TAMPEREEN TEKNILLINEN YLIOPISTO

MARKO VIITANEN

H.264 LIIKKEENESTIMOINNIN TESTAUS JA OPTIMOINTI

Kandidaatintyö

Tarkastaja: Erno Salminen
17. kesäkuuta 2010

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Signaalinkäsittelyn ja tietoliikennetekniikan koulutusohjelma
MARKO VIITANEN: H.264 liikkeenestimoinnin testaus ja optimointi
Kandidaatintyö, 25 sivua
Kesäkuu 2010
Pääaine: Tietokone- ja digitaalitekniikka
Tarkastaja: Erno Salminen, Tietokonetekniikan laitos
Avainsanat: H.264, MPEG-4, liikkeenestimointi,videonkoodaus

Tässä työssä esitellään H.264 koodausta liikkeenestimointiin painottuen. Työssä toteutettiin ohjelmisto laitteistopohjaisen liikkeenestimointilohkon testaamiseen. Lohko kehitettiin Tampereen teknillisen yliopiston tietokonetekniikan laitoksella. Lisäksi ohjelmistoon toteutettiin uusia algoritmeja, joihin laitteistolohkon suorituskykyä verrattiin. Työssä käsitellään myös erilaisien optimointimenetelmien käyttöä liikkeenestimoinnin näkökulmasta. Näillä menetelmillä saatuja tuloksia tullaan hyödyntämään laitteistolohkon jatkokehityksessä.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Digital and Computer Electronics
MARKO VIITANEN : Verification and Optimization of H.264 Motion Estimation
Bachelor of Science Thesis, 25 pages
June 2010
Major: Digital and Computer Electronics
Examiner: Erno Salminen, Department of Computer Systems
Keywords: H.264, MPEG-4, motion estimation, video compression

This paper describes H.264 video coding with motion estimation being emphasised. In the work, a software was made for testing the hardware-based motion estimation block. The block was developed in the Department of Computer Systems at the Tampere University of Technology. Furthermore, new algorithms were developed and compared with the algorithms supported by the hardware. In the work, the use of different optimisation methods is discussed from the motion estimation point of view. Results that have been obtained with these methods will be used for further development of the hardware block.

ALKUSANAT

Tämä kandidaatintyö ja siihen liittyvä tutkimus on tehty Tampereen Teknillisen Yliopiston tietokonetekniikan laitoksella kesäkuun 2009 ja kesäkuun 2010 välisenä aikana.

Haluan kiittää työtoveriani Jarno Vannetta, jonka ansiosta tämä työ on lähtenyt liikkeelle. Kiitos myös työn tarkastajalle Erno Salmiselle, joka antoi työn edetessä hyviä neuvoja.

Tampereella, kesäkuun 17. päivänä 2010
Marko Viitanen

SISÄLLYS

LYHENTEET JA MERKINNÄT

BBGDS	Block-Based Gradient Descent Search
BMA	Block-Matching Algorithm
CDS	Cross-Diamond Search
CIF	Common Intermediate Format (352×288 kuvapistettä)
D1	videomuoto, 720×576 kuvapistettä
DCT	Discrete Cosine Transform, kosinimuunnos
FBSME	Fixed Block-Size Motion Estimation
FME	Fractional-pel Motion Estimation
FS	Full Search
H.264	ITU-T:n sekä ISO:n yhdessä kehittämä videonpakkausstandardi
HDTV	High-Definition Television
HEXBS	Hexagon-Based Search
HW	Hardware
IME	Integer-pel Motion Estimation
MB	Macroblock, makrolohko
MPEG	Moving Picture Experts Group
MRF	Multiple Reference Frames
MSE	Mean Squared Error
MV	Motion Vector, liikevektori
NDS	New Diamond Search
PMV	Predicted Motion Vector
PSNR	Peak Signal-to-Noise Ratio
QCIF	Quarter Common Intermediate Format (176×144 kuvapistettä)
QP	Quantization Parameter

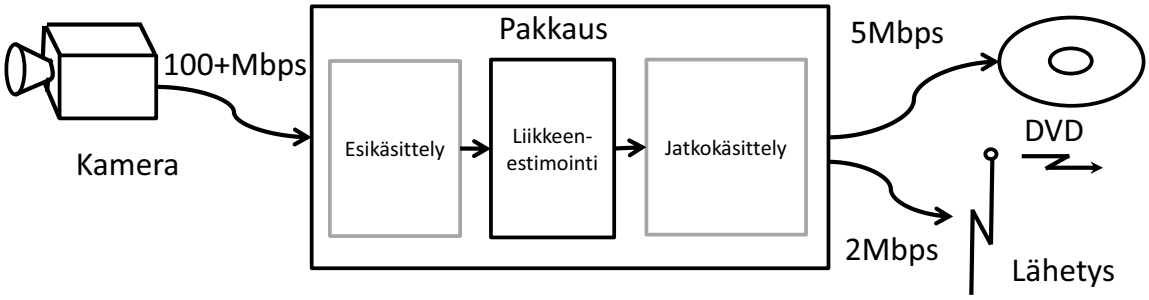
SAD	Sum of Absolute Differences
SDTV	Standard-Definition Television
SW	Software
TSS	Three-Step Search
UMHexagonS	Unsymmetrical-cross Multi-Hexagon-grid Search
VBSME	Variable Block-Size Motion Estimation

1. JOHDANTO

Liikkeenestimointi on eräs raskaimpia operaatioita videonpakkauksessa ja sen käyttämä prosessointiaika voi olla 40-60% kokonaisajasta. Uudenaikaiset videonpakkausstandardit tekevät tästä operaatioista vielä raskaamman mahdollistamalla monipuolisen lohkojen jakamisen sekä liikevektorien ennustamisen.

Tässä kandidaatintyössä esitellään ohjelmisto, jolla testataan H.264-videonpakkausstandardin mukaisesti toimivaa laitteistopohjaista liikkeenestimointia. Kuvassa 1.1 on esitetty työn aihealue. Ohjelmisto on suunniteltu ja toteutettu Tampereen teknillisen yliopiston tietokonetekniikan laitoksella. Aiemmin testauksen kohteena ollut laitteisto toteutti ainoastaan vanhempien videonpakkausstandardien kuten MPEG-2 (engl. Moving Picture Experts Group), H.263 ja MPEG-4 Visual mukaista kiinteän lohkokoon liikkeenestimointia. Uudistettu versio pystyy toteuttamaan H.264-videonpakkausstandardin mukaista vaihtuvan lohkokoon liikkeenestimointia, joka vaatii testauksessa erilaisen lähestymistavan. Testausohjelmiston kehityksen yhteydessä tutkittiin ja kehitettiin myös algoritmien erilaisia nopeutustekniikoita. Kehitettyjä tekniikoita ja tuloksia tullaan hyödyntämään laitteiston jatkokehityksen aikana.

Työssä kerrotaan aluksi yleisesti videonpakkauksesta sekä liikkeenestimoinnista niiltä osin, kun työn ymmärtämisen kannalta on oleellista. Tämän jälkeen esitellään aiemmin toteutettuja ohjelmistoja ja siirrytään kertomaan työssä toteutetusta ohjelmistosta. Työn lopussa esitetään yhteenveto toteutetun ohjelmiston suorituskyvystä.



Kuva 1.1: Työn aihealue.

2. H.264 VIDEONPAKKAUSSTANDARDI

Videon pakkaaminen perustuu peräkkäisten kuvien samankaltaisuuteen, jota voidaan hyödyntää kopioimalla osia edellisestä kuvasta seuraavaan. Normaalisti kuvat jaetaan 16×16 pikselin kokoisiin ns. makrolohkoihin, joita käsitellään erikseen. Ensimmäinen videosekvenssissä esiintyvä kuva on kuitenkin pakattava perinteisillä kuvanpakkauksen menetelmillä. Jokaiselle lohkolle tehdään tässä tapauksessa kosinimuunnos eli DCT (engl. Discrete Cosine Transform), joka muuntaa kuvan informaation taajuustason esitysmuotoon. Taajuustason esitystä voidaan kvantisoida eli pyöristää tietyille tasoille, pyöristyksen vuoksi kyseessä on häviöllinen pakkaus. Lopuksi saadut arvot pakataan vaihtuvan sananpituuden pakkauksella (entropiakoodaus), jossa yleisimmin esiintyvät arvot esitetään lyhyemmällä koodilla kuin harvemmin esiintyvät arvot. Kun saadulle binaarijonolle tehdään käänteiset operaatiot, saadaan alkuperäistä kuvalohkoa vastaava lohko, jossa pikseleiden arvot ovat hieman vääristyneet riippuen kvantisoinnin suuruudesta.

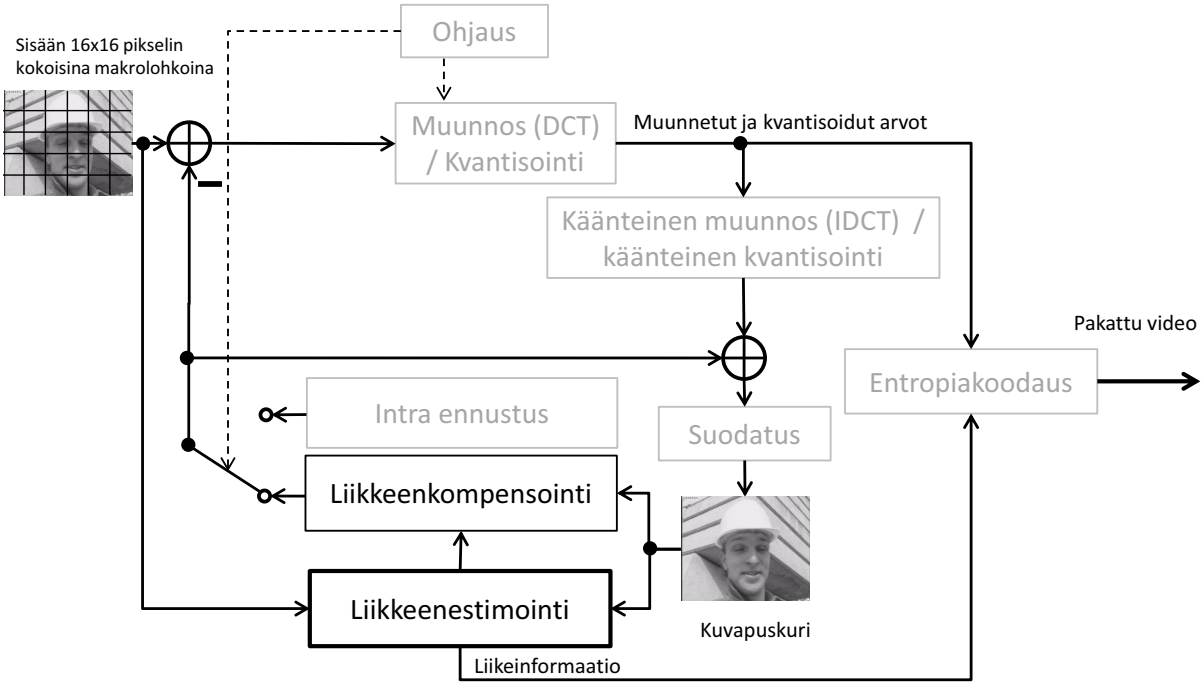
Kun edellinen kuva on olemassa, voidaan sitä hyödyntää ja etsiä käsiteltävää makrolohkoa muistuttavaa lohkoa. Jos löydetään lohko, joka muistuttaa käsiteltävää lohkoa, tallennetaan ns. liikevektori, joka osoittaa, mistä vastaavuus on löytynyt. Lisäksi lohkojen pikseleiden välinen ero lasketaan ja sille tehdään DCT ja kvantisointi. Liikevektorien avulla purkaja voi kopioida datalohkon tässä tapauksessa edellisestä kuvasta eli tehdä ns. liikkeenkompensoinnin. Näin varsinaisia pikseliarvoja ei tarvitse siirtää tälle lohkolle. Tämä vähentää tarvittavaa datamäärää verrattuna siihen, että nykyinen kuva esitettäisiin ilman toista kuvaa.

Kuvat voidaan jaotella sen perusteella käytetäänkö kuvan uudelleenkokoamisessa hyväksi muita kuvia vai pelkästään saman kuvan sisältä löytyvää tietoa. Intra-kuvat muodostetaan täysin riippumatta edellisistä kuvista, eikä liikkeenestimointia tarvita, joten ne eivät kuulu tämän työn aihealueeseen. Sen sijaan Inter-kuvien koodauksessa käytetään hyväksi edellisiä ja/tai tulevia kuvia koittaen löytää niistä yhtenevyyksiä nykyisen kuvan kanssa. Tässä työssä keskitytään kuvassa 2.1 tummennettuun Liikkeenestimointi-lohkoon, joka tuottaa liikevektoreita vertaillessaan tarkasteltavaa makrolohkoa edelliseen kuvaan. Liikevektorin valintaan vaikuttaa myös QP-arvo (engl. Quantization Parameter), joka voidaan ajatella pakkauksen tehokkuutena. QP voi olla arvoltaan 0-52 ja suuri arvo tarkoittaa suurta pakkaussuhdetta eli huonompaa kuvanlaatua. Liikkeenestimoinnissa QP vaikuttaa ainoastaan painoarvona eri

lohkokokojen sekä liikevektoreiden valintaan.

H.264, vaihtoehtoisesti MPEG-4 AVC (engl. Advanced Video Coding) tai MPEG-4 Part 10, on tämän hetken kehittynein yleisesti käytössä oleva videonpakkausstandardi. Sen käyttökohteet vaihtelevat matkapuhelimista televisioon. Standardi saatiin valmiiksi maaliskuussa 2003 [11], mutta sitä on päivitetty jatkuvasti. Tämän työn tekemisen aikana viimeksi maaliskuussa 2010. Tämä pakkaus on käytössä myös HDTV-lähetyksissä (engl. High-definition television), joiden laajamittainen käyttö on alkamassa Suomessakin [15].

H.264-pakkauksen vaatima prosessointiteho on moninkertainen verrattuna esim. DVD-videossa (engl. Digital Versatile Disc) sekä SDTV-lähetyksissä (engl. Standard-Definition Television) käytettyyn MPEG-2 -pakkaukseen. Samalla voidaan kuitenkin säästää jopa 50% vaaditusta siirtokaistasta verrattuna aiempiin pakkauksiin säilyttäen visuaalisen kuvanlaadun [11]. Perusperiaatteiltaan H.264 on hyvin samanlainen aiempien pakkausmenetelmien kanssa, mutta sisältää paljon uusia pakkausta tehostavia menetelmiä. Näitä menetelmiä ovat mm. neljänneskuvapisteen tarkkuus sekä vaihtelevankokoiset lohkot (engl. Variable Block-Size Motion Estimation, VBSME). Kuva 2.1 esittää periaatekuvan pakkaajasta, jonka sisäänmenona on 16×16 pikselin kokoisia makrolohkoja ja ulostulona pakattua videodataa. Kaikkia periaatekuvan vaiheita ei suoriteta jokaiselle kuvalle ja osa vaiheista on vapaaehtoisia.

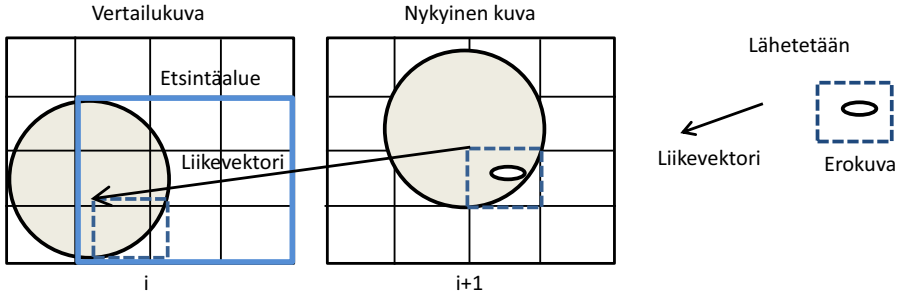


Kuva 2.1: H.264 kooderin periaatekuva, jossa videodata menee sisään vasemmalta 16×16 pikselin kokoisina makrolohkoina ja tulee ulos oikealta pakattuna bittivirtana. Tässä työssä keskitytään liikkeenestimointiin.

3. LIIKKEENESTIMOINTI

3.1 Yleisesti liikkeenestimoinnista

Tämä pakkauksessa käytetty menetelmä perustuu peräkkäisten kuvien samankaltaisuuteen. Liikkeenestimoinnin (engl. Motion Estimation, ME) tehtävänä on etsiä vertailukuvasta lohkoa, joka vastaa parhaiten käsiteltävänä olevaa nykyisen kuvan lohkoa. Kuvassa 3.1 esitetyssä esimerkissä pallo liikkuu hieman oikealle ja siihen muodostuu samalla kuvio, jota siinä ei aiemmin ollut. Kuvassa liikkeenestimointi on löytänyt lähes vastaavan lohkon vertailukuvasta. Vertailukuvan lohkon paikka ilmoitetaan liikevektorin avulla. Liikevektorin lisäksi pitää pakata lohkojen välinen ero, jotta lohko saataisiin myöhemmin purettua samanlaiseksi.



Kuva 3.1: Esimerkki lohkojen valitsemisesta. Vertailukuvaan on rajattu ns. etsintäalue, josta vastaavuuksia etsitään. Liikevektori (engl. Motion Vector, MV) kuvaa liikkeen suuntaa sekä pituutta. Lisäksi kuvassa on esitetty bittivirrassa lähetettävät tiedot.

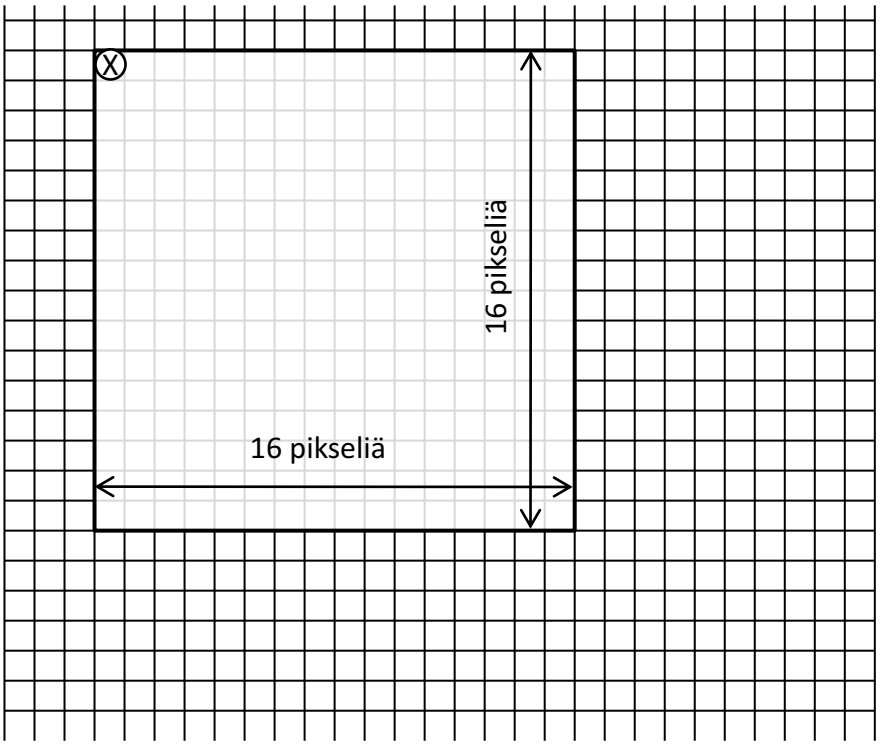
Lohkon valita voidaan tehdä monen eri kriteerin perusteella, mutta SAD-arvon (engl. Sum of Absolute Differences) tarkastelu on yksi yleisesti käytetty tapa. SAD-laskenta on suoraviivaista toteuttaa niin laitteistolla kuin ohjelmistolla, koska siinä ei käytetä kerto- tai jakolaskuja [9]. Kaava 3.1 esittää SAD-arvon laskennan kun nykyinen lohko (C) ja vertailtava lohko (R) ovat $M \times N$ pikselin kokoisia lohkoja, joita indeksoidaan i:llä ja j:llä. Lohkon ollessa 16×16 pikselin kokoinen, vaatii SAD-arvon laskeminen 256 vähennys-, yhteenlasku- sekä itseisarvo-operaatiota.

$$SAD = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} |C_{ij} - R_{ij}| \tag{3.1}$$

Algoritmit käyvät läpi ns. etsintäpisteitä, joiden mukaan vertailtava lohko valitaan vertailukuvasta. Kuvassa 3.2 piste X kuvaa etsintäpistettä ja lohko on aseteltu

sen mukaisesti. Etsintäpiste tarkoittaa siis lohkon vasemman ylänurkan koordinaatteja.

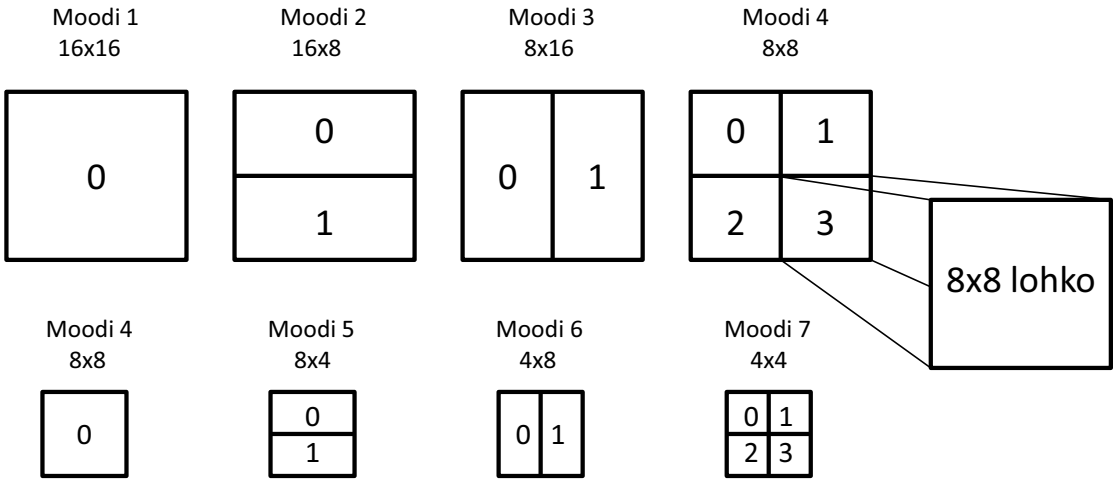
Etsinnän tuloksena löytyneen lohkon ja nykyisen lohkon välistä eroa kutsutaan residuaaliksi eli erokuvaksi. Mitä enemmän eroa löytyy, sitä suuremman datamäärän sen esittäminen vaatii. Jotta saavutetaan alhainen bittivirta, erokuvasta joudutaan hävittämään dataa kvantisoimalla. Tästä syystä on tärkeää löytää mahdollisimman hyvä tai jopa identtinen lohko. Liikkeenestimoinnissa ei pelkästään tutkita pistettä, jossa kyseinen lohko on aiemmin ollut, sillä jostain muualta voi löytyä vielä paremmin nykyistä lohkoa vastaava lohko.



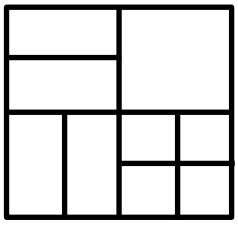
Kuva 3.2: 16×16 pikselin kokoisen lohkon asettelu etsintäpisteeseen, X kuvaa pistettä, jota ollaan tutkimassa.

3.2 Vaihtelevankokoisten lohkojen liikkeenestimointi (VBSME)

VBSME (engl. Variable Block-Size Motion Estimation) on H.264:ssä käytetty liikkeenestimointimenetelmä, jossa etsittävät lohkot jaetaan pienempiin osiin. Jokaiselle eri lohkokoolle on tehtävä erikseen etsintä. Tämän takia operaatio on H.264:ssä huomattavan raskas. Yhteensä erilaisia lohkojakoja löytyy seitsemän, joista moodit 5-7 ovat moodin 4 alimoodeja. Toisin sanoen näitä lohkoja voidaan käyttää vain, kun moodi 4 on valittu päämoodiksi. Tämän jälkeen jokaiseen moodi 4 alilohkoon voidaan valita eri alimoodi. Kuvat 3.3 ja 3.4 selventävät lohkojakoa.



Kuva 3.3: H.264:ssä käytettävät lohkojaot. Ylärivillä oikealla oleva Moodi 4 voidaan jakaa neljään lohkoon, joista jokainen voi olla joku alarivin neljästä moodista. Yhteensä mahdollisia alilohkoja yhdessä makrolohkossa voi täten olla 16.



Kuva 3.4: Esimerkki makrolohkon jakamisesta pienempiin lohkoihin.

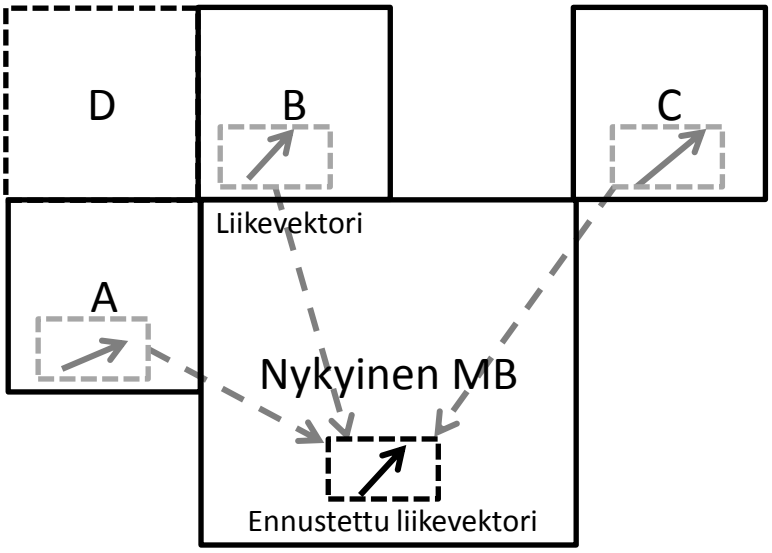
3.3 Vakiokokoisten lohkojen liikkeenestimointi (FBSME)

FBSME (engl. Fixed Block-Size Motion Estimation) on aiempien videonkoodausstandardien käyttämä liikkeenestimointimenetelmä, jossa käytetään pelkästään 16x16 kokoisia lohkoja. FBSME:tä voidaan käyttää myös H.264:n kanssa jättämällä muut kuin moodi 1 käyttämättä. Eri moodien käytöstä saatu hyöty on kuitenkin niin merkittävä, että niitä on järkevä käyttää.

3.4 Ennustetut liikevektorit (PMV)

Koska liikevektorit ovat yleensä tietyllä alueella samankaltaisia, voidaan liikevektoreita ennustaa viereisten lohkojen perusteella. H.264 määrittelee käytettävän ennustusmenetelmän ja ennustus on aina käytössä, jotta dekooderi ja kooderi saavat tulokseksi samat vektorit. Bittivirran mukana annetaan vain erovektori ennustetun ja oikean liikevektorin välillä, jolloin säästetään bittejä.

Liikevektorin ennustaminen nykyiselle lohkolle tehdään kolmen viereisen makrolohkon perusteella ottamalla näistä mediaani. Käytetyt lohkot (A,B ja C) on esitetty kuvassa 3.5 [3]. Jos lohkoa C ei ole saatavilla (kuvan reunassa), käytetään sen tilalla lohkoa D. Koska viereisissä makrolohkoissa voi olla erikokoisia lohkoja, valitaan lohko A vasemmalla olevan makrolohkon oikeasta yläkulmasta, lohkot B ja C yläpuolella olevien makrolohkojen vasemmasta alakulmasta ja lohko D vasemmalla yläkulmassa olevan makrolohkon oikeasta alakulmasta riippumatta siitä, minkä kokoinen lohko eli mikä moodi näissä kohdissa on käytössä.

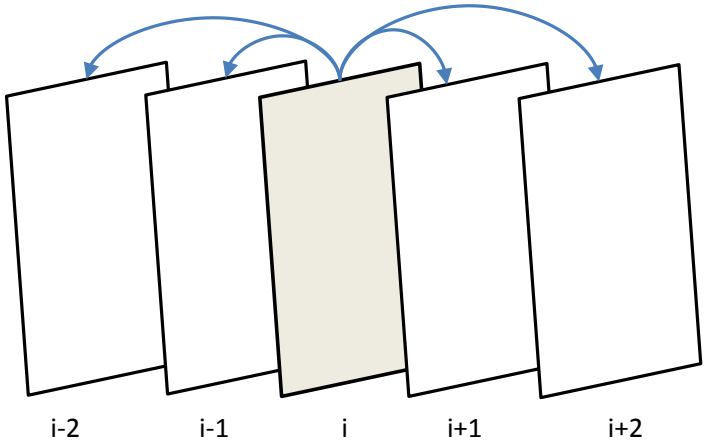


Kuva 3.5: Viereisten lohkojen sijainnit liikevektoreiden ennustuksessa, kun ennustettavana on kokonainen makrolohko.

3.5 Monta vertailukuvaa (MRF)

Monen vertailukuvan (engl. Multiple Reference Frames, MRF) käyttö on yksi uudempien videonkoodausten käyttämä menetelmä. Toisin sanoen yhdelle lohkolle voidaan etsiä vastaavuutta useasta eri kuvasta. Vastaavuuksia voi etsiä myös niin, että lohkon osat valitaan eri kuvista. Tämän ominaisuuden hyödyntäminen on hyvin raskas operaatio, koska sama etsintä joudutaan erikseen tekemään jokaiselle vertailukuvalle. H.264:ssä kuvien maksimimäärä on 16 [12]. Kuvassa 3.6 on esitetty, miten vertailukuvat voidaan valita, kun i merkitsee nykyistä kuvaa.

H.264:ssä on myös mahdollista tallentaa tiettyjä vertailukuvia kauemmin, jolloin kuva varaa yhden vertailukuvan paikan. Nämä kuvat säilyvät vertailukuvina niin pitkään, kunnes poistamisesta erikseen ilmoitetaan. Mahdollisesti yhden hyvän vertailukuvan löytyminen säästää bittejä, jos sitä voidaan käyttää useiden seuraavien kuvien ennustamiseen. Jotta tällainen vertailukuva löytyisi, on video luultavasti käytävä läpi useampaan kertaan. Tätä siis ei voida käyttää reaaliaikaisessa kuvanpakkauksessa täysin hyödyksi.



Kuva 3.6: Esimerkki vertailukuvien valitsemisesta, i on nykyinen kuva.

3.6 Lohkonetsintäalgoritmit (BMA)

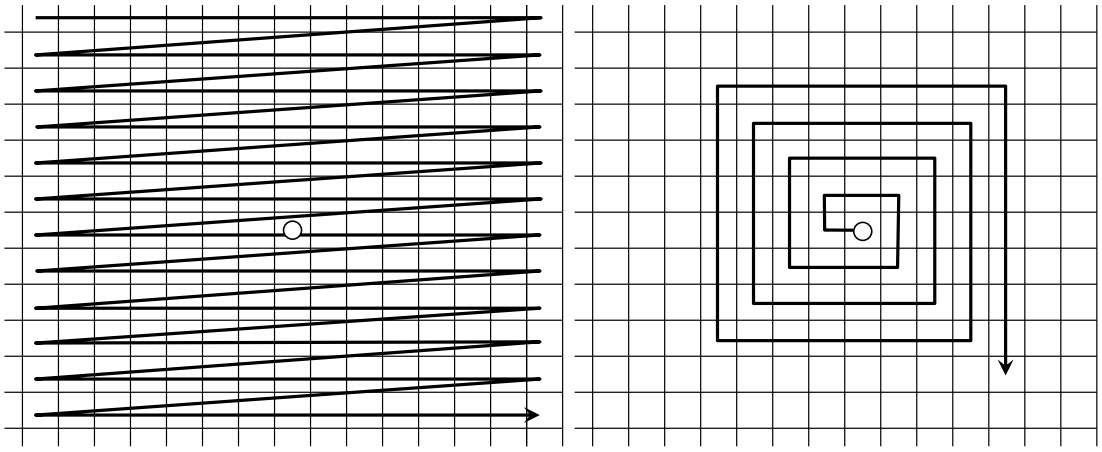
Lohkojen etsimistä varten on kehitetty paljon erilaisia algoritmeja (engl. Block-Matching Algorithms, BMA), joista sopivimman valinta riippuu käyttökohteesta. Eri algoritmit tarkistavat eri määrän pisteitä ja ”kulkevat” eri tavalla kuvassa.

Helpoin tapa etsiä vertailukuvasta täsmääviä lohkoja on käydä koko kuva piste pisteeltä läpi. Kyseinen etsintä tehdään jokaiselle moodille erikseen. FS (engl. Full Search) on tällainen algoritmi. FS toteutetaan usein niin, ettei se tarkista kaikkia pisteitä. Se voi tarkistaa esim. vain yhden vertailukuvan jokaisen pisteen tietyllä etsintäalueella alkupisteen ympärillä. Muiden algoritmien suorituskykyä verrataan usein FS algoritmiin, koska se antaa parhaan tuloksen.

Jos laskentatehoa on käytössä vain rajallisesti tai halutaan nopea reaaliaikainen pakkaus, toteutetaan ns. nopea BMA (Block-Matching Algorithm). Tällainen algoritmi jättää suurimman osan pisteistä tarkastamatta. Näiden algoritmien tuottama kuvanlaatu ei yllä aivan FS algoritmin tasolle, mutta niiden tarkastamat pisteet voivat jäädä jopa sadasosaan FS:n tarkastamista pistemäärästä, jolloin laskenta-aika pienenee samassa suhteessa.

Myös FS voidaan toteuttaa siten, että se käy vain osan pisteistä läpi. Yksi keino on raja-arvon käyttäminen. Tämä voidaan toteuttaa mm. siten, että FS aloittaa pisteiden läpikäymisen keskeltä spiraalimaisesti. Kun löytyy piste, joka on ”tarpeeksi

hyvä", voidaan suoritus keskeyttää. Raja-arvo voidaan ennustaa edellisistä makrolohkoista saaduista tuloksista. Kuva 3.7 esittää esimerkkitoteutuksia näistä.



Kuva 3.7: Esimerkkitoteutuksia Full Search -algoritmista. Vasemmanpuoleinen on ns. perinteinen toteutus, jossa käydään kaikki pisteet läpi yksitellen ylävasemmalta alkaen. Oikealla on toteutettu spiraalinen etsintä, joka voidaan lopettaa, kun löydetään riittävän hyvä piste. Ympyrä kuvaa pistettä, jossa etsittävä lohko sijaitsee.

3.7 Nopeutustekniikat

Algoritmien suoritusta voidaan nopeuttaa tekemällä niihin muutoksia, jopa niin, että laatu pysyy samana, mutta prosessointiaika pienenee. Full Search on VBSME:n yhteydessä helposti muokattavissa sellaiseksi, ettei jokaiselle moodille tarvitse laskea pisteitä uudelleen, vaan voidaan uudelleen käyttää pienimmälle moodille laskettuja SAD-arvoja. Näitä 4×4 lohkojen SAD-arvoja voidaan yhdistää, jotta saadaan isompien lohkojen arvot, eikä jokaista moodia tarvitse käydä erikseen läpi. Tämä toteutus voi yli nelinkertaistaa laskentanopeuden pitäen laadun samana. Tällä tekniikalla on tehty mm. JM:n referenssiohjelmistoon Fast Full Search -algoritmi, jota siinä käytetään vakiona.

Samanlaista toteutusta ei voida suoraan tehdä nopeille algoritmeille, koska jokaisella eri moodilla "polku" voi olla erilainen. Jos tämä kuitenkin toteutetaan, on seurattava moodin 1 reittiä, josta valitaan kaikkien moodien parhaat arvot. Tällöin laskettujen pisteiden määrä putoaa lähes kuudesosaan ja laskenta-aika jopa puolittuu. Tämä johtuu siitä ettei, etsintää tarvitse tehdä jokaiselle moodille. Tuloksien yhdistely vaatii kuitenkin hieman laskentaa. Tällä menetelmällä ei päästä aivan samaan laatuun kuin päästäisiin ajamalla algoritmi erikseen jokaisella moodilla, mutta nopeutus verrattuna laadun laskemiseen on suuri.

3.8 Yhteenveto

Taulukko 3.1 esittää käsiteltyjen tekniikoiden hyvät ja huonot puolet. PMV on tekniikoista ainoa pakollinen purettavissa olevan videodatan aikaansaamiseksi. Kuitenkin näitä kaikkia käytetään (tai voidaan käyttää) useimmissa kooderitoteutuksissa niiden antaman hyödyn takia [12].

Taulukko 3.1: Yhteenveto liikkeenestimointia parantavista tekniikoista

	Hyöty	Haitta	Idea
VBSME	Parantaa kuvanlaatua	Vaatii lisäbittejä	Monta lohkokokoa
PMV	Säästää bittejä	Ennustus voi olla huono	Ennustus
MRF	Parantaa kuvanlaatua	Vaatii laskentatehoa	Monta vertailukuvaa
BMA	Nopeuttaa suoritusta	Huonontaa kuvanlaatua	Nopea lohkojen etsintä

4. AIEMMAT OHJELMISTOTOTOTEUTUKSET

4.1 x264

Tämä avoimen lähdekoodin H.264 pakkaajatoteutus on kuvanlaadultaan sekä nopeudeltaan parempi kuin useimmat kaupallisista ohjelmistoista ja onkin hyvin laajalti käytössä [16]. Esimerkiksi Google käyttää tätä ohjelmistoa omassa videopalvelussaan (Google Video) sekä Youtubessa [10][8]. Tämä kooderi saattaa olla osittain laiton, koska kehittäjät eivät ole maksaneet H.264 lisenssejä hallinnoivalle MPEG LA:lle [8].

4.2 JM Reference Software

H.264 standardin yhteydessä kehitetty testiohjelmisto toteuttaa kaikki H.264:n ominaisuudet. Ohjelmisto sisältää dekooderin, kooderin ja muita testityökaluja. Ohjelmiston suorituskyky on alhainen, koska se on tarkoitettu lähinnä testikäyttöön. Tätä ohjelmistoa ei myöskään voi käyttää koodekkina ilman muutoksia. Tutkimuskäytössä ei yleensä muita koodereita käytetä, koska halutaan pysyä standardia noudattavassa sekä yleisesti tunnustetussa toteutuksessa. Mikään tunnettu kooderi ei toteuta vielä niin hyvin standardissa määriteltyjä ominaisuuksia kuin JM:n ohjelmisto. Kovin moni ohjelmisto ei pääse edes lähelle näiden toteuttamisessa [12].

4.3 Yhteenveto

Taulukko 4.1: Yhteenveto aiempien ohjelmistotototeutuksien ominaisuuksista

Kooderi	Nopeus	Laatu	Kieli	Täysi standardi
x264	++++	+++	C/asm	Ei
JM Ref Encoder	-	++++	C	Kyllä

Taulukossa 4.1 on esitetty näiden kahden kooderin ominaisuuksia yleisellä tasolla. Kooderitoteutuksia on olemassa useita, mutta nämä kaksi ovat avoimesti saatavilla [13]. Näiden kooderien vertaaminen tässä työssä toteutettuun ohjelmistoon ei ole mahdollista, koska tässä työssä ainoastaan liikkeenestimointi on toteutettu. Näistä

kahdesta x264 on ehdottomasti kotikäytössä ja muussakin ei-tutkimuskäytössä järkevin toteutus. Tässä työssä verrataan tuloksia JM kooderitoteutukseen, koska kummankin toteutuksen lähdekoodit ovat saatavilla ja muokattavissa, niihin on mahdollista toteuttaa samankaltaisia testijärjestelyitä. Vaikka x264 ei toteuta kaikkia H.264 ominaisuuksia, on se hyvä esimerkki todellisesta käytettävästä kooderitoteutuksesta ja siksi sitä on verrattu yleisellä tasolla JM:n toteutukseen.

5. TOTEUTETTU OHJELMISTO

5.1 Yleistä toteutetusta ohjelmistosta

Ohjelmiston tukemat etsintäalgoritmit hyödyntävät etsinnässä erikokoisia kuvioita, jotka etenevät algoritmin määräämällä tavalla parhaimpien arvojen suuntaan. Suoritus keskeytetään siinä vaiheessa, kun parempia pisteitä ei enää algoritmin avulla löydy. Tämän jälkeen voidaan vielä tarkastaa viereiset pisteet, jos niistä löytyisi parempi. Etsintäpisteille lasketaan SAD kaavalla 3.1 ja parhaimmaksi valitaan lohko, jonka virhe on pienin verrattuna alkuperäiseen lohkoon. Etsintäalgoritmit on toteutettu täsmälleen samalla tavalla kuin laitteistotototeutuksessa, jotta toteutuksien arvoja voidaan suoraan verrata ja havaita eroavuudet. Kaikki algoritmit Unsymmetrical-cross Multi-Hexagon-grid Search -algoritmia lukuun ottamassa oli toteutettu vanhaan ohjelmistoon. Algoritmit vaativat muokkauksia toimiakseen uudessa järjestelmässä. Algoritmien etsintäpolkua rajoittaa etsintäalueen koko (esim. 48×48) sekä mahdollinen etsintäpisteiden määrää rajoittava arvo (esim. 256 etsintäpisteen raja).

5.2 Tuetut lohkonetsintäalgoritmit

Tässä työssä on toteutettu 7 algoritmia, joista UMHexagonS on toteutettu aivan alusta alkaen.

5.2.1 Full Search (FS)

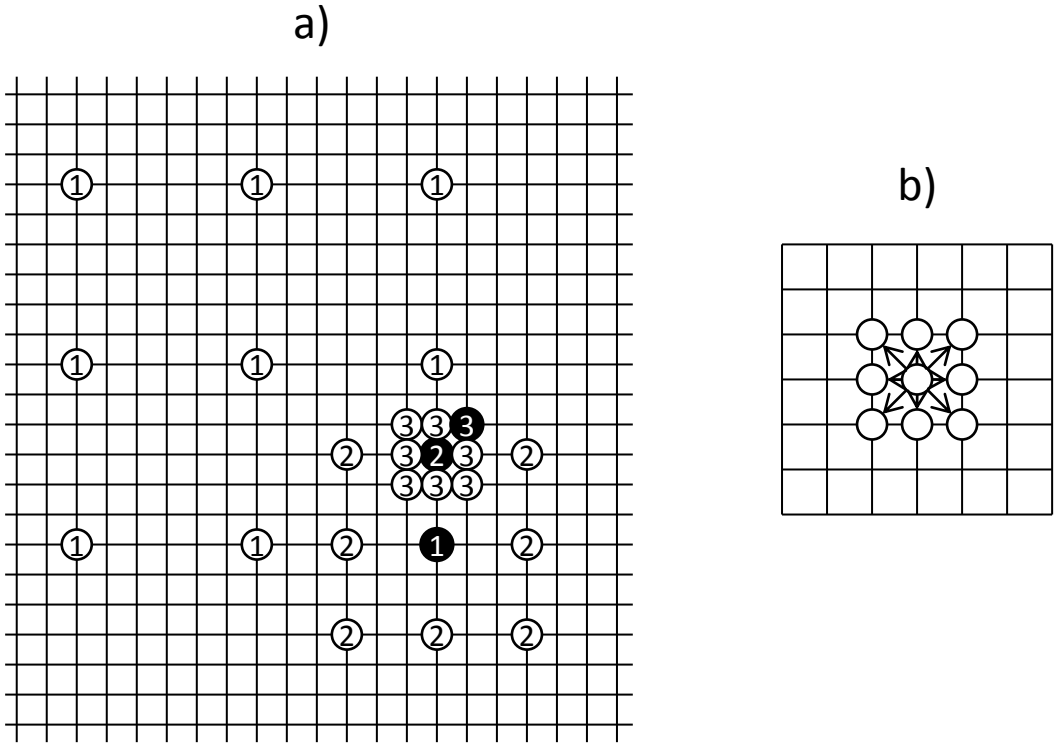
Full Search käy nimensä mukaisesti läpi jokaisen pisteen etsintäalueelta. Harva ohjelmisto toteuttaa FS-algoritmia, koska se on hidas muihin algoritmeihin verrattuna. FS laitteistotototeutuksia on olemassa ja niitä käytetään, kun halutaan hyvää kuvanlaatua eikä tehonkulutuksella tai prosessointiajalla ole niin suurta merkitystä. Laitteistolla tämä algoritmi olisi erittäin helposti toteutettavissa verrattuna muihin algoritmeihin. Testattavana oleva laitteisto ei tue FS-algoritmia, koska se pyrkii nopeaan suoritukseen.

5.2.2 Three-Step Search (TSS)

Vaikka TSS algoritmin [5] nimessä puhutaan kolmesta askeleesta voi etsintäaskelia olla enemmän. Askeleiden määrä on kiinni etsintäalueen koosta, 48×48 alueella riittää kolme askelta. Kuvassa 5.1 on kuvattu tällainen tilanne. Algoritmin käyttämän kuvion koko puolittuu jokaisella kierroksella. Suoritus jatkuu, kunnes kuvio on niin pieni, että se käy läpi vierekkäisiä kuvapisteitä.

Algoritmin suoritus aloitetaan skaalaamalla peruskuvio (Kuva 5.1b) niin isoksi, että se sijoittuu noin puoleen väliin etsintäaluetta. Skaalausarvon on hyvä olla kahden potenssi, jotta sen puolittaminen sujuu ilman pyöristämistä. Tämän jälkeen tarkastetaan kuvion jokainen piste ja valitaan SAD-arvoltaan paras. Sen jälkeen siirretään valittu piste keskelle, puolitetaan kuvion koko ja jatketaan samalla tavalla, kunnes kuvion pisteet ovat kiinni toisissaan.

Ison alkukuvion takia TSS voi helposti jättää parhaat pisteet huomioimatta. Tästä huolimatta TSS suoriutuu erittäin hyvin ja on edelleen yksi parhaista algoritmeista. Varsinkin suuren resoluution videot hyötyvät, kun etsintä ei kohdistu keskelle useimpien muiden nopeiden algoritmien tapaan.

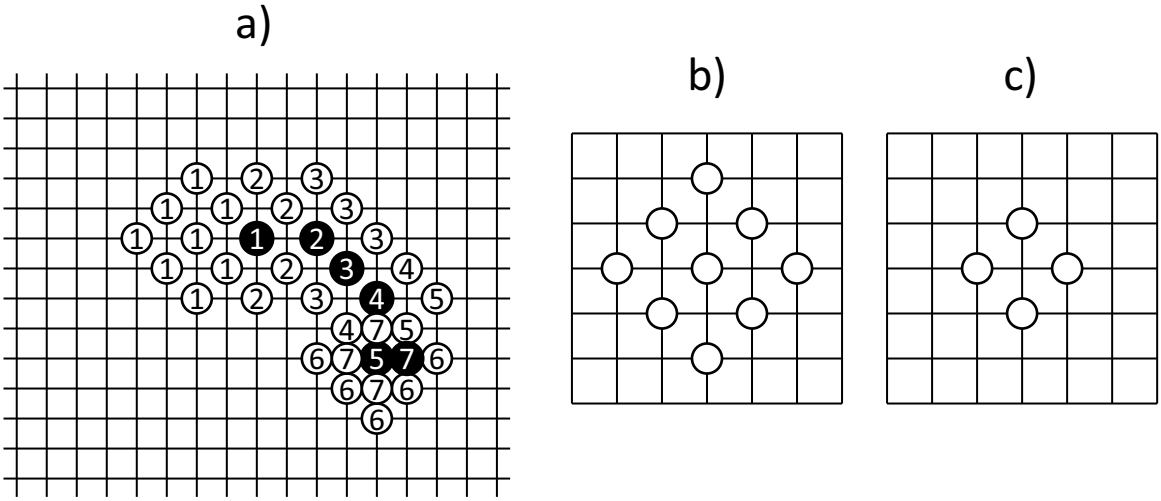


Kuva 5.1: TSS:n esimerkkireitti ja peruskuvio. Keskellä on koordinaattipiste (0,0). Kuviota laajennetaan kertomalla peruskuvion (b) pisteitä aluksi etsintäalueesta riippuvalla skaalausarvolla, valitaan paras piste ja siirretään kuvio tähän pisteeseen. Pisteiden kerroin puolitetaan joka kierroksella kunnes pisteet ovat vierekkäin. Tässä esimerkkitilanteessa etsintäalueen koko on 48×48 kuvapistettä.

5.2.3 New Diamond Search (NDS)

NDS:n [18] peruskuvio on nimensä mukaan hieman timantin muotoinen. Tämä algoritmi on yksi perusalgoritmeista ja moni algoritmi perustuu tähän. Kuvassa 5.2 on esitetty NDS:n toimintaperiaate. Algoritmi toimii seuraavasti:

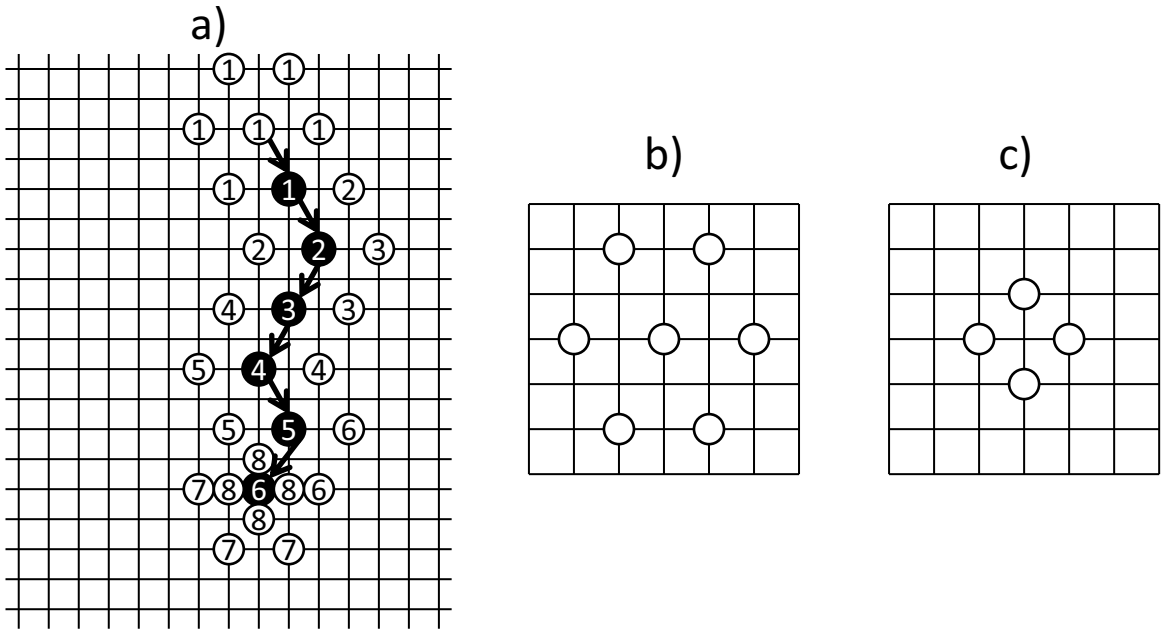
1. Aloitetaan etsintä laittamalla aloituspiste keskipisteeksi.
2. Tarkistetaan keskipiste ja sen ympärillä olevat pisteet timanttikuvion mukaisesti. Jos paras vaihtoehto ei ole keskellä, siirretään paras vaihtoehto keskipisteeksi ja jatketaan etsintää suurella timanttikuviolla (Kuva 5.2b). Jos paras vaihtoehto on keskipiste, siirrytään kolmanteen askeleeseen.
3. Käytetään pientä timanttikuviota (Kuva 5.2c) ja pisteiden läpikäynnin jälkeen valitaan paras piste.



Kuva 5.2: NDS:n esimerkkireitti, jossa parhaat pisteet jokaiselle "kierrokselle"on merkitty mustalla. Numerot kuvaavat kuvion käyttökertojen määrän kussakin kohdassa. Etsintä alkaa isolla kuviolla (b) jota siirretään aina uuteen parhaaseen pisteeseen, kunnes se löytyy keskeltä ja käytetään pientä kuviota (c).

5.2.4 Hexagon-based Search (HEXBS)

Toimii kuten DS, mutta käyttää isoa heksagonikuviota timanttikuvion sijaan. Heksagonikuvio mahdollistaa nopeamman etenemisen etsintäalueella. Haittapuolena voi olla parhaiden pisteiden jääminen reitin ulkopuolelle. HEXBS on nopeampi kuin NDS ja tuottaa lähes samantasoista laatua. Kuva 5.3 esittää HEXBS:n esimerkkireitin ja käytetyt kuviotyypit. [17]



Kuva 5.3: Hexbs:n esimerkkireitti, jossa parhaat pisteet jokaiselle "kierrokselle"on merkitty mustalla. Numerot kuvaavat kuvion käyttökertojen määrän kussakin kohdassa. Etsintä alkaa isolla kuviolla (b), jota siirretään aina uuteen parhaaseen pisteeseen, kunnes se löytyy keskeltä ja käytetään pientä kuviota (c).

5.2.5 Cross-Diamond Search (CDS)

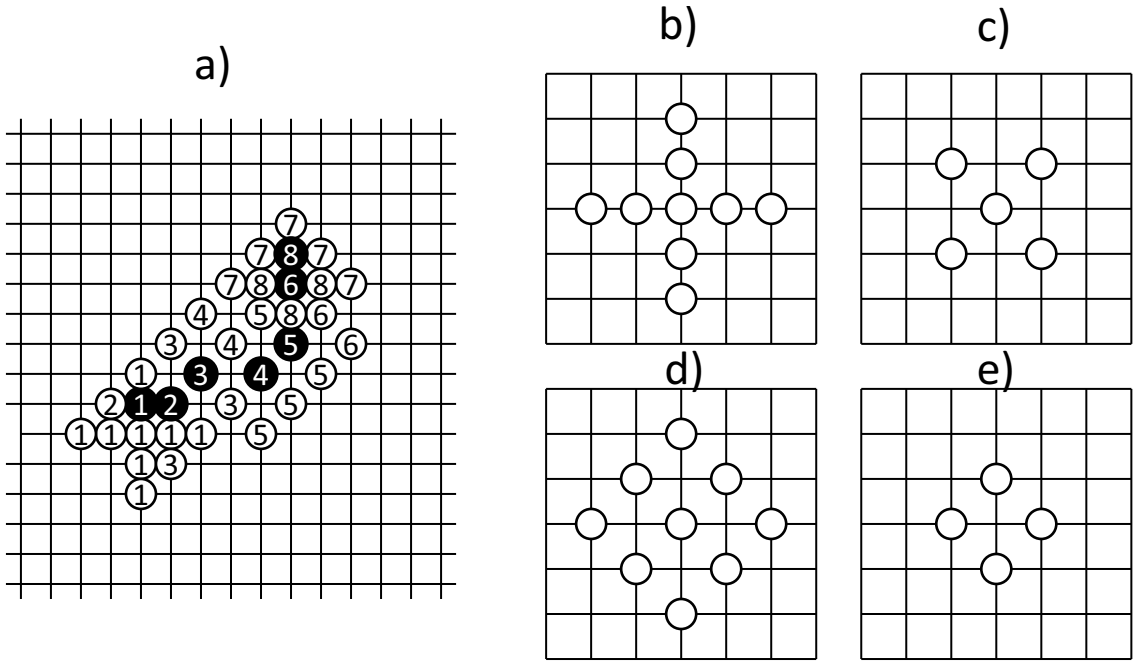
CDS voidaan ajatella NDS:n laajennettuna versiona, joka käyttää hyväkseen useampaa kuviota edetessään. Kuvassa 5.4 on esitetty esimerkkireitti sekä käytetyt kuviot. Etsiminen aloitetaan ristikuviolla (Kuva 5.4b), josta jatketaan käyttäen osittaista timanttikuviota. Timanttikuvio keskitetään ristikuvion parhaan pisteen suhteen (Kuva 5.4c). Tämän jälkeen jatketaan etsimistä kuten NDS:ssä käyttämällä isoa timanttikuviota ja kun paras piste löytyy ison timanttikuvion keskeltä, käytetään pientä timanttikuviota. [2]

5.2.6 Unsymmetrical-cross Multi-Hexagon-grid Search (UMHexagonS)

UMHexagonS algoritmia käytetään vakiona mm. x264 kooderissa, joka on paras open-source toteutus H.264:sta [16]. Tämä algoritmi tuottaa lähes Full Search -algoritmin tasoista laatua pitäen etsintäpisteiden määrän kuitenkin suhteellisen pienenä. Testattava laitteistolohko ei tue tätä algoritmia, mutta se on implementoitu testausjärjestelmään koeajoja varten. Toteutettu versio ei ole sama mitä muut järjestelmät käyttävät, koska toteutuksesta puuttuu aloituspisteen ennustaminen.

Algoritmi etenee seuraavalla tavalla:

1. Iso ristikuvio, joka liikkuu etsintäelueen reunasta reunaan sivusuunnassa ja



Kuva 5.4: CDS esimerkkireitti (a), iso ristikuvio (b), osittainen timanttikuvio (c), iso timanttikuvio (d) ja pieni timanttikuvio (e).

puoleenväliin pystysuunnassa

2. Keskitys ristikuvion parhaaseen pisteeseen ja 5×5 kokoinen Full Search
3. Keskitys parhaaseen ja jatketaan koko etsintäalueen täyttävällä laajalla heksakuviolla
4. Keskitys parhaaseen ja jatkuu kuten HEXBS

5.2.7 Block-Based Gradient Descent Search (BBGDS)

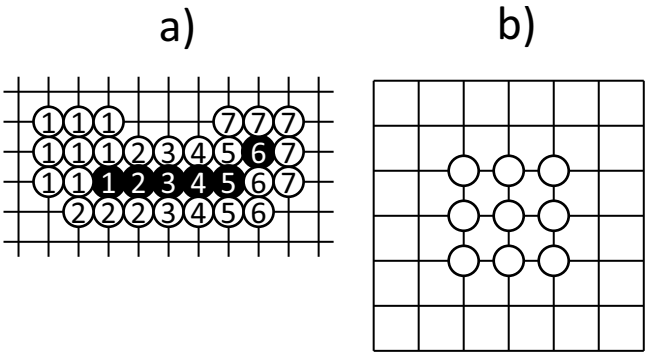
BBGDS algoritmi tekee 3×3 pikselin kokoisia FS-algoritmin tapaisia etsintöjä ja etenee kuten muut nopeat algoritmit. Käytännössä BBGDS soveltuu vain tilanteisiin, joissa liike on pientä. Kuvassa 5.5a on esitetty BBGDS:n esimerkkireitti. Huomattavaa on, että eteneminen on hitaampaa, kuin muissa algoritmeissa.

5.2.8 Yhteenveto lohkonetsintäalgoritmeista

Taulukossa 5.1 on esitetty yhteenveto ohjelmistotototeutuksen algoritmeista. Taulukko on järjetetty siten, että laadultaan paras algoritmi on ensin.

5.3 Monta vertailukuvaa

Toteutettu ohjelmisto tukee maksimissaan viittä peräkkäistä vertailukuvaa. Valitulle määrälle vertailukuvia tehdään identtinen etsintä ja löydetyistä lohkoista valitaan



Kuva 5.5: BBGDS esimerkkireitti sekä kuvio.

Taulukko 5.1: Yhteenveto lohkonetsintäalgoritmeista kun etsintäalue on 16×16 kuvapistettä

Algoritmi	Peruskuvio	Peruskuvion pisteitä	Laatu	Eri kuvioita	Koodirivejä
FS	(kaikki)	256	+++++	1	136
UmHexagonS	Heksa	5-25	++++	5	357
TSS	Neliö	9	+++	1	158
NDS	Timantti	9	++	2	251
CDS	Timantti	9	+	4	360
HEXBS	Heksagoni	7	+	2	240
BBGDS	Neliö	9	+	1	185

parhaat. Koska "kauempana"olevien vertailukuvien valitseminen lisää bittivirtaa, käytetään kuvien painokertoimina arvoja {1,3,3,5,5}. Arvot tulevat etumerkittömän exponenttisen Golomb-koodauksen vaatimasta bittimäärästä luvuille {0,1,2,3,4}. Eksponenttinen Golomb koodi muodostetaan datan esittämiseen vaadittavan bittimäärän perusteella. Jotta koodi voitaisiin purkaa, täytyy sen pituus tietää. Tämä tehdään lisäämällä koodin alkuun datan bittimäärän verran nollia sekä yksi ykkönen. Jos data on esimerkiksi bitteinä "101", muodostuu siitä Golomb koodina "0001101". Näin lisätään realistisempaa kustannusta, jos valitaan lohko kauempaa kuin edellisestä vertailukuvasta. [14]

Sekvenssin alussa (käsiteltäessa toista kuvaa) vertailukuvia on alussa käytössä vain yksi (ensimmäinen kuva). Määrä lisääntyy yhdellä kun siirrytään seuraavaan kuvaan, kunnes saavutetaan oikea vertailukuvien määrä.

6. SUORITUSKYKYANALYYSI

6.1 Mittausjärjestely

Mittauksissa käytettiin apuna itse koottua testisekvenssikirjastoa. Kirjastossa on kymmeniä erilaisia videosekvenssejä jaoteltuna niiden resoluution mukaan. Pienimmät videot ovat QCIF (engl. Quarter Common Image Format, 172x144 kuvapistettä) tasoisia ja suurimmat 1080p (1920×1080 kuvapistettä). 1080p videoihin on lisätty kahdeksan ylimääräistä riviä alareunaan, jotta niissä olisi pystytasossa kokonaismäärä makrolohkoja. Kaikki sekvenssit ovat YUV muodossa. Kirjastosta valittiin kolme erilaista sekvenssiä tässä työssä käytettäväksi.

”F1 Car” (D1, 220 kuvaa) on hyvin vaativa ja sisältää nopeaa liikettä. Siksi se antaa paljon tietoa algoritmien toimimisesta ääritilanteissa. ”Pedestrian” (1080p, 50 kuvaa) on Full HD -resoluutioinen sekvenssi joka sisältää paljon tasaista liikettä. ”Foreman” (CIF, 300 kuvaa) on hieman pienempi sekvenssi, jossa on vähemmän liikettä. Taulukossa 6.1 on esitetty videoiden pakkaamisessa on käytetyt muuttujat.

Taulukko 6.1: Eri muuttujat mittauksia tehdessä

Muuttuja	Vaihtoehtoja	Pienin arvo	Suurin arvo
Algoritmi	7	-	-
Etsintäalue	4	48×48	112×112
Moodit	3	Moodi 1	Moodit 1-7
QP	3	20	36
Sekvenssi	3	Foreman (CIF)	Pedestrian (1080p)
Vertailukuvat	5	1	5
Yhteensä	3780	-	-

Testit on suoritettu H.264 ME ohjelmistolla, joka simuloi pelkästään liikkeenestimointia. Järjestelmän sisääntulona on YUV-muotoinen videotiedosto ja muita parametreja kuten kvantisointiparametri (QP), etsintäalue, maksimi kuvamäärä ja etsintäalgoritmi. Liikkeenestimointi voidaan aloittaa toisesta kuvasta, jolloin ensimmäinen toimii vertailukuvana. Kuvan käsittelyn jälkeen kuva kootaan uudelleen saatujen liikevektoreiden mukaan ja lasketaan PSNR (engl. Peak Signal-to-Noise Ratio) alkuperäiseen kuvaan verrattuna kaavan 6.2 mukaisesti. Tätä varten tarvittava MSE (engl. Mean Squared Error) lasketaan kaavalla 6.1 m×n kokoiselle lohkolle ja MAX_I on maksimi arvo, jonka kuvapiste voi saada. Kun halutaan saada koko

sekvenssin PSNR, lasketaan keskiarvo jokaisen kuvan PSNR:stä kaavalla 6.3. Tämä laskentatapa on yleisesti käytetty, vaikka se antaa vääristyneitä tuloksia arvojen logaritmisuudesta johtuen. Tässä työssä käytetyt PSNR arvot on laskettu aina luminanssieli kirkkausarvoille.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} |C_{ij} - R_{ij}|^2 \tag{6.1}$$

$$PSNR = 20 * log_{10}(\frac{MAX_I}{\sqrt{MSE}}) \tag{6.2}$$

$$PSNR_{avg} = \frac{1}{kuvat} \sum_{f=0}^{kuvat-1} PSNR(f) \tag{6.3}$$

Tämän lisäksi tallennetaan keskimääräinen etsintäpisteiden määrä sekä tilastotietoa. Nykyinen (alkuperäinen, tiedostosta luettu) kuva siirretään uudeksi vertailukuvaksi, edellinen vertailukuva toiseksi vertailukuvaksi (jos monta vertailukuvaa käytössä) ja luetaan uusi kuva tiedostosta. Näin jatketaan, kunnes tiedosto on käyty läpi tai saavutetaan parametriksi annettu maksimi kuvamäärä. Tallennetuista arvoista lasketaan keskiarvo, joka tulostetaan ruudulle.

Tilanne ei vastaa aivan oikean H.264-kooderin liikkeenestimointia, koska tästä testijärjestelmästä puuttuu $\frac{1}{4}$ kuvapisteen tarkkuudella toimiva estimointi sekä mahdollinen kuvan suodatus H.264 standardin mukaisella suodatuksella. Kuviin ei myöskään lisätä residuaalia eli erokuvaa, joka parantaisi kuvanlaatua merkittävästi ja korjaisi huonosta liikevektorista johtuvaa kuvan huonontumista. Tämä puute on kuitenkin korjattu käyttämällä vertailukuvana aina alkuperäistä kuvaa, eikä edellistä ennustettua kuvaa, kuten oikeasti tehtäisiin. Testijärjestelmän avulla ei ole myöskään mahdollista laskea oikeaa bittivirran määrää. Testijärjestelmä käyttää vain edellisiä kuvia vertailukuvina, vaikka H.264-standardi sallisi myös tulevien kuvien käytön vertailussa.

Huolimatta eroavaisuuksista oikeaan H.264 järjestelmään, voidaan tällä testijärjestelmällä verrata eri algoritmien vaikutusta kuvanlaatuun ehkä jopa paremmin, koska nyt on jätetty pois kvantisoinnin tuoma vaikutus. Mahdollisena haittapuolena on, ettei tuloksia voi verrata suoraan muiden tekemiin tutkimuksiin. Testaukset täydellisellä kooderilla (joka lisää erokuvat sekä käyttää intra-koodattuja kuvia) osoittivat, että PSNR arvot menevät lähemmäksi toisiaan eli algoritmien laatuero tasoittuu.

Kokonaisuudessaan tehdyt testit ovat hyvin kattavia ja sisältävät paljon erilaisia tilanteita. Laatua arvioidaan laskennallisesti PSNR:llä, joka ei kerro, miltä kuva oikeasti näyttää. On myös vaikea sanoa, millainen ero PSNR:ssä on merkittävä, koska isoilla PSNR:n arvoilla pienet erot kuvassa vaikuttavat enemmän. Tyypillisesti yli

33dB PSNR tarkoittaa hyvää tai siedettävää kuvanlaatua ja 0.5dB ero on nähtävissä kuvassa [7].

6.2 Tuloksien vertailu

6.2.1 Eri algoritmien välinen vertailu

Verrattaessa algoritmeja otetaan huomioon yleensä kaksi tekijää: etsintäpisteiden määrä ja PSNR arvo. Näiden arvojen avulla voidaan nähdä, kuinka tehokkaita algoritmit ovat. Etsintäpisteiden määrän perusteella voidaan päätyä käyttämään nopeampaa algoritmia, jonka tuottama laatu on hieman heikompi.

Taulukoissa 6.2 ja 6.3 on esitetty eri sekvensseillä ja etsintäalueilla ajettu testi (QP-arvona 28). Taulukko on järjestetty PSNR arvon perusteella. Tässä testissä ovat moodit 1–7 käytössä niin, että moodit 5–7 käydään läpi vain, jos moodi 4 valitaan ”päämoodeista” parhaaksi. FS on tässä testissä ns. nopea FS, joka ei käy pisteitä moneen kertaan läpi. Tämä alentaa pisteitä/MB seitsemäsosaan alkuperäisestä säilyttäen saman laadun.

Taulukko 6.2: Algoritmien välinen vertailu, jossa sekvenssinä F1 Car (PAL), QP-arvona 28 sekä etsintäalueena 80×80

	BBGDS	**HEXBS**	**CDS**	**NDS**	**TSS**	**UMHex**	**FS**
PSNR (dB)	23.79	24.29	24.37	24.42	25.42	26.38	28.52
Pisteet/MB	91.92	78.54	118.57	108.54	204.78	492.93	3481.00

Taulukko 6.3: Algoritmien välinen vertailu, jossa sekvenssinä Foreman (CIF), QP-arvona 28 sekä etsintäalueena 48×48

	BBGDS	**HEXBS**	**CDS**	**NDS**	**TSS**	**UMHex**	**FS**
PSNR (dB)	32.79	32.29	32.78	32.86	32.96	33.42	34.26
Pisteet/MB	68.61	58.60	90.06	77.24	137.32	294.10	729.00

6.2.2 Eri lohkokokojen vaikutukset

Vaikka H.264 käyttääkin kaikkia seitsemää eri moodia, voidaan algoritmeja optimoida käyttämään vain osaa moodeista, jolloin muut moodit jäävät vain käyttämättä. Moodivaihtoehdot jaetaan usein kolmeen osaan: Käytetään moodia 1, moodeja 1–4 tai moodeja 1–7. Taulukoista 6.4 ja 6.5 nähdään, kuinka kaikkien moodien käyttäminen pelkän yhden moodin sijaan lisää kuvan laatua tässä tapauksessa yli kahdella desibelillä riippumatta algoritmista. FS saavuttaa parhaat PSNR arvot, joista muut

algoritmit selvästi jäävät. Huomattavaa on kuitenkin, että myös FS:n tulos paranee yli kolme desibeliä kasvatettaessa käytettyjen moodien määrää.

Johtuen FS:n nopeasta toteutustavasta, eri moodien läpikäyminen ei lisää siinä läpikäytyjen pisteiden määrää. Muuta prosessointia joudutaan tekemään kuitenkin hieman enemmän. Tästä toteutustavasta käytetään nimeä SAD Reuse. [1]

Taulukko 6.4: Esimerkki lohkokokojen vaikutuksesta F1 Car-sekvenssissä, kun etsintäalue on 80×80 ja QP-arvo 28

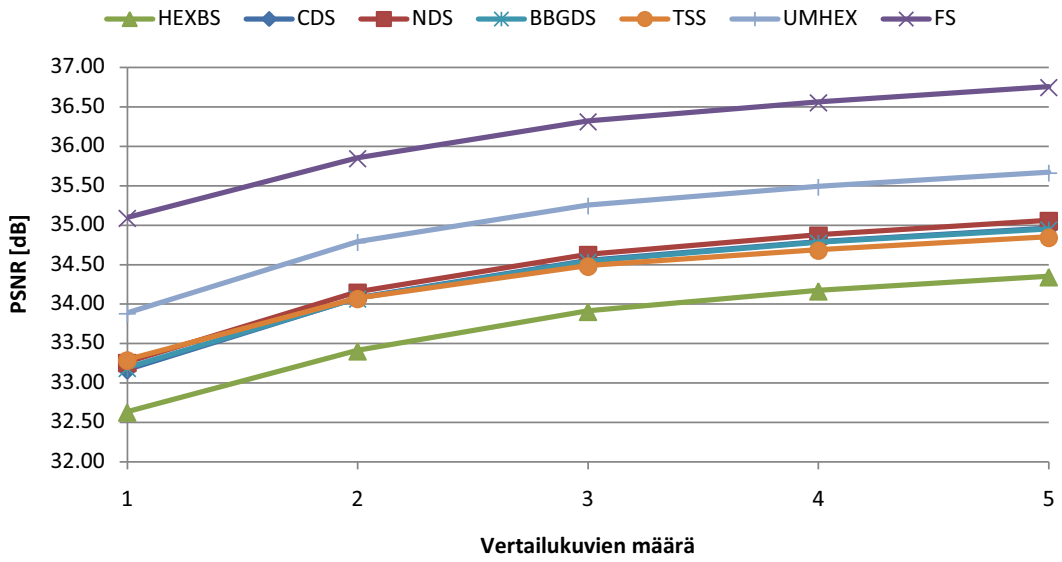
	Moodi 1		Moodit 1-4		Moodit 1-7	
Algoritmi	PSNR [dB]	Pisteet/MB	PSNR [dB]	Pisteet/MB	PSNR [dB]	Pisteet/MB
HEXBS	22.10	16.44	23.51	63.24	24.29	78.54
BBGDS	22.78	20.14	23.08	75.37	23.89	91.92
NDS	22.24	23.27	23.63	88.03	24.42	108.54
CDS	22.21	24.69	23.59	95.19	24.37	118.57
TSS	22.65	40.84	24.50	163.41	25.42	204.78
UMHex	23.64	94.56	25.28	377.99	26.38	492.93
FS	25.23	3481.00	27.00	3481.00	28.52	3481.00
AVG	22.98	528.71	24.37	620.60	25.33	653.75

Taulukko 6.5: Esimerkki lohkokokojen vaikutuksesta Pedestrian-sekvenssissä, kun etsintäalue on 80×80 ja QP-arvo 28

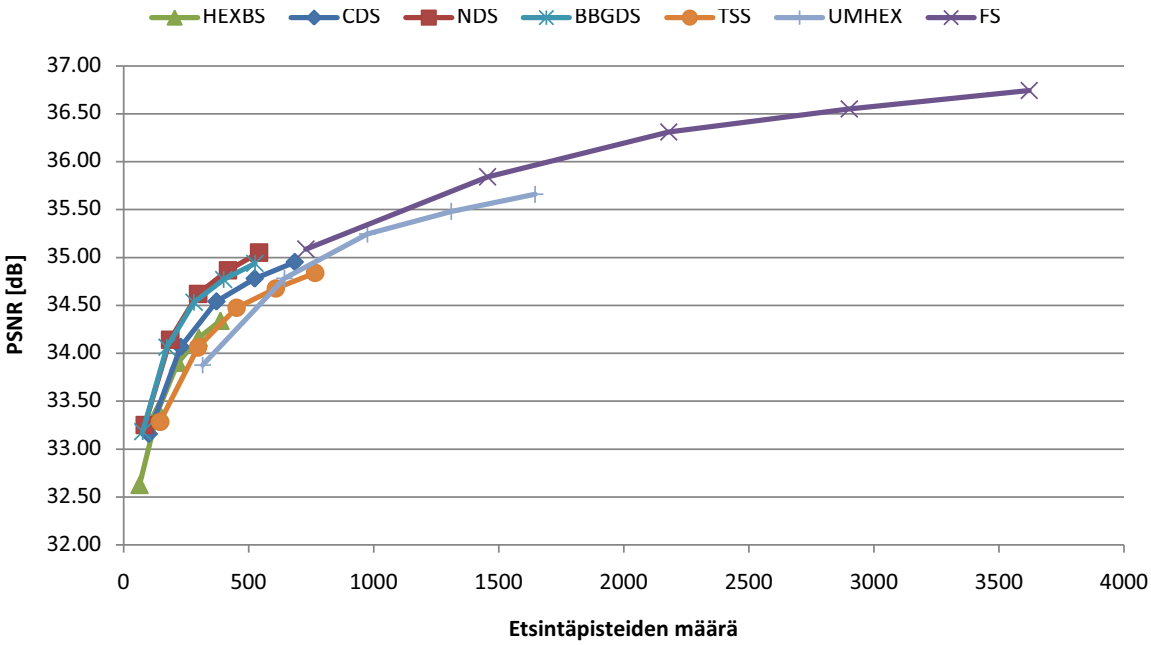
	Moodi 1		Moodit 1-4		Moodit 1-7	
Algoritmi	PSNR [dB]	Pisteet/MB	PSNR [dB]	Pisteet/MB	PSNR [dB]	Pisteet/MB
HEXBS	30.67	23.26	32.42	84.46	32.63	87.44
BBGDS	30.58	37.00	32.27	126.05	32.63	129.83
NDS	30.79	34.34	32.58	122.69	32.79	126.97
CDS	30.74	39.38	32.51	137.78	32.72	142.74
TSS	32.67	40.65	34.67	162.72	34.95	169.12
UMHex	32.71	93.99	34.47	375.60	34.76	391.26
FS	33.82	3481.00	35.76	3481.00	36.33	3481.00
AVG	31.71	535.66	33.53	641.46	33.83	646.91

6.2.3 Vertailukuvien määrän vaikutus

Usean vertailukuvan käytöllä liikkeenestimoinnissa ei ole laitteistolohkon kannalta mitään merkitystä, koska lohkoa voi käyttää siihen halutessaan. Laitteistolohko ei siis rajoita tätä millään tavalla. Kuvasta 6.1 voidaan huomata laadun parantuvan hieman jokaisella lisävertailukuvalla, mutta ainoastaan UMHexagonS pääsee FS-algoritmin tuottamaan laatuun. Vertailukuvien lisäämisellä voidaan parantaa kuvanlaatua, jos käytössä on riittävästi laskentatehoa. Kuva 6.2 esittää vastaavan kuvaajan kuin kuva 6.1, mutta nyt X-akselilla on etsintäpisteiden määrä.



Kuva 6.1: Vertailukuvien määrän vaikutus laatuun kun sekvenssinä on Foreman, QP 20, etsintäalue 48×48



Kuva 6.2: Laatu verrattuna etsintäpisteisiin kun sekvenssinä on Foreman, QP 20, etsintäalue 48×48 ja vertailukuvien määrä on välillä 1–5

7. YHTEENVETO

Tässä työssä esitettiin liikkeenestimoinnin ominaisuuksia, jotka testattiin ohjelmallisesti ja osa myös laitteistolohkon kanssa. Samalla työssä tutkittiin miten nämä eri ominaisuudet vaikuttavat kuvanlaadullisesti liikkeenestimoinnin lopputulokseen. Testauksen yhteydessä laitteistolohkosta löytyi useita vikoja verrattaessa sitä ohjelmistototeutukseen. Myös ohjelmistosta löytyi vikoja. Virheiden korjauksien jälkeen molempien järjestelmien ulostulot saatiin samoiksi.

Taulukko 7.1: Yhteenveto eri muuttujien vaikutuksesta PSNR:ään ja pistemäärään sekä huonoimmat ja parhaimmat algoritmit eri parametrien muuttuessa. Vertailuissa ovat mukana vain algoritmit, jotka on toteutettu laitteistolohkoon.

Vaikuttava parametri	Δ PSNR [dB]	Pisteet	Huonoin alg.	Paras alg.
i) Etsintäalue 48×48 -> 112×112	3.58	1.2x	BBGDS	TSS
ii) Quantization Parameter (QP) 36 -> 20	1.28	1x	HEXBS	NDS
iii) Vertailukuvien määrä 1 -> 5	0.92	5x	TSS	BBGDS
iv) Kuvakoko CIF -> 1080p	0.68	1x	BBGDS	TSS
v) Etsintäalgoritmi	0.58	3x	HEXBS	TSS

Taulukossa 7.1 on koottu yhteen kaikki tulokset ja parhaimmat algoritmit vertailtuna PSNR:n suhteen eri parametrien muuttuessa:

i) Etsintäalue vaikuttaa siihen, kuinka isolta alueelta vastaavuutta etsitään. Alueen kasvattaminen parantaa laatua huomattavasti, mutta lisää samalla etsittävien pisteiden määrää. Nopeissa algoritmeissa pisteiden määrä ei välttämättä kasva alueen laajetessa, koska suoritus voidaan lopettaa jo aiemmin.

ii) QP-arvon vaihtaminen vaikuttaa suoraan laatuun ja myös ulostulevan datan määrään. Se ei ole optimointikeino, vaan sitä käytetään säätelemään bittivirran määrää sopivaksi.

iii) Vertailukuvien määrää voidaan lisätä H.264 standardin määrittelemissä rajoissa, kun halutaan parantaa laatua. Koska jokaiselle vertailukuvalle tehdään erikseen etsintä, kasvattaa niiden lisääminen etsintäpisteiden määrää huomattavasti.

iv) Kuvakoon vaihtaminen vaikuttaa eri algoritmien kykyyn etsiä sopivia lohkoja. TSS on ainoa tässä vertailussa olevista algoritmeista, joka aloittaa etsinnän

laajalta alueelta. Tästä syystä se löytää isoilla resoluutioilla kauempana olevat parhaat pisteet.

v) Etsintäalgoritmin vaihtamisella voi olla monenlaista vaikutusta. Yleensä enemmän pisteitä läpikäyvät algoritmit saavat aikaan parempaa laatua. Koska UMHexagonS ja FS on jätetty pois laitteistototeutuksesta, on TSS toteutetuista algoritmeista paras. Huonointa laatua tuottaa HEXBS, joka kuitenkin käyttää vähiten etsintäpisteitä.

Selvästi parhaimpaan keskimääräiseen tulokseen pääsee TSS. Se onkin ainut, joka ei rajoitu tutkimaan keskipisteen ympärillä olevia pisteitä, vaan tutkii keskipisteestä kaukana olevia pisteitä. Huonoimmiksi algoritmeiksi osoittautuivat BBGDS ja HEXBS. Niiden etu onkin etsintäpisteiden vähäisessä määrässä.

LÄHTEET

[1] H. F. Ates ja Y. Altunbasak, “SAD reuse in hierarchical motion estimation for the H.264 encoder,"*IEEE Int. Conf. Acoustics, Speech, and Signal Processing* (ICASSP), pp. 905–908, 2005

[2] C. H. Cheung ja L. M. Po, “A novel cross-diamond search algorithm for fast block motion estimation,"*IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 12, pp. 1168–1177, Joulukuu 2002.

[3] ITU-T ja ISO/IEC JTC 1, “Advanced Video Coding for Generic Audiovisual Services,"ITU-T Rec. H.264 & ISO/IEC 14496-10, Version 1, Toukokuu 2003; Version 2, Tammikuu 2004; Version 3, Syyskuu 2004; Version 4, Heinäkuu 2005 [WWW]. Saatavilla: http://www.itu.int/rec/T-REC-H.264

[4] ITU-T and ISO/IEC JTC 1, “Reference software for advanced video coding,"ITU-T Rec. H.264.2 & ISO/IEC 14496-5 (MPEG-4 Reference Software), 2008 (Uusin hyväksytty) [WWW] Saatavilla: http://www.itu.int/rec/T-REC-H.264.2, uusin luonnos saatavilla: http://iphome.hhi.de/suehring/tml/

[5] T. Koga, K. Iinuma, A. Hirano, Y. Iijima, ja T. Ishiguro, “Motioncompensated interframe coding for video conferencing,"in Proc. Nat. Telecommunication Conf., New Orleans, LA, 1981, pp. G5.3.1–5.3.5.

[6] L. K. Liu ja E. Feig, “A block-based gradient descent search algorithm for block motion estimation in video coding,"*IEEE Trans. Circuits Syst. Video Technol.*, vol. 6, no. 4, pp. 419–422, Elokuu 1996.

[7] O. Lehtoranta, “Parallel Encoder Implementations for High Quality Video,"Ph.D. Thesis, Tampereen Teknillinen Yliopisto, 2007.

[8] E. Loli-Queru, (Toukokuu 1., 2010). “Why our Civilization’s video art and culture is threatened by the MPEG-LA,"Noudettu toukokuu 12., 2010 [WWW] Saatavilla: http://www.osnews.com/story/23236/

[9] I. E. G. Richardson, (2003). “H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia,"Chichester: John Wiley & Sons Ltd..

[10] x264 - a free h264/avc encoder. VideoLAN team. Noudettu toukokuu 12. 2010 [WWW] Saatavissa: http://www.videolan.org/developers/x264.html

[11] T. Wiegand, G.J. Sullivan, G. Bjontegaard ja A. Luthra, “Overview of the H.264/AVC video coding standard,"*Circuits and Systems for Video Technology, IEEE Transactions on* , vol.13, no.7, pp.560–576, Heinäkuu 2003.

[12] H.264/MPEG-4 AVC. (Toukokuu 11., 2010). In Wikipedia, The Free Encyclopedia. Noudettu Toukokuu 12., 2010, [WWW] Saatavilla: http://en.wikipedia.org/w/index.php?title=H.264/MPEG-4_AVC&oldid=361499160

[13] H.264/MPEG-4 AVC products and implementations. (Toukokuu 8., 2010). In Wikipedia, The Free Encyclopedia. Noudettu Toukokuu 12, 2010, [WWW] Saatavilla: http://en.wikipedia.org/w/index.php?title=H.264/MPEG-4_AVC_products_and_implementations&oldid=360973914

[14] Exponential-Golomb coding. (2010, Toukokuu 3.). In Wikipedia, The Free Encyclopedia. Noudettu Toukokuu 12, 2010, [WWW] Saatavilla: http://en.wikipedia.org/w/index.php?title=Exponential-Golomb_coding&oldid=359887460

[15] Teräväpiirtotelevisio. (toukokuu 7., 2010). In Wikipedia, The Free Encyclopedia. Noudettu toukokuu 12., 2010 [WWW] Saatavilla: http://fi.wikipedia.org/w/index.php?title=Teräväpiirtotelevisio&oldid=8449503

[16] x264. (Toukokuu 6., 2010). In Wikipedia, The Free Encyclopedia. Noudettu Toukokuu 12., 2010, Saatavilla: http://en.wikipedia.org/w/index.php?title=X264&oldid=360509566

[17] C. Zhu, X. Lin, ja L. P. Chau, “Hexagon-based search pattern for fast block motion estimation,"*IEEE Trans. Circuits Syst. Video Technol.*, vol. 12, no. 5, pp. 349–355, Toukokuu 2002.

[18] S. Zhu ja K. K. Ma, “A new diamond search algorithm for fast blockmatching motion estimation,"*IEEE Trans. Image Process.*, vol. 9, no. 2,pp.287–290 Helmikuu 2000.