, mitä tietoja kuvasta formaatissa oikeastaan tarvitaan ja miten tietojen tallentaminen tapahtuu. Lisäksi osa formaatin datatyypeistä oli suoraan tai osittain peräisin ICC:n määrittelyistä (ICC, 2004) eikä niitä ollut kuvattu dokumentaatiossa sen tarkemmin, siispä tällaisten datatyyppien määrittelyt täytyi käydä erikseen etsimässä ICC:n dokumenteista. Natural Visionin muista formaateista poikkeavan profiiliajattelumallin ymmärtäminen vei myös oman aikansa.

Natural Vision -formaatin kehittäjiltä saatiin esimerkkikuvatiedostojen lisäksi tiedoston luku- ja kirjoitusrutiinit hoitava ohjelma. Tiedoston lukeminen tapahtui siten, että ohjelmalle annettiin luettavan tiedoston nimi ja joukko muita parametreja erillisessä tekstitiedostossa. Ohjelma purki Natural Vision -formaatissa olevan tiedoston useaan

erilliseen tiedostoon. Koska esimerkkitiedostojen rakenne ei ollut kovinkaan selkeästi dokumentoitu, käytettiin edellä mainittua ohjelmaa tiedoston purkamiseen sen sijaan että tarpeellisia tietoja olisi yritetty lukea suoraan Natural Vision -formaatissa olevasta tiedostosta. Ohjelman tekemistä tiedostoista luettiin spektrikuvan muodostamisessa tarvittavat tiedot Matlabiin. Natural Vision -formaatissa olevien tiedostojen lukemiseen käytetyn rutiinin lähdekoodi on liitteessä 3. Tiedostojen kirjoittamista Natural Vision -formaattiin ei toteutettu, koska muissa formaateissa olevat kuvat eivät sisällä tarpeeksi informaatiota esimerkiksi väriprofiileista, jotka Natural Vision -formaattia varten tarvitaan.

Algoritmi Natural Vision -formaatissa olevan tiedoston lukemiseen:

```
avataan Natural Vision -tiedosto lukemista varten
luetaan tiedostosta kuvan dimensiotiedot
suljetaan tiedosto

avataan tekstitiedosto kirjoittamista varten
kirjoitetaan tekstitiedostoon tarvittavat parametritiedot
suljetaan tekstitiedosto

suoritetaan Natural Vision -formaatissa olevan tiedoston
    purkava NVIOSample.exe-ohjelma

avataan kuvadatan sisältävä tiedosto lukemista varten
luetaan kuvadata tiedostosta
suljetaan tiedosto

avataan kerroinmatriisin sisältävä tiedosto lukemista varten
luetaan matriisidata tiedostosta
suljetaan tiedosto

muodostetaan spektrikuva kertomalla kerroinmatriisi ja kuvadata
    keskenään ja skaalaamalla arvot siten, että suurin arvo on 1
```

Natural Vision -formaatin esimerkkikuvat olivat melko suuria erityisesti aallonpituussuunnassa (380–780nm 1 nm:n välein). Jos kuvat olisi ladattu Spectra Viewer -käyttö-

liittymään sellaisenaan, olisi niiden prosessointi vaatinut erittäin paljon muistia. Tästä syystä käyttöliittymään toteutettiin käyttäjälle mahdollisuus valita Natural Vision -formaatissa olevia kuvia ladattaessa aallonpituuksien näytteistysväliksi alkuperäistä 1 nm:ä suurempi arvo.

5.3 HDF / HDF5

HDF ja HDF5 tuntuivat ensivaikutelman perusteella olevan monikäyttöisiä formaatteja, joihin näyttäisi pystyvän tallentamaan melkeinpä millaista dataa tahansa. Molemmille formaateille on Internetistä saatavilla useita lukuohjelmia ja testitiedostoja. Matlabissa on ollut jo jonkin aikaa tuki HDF-tiedostojen lukemiseen ja kirjoittamiseen, versiossa 6.5.1 on toteutettu myös HDF5-tiedostojen lukeminen ja uusimmassa versiossa 7 tuetaan myös HDF5-tiedostojen kirjoittamista.

Kummallekaan, ei HDF- eikä HDF5-formaatille, ole olemassa standardoitua määrittelyä spektrikuvien tallentamista varten. HDF-formaatissa olevien spektrikuvien luku- ja kirjoitustestit tehtiin käyttäen Buin ja Lenzin (2004) määrittelemiä tiedostoja ja toteuttamia rutiineja. Testiformaatille ei ollut erillistä dokumentaatiota, pakettiin kuuluivat testikuvien lisäksi ainoastaan tiedoston lukemiseen ja kirjoittamiseen tarvittavat rutiinit. Buin ja Lenzin testitarkoitukseen esittelemä HDF-toteutus Matlabille on helppokäyttöinen, mutta melko suppea. Testiformaatti sisältää kuitenkin kaiken oleellisen tiedon spektrikuvasta: spektrikuvamatriisin, aallonpituusalueen rajat sekä aallonpituuskomponenttien määrän.

5.4 Binaariformaatti

Binaariformaatti toteutettiin vertailukohteeksi tutkielman muille formaateille, joten sen määrittely ja tiedostonkäsittelyrutiinit olivat alusta asti selvillä. Binaariformaatti ei sisällä spektrikuvan pakkausmahdollisuutta, mutta muuten kaikki spektrikuvatiedoston perusominaisuudet (spektrikuvamatriisi, aallonpituusalueen rajat sekä aallonpituuskomponenttien määrä) löytyvät tiedoston määrittelystä. Luku- ja kirjoitusrutiinit ovat yksinkertaisia kuten itse formaattikin.

Tiedoston lukeminen Matlabissa:

```
avataan tiedosto lukemista varten

luetaan tiedoston alusta tiedoston tunniste
luetaan tiedostosta kuvan dimensiot
luetaan tiedostosta kuvan aallonpituusarvot
luetaan tiedostosta kuvadata
    ja muunnetaan se kolmiuloitteiseksi matriisiksi

suljetaan tiedosto
```

Tiedoston kirjoittaminen Matlabissa:

```
avataan tiedosto kirjoittamista varten

kirjoitetaan tiedoston alkuun tiedoston tunniste
kirjoitetaan tiedostoon kuvan dimensiot
kirjoitetaan tiedostoon kuvan aallonpituusarvot
kirjoitetaan tiedostoon kuvadata

suljetaan tiedosto
```

6 Formaattien vertailua

Tässä tutkielmassa käsitellyt formaatit jakautuvat kahteen ryhmään: niihin, jotka on luotu suoraan spektrikuvien tallentamista varten ja niihin, jotka on alun perin suunniteltu perinteisten, yleensä kolmikanavaisten digitaalisessa muodossa olevien kuvien tallentamiseen. Tässä tutkielmassa esillä olevat perinteiset kuvaformaatit ovat kuitenkin sellaisia, että niitä pystytään kohtuullisin muutoksin käyttämään myös spektrikuvien tallentamiseen. Luvussa 4 esitellyistä formaateista MUSP, Natural Vision ja binaariformaatti ovat suoraan spektrikuvien tallentamista varten tarkoitettuja formaatteja. JPEG2000 ja TIFF ovat perinteisiä kuvantallennusformaatteja. HDF5 ja sen edeltäjä HDF ovat tieteellisen datan tallennusformaatteja, joille ei ole olemassa standardoituja määrittelyjä spektrikuvien tallentamista varten.

Luvussa 3 tutustuttiin spektrikuvan ominaisuuksiin ja luvussa 4 tutkittiin, millaisia ominaisuuksia erilaisilla kuvaformaateilla on. Ominaisuuksia tarkasteltaessa keskityttiin pääasiassa luvussa 3 esiteltyihin spektrikuvan perusominaisuuksiin. Tässä luvussa vertaillaan esiteltyjen formaattien ominaisuuksia luvussa 3 esiteltyjen spektrikuvien perusominaisuuksien pohjalta. Yhteenveto perusominaisuuksista on tehty taulukkoihin 4 ja 5. Sovelluskohtaiset lisäominaisuudet jätetään huomiotta, koska ne vaihtelevat runsaasti sovelluksesta toiseen. Eri tarkoituksia varten määriteltyjä formaattien lisäominaisuuksia ei juurikaan pysty vertailemaan keskenään. Kattava CIE:n TC8-07-työryhmän kokoama spektrikuvaformaatin standardointityöhön liittyvä formaattien ominaisuusluettelo löytyy liitteestä 1. Kuten taulukosta 4 voidaan nähdä, tässä tutkielmassa esitellyt formaatit ovat perusominaisuuksiltaan hyvin samankaltaisia. Suurimmat formaattien väliset erot tulevat esille taulukossa 5 esitellyissä kuvan koolle asetetuissa rajoituksissa ja kuvan arvojen tallennustarkkuudessa.

Formaattien dokumentaatioiden skaala vaihtelee muutaman sivun pituisesta dokumentaatiosta (MUSP) yli sadan sivun mittaiseen PDF-tiedostoon (TIFF) ja JPEG2000-formaatin kohdalla monisatasivuiseen kirjaan. Natural Vision -formaatin kohdalla dokumentaation laajuus on muutamia kymmeniä sivuja, joiden sisällön ymmärtämistä hankaloitti lukuisten eri profiilien toisistaan vain hieman poikkeavat määrittelyt ja lukuiset matemaattiset kaavat ilman sen kummempia selityksiä. Natural Vision- ja JPEG2000-formaattien dokumentaatiot osoittautuivat muita tarkasteltuja formaatteja monimutkaisemmiksi.

Kaikki formaatit on määritelty siten, että kaikki data tallennetaan yhteen tiedostoon. Natural Vision -formaatin kohdalla tosin nykyinen formaatin määrittely ei suoraan sisällä kuvatiedoston määrittelyä. Määrittelyssä esitetään kuitenkin, että profiilidata voidaan tallentaa kuvatiedoston yhteyteen. Tällöin yhtenäisyysvaatimus toteutuu myös Natural Vision -formaatissa. Natural Vision -formaatin testiversiossa tämä ominaisuus olikin totetutettu.

Spektrikuvia tallennettaessa kuvan pakkaaminen on usein perinteisiä kuvia tärkeämpää, koska spektrikuva koostuu usein monista kymmenistä, jopa sadoista kanavista ja vie siis runsaasti tilaa kovalevyltä. Kaikissa tässä tutkielmassa käsitellyissä formaateissa luvun 4.6 yksinkertaista binaariformaattia lukuunottamatta on mahdollisuus pakkaamattoman kuvan tallentamisen sijaan tallentaa kuvadata käyttäen häviöllistä tai häviötöntä pakkausta. Formaateissa käytettäviä pakkausmetodeja ovat muun muassa JPEG ja ZIP.

Alun perin spektrikuvien tallentamiseen tarkoitetut formaatit erottuvat perinteisistä kuvaformaateista siinä, että niissä on valmiina mahdollisuus kuvan aallonpituusarvojen tallentamiseen. Natural Vision -formaatti on tarkastelluista formaateista ainoa, joka mahdollistaa useammantyyppisen spektridatan tallentamisen. MUSP- ja binaariformaattien määrittelyissä on tällä hetkellä olemassa mahdollisuus tallentaa spektridataa ainoastaan reflektanssimuodossa.

Eniten formaattien välistä vaihtelua on tiedoston kokorajoituksissa ja arvojen tallennustarkkuuksissa. Näitä ominaisuuksista löytyy yhteenveto taulukosta 5. Osassa formaatteja tiedoston koolle asettavat rajoituksia johonkin kohtaan tiedostoa osoittavien kenttien maksimikoot. Esimerkki tällaisesta rajoittavasta kentästä on tunnistekenttätaulukon kenttä, joka kertoo, missä kohdassa tiedostoa tunnistekenttädata sijaitsee. Tiedoston kokorajoitus voi aiheutua myös tietyille kuvaan liittyville arvoille asetetuista rajoista, kuten kuvan dimensioiden maksimikoosta. MUSP ja JPEG2000 mahdollistavat 2^{64} tavun kokoiset tiedostot ja Natural Vision- sekä TIFF-formaateissa tiedoston maksimikoko on 2^{32} tavua. HDF-formaattiin voi tallentaa enimmillään 2 gigatavun kokoisia tiedostoja. HDF-formaatin tiedostokokorajoitus on poistettu HDF5-formaatista, jossa tiedoston kokoon vaikuttavat ainoastaan käyttöjärjestelmän asettamat rajoitukset. Luvussa 4.6 esitellyssä binaariformaatissa kuvatiedoston maksimikoko on 2^{93} tavua.

Pelkästään spektrikuvien tallentamiseen tarkoitetuissa formaateissakin on perusominaisuuksien ulkopuolisissa määrittelyissä suuria eroja. Erot formaattien määrittelyissä

johtuvat paljolti formaattien erilaisista käyttötarkoituksista. MUSP-formaatti on suunniteltu teollisuuden käyttöön, joten sen toteutus on yksinkertainen ja selkeä eikä formaattiin sisälly kovinkaan paljoa kuvaan liittyvää oheistietoa. Natural Vision -formaatti taas on monella tavoin MUSP-formaatin vastakohta. Siinä kuvadata voidaan tallentaa profiileita käyttäen useassa eri muodossa ja erilaisin oheistiedoin käyttötarkoituksesta riippuen.

Liitteessä 1 on esitelty formaattien ominaisuuksia tarkemmin CIE:n kokoamassa kuvaformaattien ominaisuustaulukossa. Liitteen 1 taulukosta nähdään, että esiteltyjen formaattien erityisominaisuudet vaihtelevat melko laajasti. Taulukoista selviää myös, kuinka paljon erilaisia määrittelyjä perinteisiin kuvaformaatteihin jouduttaisiin tekemään, jos ne otettaisiin spektrikuvakäyttöön. MUSP ja Natural Vision sisältävät valmiiksi spektrikuvan vaatimat aallonpituusarvojen ja spektrin tyypin tallennusmahdollisuudet, muihin formaatteihin nämä ominaisuudet jouduttaisiin määrittelemään erikseen.

Luvussa 4 esitellyistä spektrikuvaformaateista tarkasteltiin luvussa 5 tarkemmin niiden formaattien käytännön toteutuksia, joista oli jo valmiiksi olemassa jonkinlainen spektrikuvatoteutus. Formaateista yksinkertaisimpia ovat HDF- ja binaariformaatit. MUSP-formaatti on näitä kahta hieman monimutkaisempi, ja eniten erilaista tietoa sisältää Natural Vision -formaatti. Koska Natural Vision -formaatti on formaateista ainoa, jossa kuvaan liitetään väriprofiilidata, muita tiedostoformaatteja ei voida järkevästi muuntaa Natural Vision -formaattiin väriprofiilidatan puuttumisen takia. Muilta osin tiedostoformaattien sisällöt ovat riittävän samankaltaisia, joten lukuunottamatta muunnoksia Natural Vision -formaattiin päin kaikki eri formaattien väliset muunnokset ovat mahdollisia (taulukko 6).

Taulukko 4: Formaattien perusominaisuuksien vertailua.

	Selkeä dokumentaatio, helppokäyttöinen	Yhtenäinen	Ei pakkausta	Häviöllinen pakkaus	Häviötön pakkaus	Aallonpituus-arvojen tallennus	Spektrin tyypille vaihtoehtoja
MUSP	OK	OK	OK	OK	OK	OK	ei tällä hetkellä
Natural Vision	∼	OK, jos profiili liitetään kuvatiedostoon	Kuvatiedoston ominaisuuksia			kiinnitetty 380–780 nm	eri profiileja
JPEG2000	∼	OK	OK	OK	OK	määriteltävä	
TIFF /GeoTIFF	OK	OK	OK	OK	OK	määriteltävä	
HDF /HDF5	OK	OK	OK	riippuu sovelluksesta	OK	määriteltävä	
Binaari	OK	OK	OK	–	–	OK	ei tällä hetkellä

Taulukko 5: Formaatteihin tallennettavan kuvan koon rajoitukset.

	Tiedoston maksimikoko	Arvojen tarkkuus	Pikselien maksimimäärä	Kanavien maksimimäärä
MUSP	2^{64} B	mielivaltainen	$(2^{32}-1)*(2^{32}-1)$	65536
Natural Vision	testiversiossa 2^{32} B = 4 GB	8, 16, 32 bittiä	testiversiossa $2^{32}*2^{32}$	testiversiossa 65536
JPEG2000	2^{64} B	max. 38 bittiä	$(2^{32}-1)*(2^{32}-1)$	16384
TIFF /GeoTIFF	2^{32} B = 4 GB	mielivaltainen	$2^{32}*2^{32}$	64000
HDF /HDF5	HDF: 2 GB HDF5: –	HDF: int, uint 8, 16, 32, 64 bittiä float 32, 64 bittiä HDF5: mielivaltainen	ei rajoitettu	ei rajoitettu
Binaari	2^{93} B	32 bittiä	$2^{32}*2^{32}$	2^{32}

Taulukko 6: Spektrikuvatiedostojen muunnokset formaatista toiseen.

		mistä			
		MUSP	Natural Vision	HDF	binaari
mihin	MUSP	ei tarpeen	OK	OK	OK
	Natural Vision	–	ei tarpeen	–	–
	HDF	OK	OK	ei tarpeen	OK
	binaari	OK	OK	OK	ei tarpeen

7 Yhteenveto

CIE:n spektrikuvatyöryhmän työskentelyn seuraaminen on ollut mielenkiintoista, sillä harvoin tarjoutuu mahdollisuutta seurata uuden standardin kehittämistä ja jopa jossain määrin osallistua kehitystyöhön. Kaikki tässä tutkielmassa esille tulleet formaatit ovat eri tahojen käyttämiä ja työryhmälle esittelemiä, joten huomioonotettavia ominaisuuksia ja näkökulmia on runsaasti. Standardin kehittämisessä onkin kyse suuremmasta urakasta kuin ensivaikutelman perusteella voisi arvella.

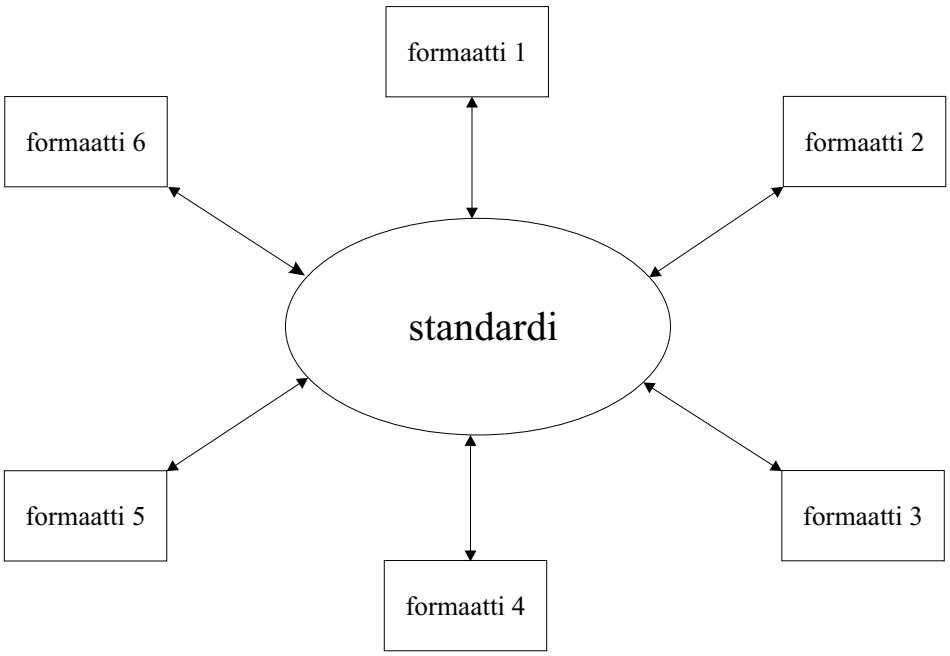
Jos spektrikuvaformaatin standardoimisen tavoitteena on nimenomaan mahdollistaa spektrikuvien vaihtaminen eri ryhmien välillä, formaatin tulee olla mahdollisimman yksinkertainen ja helppokäyttöinen. Lisäksi sen tulee sisältää mahdollisimman vähän ”ylimääräistä” dataa – tai ainakin tallennusmuodon tulee olla sellainen, että kuvan pystyy tulkitsemaan mahdollisimman vähillä tiedoilla. Kuitenkin formaatissa täytyy pystyä siirtämään kaikki sovelluksen kannalta oleellinen data. Kuten tämän tutkielman formaatteja vertaillessakin tuli esille, sovelluskohtaiset spektrikuvaformaatin tarpeet poikkeavat toisistaan paljonkin. Jos standardiformaattiin sisällytettäisiin jokaikinen tutkimusryhmien tarvitsema ominaisuus, ei standardin käyttämisellä saavutettaisi mitään hyötyä, koska useat formaatin ominaisuudet olisivat olemassa vain jotain tiettyä sovellusta varten. Pahin tilanne olisi silloin, jos formaattiin olisi aina pakko tallentaa tietyt datakentät, joita jollakin tutkimusryhmällä ei olisi ollenkaan mahdollista määrittää.

Valmiin formaatin, kuten JPEG2000, HDF ja HDF5, muokkaamisessa spektrikuvakäyttöön on sekä etuja että haittoja. Valmiille formaatille on mahdollisesti olemassa kirjastoja, joita voi käyttää datan käsittelyyn. Kirjastot eivät kuitenkaan välttämättä ole suoraan hyödynnettävissä spektrikuville, koska valmiiseen formaattiin joudutaan mitä todennäköisimmin tekemään laajennuksia spektrikuvan tallentamista varten. Tällöin vastaavat laajennukset joudutaan tekemään myös tiedoston luku- ja kirjoitusrutiineihin eli joudutaan tekemään muutoksia jo olemassa oleviin tiedostonkäsittelykirjastoihin tai tekemään kokonaan uusia kirjastoja.

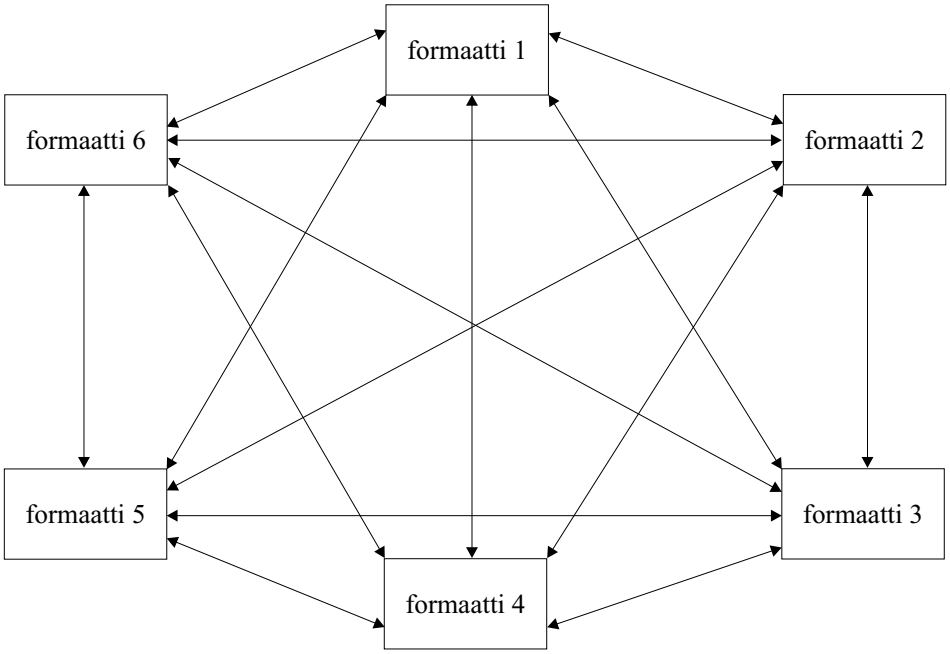
Spektrikuvan vaatimien määrittelyjen lisääminen valmiin formaatin määrittelyyn on myös huomioonotettava seikka. onnistuuko tiedoston laajentaminen suoraan jo olemassa olevia kenttiä hyödyntäen vai joudutaanko formaattiin tekemään erillisiä osia tai suurempia lisäyksiä formaatin määrittelyyn. Valmista formaattia käytettäessä voidaan myös törmätä tilanteeseen, jossa tiedostoon vaaditaan tallentamaan informaatiota,

joka ei ole ollenkaan oleellista spektrikuvia käsiteltäessä (esim. RGB-esikatselukuva). Spektrikuvaformaattia määriteltäessä optimaalisen ominaisuusyhdistelmän löytäminen ei ole kovinkaan suoraviivainen tehtävä. Määrittely voi jäädä liian ympäripyöreäksi, jolloin formaatti ei palvele tarkoitustaan. Esimerkiksi jos formaatin määrittelyssä ei ole otettu kuvan ominaisuuksia riittävän hyvin huomioon, kuvaan liittyviä tietoja voidaan joutua sijoittamaan metadatakenttiin, joiden tallennusmuoto ei ole tarkasti määritelty. Toisaalta taas määrittely voi mennä liiankin yksityiskohtaiseksi. Tällöin kuvista voidaan väkisin joutua tallentamaan tietoa, joka ei kyseisen sovelluksen kannalta ole ollenkaan tarpeellista.

Mitä sitten, jos/kun yhteinen standardiformaatti on olemassa? Kaikilla tahoilla on todennäköisesti olemassa omaan käyttöön tehtyjä ohjelmia, jotka tulkitsevat josain tietyssä formaatissa olevia tiedostoja. Muunnetaanko ohjelmat lukemaan standardiformaattia vai tehdäänkö muunnososa, joka kääntää standardiformaatissa olevan kuvan ”vanhaan” formaattiin? Tässä tosin päästään siinä mielessä helpommalla, että jokaisen ryhmän tarvitsee huolehtia pelkästään muunnoksesta oman formaattinsa ja standardin välillä (kuva 11) sen sijaan, että muunnos täytyisi tehdä jokaisen vieraan formaatin kohdalla erikseen (kuva 12).



Kuva 11: Tiedostoformaattien muunnokset standardiformaatin avulla.



Kuva 12: Tiedostoformaattien väliset muunnokset ilman standardia.

Tehtäisiinkö uudet spektrikuvasovellukset standardin valmistuttua suoraan standardia käyttäväksi? Entä olemassaolevat kuva-arkistot? Onko niitä tarpeen muuntaa standardiformaattiin vai tapahtuisiko muunnos vanhasta formaatista standardiformaattiin aina tarvittaessa eli aina, kun kuvia lähetettäisiin oman yhteisön ulkopuolelle? Jos spektrikuvatietokanta olisi julkisessa levityksessä, sen sisältämät kuvat olisi joko muunnettava standardiformaattiin tai sitten kuvien mukana olisi toimitettava ohjelma, jolla ne voitaisiin muuntaa standardiformaatin mukaiseksi. Laajan tietokannan tapauksessa jälkimmäinen vaihtoehto voisi olla kätevämpi turhan työn välttämiseksi.

Kuten kaikesta edellä mainitusta voidaan päätellä, standardiformaatin määrittäminen on kaikkea muuta kuin yksinkertainen tehtävä. Työtä on ensin formaatin määrittelemisessä ja sen jälkeen vanhojen formaattien sekä standardin välisten yhteyksien luomisessa. Toteutuessaan standardi tulee kuitenkin helpottamaan eri tahojen välistä yhteistyötä ja tietojen vaihtoa siinä määrin, että sen määrittelemisen eteen kannattaa tehdä töitä.

Viitteet

Balas, C., Papadakis, V., Papadakis, N., Papadakis, A., Vazgiouraki, E., Themelis, G. (2003) A novel hyper-spectral imaging apparatus for the non-destructive analysis of objects of artistic and historic value. *Journal of Cultural Heritage* **4**(Suppl. 1), 330-337

Brown, C.W., Shepherd, B.J. (1995) *Graphics File Formats: reference and guide*. Manning Publications Co., USA.

Bui, T. H., Lenz, R. (2004) *Quick and Dirty Implementation of HDF Interface for Multispectral Images*.

CIE, TC8-07 of Multispectral Imaging (2003, 2004) CIE:n spektrikuvatyöryhmän postituslistan viestejä ajalta 11.9.2003-13.5.2004.

Hardeberg, J. Y. (2001) *Acquisition and Reproduction of Color Images - Colorimetric and Multispectral Approaches*. Dissertation.com, USA

HDF, The National Center for Supercomputing Applications, University of Illinois (2004) *HDF Home Page*. WWW-sivusto, `http://hdf.ncsa.uiuc.edu/hdf4.html` (8.4.2004).

HDF5, The National Center for Supercomputing Applications, University of Illinois (2003) *HDF5 - A New Generation of HDF* (Introduction to HDF5, HDF5 User's Guide). WWW-sivusto, `http://hdf.ncsa.uiuc.edu/HDF5/` (6.11.2003).

Herzog, P. G., Hill, B. (2003) Multispectral Imaging and its Applications in the Textile Industry and Related Fields. *The PICS Conference*, IS&T: The Society for Imaging Science and Technology, USA, 258-263.

International Color Consortium (ICC) (2004) *ICC Profiles*. WWW-sivusto, `http://www.color.org/` (24.6.2004).

Kareinen, T. (2003) *Graafisia käyttöliittymiä spektrikuvien analysointiin*. Tietojenkäsittelytieteen erikoistyö, Joensuun yliopisto, Tietojenkäsittelytieteen laitos. (Dokumentaatio saatavilla osoitteessa `http://spectral.joensuu.fi/publications/erikoistyo_kareinen.pdf` ja ohjelmat osoitteessa `http://soho.joensuu.fi/colorlab_toolbox/`)

MacLennan, B. J. (1983) *Principles of Programming Languages: Design, Evaluation and Implementation*. Holt, Reinhart and Winston, New York.

Mehl, P.M., Chen, Y., Kim, M.S., Chan, D.E. (2004) Development of hyperspectral imaging technique for the detection of apple surface defects and contaminations. *Journal of Food Engineering* **61**(1), 67-81

MUSP Multispectral Image File Format version 1.4. Color AIXperts GmbH, Aachen, Germany (2003)

Natural Vision data file format specification version 2.0s. Akasaka Natural Vision Research Center, National Institute of Information and Communications Technology (2003)

Oja, E. (1983) *Subspace Methods of Pattern Recognition*. John Wiley & Sons, Inc., USA.

Okuyama, M., Yokoyama, N., Nakao, D., Tsumura, N., Miyake, Y. (2003) Accurate Mapping Pigmentations in Human Skin by Spatio-Temporal Modulation of Light Source in the Multi-Spectral Imaging. *The PICS Conference*, IS&T: The Society for Imaging Science and Technology, USA, 272-277.

Ritter, N., Ruth, M. (1995) *Format Specification - GeoTIFF Revision 1.0*. (Saatavilla osoitteessa `http://remotesensing.org/geotiff/spec/geotiffhome.html`, 8.1.2004)

Semeter, J., Lummerzheim, D., Haerendel, G. (2001) Simultaneous multispectral imaging of the discrete aurora. *Journal of Athmospheric and Solar-Terrestrial Physics* **63**(18), 1981-1992

Suomen standardisoimisliitto SFS ry (2004) *SFS-standardisointi*. WWW-sivusto, `http://www.sfs.fi/standard/index.html` (15.5.2004).

Taubman, D.S., Marcellin, M.W. (2002) *JPEG2000: Image Compression Fundamentals, Standards and Practice*. Kluwer Academic Publishers, USA.

TIFF 6.0 Specification Adobe Systems Incorporated, Adobe Developers Association (1992) (Saatavilla osoitteessa `http://partners.adobe.com/asn/developer/PDFS/TN/TIFF6.pdf`, 8.1.2004)

Tsumura, N., Cherdhirunkorn, K., Ikeda, T., Nakao, D., Miyake, Y. (2002) Spectral Based Color Reproduction Compatible for E-Commerce with High Compatibility. *The 10th Color Imaging Conference - Color Science and Engineering: Systems, Technologies, Applications*, IS&T: The Society for Imaging Science and Technology, USA, 246-249.

Wilkie, A., Tobler, R. F., Purgathofer, W. (1999) A Spectral Extension to the Tagged Image File Format. *The 7-th International Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media'99*. University of West Bohemia, Pilsen, Czech Republic. (Saatavilla osoitteessa `http://wscg.zcu.cz/wscg99/papers/H43-final.ps.gz`, 20.4.2004)

Wilson, P.R. (1991) Standards : Past Tense and Future Perfect? *IEEE Computer Graphics and Applications* **11**(1), 44-48

Zhang, M., Qin, Z., Liu, X., Ustin, S. L. (2003) Detection of stress in tomatoes induced by late blight disease in California, USA, using hyperspectral remote sensing. *Applied Earth Observation and Geoinformation* **4**(4), 295-310

	Helppokäyttöisyys, mm. spektrin rekonstruointi helppoa	Kanavien tallennusmuoto	Rekonstruointimatriisin tallennusmuoto	Soveltuvuus teollisuuden käyttötarkoituksiin ja kuvien vaihtoon (rekonstruointi helppoa)	Soveltuvuus arkistointitarkoituksiin
MUSP	Kyllä	Kukin omana tunnistekenttänään	Matriisitunnistekenttä	Kyllä	Kyllä, määrittelemällä lisätunnistekenttiä
NV	Jokseenkin monimutkaista	Kuvan kanavina (TIFF, JPEG2000 jne.)	Laajennettu ICC-syöttölaitteen profiili	??	Kyllä
JPEG2000	Tarvitaan oma kirjasto tarkoitusta varten, lisäksi määriteltävä spektri-metadatan muoto	Tallennettu sellaisenaan	Voidaan tallentaa metadataan	Kyllä	Tarkoitettu arkistointiin
TIFF /GeoTIFF	Lisätunnisteita määriteltävä sekä kirjastoihin tehtävä lisäyksiä tai laajennuksia	Tallennettu sellaisenaan	Ei määritelty	Kyllä	
HDF5	Lisätunnisteita määriteltävä sekä kirjastoihin tehtävä lisäyksiä tai laajennuksia	Käyttäjä voi määritellä	Ei määritelty		

	Alustariippumattomuus	Rajoitukset datan tarkkuudelle	Vapaa lisenssimaksuista	API-kirjasto saatavilla	Tiedostokoon maksimi
MUSP	Toteutettu big-endian-tavujärjestystä käyttäen	Kanavakohtainen : 65 535 Pikselikohtainen : $(2^{32}-1)*(2^{32}-1)$ Bittikohtainen: Ei ole	Kyllä	Mahdollisesti	2^{64} B
NV	Toteutettu big-endian-tavujärjestystä käyttäen	Kanavakohtainen : 65 535 Pikselikohtainen : riippuu kuvasta Bittikohtainen : 8, 16, 32 bittiä/pikseli	??	Tullaan julkaisemaan	Riippuu kuvatiedostosta
JPEG2000	Toimii kaikilla alustoilla	Kanavakohtainen : 16 384 Pikselikohtainen : $(2^{32}-1)*(2^{32}-1)$ Bittikohtainen : 38 bittiä/pikseli	Kyllä	Erilainen API jokaiselle JPEG2000:n toteutukselle.	2^{64} B
TIFF /GeoTIFF	Tavujärjestys voidaan valita tiedostokohtaisesti, joko big-endian tai little-endian	Kanavakohtainen : 64 000 Pikselikohtainen : $2^{32}*2^{32}$ Bittikohtainen : 16 bittiä/pikseli Tavukohtainen : 2^{32}	Kyllä	Tulossa (?)	2^{32} B = 4 GB
HDF5	Toimii kaikilla alustoilla	Ei rajoituksia	Toteutettu avointa lähdekoodia käyttäen	Kyllä: Java, Matlab (Matlab 7.0:ssa sekä luku että kirjoitusrutiinit)	Ei rajoituksia, ainoastaan tietokoneen tai tiedostojärjestelmän kapasiteetti voi aiheuttaa rajoituksia

	Pakkaamattoman spektrin tallentaminen mahdollista	Reflektanssin / transmittanssin / radianssin tallentaminen	Tiedoston kirjoittaminen kanava kerrallaan mahdollista	Tasoittainen pakkaus mahdollista	Lomitettu (interleaved) pakkaus mahdollista	Epälineaarinen bittien koodaus mahdollista
MUSP	Kyllä	Kyllä	Kyllä	Kyllä	Ei, mutta voidaan laajentaa	Ei, mutta voidaan laajentaa
NV	Kyllä	Kyllä	Riippuu kuvatiedostosta	Riippuu kuvatiedostosta	Riippuu kuvatiedostosta	Riippuu kuvatiedostosta
JPEG2000	Kyllä, spektriarvot ovat kanavia	Voidaan tallentaa metadataan	??			Ennen pakkaamista voidaan prosessoida kuvaa epälineaarisesti
TIFF /GeoTIFF	Kyllä	Ei määritelty	Kyllä	Kyllä	Kyllä	Vaatii lisämäärittelyjä
HDF5	Kyllä	Ei määritelty	Kyllä	Ei määritelty	Ei määritelty	Vaatii lisämäärittelyjä

	Häviötön pakkaus (ZIP, LZW jne.)	Häviöllinen pakkaus (JPEG, väreet jne.) erikseen joka kanavalle	Spatiaalinen aliotanta (erikseen joka kanavalle)	Mahdollisuus tallentaa yrityskohtaista tietoa	Tulostuslaitteen (näyttö, kirjoitin) tietojen tallentaminen mahdollista	Mahdollisuus tallentaa raakaa dataa, esim. kameran ulostuloja
MUSP	Kyllä	Kyllä (JPEG)	Kyllä, käyttämällä JPEGiä	Vaatii lisämäärittelyjä	Ei määritelty	Ei määritelty
NV	Riippuu kuvatiedosta	Riippuu kuvatiedosta	Riippuu kuvatiedosta	Vaatii lisämäärittelyjä	Kyllä, näyttölaitteiden profiili määritelty	Kyllä
JPEG2000	Kyllä	Kyllä	Kyllä	Voidaan tallentaa metadataan	Kyllä	Voidaan tallentaa metadataan
TIFF /GeoTIFF	Kyllä	Kyllä	Kyllä, jos käytetään JPEG-pakkausta	Ei määritelty	Kyllä	Ei määritelty
HDF5	Kyllä (SZIP)	Kyllä	Kyllä	Ei määritelty	Ei määritelty	Ei määritelty

	Tiedot geometristä kohdistamista varten	Joka kanavan sävykäyrä voi olla lineaarinen tai epälineaarinen	Suodattimien, sensoreiden jne. spektriset ominaisuudet	Pimeä virta joka kanavalle	Vapaasti määritettävien sovitusfunktioiden tallentaminen mahdollista	Maantieteellisen paikkainformaation tallentaminen
MUSP	Ei määritelty	Ei määritelty	Ei määritelty	Ei määritelty	Ei määritelty	Ei määritelty
NV	Oletetaan, että tallennettuna kuvatiedostoon	Kyllä	Kyllä	Kyllä	Kyllä	Ei määritelty
JPEG2000	Koodivirta sisältää referenssiristikon	?	Voidaan tallentaa metadataan	Voidaan tallentaa metadataan	Voidaan tallentaa metadataan	Voidaan tallentaa metadataan
TIFF /GeoTIFF	TIFF ei, GeoTIFF kyllä	Ei määritelty	Ei määritelty	Ei määritelty	Ei määritelty	GeoTIFF kyllä
HDF5	Ei määritelty	Ei määritelty	Ei määritelty	Ei määritelty	Ei määritelty	Ei määritelty

Tiedoston lukeminen:

```
/* This file is a Matlab mex-file.
 * When compiled in Matlab, a shared library file that can be used in Matlab *
 * for reading files in MUSP Multispectral Image File Format is generated.   *
 *
 * Author:              Tuija Jetsu, University of Joensuu, Finland
                        tuija.jetsu@cs.joensuu.fi
 * Last modification:   10.8.2004                                            */

#include <stdio.h>
#include "swap.h"
#include "matrix.h"
#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{

 FILE *spectral;
 double *spFrames, *spMatrix, *Kernel, *wlength, *resolution, *dims, tmp_double;
 char *filename, *printok;
 int i, j, buflen, status, big_endian, roundCount, pixelCount;
 uint64 offset, pos;
 unsigned long PhInt, dataWidth, dataHeight, noTags, length, frameSize, tmp_long;
 unsigned short Frames, sampleFrames, Samples, dataType;
 unsigned short byps, bips, comprType, comprLevel, tmp_short, frameOffset;
 unsigned short noFrames = 0;
 float ppiX, ppiY, wStart, wDelta, wEnd, tmp_float;
 char ftype[5], fversion[5], tag[5];

 /* --------- Processing input parameters --------- */

 /* Check for proper number of arguments. */
 if (nrhs < 1)
    mexErrMsgTxt("One input required.");
 else if (nlhs > 5)
    mexErrMsgTxt("Too many output arguments.");

 /* Both inputs must be strings. */
 if (mxIsChar(prhs[0]) != 1)
    mexErrMsgTxt("File name must be a string, give file name as 'file.aix' ");

 /* Input must be a row vector. */
 if (mxGetM(prhs[0]) != 1)
    mexErrMsgTxt("Input must be a row vector.");

 /* Get the length of the input string. */
 buflen = (mxGetM(prhs[0]) * mxGetN(prhs[0])) + 1;

 /* Allocate memory for input string. */
 filename = mxCalloc(buflen, sizeof(char));

 /* Copy the string data from prhs[0] into a C string
 * filename. If the string array contains several rows,
 * they are copied, one column at a time, into one long
 * string array. */
 status = mxGetString(prhs[0], filename, buflen);
 if (status != 0)
    mexWarnMsgTxt("Not enough space. String is truncated.");

 if (nrhs > 1) {
    /* if more than one input parameter, checking also the another one.
     * the second parameter should be string for printing options */
    if (mxIsChar(prhs[1]) != 1)
        mexErrMsgTxt("Inputs must be strings, printing can set on with parameter 'Yes' ");
    buflen = (mxGetM(prhs[1]) * mxGetN(prhs[1])) + 1;
    printok = mxCalloc(buflen, sizeof(char));
    status = mxGetString(prhs[1], printok, buflen);
    if (status != 0)
        mexWarnMsgTxt("Not enough space. String is truncated.");
 }
 else {
    buflen = 3;
    printok = mxCalloc(buflen, sizeof(char));
    printok = "No";
 }
```

```
/* --------- File processing begins --------- */
spectral = fopen(filename, "rb");

if (spectral != NULL){

   /* File opened successfully, processing file header */
   fgets(ftype, 5, spectral);
   fgets(fversion, 5, spectral);

   fread((void *)&PhInt, 4, 1, spectral);
   fread((void *)&Frames, 2, 1, spectral);
   fread((void *)&tmp_short, 2, 1, spectral);
   fread((void *)&dataWidth, 4, 1, spectral);
   fread((void *)&dataHeight, 4, 1, spectral);

   big_endian = is_big_endian();
   if (!big_endian){
       swap_u_long_4(&PhInt);
       swap_u_short_2(&Frames);
       swap_u_long_4(&dataWidth);
       swap_u_long_4(&dataHeight);
   }

   /* creating 2-D output matrix for spectral frame values */
   frameSize = dataWidth*dataHeight;
   plhs[0] = mxCreateDoubleMatrix(frameSize, Frames, mxREAL);
   spFrames = mxGetPr(plhs[0]);
   if (!spFrames){
      /* it's not worth continuing the process, if there is
       * no space for spectral frame values available. */
       mexErrMsgTxt("Out of memory");
   }

   /* if no problems with memory allocations, file processing continues */
   fread((void *)&tmp_long, 4, 1, spectral);
   if (!big_endian) swap_u_long_4(&tmp_long);
   ppiX = ((tmp_long & 0xFFFF0000) >> 16) + (float) (tmp_long & 0x0000FFFF)/65536;

   fread((void *)&tmp_long, 4, 1, spectral);
   if (!big_endian) swap_u_long_4(&tmp_long);
   ppiY = ((tmp_long & 0xFFFF0000) >> 16) + (float) (tmp_long & 0x0000FFFF)/65536;

   /* skipping 28 reserved bytes, padded with zeros */
   fseek(spectral, 28, SEEK_CUR);

   fread((void *)&noTags, 4, 1, spectral);

   /* reading away first tag-name of tag table */
   fgets(tag, 5, spectral);

   fread((void *)&offset, 8, 1, spectral);

   if (!big_endian){
       swap_u_long_4(&noTags);
       swap_u_long_8(&offset);
   }

   if (!strcmp(printok,"Yes")) {
       printf("-----------------------------------------------\n");
       printf("Header information:\n");
       printf("-----------------------------------------------\n");

       printf("File Identifier: %s\n",ftype);
       printf("File Version: %s\n",fversion);
       printf("Photometric Interpretation: %u\n",PhInt);
       printf("Number of Frames: %u\n", Frames);
       printf("Data Width: %u\n", dataWidth);
       printf("Data Height: %u\n", dataHeight);
       printf("Frame Size: %u\n", frameSize);
       printf("Pixels per Inch (x,y): %.3f, %.3f\n", ppiX, ppiY);
       printf("Number of Tags: %u\n", noTags);
       printf("Offset of the 1st tag: %u\n", offset);
   }
```

```
/* ----- Tag processing begins  ------ */
/* Finding the first tag */
fseek(spectral, offset, SEEK_SET);
for (j = 1; j <= noTags; j++) {

    pos = ftell(spectral);
    fgets(tag, 3, spectral);

    if (tag[0] == 'F' && tag[1] =='R'){
        fread((void *)&tmp_short, 2, 1, spectral);
        if (!big_endian) swap_u_short_2(&tmp_short);
        sprintf(tag, "FR%u", tmp_short);
    }
    else {
        fseek(spectral, -2, SEEK_CUR);
        fgets(tag, 5, spectral);
    }

    if (!strcmp(printok,"Yes")) {
        printf("-----------------------------------------------\n");
        printf("Tag %u: %s, offset %u\n", j, tag, pos);
        printf("-----------------------------------------------\n");
    }

    if (strcmp(tag, "A2S ") == 0) {
        /* A2S = spectral reconstruction matrix */
        fread((void *)&tmp_long, 4, 1, spectral);
        if (!big_endian) swap_u_long_4(&tmp_long);
        wStart = ((tmp_long & 0xFFFF0000) >> 16) + (float) (tmp_long & 0x0000FFFF)/65536;

        fread((void *)&tmp_long, 4, 1, spectral);
        if (!big_endian) swap_u_long_4(&tmp_long);
        wEnd = ((tmp_long & 0xFFFF0000) >> 16) + (float) (tmp_long & 0x0000FFFF)/65536;

        fread((void *)&tmp_long, 4, 1, spectral);
        if (!big_endian) swap_u_long_4(&tmp_long);
        wDelta = ((tmp_long & 0xFFFF0000) >> 16) + (float) (tmp_long & 0x0000FFFF)/65536;

        fread((void *)&sampleFrames, 2, 1, spectral);
        fread((void *)&Samples, 2, 1, spectral);
        fread((void *)&dataType, 2, 1, spectral);

        /* skipping 10 reserved bytes, padded with zeros */
        fseek(spectral, 10, SEEK_CUR);

        if (!big_endian){
            swap_u_short_2(&sampleFrames);
            swap_u_short_2(&Samples);
            swap_u_short_2(&dataType);
        }

        if (!strcmp(printok,"Yes")) {
            printf("Wavelength Values: %.3f:%.3f:%.3f\n", wStart, wDelta, wEnd);
            printf("Sample Frames : %u\n", sampleFrames);
            printf("Spectral Samples: %u\n", Samples);
            printf("Data type : %u\n", dataType);
        }

        /* creating output matrix for spectral reconstruction values */
        plhs[1] = mxCreateDoubleMatrix(sampleFrames, Samples, mxREAL);
        spMatrix = mxGetPr(plhs[1]);
        if (!spMatrix){
            mexErrMsgTxt("Out of memory");
        }

        for (i = 0; i < sampleFrames*Samples; i++) {
            if (dataType == 1){
                fread((void *)&tmp_float, sizeof(tmp_float), 1, spectral);
                if (!big_endian) swap_float_4(&tmp_float);
                spMatrix[i] = tmp_float;
            }
            else {
                fread((void *)&tmp_double, sizeof(tmp_double), 1, spectral);
                if (!big_endian) swap_double_8(&tmp_double);
                spMatrix[i] = tmp_double;
            }
        } /* for */
    } /* if tag == AS2 */
```

```
    else if (tag[0] == 'F' && tag[1] =='R'){
        /* FRxx = multispectral value frame */
        fread((void *)&byps, 2, 1, spectral);
        fread((void *)&bips, 2, 1, spectral);
        fread((void *)&comprType, 2, 1, spectral);
        fread((void *)&comprLevel, 2, 1, spectral);
        fread((void *)&frameOffset, 2, 1, spectral);

        if (!big_endian){
            swap_u_short_2(&byps);
            swap_u_short_2(&bips);
            swap_u_short_2(&comprType);
            swap_u_short_2(&comprLevel);
            swap_u_short_2(&frameOffset);
        }

        /* skipping 20 reserved bytes, padded with zeros */
        fseek(spectral, 18, SEEK_CUR);

        roundCount = 0;
        pixelCount = 0;
        for (i = 0; i < frameSize; i++) {
            tmp_short = 0;
            fread((void *)&tmp_short, byps, 1, spectral);
            if (!big_endian) swap_u_short_2(&tmp_short);
            /* some rearrangement for values in order to make them compatible for Matlab reshape */
            if ((roundCount+pixelCount*dataHeight) > frameSize){
                roundCount = roundCount+1;
                pixelCount = 0;
            }
            spFrames[noFrames*frameSize+roundCount+pixelCount*dataHeight] = (double) (tmp_short-frameOffset)/32768;
            pixelCount++;
        }

        if (!strcmp(printok,"Yes")) {
            printf("Bytes per Sample : %u\n", byps);
            printf("Bits per Sample : %u\n", bips);
            printf("Compression Type : %u\n", comprType);
            printf("Compression Quality Level : %u\n", comprLevel);
            printf("Frame offset : %f %u\n", (double) frameOffset/32768, frameOffset);
            printf("Sample values: %f %f\n", spFrames[noFrames*frameSize], spFrames[noFrames*frameSize+roundCount+pixelCount*dataHeight-1]);
        }
        noFrames++;
    } /* else if tag == FRxx */

    else if (strcmp(tag, "INAD") == 0){
        if (!strcmp(printok,"Yes")) printf("Unadjusted tag\n");
    }

    else {
        /* 3x3 Kernel tag */
        plhs[5] = mxCreateDoubleMatrix(1, 9, mxREAL);
        Kernel = mxGetPr(plhs[5]);
        for (i = 0; i < 9; i++){
            tmp_double = 0;
            fread((void*)&tmp_double, 8, 1, spectral);
            if (!big_endian) swap_double_8(&tmp_double);
            Kernel[i] = tmp_double;
        }
    }
} /* for j = 1..noTags */

fclose(spectral);

/* creating output matrix for data dimension values */
plhs[2] = mxCreateDoubleMatrix(1, 3, mxREAL);
   dims = mxGetPr(plhs[2]);
   dims[0] = (double) dataWidth;
dims[1] = (double) dataHeight;
dims[2] = (double) Samples;

/* creating output matrix for wavelength values */
plhs[3] = mxCreateDoubleMatrix(1, 3, mxREAL);
wlength = mxGetPr(plhs[3]);
wlength[0] = wStart;
wlength[1] = wDelta;
wlength[2] = wEnd;
```

```
    /* creating output matrix for resolution values */
    plhs[4] = mxCreateDoubleMatrix(1, 2, mxREAL);
    resolution = mxGetPr(plhs[4]);
    resolution[0] = ppiX;
    resolution[1] = ppiY;


  } /* if (file != NULL) */
  else {
    mexErrMsgTxt("File can't be opened");
  }
}
```

Tiedoston kirjoittaminen:

```
/* This file is a Matlab mex-file.
 * When compiled in Matlab, a shared library file that can be used in Matlab *
 * for writing files in MUSP Multispectral Image File Format is generated.   *
 *
 * Author:              Tuija Jetsu, University of Joensuu, Finland
                        tuija.jetsu@cs.joensuu.fi
 * Last modification:    10.8.2004                                            */

#include <stdio.h>
#include "swap.h"
#include "matrix.h"
#include "mex.h"

void mexFunction(int nlhs, mxArray *plhs[], int nrhs, const mxArray *prhs[])
{

 FILE *spectral;
 double *spFrames, *spMatrix, *resolution, *wlength, *dims, *frameOffset;
 char *filename, *printok;
 int i, j, buflen, status, big_endian, roundCount, pixelCount;
 uint64 offset, tagSize;
 unsigned long ppiX, ppiY, kok, des, wStart, wDelta, wEnd, tmp_long;
 unsigned long PhInt = 5, dataWidth, dataHeight, noTags, length, pos, frameSize;
 unsigned short Frames, Padding = 0, sampleFrames, Samples, dataType=1;
 unsigned short byps = 2, bips = 16, comprType = 0, comprLevel = 0, frOffset;
 unsigned short noFrames = 0, tmp_short;
 char ftype[5] = "MUSP", fversion[5] = "0100", tag[5];
 float tmp_float;

 /* --------- Processing input parameters --------- */

 /* Check for proper number of arguments. */
 if (nrhs < 7)
 mexErrMsgTxt("Seven inputs required: file name, spectral frame values, reconstruction matrix,
              image dimensions, wavelength values, image resolution and frame value offset.");

 /* First input must be string. */
 if (mxIsChar(prhs[0]) != 1)
    mexErrMsgTxt("First input must be a string (file name).");

 /* Input must be a row vector. */
 if (mxGetM(prhs[0]) != 1)
    mexErrMsgTxt("Input must be a row vector.");

 /* Get the length of the input string filename. */
 buflen = (mxGetM(prhs[0]) * mxGetN(prhs[0])) + 1;

 /* Allocate memory for input string filename. */
 filename = mxCalloc(buflen, sizeof(char));

 /* Copy the string data from prhs[0] into a C string
 * filename. If the string array contains several rows,
 * they are copied, one column at a time, into one long
 * string array. */
 status = mxGetString(prhs[0], filename, buflen);
 if (status != 0)
    mexWarnMsgTxt("Not enough space. String is truncated.");
```

```
/* --------- File processing begins --------- */
spectral = fopen(filename, "wb");

if (spectral != NULL){
   /* File opened successfully for writing    */
   /* Creating pointers to the input matrices */
   spFrames = mxGetPr(prhs[1]);
   spMatrix = mxGetPr(prhs[2]);
   dims = mxGetPr(prhs[3]);
   wlength = mxGetPr(prhs[4]);
   resolution = mxGetPr(prhs[5]);
   frameOffset = mxGetPr(prhs[6]);

   dataHeight = (unsigned long) dims[0];
   dataWidth = (unsigned long) dims[1];
   ppiX = (unsigned long) (resolution[0] * 65536);
   ppiY = (unsigned long) (resolution[1] * 65536);
   Frames = mxGetM(prhs[2]);
   sampleFrames = Frames;
   Samples = mxGetN(prhs[2]);
   frameSize = dataWidth*dataHeight;
   noTags = Frames + 1;
   big_endian = is_big_endian();
   if (!big_endian){
       /* if native byte order is little endian, swapping order before writing */
       swap_u_long_4(&PhInt);
       swap_u_short_2(&Frames);
       swap_u_long_4(&dataWidth);
       swap_u_long_4(&dataHeight);
       swap_u_long_4(&ppiX);
       swap_u_long_4(&ppiY);
       swap_u_long_4(&noTags);
   }

   /* Writing file header */
   fputs(ftype, spectral);
   fputs(fversion, spectral);
   fwrite((void *)&PhInt, 4, 1, spectral);
   fwrite((void *)&Frames, 2, 1, spectral);
   fwrite((void *)&Padding, 2, 1, spectral);
   fwrite((void *)&dataWidth, 4, 1, spectral);
   fwrite((void *)&dataHeight, 4, 1, spectral);
   fwrite((void *)&ppiX, 4, 1, spectral);
   fwrite((void *)&ppiY, 4, 1, spectral);
   fwrite((void *)&Padding, 1, 28, spectral);
   fwrite((void *)&noTags, 4, 1, spectral);

   if (!big_endian){
       /* swapping back values that are needed again later */
       swap_u_long_4(&noTags);
       swap_u_short_2(&Frames);
       swap_u_long_4(&dataHeight);
   }

   /* ---- Writing of tag-table begins ---- */

   /* writing reconstruction matrix information */
   fputs("A2S ", spectral);

   offset = 64 + noTags*20;
   tagSize = 32 + 2*sampleFrames*Samples;

   if (!big_endian){
       swap_u_long_8(&offset);
       swap_u_long_8(&tagSize);
   }
   fwrite((void *)&offset, 8, 1, spectral);
   fwrite((void *)&tagSize, 8, 1, spectral);

   /* writing frame information */
    for (i = 0; i < Frames; i++){

       fputs("FR", spectral);
       tmp_short = i;
       if (!big_endian) swap_u_short_2(&tmp_short);
       fwrite((void *)&tmp_short, 2, 1, spectral);

       if (!big_endian){
```

```
            /* swapping values back for calculations */
            swap_u_long_8(&offset);
            swap_u_long_8(&tagSize);
        }
        offset = offset + tagSize;
        tagSize = 32 + 2*frameSize;

        if (!big_endian){
            swap_u_long_8(&offset);
            swap_u_long_8(&tagSize);
        }
        fwrite((void *)&offset, 8, 1, spectral);
        fwrite((void *)&tagSize, 8, 1, spectral);
    }
    /* ---- Tag writing begins ----- */

    /* A2S = spectral reconstruction matrix tag */
    tmp_long = ftell(spectral);
    fputs("A2S ", spectral);

    wStart = (unsigned long) (wlength[0] * 65536);
    wEnd = (unsigned long) (wlength[2] * 65536);
    wDelta = (unsigned long) (wlength[1] * 65536);

    if (!big_endian){
        swap_u_long_4(&wStart);
        swap_u_long_4(&wEnd);
        swap_u_long_4(&wDelta);
        swap_u_short_2(&sampleFrames);
        swap_u_short_2(&Samples);
        swap_u_short_2(&dataType);
    }
    fwrite((void *)&wStart, 4, 1, spectral);
    fwrite((void *)&wEnd, 4, 1, spectral);
    fwrite((void *)&wDelta, 4, 1, spectral);
    fwrite((void *)&sampleFrames, 2, 1, spectral);
    fwrite((void *)&Samples, 2, 1, spectral);
    fwrite((void *)&dataType, 2, 1, spectral);
    fwrite((void *)&Padding, 1, 10, spectral);

    if (!big_endian){
        /* swapping back values that are needed again later */
        swap_u_short_2(&sampleFrames);
        swap_u_short_2(&Samples);
        swap_u_short_2(&dataType);
    }

    for (i = 0; i < sampleFrames*Samples; i++) {
        tmp_float = (float) spMatrix[i];
        if (!big_endian) swap_float_4(&tmp_float);
        fwrite((void *)&tmp_float, 4, 1, spectral);
    }

   frOffset = (unsigned short) (32768 * frameOffset[0]);

   if (!big_endian){
        swap_u_short_2(&byps);
        swap_u_short_2(&bips);
        swap_u_short_2(&comprType);
        swap_u_short_2(&comprLevel);
        swap_u_short_2(&frOffset);
    }

    for (i = 0; i < Frames; i++){
        /* FRxx = multispectral value frame tag */
        fputs("FR", spectral);
        tmp_short = i;
        if (!big_endian) swap_u_short_2(&tmp_short);
        fwrite((void *)&tmp_short, 2, 1, spectral);

        fwrite((void *)&byps, 2, 1, spectral);
        fwrite((void *)&bips, 2, 1, spectral);
        fwrite((void *)&comprType, 2, 1, spectral);
        fwrite((void *)&comprLevel, 2, 1, spectral);
        fwrite((void *)&frOffset, 2, 1, spectral);
        fwrite((void *)&Padding, 1, 18, spectral);

        roundCount = 0;
```

```
        pixelCount = 0;
        for (j = 0; j < frameSize; j++) {
            /* some rearrangement for values; getting them from Matlab matrix in row-wise order */
            if ((roundCount+pixelCount*dataHeight) > frameSize){
                roundCount++;
                pixelCount = 0;
            }
            tmp_short = (unsigned short) (32768 * (spFrames[noFrames*frameSize+roundCount+pixelCount*dataHeight]+frameOffset[0]));
            if (!big_endian) swap_u_short_2(&tmp_short);
            fwrite((void *)&tmp_short, 2, 1, spectral);
            pixelCount++;
        }
        noFrames++;
    }
      fclose(spectral);
  } /* if (file != NULL) */
  else {
    mexErrMsgTxt("File can't be opened");
  }
}
```

Luku- ja kirjoitusrutiineissa tarvittavat apuohjelmat (swap.h):

```
// modify this to satisfy needs of your operating system
typedef __int64 uint64;
// typedef long long uint64;

void swap_u_short_2(unsigned short *i2)
/* 2 byte unsigned integers */
{
 *i2=(((*i2>>8)&0xff) | ((*i2&0xff)<<8));
}

void swap_u_long_4(unsigned long *i4)
/* 4 byte unsigned long integers */
{
 *i4=(((*i4>>24)&0xff) | ((*i4&0xff)<<24) |
        ((*i4>>8)&0xff00) | ((*i4&0xff00)<<8));
}

void swap_float_4(float *f4)
/* 4 byte floating point numbers */
{
 int *i4=(int *)f4;
 *i4=(((*i4>>24)&0xff) | ((*i4&0xff)<<24) |
        ((*i4>>8)&0xff00) | ((*i4&0xff00)<<8));
}

void swap_u_long_8(uint64 *i8)
/* 8 byte unsigned long long integers*/
{
 *i8=(((*i8>>56)&0xff) | ((*i8&0xff)<<56) |
        ((*i8>>40)&0xff00) | ((*i8&0xff00)<<40) |
        ((*i8>>24)&0xff0000) | ((*i8&0xff0000)<<24) |
        ((*i8>>8)&0xff000000) | ((*i8&0xff000000)<<8));
}

void swap_double_8(double *d8)
/* 8 byte double numbers */
{
 uint64 *i8=(uint64 *)d8;
 *i8=(((*i8>>56)&0xff) | ((*i8&0xff)<<56) |
        ((*i8>>40)&0xff00) | ((*i8&0xff00)<<40) |
        ((*i8>>24)&0xff0000) | ((*i8&0xff0000)<<24) |
        ((*i8>>8)&0xff000000) | ((*i8&0xff000000)<<8));
}

int is_big_endian()
{
    long one=1;
    return !(*((char *)(&one)));
}
```

```
function img2 = nv(filename, type, sampling)
% nv reads files in Natural Vision format to Matlab
%
%       img2 = nv(filename, type, sampling)
%
%       The function reads files in Natural Vision Format to Matlab.
%
%       filename = string containing name of file where data is be stored
%       type = 0 or 1, indicates whether file should be written (0) or read (1)
%       sampling = wavelength sampling rate, wavelength values are 380:sampling:780
%       img2 = 3-D spectral image
%
%       At the moment only reading Natural Vision files is implemented
%
%       Usage: img2 = nv('samplefile.nv2', 1, 5)
%
%       2004-08-06, Tuija Jetsu

if (type == 1)
fid = fopen(filename,'r','ieee-le');
fseek(fid, 36, 'bof');
width  = fread(fid, 1, 'uint32');
fseek(fid, 40, 'bof');
height  = fread(fid, 1, 'uint32');
fseek(fid, 196, 'bof');
bands  = fread(fid, 1, 'uint16');
fseek(fid, 198, 'bof');
depth = fread(fid, 1, 'uint16');
fclose(fid);

mkdir('spvtmp')
fid = fopen('iosample.txt','w');
fprintf(fid,'menu=%u\n',type);
fprintf(fid,'width=%u\n',width);
fprintf(fid,'hight=%u\n',height);
fprintf(fid,'depth=%u\n',depth);
fprintf(fid,'chnel=%u\n',bands);
fprintf(fid,'w_hss=spvtmp/W_HSS.hss\n');
fprintf(fid,'w_trc=spvtmp/W_TRC.trc\n');
fprintf(fid,'w_ils=spvtmp/W_ILS.ils\n');
fprintf(fid,'w_ren=spvtmp/W_REN.ils\n');
fprintf(fid,'w_sts=spvtmp/W_STS.sts\n');
fprintf(fid,'w_cmf=spvtmp/W_CMF.cmf\n');
fprintf(fid,'w_d2r=spvtmp/W_D2R.txt\n');
fprintf(fid,'w_d2s=spvtmp/W_D2S.txt\n');
fprintf(fid,'w_d2c=spvtmp/W_D2C.txt\n');
fprintf(fid,'in=s\n',filename);
fprintf(fid,'out_img=spvtmp/tmp.out\n');
fclose(fid);

    system('NVIOSample.exe');

fid = fopen('spvtmp/tmp.out', 'rb');
str = ['ubit' num2str(depth/bands)];
img = fread(fid, width*height*bands, str);
fclose(fid);
img = reshape(img, [bands width*height])';

fid = fopen('spvtmp/W_D2R.txt', 'rb');
tline = fgets(fid);
W_D2R = fscanf(fid,'%g %g',[bands inf]);
fclose(fid);
W_D2R = W_D2R(:, 1:sampling:401);

img = img * W_D2R;
img = reshape(img, [width height size(img,2)]);
img2 = zeros([height width size(img,3)]);
for i = 1:size(img,3)
        img2(:,:,i) = fliplr(rot90(squeeze(img(:,:,i)),-1));
end;
clear img;
img2 = img2/max(max(max(img2)));

    rmdir('spvtmp','s');
    delete('iosample.txt');
else
    disp('Write routine for Natural Vision images not implemented')
end;
```

Tiedoston lukeminen:

```
function [img, wavelength] = read_spb(filename)
% READ_SPB reads files in Spectral Binary Format to Matlab
%
%       [img, wavelength] = read_spb(filename)
%
%       The function reads files in Spectral Binary Format to Matlab. It returns
%       spectral image as 3-D matrix and wavelength values as 3-component vector.
%
%       filename = string containing name of spectral file from where data will be read
%       img = 3-D spectral image
%       wavelength = vector including wavelength values, [start resolution end]
%
%       This function is meant for reading .spb-files.
%
%       Usage: [img, wavelength] = read_spb('spectralfile.spb')
%
%       write_spb
%
%       2004-08-06, Tuija Jetsu

fid = fopen(filename, 'rb','ieee-le');
tmp = fread(fid,3,'char');
dims = fread(fid,3,'uint32');
wavelength = fread(fid,3,'float32');
img = fread(fid, dims(1)*dims(2)*dims(3), 'float32');
img = reshape(img, [dims(1) dims(2) dims(3)]);
fclose(fid);
```

Tiedoston kirjoittaminen:

```
function write_spb(filename, img, wavelength)
% WRITE_SPB writes files to Spectral Binary Format in Matlab
%
%        write_spb(filename, img, dims, wavelength)
%
%       The function writes files to Spectral Binary Format in Matlab.
%
%       filename = string containing name of spectral file where data will be stored
%       img = 3-D spectral image
%       wavelength = vector containing wavelength values, [start resolution end]
%
%       This function is meant for writing .spb-files.
%
%       Usage: write_spb('spectralfile.spb', img, wavelength)
%
%       read_spb
%
%       2004-08-06, Tuija Jetsu

fid = fopen(filename, 'w+b','ieee-le');
fwrite(fid,'SPB','char');
fwrite(fid,size(img),'uint32');
fwrite(fid,wavelength,'float32');
fwrite(fid,img,'float32');
fclose(fid);
```