H.264 MPEG4/10 AVC

Tutkielma eräästä videokoodausstandardista

Jussi Hanhijärvi

ISBN 978-952-92-5250-3

Nimike: H.264 MPEG4/10 AVC
Tutkielma eräästä videokoodausstandardista

Tekijä: Jussi Hanhijärvi

Julkaisija: Jussi Hanhijärvi

Luokka: UDK 621.397
YKL 62.677

Kuvaus: Yleiskuvaus H.264 videokompressiostandardin keskeisistä piirteistä ja sovellusmahdollisuuksista

Avainsanat: Video, kompressio, kvanttisointi, kosinimuunnos, liikevektori, standardointi, muunnosalueen koodaus, profiili, ennuste

ISBN: 978-952-92-5250-3

Laji: Tietokirja, tekniikka

Julkaisupaikka: Helsinki

Julkaisuaika: Helmikuu 2009

Formaatti: pdf

Kieli: Suomi

H.264 MPEG4/10 AVC

Tutkielma eräästä videokoodausstandardista

Laadittu tekijän omaksi iloksi ja huviksi

25.2.2009

Jussi Hanhijärvi

H.264 on uusi videokompressiostandardi, jonka voidaan katsoa saaneen lähes lopullisen asunsa kesällä 2005. Standardi lupaa olennaisesti suuremmat kompressiotehokkuudet kuin mitkään aikaisemmat julkisesti hyväksytyt standardit pystyvät tarjoamaan. Standardia suunniteltaessa on ajateltu hyvin laajaa kirjoa erilaisia sovelluskohteita. Varsinaisen videon koodauksen lisäksi siihen on rakennettu mekanismeja, joilla voidaan sovittaa koodaus moneen erityyppiseen alustaan kuten DVD tallenteet, verkkoliikenne, kuvapuhelinsovellukset ja yleisradiolähetykset.

Sillä on mahdollista kompressoida sekä lomitettua että lomittamatonta videota vielä erittäin suurillakin nopeuksilla. Se sisältää mahdollisuuden sopeutua ympäristöihin joissa siirtotien välityskapasiteetti voi vaihdella hetkittäin. Koodattu data esitetään muodossa jossa virheistä toipuminen on aikaisempaa tehokampaa. Mutta toisaalta kustannuksena on entistä suurempi kompleksisuus. Olennaisilta osiltaan se on myös toteutuskelpoinen VLSI piireillä.

H.264 on tyypillinen liikekompensoitu muunnosalueen hybridikooderi, jonka olennaiset piirteet muistuttavat paljon aikaisempia standardoituja koodekkeja. Parantunut suorituskyky ei ole saavutettu millään yksittäisellä tekniikalla vaan useammalla pienellä tekniikalla yhtä aikaa. Siinä on erotettu erikseen videokoodaus ja sopeuttaminen erilaisiin välitysteihin.

Seuraavassa pyritään antamaan jonkinlainen yleiskuva standardin sisällöstä, sovelluksista, eri toteutuksista, lopulta ehdotetuista parannuksista ja suorituskykytesteistä.

Sisällysluettelo:

1. Johdanto

Kun video koodausta tosissaan ryhdyttiin kehittämään 70-luvun puolivälissä, tavoitteena siirtää jonkinlaista videota puhelinlinjoja pitkin, esitettiin seuraavanlainen ajatus: Lähetetään sekvenssin ensimmäinen kuva sellaisenaan, seuraavaksi kuvaksi lähetettävässä jonossa muodostetaan erotuskuva ensimmäiseen nähden. Lähetetään siis vain poikkeamat ensimmäiseen nähden. Kolmas lähetyskuva taasen poikkeama toiseen nähden jne. Vastaanotossa rekonstruoitiin koko kuvajono alkuperäisen kaltaiseksi. Menettelyssä on haittapuolena se, että virheet kasaantuvat hyvin nopeasti jo muutamassa sekunnissa, tuloksena melkoista ”puuroa”. Tilannetta pyrittiin parantamaan lähettämällä avainkuva, koodaamaton kuva, säännöllisin väliajoin. Videosekvenssi jaettiin siis jonkinlaisiin kuvaryhmiin GOP (Group Of Pictures), joissa avainkuva toistettiin säännöllisin välein.

Hyvin nopeasti opittiin, että itse asiassa avainkuvia ei kannata lähettää sellaisenaan vaan tiivistettynä. Tähän tiivistämiseen on ehdotettu lukuisia eri menetelmiä. On käytetty fraktaaleja, vektorikvanttisointia, aallokkeita ja muita keskenään kilpailevia menetelmiä. Kullakin menetelmällä on omat hyvät ja toisaalta myös huonot puolensa. Vuonna 74 esittivät tutkijat Ahmed, Natarajan ja Rao [AHME74] menettelyn, missä kuva kannattaa koodata ns. diskreetillä kosinimuunoksella DCT:llä. Tässä kuvan kukin värikanava on jaettu 8 x 8 lohkoihin. Kullekin lohkolle tehdään kosini muunnos. Idea oli hyvin kantava, sikäli että sen muodostaminen on kohtalaisen nopeaa nykyisillä tietokoneilla, on hyvin lähellä ideaalista Karhunen-Loeve muunnosta ja on sovellettu hyvin monissa kuvan ja videon tiivistämissovelluksissa.

Muunnoksen etuna on se, että kaikista 64:stä kertoimesta riittää ottaa huomioon vain muutama kerroin. Muunnos pakkaa siis tehokkaasti. Sanotaan alkuperäisen kuvan sisältävän paljon spatiaalista redundanssia, jonka kosini muunnos suodattaa tehokkaasti pois. Tuonnempana käsitellään erästä kokonaisluvuilla tehtyä variaatiota alkuperäisestä kosinimuunnoksesta. Sellaista muunnosta, joka kohdistuu vain kuvaan itseensä eikä muihin sekvenssin kuviin sanotaan intrakoodatuksi kuvaksi tai lyhyesti I-kuvaksi.

Kosini muunnoksen tuloksena saadaan siis yhtä monta kerrointa kun lohkossa on elementtejä. Yhden värikanavan 8 x 8 alueelta saadaan 64 reaalilukuista kerrointa. Näistä vain muutama on merkittävä ja lisäksi muunnoksen tuloksena on aivan liian tarkka lukuarvo (”liikaa numeroita”). Kvanttisoinniksi sanotaan prosessia jossa kertoimia typistetään ja niiden lukumäärää vähennetään. Tuloksena on merkittävä I kuvien kompressio.

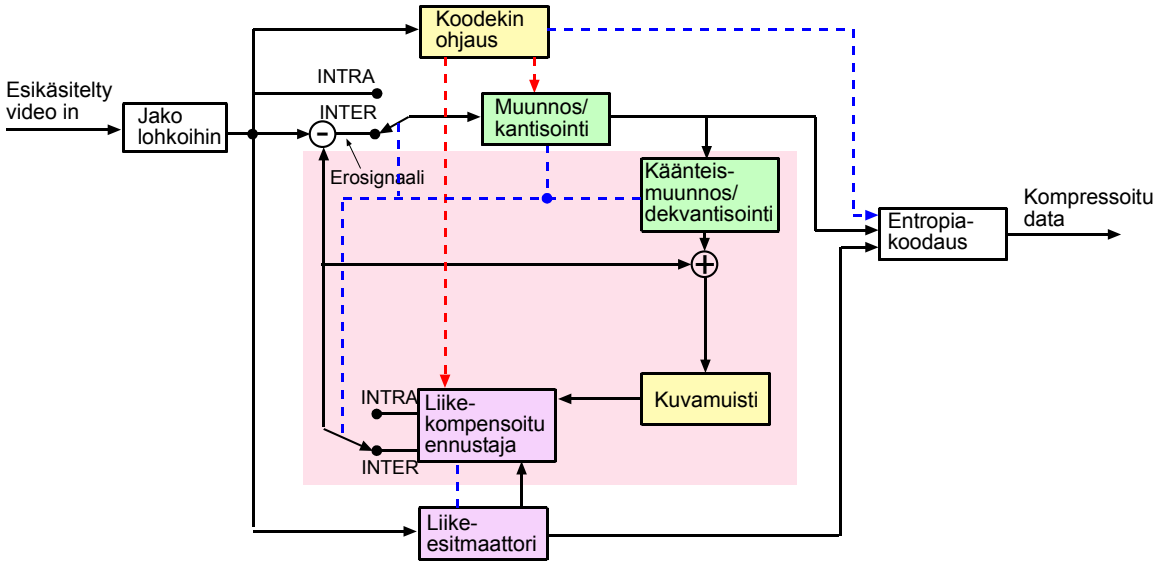
Hyvin nopeasti videokoodauksen alkuaikoina huomattiin se ilmeinen tosiasia, että peräkkäisissä kuvissa on paljon samankaltaisia elementtejä hieman eri paikoissa. Esimerkiksi käsi on saattanut liikkua hieman eri paikkaan myöhemmissä kuvissa. Peräkkäisistä kuvista ei erotuskuvan lähettäminen sellaisenaan olekaan järkevää. Sen sijaan kannattaisi tutkia mihin suuntaan tietty elementti on liikkunut. Varsinkin kun I kuvissa tämä elementti on jo koodattu. Peräkkäisistä kuvista pyritään etsimään liikevektoreita tietyille lohkoille. Nyt riittää, että lähetään lohkotiedon lisäksi tämä liikevektori sekä mahdolliset poikkeamat (nämäkin DCT muunnettuna). Aikaisemmissa standardeissa tämä liikevektoreihin liittyvä lohko oli kooltaan kiinteä ja melko suuri (16 x 16). Sitä kutsuttiin makrolohkoksi. Nykyisissä videokoodekeissa makrolohkon koko saattaa vaihdella.

Sanotaan, että video sisältää paljon temporaalista redundanssia, jolla tarkoitetaan sitä, että ajallisesti sekvenssi sisältää paljon samankaltaista tietoa siirryttäessä kuvasta toiseen. Tehokkaalla liikevektoreiden muodostamisella ja peräkkäisistä kuvista saman elementin poistaminen vähentää olennaisesti lähetettävän datan määrää ilman merkittävää laadun heikkenemistä.

No hyvä. Liike-estimaattorilla ja muunnoksella saadaan kohtalaisen hyvä koodekki jo aikaiseksi. Jonkin verran jää vielä jäljelle sellaisia osia mitä ei ole kovin hyvin voitu ennustaa liikevektoreilla. Itse alkuperäisestä video sekvenssistä vähennetään vielä ennustettu kuva. Siis kun kun kuvaryhmän, GOP:n, ensimmäinen intrakoodattu kuva on muodostettu, muodostetaan tälle käänteismuunnos, tallennetaan kuvamuistiin, lasketaan liikevektoreiden paikat kuvassa ja pyritään ennustamaan millainen olisi seuraava koodattava kuva. Tämä vähennetään sitten alkuperäisen sekvenssin kuvasta. Vähennyslaskun tuloksesta tehdään kosinimuunnos ja käytetään uudelleen hyväksi. Nyt varsinainen lähettävä kuvajoukko muodostuu aika-ajoin lähettävistä I kuvista, joiden välissä on ennustettuja kuvia. Nämä ennusteet voidaan muodostaa joko aikaisemmista kuvista ns. P kuvista tai tulevista kuvista ns. B kuvista. Temporaalisesta redundanssista johtuen voivat nämä erotuskuvista tehdyt muunnokset olla datamäärältään hyvin pieniä.

Lopulta ennen koodatun videon lähettämistä lähetyskanavaan , DVD-levylle internettiin tai muualle pyritään siihen että usein toistuvat koodisanat koodataan lyhyemmiksi kuin harvoin toistuvat. Muodostetaan ns. entropiakoodaus.

Oheisessa kuvassa 1 on esitetty edellä kuvattu ns. hybridi video enkooderi malli (lähes kaikkia laajemmalti käytettyjä videokoodaustapoja luonnehtiva kaaviokuva).



Kuva 1 *Hybridin videoenkooderin toimintaperiaate: Aikaisemmista kuvista muodostettu liikekompensoitu ennuste vähennetään prosessoitavasta kuvasta, jäännös eli liikekompensoitu erotuskuva (residuaalinen kuva) muunnetaan lohkomuuntimella (kuten DCT:llä), kvantisoidaan ja koodataan lähetystä varten. Jaksollisin välein lähetetään yksinomaan lohkomuuntimella muodostetut avainkuvat.*

1.1 H264:n historia

H. 264 on uusin videon kompressioon tarkoitettu julkisesti ja kansainvälisesti hyväksytty standardi. Standardia laati työryhmä ITU-T Q.6/SG16 VCEG (Video Coding Experts Group), projektina H.26L.

Kun monet yleismaailmalliset standardointijärjestöt havaitsevat sellaisen uuden teknisen ongelman, joka edellyttää yhteistyötä ja sopimista esittävät ne kansainväliselle yhteisölle avoimen kutsun tai pyynnön erilaisista teknisistä ehdotuksista standardointiyön pohjaksi. Näitä teknisiä ongelmia voi olla hyvinkin monenlaisia. Videon ja audion koodauksessa ensisijaisesti pyritään siihen, että laitteet ja niiden osat olisivat keskenään yhteensopivia. Tällöin pyritään kaikista ehdotuksista valitsemaan sellaiset, jotka vastaisivat mahdollisimman hyvin sen hetkistä teknistä tietämystä. Tässä yhteydessä on itse asiassa tärkeätä huomata, että lähes kaikki videokompressiostandardit määrittelevät vain sen miten streami pitää purkaa eikä sitä miten se muodostetaan. Ratkaisevat vastaanoton ongelman.

Kesäkuussa 2000 MPEG yhteisö esitti kutsun, jonka tavoitteena olisi muodostaa uusi tapa muodostaa videon säästökoodaus, kompressio. ITU-T VCEG työryhmällä oli työn alla H.26L projekti. ITU myös ehdotti MPEG:lle, että projekti voisi vastata niitä tavoitteita, mitä uudelle kompressiotavalle voitaisiin asettaa. Suoritetuissa testeissä H.26L osoittautuikin kaikista ehdotetuista parhaimmaksi ja kompressiotehokkuus oli merkittävästi parempi kuin jo MPEG-4 osaan 2 hyväksytty standardi (H.263)[1]. Testien seurauksena joulukuussa 2001 ISO MPEG ja ITU VCEG ryhmä yhdistivät voimavaransa saattaakseen H.26L:n loppuun. Tässä yhteydessä on hyvä muistaa, että MPEG-4 standardiperheessä on itse asiassa kaksi eri videokoodausstandardia MPEG-4/2 ja MPEG-4/10.

Kun MPEG-4/10 standardi valmistui vuoden 2002 loppupuolella on sitä ryhdytty nimittämään H.264:ksi , H.264/AVC:ksi (Advanced Video Coding) tai lyhyesti vain AVC:ksi. Koska standardi täydentää MPEG-4 standardiperhettä kutsutaan sitä myös nimillä MPEG-4/AVC tai MPEG-4 part 10:ksi, myös virallista ISO/IEC nimeä ISO/IEC 14496 Part 10 tapaa kirjallisuudessa käytettävän. Koska standardi syntyi kahden järjestön yhteistyönä, niin monissa yhteyksissä myös nimi JVT (Joint Video Team) esiintyy. JVT:n alaisuudessa laaditaan myös standardista skaalattavaa versiota. Skaalattavaa versiota laatii ryhmä SVC (Scalable Video Coding). Tämän ryhmän työn tuloksista saatetaan käyttää taasen nimeä H.264/SVC. Tässä raportissa, sekaannusten välttämiseksi, käytetään koko videokompressiostandardista ytimekkäästi nimeä H.264.

Jotta maallikko menisi oikein sekaisin näistä nimityksistä, todettakoon, että MPEG standardiperheitä on useita eri. Puhutaan MPEG-1:stä, MPEG-2:sta, MPEG-4:stä, MPEG-7:stä ja MPEG 21:stä:

- MPEG-1 käsittelee video ja audio koodausta. Video osuus on jo hieman vanhentunut mutta audiokoodaus (osassa 3) on vielä sangen elinvoimainen tekniikka.

[1] Mielenkiintoisena yksityiskohtana mainittakoon, että H.26L työn alkuvaiheessa ja monissa MPEG kutsuun vastatuista ehdotuksista oli myös esitetty aallokkeisiin perustuvia lähestymistapoja. Mutta ilmeisesti ehdotusten arvioijat katsoivat, että teknologia ei ole vielä riittävän kypsä ja siksi eivät ehkä saaneet riittävää jatkotoimenpiteisiin perustuvaa kannatusta.

- MPEG-2 on erittäin laajalle levinnyt. Sitä sovelletaan mm. DVD levyissä, DVB televisiolähetyksissä. Standardiin sisältyvä System osio kuvaa sitä miten eri lähteistä tulevat osat (teksti, kuvat, video, ääni jne). liitetään yhteen. Eikä toistaiseksi kovin paljon parempaa ole laajemmalti hyväksytty.
- MPEG-4 on yleinen multimedia standardi, joka kahden video osuuden lisäksi sisältää paljon muitakin seikkoja kuten virtuaalitodellisuuden muodostamista, vastaanoton synkronointia jne.
- MPEG-7 taasen on keskittynyt sisällön kuvauksiin ja metadata formaatteihin, eräänlaisena XML laajennuksena.
- MPEG-21 taasen on keskittynyt multimediadatan palvelurakenteisiin eli niihin toimintoihin mitä esiintyy multimedian tuottamisessa, tallentamisessa, jakamisessa ja uudelleen käytössä.

H.264 standardin kehityksen virstanpylväät ovat olleet seuraavat [NILS05, WIEG03a]:

- Varhain 1998 ITU-T:n videon asiantuntijaryhmä VCEG (ITU-T SG16 Q.6) pyytää ehdotuksia projektille H.26L, jonka tavoitteena on kaksinkertaistaa koodaustehokkuus silloisiin tehokkaimpiin koodekkehin (H.263, H.263+) nähden.
- Lokakuu 1999 H.26L:n ensimmäinen versio julkaistaan
- Joulukuu 2001: JVT perustetaan VCEG:n ja MPEG:n väliseksi yhteistyöelimeksi, jonka tehtävänä on uusi videon säästökoodausstandardi.
- Toukokuu 2002: MPEG komitea luonnos julkaistaan.
- Maaliskuu 2003: ITU-T ja MPEG formaali standardiehdotus (FDIS)
- Toukokuu 2003: ITU-T hyväksyntä
- Joulukuu 2003: Verifiointi testit
- Toukokuu 2004: Ensimmäiset korjaukset hyväksytään
- Maaliskuu 2005: Erilaiset toimintatavat FREext (Fidelty Range Extension), konformanssi- ja referenssiohjelmisto julkaistaan.
- Kesäkuu 2005: FREext, konformanssi ja muut tekijät hyväksytään. Standardin voidaan katsoa saaneen lopullisen asunsa. Jotta tästä tulisi kansainvälinen, virallinen ISO standardi, käynnistyy melko mittava kansainvälinen äänestyskierros.

Koska standardi määrittelee vain bittivirran syntaksin ja dekoodausprosessin, niin on melko todennäköistä, että H.264:ään liittyvä tutkimus- ja kehitystoiminta ei pysähdy tähän, vaan lähiaikoina tullaan näkemään paljonkin uusia parannuksia. Varmaankin tullaan näkemään esimerkiksi tekstuurianalyysin soveltamista, morfologista koodausta jne. ainakin enkooderin puolella. Julkaisun pitäisi ymmärtääkin olevan jonkinlainen ”state of the art” julkaisu liikekompensoidusta hybridistä videokoodauksesta, jota ovat olleet laatimassa sadat asiantuntijat ympäri maapalloa ja jonka taustalla on mittava määrä tutkimusta.

H.264:stä on julkaistu monia hyviä yleiskuvauksia, mainittakoon[2]:

- IEEE transactions on circuits and system for video technology, kesäkuun 2003 teemanumero[3] oli omistettu kokonaan H.264:lle. Monet kirjoittajat osallistuivat aktiivisesti standardin kehittämiseen. Artikkeleiden voidaan katsoa muodostavan keskeisen referenssin standardin tilaan vuonna 2003.
- Sullivan et al. julkaisu [SULL04b] sisältää yleiskuvan ja johdannon FRExt laajennuksiin.
- Sullivan ja Wiegand:n artikkeli [SULL04a] sisältää lyhyen yleiskuvauksen aiheeseen.
- Puri et al. julkaisu antaa implementointilähtöisen kuvauksen standardista [PURI04].
- EBU Technical Review lehdessä on julkaistu Schäfer et. al kirjoittama artikkeli [SCHÄ03]. Artikkeli ei ole niinkään kirjoitettu videoyhteisön asiantuntijoille tai toisille tutkijoille vaan antaa kohtalaisen kansantajuisen yleiskuvan aiheesta.
- Ian Richardson on kirjoittanut kirjan aiheesta [RICH03], samoin hänellä on kokoelma ns. White Paper:eita H.264:stä, joista vanhimmat ovat jo vuodelta 2002 ja uusimmat 2004 [RICH04]. Samoin hän on laatinut hienon webbisvun aiheesta url:ssä: http://www.vcodex.com/h264.html. Moniin siellä esitettyihin käsityksiin kannattaa kuitenkin suhtautua hieman varovasti.
- Jörn Ostermann et al. julkaisivat kattavan yleistajuisen esityksen H.264 standardista, suorituskyvystä ja kompleksisuudesta [OSTE04]

Koska JVT toimii avoimena prosessina niin internet sisältää myös paljon kehitystrendejä kuvaavaa julkista materiaalia, kuten muistioita, tutkielmia, kontribuutioita, testimateriaaleja jne. Mainittakoon mm. seuraavat URL:t, joista standardin syntymiseen johtaneet julkaisut ovat löydettävissä kuten myös tulevat parannukset:

- JVT:n ja VCEQ:n ftp-protokollalla osoitettava paikka http://ftp3.itu.int/av-arch sisältää erilaisia teknisiä tarkasteluja ja yksityiskohtia, tietokoneohjelmia mm. erilaisia apuvälineitä, testisekvenssejä (yleensä RAW kuvina). Paikka on tutustumisen arvoinen, mutta vaatii melkoisesti sitkeyttä kun sadat tai tuhannet dokumentit ovat samanarvoisessa asemassa, jaoteltuna korkeintaan kokouspaikan ja –ajan mukaan. Mutta toisaalta sieltä on löydettävissä myös kaikki dokumentit, jotka ovat ohjanneet standardin kehitystä.
- Site ftp://standards.polycom.com ylläpitää myös standardiluonnosta ja dokumentteja koskien referenssitoteutusta.
- IMTC (International Multimedia Telecommunication Consortium) ylläpitää erilaisia toimintaryhmiä, joista yksi on VCEQ list palvelin http://mail.imtc.org/read/?forum=vceg. Liikennöinti ei tosin ole kovin vilkasta mutta sisältää ilmoituksia erilaisista tilaisuuksista. Liittyminen tapahtuu http://www.imtc.org:n kautta. Samoin Aachenin yliopisto ylläpitää kaksi JVT:hen liittyvää julkista list palvelinta
    - http://mailman.rwth-aachen.de/mailman/listinfo/jvt-experts

[2] Aiheen keskeisen merkityksen vuoksi julkaisujen määrä on kasvussa.
[3] Teemanumero sisälsi paperit [HORO03], [MALV03], [WIEN03], [LIST03], [MARP03], [KARC03], [WENG03], [STOC03], [RIBA03], [WIEG03a], [WIEG03b], [LAPP03], [LUTH03].

- http://mailman.rwth-aachen.de/mailman/listinfo/jvt-bitstream

  Viimeksi mainituissa liikennöinti oli kohtalaisen vilkasta lähes päivittäistä, standardin julkaisun päivien seutuvilla, nykyään liikennöinti kuolee hitaasti pois. Suurin osa kirjeenvaihdosta koskee erilaisia detaljeja joko referenssitoteutuksesta tai standardin tekstistä, jossain määrin postituslistat ovat myös suunnattu kehitystyön hallinnointiin. Listoilla ei juurikaan käydä keskustelua (ja siksi on hieman tylsä). H.264:n projektin johtajat yleensä vastaavat täsmällisesti ja hyvin määriteltyihin kysymyksiin.

- MPEG:n suhteen on kaksi paikkaa mainitsemisen arvoisia MPEG yhteisön ”virallinen” site on osoitteessa http://www.chiariglione.org/mpeg/ ja toinen ns. MPEG teollisuusfoorumi (http://www.mpegif.org) Ensin mainittu sisältää kaikkea MPEG:iin liittyviä seikkoja kun taas jälkimmäinen on suuntautunut enemmän tai vähemmän MPEG-4:ään. Teollisuusfoorumin url on kohtalaisen hyvin järjestetty. Näistä kahdesta paikasta kannattaa yleensä aloittaa kun MPEG kirjainyhdistelmä tulee vastaan – ties missä yhteydessä.

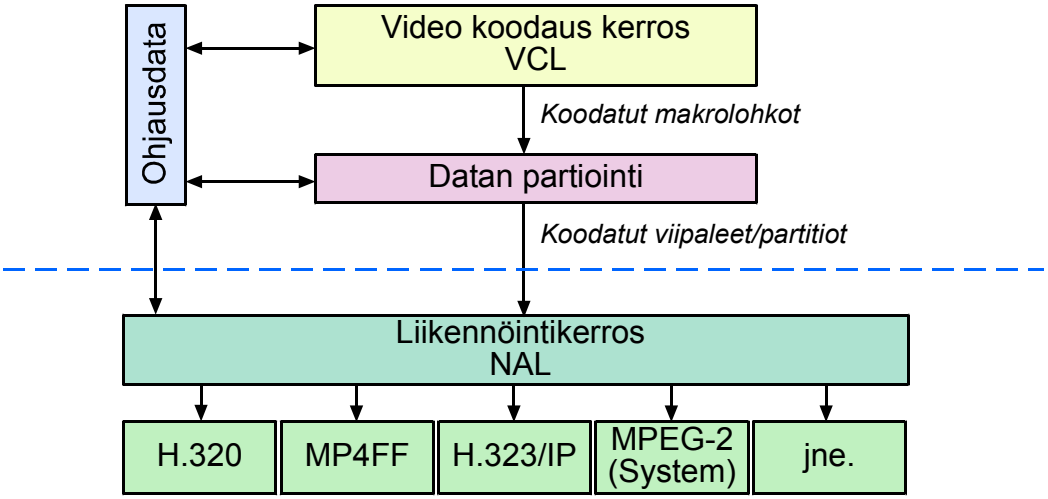
- H.264:n ns. referenssiohjelmisto (sekä en- että dekooderi) on saatavissa osoitteessa http://iphome.hhi.de/suehring/tml/

2 Yleistä

H.264:n suunnittelun lähtökohtana oli tehdä mahdollisimman tehokas nykytietämystä vastaava koodekki, jota voidaan soveltaa mahdollisimman moneen sovellukseen. Toivottavaa olisi myös, että se voitaisiin toetutta myös VLSI-piireillä tehorajoitteisissa sovelluksissa (kuten kännyköissä). Kompressiotehokkuus MPEG-2 videokoodaukseen näden on kasvanut 1.5 – 2.2 kertaiseksi ja vastaavasti toteutuksen kompleksisuus 2.5 – 4 kertaiseksi [SUNN05, SCHÄ03]. Standardin suunnittelijoiden mielessä oli mm. seuraavia sovelluskohteita [WIEG03b]:

- Televisio-ohjelmien välittäminen (broadcasting) eri medioita pitkin kuten satelliittien ja kaapelien kautta, maanpäällisenä radiotaajuisena lähetyksenä, laajakaistaverkkoja pitkin jne.
- Tallennus eri optisiin tai magneettisiin tallennusvälineisiin kuten kovalevyihin HD-DVD jne.
- Keskustelu- tai vuorovaikutussovellukset teknisinä alustoina esimerkiksi ISDN, kaapelimodeemit, ethernet, langattomat ja mobiilialustat.
- Streaming palvelut eri langallisilla ja langattomilla alustoilla.
- Mulltimedia messaging palvelut (MMS)

Näinkin laajan kirjon sovelluksia on saavutettu suunnittelemalla videon koodaus huolellisesti, liitämällä eri toimintatapoja (profiileja ja laajennuksia) sekä erottamalla varsinainen videon koodaus liikennöinnistä. Niinpä H.264 on perusrakenteeltaan kaksi-kerroksinen: erikseen on luotu mekanismit lähetys- ja vastaanottoliikennöintiin sopeutumiseksi (ns. NAL, Network Abstraction Layer) ja toiseksi varsinaiseen videokoodaukseen (VLC, Video Coding Layer) kuva 2. NAL kerroksessa muotoillaan videokoodauksen toimittama data välitystiehen sopivaksi sekä lisätään vaadittavat otsakkeet ja muut liikennöinnin edellyttämät seikat kuten paketointi toimitettavaan datavirtaan.



Kuva 2: *H.264:n looginen rakenne: Varsinainen videokoodaus on erotettu liikennöinnistä, joka edelleen sopeutuu hyvin monenlaiseen eri lähetys- ja vastaanottotapaan.*

Yleisradiosovellusten kannalta on mielenkiintoista havaita, että NAL sallii MPEG-2 System standardissa esitetyn multipleksoivan paketoinnin, jolloin standardia voidaan soveltaa melko joustavasti kaikissa DVB sovelluksissa (siis DVB-T, DVB-S2, DVB-H). Samoin liittyminen H.32x sarjan liikennöintistandardeihin mahdollistavat langattoman ja langallisen vuorovaikutusformaatit esimerkiksi videoneuvottelut, kuvapuhelimet ja monet muut sovellukset. Erilaiset liitokset IP protokolliin (kuten UDP/IP ja RTP/IP) ovat standardia suunniteltaessa olleet keskeisiä näkökohtia [WIEG03b].

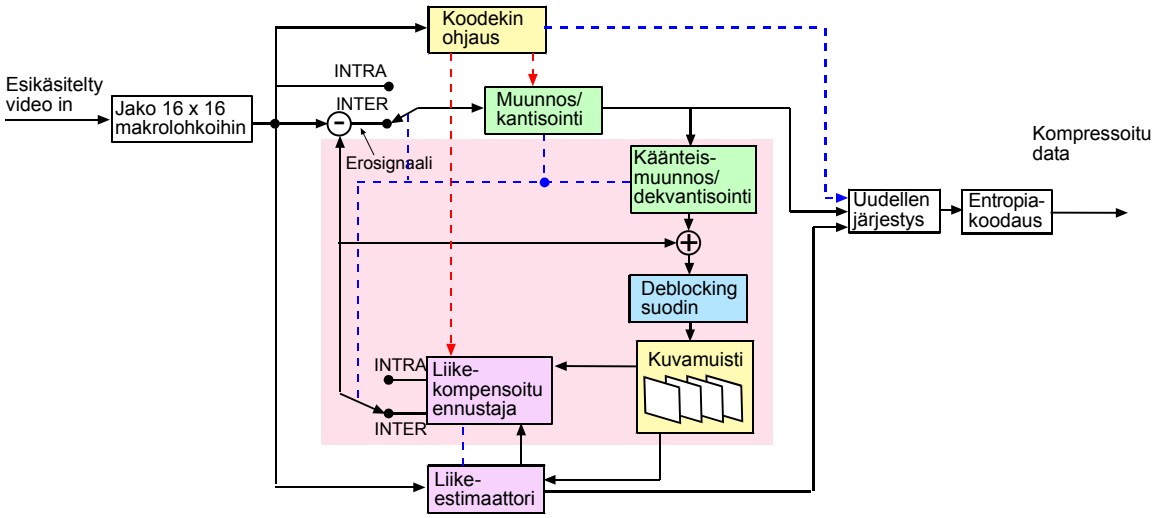
H.264:n videokoodauskerros (VCL) ei perustu niinkään mihinkään yksittäisen tekniikan soveltamiseen vaan useampaan pieneen parannukseen eri kompression osa-alueilla kuten liikekompensoinnissa, intrakoodauksessa, takaisinkytkentäketjussa olevaan lohkorajat häivyttävään suodatukseen, kehittyneeseen kvanttisointiin, joustavaan makrolohkojen ryhmittelyyn jne. Loppukäyttäjän kannalta merkittävin parannukset ovat mm:

- Selkeästi parantunut laatu sekä alhaisilla että korkeammilla siirtonopeuksilla. Se mahdollistaa DVD-laatuisen videon lähettäminen nykyisin käytössä olevia siirtokanavia pitkin, ilman merkittävää kustannusten nousua [KOSS03, SCHÄ03], jopa niinkin alhaisilla nopeuksilla kuin 4 Mb/s (DVD-laatu) ja 2 Mb/s (SDTV-laatu).
- Palautumiskyky vioista. H.264:ssä on olemassa työkaluja ja algoritmeja, joilla voidaan joustavasti käsitellä pakettien katoamista paremmin kuin aikaisemmissa standardeissa.
- Verkkoystävällisyys. Erottamalla toisistaan liikennöintirajapinnat varsinaisesta signaalinkäsittelystä voidaan H.264:llä koodatut sekvenssi välittää hyvinkin eri tyyppisiin verkkoihin [KOSS03].
- Laaja kokoelma toimintatapoja ja kuvaformaatteja.
- Mahdollisuus liittää saumattomasti muuhun videonkäsittelyyn ja hyvin vaihtelevan tyyppisiin alustoihin.

3 Videokoodauskerros VCL

H.264:n videokoodausarkkitehtuuri periytyy aikaisemmista video standardeista. Lyhyesti sanottuna: se on lohkopohjainen muunnosalueen liikekompensoitu ns. hybridikooderi. Tässä spatiaalinen redundanssi minimoidaan muunnosalueen koodauksella sekä

temporaalinen redundanssi kehittyneellä liikevektoreiden valinnalla ja ennustevirhe pienennetään toistamalla jäännökselle muunnosalueen koodausta (residuaalinen koodaus). H.264:n toimintaperiaatetta hahmottaa oheinen kuva 3:



Kuva 3: *H.264:n VCL:n rakenne (hyvin samankaltainen kuvan 1 kanssa)*

Vaikka toteutusalgoritmit monien nykyisten standardoitujen kompressiostandardien kanssa poikkeavat ratkaisevasti niin H.264:ssä on laadittu samassa hengessä kuin monet aikaisemmin toteutetut koodekit. Parantunut suorituskyky ei johdu minkään yksittäisen tekniikan soveltamisesta, vaan useamman tekijän yhteisvaikutuksesta. Näitä tekijöitä ovat mm. seuraavat [WIEG03b, SCHÄ03]:

- Kehittynyt liikekompensointi. Ennusteiden muodostamisessa lohkojen koot voivat olla vaihtuvanmittaisia, referenssikuvia voi olla useampi. Ennusteet voidaan painottaa tietyllä kertoimella ja niissä voi olla myös offset, jolloin ennustetarkkuus myös kasvaa.
- Makrolohkot voidaan ryhmitellä itsenäisesti käsiteltäviin viipaleisiin, joista kukin voidaan koodata ns. I, P ja B viipaleena eli intrakoodattuna sekä eteenpäin ennustavana (P) ja eteenpäin ja taaksepäin ennustavana (B) interkoodauksena.
- 16 x 16 makrolohkot voidaan partitioida pienempiin osiin ennusteiden muodostamista varten. Partitiointi voi olla niinkin pieni kuin 4 x 4.
- Referenssikuvia voi olla useita, enkooderi voi päättää tehdäänkö viitteet edelliseen kuvaan tai tätä aikaisempiin, joten esimerkiksi P kuvassa (viipaleessa) viittaukset voidaan tehdä jopa ennen edeltävään I kuvaan. (MPEG 2:ssa ennusteet muodostetaan aina peräkkäisten I kuvien välissä).
- Koodekin takaisinkytkentälenkissä on lohkorajat hävittävä de-blocking suodatin. Jolloin silmin havaittavat lohkorajat myös minimoituvat.
- Muunnokset tehdään joko 4 x 4 tai 8 x 8 (FREext) diskreetistä kosinimuunnoksesta johdetulla kokonaisluku muunnoksella
- Aikaisemmissa standardeissa käytettiin muuttuvamittaista, Huffman tyyppistä entropiakoodausta, H.264:ssä entropia koodaukseen voidaan käyttää kolmea eri koodaustapaa CABAC (Context Adaptive Binary Arithmetic Coding) tai CAVLC (Context Adaptive Variable Length Coding) sekä Golomb koodausta.
- Parantunut kyky toipua liikennöintiverkon virheistä ja datan häviämisestä. Tämä on saavutettu monilla eri mekanismeilla. Mainittakoon parametrien ja NAL:n

syntaksi, viipaleiden ja makrolohkojen järjestys, redundanttien kuva-alueiden lähettäminen ja datan partitiointi, monien muiden seikkojen ohella.

Oheiseen taulukkoon on hahmoteltu joitakin keskeisiä ominaisuuksia eri koodereista. H.264:n monista eri toimintatavoista on otettu mukaan vain Main profiili ja FRExt laajennuksista High profiili.

Piirre	MPEG-1	MPEG-2	MPEG-4 ASP	MPEG-4, AVC, Main	MPEG-4, AVC, High	VFW-9	Ogg Theora
B-kuvat	X	X	X	X	X	X	
Viipaleet Vikojen ehkäisy)	X	X	X	X	X	X	
Lomitus		X	X	X	X	X	
Entropia koodaus	Huffman	Huffman	Huffman	Exp-colomb, adaptiivinen aritmeettinen	Exp-colomb, adaptiivinen aritmeettinen	Huffman, bitplane	Adaptiivinen Huffman
Liike-ennuste lohko koko	16 x 16	16 x 16	16 x 16, 8 x 8	16x16,16x8, 8x8,8x4,4x4	16x16,16x8, 8x8,8x4,4x4	16x16,8x8, 8x4, 4x4	16x16 8x8
Liikkeen haun tarkkuus (pikseliä)	1, ½	½	½, ¼	¼	¼	½, ¼	½
Globaali liikkeen kompensointi			X				
Intra ennuste	DC, 8x8	DC, 8x8	AC, 8x8	Spatial, 16x16, 4x4	Spatial, 16x16, 8x8 4x4	AC 8x8	DC 8x8
Spatiaaliset muunnokset	DCT 8x8	DCT 8x8	DCT 8x8	HCT 4x4	HCT 8x8, 4x4	8x8, 8x4, 4x4	DCT 8x8
Bittitäsmällinen dekoodaus				X	X	X	X
Häviötön toimintatapa					X		
Sovelluskohtainen kvanttisointimatriisi	X	X	X		X		X
De-block suodatus				X	X	X	X
Monikertaiset referenssiframet				X	X		2
Painotetut ennusteet				X	X	PP	
YUV väriformaatti	4:2:0	4:2:0, 4:2:2,4:4:4	4:2:0	4:2:0	4:0:0,4:2:0, 4:2:2, 4:4:	4:2:0	4:2:0, 4:2:2, 4:4:4

Taulukko 1: *Video koodekkien vertailua: MPEG-4 ASP = MPEG-4 osa 2 (H.263), lähes identtinen DivX:n kanssa, VFW-9, Microsoftin Video for Windows-9, Ogg Theora Open source Xihp.org:n julkaisema koodekki (tiedot ovat vuodelta 2006)*

3.1 Makrolohkoihin jako, viipaleet ja viipaleryhmät

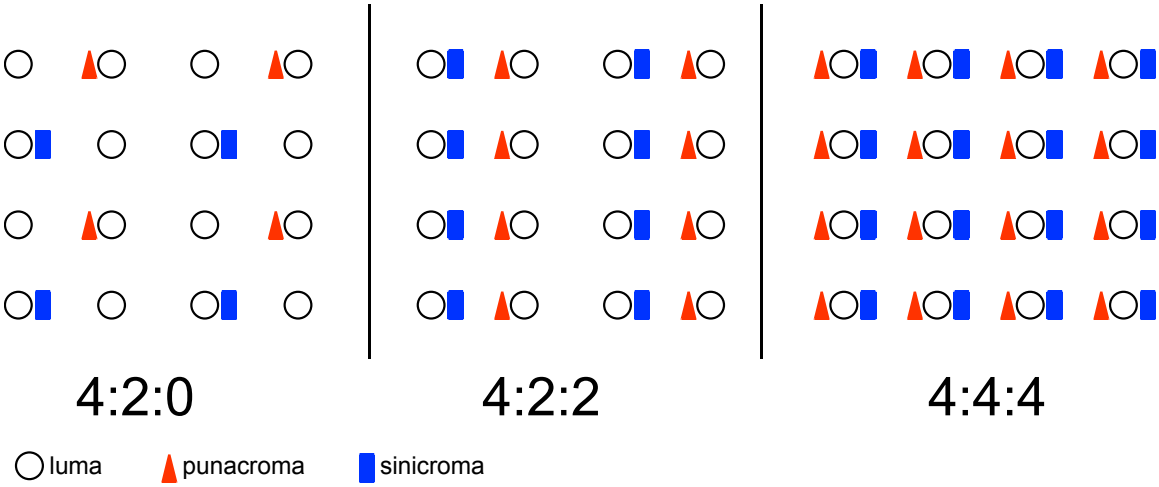
Videosekvenssi koostuu siis sekvenssistä, videovirtaan kuuluvista, runsaasti keskinäistä temporaalista ja spatiaalista redundanssia sisältävistä peräkkäisistä kuvista. Kuvien sisältämistä pikseleistä otetaan 8, 10 tai 12 bittisiä näytteitä kustakin värikanavasta. Värikanavina voi olla esimerkiksi RGB tai televisiotekniikassa tärkeä YCbCr, jonka

komponenttina siis on kirkkaus tieto, luma, (Y) sekä puna (Cr) ja sini (Cb) cromat[4]. Tässä yhteydessä on tärkeä pel:n käsite. Tällä tarkoitetaan pienintä osoitettavissa olevaa kuvan alkiota. Siis esimerkiksi RGB pikseli jakautuu 3:een pel:iin punaiseen (R), vihreään (G) ja siniseen (B).

Värikomponenteista voidaan ottaa erilailla näytteitä esimerkiksi YCbCr: maailmassa standardi tuntee seuraavat näytteenottotavat [ITUT05]:

- 4:2:0 kutakin neljää lumanäytettä (2 x 2 matriisissa) kohden otetaan yksi yksi sinicroma- ja yksi punacromanäyte
- 4:2:2 Kahta lumanäytettä (2 x 1 matriisissa) kohden muodostetaan yksi sini- ja yksi punacroma näyte
- 4:4:4 croma- ja lumataulukot ovat samankokoisia eli kaikkia näytteitä otetaan yhtä paljon

Oheinen kuva 4 hahmottaa tätä näytteenottoa:



Kuva 4: Eri näytteenottotapoja

Jokainen sekvenssin kuva on jaettu kiinteään määrään makrolohkoja joista jokainen sisältää (4:2:0 tapauksessa) esimerkiksi 16 x 16 näytettä luma ja 8 x 8 näytettä kummallekin croma komponentille. H. 264:n uutena piirteenä onkin mahdollisuus partitioida, jakaa, kuva niinkin pieniin osiin kuin 4x4 aikaisempien standardien kiinteän 16x16 lohkokoon sijaan.

Makrolohkot voidaan ryhmitellä hyvin joustavasti viipaleisiin. Viipaleet ovat annetun kuvan sellaisia alueita joita voidaan prosessoida toisistaan riippumattomasti. Tällainen viipaleisiin jako on eräs ratkaiseva erottava tekijä aikaisempiin standardeihin nähden. Tietyt kuvan osat esimerkiksi paljon yksityiskohtia tai liikettä sisältävät osat saattavat vaatia keskimääräistä suuremman prosessoinnin. Viipaleisiin jako tarjoaa joustavan mahdollisuuden tähän.

[4] Standardissa käytetään termiä luma ja croma termejä valoisuudelle ja värille. Syynä sille miksi ei käytetä luminanssi ja krominanssi termejä on se, että yleisessä valoon liittyvässä fysiikassa näillä on huomattavasti laajempi merkitys.

Yksi kuva voi sisältää yhden tai useamman viipaleen ja niiden sisältö prosessoidaan vasemmasta ylänurkasta lähtien alas oikean alanurkkaan. Jokainen viipale on itsenäinen siinä mielessä että niiden syntaksielementit voidaan jäsentää suoraan bittivuosta (streamista). Periaatteessa viipaleen edustama data voidaan dekoodata käyttämättä muita kuvan sisältämien viipaleiden dataa. Esimerkiksi tehokas lohkorajat häivyttävä deblocsuodatus edellyttää toisinaan, että naapuriviipaleen lohkoja voidaan käyttää osana koodausprosessia. Viipaleita voidaan käyttää mm. [SULL04a]:

- Virheisiin sopeutumiseen, kunkin viipaleen alku mahdollistaa uuden synkronointipisteen, josta dekoodaus prosessi voidaan alustaa uudelleen. Kun aikaisemmissa standardeissa lähtökohta oli aina I kuva, niin tässä voidaan kohdistaa vaivannäkö pienempiin osa-alueisiin.
- Hyvin segmentoidun hyötykuorman luomiseen, viipaleet voidaan muodostaa sellaisiin pakettirajoihin, jotka vastaavat verkon maksimisiirtoyksiköitä (esimerkiksi monissa internet sovelluksissa tämä on 1500 tavua).
- Rinnakkaiseen prosessointiin, jokainen viipale voidaan en- ja dekoodata toisistaan riippumattomasti.

Koska jokainen viipale voidaan dekoodata toisistaan riippumattomasti, niin mitään erityistä koodausprosessin järjestystä näillä ei tarvitse määritellä. Tähän liittyy läheisesti ns. arbitrary slice ordering ASO:n käsite. Tällä tarkoitetaan sitä, että kun ASO:a käytetään voivat viipaleet esiintyä missä järjestyksessä tahansa varsinaisessa bittivirrassa. Kun taas ASO:a ei käytetä pitää viipaleiden sisältämien makrolohkojen olla sellaisessa järjestyksessä missä viipaleiden sisältämät ensimmäiset makrolohkot kasvavat samassa järjestyksessä kuin missä videon sisältämä kuva pyyhkäistään (kameran kennolta) [SULL04a].

Viipaleet voivat olla myös redundantteja eli voidaan lähettää kahdennettuja osia tietystä kuvanalueesta. Redundanteilla viipaleilla pyritään usein parantamaan enkoodatun videon robustisuutta. Toisin sanoen häiriöllisissä olosuhteissa voitaisiin pyrkiä siihen, että videon tärkeät osa-alueet (esimerkiksi kasvot) toistuvat mahdollisimman virheettömästi kun taas epäolennaisissa osissa voi olla merkittävä määrä virheitä.

3.1.1 Joustava makrolohkoihin järjestäminen FMO

Viipaleiden virheisiin sopeutuminen voidaan parantaa tekniikalla, jota H.264 standardoijat kutsuvat nimellä felxible macroblock ordering (FMO). FMO:lla voidaan modifioida tapaa miten makrolohkot liittyvät viipaleisiin. Esimerkiksi FMO:lla voidaan kuva jakaa lomitettuihin viipaleisiin, hajotettuun makrolohkojen allokointiin, jakaa yhteen tai useampaan kuvan ”etualaa” kuvaavaan viipaleryhmään jne.

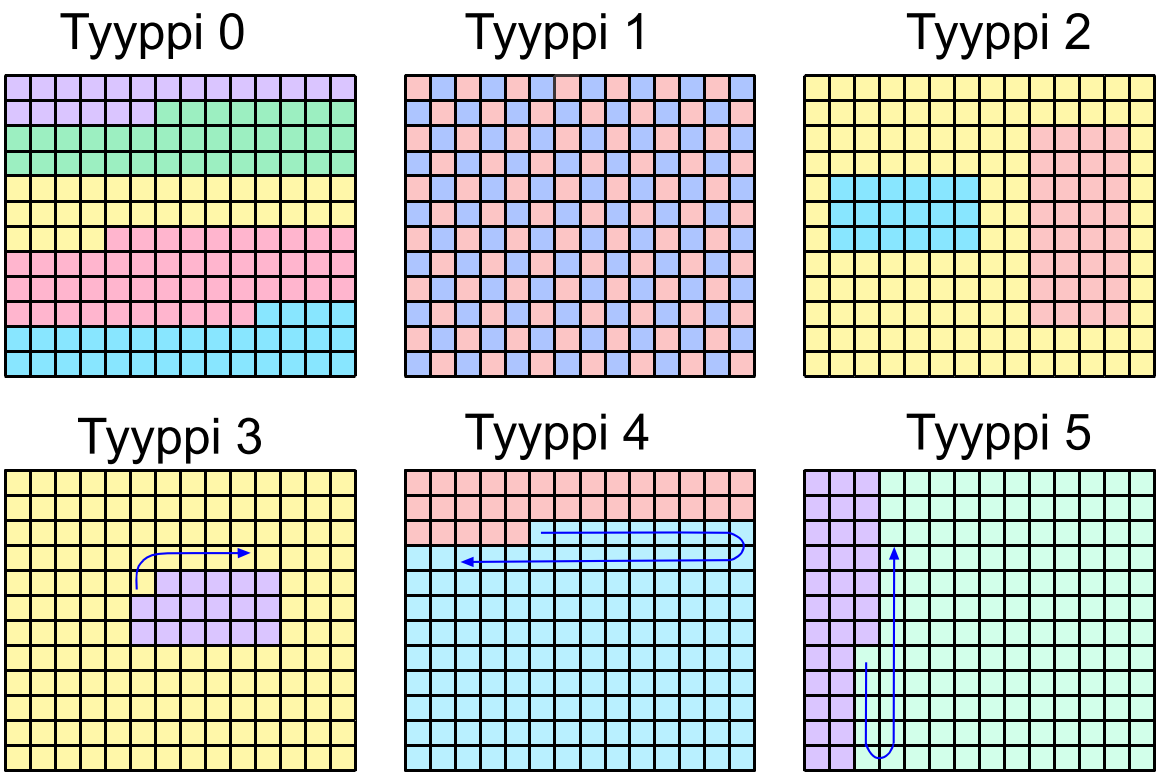
Standardi tuntee 7 erillaista viipalejakoa (vrt. myös oheiseen kuvaan) tyyppi 0 – tyyppi 6. Näistä tyyppi 6 voi olla täysin satunnainen ja käyttäjän itsensä määrittelemä:

Tyyppi 0: samanlaisena toistuva koko kuvan täyttävä pituuskoodi. Dekooderin pitää tuntea pituuskoodi, jotta se voisi dekoodata kuvan. Tyyppi 0:n jako on sama kuin ei käytettäisi ollenkaan FMO:ta.

Tyyppi 1: ns. skatteroidut viipaleet, tässä käytetään sekä en- että dekooderin tuntemaa matemaattista, makrolohkot järjestävää, funktiota. Oheisessa kuvassa 5 on esitetty hyvin yleisesti käytetty shakkilauta järjestelyä.

Tyyppi 2: Tässä viipalejaossa on voidaan merkitä suorakaiteen muotoiset alueet ns. ROI:t (Region Of Interest) eli alueet, jotka ovat kaikkein kiinnostavimmat. Sekä en- että dekooderi tuntee kunkin ROI:n vasemman ylänurkan ja oikean alanurkan.

Tyyppi 3 – 5: ovat dynaamisia tyyppejä, joilla mahdollistetaan viipaleryhmien koon dynaamisen kasvun ja pienenemisen eri kuvien välillä jaksollisesti. En- ja dekooderin pitää tuntea kasvunopeus, kasvun suunta ja sijainti jakson sisällä.



Kuva 5: *Eri FMO tyypit (kukin väri edustaa eri viipaletta*)

Viipale tyyppi 1 on sikäli mielenkiintoinen, että tällä mahdollistetaan jossain määrin videokonferenssin yksityisyys. Eri viipaleet lähetetään eri paketeissa. Kun sekä lähettäjä että vastaanottaja tietävät miten pakettien sisältämät viipaleet järjestetään voivat he muodostaa sisältörikkaan yhteyden. Ulkopuolisella, joka tätä järjestystä ei tunne on suuria vaikeuksia konstruoida videosekvenssi järjelliseen muotoon. Lisäksi jos siirrossa tapahtuu paljon virheitä ja yksittäisten makrolohkojen katoamista voidaan naapuriviipaleita käyttää virheen vaikutuksen peittämiseen (esimerkiksi toistamalla virheen esiintymäkohdalla naapurilohko).

Viipale tyyppi 2:ta voidaan käyttää esimerkiksi silloin kun kuvassa on jotakin mielenkiintoista (kuten kasvot) ja suhteellisen merkityksettömiä muita osia (esimerkiksi tausta). Nyt voidaan säästää siirtokaistaa merkittävästi kompressoimalla paljon taustaa ja kasvoja vähemmän. Eräs toinen sovellus voisi olla eri maksuTV palvelut. Video sekvenssi on kaikille avoin paitsi tietyt kohteet (esimerkiksi jalkapallopelin pallo). Vasta maksamalla käyttömaksu saadaan myös piilossa oleva alue (pelin pallo) näkyviin.

3.1.2 Viipaleiden käsittely

Viipaleita voidaan käsitellä eri tavoilla. H.264 sisältää 5 eri käsittelytapaa nimittäin [SULL04a]:

- I viipaleet, joissa kaikki makrolohkot intrakoodataan eli koodauksessa hyödynnetään ainoastaan spatiaalista korrelaatiota ilman mitään referenssiä muihin sekvenssin kuviin (ja näiden sisältämiin viipaleisiin). Näille tehdään siis pelkästään muunnosalueen koodaus. I viipaleet toimivat myös synkronointikohtina, joista dekoodaus voi lähteä käyntiin
- P viipaleet, joissa makrolohkot koodataan inter ennusteina ja jossa käytetään korkeintaan yhtä liike-ennustesignaalia per ennustelohko. Tässä koodaustavassa hyödynnetään sekvenssin peräkkäisten kuvien välistä temporaalista redundanssia eli sekvenssin toisesta kuvasta tai viipaleesta ns. referenssikuvasta tai referenssiviipaleesta muodostetaan liikevektori.
- B viipaleet, joissa makrolohkot voidaan koodata, P tyypin koodauksen lisäksi, muodostamalla inter ennusteita kahdella liike kompensoidulla ennusteella per ennustelohko. B viipaleiden käyttämät referenssiviipaleet voivat olla menneisyydestä tai tulevaisuudesta tai kummastakin. Ennusteet yhdistetään painotettuna keskiarvona. B viipaleet sinänsä eivät voi muodostaa referenssiviipaleita toisille (kuten I ja P viipaleet).
- SP viipaleet, niin kutsutut switching P viipaleet, jotka koodataan siten, että tehokas kytkeytyminen eri streamien tai saman streamin eri paikkojen välillä olisi mahdollinen vähäisellä määrällä lisäbittejä
- SI viipaleet, niin kutsutut switching I viipaleet, jotka mahdollistavat SP viipaleiden täsmällisen sovituksen esimerkiksi videon satunnaishaussa tai virheestä toipumisessa. SI viipaleisiin sovelletaan intra koodausta.

Kolme ensimmäistä (I, P ja B) ovat periaatteiltaan hyvin samankaltaisia kuin aikaisemmissa standardeissa on käytetty (paitsi, että viitteet voidaan tehdä viipaleisiin eikä pelkästään kuviin). Aktuaalinen toteutus kuitenkin poikkeaa huomattavasti siitä mitä vanhemmat standardit kuvailivat toimintamekanismeista. Esimerkiksi referenssejä voi olla useampia ja referenssinä voi olla myös ennen aikaisempaa I kuvaa esiintyvä kuva. SP ja SI ovat täysin uusia rakenteita.

P ja B viipaleissa hyödynnetään siis sekvenssiin liittyvää temporaalista redundanssia. P ja B viipaleita ei voida dekoodata ilman että olisi korrektisti dekoodattu näiden menneisyydessä tai tulevaisuudessa esiintyviä referenssiviipaleita. Tähän liittyy yksi vakava siirtotien tai videopalvelimen (enkoodausmoottorin) tahattoman toimintahäiriön aiheuttama ongelma. Tapauksissa missä enkooderin ja dekooderin käyttämä referenssiviipale ei ole identtinen esimerkiksi siirrossa tapahtuvan virheen seurauksena niin peräkkäisten kuvien rekonstruktio poikkeaa alkuperäisestä. Tällainen virhe ei pelkästään kohdistu yksittäiseen kuvaan vaan voi ajallisesti kehittyä referenssiltään virheellisen liikekompensoinnin edistymisen myötä [KARC03]. Tämänkaltaisia probleemeja on perinteisesti ratkaistu sijoittamalla riittävän tihein välein bittijonoon I koodattuja kuvia tai viipaleita, jolloin myös tuhlataan koodaustehokkuutta. H.264:ssä voidaan käyttää SP ja SI kuvia tällaisten probleemien ratkaisemiseksi.

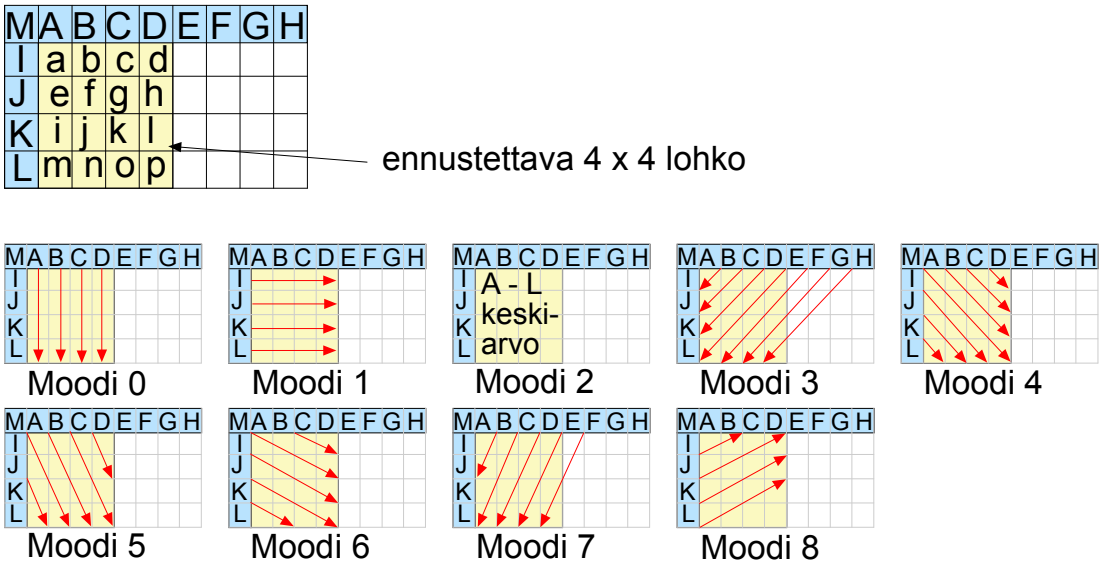
SP ja P kuvien (viipaleiden) merkittävänä erona on se että SP mahdollistaa identtisten kuvien rekonstruktion jopa silloin kun ennusteet on laadittu kokonaan toisista kuvista. Tämän vuoksi SP.tä voidaan käyttää I kuvien (viipaleiden) sijasta sovelluksissa missä bittivuo joudutaan keskeyttämään (esim. toisen vuon esiintyessä bitstream slicing), kytkeytymisessä toiseen esim. nopeudeltaan pienempään vuohon, pikakelaukseen eteen tai taaksepäin tai virheestä toipumiseen [KARC03]. SI on hieman samankaltainen kun SP: SI:ssä käytetään vain spatiaalista ennustamista kun taasen SP:ssä temporaalista ennustamista.

3.2 Intrakoodattujen viipaleiden ennustaminen

Kompressiotehokkuutta voidaan kasvattaa olennaisesti kun otetaan huomioon yksittäisen kuvan lohkojen keskinäistä redundanssia. Voidaan muodostaa ennusteita yksittäisen kuvan sisällä, muodostetaan ns. intraennusteita. Intraennusteiden sanotaan siis olevan koodaustavan jossa hyödynnetään vain kuvaan sisältyvää spatiaalista redundanssia. Intraennusteiden muodostamisessa H. 264:ssä hyödynnetään koodattavan lohkon naapurilohkojen välistä mahdollista spatiaalista korrelaatiota. Koodaustavoissa voidaan käyttää kahta erilaista ennustetyyppiä, joitta kutsutaan joko Intra_4x4:ksi tai Intra_16x16:ksi [SULL04a]. Poikkeuksellisiin tapauksiin on olemassa vielä kolmas tyyppi I_PCM:

- Intra_4x4:ssa ennustetaan jokainen 4x4 luma lohko erikseen, se soveltuu hyvin sellaisiin kuvan osiin missä on merkittävä määrä yksityiskohtia.
- Intra_16x16 taasen on tarkoitettu ennustamiseen ja jäännöskoodaukseen koko 16 x 16 lumalohkolle ja on tarkoitettu kuvan tasaisten ja ”pehmeiden” alueiden koodaamiseksi

Luma prosessoinnin lisäksi voidaan myös laatia vastaavat erilliset croma ennusteet. Aikaisemmissa standardeissa (kuten H.263+ ja MPEG-4/2) intraennusteet tehtiin muunnosalueessa kun H.264:ssä ennusteet tehdään spatiaalisessa alueessa. Tämä tehdään vertailemalla tutkittavaa jo koodattuihin naapurilohkoihin. Vertailtavat referenssilohkot ovat koodattavan vasemmalla tai yläpuolella. Viitteet tehdään jo lähetettyihin makrolohkojen (tai osalohkojen) reunoihin. Koska tällainen ennustaminen voi kuitenkin johtaa spatio temporaaliseen virheen leviämiseen, niin voidaan valita myös sellainen rajoitettu koodaustapa, jossa ennusteet tehdään intrakoodatuista naapuri lohkoista.



Kuva 6: Luminanssilohkojen spatiaalinen ennustaminen, jo koodattujen avulla, nuolet kuvaavat extrapolointisuuntia, keltaisella on merkitty ennustettava lohko. Itse asiassa ennusteet tehdään suuremmille alueille (makrolohkojen kokoisina), yksinkertaisuuden vuoksi tähän on vain piirretty alueet 4 x 4:n kokoisina.

Ennusteiden luomista havainnollistaa oheinen kuva 6 (mutta suuremmille alueille kuin kuvan esittämä 4 x 4). Kuvassa koodattavat pel:it on merkitty pienin kirjaimin (a – p). Lohkon vasemmalla ja yläpuolella olevat ennustuksessa käytettävät näytteet (A – M) muodostavat lähtökohdan ennusteen muodostamiselle. Moodissa 1 kaikki vertikaaliset pelit muodostetaan vastaavasta koodatusta siis a,e,i ja m saavat arvon A, b,f,j,n arvon B jne. Moodissa 2 korvataan pel:it a-p näytteiden A-L keskiarvolla. Moodista 3 ylöspäin muodostetaan ennusteet painotettuna keskiarvona. Siis jos esimerkiksi valitaan moodi 4 niin näytteet i ja n lasketaan pyöristämällä (I/4 + J/2 + K/4), moodissa 6 taasen e ja k asetetaan yhtä suureksi kuin (I/2+J/2). jne. Enkooderi voi valita parhaimman ennusteen [ITUT05].

Edellä kuvatun 4 x 4 luminanssi ennusteen lisäksi standardi mahdollistaa myös ennusteiden tekemisen koko 16 x 16 makrolohkosta (intra_16x16 moodi). Toimintatapoja tässä on 4 eli vertikaalinen, horisontaalinen, keskiarvo ja taso. Kolme ensin mainittua lasketaan samankaltaisesti kuin intra_4x4 moodi_0:ssa, moodi_1:ssä ja moodi_2:ssa. Taso-toimintatapa taasen on paikkariippuva lineaarinen kombinaatio, joka mallintaa ennustettavan lohkon tasona. 8 x 8 cromasignaaleille on myös (luonnollisesti) määritelty 4 intra_16 x 16:n kaltaista toimintamoodia.

3.3 Muunnosalueen kokonaisluku koodaus

Monet aikaisemmat standardit, kuten MPEG-1, MPEG-2 ja H.263 käyttävät muunnosalueen koodaukseen reaalilukuarvoista diskreetiä kosinimuunnosta (DCT) 8 x 8 alueille. H.264:ssä käytetään eräänlaista 4 x 4 pseudo-DCT muunnosta. Muunnos on kokonaislukumuunnos perinteiseen katkaistun reaalilukumuunnoksen sijasta. Syynä uudenlaiseen muunnoksen tarpeeseen oli monia. Yksi suuremmista oli se, että H.264:ssä on suuret vaatimukset spatiaalisen ennusteen jälkeisten jäännösten koodaukselle, residuaaliselle koodaukselle. Aikaisemmissa standardeissa jäännösten koodaminen saattoi aiheuttaa dekoodauksessa, kumulatiivisen laskentavirheen (driftIn) seurauksena suuriakin virheitä (poikkeamia dekoodatulle datalle enkooderin ja dekooderin välillä). Drift on seurausta siitä että käänteismuunnoksen suoritus ei ole

aivan täsmällinen prosessi eri implementaatioissa. Ensinnäkin se on tehty reaalilukumuunnoksena, jonka tarkkuus annettujen toleranssien puitteissa voi vaihdella ns. round-off error seurauksena. Toiseksi dekoodaus tehdään usein eri alustoilla kuin enkoodaus, jolloin kahden eri alustan välillä saattaa olla myös erilainen reaalilukumoottori. H.264:ssä käytetään erittäin laajasti ennusteita, jolloin drift voisi muodostua ratkaisevaksi laatua heikentäväksi tekijäksi. Esimerkiksi aikaisemmissa standardeissa ennustevirheen kasautuminen voi tapahtua kerran P kuvassa mutta H. 264:ssä useammin [MAV03]. Esimerkiksi I kuvan 4 x 4 alue voidaan muodostaa spatiaalisina ennusteina toisista lohkoista. Malvar et al. esittivät esimerkiksi, että koska CIF kuvan leveys on 88 kpl. 4 x 4 alueita, niin ennusteen drift voi jokaista riviä kohden kasvaa 88 kertaa dekoodattaessa yksi I kuvan rivi. [MALV03]. On siis selvää, että H.264:n muunnoksen pitäisi olla vapaa driftistä.

Toisaalta valitun arkkitehtuurin pitäisi olla nopea ja kompleksisuudeltaan mahdollisimman alhainen, jotta tämä voitaisiin toteuttaa helposti erilaisissa HW arkkitehtuureissa (kuten ASIC-piireissä, media prosessoreissa, DSP:ssä ja yleiskäyttöisessä CPU:ssa). Standardiehdotuksen yksi alkuperäisistä ongelmista oli se, että käänteismuunnokselta vaadittiin 32 bittinen kertolasku ja 32 bittinen muistin osoitus. Myöhemmin kiristettiin vaatimuksia 16 bittisiin arkkitehtuureihin, lisäksi oli toivottu matemaattisesti täsmällinen ratkaisu.

Kuten edellä tuli mainittua, niin H.264 käyttää laajasti hyödyksi ennusteita ennen varsinaista muunnosta. Käyttämällä 4 x 4 liikesegmentointia tai spatiaalisia ennusteita voidaan merkittävästi minimoida 4 x 4 alueiden välisiä korrelaatioita [MALV03]. Joten sangen luonteva valinta on käyttää 4 x 4 muunnoksia 8 x 8 muunnoksen sijasta. Muunnos tehdään kaksivaiheisesti: Ensimmäiseen vaiheen muunnostuloksista ryhmitellään naapurilohkon DC kertoimet 4 x 4 lohkoihin, joille tehdään uusi muunnos.

3.3.1 4x4 kokonaislukumuunnos

Suhteellisen helpon laskennan, nopeiden algoritmien ja läheinen approksimaatio (statistisesti optimaalisen) Karhunen-Loeve muunnoksen kanssa on diskreetti kosinimuunnos DCT ollut erittäin suosittu monissa video- ja still-kuva standardeissa kuten JPEG:ssa ja MPEG-1,2:ssa. Diskreetti kosini muunnos määritellään [HANH03, GONZ02] seuraavasti:

$$D(i,j) = \frac{1}{\sqrt{2N}} c_i c_j \sum_{m=0}^{M-1} \sum_{l=0}^{N-1} X(m,l) \cos\left[\left(m+\frac{1}{2}\right)\frac{i\pi}{N}\right] \cos\left[\left(l+\frac{1}{2}\right)\frac{i\pi}{N}\right]$$
$$c_u = \begin{cases} \frac{1}{\sqrt{2}}, kun\ u = 0 \\ 1, kun\ u > 0 \end{cases} \quad (1)$$

Kun kuvan koko M x N ja X(k,l) on pel:n arvo kuvan pisteessä k, l. Muunnos (1) on separoituva joten muunnos voidaan kirjoittaa matriisimuotoon $\mathbf{Y} = \mathbf{HXH}^T$ missä **X** on muunnettava kuva ja $\mathbf{H}^T$ on matriisin **H** transpoosi. Toisin sanoen kuva X kerrotaan ensin vasemmalta diskreetillä kosinimuunnoksella **H**, tämä muuntaa kuvan rivit, sarakkeet taasen muunnetaan kertomalla kosinimuunoksen transpoosilla. Matriisi H voidaan nyt kirjoittaa muotoon:

$$\mathbf{H}_{k,n} = H(k,n) = c_k \sqrt{\frac{2}{N}} \cos\left[\left(n+\frac{1}{2}\right)\frac{k\pi}{N}\right]$$

missä (2)

$$c_k = \begin{cases} \frac{1}{\sqrt{2}}, \text{kun } k = 0 \\ 1 \text{ muulloin} \end{cases}$$

Muunnos on ortogonaalinen eli käänteismatriisi on sama kuin sen transpoosi eli $\mathbf{x} = \mathbf{H}^{-1}\mathbf{y} = \mathbf{H}^{T}\mathbf{y}$.

Aikaisemmissa standardeissa (JPEG, MPEG-1, MPEG-2 jne.) muunnos kohdistettiin 8x8 alueeseen niin H.264:ssä muunnettava alue on kooltaan 4x4. Kun N on 4 voidaan kaavasta (2) helposti laskea, esimerkiksi:

$$H(0,1) = \frac{1}{\sqrt{2}}\sqrt{\frac{2}{4}}\cos\left[\left(1+\frac{1}{2}\right)\frac{0\pi}{4}\right] = \frac{1}{2} = \frac{1}{\sqrt{2}}\sqrt{\frac{2}{4}}\cos\left[\left(2+\frac{1}{2}\right)\frac{0\pi}{4}\right] = H(0,2) \quad jne.$$

$$H(1,1) = 1\sqrt{\frac{2}{4}}\cos\left[\left(1+\frac{1}{2}\right)\frac{1\pi}{4}\right] = \sqrt{\frac{1}{2}}\cos\left[\frac{3\pi}{8}\right] = \frac{\sqrt{2\left(\sqrt{2}+2\right)}}{4} = -H(3,3) \quad jne.$$

Muunnos **H** voidaan siis esittää muodossa,:

$$\mathbf{H} = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \tag{3}$$

missä

$$a = \frac{1}{2} = 0.5$$

$$b = \frac{1}{\sqrt{2}}\cos\left(\frac{\pi}{8}\right) = \frac{\sqrt{2\left(\sqrt{2}+2\right)}}{4} \approx 0.653281$$

$$c = \frac{1}{\sqrt{2}}\cos\left(\frac{3\pi}{8}\right) = \frac{\sqrt{2\left(2-\sqrt{2}\right)}}{4} \approx 0.27098$$

Valitettavasti muunnoskertoimet b ja c ovat irrationaalilukuja ja tietokoneessa äärellisellä laskentatarkkuudella tapahtuu ns. round-off error eli jos $\mathbf{y} = \mathbf{H}\,\mathbf{x}$ ja käänteismuunnos $\mathbf{u} =$ round $\{\mathbf{H}^{T}\mathbf{y}\}$ niin emme saa täsmällisesti u(n) = x(n) kaikilla n:llä, erityisesti silloin kun laskenta tehdään eri koneilla, joissa kummassakin on eri tapa tehdä reaalilukuaritmetiikkaa. Joten olisi toivottavaa korvata kertoimet kokonaislukukertoimilla. Standardin varhaisessa vaiheessa kertoimet a, b ja c kerrottiin 26:lla ja pyöristettiin lähimpään kokonaislukuun. Tuloksena saatiin:

$$a = \frac{26}{2} = 13$$

$$b = \frac{26}{\sqrt{2}}\cos\left(\frac{\pi}{8}\right) = \left\lfloor \frac{26\times\sqrt{2\left(\sqrt{2}+2\right)}}{4} \right\rfloor = 17 \approx 16.9853 \approx 26\times 0.653281$$

$$c = \frac{1}{\sqrt{2}}\cos\left(\frac{3\pi}{8}\right) = \left\lfloor \frac{26\times\sqrt{2\left(2-\sqrt{2}\right)}}{4} \right\rfloor = 7 \approx 7.0355 \approx 26\times 0.27098$$

missä $\lfloor \bullet \rfloor$ on pyöristys lähimpään kokonaislukuun. H on kohtalaisen lähellä skaalattua DCT:tä ja sillä on vielä sama normi eli $2\times 13^2 = 17^2 + 7^2$.

Tarkastellaan oheista lukuarvoista esimerkkiä:

$$\mathbf{x} = \begin{bmatrix} 5 \\ 9 \\ 1 \\ 19 \end{bmatrix}, \qquad \mathbf{X} = \begin{bmatrix} 5 & 11 & 8 & 10 \\ 9 & 8 & 4 & 12 \\ 1 & 10 & 11 & 4 \\ 19 & 6 & 15 & 7 \end{bmatrix}$$

kertolaskun tuloksena saadaan:

$$\mathbf{y} = \mathbf{H}\mathbf{x} = \begin{bmatrix} 442 \\ -182 \\ 182 \\ -234 \end{bmatrix}, \qquad \mathbf{Y} = \mathbf{H}\mathbf{X}\mathbf{H}^{\mathbf{T}} = \begin{bmatrix} 23660 & -52 & -1014 & 754 \\ -2236 & -3240 & 286 & -6086 \\ 3718 & 2054 & 1352 & 3172 \\ -2730 & -2030 & -6344 & -816 \end{bmatrix}$$

$$\frac{\mathbf{H}^T\mathbf{y}}{26^2} = \begin{bmatrix} 5 \\ 9 \\ 1 \\ 19 \end{bmatrix}, \qquad \mathbf{X} = \frac{\mathbf{H}^T\mathbf{Y}\mathbf{H}}{26^4} = \begin{bmatrix} 5 & 11 & 8 & 10 \\ 9 & 8 & 4 & 12 \\ 1 & 10 & 11 & 4 \\ 19 & 6 & 15 & 7 \end{bmatrix}$$

Valituilla parametreilla (a = 13, b = 17 ja c = 7) on vain ongelmana dynaamisen alueen kasvu eli luvuista tulee kooltaan liian suuria, kuten yo. esimerkistä nähdään. Siis jos yksiulotteisen muunnoksen suurin arvo max { |x(n)| } = A jollakin n.llä, niin muunnetulle on tämä kasvanut jo max { |y(n| } = 52 A. Eli dynaaminen alue kasvaa 52 kertaiseksi. Koska laskemme kaksiulotteisen muunnoksen niin dynaaminen alue kasvaa jo toiseen potenssiin eli 52^2= 2704. Koska $\log_2(2704)$ = 11.4009 tarvitaan 12 lisäbittiä tallentamaan **Y X**:n sijasta. Seurauksena on tarve käyttää 32 bittistä aritmetiikkaa. Tämän ongelman ratkaisemiseksi kertoimen 26 sijasta käytetään kerrointa 2.5 jolloin:

$$a = \left\lfloor \frac{2.5}{2} \right\rfloor = 1$$

$$b = \frac{1}{\sqrt{2}}\cos\left(\frac{\pi}{8}\right) = \left\lfloor \frac{2.5 \times \sqrt{2\left(\sqrt{2}+2\right)}}{4} \right\rfloor = 2$$

$$c = \frac{1}{\sqrt{2}}\cos\left(\frac{3\pi}{8}\right) = \left\lfloor \frac{2.5 \times \sqrt{2\left(2-\sqrt{2}\right)}}{4} \right\rfloor = 1$$

Joten vaadittava lisäbittien tarve on redusoitunut 6:een. Muunnosmatriisi voidaan kirjoitta muotoon [MALV03]:

$$\mathbf{H} = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 2 & 1 & -1 & -2 \\ 1 & -1 & -1 & 1 \\ 1 & -2 & 2 & -1 \end{bmatrix} \tag{4}$$

Matrisin (4) rivit ovat ortogonaalisia mutta näillä ei ole ikävä kyllä sama normi. Tämä voidaan kuitenkin kompensoida helposti kvanttisointiprosessissa. Koska matriisi on ortogonaalinen mutta riveillä ei ole sama normi voitaisiin käänteismuunnoksessa käyttää matriisin H transpoosia ja kompensoida eri normit. Asia nähdään helposti seuraavalla laskulla:

$$\mathbf{H}^T = \begin{bmatrix} 1 & 2 & 1 & 1 \\ 1 & 1 & -1 & -2 \\ 1 & -1 & -1 & 2 \\ 1 & -2 & 1 & -1 \end{bmatrix}, \qquad \mathbf{H}\mathbf{H}^T = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 10 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 10 \end{bmatrix}$$

Jotta yhdistetyn pyöristysvirheet tulisivat minimoitua pitäisi dynaaminen alue jälleen pienentää. Ongelma on niissä parittomissa symmetrisissä kantafunktioissa, jonka huippuarvo on 2. Siis H matriisin riveissä 2 ja 4. Nämä rivit kerrotaan ½:lla. Tällä tavalla saavutetaan dynaamisen alueen kasvu ja lisäbittien redusointi 6:sta 4:ään. Joten skaalaamista vaille oleva H:n käänteismatriisi on muotoa:

$$\tilde{\mathbf{H}}^{-1} = \begin{bmatrix} 1 & 1 & 1 & 1/2 \\ 1 & 1/2 & -1 & -1 \\ 1 & -1/2 & -1 & 1 \\ 1 & -1 & 1 & -1/2 \end{bmatrix} \tag{5}$$

matriisin (5) skaalauskertoimet saadaan helposti laskettua:

$$\tilde{\mathbf{H}}^{-1}\mathbf{H} = \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 5 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 5 \end{bmatrix} \tag{6}$$

joten

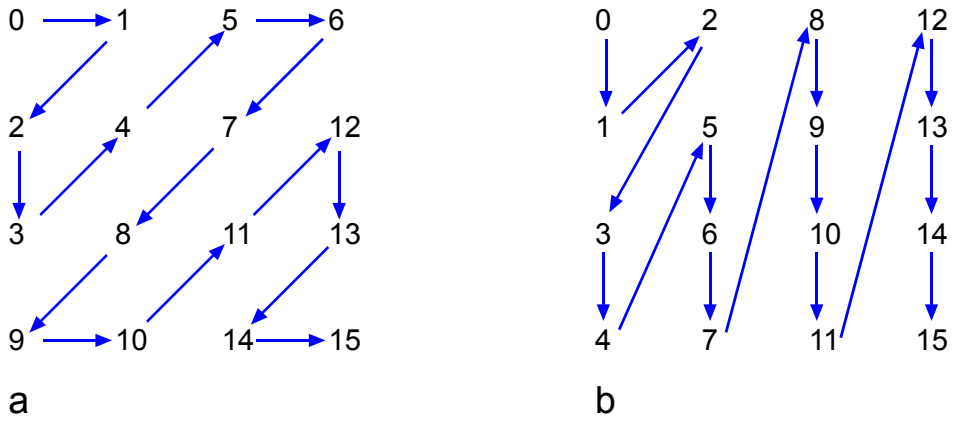
$$\tilde{\mathbf{H}}^{-1} diag\left\{\frac{1}{4},\frac{1}{5},\frac{1}{4},\frac{1}{5}\right\}\mathbf{H} = \mathbf{I} \quad (7)$$

Kuten pitääkin olla.

3.3.2 Zig-zag skannaus ja DC komponettien Hadarnmard muunnos

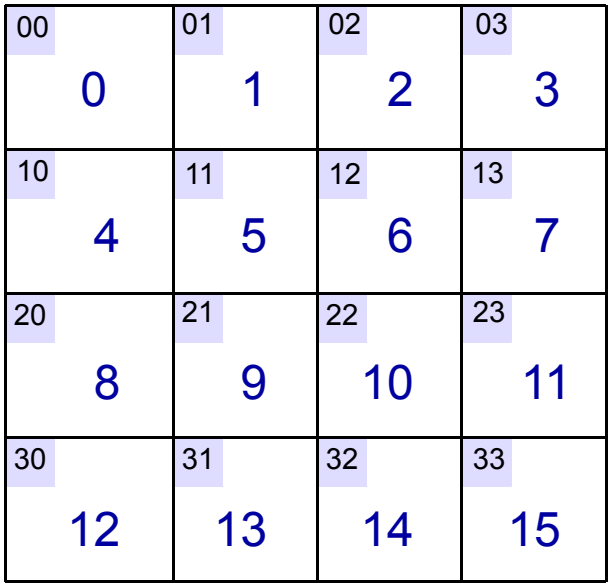
MPEG-2:ssa kuten myös JPEG:ssä tehdään 8x8 muunnokselle zig-zag skannaus, jolla järjestetään muunnoksen taajuuskomponentit tärkeysjärjestykseen. Skannauksen tulokset johdettiin kvanttisointiin ja VLC kooderille. MPEG-2:ssa oli myös määritelty ns. vaihtoehtoinen skannaustapa, joka oli erityisesti tarkoitettu lomitetulle videolle.

H.264:ssä on myös määritelty 4 x 4 muunnoksille oheisen kuvan 6 mukaiset 2 erilaista kertoimien läpikäyntitapaa:



Kuva 6: Muunneltujen lohkojen zig-zag skannaus

Kun makrolohkot muunnetaan 4 x 4 muunnoksella ei lomitetusta videosta niin seurataan kuvan 6 a) tapausta, jos taasen on kyse lomitetusta videosta niin kentät käydään läpi b) tapauksen mukaisesti. 2x2 croma komponentit (4:2:0 näytteenotossa) sekä FRExt laajennusten mukanaan tuomat muut lohkokoot käydään läpi samankaltaisesti.



Kuva 7 4 x 4 muunnoksen DC komponentit makrolohkon sisällä

Kun toimintamoodi on Intra_16x16 niin yksi makrolohko sisältää 16 luma osalohkoa kuvan 7 mukaisesti järjestettynä. Kunkin 4x4 muunnoksen DC komponentille tehdään vielä lisämuunnoksena seuraava Hadarmard muunnos:

$$f = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} & c_{02} & c_{03} \\ c_{10} & c_{11} & c_{12} & c_{13} \\ c_{20} & c_{21} & c_{22} & c_{23} \\ c_{30} & c_{31} & c_{32} & c_{33} \end{bmatrix} \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{bmatrix} \tag{8}$$

Missä c_{xy} on kunkin osalohkon DC komponnetti. Vastaava menettely tehdään myös croma komponenteille, 4:2:0 tapauksessa tämä muunnos on määritelty

$$f = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{bmatrix} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \tag{9}$$

3.4 Interkoodaus ja liikkeen estimointi

Kuten edellä on jo tullut ilmi, niin temporaalisesti ennustettuja viipaleita kutsutaan joko P tai B viipaleiksi. P viipaleessa ennusteet tehdään makrolohkolle tai tämän (tuonnempana selitettävälle) osajoukolle partitiolle. Ennuste voidaan laskea yhdestä tai useammasta menneisyydessä tai tulevaisuudessa esiintyvästä kuvasta. P viipaleessa liikevektoreita on vain yksi kun taasen B viipaleessa liikevektoreita on kaksi. Ennusteita laskettaessa tehdään sellainen oletus, että liikkeet ovat translationaalisia siirtymiä, joten rotaatioita ei oteta huomioon.

H.264:n ehkä merkittävin kompressiotehokkuudessa saavutettu säästö muodostuu tavasta miten liike-estimointi ja –kompensointi tehdään. Tehokkuuden lisäys liike-estimoinnin suhteen on seurausta siitä, että a)lohkorakenteiden koko voi vaihdella, b) voidaan käyttää sub-pel tarkkuutta liikevektoreiden laskennassa, c) voidaan hyödyntää moninkertaisia referenssi frameja, d) uudenlaisesta tavasta tehdä kaksisuuntaisia ennusteita B viipaleissa.

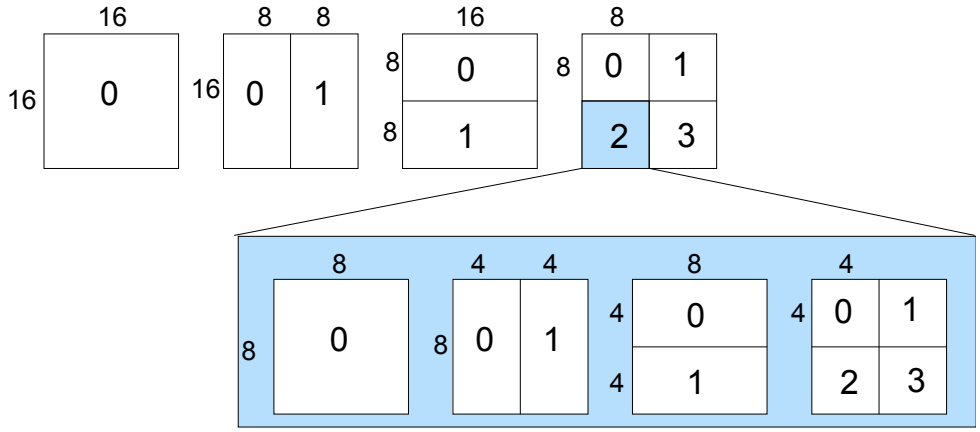
Tunnetaan useitakin menetelmiä muodostaa liikevektoreita. Näissä pyritään ns. lohkojen sovittamiseen, missä tietyn hakuikkunan puitteissa edeltävistä (tai seuraavista) kuvista sellainen lohko joka mahdollisimman hyvin vastaa tarkastelun alaista lohkoa [HANH03]. Tällaista paikkasiirtymää kutsutaan siis liikevektoriksi. Monissa aikaisemmissa standardeissa käytettiin kiinteää lohkokokoa haun pohjana (FBME fixed block motion estimation). Esimerkiksi MPEG-1, MPEG-2, H.261 ja H.263 käyttivät kiinteää 16 x 16 lohkokokoa.

Koko kompressioprosessin laskennallisesti työläin vaihe on liikevektoreiden hakeminen. Tämän vuoksi koko hakuprosessi määrätyn hakuikkunan puitteissa on saanut paljon tutkimuksellista mielenkiintoa osakseen. Sovituksen päätöskriteerit voivat perustua esimerkiksi pelien erotuksen neliöiden summan minimointiin (SSD Squared Sum Difference) tai keskivirheeseen (MSE Mean Square Error), absoluuttierojen summaan (SAD Sum of Difference) tai tämän keskiarvoon MAD (Mean Absolute Difference), normalisoituun ristikorrelaatioon tai muihin tilastollisessa analyysissa esiintyviin menetelmiin. Varsinaiseen hakuun voidaan soveltaa monia eri menetelmiä. Esimerkiksi kolmen askeleen haku (TSS three step search), parannettu kolmen askeleen haku (NTSS), neljän askeleen haku (FSS), timanttihaku (DS Diamand Search), kuusikulmainen haku (HBS Hexacon Based Search) sekä monia muita (yksityiskohtaisempi kuvaus löytyy esim. [HANH03]). Timanttihaku, DS, on erityisen sovelias kun liikkeet ovat pieniä. HBS on nopea menetelmä, jonka suorituskyky on lähellä DS:ää ja soveltuu hyvin videoihin, joissa on nopeaa liikettä. Näissä hakualgoritmeissa on pahana puolena, että ne helposti tulevat vangituiksi lokaalisiin minimeihin. Tämän ongelman ratkaisemiseksi on ehdotettu soveltavaksi mm. DS:n ja TSS:n yhdistämistä tai valitsemalla hakukeskukset vastakkaisten lohkojen liikevektoreista. Myös muunlaisia alustusmenetelmiä on sovellettu laajalti.

3.4.1 vaihtuva lohkokoko

Jo H.263 standardin parannettua versiota H.263+ standardoitaessa otettiin huomioon mahdollisuus käyttää pienempiä lohkokokoja kuin 16 x 16. Tämä siksi, että oli havaittu kompressiotehokkuuden kasvaneen huomattavasti, tosin laskentatehokkuuden kustannuksella. H.264:ssä korvattiin kiinteä lohkokoko mahdollisuudella käyttää 7:ää eri lohkokokoa sekä moninkertaisia viitekuvia ja –viipaleita. Oheiseen kuvaan 8 on hahmoteltu erilaisia tapoja jakaa ennusteviipaleiden lohkot pienempiin osiin.

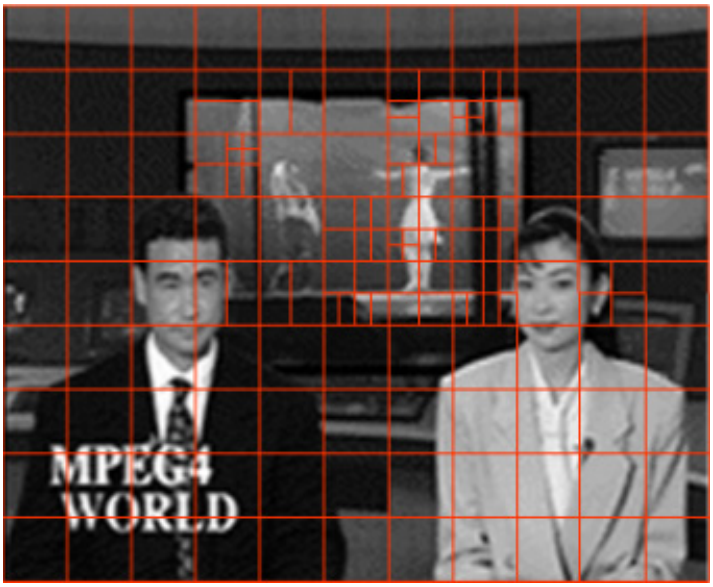
Tällaista tapaa jakaa makrolohkot liikekompensoituihin pienempiin osiin kutsutaan puumaiseksi liikekompensoinniksi (tree structured motion compensation). Jokaista jakoa kohden muodostetaan erillinen liikevektori. Esimerkiksi jakamalla alas 4 x 4 alilohkoihin on liikevektoreita jo 16 kpl yhtä makrolohkoa kohden. Jokainen liikevektori pitää koodata ja lähettää, lisäksi käytetty jaon tyyppi pitää enkoodata osaksi kompressoitua bittijonoa. Käyttämällä suuria lohkokokoja (esim. 16 x 16, 16 x 8 tai 8 x 16) niin suhteellisen pieni määrä bittejä tarvitaan kuvaamaan liikevektoreita ja valitun jaon tyyppiä Suurilla lohkokooilla liikekompensoidut jäännökset kuitenkin sisältävät runsaasti (visuaalista) “energiaa” paljon yksityiskohtia sisältävissä kuvan osissa. Nämä energiset osat vaativat suhteellisen suuren kaistan. Toisaalta valitsemalla pieni jakokoko (esimerkiksi 8 x 4, 4 x 4 jne) saadaan pienempi liikekompensoinnin jälkeinen virhejäännös mutta vastaavasti suurempi määrä bittejä joudutaan käyttämään liikevektoreihin ja valittuihin jakokokoihin. Siispä sopivan jaon valinnalla on merkittävä osuus kompression suorituskykyyn.



Kuva 8: Eri tapoja jakaa H.264:ssä 16 x 16 kokoinen lohko osalohkoihin liike-ennustamista varten. 8 x 8 lohkokoko voidaan vielä jakaa pienempiin alilohkoihin aina 4 x 4 kokokoon saakka.

Cromakomponentitkin voidaan jakaa samalla tavalla kuin lumakomponentit makrolohkon sisällä. Koska cromakomponenttien resoluutio on puolet lumasta (4:2:0 näytteenotolla) niin jaetut cromalohkot ovat myös pienempiä (16 x 16 lohkosta saadaan 8 x 8 cromalohko, 16 x 8:sta saadaan 8 x 4, 4 x 4 muuntuu 2 x 2:ksi jne).

Oheiseen kuvaan 9 on hahmoteltu eräs osajako. Kuvaa silmäiltäessä nähdään että kuvan spatiaalisesti ja temporaalisesti lähes homogeeniset osat, kuten tausta ja miehen puku kannattaa koodata 16 x 16 koossa. Naisen vaaleapuku sisältää suuria lokaalisen kontrastin alueita. Mutta kun nämä säilyvät suhteellisen pitkässä sekvenssissä stationäärisenä niin ne koodataan myös 16 x 16 moodissa. Taustan tanssijat ovat kohtalaisessa liikkeessä joten nämä kannattaa jo koodata pienemmällä lohkokoolla.



Kuva 9: Lumakanavan osalohkojako, kuva 176 x 144 ns. ”news” sekvenssi

Savuttaakseen maksimaalisen koodaustehokkuuden ja minimoidakseen tuloksena saatavat databittien lukumäärää, liikevektoreiden osalohkoihin jako pitäisi tehdä mahdollisimman optimaalisesti. Menettelyä kutsutaan nopeuspoikkeama optimonniksi RDO:ksi (Rate Distortion Optimization). Käytetty RDO voidaan lyhyesti kuvat seuraavasti [WU05]: Enkooderi kompressoi makrolohkon kaikilla mahdollisilla toimintamoodeilla,

kuten eri spatiaalislla ennusteilla, eri lohkokoolla liike-ennusteissa, moninkertaisilla referenssikuvilla jne. Se koodaustapa joko tuottaa pienimmän nopeuspoikkeaman valitaan lopulliseen koodaukseen. Kuitenkin nopeuspoikkeaman kustannukset voidaan saada vasta tietyn toimintasekvenssin jälkeen. Esimerkiksi interkoodauksessa kustannus lasketaan vasta estimoinnin ja liikekompensoinnin, kokonaislukumuunnoksen, kvanttisoinnin , käänteiskvanttisoinnin ja entropiakoodauksen jälkeen. Tällaisen RDO:n haittana on, kuten voidaan arvata, erittäin kallis laskennallinen hinta ja kompleksinen toteutus [WU05].

Toistaiseksi inter-toimintatavan RDO liike-estimointi tehdään siis soveltamalla kaikkia lohkokokoja, jotta löydettäisiin se millä on pienin RD kustannus. Eli kokeillaan kaikkia ja valitaan halvin. On selvää että tämä on lakennallisesti kallis ja kuluttaa huomattavasti muistia jo tällainen haku tehtäisiin puhtaasti raa'alla voimalla. Joten erilaiset optimointitekniikat olisivat tarpeellisia. Eräs näistä on ns. Lagrangen kertoimien hyödyntäminen [WU05, TU05]. Menettelytavassa pitää löytää lohko S_k kaikissa mahdollisissa toiminta moodeissa I_k minimoimalla Lagrangen kustannus:

$$J_{N_1\times N_2}(S_k, I_k \mid Q_P) = F_{N_1\times N_2}(S_{k,} I_{k|} \mid Q_P) + \lambda R(MVD_{N_1\times N_2}(S_{k,} I_{k|} \mid Q_P)) \tag{10}$$

missä s on tietty (aikaisempi) referenssilohko ja c tarkastelun alainen lohko, N_1, N_2 on käytetty lohkokoko (joka voi siis olla 16 x 16, 16 x 8 jne.) ja Q_P tietty kvanttisointiparametri, Wu & al ehdottivat SSD:n käyttämistä $F_{N_1\times N_2}(S_{k,} I_{k|} \mid Q_P)$:ssä kun taas Tu & al ehdottivat SAD:ta eli:

$$\begin{aligned} SSD_{N_1\times N_2}(s,c) &= \sum_{i=0}^{N_1}\sum_{j=0}^{N_2}\left[s(i,j,t) - c(i,j,t)\right]^2 \\ SAD_{N_1\times N_2}(s,c) &= \sum_{i=0}^{N_1}\sum_{j=0}^{N_2}\left| s(i,j,t) - c(i,j,t)\right| \end{aligned} \tag{11}$$

missä $s(i,j,t)$ ja $c(i,j,t)$ ovat tietyt pel:ien arvot pisteessä i,j lohkoissa c ja s. Lopulta λ_M on tiettyyn moodiin liittyvä Lagrangen kerroin, funktio $R(MVD_{N_1\times N_2}(s,c))$ taasen kuvaa tiettyyn moodiin ja kvanttisointiparametriin liittyvää bittien lukumäärää, eri liike-vektoreiden erotukseen.

Lagrangen kertoimelle λ_M on määritelty tietyllä kvanttisointiparametrilla Q_P:

$$\lambda_M = \begin{cases} 0.85\times 2^{Q_P/3}, & \text{P kuville} \\ \max\left(2, \min\left(4, \frac{Q_P}{6}\right)\right)\times 0.85\times 2^{Q_P/3}, & \text{B kuville} \end{cases} \tag{12}$$

3.4.2 Subpel liikevektorit

Merkittävä kompressiotehokkuuden kasvu saavutetaan H.264:ssä käyttämällä liikevektoreille tarkempia ennusteita kokonaislukunäytteiden sijasta puoli ja neljännes positioita. Tämä tarkoittaa sitä, että lumakanavan referenssimakrolohkon ja koodattavan lohkon siirtymän laskenta on pienimmillään ¼ pel:n päässä toisistaan. Erään H.264

implementaation (UB Video) suunnittelijat väittävät säästön olevan jopa 20 % siirryttäessä kokonaislukupositioista ¼ positiotarkkuuteen [UBIV02].

Jos liikevektori osoittaa suoraan kokonaislukupositioon niin ennusteena käytetään vastin pel:jä referenssikuvassa. Muussa tapauksessa käytetään välipisteitä. Puolipel vastinpisteet lasketaan yksiulotteisella FIR suodattimella[5] horisontaaliseen ja vertikaaliseen suuntaan. Neljännespelin tarkkuus generoidaan laskemalla keskiarvo kokonaisluku- ja puolikaspositioiden välillä. Oheiseen kuvaan 6 on hahmotettu suodattimen toimintaperiaate välipositioissa oleville näytteille a – k ja n – r. Esimerkiksi näytteet puolikas positioissa b ja h saadaan laskemalla aluksi välimuuttujat b_1 ja h_1 seuraavasti [WIEG03a]:

$$b_1 = (E - 5F + 20G + 20H - 5I + J)$$
$$h_1 = (A - 5C + 20G + 20M - 5R + T)$$

Lopullinen ennuste kohdassa b ja h saadaan seuraavasti ja leikataan välille 0 – 255 (laskutoimitus $\gg k$ on vasemmalle siirto k bittiä eli jakolasku jonka nimittäjänä on 2^k)

$$b = (b_1 + 16) \gg 5$$
$$h = (h_1 + 16) \gg 5$$

Vertikaalisti ja horisontaalisti puolivälissä olevan näytteen j arvo saadaan vastaavasti:

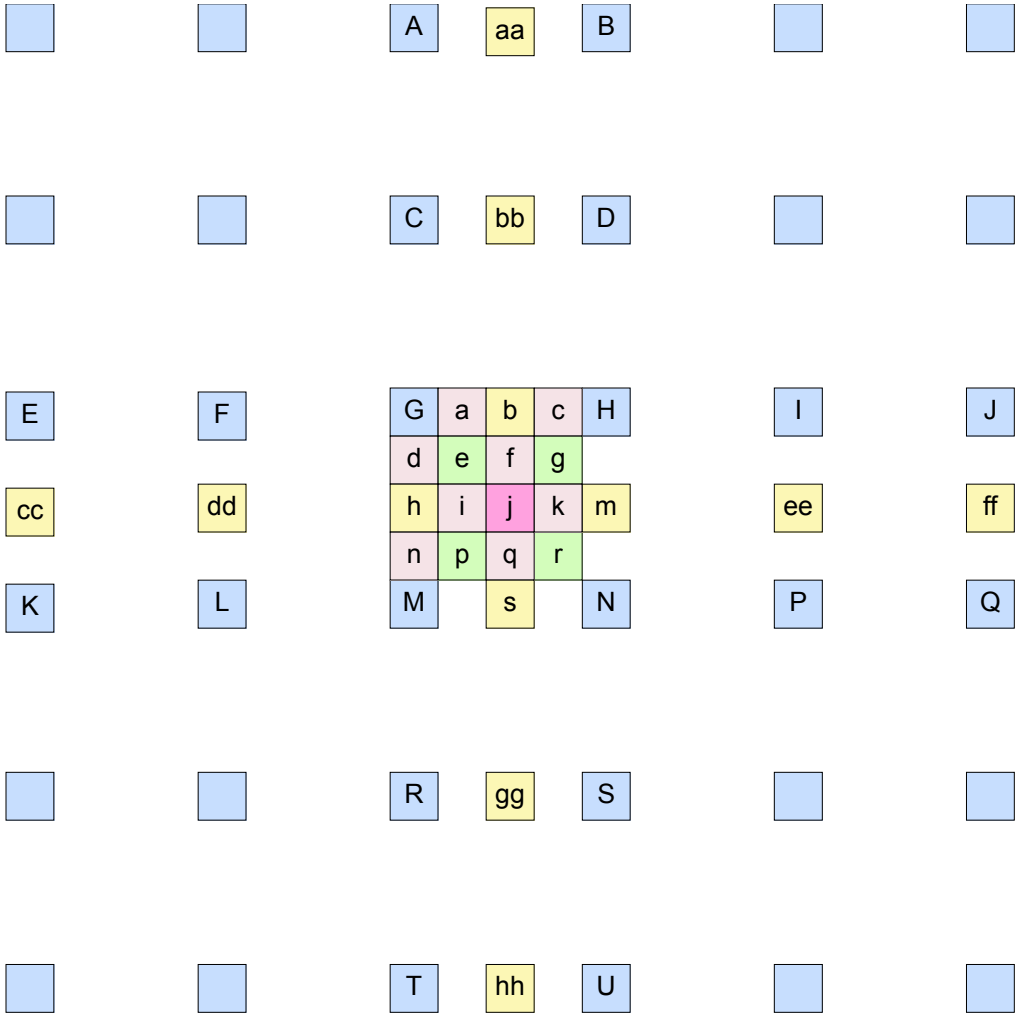
$$j_1 = cc - 5dd + 20h_1 + 20\, m_1 - 5ee + ff$$

missä väliarvot cc,dd, ee, m_1 ja ff on laskettu kuten h_1. Lopullinen arvo j lasketaan

$$j = (j1 + 512) \gg 10$$

ja leikataan välille 0 – 255.

[5] Suodin on ns. 6 tapin suodin, jonka parametrit ovat (1, -5, 20, 20, -5, 1).



Kuva 10: ¼ pel:in liikevektoreiden muodostus

Neljännespositioissa olevat näytteet a, c, d, n, f, i, k ja q lasketaan ylöspäin pyöristämällä kaksi lähintä näytettä, jotka ovat siis kokonaisluku ja puolipositioissa [WIEG03a]. Siis esimerkiksi:

$$a = (G + b + 1) \gg 1$$

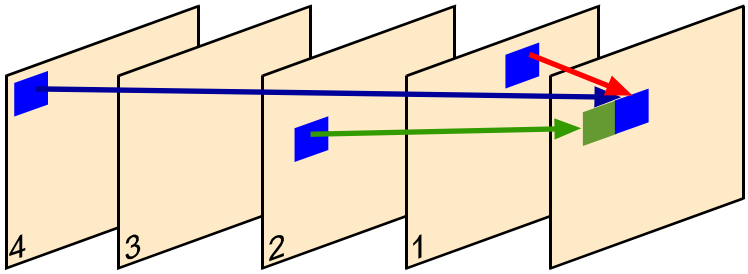
Vastaavasti neljännes positioissa e, g, p ja r lasketaan ylöspäin pyöristettynä keskiarvona diagonaalisuuntaan kummastakin puoliposition arvosta. Esimerkiksi e:n arvoksi saadaan laskemalla:

$$e = (b + h + 1) \gg 1$$

Jos luma- ja cromanäytteet on otettu suhteessa 2:1, niin ¼ luma pel:in liikevektorin erotuskyky merkitsee cromalle 1/8 pel:n tarkkuutta. Ennustearvo cromalle lasketaan aina bi-lineaarisella interpolaatiolla [WIEG03b, RICH04].

3.4.3 Moninkertaiset referenssiframet

Kuvien väliseen intrakoodaukseen H.264:ssä on mahdollista käyttää useampaa referenssiframea, ankkurikuvaa tai viipaletta. Näitä voi olla parhaimmillaan 5 aikaisempaa tai tulevaa. Näin voidaan saavuttaa monissa tapauksissa parempi subjektiivisesti koettu laatu sekä tehokkaampi videon koodaus (n. 5 - 10 % nopeusreduktio). Oheinen kuva 11 hahmottaa moninkertaisten ankkurikuvien käytön P viipaleiden ennustamisessa.



Kuva 11: Moninkertaiset referenssiframet P ennusteille.

Jotta moninkertaisia kuvia voitaisiin käyttää, pitää sekä enkooderissa että dekooderissa olla monikuvainen kuvamuisti (decoded picture buffer DPB). Monikuvaennusteissa pitää varsinaisen liikevektorin lisäksi lähettää tieto ankkurilohkon sijainnista kuvamuistissa. DPB:n viiteindeksit siis liittyvät kuhunkin liikekompensoituun 16 x 16, 16 x 8, 8 x 16 tai 8 x 8 lumalohkoon. Liikekompensoinnissa 8 x 8 pienemmille alueille käytetään samaa referenssi-indeksiä kuin edeltävälle 8 x 8 lohkolle.

H.264 sisältää P viipaleiden makrolohkoille myös ns. SKIP moodin. Tässä toimintatavassa ei lähetetä kvanttisoitua erosignaalia eikä liikevektoria liittyvine referenssi-indekseineen kuvamuistissa. Suurin etu P_Skip toimintatavassa on siinä että vain muutamalla bitillä voidaan kuvata suuria muuttumattomia alueita, koko viipale säilyy samana kuvasta toiseen.

3.4.4 Interkuva ennuste B viipaleissa

Aikaisemmissa standardeissa muodostettiin B kuvat menneestä ja tulevasta referenssinä toimivista kuvista (yhden GOP:n puitteissa). Ennuste muodostetaan eteenpäin ja taaksepäin suuntautuneista ennustesignaalien lineaarisesta kombinaatiosta. Tämä kombinaatio useimmiten muodostetaan liikekompensaatiolla [FLIE03]. H.264:ssä on laajennettu tällaista näkemystä siten että referenssikuvat voivat sijaita sekvenssissä muuallakin. Ensinnäkin kuten edellä todettiin referenssikuvia voi olla useampi ja toiseksi niiden ajallinen järjestys voi olla toisenlainen. On nimittäin havaittu, että myös tuleva/tuleva (forward/forward) ja mennyt/mennyt (backward/backward) kombinaatiot voivat muodostaa erittäin tehokkaat ennustesignaalit mennyt/tuleva kombinaation lisäksi.

Ehkä merkittävin P ja B viipaleiden välinen liikekompensoinnin kannalta nähty ero on siinä, että B viipaleissa voidaan käyttää painotettua keskiarvoa kahdesta liikekompensoidusta ennusteesta. Tähän B viipaleiden ennustamiseen käytetään kahta

erillistä kuvapuskuria, listaa, referoiduista kuvista list 0 ja list1[6]. Mitkä framet aktuaalisti esiintyvät näissä puskureissa, riippuu puskurien ohjausjärjestelmästä. Tällä mekanismilla voidaan myös luoda MPEG-2:n kaltaisia B-frameja.

B viipaleita voidaan muodostaa 4 eri tavalla [WIEG03, FLIE03]:

- list 0, ennusteet muodostetaan yksinomaan käyttämällä list0:n kuvia referenssinä.
- list 1, jossa hyödynnetään puskuria list 1
- bi-predictive, jossa ennustesignaali muodostetaan painotettuna keskiarvona list 0:n ja list 1:n ennustesignaaleista
- direct toimintatapa johdetaan aikaisemmin lähetetysitä syntaksielementeista ja voi olla peräisin joko list0:sta, list1:stä tai bi-predictive

3.5 Deblock suodatus

Eräs pahimmista subjektiivisesti koetuista vioista, mitä korkeasti kompressoidussa videossa voi esiintyä, on lohkoistuminen, ns. pikselöinti. Ilmiö näkyy lohkorajoissa lokaalin kontrastin suurina gradientteina, hyppäyksinä lohkosta toiseen. Kuva on ikään kuin rakentunut mosaiikeista. H.264:ssä käytetään rekonstruktiosuodatinta ns. de-blocking suodinta ilmiön minimoimiseksi. De-blocking suodinta sovelletaan H.264:n enkooderissa käänteismuunnoksen jälkeen ennen tallentamista kuvamuistiin ja tulevia ennusteita. Dekooderissa taasen suodin esiintyy ennen kuvan rekonstruointia ja makrolohkon näyttämistä. Suodattimella on kaksi tehtävää [RICH03]:

- Lohkojen reunat tulevat pehmennettyä, jolloin lohkon subjektiivisesti mielletty laatu paranee.
- Suodatettua makrolohkoa käytetään tulevien framien liikekompensoituun ennustamiseen, jolloin tuloksena pienemmät ennusteen jälkeiset jäännökset (kompressiotehokkuus paranee).

3.6 Kvanttisoija

Kvanttisoija on häviöllisessä kompressiossa yksi keskeisimpiä vaiheita signaalin häviön muodostamisessa. H.264:ssä käytetään skalaarikvanttisoijaa . Annetulla askelpituudella Q_s (yleensä kokonaisluku) tällainen kvanttisoija on muotoa [MALV03]:

$$X_q(i,j) = sign\left\{\mathbf{Y}(i,j)\right\}\frac{|\mathbf{Y}(i,j)| + f(Q_s)}{Q_s} \tag{13}$$

Missä i ja j ovat rivi ja sarake indeksit sekä $f(Q_s)$ ohjaa kvanttisointia lähellä 0:aa (ns. kuollut alue). Dekooderi suorittaa käänteiskvanttisoinnin yksinkertaisesti skaalamalla vastaanotetun datan kertoimella Q_s:

$$Y_r(i,j) = Q_s Y_q(i,j) \tag{14}$$

Varhaisessa H.264:n suunnitteluvaiheessa oli haluttu ominaisuudeksi mahdollisuus vaihdella kvanttisoinnin askelpituutta. Kvanttisoinnin askelpituus kasvaa likimäärin 12 %

[6] Standardin varhaisessa vaiheessa näistä käytettiin nimeä ns. forward ja backward reference set. Nimitys on sikäli harhaanjohtava, että näistä saa helposti virheellisen mielikuvan referenssikuvien ajallisesta järjestyksestä joten nimitys list 0 ja list 1 puoltavat paikkansa.

jokaista kvanttisointiparametrin kasvua kohden, siten että jokainen kuuden suuruinen kasvu parametrissa kaksinkertaistaa kvanttisoinnin askelpituuden [MALV03]. Näin mahdollistetaan hyvin laaja kokoelma erilaisia laatutasoja. Kaavan (13) suurimpana haittapuolena on se, että tämä vaatii jakolaskun. Tämän välttämiseksi alkuperäisessä standardiehdotuksessa (13) on korvattu seuraavalla kaavalla:

$$\begin{aligned} X_q(i,j) &= sign\{\mathbf{X}(i,j)\}\left[\left(|\,\mathbf{Y}(i,j)\,|\,A(Q) + f\,2^L\right) \gg L\right] \\ X_r(i,j) &= X_q(i,j)B(Q) \\ x_r &= \left(\mathbf{H}^T X_r + 2^{N-1}\left[1\ 1\ 1\ 1\right]^T\right) \gg N \end{aligned} \tag{15}$$

Tässä laskutoimitus $\gg k$ on k:n bitin siirto oikealla ja vastaa jakolaskua 2^k:lla parametri Q saa arvot 0:sta Q_{max} saakka. Kvanttisoinnissa ja käänteiskvanttisoinnissa käytettävät parametrit (taulukot) A(Q) ja B(Q) ovat suunniteltu siten, että Q:n arvolla 0 saavutetaan hienoin kvanttisointi ja arvolla Q_{max} karkein kvanttisointi. Parametrit L ja N valittiin kompromissina siten, että suuret arvot vähentävät approksimointivirhettä kun taas pienet arvot pienentävät Y_r:n ja $|\,\mathbf{Y}(i,j)\,|\,A(Q)$:n dynaamista aluetta [MALV03].

Ulkoasustaan huolimatta kaava (15) on suhteellisen yksinkertainen. Se voidaan kuitenkin vielä yksinkertaistaa ja sovittaa paremmin 16 bittiseen arkkitehtuuriin. Tavoitteen saavuttamiseksi redusoidaan B(Q):n arvoja sekä parametreja L ja N.

Laatimalla B(Q) sellaiseksi että se kasvaa kaksinkertaiseksi jokaista Q:n 6:n kasvua voidaan edelleen pienentää kvanttisointi ja reproduktio taulukoita Näiden syiden vuoksi (15) oli ehdotettu korvattavaksi seuraavalla kaavalla [MALV03]:

$$X_q(i,j) = sign\{\mathbf{Y}(i,j)\}\left[\left(|\,\mathbf{Y}(i,j)\,|\,A(Q_M,i,j) + f\,2^{17+Q_E}\right) \gg (17+Q_E)\right]$$

missä

$$Q_M \equiv Q \bmod 6$$

$$Q_E \equiv \frac{Q}{6} \tag{16}$$

Kaavan yksi suuri etu on siinä että muistin määrää ei tarvitse lisätä laajalla dynaamisella alueellakaan. Ominaisuus on seurausta siitä, että yhden askeleen muutos Q_E:ssä myös kaksinkertaistaa nimittäjän. Normaalisti enkooderi valitsee parametrin f alueelta 0 – ½. Samoin rekonstruktiolle on voimassa kaava:

$$X_r(i,j) = X_q(i,j)B\left(Q_M,i,j\right) \ll Q_E \tag{17}$$

Kaavoissa (16) ja (17) tekijät $A(Q_M,i,j)$ ja $B\left(Q_M,i,j\right)$ riippuvat muunnoskertoimista lohkon sisäisistä positioissa (i,j). Näin saadaan kompensoitua eri skaalaustekijät kokonaislukumuunnoksessa **H** (4). Nämä tekijät on määritelty seuraavasti:

$$A(Q_M, i, j) = M(Q_M, r)$$
$$B(Q_M, i, j) = S(Q_M, r)$$
$$r = \begin{cases} 0, (i,j) = \{(0,0),(0,2),(2,0),(2,2)\} \\ 1, (i,j) = \{(1,1),(1,3),(3,1),(3,3)\} \\ 2, \text{muulloin} \end{cases}$$

Lopulta taulukot S ja M ovat määritelty 6 x 3 taulukkona [MALV03]:

$$M = \begin{bmatrix} 13107 & 5243 & 8066 \\ 11916 & 4660 & 7490 \\ 10082 & 4194 & 6554 \\ 9362 & 3647 & 5825 \\ 8192 & 3355 & 5243 \\ 7282 & 2893 & 4559 \end{bmatrix}, \quad S = \begin{bmatrix} 10 & 16 & 13 \\ 11 & 18 & 14 \\ 13 & 20 & 16 \\ 14 & 23 & 18 \\ 16 & 25 & 20 \\ 18 & 29 & 23 \end{bmatrix}$$

Rekonstruktion jälkeen lopullinen skaalaus tulee olemaan [ITUT05]:

$$x_r = \left(\tilde{H}_{inv} X_r + 2^5 [1\,1\,1\,1\,1]^5 \right) \gg 6 \tag{18}$$

Edellä olevasta kuvauksesta nähdään että muunnos ja kvanttisointi prosessit voidaan laskea 16 bittisellä kokonaislukuaritmetiikalla kun syötteen dynaaminen alue on 9 bittiä. Tämä 9 bittisyys johtuu siitä että 8-bittisella datalla ennusteiden jäännökset ovat dynaamiselta alueeltaan 9 bittiä. Poikkeuksena tästä on kaavassa (16) tekijä $|\mathbf{Y}(i,j)|\,A(Q_M,i,j)$ jonka dynaaminen alue on 32 bittiä, mutta (16):n lopullinen ulos on kuitenkin 16 bitin sisällä. Standardin uusimmissa versioissa sallitaan myös 10 ja 12 bittinen data, jolloin vastaavasti laskennan dynaaminen alue kasvaa.

3.7 Entropiakoodaus

Entropiakoodauksella tarkoitetaan häviötöntä koodaustapaa, jolla korvataan aktuaalinen data koodatulla versiolla, pyritään vähentämään koodisanoihin sisältyvää redundanssia. Koodattu versio voi siis merkittävästi pienentää lähettävän datan määrää. H.264:ssä voidaan käyttää useampaa eri entropiakoodaustapaa.: ns. Golomb koodeja, sisältöön adaptoituvaa muuttuvamittaista pituuskoodausta (CAVLC Context Adaptive Variable Length Coding) tai sisältöön adaptoituvaa aritmeettista koodia (CABAC Context Adaptive Binary Arithmetic Coding). Näitä koodeja sovelletaan bittijonon ns. syntaktisiin elementteihin. Näitä elementtejä ovat mm.:

- Ylemmän tason syntaktiset elementit, joita ovat esimerkiksi sekvenssin, kuvan ja viipaleiden otsakkeet. Näihin voidaan soveltaa joko kiinteän mittaista koodausta tai yleisempää VLC (UVLC) koodausta.
- Viipaleiden hyötydata kuten viipaleen pituus ja makrolohkojen ”skip” indikaattorit (joihin sovelletaan CAVLC ja CABAC koodausta).
- Makrolohkon tyyppi

- Makrolohkon koodattu lohkorakenne, joka ilmaisee mikä joukko 4 x 4 (tai 8 x 8) muunnoslohkoista on nollasta poikkeavia kertoimia.
- Makrolohkon kvanttisointiparametrit,, koodattuna erotuksena edellisestä makrolohkosta.
- Referenssikuva indeksit (kuvamuistissa).
- Liikevektorit MV, erotuksena naapuridatasta muodostetuista liikevektoriennusteista.
- Kvanttisoidut muunnoskertoimet joko 8 x 8 tai 4 x 4 muunnoksesta tai Intra_16x16 muunnoksessa käytettävien DC komponenttien Hadamard muunnoksen tuloksista.

3.7.1 Golomb koodi

Yksinkertaisin H.264:ssä käytetyistä entropiakoodaustavoista on exponentiaalinen Golomb koodi. Koodia käytetään kaikille generoitavan bittijonon syntaktisille elementeille paitsi kvanttisoiduille muunnoskertoimille suoraan. Niinpä, sen sijaan että suunniteltaisiin jokaiselle yksittäiselle syntaksielementille omat VLC taulukko riittää määritellä kuvaus yhdelle koodisanataulukolle. Kuvaus tälle taulukolle on tehty datan tilastollisen jakauman perusteella. Käytetty exponentiaalinen Golomb (Exp-Golomb) koodi noudattaa oheisen taulukon mukaista rakennetta [PURI04]:

rakenne	Bittijono	codeNum	Syntaksielementin arvo
1	1	0	0
$01x_0$	010	1	1
	011	2	-1
$001x_1x_0$	00100	3	2
	00101	4	-2
	00110	5	3
	00111	6	-3
$0001x_2x_1x_0$	0001000	7	4
	0001001	8	-4
…	…	…	…

Exp-Colomb koodit seuraavat siis taulukon mukaista rakennetta, joka muodostuu etuliitteestä (1, 01, 001, …) ja jälkiliitteestä (x_0, x_1x_0, $x_2x_1x_0$, …), missä x_i on joko 0 tai 1. Jonojen x_i kaikkien mahdollisten kombinaatioiden lukumäärä kuvastaa niiden koodien lukumäärää tietylle etuliitekoodille. Jokaiseen koodattavaan bittijonon (syntaktiseen) elementtiin liitetään tyyppi, joka kertoo miten data liitetään koodiin. Näitä tyyppejä ovat etumerkitön eksponenttiaalinen ue(v), etumerkillinen exponentiaalinen se(v), mapattu exponentiaalinen me(v) tai katkaistu eksponenttialinen te(v) tyyppi. Yllä olevassa taulukossa on esitetty miten koodisanat on liitetty ue(v) koodeihin. Lopulta taulukon viimeinen sarake kuvaa miten syntaksielementin etumerkillistä arvo voida käyttää koodisanojen muodostamiseen.

3.7.2 CAVLC

MPEG-2:ssa käytettiin entropiakooderina vaihtuvanmittaista pituussuuntaista koodia (VLC). Hyvin pitkälle se on klassinen Huffman tyyppinen koodi, jossa todennäköisimmille koodisanoille oli varattu lyhyemmät koodisanat ja vähemmän todennäköisille pidemmät. MPEG-2:ssa ja monissa muissa standardeissa zig-zag skannatut muunnoskertoimet kuvataan aluksi yksiulotteiselle taulukolle ja sitten koodataan käyttämällä pituussuuntaista ja vaihtuvanmittaista koodia. H.264:ssä muunnetaan hieman tätä lähestymistapaa. Tässä kvanttisoitujen muunnoksessa syntyneiden kertoimien koodaamiseen käytetään sisältöön adaptoituvaa muuttuvanmittaista pituuskoodia CAVLC (Context Adaptive Variable Length Code). Sen sijaan että pidättäydyttäisiin yhdessä VLC taulussa, H.264 CAVLC menettelyssä hyödynnetään symbolien välistä redundanssia valitsemalla koodattavalle symbolielementille sopiva VLC taulu useammasta taulusta (4 kpl). Valinta tehdään, sen mukaan mikä symboli on jo lähetetty, aikaisemmasta lohkosta [BJØN02].

Valitettavasti CAVLC ja yleisemmässä VCL menettelyssä on kuitenkin ongelmana se, että menettely ei täysin pysty mukautumaan todelliseen ehdolliseen symboli-statistiikkaan. Symbolitodennäköisyydet, jotka ovat suurempia kuin 0.5 ei pystytä täysin tehokkaasti esittämään. Syynä tähän on se, että VLC:n alaraja on 1 bit/symboli. Seurauksena onkin se, että koodausta ei voida aina käyttää pienemmällä aakkostolla. Tämän vuoksi käytettyä aakkostoa joudutaan laajentamaan eli lisäämään ”run”-symboleille peräkkäisiä 0 arvoisia tasoja. Toisaalta, koska eri syntaktisia elementtejä sisältävät VLC taulut valitaan aikaisemmin koodattujen syntaktisten elementtien pohjalta, niin kasvanut mukautuminen mahdollistaa myös tehokkaamman koodauksen verrattuna tapaukseen missä käytetään vain yhtä taulua. CAVLC koodausta käytetään vain muunnoksen tuloksena syntyneiden kvanttisoitujen kertoimien entropiakoodaukseen.

Muunnoskertoimien zig-zag skannauksen jälkeen muunnoskertoimien statistisessa jakaumassa pienitaajuisimmat kertoimet ovat melko suuria pienentyen edettäessä suurempitaajuisiin spatiaalisiin komponetteihin. Siis esimerkiksi 4 x 4 lumalohkon kvanttisoitudut muunnoskertoimet voisivat olla esimerkiksi:

$$\begin{bmatrix} 7 & 6 & 0 & 0 \\ -2 & -1 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

lohkon zig-zag skannaus tulos tuottaa seuraavan sekvenssin (DC:stä suurempiin taajuuskomponentteihin):

7, 6, -2, 0, -1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,0

Datasta lasketaan seuraavat suureet entropiakoodauksen yhteydessä:

1. Nollasta poikkeavien kertoimien lukumäärä (totalCoeff) esimerkissä totalCoeff = 5, näitä kertoimia voi olla siis yhteensä 0 – 16 kpl.
2. ”trailing” 1 määrä (T1), kertoo niiden itseisarvoltaan 1:n määrä skannauksen lopussa, em. esimerkissä tämä määrä on 2. T1 lukumäärä voi olla välillä 0 - 3,

jos ±1 arvoja on enemmän kuin 3 niin vain 3 viimeistä pidetään ”erityistapauksina”, trailing 1:nä ja loput koodataan normaalisti.

3. Jäljelle jäänneiden nollasta poikkeavien kertoimien suuruuden ja etumerkin koodaus (level). T1:ssä tarvitaan vain etumerkkitieto, koska arvot ovat joko +1 tai -1, koodina 0 tai 1. Yleensä kertoimet ovat vähemmän levinneet viimeisen nollasta poikkeavan kertoimen jälkeen kuin ennen tätä. Tämän vuoksi kertoimet koodataan päinvastaisessa järjestyksessä kuin skannattaessa. Esimerkissä -2 on ensimmäinen koodattavan kertoimen arvo. Aloittavaa VLC:tä käytetään tähän. Kun koodataan seuraavaa kerrointa (esimerkin kerroin 6) voidaan käyttää uutta VLC:tä, joka perustuu juuri koodattuun kertoimeen. Tällä tavalla adaptoituminen tapahtuu käyttämällä useampaa VLC taulukkoa. Tähän mukauttamiseen on H.264:ssä varattu 6 erilaista exp-Golomb taulua.
4. Etumerkkitieto. Kertoimien etumerkin esittämiseen tarvitaan 1 bitti. T1:ssä tieto lähetetään yksittäisenä bittinä. Muilla kertoimilla etumerkki sisältyy exp-Golomb koodiin.

Jokaisen nollasta poikkeavan kertoimen sijainti koodataan määrittelemällä 0:n sijainti ennen viimeistä nollasta poikkeavaa kerrointa. Tämä voidaan jakaa kahteen osaan:

5. Nollien lukumäärä, TotalZeroes, koodisana määrittelee nollien lukumäärän viimeisen nollasta poikkeavan kertoimen ja skannauksen alun välillä. Em. esimerkissä tämä on 3. Tähän tarkoitukseen on määritelty 15 taulukkoa N:n arvoille 1 – 15.
6. ”runBefore” Edellä mainitussa esimerkissä pitää määritellä miten 3 nollaa ovat jakautuneet. Ensin 0:n lukumäärä ennen viimeistä kerrointa pitää koodata. Esimerkissä tämä on 2. Koska sen pitää olla lukuarvo välillä 0 – 3 käytetään sopivaa taulukkoa. Nyt on yksi 0 vielä jäljellä. Nollien lukumäärä ennen toiseksi viimeistä kerrointa pitääkin olla joko 0 tai 1. Esimerkin tapauksessa se on 1. Jäljellä ei ole yhtään 0:aa jäljellä eikä muuta informaatiota pidä koodata.

Koodin muodostaminen etenee siis seuraavasti (olettaen, että lohko on ensimmäinen):

Elementti	Arvo	koodi
coeff_token	totalCoeff = 5, T1 = 2	0000 0010 1
T1 etumerkki(1)	+	0
T1 etumerkki (0)	-	1
Level (2)	-2⇒-1[7]	01
Level(1)	6	00010
Level(0)	7	000100
TotalZeroes	3	111
runBefore(1)	zerosLeft=3, run_before=2	01
runBefore(0)	zerosLeft=1, run_before=1	Ei koodia viimeinen
Lopullinen koodi	0 0000 0101 0101 0001 0000 1001 1101	
(heksalukuna)	0055109C	

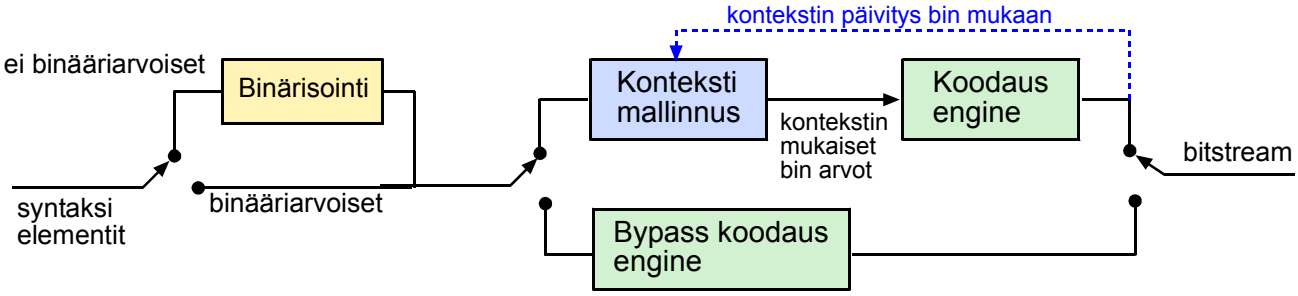
[7] Jos T1 on pienempi kuin 3 niin ensimmäinen ei T1 koodataan tilan säästämiseksi erikoisella tavalla eli negatiivinen luku inkrementoidaan ja positiivinen dekrementoidaan

3.7.3 CABAC

Sisältöön pohjautuvan adaptiivisen binäärisen aritmeettisen koodin, CABAC:n, käytöllä on H.264:ssä pyritty parantamaan suorituskykyä suhteessa CAVLC koodaukseen. Tosin hintana on lisääntynyt kompleksisuus. Tässä entropiakoodaustavassa yhdistetään adaptiivinen binäärinen adaptiivinen koodaus sisällön probabilistiseen mallintamiseen. CABAC koodauksella on sellaisissa testisekvensseissä, joita käytetään tyypillisesti broadcast sovelluksissa 30 – 38 dB (PSNR) signaalilaadulla, havaittu saavutetun 9 – 14 % kompressiotehokkuuden lisäys [MARP03]. Oheinen tarkastelu on hyvin ylimalkainen ja aiheesta syvällisemmin kiinnostunutta lukijaa kehotetaan tutustumaan Marpe et. al erinomaiseen artikkeliin [MARP03]

CABAC:ssa koodattavan symbolin ehdolliset todennäköisyydet mallinnetaan soveltamalla erilaisia todennäköisyysjakaumia symbolien välillä. Liikevektoreille muunnoskertoimille jne. on eri todennäköisyysmallit. Symboli sekvenssit pyritään kuvaamaan erilaisina binääripäätöksinä, jotka koodataan aritmeettisella enkooderilla käyttäen aikaisemmasta sisällön mallinnuksesta syntyneitä estimaatteja. Koodauksessa syntyneitä estimaateilla päivitetään sisällön estimointia seuraavaa koodausvaihetta varten. Periaatteessa näin pidetään kirjaa kehittyvästä statistiikasta.

CABAC:n käytöllä saavutetaan kolme merkittävää etua. Ensinnäkin voidaan käyttää biteille ei-kokonaislukuja per symboli. Toiseksi erittäin suuri tilastollinen adaptiviteetti mahdollistaa kooderin mukautumisen erilaisiin symbolien statistiikkoihin. Kolmanneksi kontekstin valinnalla voidaan mukautua sellaiseen statistiikkaan, joka on keskeistä koodattavalle datalle. CABAC koodausta voidaan siis soveltaa huomattavasti suuremmalle joukolle syntaksi elementtejä kuin mitä CAVLC koodauksella voitaisiin. Kun entropiakoodaukseksi on valittu CABAC toimintatapa, niin voidaan koodata makrolohkon tyyppi, intra ennuste toimintatavat, liikevektorit, referenssikuvaindeksit, muunnoskertoimet ja monet muut keskeiset bittivirrassa esiintyvät elementit adaptiivisesti. Tuloksena saadaan huomattavan kompakti koodi (kun CAVLC:ssä koodattiin ainoastaan kvanttisoidut muunnoskertoimet adaptiivisesti). Toisaalta CABAC toimii olennaisesti sarjamuotoisesti ja on hyvin laskentaintensiivinen. Erityisesti suurilla pikseli ja datanopeuksilla voi enkooderin laskentakyky tulla vastaan. Oheiseen kuvaan 4.8 on hahmoteltu CABAC koodauksen yleinen rakenne:



Kuva 12: CABAC kooderin rakenne

Koodauksen ensimmäisessä vaiheessa ei binääriarvoinen syntaksi elementti binärisoidaan eli muodostetaan kuvaus binäärisiin päätöksiin ns. bin sekvensseihin. Olennaisesti tämä on käytetyn aakkoston redusointi, jonka tuloksena saadaan yksikäsitteinen väliaikainen binäärinen koodisana. Käytännössä tämä tehdään käymällä läpi tiettyä puumaista rakennetta, jossa kussakin haarassa tehdään päätös koodisanasta. Jos taasen syntaksi elementti on jo valmiiksi binäärisessä muodossa niin vaihe ohitetaan.

Mahdollisen binärisoinnin jälkeen tehdään kontekstin tai asiayhteyden mallinnus. Yksi aritmeettisen koodauksen tärkeä ominaisuus on mahdollisuus hyödyntää selkeää rajapintaa mallinnuksen ja koodauksen välillä. Mallinnuksessa liitetään tietty todennäköisyysjakauma symboleille. Jakaumaa hyödynnetään sitten tulevissa koodausvaiheissa, joissa generoidaan jakaumaa vastaava bittisekvenssi. Niinpä käytetty malli olennaisesti määrää koodin ja tehokkuuden. Joten on ensiarvoisen tärkeää suunnitella sellainen malli, joka tutkii laajalla skaalalla symbolien keskinäisiä tilastollisia riippuvuuksia ja joka myös päivittää itseään enkoodauksen edistyessä. CABAC:ssa käytetään neljä eri tyyppistä perusmallia. Ensimmäisessä mallissa käytetään context templatea, joka pohjautuu kahteen menneisyydestä peräisin olevaan kahteen naapuri syntaksi elementtiin: Sopivien naapureiden valinnat perustuvat pitkälle koodattavan elementin ominaisuuksiin. Toinen tyyppi mallintaa yksinomaan makrolohkon (ja alilohkon) inter koodaustyyppejä (esim. p_8x8, p_8x16 P/SP-tyypin lohkoille). Kolmatta ja neljättä tyyppiä sovelletaan yksinomaan residuaaliseen jäännösdataan (kvanttisoituihin muunnoslohkoihin) [MARP03].

Aritmeettisessa kooderissa suoritetaan varsinainen koodaus. Kaikissa aritmeettisissa koodereissa on kolme perusaskelta:

- Seuraava koodattava symboli
- Nykyinen todennäköisyysintervalli, koodauksen alussa tämä asetetaan tiettyyn arvoon esim. puoliavoimeen(0, 0.5].
- Todennäköisyydet, jotka malli asettaa eri symboleille koodauksen tietyllä askeleella, nämä voivat muuttua (adaptiivisesti) siirryttäessä askeleesta toiseen.

Kooderi jakaa koodattavan intervallin osaintervalleiksi, joista jokainen edustaa tiettyä osaa nykyisestä intervallista ja on verrannollinen symbolin todennäköisyyteen, askeleessa käytettyyn kontekstiin. Kun tietty osaintervalli vastaa seuraavaksi koodattavaa aktuaalista symbolia tulee tämä intervalli seuraavassa vaiheessa käsiteltäväksi (jaettavaksi) intervalliksi. Aritmeettinen koodaus muistuttaa hyvin paljon Huffman koodausta (joista edellä kuvattu VLC on erikoistapaus)[8]. Mutta toisaalta koska aritmeettinen koodaus muuttaa koko viestin yhdeksi numeroksi lukujärjestelmässä b sen sijaan, että muuntaisi viestin jokaisen symbolin sarjaksi lukuja b:ssä. Tämän vuoksi aritmeettinen koodaus lähestyy optimaalista entropiakoodausta paljon lähemmäksi kuin mitä Huffman koodaus pystyisi.

H.264:ssä käytetty binäärinen aritmeettinen koodaus (BAC) on olennaisesti rekursiivinen intervallin osajako kooderi, joka toteuttaa seuraavanlaisen kertolaskutoimituksen: Oletetaan, että vähiten todennäköisen symbolin (LPS) estimaatti on aluksi annettu ja kuuluu puoliavoimeen väliin $p_{LPS} \in (0, 0.5]$. Lisäksi oletetaan, että annettu intervalli voidaan kuvata alarajalla L ja intervallin leveydellä (range) R. Näillä olettamuksilla annettu intervalli jaetaan kahdeksi osaintervalliksi, joista toisen leveys on:

$$R_{LPS} = R \times p_{LPS} \tag{19}$$

ja liittyy LPS:ään ja toisen leveys on taasen:

$$R_{MPS} = R - R_{LPS} \tag{20}$$

[8] Itse asiassa voidaan osoittaa, että Huffman on erikoistapaus aritmeettisesta koodauksesta.

joka taasen liittyy todennäköisimpään symboliin (MPS). MPS:n todennäköisyysestimaatti on täten $1 - p_{LPS}$. Tehdään päätös koodattavan symbolin kuulumisesta joko LPS:ään tai MPS:ään. Tämän mukaan valitaan vastaava osaintervalli uudeksi intervalliksi. Binäärinen arvo, joka osoittaa valittua intervallia esittää toistaiseksi tehtyjä binäärisiä päätöksiä. Päätösprosessin tuloksena saadaan binäärinen sekvenssi. Uusi intervallin leveys R vastaa binääristen symbolien todennäköisyyksien tuloa.

Käytännön toteutuksissa binäärisessä aritmeettisessa koodauksessa eräs suurimmista ongelmakohdista on kertolasku (19), jolla tehdään osaintervallien jako. On paljon tutkittu menetelmiä laskennan nopeuttamiseksi. Nämä perustuvat joko leveyden R tai todennäköisyyden p_{LPS} kertolaskua välttävään approksimaatioihin. Näitä menetelmiä on nimetty mm. Q, QM ja MQ kooderi [PENN88, RISS89].

Vaikka MQ kooderi on ehkä yksi parhaimmista aritmeettisista koodereista, niin kuitenkin se ei ole riittävän tehokas videon koodauksessa (suuremmilla nopeuksilla). Tämän vuoksi H.264:ssä on kehitetty vaihtoehtoinen kooderi ns. M-kooderi, modulo kooderiksi kutsuttu [MARP03]. Kooderissa ei tarvita ajonaikaista kertolaskua. Kooderin perusidea on kuvata R:n levyinen käytössä oleva intervalli $[R_{\min}, R_{\max})$ ja LPS:ään liittyvä todennäköisyysalue joukkoon vastaavia arvoja $Q = \{Q_0, \ldots, Q_{K-1}\}$ ja $P = \{p_0, \ldots, p_{N-1}\}$. Nyt voidaan kertolasku (19) approksimoida etukäteen lasketulla $K \times N$:n kokoisella taulukolla, jonka tekijät ovat siis:

$$Q_\rho \times p_\sigma, \quad 0 \le \rho \le K-1, \quad 0 \le \sigma \le N-1 \tag{21}$$

Laskennan tehokkuuden pitämiseksi kohtalaisen korkeana joudutaan valitsemaan ihanteellinen taulukon koko ja riittävän hyvä approksimaatio täsmällisestä osajaosta. H.264:ssä käytetään K= 4 kvanttisoitua leveysarvoa ja N = 64 LPS:ään liittyvää todennäköisyysarvoa.

4 Verkkokerros NAL[9]

Kuten aikaisemmin tuli jo mainittua standardissa on erotettu toisistaan varsinainen videonkäsittely VCL:ksi (Video Coding Layer) ja verkkoon liityntä NAL:ksi (Network Abstraction Layer). Rakenteella on tavoiteltu monia asioita. Ensinnäkin videon koodaus voidaan tehdä suhteellisen riippumattomaksi käytetystä protokollista[10] ja verkkoarkkitehtuurista. Toiseksi järjestelmä on haluttu sopeuttaa mahdollisimman monenlaiseen sovellukseen kuten [WENG03]:

- RTP/IP sovelluksiin sekä langallisissa että langattomissa ja näiden yhdistelmäsovelluksissa.
- Tiedostojärjestelmiin kuten kovalevytallennuksiin ja erilaisiin DVD arkkitehtuureihin (puna- ja sinilaser)
- H.32x langallisiin ja langattomiin sovelluksiin
- MPEG-2 System mukaiset broadcast sovelluksiin kuten DVB, IPTV paluukanavaiset hybridijärjestelmät jne.

Tällainen jako voidaan tehdä myös toisella tavalla eli tarkastelemalla eri osajärjestelmien toiminnallisuutta:

- Kaksisuuntaiset keskustelusovellukset, joita ovat esimerkiksi videoneuvottelut ja videopuhelimet. Näitä sovelluksia luonnehtivat erittäin tiukat viivevaatimukset. Tavoitteena on saavuttaa korkeintaan 100 ms päästä-päähän latenssiaika (mm. huulisynkronoinnin vuoksi). Järjestelmät voivat olla point to point tai point to multipoint. Sovellukset vaativat siten reaaliaikaisen enkooderin ja dekooderin, joiden keskeisiä ominaisuuksina pitäisi olla erilaisten koodausparametrien sekä verkon vikastatistiikkaan mukaisen virheistä toipumismekanismien säätö reaaliaikaisesti.
- Etukäteen koodatut video streamit, jotka tallennetaan massamuistilaitteisiin. Liikennöinnin kannalta erilaiset luotettavat download protokollat kuten ftp ja http ovat olleet tavanomaisia välitystapoja. Videokooderi optimoi näissä sovelluksissa suurimpaan mahdolliseen aiotun sovelluksen rajoittamaan koodaustehokkuuteen ja laatuun. Sinänsä tässä ei ole juurikaan viive- tai latenssiaikavaatimuksia. Koska sovellukset eivät vaadi reaaliaikaisuutta, niin laskennan kompleksisuus ei ole keskeisimpiä suunnitteluseikkoja.
- Streaming sovellukset. Kirjallisuudessa ei esiinny mitään yhteisesti hyväksyttyä määritelmää siitä mitä ”streaming”:lla itse asiassa tarkoitetaan. Laajasti ymmärrettynä sillä kuitenkin tarkoitetaan sellaista siirtopalvelua, missä videota voidaan alkaa toistamaan ennen kuin koko videoon sisältyvä bittivuo on lähetetty, esimerkiksi IPTV protokollan alaisuudessa. Viivevaatimusten suhteen streaming sovellukset ovat edellä mainitun kahden välillä. Lähetyksen käynnistymiseen voi kulua muutamakin sekunti. Bittivuo voi olla etukäteen tallennettu ja lähetetään tarpeen vaatiessa tai elävä sessio joka kompressoidaan reaaliajassa. IP

[9] Luvun tarkastelut perustuvat pitkälle Stephan Wengerin erinomaiseen artikkeliin H.264/AVC Over IP [WENG03].

[10] Tosin IP protokollien maksimaalinen pakettikoko ilman uudelleen fragmentointia on n. 1500 tavua (MTU). Tällöin tehokkain liikennöinti voidaan saavuttaa kun yhdessä paketissa lähetettävä viipale on tätä pienempi.

protokollien ollessa kyseessä streami voi olla koodattu useammalla kuin yhdellä esitystavalla ja eri bittinopeuksilla. Näin saadaan palveltua eri nopeuksilla olevaa loppukäyttäjää. Samoin streami voidaan lähettää yhden tai useamman multicast kanavan kautta monelle käyttäjälle. Streaming sovellukset käyttävät siirrossa usein epäluotettavia siirtoprotokollia (kuten UDP:n päälle tehtyä RTP:tä), jolloin vastaanottajan näkemän virheen käsittely on luotettavan toiminnan kannalta hyvin tärkeä suunnitteluparametri. Erityisesti virheen käsittely korostuu kun loppukäyttäjän ja palvelimen välillä ei ole kättelevää tai takaisinkytkettyä protokollaa.

Näinkin laaja sovelluskirjo asettaa tietenkin myös vaatimuksia siirrossa esiintyvien virheiden käsittelyyn. Virheiden käsittely ei ole uusi asia, vaan myös vanhemmissa standardeissa on asiaan kiinnitetty jo huomiota ja kehitetty erilaisia menetelmiä ja työkaluja. Esimerkiksi standardeissa H.261, H.262, H.263 ja MPEG-1/2 esiintyi mm. seuraavia:

- Eri tapoja kuvien segmentointiin (kuten GOP, group of pictures)
- Intra makrolohkojen, viipaleiden ja kuvien sijoitus bittivuohon.
- Vuon hyvin määritelty syntaksi

Uudemmassa H.263:ssa laajennettiin näitä mm seuraavilla [WENG03]:

- Referenssikuvan valinta, joko takaisinkytkettynä tai ilman, GOB tai makrolohkotasoisena
- Datan partitiointi jakaminen sopiviin ryhmiin.

Lopulta H.264:ään oli lisätty muutama työkalu virheisiin sopeutumiseksi:

- Parametrijoukot
- Joustava makrolohkojen järjestäminen (FMO)
- Redundanttit viipaleet.

Intra koodattujen kuvien sijoitus itse videovuohon on ollut keskeinen mekanismi ns. drift virheiden eliminoimiseksi tai kompensoimiseksi. Tässä kun tavallaan voidaan synkronoida tai alustaa koko dekoodaus uudelleen kuvatasolla aina I kuvan esiintyessä. Esimerkikksi MPEG-2:ssa GOP muodostui aina intrakoodatusta kuvasta seuraavaan intrakoodattuun kuvaan saakka. Kun H.264:ssä pyrittiin mahdollisimman suureen koodaustehokkuuteen, niin tällainen driftin katkaiseminen on, ikävä kyllä, monissa tapauksissa invalidisoitu. Ensinnäkin spatiaalinen intrakoodaus ennusteet voidaan tehdä (inter) ennustetuista makrolohkoista. Toiseksi inter ennusteiden referenssiviipaleina voi olla viipaleet, jotka esiintyvät ennen intrakoodattua kuvaa (viipaletta).

Sopivan referenssikuvan valinnalla voidaan myös vaikuttaa virheisiin sopeutumiseen. Takaisinkytketyissä järjestelmissä enkooderi voi vastaanottaa tietoa hävinneistä tai särkyneistä kuva-alueista. Vasteena tämä voi reagoida lähettämällä vanhemman oikeaksi tiedetyn makrolohkon (viipaleen) tai viitteen tähän, jotta dekooderi voisi tehdä korrektin ennusteen sen sijaan, että enkooderi lähettäisi siirron kannalta kalliimman intrakoodatun informaation kokonaan [WENG03].

Tavallisesti makrolohkon kaikki symbolit koodataan yhteen bittijonoon. H.264:ään sisältyvässä datan partitioinnissa kuitenkin luodaan 3 bittijonoa per viipale. Kuhunkin jonoon, viipaleen kaikki keskenään läheisessä semanttisessa yhteydessä olevat osat kootaan yhteen partitioiksi, osajoukoksi. H.264:ssä käytetään seuraavia partitioita:

- Partitio A. Otsaketieto, mukaan lukien MB tyypit, kvanttisointiparametrit ja liikevektorit. Partitio on tärkein, sillä ilman tätä muiden partitioiden käsittely invalidisoituu.
- Partitio B. Intra partitio, sisältää Intra CBP[11] ja muunnoskertoimet. Jotta partitio B olisi käyttökelpoinen pitää partitio A olla vastaanotettu korrektisti. Koska intra informaatio voi estää monissa tapauksissa drift-virheen syntymisen, niin osajoukko B on myös virheisiin sopeutumisen kannalta tärkeämpi kuin partito C.
- Partitio C. Inter partitio, sisältää inter CBP ja inter kertoimet. Monissa tapauksissa partitio on, koodatussa viipaleissa, kooltaan suurin. Virheen käsittelyn kannalta partitio on myös vähiten merkitsevä kun se ei sisällä informaatiota, jolla voitaisiin resynkronoida dekooderi kun siirtovirhe on tapahtunut.

Kun datan partitiota käytetään, niin enkooderi pistää erityyppiset symbolit kolmeen eri puskuriin. IP liikenteessä viipaleiden koot pitäisi valita sellaiseksi, ettei mikään partitio johda sellaisiin pakettikokoihin, jotka ovat suurempia kuin MTU.

H.264:ssä ainoastaan sellainen tieto, joka muuttuu kuvasta toiseen (esimerkiksi temporaaliset viitteet) on sisällytetty viipaleeseen. Kaikki muu dekoodauksessa tarvittava data (kuten kuvan koko, käytetty entropiakoodaus jne.) lähetetään erikseen hyvissä ajoin ennen käyttöä, viipaleen koodaamista. Aikaisemmissa standardeissa erilaiset otsaketiedot olivat synkronissa varsinaisen bittijonon kanssa, jolloin muutos kuvan parametreihin vaikutti kaikkiin viipaleisiin ja tuleviin kuviin välittömästi. Lähetettäessä parametrit erikseen tällainen synkronointi on vaikeaa. H.264:ssä tällaiset parametrit onkin sisällytetty em. parametriryhmiin. Synkronointi voidaan saavuttaa lähettämällä viite tiettyyn mahdollisesti ja paljon aikaisemmin lähetettyyn parametriryhmään. Ryhmiä voidaan lähettää useita tai ne on voitu ”polttaa” kiinteästi dekooderiin (ohjelmakoodiin tai ROM muistiin).

Erikseen, usein jo istunnon alussa, lähetettävien parametriryhmien käytöllä on monia etuja. Ensinnäkin ne voidaan lähettää luotettavalla protokollalla. Toiseksi kuvapuhelinjärjestelmissä lähettävä ja vastaanottava pää voivat jo tunnistaa keskinäiset yhteensopivuutensa ennen varsinaista liikennöintiä, videon lähettämistä.

FMO (flexible Macroblock Ordering) mekanismilla mahdollistetaan makrolohkojen liittämisen viipaleisiin toisella tavalla kuin mitä kameralta skannatessa nämä esiintyivät. FMO:ssa jokainen makrolohko on staattisesti liitetty tiettyyn viipaleryhmään ns. makrolohko allokointi kartan kautta (MBAmap). Itse viipaleessa makrolohkojen koodaus tehdään normaalin skannauksen tapaan [WENG03].

[11] CBP = coded block pattern, kertoo mikä joukko intrakoodatusta lohkosta sisältää 0:sta poikkeavia kertoimia

0	1	2	3	4	5
6	7	8	9	10	11
12	13	14	15	16	17
18	19	20	21	22	23

Kuva 13: 6 x 4 makrolohkoa sisältävä kuva, jaettuna kahteen viipaleryhmään. Sinertävät kuuluvat viipaleeseen 0 ja kellertävät viipaleeseen 1.

Oheiseen kuvaan 13 on hahmoteltu eräs FMO. Kuvan oletetaan olevan niin pieni, että kaikki makrolohkot on allokoitu joko viipaleeseen 0 tai viipaleeseen 1. Lisäksi oletetaan, että siirron aikana jokin viipaleryhmään 1 kuuluva informaatio häviää esimerkiksi kuvan lohko 15. Koska jokaisella hävinneellä makrolohkolla on spatiaalinen naapuri, joka kuuluu toiseen viipaleeseen, on virheistä toipumismekanismilla kuitenkin riittävästi informaatiota, jotta virheen vaikutus voitaisiin piilottaa. Tällainen piilottaminen, rekonstruktio, voitaisiin tehdä esimerkiksi lohkolla 21. Wenger raportoi sellaisesta videoneuvottelusovelluksesta, missä kuvan koko on CIF ja hävikki on jopa 10 % bittivirrasta. Hävikkien visuaalinen vaikutus on kuitenkin niin pieni kuvatulla virheenkorjausmekanismilla, että vain harjaantunut silmä pystyy havaitsemaan ympäristön olevan näinkin häviöllinen [WENG03]. Joustavasta makrolohkojen järjestyksestä maksettu hinta on hieman pienempi koodaustehokkuus ja erittäin optimoiduissa ympäristöissä hieman kasvanut viive [WENG03].

Virheiden käsittelymekanismina H.264:ssä on myös mahdollisuus käyttää redundantteja viipaleita (RS). Eli samassa bittivirrassa voi esiintyä jo koodatun tiettyyn viipaleseen sisältyvän makrolohkon lisäksi toinen, yleensä eri parametreilla koodattu sama makrolohko, toisintona. Esimerkiksi ensimmäinen voidaan koodata alhaisella kvanttisointiparametrilla (ja siksi omaa korkean laadun) jälkimmäinen voidaan koodata suurella kvanttisointiparametrilla (ja siksi on kooltaan pienempi ja laadultaan heikompi). Dekooderi pyrkii ensisijaisesti dekoodamaan ensimmäisen viipaleen ja hylkäämään jälkimmäisen. Mutta mikäli ensimmäinen viipale puuttuu (esimerkiksi pakettihäviön seurauksena), niin voidaan käyttää redundanttia viipaletta rekonstruktioon. RS suunniteltiin alun perin mobiileihin ympäristöihin mutta on myös aivan yhtä tehokas langallisissa IP verkoissa.

4.1 NALU:t

Koodattu video on järjestetty ns. NAL yksikköihin NALU:hin. Yksinkertaisesti sanottuna NALU on vaihtuvanmittainen tavujono, joka sisältää tiettyyn luokaan kuuluvia syntaksi-elementtejä. Esimerkiksi on olemassa NALU:ja, jotka sisältävät koodattuja viipaleita (edellä mainituja A, B tai C tyypin partitioita) tai muita kuvaan sisältyvä dekoodauksessa tärkeitä parametreja sisältäviä sekvenssejä.

0	1	2	3	4	5	6	7
T					R		F

Kuva 14: NALU otsikko

Jokainen NALU sisältää yksitavuisen otsakkeen (kuva 14) ja vaihtelevan määrän datatavuja, jotka sisältävät koodattavat symbolit varsinaisena hyötykuormana. Otsikko muodostuu seuraavasta kolmesta kentästä [WENG03]:

- T 5 bittinen kenttä, joka luonnehtii NALU:n 32 eri tyyppiin, joista tyypit 1 -12 on tällä hetkellä määritelty H.264:ssä. Tyypit 24 – 31 on määritelty muualla kuin H.264 standardissa mm. RTP hyötykuormamäärittelyssä. Kaikki muut arvot (13 - 23) on taasen määritelty tulevaa käyttöä varten
- R määrittelee NAL yksikön tärkeyden tulevassa rekonstruktiossa. Arvo 0 ilmaisee, että NALU:a ei käytetä ollenkaan rekonstruktioon (jolloin dekooderi voi hylätä tämän ilman drift riskiä). Kun arvo on suurempi kuin 0 niin NALU on tärkeä muodostettaessa katseltavaa videota (mitä suurempi arvo sen suurempi merkitys).
- F ns. forbidden_bit, enkoodauksessa tämä asetetaan 0:ksi. Liikennöintiverkko voi asettaa tämän 1:ksi kun se havaitsee virheen. Kentän hyödyllisyys käy ilmi ns. heterogeenisissa verkoissa (kuten langaton/langallinen yhdistelmäverkko). Esimerkiksi gateway voi havaita saapuvan NAL:n tarkistussummassa virheen, jolloin se voisi hylätä koko NALU:n ja välittää vain virheettömät NALU:t eteenpäin. Toisaalta se voi merkitä NALU otsakkeen F bitin yhdeksi jolloin päätös NALU:n hylkäämisestä tehdään vasta dekooderissa, joka tällöin tietää, että NALU:ssa voi (ei kuitenkaan välttämättä) esiintyä virhe (virhe kun voi esiintyä vain tarkistussumassa vaan ei hyötykuormassa).

4.1.1 NALU:t fragmentoiduissa bittivuossa

Kaikki aikaisemmat standardit perustuivat liikennöinnissään bittistreameihin eli korkeamman tason syntaktiset elementit erotettiin toisistaan start koodilla. Tällä koodilla pyrittiin uudelleen synkronoimaan vastaanotto, jos bittivirrassa esiintyi jonkinlaista korruptiota esimerkiksi bittivirheiden tai suoranaisten häviämisten seurauksena. H.264:ää voidaan myös käyttää tämänkaltaista kehystämistä, esimerkiksi H.320 tai MPEG-2/H.220.0 (system) sovelluksissa. Näissä lähetetään osa NALU:sta tai koko NALU järjestettynä vuona (streamina), jossa NAL yksikön alku ja loppu pitää tunnistaa ja erottaa varsinaisesta koodatusta hyötydatasta.

Tällaisiin järjestelmiin H.264 määrittelee tavu stream formaatin: Formaatin mukaan jokainen NALU varustetaan 3 kolmitavuisella start-etuliitekoodilla. NAL yksikön reunat voidaan tämän jälkeen löytää etsimällä bittijonosta yksikäsitteiseksi muodostettu start etuliite. Lisäksi voidaan lisätä yksi tavu koodattavaa kuvaa kohden, jolla voidaan parantaa bittistremin kohdistamista tavurajoihin.

4.1.2 NALU:t paketti liikennöinnissä

Muissa järjestelmissä (esimerkiksi internet siirtoprotokollissa, RTP:ssä ja muissa) koodattu data siirretään paketeissa. H.264:n paketointia kutsutaan ”yksinkertaiseksi paketoinniksi”. Paketoinnin säännöstö on nimittäin sangen suoraviivainen:

1. pistä yksi NALU (mukaan lukien otsake) yhteen protokollan hyötykuorma pakettiin.
2. pistä protokollan määritysten mukainen otsake pakettiin.
3. Lähetä.

Kuten aikaisemmin mainittiin, niin IP fragmentoinnin estämiseksi H.264 videokoodauskerroksen ei koskaan pitäisi generoida NALU:ja jotka olisivat suurempia kuin MTU. Mutta käytännössä esiintyy paljonkin tilanteita jossa tätä vaatimusta ei voida täyttää. Tällaisissa tapauksissa voitaisiin luottaa IP protokollan mukaisiin fragmentointeihin 64 ktavun NALU:hin saakka. Mutta näin ei kuitenkaan saada sitä

hyötyä mitä sovelluskerroksen suojausmekanismit tarjoavat kuten datanpartitiointia edellä mainittuihin luokkiin A, B ja C. Toiseksi saattaa esiintyä sovelluksia (kuten digitaalinen elokuvateatteri), jossa pakettien koko ylittää helposti 64 kilotavun rajan. Samoin aggregaatio on mahdollista eli hyvin pienet paketit kuten muutaman tavun oheisinformaatio voidaan liittää toisen NALU:n kanssa yhdeksi RTP paketiksi. Moniin protokolliin sisältyvä ikkunointi taasen saattaa aggregaation seurauksena viivästyttää lähetystä.

5. FRExt

Alkuperäiseen 2003 standardiluonnokseen tehtiin paljon laajennuksia, jotka tunnetaan nimellä Fidelty Range Extension (FRExt). Toukokuussa 2004 laajennukset tulivat lisäyksenä standardiluonnokseen ja standardin kesällä 2005 hyväksyttyyn versioon tulivat integroiduksi osaksi varsinaista virallista julkaisua. Monissa sovelluksissa väitetään laajennusten parantaneen koodaustehokkuutta MPEG-2:een nähden suhteessa 3:1 [katso esimerkiksi SULL04b]. Ohessa tarkastelemme hieman näitä laajennuksia.

Vaikka alkuperäinen H.264 toukokuulta 2003 oli ensisijaisesti suuntatutunut ”entertainment”-sovelluksiin, niin sillä oli jo ”alkumetreillä” hyvin paljon eri sovelluskohteita. Sovelluksia, joissa video perustuu 4:2:0 croma näytteen ottoon 8 bitin resoluutiolla. Standardiluonnos ei vielä sisältänyt kunnollista tukea vaativille ammattimaiselle sovelluksille kuten studio editointi, filmien post-prosessointi ja muut vastaavanlaiset sovellukset. Tämän vuoksi standardia laajennettiin:

- Sallimalla suuremmat resoluutiot kuin 8 bittiä (10 ja 12 bitin sovellukset).
- Värien esittäminen tarkemmaksi kuin mitä tavalliset kuluttajasovellukset edellyttävät eli mahdollistettiin 4:4:4 ja 4:2:2 näytteenotto.
- Laajempi tuki joustavalle editoinnille, kuten alpha- kanavaan perustuvaan häivytykseen.
- Mahdollista käyttää erittäin korkeita bittinopeuksia aivan 240 Mb/s saakka.
- Suuret kuvakoot (4096x2304 saakka)
- Sallittiin mahdollisuus esittää osia sekvenssin kuvista häviöttöminä
- Mahdollisuus esittää video RGB värimaailmassa.

Tuloksena tehdyistä tarkasteluista laadittiin 4 uutta profiilia täydentämään Main profiilia. Näille uusille profiileille annettiin nimet High, High 10, High 4:2:2 ja High 4:4:4. Monista lisäyksistä esittelemme ohessa lyhyesti laajennukset muunnosalueen koodaukseen, katsojan havaintokykyyn mukautuva kvanttisointi, häviötön koodaustapa ja väriavaruuden muutokset.

5.1 Muunnosalueen koodaus

Kuten edellä mainittiin, 16 x 16 makrolohko voidaan jakaa pienempiin osiin aina 4 x 4 lohkokokoon saakka, joille tehtiin laskennallisesti yksinkertainen muunnos. Tämä voitiin tehdä kokonaan 16 bittisellä kokonaisluku aritmetiikalla kun resoluutio oli 8 bittiä. Joissakin tapauksissa (yhdistettynä deblock suodatukseen) ei tällä muunnoksella voitu esittää hienon hienoja yksityiskohtia. Tämän vuoksi otettiin FRExt laajennuksiin mukaan

myös kohtalaisen yksinkertainen 8 x 8 muunnosmatriisi. Samankaltaisella päättelyllä kuin aikaisemmin tehtiin, päädyttiin seuraavaan muunnosmatriisiin (emme toista laskelmia tässä):

$$\mathbf{H}_{8x8} = \begin{bmatrix} 8 & 8 & 8 & 8 & 8 & 8 & 8 & 8 \\ 12 & 10 & 6 & 3 & -3 & -6 & -10 & -12 \\ 8 & 4 & -4 & -8 & -8 & -4 & 4 & 8 \\ 10 & -3 & -12 & -6 & 6 & 12 & 3 & -10 \\ 8 & -8 & -8 & 8 & 8 & -8 & -8 & 8 \\ 6 & -12 & 3 & 10 & 10 & -3 & 12 & -6 \\ 4 & -8 & 8 & -4 & -4 & 8 & -8 & 4 \\ 3 & -6 & 10 & -12 & 12 & -10 & 6 & -3 \end{bmatrix} \quad (22)$$

Kuten aikaisemmin todettiin, niin 16 x16 lohkon sisältyvät 4 x 4 muunosten DC komponenteille tehtiin 4 x 4 Hadarmad muunnos Intra_16x16 toimintamoodissa. 4:2:0 tyyppisen videon 4 x 4 muunnoksen croma komponenttien DC tekijöille tehdään vastaavasti 2 x 2 Hadramad. 4:4:4 värimaailmassa väritekijöitä on kaksinkertainen määrä 4:2:0:aan nähden joten croma muunnoksessa tämän DC tekijöille tehdään myös 4 x 4 Hadarmard.

5.2 Perseptuaallinen kvanttisointi skaalaus

FRExt laajennukset tukevat sellaista jo MPEG-2:ssa vakiintunutta piirrettä, jonka mukaan kvanttisoinnissa skaalausmatriisina voidaan myös käyttää havaitsemiseen paremmin soveltuvaa skaalausta. Enkooderi voi päättää kutakin muunnoslohkon kokoa kohden ja erikseen intra- tai interkoodausta kohden räätälöidyt skaalauskertoimet dekooderin tehdessä käänteiskvanttisoinnin. Näin voidaan säätää kvanttisoinnin laatua ihmisen näkökyvyn mukaiseksi eri virhetyypeillä [SULL04b]. Tällä ei juurikaan paranneta objektiivista laatua (esimerkikksi mitattuna PSNR:n avulla) vaan subjektiivista, koettua, laatua. Standardissa on määritelty oletusmatriisit kvanttisoinnille mutta enkooderi voi lähettää räätälöidyt arvot dekooderille joko sekvenssi tai kuvatasolla.

5.3 Häviötön makrolohko toimintatapa

Kun videon laatu on korkea (so. kun kvanttisointiaskel on hyvin pieni) on mahdollista joissakin harvinaisissa tapauksissa, että kompression sijasta tapahtuu datan ekspanssio. Lisäksi toteutusnäkökulmasta katsottuna olisi kätevää, että olisi mahdollista määritellä yläraja dekooderissa prosessoitavien yhteen makrolohkoon kuuluvien bittien määrälle. Tämän vuoksi standardiin on lisätty PCM makrolohko toimintamoodi, jossa kuvasta otetut näytteet lähetetään suoraan ennustamatta, muuntamatta tai kvanttisoimatta eli häviöttömästi.

Häviötön toimintatapa on kompression kannalta katsottuna tehoton, sen ei ole edes tarkoitettu olevan tehokas. Sen on tarkoitettu olevan yksinkertainen ja asettavan minimi ylärajan niiden bittien lukumäärälle, jotka esittävät makrolohkoa tietyllä tarkkuudella. Häviötön toimintatapa on mahdollista vain Hi444P profiilissa. Häviötön toimintatapa on kuitenkin kohtalaisen tehokas tietyssä mielessä. Ei kuitenkaan saman vertainen kuin parhaimmat still kuva enkooderit, mutta yhteistoiminnassa interkuva ennusteiden kanssa se voi olla erittäin tehokas lisäys aika-ajoin tapahtuvana koodauslisänä.

5.4 Värimaailman esitykset

Aikaisemmissa standardeissa värimuunnokset ovat hyvin pitkälle perustuneet liukuluku laskentaan ja näin ovat olleet ”herkkiä” pyöristysvirheille ja näiden kumulatiivisille kasvulle. Normaalisti videota kuvataan ja esitetään RGB maailmassa. Värikomponentit ovat olleet vahvasti keskenään korreloituneita. Lisäksi jo 50 luvulla havaittiin, että ihmissilmä havaitsee herkemmin luminanssimuutokset kuin krominanssimuutokset (saturaatio ja sävy). Perinteisesti TV tekniikassa onkin ennen muuta signaalikäsittelyä tehty RGB – YCbCr muunnos. Kompressio on sittemmin tehty YCbCr signaalille. Tällainen ”TV:stä tuttu” muunnos voidaan määritellä:

$$
\begin{aligned}
&Y = K_R * R + \left(1 - K_R - K_B\right) * G + K_B * B, \quad K_R = 0.2126, \quad K_B = 0.0722 \\
&Cb = \frac{1}{2}\left(\frac{B - Y}{1 - K_B}\right) \\
&Cr = \frac{1}{2}\left(\frac{R - Y}{1 - K_R}\right)
\end{aligned}
\tag{23}
$$

Missä R vastaa punaista, G vihreätä ja B sinistä (kts. detaljit esim. [HANH03]). Kaavan (23) lähestymistavassa on vain kaksi ongelmaa [SULL04b]. Ensinnäkin koska näytteet esitetään kokonaislukuina (8, 10 tai 12 bittisinä) niin kaava (23) aiheuttaa pyöristysvirheitä sekä eteenpäin että taaksepäin muunnoksessa. Toiseksi koska muunnos ei alun perin oltu suunniteltu digitaaliseen datan siirtoon ja käsittelyyn niin se on vähemmän ihanteellinen kompromissi muunnoksen kompleksisuuden[12] ja koodaustehokkuuden välillä. FRExt laajennuksissa käsiteltiin jälkimmäistä ongelmaa esittelemällä uusi väriavaruus YCgCo, missä g viittaa vihreään ja o oranssiin cromaan. Muunnos on huomattavasti (23) yksinkertaisempi:

$$
\begin{aligned}
&\mathrm{Y} = \frac{1}{2}\left(G + \frac{R + B}{2}\right) \\
&\mathrm{Cg} = \frac{1}{2}\left(G - \frac{R + B}{2}\right) \\
&\mathrm{Co} = \frac{R - B}{2}
\end{aligned}
\tag{24}
$$

Selvästi muunnos (24) pienentää kompleksisuutta ja voi jopa parantaa koodaustehokkuutta. Mutta se ei kuitenkaan täysin poista pyöristysvirheitä, tosin tämä voidaan pienentää käyttämällä kahta ylimääräistä bittiä croma:n esitys tarkkuudessa.

FRExt itse asiassa menee hieman pidemmälle esittämällä muunnoksesta (24) variaation jossa ei esiinny kahta ylimääräsitä bittiä eikä aiheuta pyöristysvirheitä. Värimuunnos RGB:stä YCgCo:hon tehdään seuraavasti:

[12] Esimerkiksi on vaikea toteuttaa kertoimia 0.2126 ja 0.02722 puhtaasti kokonaisluku-järjestelmissä)

$$\begin{aligned} \mathrm{Co} &= R - B \\ t &= B + (\mathrm{Co} \gg 1) \\ \mathrm{Cg} &= G - t \\ \mathrm{Y} &= t + (\mathrm{Cg} \gg 1) \end{aligned} \tag{25}$$

missä t on laskennassa käytetty ”välimuuttuja”. Käänteismuunnokselle RGB:hen taasen on määritelty:

$$\begin{aligned} t &= \mathrm{Y} - (\mathrm{Cg} \gg 1) \\ G &= t + \mathrm{Cg} \\ B &= t - (\mathrm{Co} \gg 1) \\ R &= B + \mathrm{Co} \end{aligned} \tag{26}$$

6 Profiilit ja tasot

Kuten MPEG-2:ssa on H.264:äänkin sisällytetty profiilien ja tasojen käsitteet. Profiileilla ja tasoilla pyritään takaamaan sellaisten sovellusten yhteensopivuus, joilla on samankaltaiset toiminnalliset vaatimukset. Profiililla tarkoitetaan menetelmiä ja algoritmeja joilla voidaan muodostaa yhdenmukaiset bitstreamit. Tasoilla taasen asetetaan erilaisia kompressiossa käytettäviä avainparametreja. Näillä parametreilla on vaikutus saavutettavaan kompressiotehokkuuteen ja tätä kautta lopullisen kuvan esitystaajuuteen.

Vuonna 2005 hyväksytyssä ”lopullisessa” srtandardissa on määritelty 7 profiilia, joista High profiilit tunnetaan myös nimella FRExt laajennukset. Profiilit ovat :

- Baseline (BP) on suuntautunut reaaliaikaiseen koodaukseen laskennallisesti rajallisissa laitteissa (kuten kämmentietokoneissa), tukee I ja P viipaleita CAVLC entropia koodausta, sekä SP ja SI viipaleita sekä muita perusominaisuuksia.
- Main (MP) on suunnattu erityisesti broadcast ympäristöön. Tukee sekä lomitettuja että ei-lomitettuja kuvasekvenssejä, I, P ja B viipaleita, makrolohko tasoista adaptiivista kuva/kenttä koodausta (lomitetussa videossa), entropia koodauksessa voidaan käyttää CAVLC ja CABAC
- Extended (XP) on suunnattu vika-alttiisiin kanaviin (kuten langattomiin verkkoihin), tuki I,P,B, SP ja SI viipaleisiin, vain CAVLC entropiakoodaus sekä tuki lomitettuun ja progressiiviseen videoon.
- High laajentaa main profiilia erityisesti HD sisältöjen koodaukseen, käyttää adaptiivista 8x8 tai 4x4 muunnosta sekä 8 bitin resoluutiota 4:2:0 näytteenotolla.
- High 10 (Hi10P) high profiilin laajennus 10 bittisiin sisältöihin 4:2:0 näytteenotolla.
- High 4:2:2 (Hi4222P) high profiilin laajennus, siinä on 4:2:2 croma näytteenotto 10 bittisissä järjestelmissä, erityisesti tarkoitettu videon tuotantoon ja editointiin
- High 4:4:4 (Hi444P) croma formaatti 4:4:4 aina 12 bittiseen resoluutioon saakka, sallii häviöttömän toimintamoodin ja suoran RGB signaalin käsittelyn. Profiili on tarkoitettu sekä grafiikan kompressioon että vaativaan professionaaliseen tuotantoon.

Oheiseen taulukkoon on tehty yhteenveto kolmen perusprofiilin ominaisuuksista [SULL04b]:

Menettely	Baseline	Main	Extended
I ja P viipaleet	X	X	X
CAVLC	X	X	X
CABAC		X	
B viipaleet		X	X
Lomitettu koodaus		X	X
Joustava virheenkäsittely FMO, ASO, RS	X		X
Parannettu virheenkäsittely DP			X
SP ja SI viipaleet			X

Yhteenveto FRExt ominaisuuksista on tehty oheiseen taulukkoon [SULL04b]:

Menettely	High	High 10	High 4:2:2	High 4:4:4
Main profiilin menettelyt	X	X	X	X
4:2:0 formaatti	X	X	X	X
8 bitin resoluutio	X	X	X	X
8x8/4x4 muunnos	X	X	X	X
Perseptuaaliset kvanttisointi matriisit	X	X	X	X
Erilliset Cb ja Cr kvanttisoinnin ohjaus	X	X	X	X
Musta-valkoinen video	X	X	X	X
9 ja 10 bitin resoluutio		X	X	X
4:2:2 formaatti			X	X
11 ja 12 bin resoluurio				X
4:4:4 formaatti				X
Kehittynyt värimuunnokset				X
Ennustava häviötön koodaus				X

Tasoina taasen on määritelty 16, joilla määritellään esimerkiksi kuvan koko makrolohkoina, dekooderin prosessointinopeus (makrolohkoa sekunnissa), monikuvapuskurin koko jne. Itse asiassa enkooderin ja dekooderin yhteistoiminnallisuuden määräämiseksi pitäisi varsinaisen profiilin lisäksi määritellä käytetty taso [PURI04]. Tasot ovat kytkeytyneitä kuvan kokoon ja tätä kautta myös maksimi kuvataajuuteen. Tasot on numeroitu seuraavasti [ITUT05]:

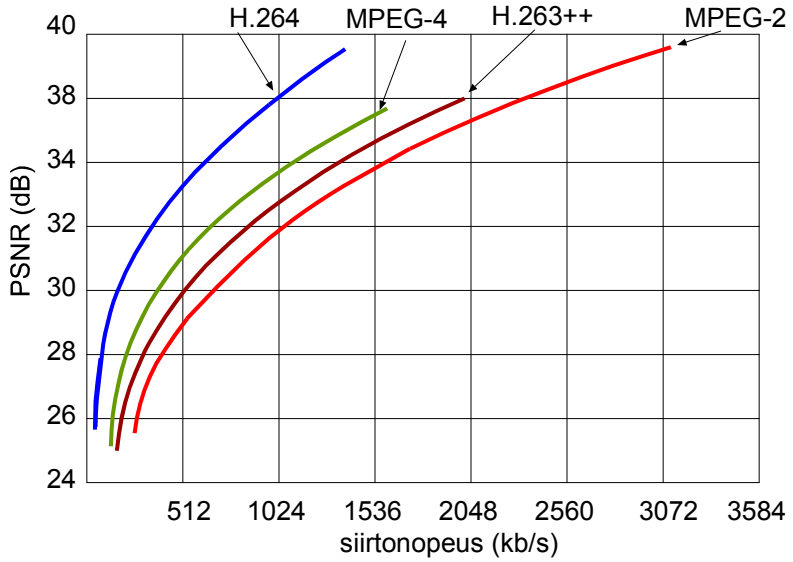
taso	Maks. kuvan koko (näytettä)	Framerate kuvaa/s				Maks. videon bitrate kb/s	Max dekood. kuvamuisti (1024 B 4:2:0:lla)	Mkas kuvan koko makrolohkoja	Maks. makrolohkoa/ s
		720p HD	1080 HD	XGA (1024x768)	CIF (352x288)				
1	25 344	-	-	-	-	64	148.5	99	1 485
1b	25 344	-	-	-	-	128	148.5	99	1 485
1.1	101 376	-	-	-	7.6	192	337.5	396	3 000
1.2	101 376	-	-	-	15.2	384	891.0	396	6 000
1.3	101 376	-	-	-	30	768	891.0	396	11 880
2	101 376	-	-	-	30	2 000	891.0	396	11 880
2.1	202 752	-	-	-	50	4 000	1 782.0	792	19 800
2.2	414 720	-	-	-	51.1	4 000	3 037.5	1 620	20 250
3	414 720	-	-	-	102.3	10 000	3 037.5	1 620	40 500
3.1	921 600	30.0	-	35.2	172.0	14 000	6 750.0	3 600	108 000
3.2	1 310 720	60.0	-	70.3	172.0	20 000	7 680.0	5 120	216 000
4	2 097 152	68.3	30.1	80.0	172.0	20 000	12 288.0	8 192	245 760
4.1	2 097 152	68.3	30.1	80.0	172.0	50 000	12 288.0	8 192	245 760
4.2	2 228 234	145.1	64.0	172.0	172.0	50 000	13 056.0	8 704	522 240
5	5 652 480	163.8	72.3	172.0	172.0	135 000	41 400.0	22 080	589 824
5.1	9 437 184	172.0	120.5	172.0	172.0	240 000	69 120.0	36 864	983 040

Tasot määrittelevät siis dekooderi toteutuksessa tarvittavan muistin ja laskentatehon määrän. Kun bittistreamille on määritelty tietty profiili ja taso niin dekooderin pitää pystyä käsittelemään kaikkia tasoja jotka ovat tämän alapuolella. Tasoissa on määritelty vain kuvan maksimikoko (luma) pel:inä. Miten nämä on jaettu horisontaalisesti ja vaakasuorasti niin sitä ei ole määritelty, paitsi että horisontaalinen ja vertikaalinen

ulottuvuus ei voi olla suurempi kuin neliöjuuri (8 x kuvan_koko). Esimerkiksi 720 x 1280 8 bitin kuva sisältää (4:2:0) 921 600 näytettä jolloin tämä kuuluu tasoon 3.1, samaan tasoon kuuluu myös XGA. Käyttäjällä on täysi vapaus valita aktuaaliset ulottuvuudet. Mikäli tietty kuvakoko on pienempi kuin tietyllä tasolle on määritelty niin framerate voi olla suurempi aina 172 kuvaan/sekunnissa saakka. Taulukkoa silmäiltäessä mielenkiintoinen seikka on se, että kuvamuistin koko on määritelty kilotavuina. Siis esimerkiksi tasolla 3.1 jo niinkin suuri kuin 6 750 kB. Pienimmilläänkin tämä on 148.5 kilotavua

7 Suorituskyky

H.264:n suorituskykyä on tutkittu myös paljon. Yksi monista tuloksista on esitetty Schäfer & al julkaisussa [SCHÄ03]. Tutkimuksessa vertailtiin MPEG-2, MPEG-4, H263++ ja H.264 kahdella eri esitysformaatilla[13]. Käytettyjä standarditestisekvenssejä oli 8 (4 kappaletta CIF ja 4 QCIF formaatissa). MPEG-2:sta käytettiin MP@ML (main profile/main layer), H.263++:sta HLP formaattia ja MPEG-4:stä ASP profiilia ja H.264:stä käytettiin main profiilia. Sekvenssin alussa tehtiin intrakoodaus, jokaisen peräkkäisen P kuvan välissä käytettiin kahta B kuvaa. B kuvia ei käytetty referenssikuvina, joten niiden rooli oli samankaltainen kuin MPEG-2:ssa. Oheisen kuvan 4.6 käyrästöön on piirretty testitulokset yhdestä CIF testivideosta (”Tempete”).



Kuva 15: *Eri koodekkien suorituskyky ”Tempete”-testivideolla [SCHÄ03], kuva julkaistu Thomas Wiegandin luvalla*

Oheisessa taulukossa on kuvattu keskimääräinen säästö suhteessa muihin koodekkeihin, kaikilla testeissä käytetyillä sekvensseillä Schaäfer et al mukaan. Tutkijat arvelevat merkittävän hyödyn saavutetun koska H.264:ssä käytetään kehittynyttä liikemallia ja sisältöön mukautuvaa aritmeettista koodausta.

Koodekki	MPEG-4 ASP	H.263 HLP	MPEG-2
H.264	38.62 %	48.80 %	64.46 %
MPEG-4 ASP	-	16.65 %	42.95 %
H.263 HLP	-	-	30.61 %

7.1 MPEG formaali verifiointitesti

Ns. formaali testiraportti [ISO03] koodekista H.264 perustuu standardiluonnokseen vuodelta 2003 ja tämän ns. referenssitoteutukseen. Testi oli ns. verifiointitesti, jolla pyrittiin selvittämään H.264:n kompression suorituskyky suhteessa aikaisempiin MPEG

[13] Käytetyt formaatiti olivat CIF, 360x288, 29.97Hz, progressiivisena ja QCIF 180 x 144, 15 ja 30 Hz.

standardeihin näiden tarkoitetuilla sovellusalueilla [OELB04, ISO03]. Testitulokseen vaikutti olennaisesti se, että toteutus ei ollut mitenkään optimoitu kompleksisuuden ja tehokkuuden suhteen eikä se sisältänyt FRExt laajennuksia. Kun taas vertailukohtana käytettyä MPEG-2:ta oli jo kehitetty lähes 10 vuotta. Testien aikaan H.264 sisälsi 3 profiilia, nimittäin baseline, main ja extended profiilit. Testeistä yksi oli suunnattu baseline ja loput 3 oli suunnattu main profiiliin. Testisekvenssit olivat jo alalla vakiintuneita sekvenssejä (eli Foreman, Head with Classes, Paris, PanZoom, Husky, Tempete, Footbal, Harbour ja Crew), joiden arvellaan tuovan hyvin esille koodekkien intra ja interominaisuuksia. Kuitenkin on huomattava, että testeissä ei käytetty ns. hollywood materiaalia. Oheiseen taulukkoon on kuvattu testeissä käytetyt profiilit ja bittinopeudet:

MD baseline	Koodekki	H.264 Baseline @L 2			
	Vertailu koodekki	MPEG-4/2 SP @L3			
	Resoluutio	CIF (352 x 288)		QCIF (176 x 144)	
	Sekvenssit	Foreman, Head with Glases, Paris, PanZoom			
	Kuvataajuus	15		10	
	Bittinopeudet kb/s	768, 284,192, 96		192, 96, 48, 24	
	Intra päivitysnopeudet	Ei rajoituksia			
MD Main	Koodekki	H.264 Main @L2			
	Vertalu koodekki	MPEG-4/2 ASP@L3			
	Resoluutio	CIF (352 x 288)		QCIF (176 x 144)	
	Sekvenssit	Mobile & Calendar, Husky	Tempete, Football	Mobile & Calendar, Husky	Tempete, Football
	Kuvataajuus	12	15	8	10
	Bittinopeudet kb/s	768, 384, 192, 96		192, 96, 48, 24	
	Intra päivitysnopudet	2 s			
SD main	Koodekki	H.264 Main @ L3			
	Vertalu koodekki	MPEG-2 MP@ML (MPEG-2 TM5 &HiQ)			
	Resoluutio	SD			
	Sekvenssit	Mobile & Calendar		Tempete, Football	
	Kuvataajuus	50 kenttää/s		60 kenttää/s	
	Bittinopeudet Mb/s	6, 4, 3, 2.25, 15 (vain H.264)			
	Intra päivitysnopudet	0.5 s			
HD main	Koodekki	H.264 Main @L4			
	Vertalu koodekki	MPEG-2 MP@HL (MPEG-2 TM5 & HiQ)			
	Resoluutio	720p60	1080i30	1080p25	
	Sekvenssit	Harbour, Crew	Stockholm Pan, New Mobile & Calendar	Vintage Car, Riverbed	
	Kuvataajuus	60	60 kenttää/s	25	
	Bittinopeudet Mb/s	20, 10, 6	20, 10	20, 10, 6	
	Intra päivitysnopudet	0.5			

Testit tehtiin DSCQS ja DSIS[14] menetelmällä, jossa arvosana annettiin 0 – 100, joista muodostettiin arvosanat 1 - 5 ns OS (standard opinion scale). OS arvosnat käsiteltiin tilastollisesti joista muodosttteiin keskiarvo (Mean Opinion Scale MOS) sekä luotettavuus intervallit. Testit tehtiin 4 laboratoriossa, nimittäin Fondazione Ugo Bordoni (FUB, DSIS), Istituto Superiore delle Comunicazioni e della Technologia delle Informazioni (ISCTI,

[14] DSIS menetelmässä katsojalle esitetään referenssi sekvenssi jota seuraa 2 – 3 sekunnin harmaa jakso jonka jälkeen esitetään tutkittava sekvenssi (tämä voidaan toistaa parikin kertaa). Tämän jälkeen pyydetään katsojaa antamaan arvosana tutkittavasta. Erona DSCQS:n on se, että jälkimmäisessä annetaan arvosana kullekin katsotulle sekvenssille.

DSIS), Munich University of Technology (TUM, DSCQS) sekä National Institute of Standards and Technology (NIST, DSCQS). Näin saatiin hieman kulttuurista riippuvia tekijöitä hieman minimoitua.

Testien tuloksena H.264 osoittautui olevan:

- MD baseline testeissä ainakin 2 kertaa tehokkaampi kuin MPEG-2 14 jaksossa 18:sta.
- MD main testeissä ainakin 2 kertaa tehokkaampi 18 tapauksessa 25:stä
- SD main testeissä vähintään 1.5 kertaa tehokkaampi 8 tapauksessa 12:sta verrattaessa MPEG-2 HiQ, 3: ssa tapauksessa H.264 oli 2 kertaa tehokkampi ja 1:ssä (mobile & calendar) jopa 4 kertaa. Sen sijaan ”Husky” sekvenssissä mitään suurempaa koodaustehokkuuden kasvua ei havaittu[15].
- HD main testeissä verrattaessa MPEG-2 HiQ saavutettiin 1.7 kertainen koodaustehokkuus 7:ssä tapauksessa 9:stä, joissa 3:ssa parannus oli kaksinkertainen ja 1 osoitti 3.3 kertaisen parannuksen.

Oheiseen taulukkoon on koottu HD testien tulokset (muut tulokset kiinnostunut lukija löytää dokumentista [ISO03]).

HD H.264 Main verrattuna MPEG-2 HiQ						
	720p60		1080i30		1080p25	
	Crew	Harbour	Stockholm Pan	Mobile & Calendar	River Bed	Vintage Car
20 Mb/s	T	T		T	T	T
10 Mb/s	2x	T	1x	T, 2x	> 1x	T, 2x
6 Mb/s	1.7x	T, 3.3 x	Ei mukana testeissä		> 1.7x	1.7x
HD H.264 main verrattuna MPEG-2 TM5						
20 Mb/s	T	T		T	T	T
10 Mb/s	2x	T	2x	T, 2x	>1x	T, 2x
6 Mb/s	1.7x	T, 1.7x	Ei mukana testeissä		>1.7x	1.7x

Selityksiä

T	Tällä nopeudella on saavutettu transparenttisuus
Nx	H.264:llä saavutettu N kertainen tehokkuuden lisäys
	Laatu erinomainen muihin koodekkeihin nähden
	Ainakin kaksinkertainen tehokkuuden lisäys
	1 – 2 kertainen koodaus tehokkuuden lisäys
	Sama tehokkuus kuin kaikilla vertailtavilla koodekeilla
	Tilastollisesti ei vakuuttava.

Suoritetut testit osoittivat, että H.264 on merkittävästi parempi kuin aikaisemmat standardikoodekit. Standardi on myös olennaisesti parantunut testien jälkeen. Mutta toisaalta koska standardi sisältää suuren joukon asetettavia parametreja kuten kaksisuuntainen ennustus, useampia referenssikuvia, 4 x 4 alilohkot, CABAC

[15] Monissa muissa testeissä kompressoitu “husky” sekvenssi on jopa parempi kuin alkuperäinen kompresoimaton [HANH03]! Sekvenssissä koira ”husky” juoksee detaljoidussa taustassa, kamera panoroi koiran liikkeitä

entropiakoodauksen jne. Testi ei edellyttänyt reaaliaikaista koodausta, jolloin voitiin hakea kaikista kombinaatioista paras mahdollinen. Reaaliaikaisessa koodauksessa taasen ei juurikaan ole mahdollista hakea systemaattisesti paras kombinaatio [OELB04].

7.2 Blu-Ray yhteisön tekemät testit

Wedi et. al ehdottavat kahta muutosta alkuperäiseen H.264 standardiluonnokseen vuodelta 2003, jotta subjektiivinen kuvanlaatu paranisi ja standardi olisi joustavampi [WEDI04a]:

- 8 x 8 kokonaislukumuunnos johon liittyy kvanttisointimatriisi
- kvanttisointi käyttäen adaptiivista kuollutta aluetta.

Syynä näihin lisäystarpeisiin oli Blu-ray Disc Fouders:n (BDF) tekemät testit ns. Hollywood materiaalille. Nämä osoittivat, että H.264 ei pystynyt toimittamaan HD elokuvissa parempaa kuvanlaatua kuin mitä MPEG-2 suurilla kompressionopeuksilla (yli 20 Mb/s) pystyi. Testien seurauksena BDF valitsi MPEG-2:n ainoaksi koodekiksi BRD-ROM:lle, sinilaser levyille. Myöhemmin syksyllä 2004 FRExt laajennusten tultua osaksi standardia niin myös H.264 on mukana uusimmassa sini-laser suosituksissa.

Testit tehtiin seuraavasti [WEDI04a]:

Testisekvenssit:	Eri Hollywood elokuvastudioiden toimittamia elokuvaleikkeitä
Sekvenssien formaatti:	1920 x 1080, 24 fps, 4:2:0, 8 bittiä
Erityisominaisuuksia:	Leikkeet sisälsivät filmirakeisuutta
Koodekit:	H.264 MP (referenssiohjelmisto, kiinteä QP), MPEG-2 MP@HL (muuttuva QP)
Bittinopeudet:	7, 12, 15, 20 ja 24 Mb/s

BDF:n johtamat subjektiiviset testit tehtiin kuudessa laboratoriossa. Tuloksena esitettiin seuraavat johtopäätökset [WEDI04]:

- Suuremmilla bittinopeuksilla (20 ja 24 Mb/s) testatuilla sekvensseillä MPEG-2 osoitti paremman subjektiivisen laadun kuin H.264/AVC
- H.264:n subjektiivinen kuvan laatu ei ole riittävä sini-laser levyille: Hienojakoinen tekstuuri ja filmin rae puuttuivat.
- Alemmilla siirtonopeuksilla (< 15 Mb/s) sekä H.264 että MPEG-2 kuvan laatua ei voitu hyväksyä epästabiilin ”hyppivän kohinan” takia.

Vaikka ISO formaalit verifiointitestit osoittivat, että H.264:llä voitiin saavuttaa transparentti laatu 20 Mb/s, niin kuitenkin erona BDF testeihin nähden oli käytetty testimateriaali, vakiintunette testisekvenssit versus ”hollywood”-materiaali. Koettuun laatuun vaikuttaa myös käytetty materiaali, BDF testeissä käytettiin alunperin filmipohjaista materiaalia joissa usein oli hienojakoinen tekstuuri.

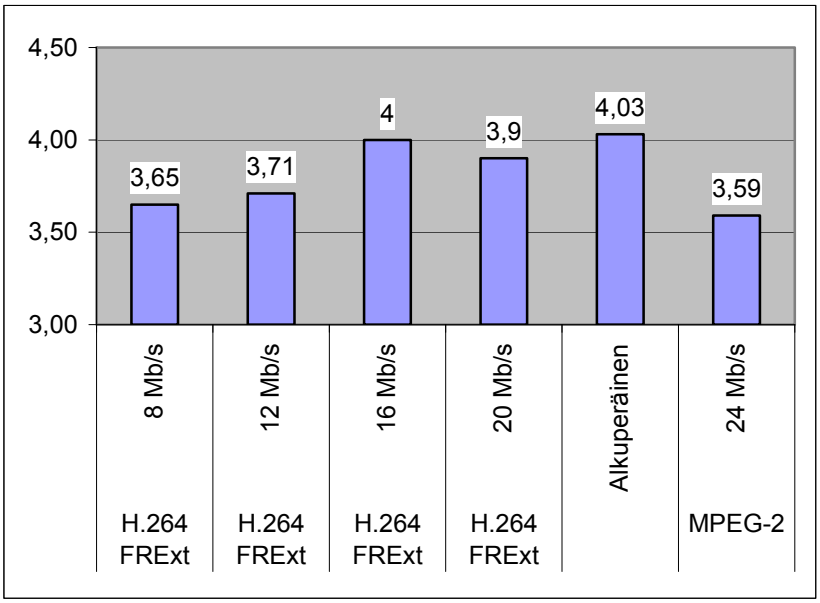
7.3 FRExt laajennukset

Wedi ja Kashiwagi dokumentoivat testit, missä H.264:ään oli otettu huomioon H.264 FRExt laajennukset [WEDI04b]. Testit tehtiin BDF:n johdolla toukokuun 14. 2004. Evaluaation tulokset esitetään mieluusti eri H.264 yleisesityksissä, siitäkin huolimatta, että testaustapaa ei kovin huolellisesti ole dokumentoitu (testien toistettavuus voi olla vaikeasti saavutettavissa).

Testit tehtiin ns. sokeana testinä. Sokeassa testissä arvioijat eivät tiedä mikä sekvenssi on kyseessä. Käytetyt koodekit olivat H.264 FRExt nopeuksilla 8, 12, 16 ja 20 Mb/s ja 24 Mb/s. Kooderilla käytettiin seuraavia asetuksia:

- 4:2:0, 8 bittiä
- alkuperäisten kuvien koko 1080p23.98
- enkkodattu 8 x 8 muunnoksella ja skaalausmatriisilla.

Oheiseen kuvaan on hahmoteltu eri sekvensseillä muodostettu keskimääräinen arvosana. Koska H.264 FRExt nopeudella 8 Mb/s sai keskimääräisen arvosanan 3.65 (melko hyvä – hyvä) asteikolla 0 - 5 ja MPEG-2 sai nopeudella 24 Mb/s arvosanan 3.59, niin usein on tehty sellainen johtopäätös, että H.264 on 3 kertaa parempi kuin MPEG-2. Mutta toisaalta materiaalina oli vain yksi leike (josta arvioijat pitivät 4.03 arvoisena), 16 Mb/s on parempi kuin 20 Mb/s sekä ero 12 Mb/s ja 8 Mb/s välillä on vain 0.06 yksikköä, niin tällaisten johtopäätösten tekeminen voi olla hieman nopeaa. Sitä paitsi tuloksille ei laskettu tilastollista luotettavuusintervalliakaan.



Kuva 16: FRExt laajennusten jälkeinen arviointi eri nopeuksilla.

7.4 WMV9 ja H.264:n vertailu

Nokian tutkijat T. Mustonen, M. Pölönen, A. Siren ja J. Oja ovat tehneet subjektiiviset testaukset microsoftin video koodekin 9 (WMV 9) ja H.264:n välillä [MUST03]. Tutkimus tehtiin kahtena tutkimussarjana. Käytetyt sekvenssit olivat osittain samoja mitä yleensä käytetään ja ensimmäisessä sarjassa 3 uutta:

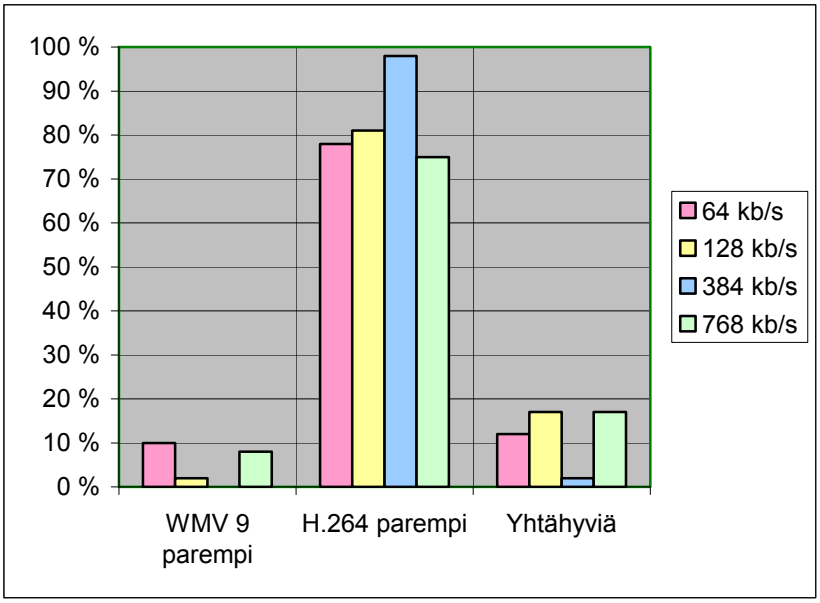
	Leike	Kesto	Formaatti	Frame rate kuvaa/s	Kuvaus
Sarja 1	Axe	51 s	QCIF	15	Mainos, liikettä, värejä
	Ceal	21 s	SQCIF	25	Tummasävyinen musiikkivideo
	Football	8 s	QCIF	15	Nopeat kameran ja kohteen liikkeet, värejä
	Foreman	10 s	QCIF	15	Puhuva pää, geometrisiä muotoja nopeat kameran liikkeet
	Musa	15 s	QCIF	12	Musiikkivideo, animaatiota
Sarja 2	Canoe	7 s	CIF	15/30	
	Football	8 s	CIF	15/30	Sama kuin yllä
	Foreman	8 s	CIF	15/30	Sama kuin yllä
	Tempete	8 s	CIF	15/30	Kamera zoom, spatiaalisia yksityiskohtia nopea ja satunnainen liike

QCIF = 176 x 144, SQCIF = 128 x 96 ja CIF = 352 x 288 kaikki sarjan 1 kokeet tehtiin nopeuksilla 64 ja 128 kb/s ja sarjan 2 nopeuksilla 384 ja 768 kb/s

Sarjan 1 testit tehtiin sekä asiantuntija lausuntoina ns parivertailuna (ITU-T R. P 910 standardin mukaisesti). Tässä menetelmässä kaksi tutkittavaa leikettä (WMV 9 ja H.264) näytetään vierekkäin, leikkeet A ja B. Katsojalta kysytään arvostelua siten, että hän voi valita

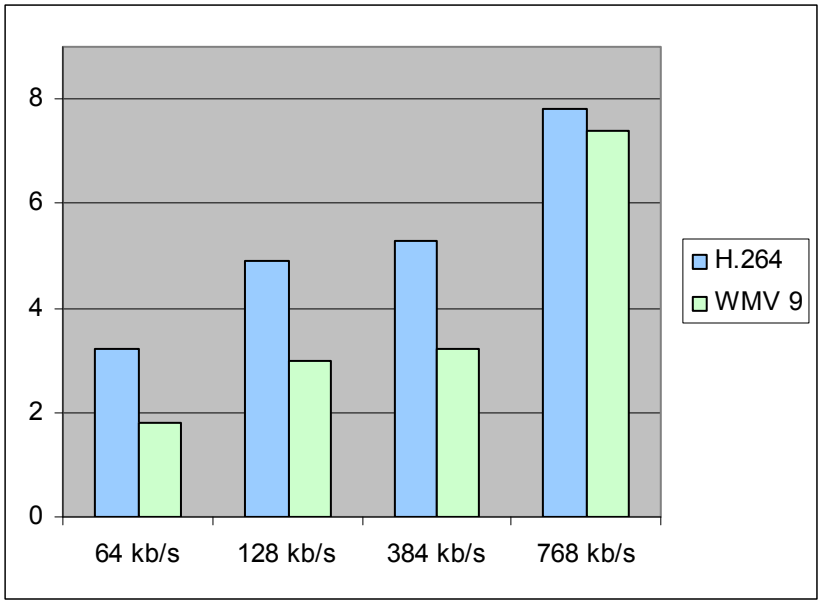
- A on parempi kuin B
- A on yhtä hyvä kuin B
- B on parempi kuin A.

Arvostelijat voivat katsoa pareja useampaan kertaan ennen arvosanan antamista. Leikkeiden sijainnit olivat myös satunnaisesti valittu. Oheiseen kuvasarjaan on kuvattu tulokset (julkaistu Terhi Mustosen luvalla):



Kuva 17: Koodekkien WMV 9 ja H.264 parivertailu.

Testisarja 2 tehtiin ns. Single stimulus testinä. Tässä menetelmässä leikkeet esitetään yksi kerrallaan ja pyydetään katsojalta arvosanaa. Arvosana annettiin jatkuvalla asteikolla 0 – 9 (huono – erinomainen). Tuloksista laskettiin myös luotettavuus intervallit. Oheisiin kuviin 17 ja 18 on piirretty Mustonen et al. tutkimusten tuloksia.



Kuva 18: Koodekkien WMV 9 ja H.264 arviointi.

7.5 Suorituskykyyn vaikuttavia seikkoja

Puri et al implementoivat tänään jo hieman vanhaa ns. public domain H.264 referenssiohjelmistoa JM 6.1e. He raportoivat seuraavat suorituskykyyn vaikuttavia seikkoja [PURI04]:

- Kuten aikaisemmissa standardeissa on havaittu niin B kuvat muodostavat keskeisen tekijän koodaustehokkuussäästöissä. Käytettäessä 1 B kuvaa saavutetaan 10 – 24 % säästö koodaustehokkuudessa 2 B kuvalla 14 – 34 % suhteessa tapaukseen missä ei käytetä ollenkaan B kuvia.
- H.264 sallii suuren määrän eri toimintatapoja, hyvä toimintatavan valinta on toiseksi tärkein B kuvien valinnan jälkeen suorituskykyyn vaikuttava tekijä. Käytettäessä suurempia määriä (esim. 2) B kuvia niin oikean toimintatavan valinta muodostuu vielä tärkeämmäksi. Vertailtaessa ei-RD optimoitua toimintatapaa RD optimoituun käytettäessä 1 B kuvaa saavutettiin 5 – 13 % koodaustehokkuuden kasvu ja kahdella B kuvalla 10 – 22 % kasvu.
- CABAC:lla saavutettu etu voi olla myös tärkeä ja Puri et. al. mukaan vaihtelee sekvenssin ominaisuuksien ja koodauksessa käytettyjen bitti-nopeuksien mukaan. Keskimäärin saavutettiin 7 % koodaustehokkuuden parannus (6.8 % 1:llä B kuvalla ja 7.2 % 2:lla kuvalla).
- Moninkertaisia referenssikuvia käytettäessä saavutettu etu on tyypillisesti alhainen mutta toisinaan näillä voi olla merkitystä koska referenssikuvien määrän vaikutus on hyvin näkymä riippuvainen. Lisäksi moninkertaisilla referenssikuvilla on suurempi vaikutus kun B kuvia on vähän. Esimerkiksi käytettäessä 1 B kuvaa 3:lla referenssikuvalla suhteessa 1:een kuvaan on 2 – 8 % vaikutus. 5 referenssikuvaa tuotti 2 – 10 % edun. 2:lla B kuvalla taasen käyttämällä 5 referenssikuvaa suhteessa 1:een tuotti vain 1- 6 % edun.
- De-blocking suodattimen suoranainen vaikutus koodaustehokkuuteen on kohtalaisen alhainen (tyypillisesti 0.5% - 2%), kuitenkin sillä on suurempi

merkitys silmin havaittavaan kuvan laatuun. Suodatin kyllä pehmentää lohkorajoja mutta koska muunnosalueen koodauksen koko oli vain 4 x 4, niin joitakin hienoja yksityiskohtia voidaan menettää erityisesti alhaisemmilla nopeuksilla[16].

- Liikekompensoinnin suhteen käyttämällä suurempaa kuin kuin 8 x 8 lohkokokoa kaikista mahdollisista vaihtoehdoista aiheutti 1 B kuvan tapauksessa keskimäärin 1.75 % hävikin ja 2 B kuvan tapauksessa 2.5 %.
- Lomitetun videon koodauksessa kuva adaptiivinen kuva/kenttä koodaus tuotti 6 – 32 % edun puhtaasti kuva koodaukseen nähden kun taas makrolohko adaptiivien kuva/kenttä koodaus tuotti 11 – 34 % edun.
- Kaikki muut menetelmien ja enkoodauksen vaihtoehdot tuottivat hyvin markinaaliset edut [PURI04].

[16] Itse asiassa yksi suurimmista syistä 8 x 8 muunnoksen käyttöönottamiseksi FRExt laajennuksissa oli hienon hienojen yksityiskohtien (kuten filmin rakeisuuden) menettäminen.

Kirjallisuusviitteet

[AHME74] Ahmed N., Natarajan T., Rao K.R.: Discrete Cosine Transform, IEEE Trans. Computers vol C 23, Jan 1974

[HANH03] Hanhijärvi J.: Kuvien ja videon kompressiosta, YLE tekniikka, 2003 Helsinki, ISBN 951-43-0845-X

[HORO03] Horowitz M., Joch A., Kossentini F.., Hallapuro A.; H.264/AVC baseline profile decoder complexity analysis, IEEE Transactions on circuits and systems for video technolgy, vol 13, no 7, July 2003

[ISO03] ISO: Report of the formal verification tests on AVC (ISO/IEC 14496-10|ITU-T Rec. H.264), ISO/IEC JTC1/SC29WG11, MPEG2003/N6231, December 2003, Waikoloa

[KARC03] Karczewicz M., Kurceren R.: The SP- and SI-frames design for H.264/AVC, IEEE transactions on circuits and systems for video technology, vol 13, no 7, July 2003

[LAPP03] Lappalainen V., Hallapuro A., Hämäläinen T.: Complexity of optimized H.26L video decoder implementation , IEEE transactions on computer graphics and systems for video technology, vol 13 no 7 2003

[LIST03] List P., Joch A., Lainema J., Bjøntegaard G., Karczewicz M.: Adaptive deblocking filter, IEEE Transactions on circuits and systems for video technology, vol 13, no. 7 July 2003

[LUTH03] Luthra A., Sullivan G., Wiegand T. : Introduction to the special issue on the H.264/AVC video coding standard, IEEE Transactions on circuits and systems for video technology, vol 13, no 7, July 2003

[MALV03] Malvar H., Hallapuro A., Karczewicz M., Kerofsky L.: Low-complexity transform and quantization in H.264/AVC, IEEE transactions on circuits and systems for video technology, vol 13, no 7, July 2003

[MARP03] Marpe D., Schwarz H., Wiegand T.: Context-based adaptive binary arithmetic coding in the H. 264/AVC video compression standard, IEEE transactions on circuits and systems for video technology vol 13. no 7, July 2003

[MUST04] Mustonen T., Pölönen M., Sirén A., Oja J.: Subjective Video Quality of WMV9 and H.264/AVC, Comparison of Codec Performance at Four Target Bit Rates, TDoc: AHVIC-004, 3GPP Video Codec Ad-Hoc Meeting, October 28 -30, 2003

[NILS05] Nilsson M.: The Advanced Video Coding Standard, IEE International conference on Visual Information Engineering, Glasgow 2005.

[OELB04] Oelbaum T., Baroncini V., Tan T., Fenimore C.: Subjective quality assessment of the emerging AVC/H.264 video coding standard, IBC 2004 Conference Publication

[OSTE04] Ostermann J., Bormans J., List P., Marpe D., Narroschke M., Pereira F., Stockhammer T., Wedi T.: Video Coding with H.264, Tools, Performance, and Complexity, IEEE Circuits and System Magazine 2004

[PENN88] Pennebaker W., Mitchell J., Langdon G., Arps R.: An overview of the basic principles of the Q-Coder adaptive binary arithmetic coder, IBM J. Res Dev. Vol 32 1988

[PURI04] Puri A., Chen X., Luthra A.: Video coding using the H.264/MPEG-4 AVC compression standard, Signal Processing : Image Communication 19, 2004, Elsevier

[RIBA03] Ribas-Corbera J., Chou P., Regunathan S.: A generalized hypothetical reference decoder for H.264/AVC IEEE Transactions on circuits and systems for video technology, vol 13, no 7, July 2003

[RICH03] Richardson I.: H.264 and MPEG-4 Video Compression: Video Coding for Next-generation Multimedia, JohnWiley & sons September 2003, ISBN 0-470-84837-5

[RICH04] Richardson I.: H.264 White papers, at http://www.vcodex.com/h264.html

[RISS89] Rissanen J., Mohiuddin K.: A multiplication free multialphabet arithmetic code, IEEE transaction communication vol 37 February 1989

[SCHÄ03] Schäfer R. , Wiegand T., Schwarz H.: H.264/AVC standard, EBU Technivcal review, January 2003

[SULL04a] Sullivan G., Wiegand T.: Video Compression, from concepts to the H.264 Standard, Proceedings of the IEEE, December 2004

[SULL04b] Sullivan G., Topiwala P., Luthra A.: The H.264/AVC Advanced Video Coding Standard: Overview and Introduction to the Fidelity Range Extensions, SPIE Confernce on Applications of Digital Image Processing XXVII, Special session on Advances in the New Emerging Standard H.264/AVC, August 2004

[SUNN05] Sunna P.: AVC/H.264 – an advanced video coding system for SD and HD broadcasting, EBU technical review – April 2005

[STOC03] Stockhammer T., Hannuksela M., Wiegand T.: H.264/AVC in wireless environments, IEEE Transactions on circuits and systems for video technology, vol 13, no. 7, July 2003

[TU05] Tu Yu-Kuang, Yang Jar-Ferr, Sun Ming-Ting, Tsai Yesheng T.: Fast variable-size block motion estimation for efficient H.264/AVC encoding, Singal Processing Image Communication, Elsevier 2005

[WEDI04a] Wedi T., Kashiwagi Y., Takahashi T.: H.264/AVC for Next generation Optical Disc: A Proposal on FRExt Profiles, Document JVT-KO25r1, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG, Munich March 2004.

[WEDI04b] Wedi T., Kashiwagi Y.: Subjective quality evaluation of H.264/AVC FRExt for HD movie content, Document JVT-LO33, Joint Video Team (JVT) of ISO/IEC MPEG & ITU-T VCEG

[WENG03] Wenger S.: H.264/AVC Over IP, IEEE Transactions on circuits and systems for video technology vol 13, no 7, July 2003

[WIEG03a] Wiegand T., Sullivan G., Bjøntegaard G., Luthra A.: Overview of the H.264/AVC video coding standard, IEEE transactions on circuits and systems for video technology, vol 13, no 7, July 2003

[WIEG03b] Wiegand T., Schwarz H., Joch A., Kossentini F., Sullivan G.; Rate-constrained coder control and comparison of video coding standards, IEEE Transactions on circuits and systems for video technology, vol 13, no 7, July 2003

[WIEN03] Wien M.: Variable block-size transforms for H.264/AVC, IEEE transactions on circuits and systems for video technology, vol 13, no 7, July 2003

[WU05] Wu D., Pan F., Lim K., WU S., Li Z., Lin X., Rahardja S., KO C.: Fast Intermode Decision in H.264 Video Coding, IEEE Transactions on Circuits and Systems for Video Technology vol 15. no 6, July 2005