

TARTU ÜLIKOOL
Füüsika-keemiateaduskond
Eksperimentaalfüüsika ja tehnoloogia instituut

LAAS TOOM

DVB VIDEOSIGNAALI DCT SIMULEERIMINE ARVUTIL

Magistritöö

Juhendaja: dotsent, füüs.-mat. knd ANDO OTS

Tartu 2005

Sisukord

1 SISSEJUHATUS.....	4
2 MPEG-2 KODEERIMINE.....	7
2.1 MPEG-2 pakkimise põhimõtetest.....	8
2.2 Värviteisendus.....	11
2.2.1 RGB ja YCbCr.....	11
2.2.2 Värvivähendus, 4:2:0 ja 4:2:2 meetodid.....	11
2.3 Diskreetne koosinusteisendus (DCT).....	13
2.3.1 Diskreetne Fourier' teisendus (DFT).....	13
2.3.2 Diskreetne lainekeseteisendus (DWT).....	14
2.3.3 Ühemõõtmeline diskreetne koosinusteisendus (DCT).....	15
2.3.4 Kahemõõtmeline diskreetne koosinusteisendus (2D-DCT).....	18
2.4 Kvantimine.....	20
2.5 Sikk-sakk esitus.....	22
2.6 Muutuva pikkusega kodeerimine.....	22
2.7 Liikumise ennustamine ning MPEG-2 kaadrite tüübid.....	24
2.8 MPEG-2 profiilid ja tasemed.....	26
3 VALMINUD TARKVARA KIRJELDUS.....	28
3.1 Programmi DVBSim graafiline liides ja võimalused.....	29
3.2 Moodul 1: Ühemõõtmeline koosinusteisendus.....	30
3.3 Kahemõõtmelise DCT näide.....	33
3.4 Kvantimine.....	40
4 DIGITAALSE VIDEOEDASTUSE PÕHIMÕTTED.....	45
4.1 DVB süsteemi ülesehitus.....	45
4.1.1 Kohandamine ja energia hajutamine.....	46
4.1.2 Välimine kodeerimine.....	47
4.1.3 Baitide ringvaheldi.....	48
4.1.4 Sisemine kodeerimine.....	48
4.1.5 Põhiriba vormimine.....	49
4.1.6 QPSK modulaator.....	49
5 KOKKUVÕTE.....	51
6 KASUTATUD KIRJANDUS.....	52
7 SUMMARY.....	54
8 KASUTATAVAD LÜHENDID.....	55
9 LISA 1: PROGRAMMI KOOD.....	57
9.1 Pakett dvbsim.....	57
9.1.1 DVBSim.java.....	57
9.1.2 MainProgram.java.....	57
9.1.3 DCT.java.....	58
9.1.4 MatrixPanel.java.....	60
9.1.5 TablePanel.java.....	64
9.1.6 IntegerEditor.java.....	66
9.2 Pakett dvbsim.dct1d.....	68
9.2.1 DCT1D.java.....	68
9.2.2 VectorPanel.java.....	74
9.2.3 DoubleVectorPanel.java.....	78
9.2.4 SisendiHiireAdapter.java.....	80
9.2.5 KomponentiValikListener.java.....	82

9.2.6	ControlListener.java.....	82
9.2.7	CheckBoxAdapter.java.....	83
9.2.8	BigPixelPanel.java.....	84
9.3	Pakett dvbsim.lihtnedct.....	86
9.3.1	LihtneDCT.java.....	86
9.4	Pakett dvbsim.kvantimine.....	97
9.4.1	Kvantimine.java.....	97
9.4.2	ImagePanel.java.....	111
9.4.3	ClickableImagePanel.java.....	111
9.4.4	ZigZagPanel.java.....	113
9.4.5	ClickableImageMouseListener.java.....	116
9.4.6	ButtonAdapter.java.....	117
9.4.7	KomponentValikListener.java.....	118
9.4.8	KvaliteetListener.java.....	119
9.4.9	KvandiValikListener.java.....	119
9.4.10	PildiValikListener.java.....	120
9.4.11	CheckBoxAdapter.java.....	120
9.4.12	DCKvaliteetListener.java.....	121
10	LISA 2: CD VALMINUD TARKVARAGA.....	122

1 SISSEJUHATUS

Tänapäevase infoühiskonna kiire areng pakub inimestele erinevaid digitaalseid teenuseid, nagu internet, e-post jne. Järjest enam digitaliseeritakse ka valdkondi, mis otseselt ei puutu infotehnoloogiasse, pakkudes kasutajatele e-valimiste, e-mediitsiini ja paljude muude elualade veebikeskkondi. Juba mitmeid aastaid on võimalik poodidest ja videolaenutustest lisaks videokassetidele saada filme ka digitaalsetel andmekandjatel ning ilmselt pole kaugel aeg, mil tavapärased videokassetid kasutuselt kaovad, nagu seda on praktiliselt juba teinud helilindid ja -kassetid.

Siiski on tavainimesteni jõudnud üsna vähe infot digitaaltelevisiooni võimalustest ja kasutuselevõtust. Paljud kodukasutajad ei tea sedagi, et satelliittelevisiooni vastuvõtmiseks on vaja eraldi seadet just seetõttu, et antennist tulev info on digitaalne ja televiisoriga ühendamiseks on vajalik see analoogseks teisendada. Veelgi vähem osatakse seda tehnoloogiat seostada terminiga DVB (*Digital Video Broadcasting*), mis kujutab endast 90. aastate keskel Euroopas algatatud ning ülemaailmselt kasutusele võetud digitaaltelevisiooni standardite kogu. Viimasel ajal on hoogustunud lisaks SAT-TV'le ka maapealse televisiooni digitaliseerimine. Ka Eestis on Tallinna lähiümbruses võimalik juba 2004. aasta algusest spetsiaalse seadme abil eestikeelset digitaaltelevisiooni vastu võtta ja aastaks 2015 on plaanitud nii Eestis kui Euroopas analoogtelevisiooni täielik väljalülitamine [1].

Kõik see tingib vajaduse ülikoolides tudengitele digitaaltelevisiooni põhimõtete õpetamiseks. Tartu Ülikooli telekommunikatsiooni aluste loengu kuulajateks on teise kursuse tudengid, kellel puudub varasem kokkupuude telekommunikatsiooni ja digitaaltelevisiooniga. Seetõttu on loengu (ja ka selle juurde kuuluva praktikumi) eesmärgiks anda ülevaade raadio-, tele- ja mobiilside põhimõtetest ja seadmetest. Kursuses käsitletakse signaalide moduleerimist, saatmist ja vastuvõttu, stereosignaali ja TV-kujutise töötlust, digitaalsignaali kodeerimist ja tihendamist info edastamisel.

Analoogsignaali töötluste puhul on võimalik kasutada eriseadmeid loengutes teema edastamiseks ja tudengitel praktikumis, mille kohta on valminud ka tööjuhend [2]. Nimetatud töövahendid on spetsiaalselt õppetööks valmistatud. Digitaalsignaali töötluste õpetamiseks aga vajalikud seadmed Tartu Ülikoolil puuduvad. Seetõttu toimub hetkel antud valdkonna esitamine vaid loengukorras, näidates jutule lisaks valemeid, blokkiskeeme ja iseloomulikke pilte

projektoril.

Telekommunikatsiooni aluste loeng käsitleb digitaaltelevisiooni just DVB, eriti satelliittelevisiooni seisukohast (jättes kõrvale kõik teised erinevad digitaaltelevisiooni standardid, mis on väljaspool Euroopat arendatud), keskendudes põhiliselt pilditöötluse osale ehk MPEG (*Motion Pictures Experts Group*) standardile ja seetõttu on vaadeldavateks digitaalsignaali töötlemise põhitöövahenditeks diskreetne koosinusteisendus (DCT) ja kvantimine, kuna need on olulisimad osad signaali entroopia vähendamises ja andmete tihendamises. Loengu kuulajaskonnal on küll läbitud matemaatilise analüüsi kursused, kuid sellest hoolimata on enamusel neist ettekujutus koosinussignaalist kui „mingist perioodilisest võnkumisest“ ja puudub sügavam arusaam kasvõi sel tasemel, et mis tulemuse annab koosinuse summeerimine üle terve perioodi. Seetõttu põhjustab valemeis koosinuse esinemine olukorra, kus tudengid õpivad vajalikud valemid pähe, neist ise aru saamata (tihti isegi ei üritata aru saada).

Loengumaterjalides toodud näitlikustavad pildid ja joonised küll selgitavad teemat lähemalt, kuid kuna puudub võimalus visuaalselt jälgitavate praktiliste näidete („elusate piltide“) kasutamiseks, siis jääb ka see ikkagi väheseks – oleks vaja vahendit, millega saaks tudengitele anda olulisematest põhimõtetest visuaalne ja kvalitatiivne arusaam.

Käsitlev valdkond on alles arendamises ja isegi kõik valmivad standardid on saadaval vaid tasuliselt, seetõttu on ka kogu õppematerjal enamasti ülikoolide sisemiseks kasutamiseks, mida avalikult ei jagata. Eriti suur puudus on näitlikest õppevahenditest ja tarkvarast, millistest on leida vaid üksikuid (enamasti liialt vähese funktsionaalsusega) programmijuppe ja nendest korraliku õppevahendite komplekti koostamine oleks samaväärne töö uue tarkvara valmistamisega, kuna vahelt puudu jäävad osad tuleks nagunii ise programmeerida, sealjuures ületada muuhulgas ka erinevatest kasutajaliidese keeltest tulenevad takistused – põhiline osa saadaolevast tarkvarast on saksakeelne, mida loengu kuulajaskond reeglina ei valda.

Seetõttu on antud töö eesmärgiks koostada tarkvara, mille abil on lektoril võimalik loengu jooksul kuulajatele demonstreerida digitaalse pilditöötluse põhimõtteid. Kuna DVB standard kasutab sisemiseks videoformaadiks MPEG-2 standardit, siis keskendub valminud tarkvara just selle standardi pilditöötluse aspektidele, pakkudes seega võimalust:

- visualiseerida diskreetse koosinusteisenduse rakendamist

- laiendada eelmist punkti kahemõõtmelisele juhule
- uurida kvantimise põhimõtteid ja parameetrite tähtsust.

Käesolev töö kirjeldab esmalt MPEG-2 standardit ja selle infotihendamise meetodeid, selgitades ka miks üldse on video pakkimine vajalik. Seejärel analüüsitakse valminud tarkvara ja lõpuks kirjeldatakse ka DVB süsteemi põhimõtteid DVB-S ehk satelliittelevisiooni näitel.

2 MPEG-2 KODEERIMINE

Euroopas kasutatava tavateleviseiooni standardi PAL (*Phase Alternation Line*) süsteemis televisioonipildi mõõtudes 720x576 punktilise pildi maht 24-bitise värvisügavuse puhul on 9,4 Mbit. Pakkimata video puhul on andmemahud juba kümneid kordi suuremad: samasuure pildi 25 kaadrit sekundis edastamiseks kulub 230 Mbit/s. Sellise andmehulga puhul mahutaks üks CD vaid 24 sekundit filmi ja DVD ligikaudu 3 minutit. Ühtlasi on selline andmehulk kasutuskõlbmatu ka DVB-S süsteemides, mille maksimaalne andmeedastuskiirus on 41,570 Mbit/s [3] või kuni 55 Mbit/s [4]. Edastussüsteemide (satelliidid, maapealne ringhääling) puhul on lisaks videopildile vaja edastada ka veakorrektsiooniinfo, et vältida eetrihäiretest tulenevaid bitivigu. Eestis kasutatavate parameetrite puhul on kasutatavaks edastuskiiruseks 19 – 24 Mbit/s [1]. Seega on ülioluline suuta tihendada digitaalvideot selliselt, et ka väiksemate andmeedastuskiiruste puhul oleks võimalik rahuldava kvaliteediga pilti näha.

Seetõttu on läbi aegade välja töötatud erinevaid andmete pakkimise meetodeid. Kasvõi näiteks PAL televisioonis on sagedusriba kitsendamise huvides võetud kasutusele poolkaadrite süsteem, millele on samas sagedusalas lisatud ka värviinfo, kasutades faasimodulatsiooni. Lisaks on värvide edastamisel kasutatav ribalaius 1,5 MHz (võrrelduna heleduse 5 MHz ribaga), mis tähendab, et värvide detailsuse vähendamise arvelt hoitakse eetriressurssi kokku. Ka seda võib vaadelda andmete tihendamisena.

Kaheksakümnendate lõpus hakati välja töötama ka ühtset standardit digitaalse video pakkimiseks – see sai endale nimeks MPEG (*Motion Pictures Experts Group*) ja kasutas kujutise pakkimise baasmeetodina juba varem valminud JPEG (*Joint Photographic Experts Group*) standardile analoogilist lähenemist.

Esimene MPEG standard (nüüdse nimega MPEG-1) oli mõeldud digitaalsete andmekandjate jaoks (eriti CD-ROM) ning võimaldas maksimaalset andmekiirust 1,5 Mbit/s [5]. Paar aastat hiljem (so juba enne MPEG-1 avaldamist 1993. aastal) alustati teise, suuremaid edastuskiirusi ja poolkaadreid pakkuva standardi väljatöötamist, mis sobiks ka televisiooni edastamiseks ning aastal 1994 see ka standardiseeriti MPEG-2 nime all. See standard võimaldab andmeedastust 4 Mbit/s kuni 100 Mbit/s, pakkudes erinevaid profiile ja tasemeid, et edastamise parameetrid saaks sobitada programmi vajadustega [6].

Siinkohal tekib õigustatud küsimus: mis otstarbel toetab MPEG-2 standard nii suuri andmeedastuskiirusi, kui kogu standardi mõte on andmete tihendamine? Põhjus on aga selles, et

MPEG-2 väljatöötamisel arvestati ka professionaalsete filmistuudiotel vajadustega ja nimetatud 100 Mbit/s andmekiirus on kasutusel 1920x1152 pikslilise kõrgekvaliteedilise video puhul, mida tavaolukorras digitaaltelevisiooni kaudu ei edastatagi.

2.1 MPEG-2 pakkimise põhimõtetest

Tihendamata video sisaldab mitmesugust infoliasust, mille vähendamise arvelt saab infot tihendada. Liasuse liigid on:

- geomeetiline – naaberpikslite väärtused ei ole (enamasti) üksteisest sõltumatud, kuna nad kuuluvad mingi suurema objekti koosseisu (näiteks taevast kujutavas kaadris sinised pikslid)
- ajaline – üksteisele järgnevad kaadrid on sarnased (va juhul, kui toimus stseeni vahetus)
- entroopia – mittejuhuslikes digitaliseeritud signaalides esinevad mõned väärtused tihemini kui teised. Seda teades võib sagedasemad väärtused kodeerida lühema koodiga (nt morse kood)
- psühho-visuaalne – inimese silma ja aju omapärade ära kasutamine; jaguneb omakorda kaheks: geomeetrilise eraldusvõime piir ja ajalise eraldusvõime piir.

Loetletud liiasuste tundmaõppimise abil on arendatud mitmed meetodid videost info väljajätmiseks, ilma et *näiline* kvaliteet langeks.

Osutub, et selline infotihendamine on väga efektiivne ja nii on Eesti digitaaltelevisioonis kasutatavate parameetrite puhul võimalik ühe analoogtelevisioonikanali jaoks vajamineva ribalaiusega edastada 4-5 digitaaltelevisiooni programmi, lisaks veel mitmesuguseid muid teenuseid, nagu elektrooniline telekava või mitmekeelsed subtiitrid [7].

MPEG-2 kodeerimises on väga paljuski tegemist inimaju „petmisega“, kasutades ära silma või aju puudusi. Inimese silm koosneb rohkem kui 100 miljonist kepikesest, mis on paigutatud üle terve võrkkesta, kattes ka äärealad. Kepikesed ei reageeri värvile, kuid on eriti tundlikud nõrgale valgusele, tehes võimalikuks nägemise pimedas. Silma läätse fookuses asub väga väikesel alal umbes 6 miljonit kolvikest, mis reageerivad erinevatele värvidele ja ka üldisele heledusele [8].

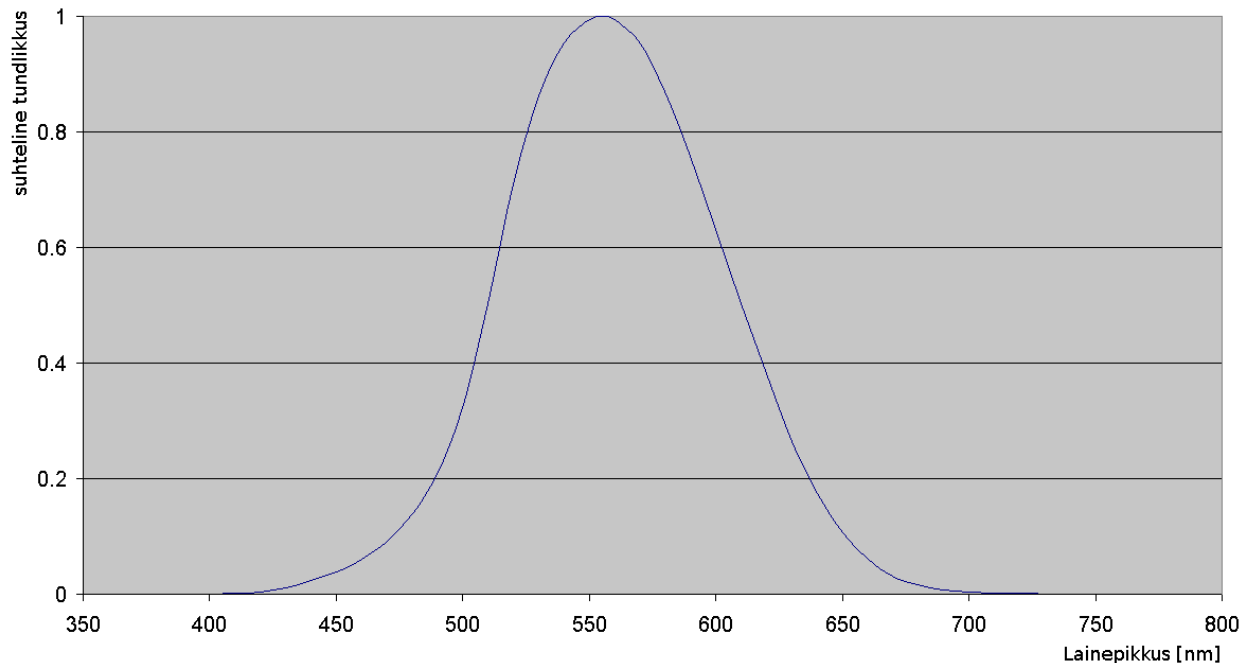
Põhimõtteliselt tähendab see, et inimene on võimeline värve tajuma vaid vaatevälja keskosas, ülejäänud alade „värvimisega“ tegeleb aju nägemiskeskus.

Joonisel 1 on toodud normaalse inimsilma tundlikkus heledusele [9]. On näha, et kõige suurem tundlikkus on lainepikkuse 550 nm juures, mis jääb rohelisse piirkonda. Vaadeldes ka joonisel 2

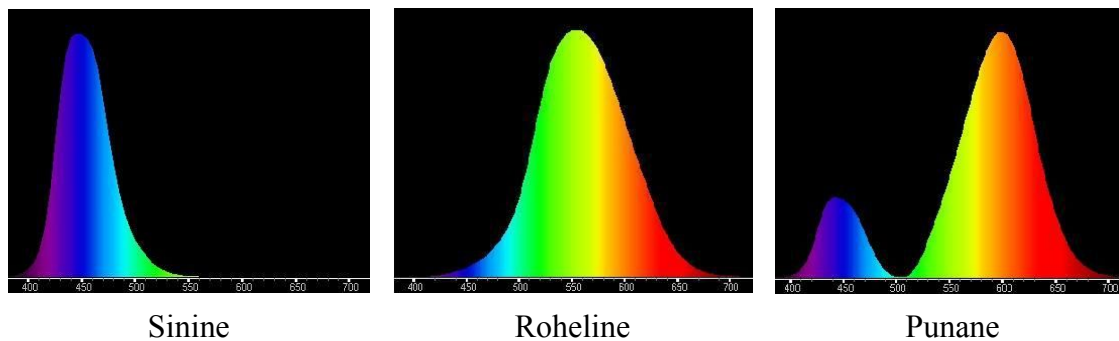
toodud graafikuid [10] silma tundlikkusest erinevatele värvidele, saame selgust, miks kasutatakse televisioonis värvikomponentidest heleduse arvutamiseks valemit (2.1) [2]:

$$Y = 0,3 R + 0,59 G + 0,11 B \quad (2.1)$$

Sellest valemist on näha, et enamus heleduse infost võetakse rohelisest värvist, sinist aga kõige vähem, kuna osa sellest spektrist sisaldub ka punase värvuse aistingus (nagu jooniselt 2 näha).



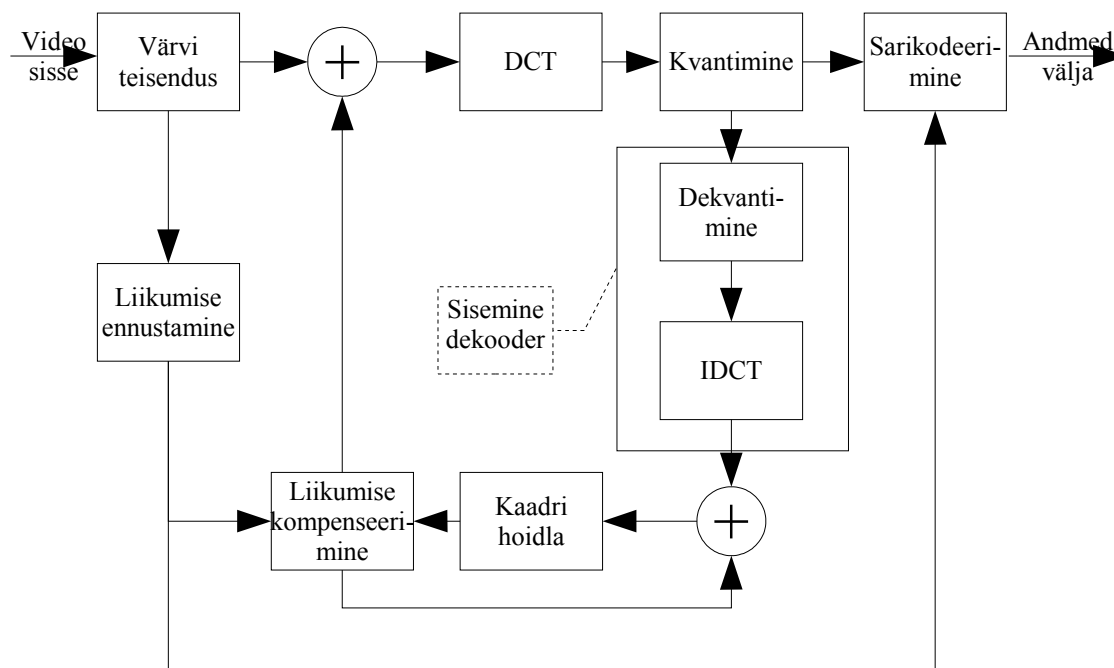
Joonis 1: Silma tundlikkus heledusele [9]



Joonis 2: Erinevaid värvusaistinguid tekitavate kolvikeste tundlikkuspäikkonnad. [10]

Allikas [8] kirjeldab ka katseid, kus on uuritud kolvikeste poolt tekitatavaid elektrilisi signaale erinevatele valgustele reageerides ja need annavad alust arvata, et ka inimese silm teisendab RGB värviinfo ümber heleduse ja kahe värvidiferentsignaali ühendiks. Kui nende signaalide selline esinemine tõestust leiab, siis annab see korrektse selgituse, miks inimsilm on heledusele märgatavalt tundlikum kui värvusele.

Lisaks inimese taju otseste nõrkuste ärakasutamisele rakendatakse MPEG-2 pakkimises veel ka teiste loetletud infoliasuste eemaldamise meetodeid. Joonisel 3 on toodud MPEG-2 süsteemi põhimõtteskeem.



Joonis 3: MPEG-2 süsteemi põhimõtteskeem

MPEG süsteemi sisenevale videole tehakse esmalt värviteisendus, et saada RGB komponentidest heleduse ja värvuse info eraldi. Saadud tulemusele tehakse DCT (*Discrete Cosine Transform*) teisendus ning kvantimine, misjärel läbitakse sisseehitatud dekodeer. Dekodeeritud pilti võrreldakse samaaegselt teostatud liikumise ennustusega ning ennustuse vead saadetakse uuesti DCT teisendusse, kuid seekord suunatakse teisenduse tulemused sarikodeerimisse ning väljundisse. Sarikodeerimisse lähevad ka liikumise ennustuse tulemused. Sisemise dekodeeri ülesanne ongi tagada, et ennustamise vead oleks leitud sama info alusel, mis on kättesaadav vastuvõtjale ja ei kasutataks võrdlust originaalpildiga.

Kõigist neist süsteemi osadest tuleb lähem ülevaade järgmistes alamjaotistes.

2.2 Värviteisendus

2.2.1 RGB ja YCbCr

Nagu eespool välja toodud, on inimese silma lahusvõime heleduse suhtes suurem kui värvi suhtes. Seega saab videopilti tihendada, vähendades värviinfo hulka kaadris. Selleks tuleb aga esmalt RGB värvisüsteemis olev pilt teisendada teise värvisüsteemi, kus on eraldi heleduse ja värvuse info. MPEG-2 kasutab selleks YCbCr värviruumi, kuid heleduse arvutamiseks on võimalik kasutada mitme erineva standardi poolt kehtestatud valemeid. Selles töös kasutame võimalikest variantidest FCC (*Federal Communications Commission*) ettekirjutustele vastavaid valemeid, et säilitada kooskõla allikaga [2].

Vastavad YCbCr arvutamise valemeid on [6]:

$$\begin{aligned} E_Y &= 0,3 \cdot E_R + 0,59 \cdot E_G + 0,11 \cdot E_B \\ E_{PB} &= -0,169 \cdot E_R - 0,331 \cdot E_G + 0,5 \cdot E_B \\ E_{PR} &= 0,5 \cdot E_R - 0,421 \cdot E_G - 0,079 \cdot E_B \end{aligned} \quad (2.2)$$

Valemis (2.2) on E_R , E_G ja E_B analoogsed RGB väärtused vahemikus [0; 1].

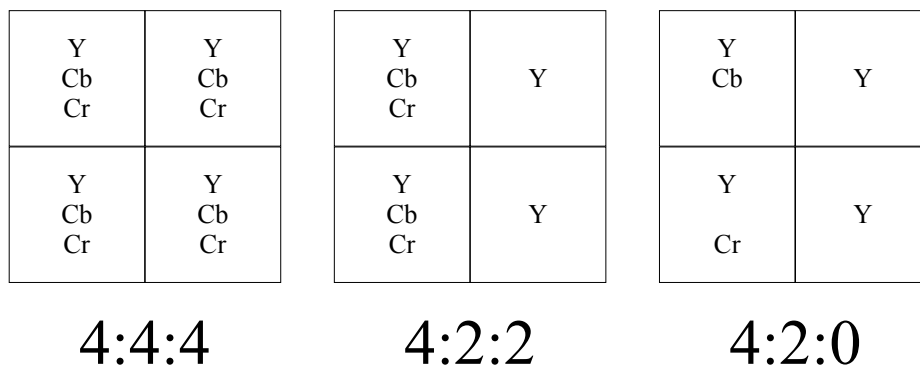
2.2.2 Värvivähendus, 4:2:0 ja 4:2:2 meetodid

Kui heledus ja värvus on eraldi viidud, siis on võimalik rakendada värviinfo vähendamise meetodeid, inglisekeelse nimetusega *color sub-sampling*. Selleks defineerib MPEG pildi töötlemise ühikud:

- *blokk* – 8x8 pikselit hõlmav ala; see on ühtlasi ka kogu MPEG'i väikseim pildiühik
- *makrolokk* – neljast blokest koosnev ruudukujuline ala pildist, mille kohta on esitatud (vastavalt formaadile) kõik kolm värvikomponenti Y, Cb ja Cr; seega võib makrolokki vaadelda kui kolmekihilist struktuuri, igas kihis 4 blokki.

Värvivähenduse meetoditena on MPEG-2 kasutusel põhiliselt 4:2:0, kuid ka näiteks 4:2:2 meetodi kasutamine on lubatud. Numbrid meetodi nimetuses tulenesid algselt YCbCr signaalide analoog-digitaal konvertimisel kasutatava taktsageduse kordajatest: 4:2:2 puhul heledusele 4x3,75 MHz, Cb ja Cr 2x3,75 MHz. MPEG puhul aga võib neid arve vaadelda ka ühes makrolokis sisalduva Y, Cb, Cr blokkide arvuna. Joonisel 4 on toodud näide ühes makrolokis sisalduvate värviblokkide arvu ja paigutuse kohta.

Nagu näha, on tihendamata pilt formaadis 4:4:4 ehk ühes makroblokis iga bloki kohta kõik kolm komponenti, kokku 12 blokki. 4:2:2 formaadi puhul on igas reas 2 heleduse blokki ja kummagi värvusetüübi kohta üks blokk, kokku 8 blokki.



Joonis 4: Värviblokkide paigutus makroblokis erinevate formaatide korral

Siinkohal on oluline tähelepanu juhtida asjaolule, et 4:2:0 ei tähenda, et selles formaadis Cb värvuse infot ei edastatagi. See on hoopis kokkuleppeline nimetus formaadile, kus üle rea vahelduvad 4:2:0 ja 4:0:2 formaadid ehk ühes reas on ainult Cb ja teises ainult Cr värvuse blokk, makrobloki peale kokku 6 blokki

Pildi edastamisel saadetaksegi joonisel 4 toodud blokid, kuid töötlemisel interpoleeritakse värvuseblokke 4:2:2 puhul ühes reas üle kahe blokki, 4:2:0 puhul üle terve makrobloki. Seega on värvivähendamine kadudega tihendamine.

Värviinfo vähendamise teel saavutatud kompressiooni võtab kokku tabel 1.

Formaat	Bitte makroblokis	Edastuskiirus	Tihendus
4:4:4	$(4 + 4 + 4) \times 8 \times 8 \times (8 \text{ bitti})$ = 6144 bitti	237,3 Mbit/s	puudub
4:2:2	$(4 + 2 + 2) \times 8 \times 8 \times (8 \text{ bitti})$ = 4096 bitti	158,2 Mbit/s	3:2
4:2:0	$(4 + 1 + 1) \times 8 \times 8 \times (8 \text{ bitti})$ = 3072 bitti	118,6 Mbit/s	2:1

Tabel 1: Värvivähenduse teel tihendamine

2.3 Diskreetne koosinusteisendus (DCT)

Enne koosinusteisenduse uurimist vaatleme ka kahte DCT-le sarnast teisendusmeetodit, mida samuti tihti kasutatakse pilditöötluses.

Kõikide järgnevalt vaadeldavate teisenduste eesmärgiks on geomeetrisest infost saada mingis muus formaadis info, milles on lihtne eraldada olulist vähemolulisest (loomulikult jällegi tekitades signaali kadusid, kuna alguses signaalis on täiesti ebaoluline ainult müra, mille eemaldamine peaks olema juba varem tehtud parimal võimalikul moel, seega siinkohal signalist mingi osa väljajätmisega vähendada informatsiooni hulka). Järgnevalt vaatleme kolme sellist teisendust ning analüüsime nende omadusi.

2.3.1 Diskreetne Fourier' teisendus (DFT)

Fourier' teisendus on juba ammu tuntud ja ka palju kasutatud leidnud, kuna tema põhiliseks funktsiooniks on teisendada signaal aeg-esitusest sagedusesitusse. Sagedusesitus on signaali analüüsimine (näiteks filtrite kavandamine) palju lihtsam ja arusaadavam. Samuti on mitmed tehted (nagu sidumi võtmine) sagedusesitusel lihtsamad teostada kui aeg-esituses.

Fourier' teisendus üldvalemiga kehtib pideva aja signaalidele. Diskreetsetele signaalide jaoks on valemiga (2.3) leitav diskreetne Fourier' teisendus [11].

$$X(\omega) = \sum_{n=-\infty}^{\infty} x[n] \cdot e^{-j\omega n} \quad (2.3)$$

Kuna pildid on lõplike mõõtmetega ja defineeritud vahemikus $[0;N]$, siis võib valemi (2.3) kirjutada kujul [12]:

$$X(\omega) = \sum_{n=0}^{N-1} x[n] \cdot e^{-j\omega n} \quad (2.4)$$

DFT arvutamine otsesel kujul on aeganõudev protsess keerukusega $O(N^2)$, kuid on välja töötatud ka nn kiire algoritm (FFT – *Fast Fourier Transform*), mille keerukus on $O(N \log_2 N)$. Siiski on DFT pilditöötluse seisukohast halb, kuna kiire algoritm opereerib *ainult* kompleksarvudega, mis tähendab, et reaalarvulise pildiinfo puhul pool arvutustulemustest on ebavajalikud, kuid nendest lahti saamiseks tuleb omakorda lisatööd teha. Samuti ei ole DFT-l energiakoondamise omadust nagu järgnevatel algoritmidel. Veel üheks DFT puuduseks pilditöötluse seisukohast on Gibbsi fenomen, mistõttu taastatud pildil esineb järskude üleminekute ümbruses ebatäpsusi, võrreldes algse pildiga [13].

2.3.2 Diskreetne lainekesteteisendus (DWT)

Lainekeste teisendus on sarnane DFT-le, kuid kasutab baasifunktsioonina kompleksse eksponendi asemel lokaliseeritud lainekesti. Lainekeste teisendus defineerib vaid põhimõtte, konkreetsed lainekested võib valida vastavalt vajadusele. Esimene laineke oli Haari laineke (1909. aastal Alfred Haari poolt välja pakutud), mis on ühtlasi ka kõige lihtsam lainekestekuju.

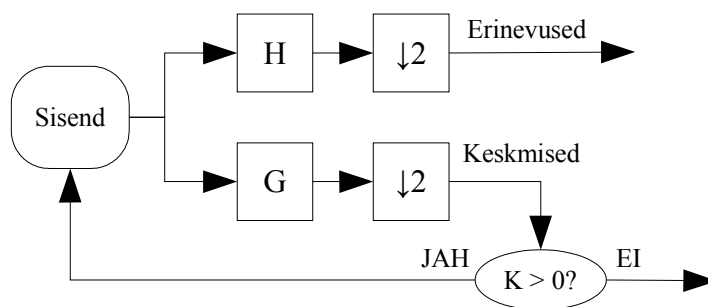
Lainekesteks arendamisel esitatakse funktsioon baaslainekestete translatsioonide ja mastaabimuutuste lineaarkombinatsioonina. Üldine valem selleks on [12]:

$$W(a, b) = \frac{1}{\sqrt{2}} \sum_{t=b}^{t=aL+b-1} x(t) h\left(\frac{t-b}{a}\right) \quad (2.5)$$

Kus L on lainekestekuju $h(t)$ ulatus (määramispiirkonna pikkus).

Nagu DFT puhul, nii ka DWT puhul on välja töötatud kiire algoritm, mis asendab valemis (2.5) kirjeldatud summa leidmise püramiidalgoritmiga mille igal sammul kasutatakse kahte filtrit ja vähendatakse lugemite arvu poole võrra. Joonisel 5 kujutatud algoritmi kohaselt jagatakse sissetulev info kõrg- (H) ja madalpääsfiltri (G) abil kaheks ja mõlemast infohulgast kustutatakse iga teine lugem. Kui tulemuseks saadud vektori mõõt on ikka veel suurem kui üks element, siis saadetakse keskmistatud lugemid (madalpääsfiltri väljund) uuesti sama süsteemi sisendisse. Lõpptulemuseks on algandmetega samapikk vektor, kus esimene element on kõikide väärtuste keskmine (alaliskomponent) ja järgnevad lugemid on täiendav info.

Kahemõõtmelise info (näiteks pildi) töötlemisel kasutatakse põhimõtteliselt sama algoritmi, töödeldes igal sammul esmalt read ja seejärel veerud ja sisendisse saadetakse 1/4 algsest maatriksist [14].



Joonis 5: Kiire DWT blokk skeem (joonis allika [12] ainetel)

DWT on väga hea meetod piltide kompressimiseks (ja ka üldisemaks töötlemiseks), kuid MPEG-

2. puhul ei ole ta kasutusel ilmselt kahel põhjusel, millest esimene on see, et kiire algoritmi puhul peavad pildi mõõtmed olema esitatavad kujul $2^m \times 2^n$, mis enamuste juhtudel nii ei ole. Seega tuleb pilt kas sobivaks kärpida või suurendada mõõtmeid järgmise kaheastmeni (lisades pilti musti piksleid).

Teine põhjus on ilmselt see, et kuigi lainekesteks-teisendus on matemaatikas tuntud juba 20. sajandi algusest, ei leidnud see laialdast rakendust enne 80.-90. aastaid. Selleks ajaks oli aga juba JPEG standard loodud, mis oli ka aluseks MPEG kompressimisel. Nüüdseks on küll välja töötatud uus standard JPEG-2000, mis kasutab pakkimiseks DWT meetodit, kuid kuna selle integreerimine olemasolevate standarditega on liiga keeruline, siis DWT-d kasutava videostandardi nimeks sai MPEG-4¹ ja see on täiesti eraldiseisev ning ebaühilduv eelmiste MPEG standarditega (sellele on lisatud ka terve rida uusi aspekte, et katta ka näiteks arvutigenereeritud objektide efektiivne edastamine).

Samuti on tekkinud ka täiesti uusi lainekeste meetodit kasutavaid videopakimise standardeid, mis näiteks kasutavad kolmemõõtmelist DWT-d (töödeldes korraga mitut kaadrit) [15].

2.3.3 Ühemõõtmeline diskreetne koosinusteisendus (DCT)

Diskreetne koosinusteisendus, mis on defineeritud valemiga (2.10), on väga sarnane DFT-le, erinedes ainult kasutatava baasfunktsiooni poolest: koosinus DFT kompleksse eksponendi asemel .

$$X(u) = C(u) \cdot \sum_{i=0}^{N-1} x(i) \cdot \cos \frac{(2i+1) \cdot u \cdot \pi}{2 \cdot N} \quad (2.6)$$

$$C(u) = \begin{cases} \frac{1}{\sqrt{2}} & kui \quad u=0 \\ 1 & kui \quad u>0 \end{cases}$$

Tõesti, kirjutades diskreetse Fourier' teisenduse (2.4) Euleri valemi (2.7)

$$e^{\pm j\theta} = \cos \theta \pm j \cdot \sin \theta \quad (2.7)$$

abil ringi, saame:

$$X(\omega) = \sum_{n=0}^{N-1} x[n] \cdot [\cos(\omega \cdot n) - j \cdot \sin(\omega \cdot n)] \quad (2.8)$$

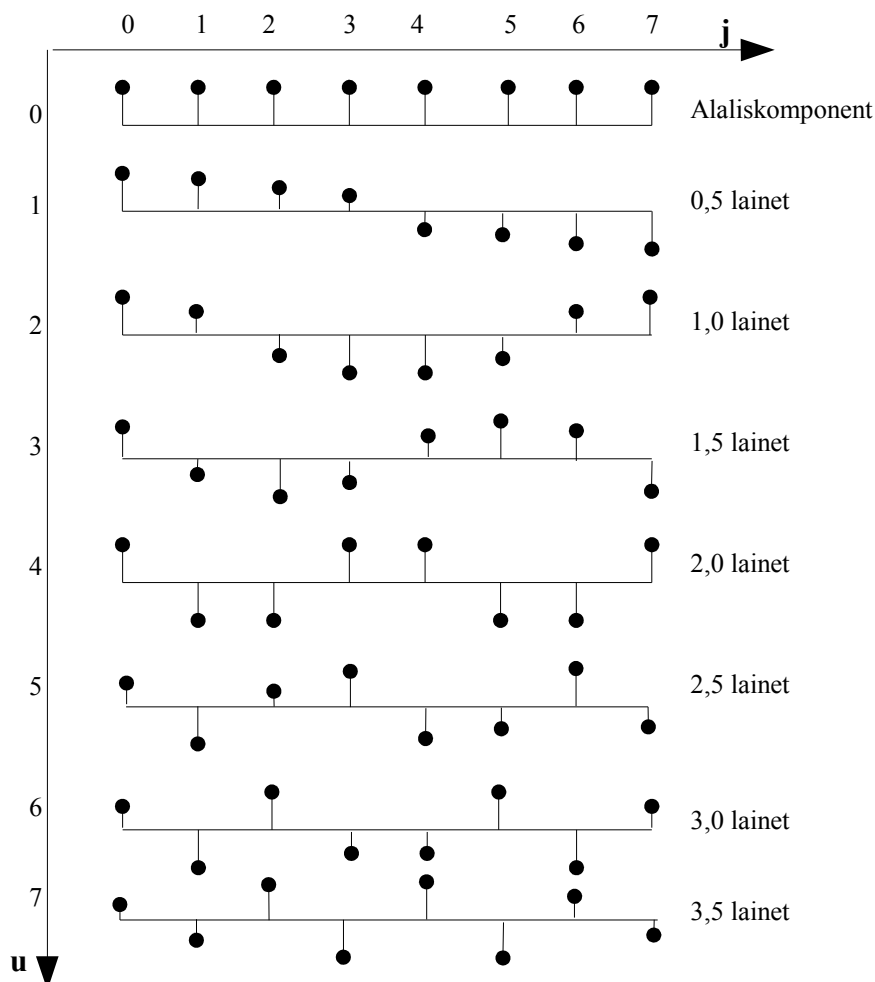
ja teades, et reaalse paarisfunktsiooni DFT on samuti reaalne ja paaris [11], on näha, et

¹ Algselt oli ka MPEG-2 standard mõeldud suhteliselt kitsa ettekirjutusena (nagu seda oli MPEG-1) ja MPEG-3 pidi saama järgmiseks kõrgekvaliteediliseks televisioonistandardiks, kuid enne MPEG-2 avaldamist integreeriti sellesse ka MPEG-3 osad ning standardist sai nõ raamistik erinevate profiilide ja tasemetega (kokku 20 kombinatsiooni).

diskreetne koosinusteisendus on samaväärne $2N$ -punktilise diskreetse Fourier' teisendusega, mis on rakendatud vektorile $x' = (x, x_{rev})$, kus x_{rev} on peegeldatud versioon x -ist (seega x' on paaris) [16].

Siit järeldub, et kuna tavaline $2N$ -punktiline DFT jätkab signaali perioodiliselt ilma peegeldamata, nii et $x' = (x, x)$, võib jätkukohas tekkida signaali järsk muutus, mis tekitab Fourier' spektris kõrgete sageduste suuremaid väärtusi võrreldes DCT esitusega, kus enamus signaali energiast on koondatud madalate sageduste piirkonda [12].

Kuigi N -punktilises koosinusteisenduses võib N olla vabalt valitud arv, on enamasti kasutusel 8-punktiline koosinusteisendus, kuna see annab üsna hea entroopia vähenemise, samas kus suuremate N väärtuste puhul on entroopia küll veelgi väiksem, kuid blokkidevaheline järsk üleminek (peale kvantimist ja signaali taastamist) on juba silmatorkavalt liiga suur. Seetõttu vaatleme edaspidi 8-punktilise koosinusteisenduse omadusi.

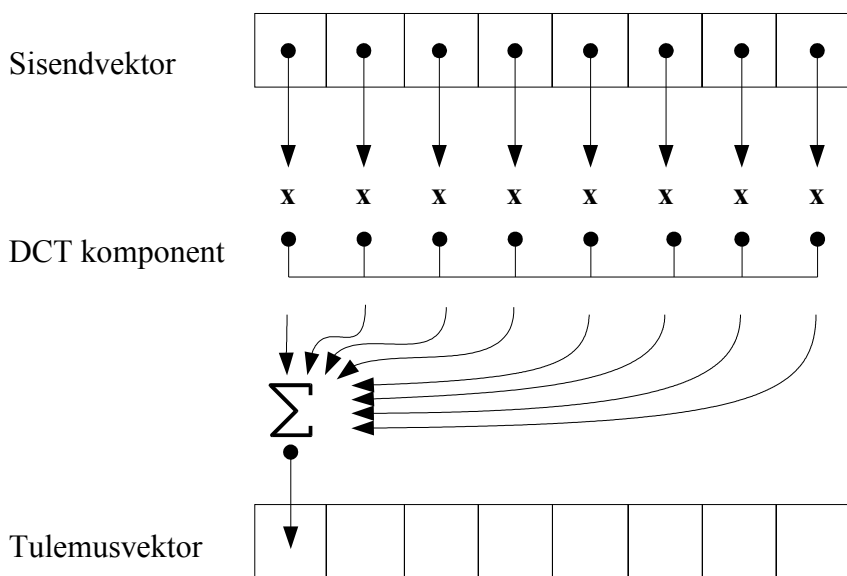


Joonis 6: Ühemõõtmelise DCT baasfunktsioonid: igas reas on parameetriga u määratud lainekuju, parameetri j abil läbitakse kõik väärtused järjestikku

Ühemõõtmelise koosinusteisenduse rakendamisel mingile sisendsignaalile jagatakse signaal 8-
 elemendilisteks vektoriteks x ja DCT arvutatakse igal vektoril teistest sõltumatult. Tulemuseks
 on samuti 8-elemendiline vektor, kus iga element on vastava DCT baasfunktsiooni osakaal selles
 infojadas. Peale kõikide vektorite teisendust on saadud sama pikk infojada DCT koefitsiente,
 kuid DCT omaduste tõttu on seda jada lihtsam tihendada. Joonisel 6 on kujutatud
 ühemõõtmelise DCT baasfunktsioonid.

Peale aliskomponendi on kõikide ülejäänud komponentide väärtuste summa üle terve perioodi
 null, seega annavad nende komponentidega vektori x läbikorrutamise nullist erineva tulemuse vaid
 juhul, kui x sisaldab sobivaid võnkumisi.

Joonisel 7 on kujutatud aliskomponendi arvutamine [17]. Nagu näha, on nn aliskomponent
 ehk koosinusteisenduse esimene element lihtsalt aritmeetiline keskmine üle terve vektori x . Kuna
 antud juhul vaatleme DCT-d just pilditöötluse seisukohast, siis on näha, et ainult positiivseid
 väärtusi omava x puhul (näiteks pildibloki üks rida) on ka aliskomponent alati positiivne.



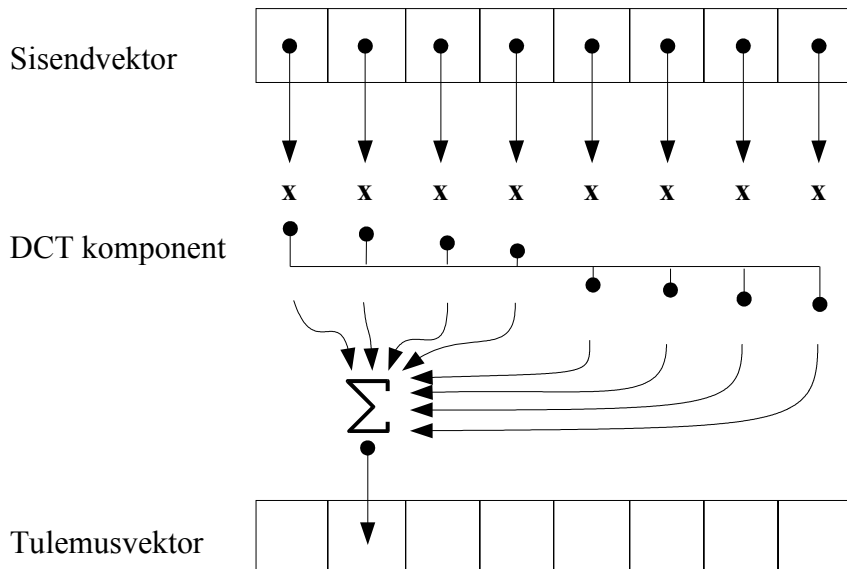
Joonis 7: DCT aliskomponendi arvutamine [17]

Iga järgneva komponendi arvutamine käib analoogiliselt joonisel 8 kujutatule. Tuleb tähele
 panna, et iga vahelduvkomponendi puhul tuleb maksimaalne summa vaid siis, kui sisendis x
 esineb täpselt sama sagedusega võnkumine. Konstantse x puhul on summa loomulikult null.

Näiteks oletame, et $x[0]$ on positiivne arv ning 5. positsioonist alates on x väärtused negatiivsed.
 Sel juhul on esimese nelja väärtuse summa positiivne ning viimase nelja positsiooni liitmisel

(kuna nii x kui DCT on negatiivsed, seega korrutis positiivne) summa kasvab veelgi.

On ilmne, et vastupidiste x -i väärtuste puhul (kui algab negatiivselt ja lõpeb positiivselt) on kogu summa negatiivne arv. Seega näeme, et DCT komponendi märk näitab, kas vastav baasfunktsioon on sisendi suhtes sama- või vastupidi polaarsusega (võrreldav kahe vastasfaasis signaaliga).



Joonis 8: DCT vahelduvkomponentide arvutamine [17]

Ilmne on ka see, et kuna kõik baasfunktsioonid on omavahel erinevad, siis sama sisendi x puhul saab ainult üks neist omada maksimaalset väärtust (teiste puhul mingid positsioonid satuvad negatiivseks ja vähendavad kogu summat). Siit on ka näha koosinusteisenduse info tihendamiseks kasutamise põhjus – garanteeritud on, et sama sisendi puhul kõik komponendid ei saa omada maksimumväärtusi.

2.3.4 Kahemõõtmeline diskreetne koosinusteisendus (2D-DCT)

Kui koosinusteisendusega töödeldakse vektorite asemel matrikseid (näiteks pilte), siis tuleb rakendada DCT-d igale reale, kuid arusaadav on, et ka veergudes olevat entroopiat saaks vähendada, kui rakendada DCT-d ka veergudele. Seetõttu võibki lihtsamal juhul rakendada DCT esmalt ridadele ja seejärel veergudele, mis annaks koguvalemiks:

$$X(u, v) = C(v) \cdot \sum_{j=0}^{N-1} \left[C(u) \cdot \sum_{i=0}^{N-1} x(i, j) \cdot \cos \frac{(2i+1) \cdot u \cdot \pi}{2 \cdot N} \right] \cdot \cos \frac{(2j+1) \cdot v \cdot \pi}{2 \cdot N} \quad (2.9)$$

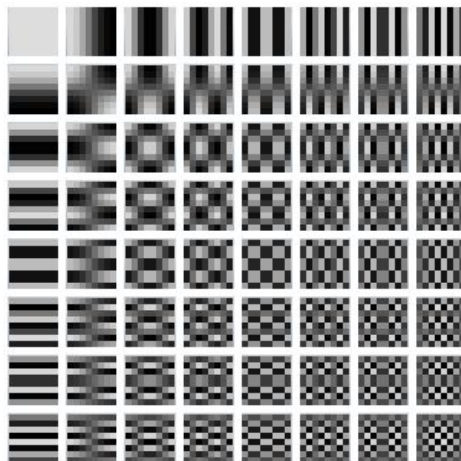
Valemi (2.9) võib aga teisendada ümber kujule:

$$X(u, v) = \frac{2}{N} \cdot C(u) \cdot C(v) \cdot \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} x(i, j) \cdot \cos \frac{(2i+1) \cdot u \cdot \pi}{2 \cdot N} \cdot \cos \frac{(2j+1) \cdot v \cdot \pi}{2 \cdot N} \quad (2.10)$$

Kordajad $C(u)$ ja $C(v)$ on leitavad samal viisil, mis valemis (2.6).

Nii nagu ühemõõtmelisel juhul, võrreldakse ka kahemõõtmelise DCT puhul sisendsignaali, milleks on enamasti 8×8 elemendiline maatriks x , DCT baasfunktsioonidega ning väljundiks on 8×8 maatriks koefitsientidega.

Joonis 9 annab hea visuaalse ülevaate kahemõõtmelise DCT baasfunktsioonidest (8×8 -punktilisel juhul). Koosinusteisenduse väljundmaatriksis on samadel kohtadel vastava baasfunktsiooni koefitsient. Väljundi esimeses reas on vertikaalseid servi ning esimeses veerus horisontaalseid servi tuvastavad komponendid. Diagonaalil asetsevad komponendid töötlevad pilti mõlemas suunas võrdselt.



Joonis 9: Kahemõõtmelise DCT igale koefitsiendile vastab üks nendest 64 baasifunktsioonist

DCT esitusest pildi taastamiseks tuleb kasutada valemit:

$$x(i, j) = \frac{2}{N} \cdot \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} C(u) \cdot C(v) \cdot X(u, v) \cdot \cos \frac{(2i+1) \cdot u \cdot \pi}{2 \cdot N} \cdot \cos \frac{(2j+1) \cdot v \cdot \pi}{2 \cdot N} \quad (2.11)$$

Terve pildi 2D-DCT-ga töötlemisel jagatakse esmalt pilt 8×8 blokkideks ning seejärel töödeldakse iga blokk eraldi. Kui pilt ei ole hallskaalas, vaid sisaldab ka värve (kas siis RGB või YCbCr formaadis), siis võib seda olukorda vaadelda kui kolme erinevat pilti ja iga värvikomponendi jaoks eraldi DCT blokid arvutada.

Kuna DCT teisenduse puhul ei ole koefitsientide väärtused enam vahemikus [0;255] vaid [-2048;2047] (tegelikult maksimum 2040, kuna sisendi väärtused ei ulatu 256-ni), siis ei saa DCT teisendust tihti mahutada samasse andmeformaati, kus oli algne pilt, kuna pildilt eeldatakse piksli maksimumväärtust 255. Seetõttu on mõistlik DCT kujutisele rakendada koheselt kvantimine ning seejärel juba edastada pilt koefitsientide kujul.

2.4 Kvantimine

Kuna DCT on kadudeta kodeerimine (kui mitte arvestada ümmardamise vigu), siis entroopia vähendamiseks tuleks DCT koefitsientidele rakendada ka kvantimist, vähendades seeläbi erinevate väärtuste arvu ja suurendades nende esinemise tõenäosust. Seda saab omakorda kasutada hiljem sari- või mõne muu tõenäosusliku koodi tekitamiseks.

Kvantimise põhimõtteks on sisendi iga elemendi läbijagamine teadaoleva konstandiga (*kvandiga*) ja jagatise ümmardamine (näiteks täisarvuni). Hiljem sama arvuga korrutades on tulemuseks signaal, mille iga väärtus erineb originaalist maksimaalselt kvantimise sammu võrra.

Teades, et DCT teisenduse maatriksis on erinevad koefitsiendid pildi kvaliteedi seisukohast erineva tähtsusega, on kvantimise eesmärgiks vähendada infomahtu just selles piirkonnas, mis on inimsilmale vähemoluline – pisidetamid. Seetõttu kasutatakse nii JPEG kui MPEG standardites konstantse kvanti asemel kvantimise tabelid. Need on $N \times N$ maatriksid, mille iga elementi $W_{i,j}$ kasutatakse DCT kujutise elemendi $X_{i,j}$ kvantimiseks. Seega tuleks kvantimise tabel koostada selliselt, et madalamad sagedused kvanditakse väiksema kvandiga (ehk neid esitatakse täpsemini) ja suuremad sagedused suurema sammuga. Kuigi MPEG-2 standard lubab ühe kaadri edastamisel kasutada korruga kuni nelja erinevat tabelit, jääb nende otstarvete selgitamine selle töö eesmärkidest välja. Tabelis 2 on toodud MPEG-2 standardi põhiline kvantimistabel [6].

8	16	19	22	26	27	29	34
16	16	22	24	27	29	34	37
19	22	26	27	29	34	34	38
22	22	26	27	29	34	37	40
22	26	27	29	32	35	40	48
26	27	29	32	35	40	48	58
26	27	29	34	38	46	56	69
27	29	35	38	46	56	69	83

Tabel 2: MPEG-2 standardis defineeritud kvantimistabel. Kasutatakse nii heleduse kui värvuse kvantimiseks

Kuigi kvantimine võib endast kujutada lihtsat jagamistehet (2.12), kasutatakse MPEG-2 standardis kvantimiseks valemit (2.13) [18].

$$Q_{i,j} = \frac{DCT_{i,j}}{W_{i,j}} \quad (2.12)$$

$$Q_{i,j} = \frac{32 \cdot DCT_{i,j}}{2 \cdot Kvaliteet \cdot W_{i,j}} + k \quad (2.13)$$

Kus

$$k = \begin{cases} 0 & I\text{-kaadrite jaoks} \\ \text{sign}(DCT[i, j]) & \text{mitte } I\text{-kaadrite jaoks} \end{cases} \quad (2.14)$$

Valemis (2.13) toodud suurus *Kvaliteet* on 5-bitine arv, mida võib kasutada jooksvaks andmemahu muutmiseks ilma, et oleks vaja uusi kvantimistabeleid kaasa pakkida (viis bitti võrrelduna 4x64x8 bitti). Kooderi väljundisse jõudev infohulk sõltub pildi keerukusest ja liikumisest kaadris, samas on vaja tagada konstantne bitivoog. Infohulga mõjutamiseks kõige lihtsam viis ongi just kvantimise sammuga: kui väljundis tekib bittide „ülejääk“, siis suurendatakse parameetri *Kvaliteet* abil kvantimise sammu, kui aga infot on liiga vähe, võib kvantimise sammu vähendada.

Sellisel kujul kvantimist ei rakendata aga alaliskomponentidele, vaid nende edastamisel muudetakse kasutatavate bittide arvu ehk edastustäpsust. DCT komponendid on algselt 11-bitised ja nende jagamisel arvuga $x \in \{1, 2, 4, 8\}$ on tulemuseks 11, 10, 9 või 8-bitised arvud.

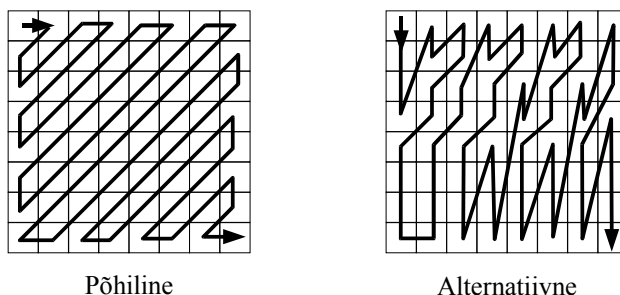
Taastamisel korrutatakse alaliskomponent uuesti sama arvuga x ehk taastatakse algne täpsus, muutes lisatud bitid nullideks.

Kvantimise pööramiseks tuleb lihtsalt rakendada valem (2.12) või (2.13) tagurpidi ja kuivõrd see on analoogiline operatsioon kvantimisele, siis lähemalt seda ei vaatle.

2.5 Sikk-sakk esitus

Kuna maatriksi kujul andmeid salvestada ega edastada (näiteks satelliidi kaudu) ei ole võimalik, siis tuleb maatriks teisendada mingiks vektoriks, et teda saaks jadaühenduses edastada. Kõige lihtsam viis selleks on edastada kõik read üksteise järel. Kuid osutub, et teistsuguste järjestuste puhul on võimalik paremini ära kasutada DCT omadust tekitada kõrgemate sageduste piirkonda palju nulliseid koefitsiente, mida on kasulik sarikoodis edastada. Sikk-sakk esituse eesmärgiks ongi DCT maatriksi elementide selline ümberpaigutamine, et kõrgsageduslike komponentide nullised lugemid satuks järjestikku.

MPEG-2 standard defineerib kaks sikk-sakk esituse viisi (joonis 10): neist esimene on tavaline sikk-sakk, teine aga kohandatud poolkaadreid kasutava video jaoks.



Joonis 10: MPEG-2 sikk-sakk esituse viisid

Alternatiivne sikk-sakk esituse meetod sobib eriti selliste piltide jaoks, kus on oluline vertikaalsuunalise info parem edastamine (näiteks poolkaadrite kasutamisest tingitud niigi halb vertikaallahutus), kuna see meetod võtab vertikaalkomponente eelisjärjekorras.

2.6 Muutuva pikkusega kodeerimine

Muutuva pikkusega kodeerimine ehk VLC (inglise keeles *Variable Length Coding*) on selline info pakkimise meetod, kus statistiliselt suurema tõenäosusega sümbolid edastatakse lühema koodiga, vähemtõenäolised pikema koodiga. Iga kood on unikaalne ja ühegi koodi algus ei sisalda teist koodi (st lühemad koodid ei sobi kokku ühegi pikema koodi algusega). Üheks

tuntuimaks VLC koodi näiteks on Morse kood, kus inglise keeles sagedasemad tähed e ja t edastatakse ühekohaliste sümbolitega, vastavalt punkt ja kriips, harvemad tähed nagu q (- - -) ja y (- - -) kolme kriipsu ja ühe punktiga.

Info- ja telekommunikatsioonitehnoloogias kasutatakse VLC koodina tihti Huffmani koodi. Huffmani koodi puhul ei ole kasutatav kooditabel eelnevalt kokku lepitud, vaid see genereeritakse kodeeritavate andmete pealt. Muidugi ei ole keelatud sarnaste andmete pideval edastamisel kooditabel statistiliselt paika panna ja salvestada saatjasse ning vastuvõtjasse, et igal edastusel seda mitte uuesti arvutada. See valik tuleb teha lähtudes konkreetsetest vajadustest, kaaludes arvutusmahu (või -aja) ja koodi efektiivsuse suhet, kuna iga edastamisega uue tabeli genereerimisel väheneb info entroopia, kuid arvutamisele kulub teatav aeg.

Üks kooditabeli genereerimise võimalikke algoritme on järgnev:

1. arvutada kõikide erinevate sümbolite esinemise sagedused edastatavas andmekogumis
2. grupeerida kaks kõige harvemini esinevat sümbolit üheks sümboliks, liites nende tõenäosused
3. korrata sammu 2, kuni alles jääb vaid kaks sümbolit: viimati genereeritud grupp ja kõige tihemini esinev sümbol
4. sammude 2 ja 3 käigus valminud kahendpuu igale lehele seada vastavusse 1 ja igale hargnemiskohale 0.
5. läbida kahendpuu¹ ja koostada tabel leitud lehtede võtmetest (nendes salvestatud sümbolitest) ja leheni jõudmiseks läbitud tippude koodidest (k.a. lehe enda kood).

Tulemuseks saadud tabelis on esimeses tulbas kodeeritav sümbol ja teises tulbas temale vastav binaarkood. Üldjuhul lõpeb kood alati 1-ga, erandiks algse tõenäosustabelis eelviimasel kohal paiknev sümbol, mille kood koosneb ainult nullidest.

Punktis 5 saadud tabeli abil kodeerida andmed. Vastuvõtja poolel on mõistlik kasutada tabeli asemel punktis 4 tekitatud kahendpuud (mida saab lihtsasti antud tabelist genereerida), kuna kahendpuust otsimine on kiirem kui terve tabeli läbi vaatamine.

Muidugi ei ole esitatud algoritm ainukene VLC kooditabeli tekitamise viis. Kui samas koodiruumis on soov edastada ka muud infot lisaks VLC koodile (näiteks kodeerimata sümboleid), siis võib VLC kooditabeli tekitada mitteautomaatselt. Sellisel juhul võib näiteks kümne sagedasema sümboli järel VLC kood pikeneda mitme biti võrra, mille arvelt saab edastada mittekodeeritud sümboleid. Selle tulemusena siis dekodeerimisel otsitakse esmalt

¹ läbimise järjestus pole koodi seisukohast oluline, kuid võib mõjutada kodeerimise algoritmi kiirust, sõltuvalt kasutatavatest andmestruktuuridest

kooditabel läbi, kui vastet ei leita, käsitletakse seda infohulka kui kodeerimata juhtu. Modifitseeritud VLC koodi näiteks võib tuua MPEG-2 kasutusel olevad tabelid, kus on võimalike erinevate sümbolite (milleks on nii positiivsed kui negatiivsed arvud) hulka vähendatud poole võrra, võttes esmalt arvust absoluutväärtuse ja lisades seejärel VLC koodi lõppu ühe biti, mis näitab arvu tegelikku märki.

Näide MPEG-2 bloki kodeerimisest:

Peale sikk-sakk esitust võib üks DCT blokk välja näha selline (lõpeb nullidega):

1275 1 -19 -5 0 0 0 2 3 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 ...

Alaliskomponent (1275) asendatakse eraldi kooditabelist leitava väärtusega ja saadetakse eraldi.

Ülejäänud vektori võib ümber kirjutada selliselt:

0, 1 | 0, -19 | 0, -5 | 3, 2 | 0, 3 | 6, 1 | 9, 1 | *bloki lõpp*

Kus iga nullist erineva lugemi ette kirjutatakse temale eelnevate nullide arv. Vastavad terminid oleks *jooks* nullide arvu kohta ja *tase* nullidele järgneva koefitsiendi kohta (inglise keeles *run* ja *level*). MPEG-2 väljatöötamisel uuriti väga suurt hulka sellist infot ja statistilise esinemise tõenäosuse alusel koostati Huffmani koodi tabel levinuimatest kombinatsioonidest. Antud tabelid on toodud allika [6] lisa B. Nendes tabelites on tõenäolisematele kombinatsioonidele pandud vastavusse lühem kood, harvematele pikem kood. Kuna pole mõtet kõiki võimalikke kombinatsioone tabelisse kanda, siis tabelist puudevate kombinatsioonide edastamiseks on üks spetsiaalne paakood mille järel tuleb 6-bitine *jooks* ja 12-bitine *tase*.

Meie näite saab selle tabeli alusel ringi kirjutada:

$\overbrace{110}^{0,1}$
 $\overbrace{0000}^{0,-19}$
 $\overbrace{0000}^{0,-5}$
 $\overbrace{0111}^{3,2}$
 $\overbrace{0010}^{0,3}$
 $\overbrace{0100}^{6,1}$
 $\overbrace{0001}^{9,1}$
 $\overbrace{010000}^{\text{bloki lõpp}}$
 $\overbrace{1010}^{\text{bloki lõpp}}$
 $\overbrace{10}^{\text{bloki lõpp}}$

Sellele lisandub veel alaliskomponendi kodeerimiseks kuluv bittide hulk, mis sõltub sellest, millisel hetkel see blokk välja saadetakse, olles maksimaalselt (selle bloki korral) 13 bitti.

Seega on peale kodeerimist näiteblokk esitatav vaid 72 bitiga, võrrelduna 512 bitiga, kui DCT teisendust ja kodeerimist poleks tehtud.

Nagu öeldud, märgib koodi lühidus selle kombinatsiooni esinemissagedust ja on loogiline, et *bloki lõpp* kood ainult 2-bitine, kuna igas blokis tuleb see kood kindlasti edastada.

Siinkohal tasub mainida, et toodud näide ei olnud spetsiaalselt genereeritud, vaid võetud ühe tavalise foto DCT teisendusest ja kuna kõik kombinatsioonid olid tabelis esindatud, mitmedki päris lühikeste koodidega, siis näitab see MPEG-2 standardis defineeritud Huffmani koodi kvaliteeti.

2.7 Liikumise ennustamine ning MPEG-2 kaadrite tüübid

Nagu eespool mainitud, kasutab MPEG info pakkimiseks ära ka videopiltide ajalist liiasust ehk järjestikuste kaadrite sarnasust. Kõige lihtsam moodus selle teostamiseks on ennustada järgmist kaadrit eelmise info põhjal. MPEG aga kasutab veelgi keerukamaid meetodeid, kasutades ära teadmist, et videofilmi kaadrite omavaheline erinevus ei ole juhuslik vaid enamasti seotud mingi liikumisega selles kaadris. Seetõttu on lootust leida sama objekti järgmisel kaadril pisut teises kohas. Liikumise ennustamise ühikuks on makroblokk ehk 16x16 piksliline osa pildist vastavas värviformaadis (enamasti 4:2:0), kuid poolkaadri režiimis võib kasutada ühes poolkaadris ka 16x8 ala, kuna poolkaadris on ridu kaks korda vähem kui progressiivses kaadris.

Ennustamise põhimõtteks on võrrelda kahte kaadrit ja otsida sealt eelmises kaadris leidunud makroblokkidele sarnaseid makroblokke ja esitada nende asukoha muutus vektorina. Otsimist teostatakse *otsimise aknas*, milles kontrollitakse iga võimalik makrobloki paigutus läbi ja rahuldutakse kõige väiksema veaga paigutusega. Kuna MPEG-2 standard defineerib ainult dekodeerimise protsessi, mitte kodeerimise, siis on liikumise ennustamise algoritmi väljatöötamine koodeeri tootja ülesanne ja koht, millest sõltub koodeeri töökiirus ja kvaliteet (tihendamise võime). Liikumise ennustamise üheks olulisemaks parameetriks ongi just otsimise akna suurus, kuna suurema akna puhul on tõenäolisem leida parem vaste, kuid väiksema akna puhul on töötlemise kiirus suurem (kuna on vähem kombinatsioone läbi kontrollida). Kui makrobloki uus asukoht on leitud, siis tekitatakse sellest liikumise vektor (*motion vector*), mille alusel dekodeerija saab kaadri taastada.

Kuna selge on, et ennustamine pole alati (tihti) täpne, siis on koodeeris integreeritud ka dekooder, mis kasutab ennustatud liikumisvektoreid üritab ja taastada pilti. Seda tulemust võrreldakse originaaliga ja leitud erinevused töödeldakse DCT abil nagu eelpool kirjeldatud ning saadetakse koos liikumisvektoritega. Liikumisvektorid ise pakitakse kohe VLC koodi.

MPEG standardis on defineeritud kolme liiki kaadrid:

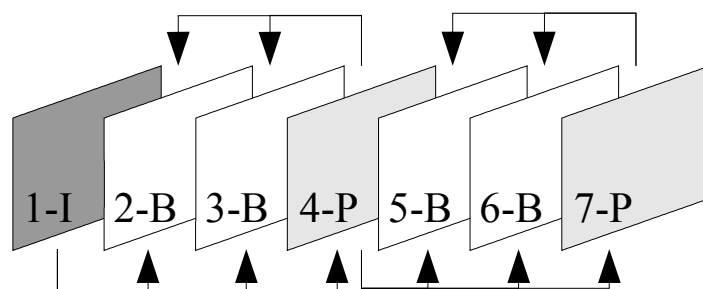
- *I-kaader* – kaader, mille vastuvõtjas kuvamiseks ei ole teisi kaadreid vaja. Nimetus tuleneb inglisekeelsest sõnast *Intra-Coded* ehk sisemiselt (iseendaga) kodeeritud. Seda tüüpi kaadri puhul ei toimu ennustamist, vaid kogu kaader töödeldakse DCT-ga, liikumisvektoreid juurde ei lisata.
- *P-kaader* – ennustatud kaader, mille esitamiseks on vaja talle eelnevat I- või P-kaadrit.

Nimetus sõnast *Predicted*.

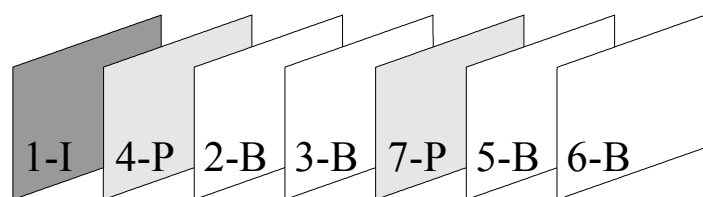
- *B-kaader* – *Bidirectionally Predicted* – kahesuunaliselt ennustatud kaader, mille esitamiseks on vaja talle eelnevat ja järgnevat I- või P-kaadrit.

Tüüpiliselt vajab I-kaader kolm korda rohkem bitte kui P-kaader, mis on omakorda 50% suurem B-kaadrist [5].

B-kaader on eriti kasulik juhul kui liikumine tõi nähtavale väga detailiderohke ala, mille P-kaadrites esitamine tähendaks suurt erinevust kahe kaadri vahel. B-kaadrite puhul aga võetakse keskmine edasisuunas ennustatud liikumisest ja tagasisuunas ennustatud liikumisest, mis tähendab, et kuna järgnevas P-kaadris on see sama ala samuti nähtaval, siis konkreetne B-kaader võib selle info esitada pisut vähema detailsusega (keskmistatult selle ala puudumisega). Kui aga järgnevas P-kaadris seda ala enam näha polnud (näiteks liikus kõnealune makroblokk oma algsele asukohale tagasi), siis pole tarvidust ka B-kaadris seda ala välja tuua, kuna inimsilm nahunii ei märka nii kiiret muutust. Samas vajab B-kaadri töötlemine rohkem arvutusressurssi ja aega, mis mõnel juhul pole võimalik.



Esitusjärjestus



Edastusjärjestus

Joonis 11: MPEG pildigrupp: nooled tähistavad ennustamise suunda, numbrid esitamise järjekorda, tähed kaadri tüüpi

Kuna B-kaadrite esitamiseks on vaja teada nii eelmist kui järgmist kaadrit, siis on defineeritud selline struktuur nagu pildigrupp (*Group of Pictures – GOP*). Joonisel 11 kujutatud pildigrupi tüüpi tähistatakse IBBPBBP, kuid edastamisel järjestatakse kaadrid ümber selliselt, et kõik ennustamise aluseks võetud kaadrid edastatakse enne nende pealt ennustatud kaadreid. See

järjestus puudutab just B-kaadreid, kuna nende taastamiseks on vaja ka tulevikukaadreid.

2.8 MPEG-2 profiilid ja tasemed

Et toetada erinevate vajadustega valdkondi, defineerib MPEG-2 standard erinevaid profiile, mis määravad ära kasutatavad alamhulga MPEG-2 süntaksist. Iga profiili jaoks on erinevaid tasemeid, mis määravad selles profiilis kasutatavate võimaluste täpsed parameetrid, nagu pildi mõõtmed, bitikiirus jne. Profiilid ja tasemed on organiseeritud selliselt, et iga kõrgem profiil või tase on eelmiste ülemhulk, seega dekodeer peab suutma töödelda ka madalama taseme või profiili videot. Näiteks lihtsas profiilis (*Simple Profile*) ei ole lubatud B-kaadrite kasutamine, kuna see nõuab kaadrite ümberjärjestamist, mis mõnel juhul pole võimalik.

Enamus MPEG-2 dekodeeritest toetavad põhiprofiili põhitaset (MP@ML).

3 VALMINUD TARKVARA KIRJELDUS

Kirjeldatud loengu esitamisega seotud probleemide lahendamiseks valmis õppetarkvara nimega DVBSim, mille abil on lektoril (või ka tudengil) võimalik visuaalselt demonstreerida DVB kujutise töötles kasutatavate meetodite tööpõhimõtteid. Järgnevalt vaatleme lähemalt programmi võimalusi ja realisatsiooni.

Tarkvara on realiseeritud programmeerimiskeeles Java ning testitud JRE (*Java Runtime Environment*) versioonil 1.5.0 Windows keskkonnas. Kuna Java tehnoloogia on kiiresti arenev, siis on programmi töötamist vanemate JRE versioonidega testitud vaid pisteliselt, sest juba ainuüksi turvalisuse tõttu on soovitatav alati paigaldada viimane versioon Javast. Kuna erinevate organisatsioonide poolt välja antud Java virtuaalmasinad (JVM) katavad erinevad alamhulgad kogu Java keele spetsifikatsioonist, siis ei ole välistatud, et antud programm kasutab mõnda ainult firma Sun JRE-s esinevat omadust ja on seetõttu mitteühilduv teiste JVM-idega. Kuna aga Sun'i virtuaalmasin on saadaval kõikide levinuimate operatsioonisüsteemide jaoks (sh ka Linux, kus valminud programm ka edukalt töötab), siis ei testita selle tarkvara toimimist teiste virtuaalmasinate peal.

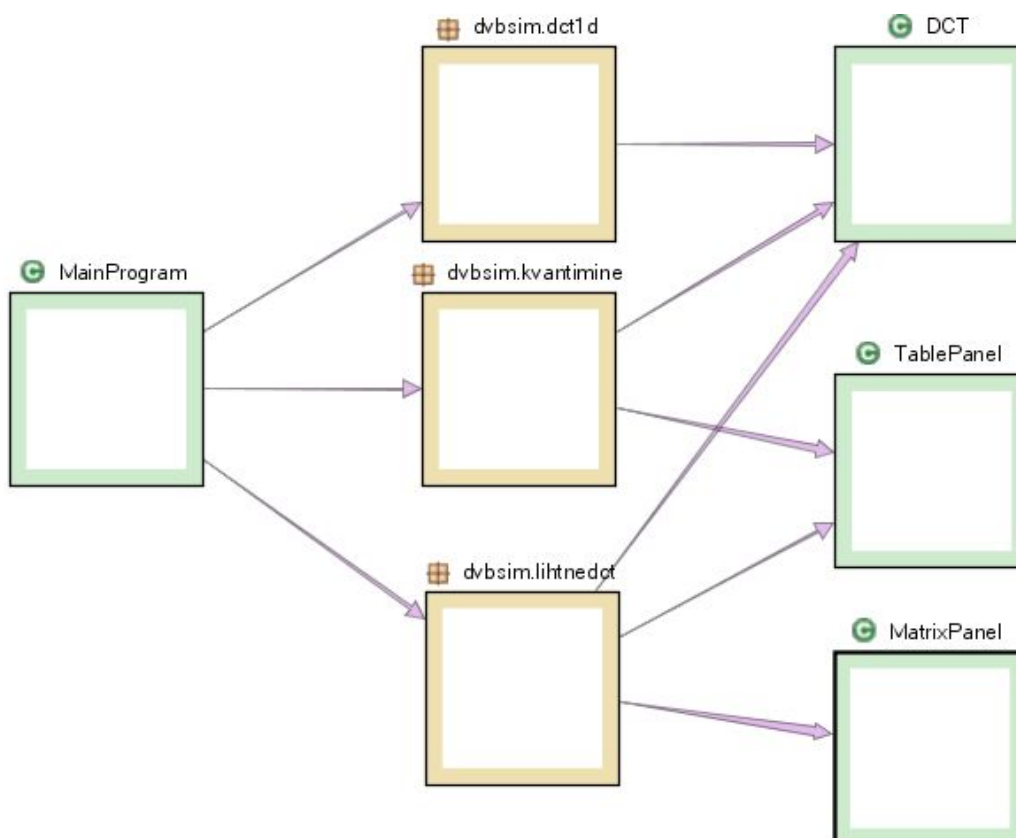
Nagu valdkonda tutvustavas peatükis on välja toodud, koosneb DVB kujutise töötles põhiliselt MPEG-2 kodeerimisest. Seetõttu keskendub ka valminud programm just MPEG-2 tihendamismeetodite tutvustamisele.

Siiski ei ole programmi eesmärgiks tutvustada tervet MPEG-2 standardit, vaid ainult seda osa, mis puudutab digitaaltelevisiooni (lähtuvalt loengus käsitletavast materjali ulatusest). Seetõttu on mõnede programmi osade koostamisel realiseeritud nii konkreetse probleemi põhimõtteline lahendus kui ka MPEG-2 standardis kehtestatud variant. Kuna digitaaltelevisioonist arusaamisel valmistab tudengitele enim probleeme just diskreetne koosinusteisendus ja kvantimine, siis ka loengus ja seega ka valminud programmis, pühendatakse nendele osadele enamus ajast.

Kuna valminud tarkvara on mõeldud kasutamiseks Tartu Ülikooli telekommunikatsiooni aluste loengus, kus töökeeleks on eesti keel, siis on ka programmi kasutajaliides eesti keelne.

3.1 Programmi DVBSim graafiline liides ja võimalused

Loengus keeruliste probleemide esitamisel, nagu seda on koosinusteisendus ja kvantimine, alustatakse lihtsamatest näidetest ja liigutakse sujuvalt edasi keerulisematele. Seetõttu on ka DVBSim programm üles ehitatud modulaarsena, kus iga moodul käsitleb erinevat aspekti kogu teemast, nii et järgnevad moodulid eeldavad arusaamist eelnevatest ja kasutavad nendes tutvustatud meetodeid sisemiselt oma töös.



Joonis 12: Programmi pakistruktuur. Joonisel pakikese tähisega moodulite paketid ja C-kujulise märgikesega abistavad klassid (sh põhiprogrammi käivitav MainProgram). Noole suund tähistab välja kutsumise suunda.

Joonisel 12 on väljakutsumise järjekorras programmis kasutatavad funktsionaalsed osad. Esmalt käivitatakse põhiprogramm (MainProgram), mis laeb sisse kolm moodulit (joonisel pruuni raamiga), mis omakorda kasutavad kolme põhipaketis sisalduvat klassi korduvate meetodite (nagu DCT teisendus) hoidmiseks. Hiljem on võimalik sellesse struktuuri mooduleid juurde lisada, muutes ainult põhiprogrammi selliselt, et ka uus moodul graafilisse liidesesse integreeritaks.

Järgnevates peatükkides analüüsime iga moodulit eraldi ning kirjeldame selle kasutusvõimalusi.

3.2 Moodul 1: Ühemõõtmeline koosinusteisendus

Enne pilditöötluse juurde asumist tuleb tudengitele selgeks teha koosinusteisenduse töötamise põhimõtte ja omadused. Selleks on sobiv alustada ühemõõtmelise juhuga, kuna selle graafiliselt kujutamine ning uurimine on lihtsam teostada ja tudengitele ka arusaadavam. Seetõttu ongi programmi esimeseks mooduliks ühemõõtmelise DCT uurimine. Joonisel 13 toodud ekraanipildil on näha selle mooduli graafilise liidese ülesehitus. Nagu näha, on ekraanil neli graafikut: sisendsignaali, selle DCT kujutis, uuritav DCT komponent nii eraldi kui koos sisendiga ühel graafikul. Lisaks on välja toodud ka iga DCT baasi puhul kasutatav valem ja sisendsignaali vastavate pikselite heledused (varieerudes mustast valgeni nii negatiivsete kui ka ainult positiivsete sisendi väärtuste puhul).

Kuna graafikud on kohandatud näitamaks maksimumväärtust 255, siis tuleb vahemikus $[-1;1]$ muutuva koosinusteisenduse baasifunktsiooni visualiseerimiseks ka see teisendada sobivasse vahemikku. Selleks on DCT baasi iga väärtust korrutatud graafiku maksimumväärtusega 255 ehk võetud selle komponendi ja „valge signaali“ korrutis, mis teeb selle komponendi ühtlasi ka sisendsignaali samas vahemikus olevaks. DCT kujutise arvutamisel kasutatakse muidugi baasi algseid väärtusi.

Koosinusteisenduse uurimiseks on vajalik sisendsignaali muuta, milleks on erinevaid võimalusi:

- käsitsi, hiirega graafikul tulpi liigutades
- inverteerida olemasolevat signaali
- lasta arvutil signaal genereerida
- võtta sisendiks DCT baasiga sarnane signaal.

Neist kõige põhilisemaks töövahendiks osutub ilmselt hiirega signaali muutmine, kuid ka peegeldamine on oluline, kuna näitab mismoodi muutub DCT teisendus kui sisendsignaali on vastupidiste väärtustega.

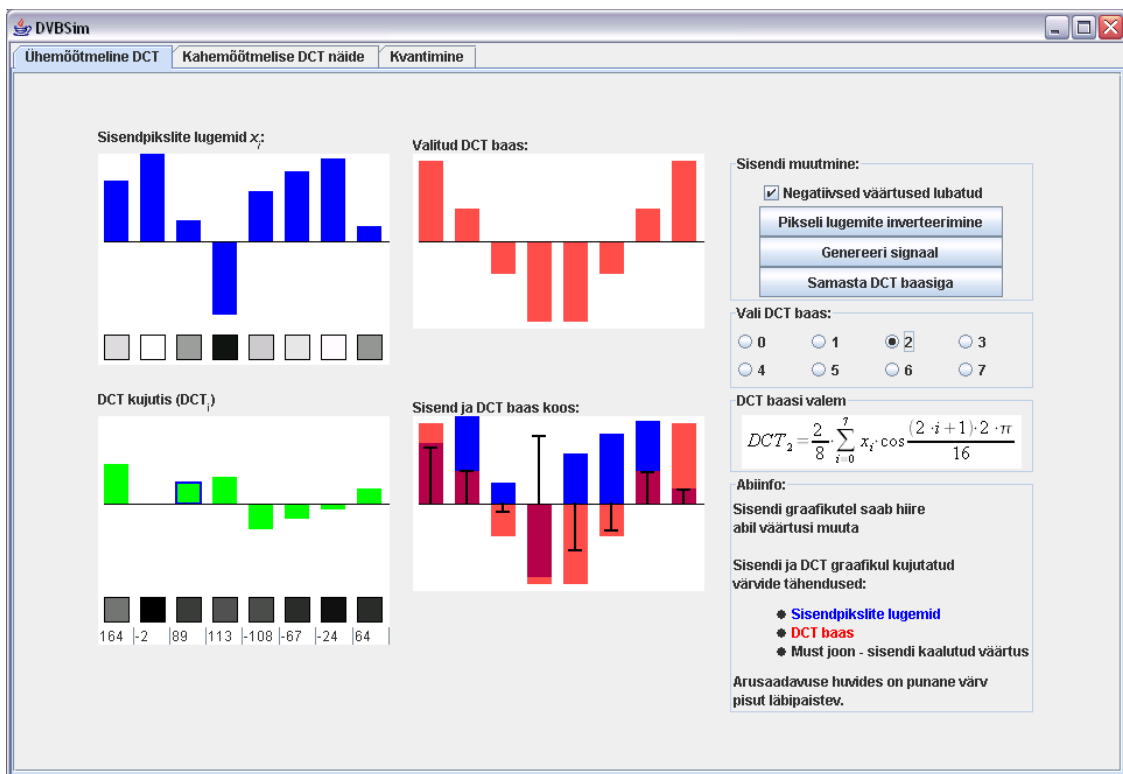
Kõikide kirjeldatud sisendi muutmise protsessides arvestatakse ka seda, kas sisendis on negatiivsed väärtused lubatud. See valik on oluline, kuna pilditöötluses on signaal alati positiivne - seega ka alaliskomponendiga. Ainult positiivse sisendi puhul inverteeritud väärtuste leidmine on tavalisest inverteerimisest erinev seetõttu, et vastandaruuliste väärtuse leidmise asemel arvutatakse tulemus valemiga (3.15):

$$x_i = \text{MAX_VAL} - x_i \quad (3.15)$$

kus MAX_VAL on sisendis lubatud maksimumväärtus 255.

Arvuti poolt tekitatud signaal saadakse lihtsalt juhuarvude generaatori abil. Keerulisemate signaalide (näiteks sinusoidi, kolmnurk- või kastsignaali) generaatorite realiseerimine ei ole mõttekas, kuna lektorile on lihtsam soovitud signaali kuju hiire abil paika panna.

DCT komponendiga samastatud signaali tekitamisel võetakse sisendiks samad väärtused, mida kasutatakse selle DCT komponendi graafikul kujutamiseks. Kui sisendis on negatiivsed väärtused keelatud, siis tuleb see signaal teisendada ka positiivsesse vahemikku.



Joonis 13: Ekraanipilt: ühemõõtmelise DCT moodul.

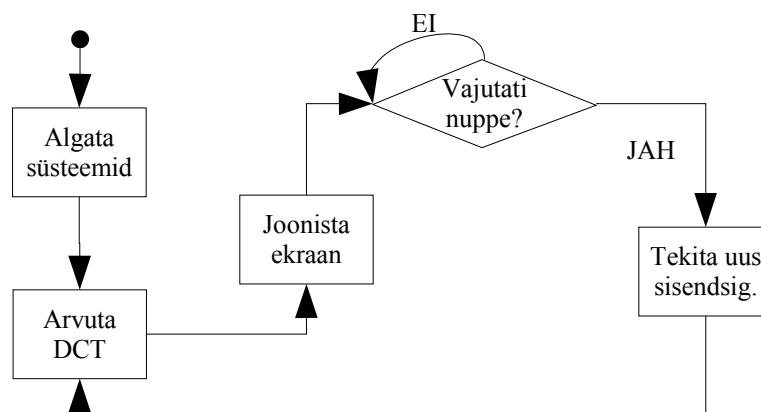
Joonisel 13 on kujutatud olukord, kus sisendsignaal on genereeritud arvuti poolt ja uurimiseks on valitud DCT 3. komponent.¹ Kogu koosinusteisendust kujutaval graafikul on samuti värvilise raamiga ümbritsetud uuritavale komponendile vastav tulp. Sisendsignaali muutmisel arvutatakse jooksvalt välja nii kogu koosinusteisenduse kujutis kui ka konkreetse komponendiga

¹ Kuna DCT komponente loendatakse alates nullist, siis on selguse huvides ka programmis nummerdatud komponente alates nullist, et säiliks side valemi ja programmi graafiku vahel.

läbikorrutamisel saadavate üksikute tegurite väärtused, mis on joonisel kujutatud mustade joontena. Arusaadavuse huvides on erinevat tüüpi signaalid (nagu sisendsignaali, baas ja DCT) kujutatud omavahel erinevate värvidega. Sisendi, koosinuse komponendi kui ka vastava korrutise ühel graafikul kujutamine teeb nendevahelise sõltuvuse hästi jälgitavaks – kui koosinuse komponendi väärtus on väike, siis ei saa loomulikult ka selle korrutis sisendiga olla suur, samuti on näha korrutise märgi sõltuvus teguritest.

Siinkohal tuleb märkida, et kõnealusel sisendit ja DCT komponenti koos kujutaval graafikul on arusaadavuse huvides koosinuskomponendi värv pisut läbipaistev, kuna ta joonistatakse sisendsignaali peale ja on vaja, et tema tagant ka sisend igal hetkel nähtav oleks. Seega tuleb joonisel näha olevat kolmandat värvi tulpa mõista kahe erineva signaali kattumisena.

Mooduli funktsionaalne diagramm on toodud joonisel 14. Esmalt genereeritakse algne signaal, arvutatakse sellele vastav DCT kujutis ning joonistatakse kogu graafiline kasutajaliides. Seejärel jääb programm ootama kasutaja poolset muutuste tegemist.



Joonis 14: Ühemõõtmelise DCT mooduli blokk skeem

Enamus selle mooduli programmikoodist tegeleb graafilise liidese tekitamise ja muutmisega, kuna DCT arvutamise blokk on üpris lühike. Lähtetekst 1 toob kaks meetodit, mida kasutatakse ühemõõtmelise koosinusteisenduse arvutamiseks. Meetod *getDCT* võtab oma parameetrikts ühemõõtmelise massiivi sisendinfoga ja tagastab samapika massiivi DCT teisendusega, mille iga komponendi arvutamiseks kutsus välja meetodi *dctLugem*. Viimases meetodis kasutatakse ühe komponendi leidmisel kiiruse huvides varem ette arvutatud tabelit *cos_table* kõikide vajalike koosinuste väärtustega valemist (2.10).

```

// meetod võtab sisendiks vektori ja väljastab
// samuti vektorina selle DCT teisenduse
public int[] getDCT(int[] in){
    int[] out = new int[in.length];
    for (int i = 0; i < in.length; i++){
        out[i] = dctLugem(i,in);
    }
    return out;
}

// meetod ühe DCT lugemi arvutamiseks
private final int dctLugem(int u,int[] data){
    // väljundi hoidja
    dct_out = 0;
    int i,j;
    for (i = 0; i < N; i++){
        dct_out += data[i] * cos_table[u][i];
    }
    return (int)((dct_out * C[u] * 2)/N);
}

```

Lähtetekst 1: Ühemõõtmelise DCT arvutamise meetodid

Erinevalt teistest moodulitest, on selles moodulis kasutusel lisaks harilikele nuppudele ja valikukastidele ka otsene hiire liikumisest ja nupuvajutustest sõltuv Java2D animeerimine sisendsignaali tulpdiagrammi genereerimiseks [20]. Selle jaoks on koostatud hiire liikumise ja nupuvajutuste jälgijad, mis on lisatud sisendigraafikule. Et pakkuda kasutajal võimalust ka DCT komponendi ja sisendi ühisgraafikul sisendi suurust muuta, on lisatud hiire jälgija sama isend [19]¹ ka ühisgraafikule, nii et ühisgraafikul hiire liigutamisel on tulemus sama, mis sisendi graafikul tegevusi sooritades.

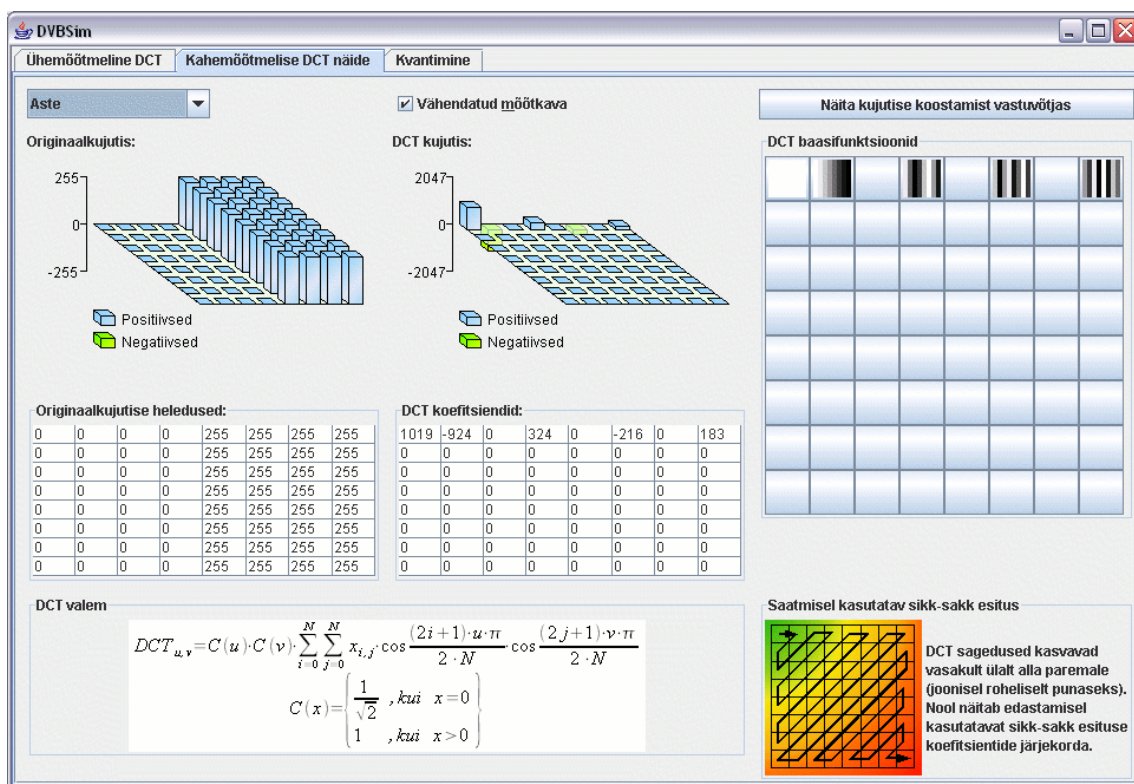
3.3 Kahemõõtmelise DCT näide

Ühemõõtmelisest koosinusteisendusest edasi kahemõõtmelise juurde siirdudes ei ole enam kuigi lihtne teha visuaalselt jälgitavat ent samas piisava lihtsuse ja kiirusega kasutatavat näidisprogrammi, kus kasutaja saab sisendit ise muuta ja sellele vastavat DCT teisendust jälgida.

¹ Java eesti keelne terminoloogia on võetud J. Kiho Väikesest Java leksikonist [19]

Piisavalt väikeste maatriksi mõõtmete puhul on DCT elemente liiga vähe ja ei avaldu nende tegelik kuju, samas suuremate mõõtmete puhul on sisendmaatriksi muutmine tülikas. Näiteks ei ole eriti reaalne ekraanile 64 liugurit paigutada, mille abil igat komponenti muuta saaks. Samuti on tabelis nende ühekaupa väärtuste andmine tülikas.

Seetõttu piirdubki 2D-DCT demonstratsiooni moodul vaid ette defineeritud sisendsignaalide ja nende koosinusteisenduste näitamisega (joonis 15). Toodud sisendsignaalid on aga iseloomulike omaduste välja toomiseks spetsiaalselt koostatud. Näiteks joonisel 15 kujutatud signaali puhul tuleb tähele panna, et tegemist on vertikaalsuunas (vt joonisel toodud tabel) muutumatu signaaliga, mistõttu ka vertikaalsuunalised DCT koefitsiendid on nullid. Samas on horisontaalsuunas järsk tõus, mis põhjustab ka kõrgeima koefitsiendi tugeva väärtusega ilmumise DCT kujutisse. Lisaks tuleb tähele panna, et sisendsignaal on oma keskpunkti suhtes paaritu, mistõttu põhjustab mitte-täisarv koosinusvõnkumisi sisaldavate komponentide ($u = 1,3,5,7$) suuri väärtusi, samas nullistades keskpunkti suhtes sümmeetrilised komponendid.



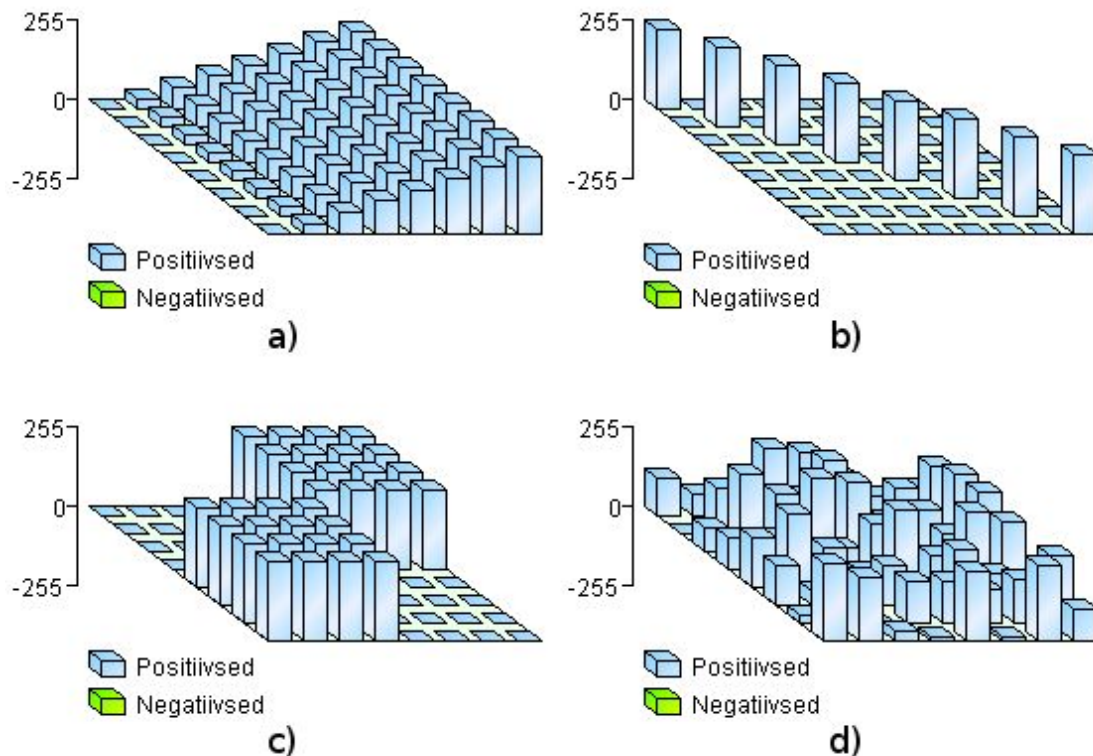
Joonis 15: Ekraanipilt: kahemõõtmelise DCT näide

Ekraan on jaotatud loogiliselt kolme ossa: sisendsignaali ja DCT kujutise osad ning DCT teisendust iseloomustavate abinuppude osa. Nii sisendsignaal kui ka DCT kujutis on esitatud nii visuaalselt 3-mõõtmelise maatriksi näol kui ka arvudena tabelis. Rippmenüü abil on võimalik valida erinevate algkujutiste vahel:

- Ühtlane signaal
- sujuv tõus
- kahe-suunaline sujuv tõus
- aste
- veerandid
- diagonaal
- rist
- suvalised väärtused

Joonisel 16 on toodud nendest mõnede näited.

Sisendiks suvaliste väärtuste korduv valimine genereerib alati uue signaali, seega on võimalik demonstreerida juhuslikult valitud signaalide omadusi (kõrgemate sageduste suuremad väärtused jne). Tavaliste piltide töötlemisel nii kõrgeid sagedusi enamasti ette ei tule, kuna 8x8 pikslit on liiga pisike ala nii kiireteks muutusteks.

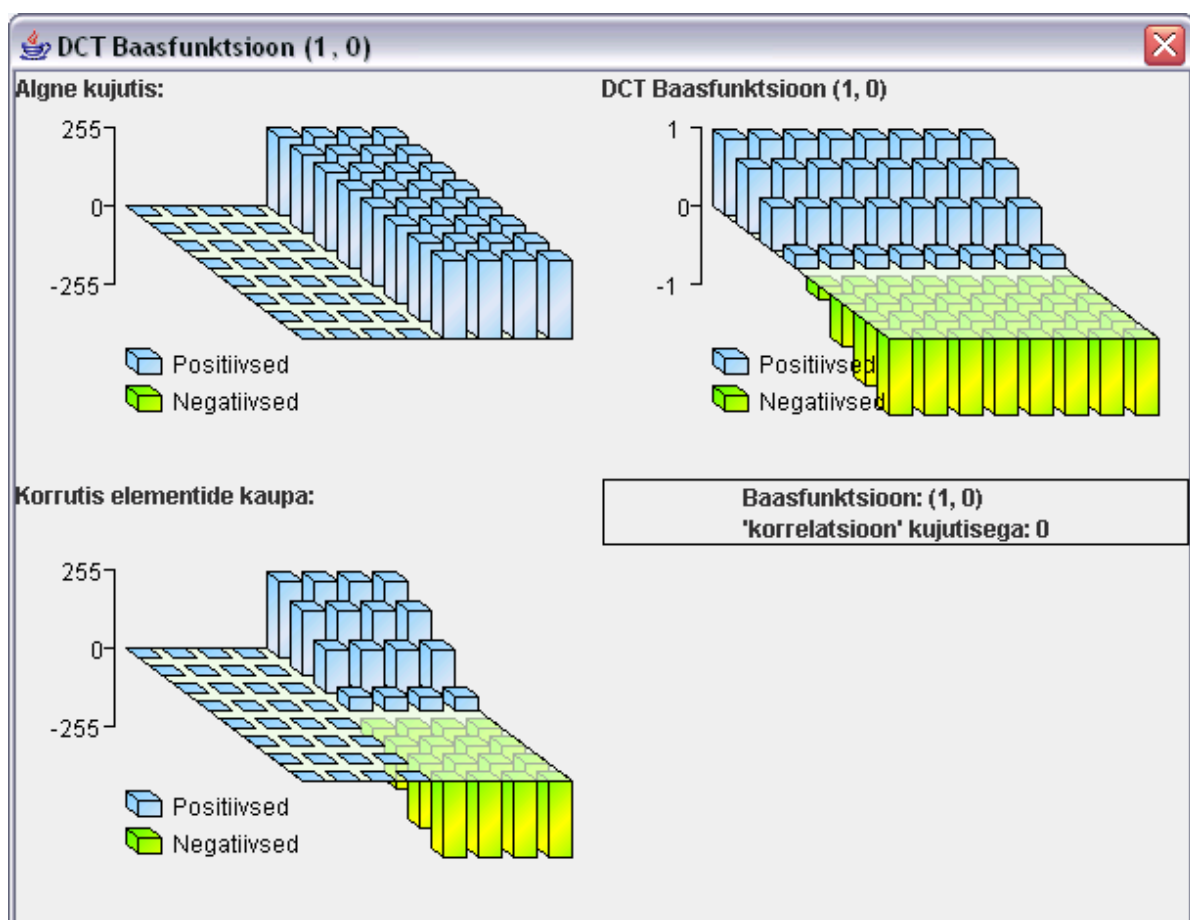


Joonis 16: Kahemõõtmelise DCT näites kasutatavaid sisendkujutisi: a) sujuv tõus, b) diagonaal, c) veerandid ja d) suvalised väärtused

Kuna DCT signaali maksimumväärtused võivad ulatuda kuni 2040 siis algse pildiga kõrvuti näitamiseks tuleb DCT kujutist vähendada. Programmile on lisatud aga ka valik DCT kujutise

näitamiseks originaaliga samas mõõtkavas, mistõttu võib osa graafikust ekraanist välja ulatuda. Informatiivsuse huvides on lisaks kolmemõõtmeliste graafikutele samad maatriksid esitatud ka tabelina, et oleks konkreetsed arvvaartused näha.

DCT kujutise lähemaks uurimiseks on ekraanile lisatud ka kõigile 64 baasifunktsioonile vastavad nupud, millel on kujutatud vastav baasifunktsioon, kui see DCT koefitsient oli nullist erinev. On ilmne, et valge pildi puhul on vaid üks komponent nullist erinev, kuid kiiremini muutuvate piltide (näiteks juhusliku signaali) puhul on tihti kõik 64 baasi nullist erineva väärtusega. Nimetatud nuppudel vajutades avaneb aken (joonis 17), kus on näidatud nii algne kujutis, valitud DCT baas kui ka nende korrutis elementide kaupa.

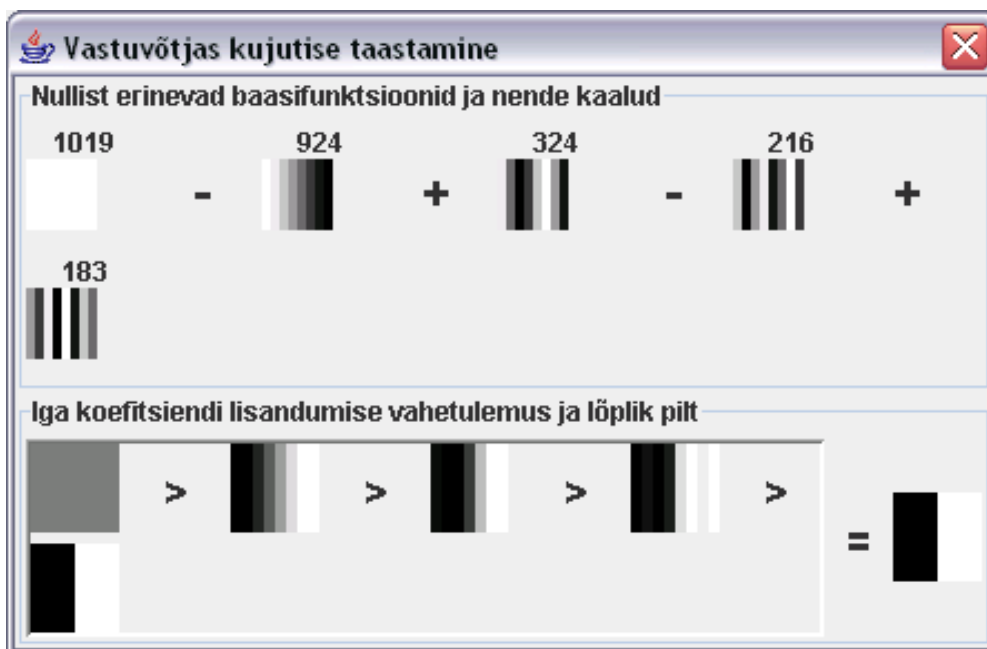


Joonis 17: Ekraanipilt: 2D-DCT baasifunktsiooni uurimine. Toodud on nii algne kujutis, valitud baasifunktsioon kui ka nende maatriksite korrutis elementide kaupa.

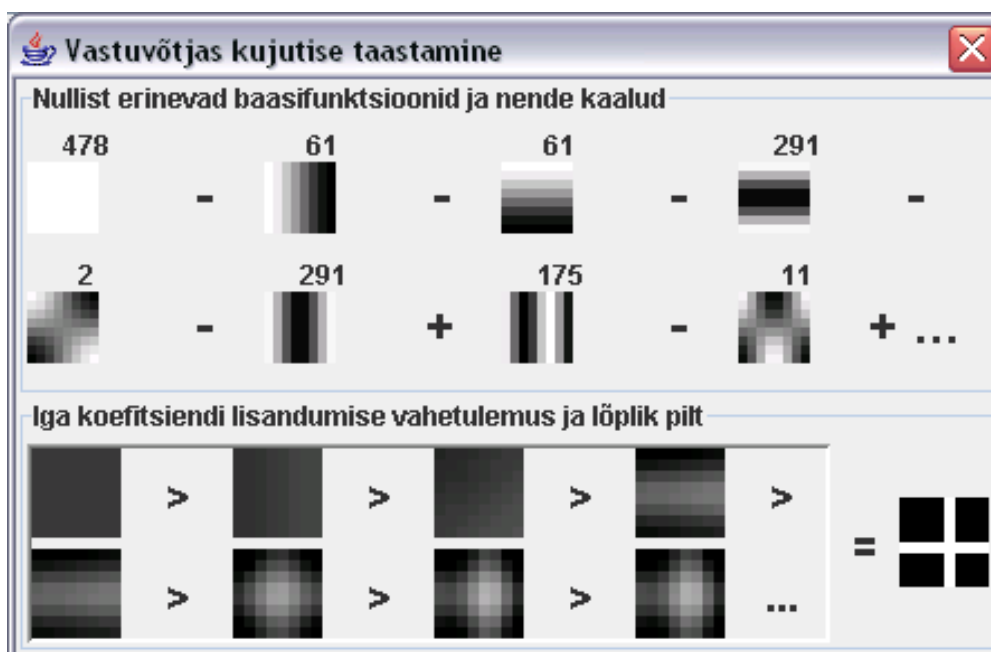
Nagu näha, on valitud baasi kuju selline, et summeerides korrutised kokku, on tulemuseks null.

Selles moodulis demonstreeritakse ka põhimõtet, mille alusel hiljem vastuvõtjas kujutis taastatakse (joonis 18). Selleks näitab programm sikk-sakk esituses kaheksa esimese nullist erinevale lugemile vastavaid baasifunktsioone, koos lugemitega ja tehtmärgiga – kas liidetakse

tulemusele või lahutatakse sellest. Samuti on näidatud ka erinevad kujutised, mis saadaks, kui vastuvõtja edastaks vaid mõned esimestest mittenullistest lugemitest. Joonisel 19 toodud juhul on näha, et ükski kaheksast vahepealsest juhust ei sarnane eriti tegelikule pildile (erinevalt joonisel 18 näidatud juhust).



Joonis 18: Ekraanipilt: kujutise taastamine vastuvõtjas sisendsignaali "aste" puhul. Taastamiseks piisab 5 nullist erineva lugemi edastamisest.



Joonis 19: Ekraanipilt: kujutise taastamine vastuvõtjas sisendsignaali "rist" puhul. Kvaliteetseks taastamiseks on vaja edastada praktiliselt kõik 64 koefitsienti

Selles moodulis kasutatava kahemõõtmelise koosinusteisenduse arvutamiseks rakendatakse algoritmi 1, mis vastab eelpool kirjeldatud valemile (2.10) ehk iga koefitsient arvutatakse korruga välja. Alternatiivsete lahendustena võib arvutada esmalt kõikide ridade ja seejärel veergude koosinusteisendused.

$$DCT(sisend) = \begin{cases} tulemus \leftarrow \text{maatriks}(8,8) \\ \forall u \in \{0, 1, \dots, 7\} \\ \quad \forall v \in \{0, 1, \dots, 7\} \\ \quad \quad tulemus_{u,v} \leftarrow DCTLugem(u, v, sisend) \\ \leftarrow \text{tulemus} \end{cases}$$

Algoritm 1: Kahemõõtmelise DCT arvutamine

Algoritmis 1 kasutatav funktsioon *DCTLugem()* on üksiku DCT koefitsient arvutamiseks, mis leitakse algoritmiga :

$$DCT(u, v, sisend) = \begin{cases} N \leftarrow 8 \\ \leftarrow \frac{2}{N} \cdot C_u \cdot C_v \cdot \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} sisend_{u,v} \cdot KOOSINUS_{u,i} \cdot KOOSINUS_{v,j} \end{cases}$$

Algoritm 2: Kahemõõtmelise DCT ühe koefitsiendi arvutamine.

Kiiruse huvides on valemis (2.10) kasutatavad koosinuste väärtused varem ette arvutatud ja tabelisse *KOOSINUS* salvestatud, mille iga rida vastab ühele DCT koefitsiendile ja iga veeru indeks vastab ühele sisendi väärtusele. Samal põhjusel on ette arvutatud ka igale koefitsiendi rea ja veeruindeksile vastavad kordajad C_i . Lähtetekst 2 on neile algoritmidele vastav programmikood.

Käsitletavas moodulis kujutatakse maatrikseid kolmemõõtmeliste tulpdiaagrammidena, kuid kahjuks ei õnnestunud leida ühtegi vabavaralist (või ka tasuta kasutatavat, kuid eelkompileeritud) teeki selliste graafikute joonistamiseks, kuna kõikide teekide arendajad on sellesse piisavalt palju aega investeerinud, et levitavad oma tarkvara vaid raha eest. Seetõttu koostati klass *MatrixPanel*, mis visualiseerib etteantud maatriksi tulpdiaagrammi näol. Kuigi Java keel pakub vahendeid ka kolmemõõtmelises ruumis graafika töötlemiseks, otsustas autor, et 3D graafika kasutamine oleks antud tarkvara liiga mahukaks (ja arvutusressursinõudlikuks) teinud. Seetõttu realiseeriti tulpdiaagrammid lihtsa visuaalse perspektiivi efekti tekitamisega. Kuna iga tulba tekitamine kolmest perspektiivi jaoks väljaväänatud riskülikukujulisest tahust on liiga aeglane meetod, siis koostatakse tulbad hoopis hulknurgast, mille väliskuju ühtib kolmemõõtmelise tulba näilise

väliskujuga ja mille peale joonistatakse eraldi hulknurga abil vaataja poole jäävad servad. Vastava algoritmi 3 puuduseks võrreldes 3D meetodiga on see, et diagrammi ei ole võimalik parema vaatenurga saamiseks pöörata. Siiski, arvestades antud tarkvara otstarvet, on see puudus väikese tähtsusega.

$$\left| \begin{array}{l} \forall x \in M, x < 0 \\ | \text{joonistaNegatiivneTulp}(x) \\ \text{joonistaVahealus}() \\ \forall x \in M, x \geq 0 \\ | \text{joonistaPositiivneTulp}(x) \end{array} \right.$$

*Algoritm 3: Tulpdigrammide
joonistamise algoritm*

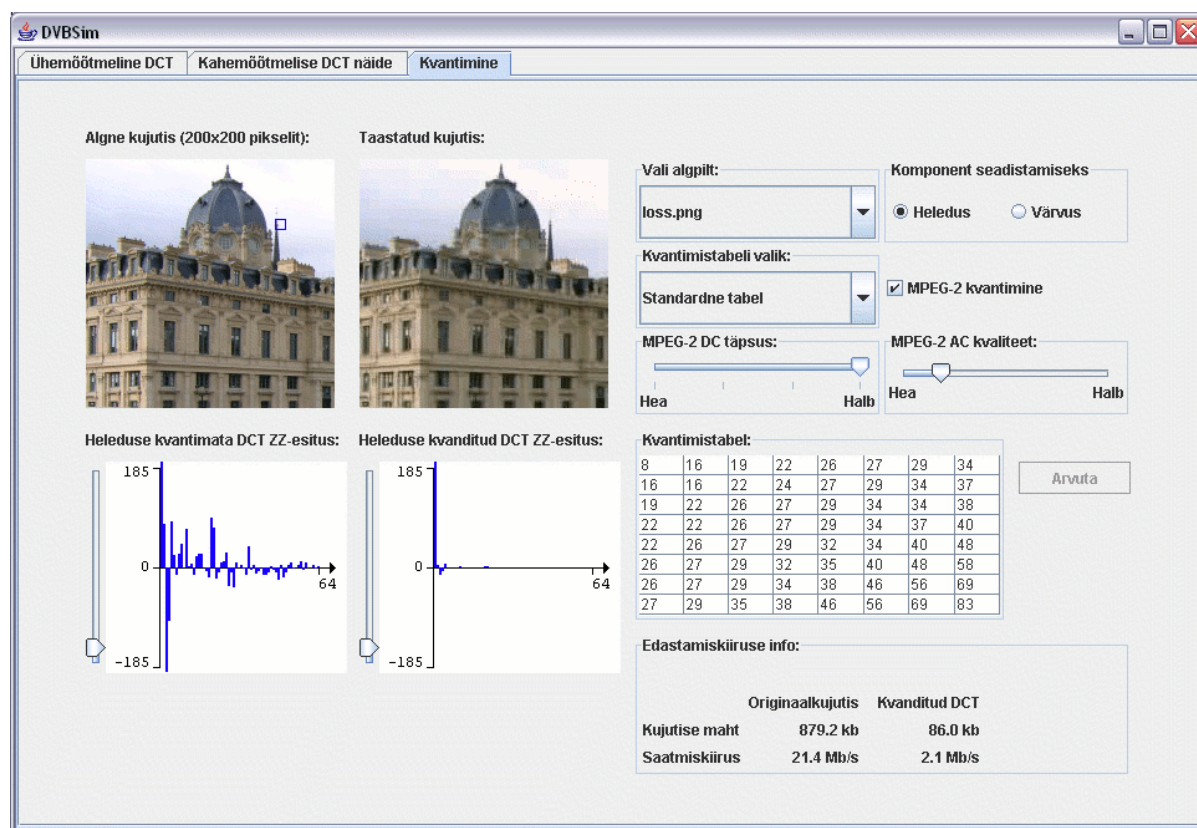
```
// meetod tagastab maatriksi DCT teisenduse
public int[][] getDCT(int[][] in){
    int[][] out = new int[in.length][in[0].length];
    for (int i = 0; i < in.length; i++){
        for (int j = 0; j < in[0].length; j++){
            out[i][j] = dctLugem(i,j,in);
        }
    }
    return out;
}

// meetod ühe DCT lugemi arvutamiseks
private final int dctLugem(int u, int v, int[][] data){
    // väljundi hoidja
    dct_out = 0;
    int i,j;
    for (i = 0; i < N; i++){
        for (j = 0; j < N; j++){
            dct_out += data[i][j] * cos_table[u][i] *
                cos_table[v][j];
        }
    }
    // eemaldatai Math.round()
    return (int)((dct_out * C[u] * C[v] * 2)/N);
}
```

Lähtetekst 2: Kahemõõtmelise koosinusteisenduse arvutamine

3.4 Kvantimine

Kõige rohkem saab programmiga tööd teha muidugi kvantimise moodulis, kuna selle juures on kõige rohkem parameetreid, mille muutmine tulemust mõjutab. Joonisel 20 on toodud ekraanipilt ühest töödeldud pildist. Ekraan jaotub jällegi osadeks: algse kujutise info, taastatud kujutise info ja kvantimise valikud. All paremas servas antakse ka hinnang konkreetsete valikute mõjust pildi suurusele ja ka ligikaudsele vajaminevale andmeedastuskiirusele, et sellist pilti 25 kaadrit sekundis edastada¹. Nagu näha, on joonisel näidatud valikute puhul tihendus umbes kümnekordne, samas algse ja taastatud pildi erinevus pole silmaga märgatav.



Joonis 20: Ekraanipilt: kvantimine

Programm võimaldab:

- valida algpilti, valikuteks on erinevad mustrid mustvalgel ja ka värvilisel kujul (eesmärk näidata inimsilma värvitundlikkuse erinevust heleduse tundlikkusest) ja ülal joonisel kujutatud pilt.
- Valida, kas tehtavad muutused kehtivad heleduse või värvuskomponendi kvantimise puhul.

¹ Andmeedastus kiiruse hinnangus on arvestatud vaid kvantimisest tulenevat erinevust algse ja kvanditud pildi vahel ja ei ole arvestatud teisi MPEG-2 pakkimismeetodeid nagu 4:2:0 värvivähendus ja liikumise ennustamine ega ka satelliidi kaudu edastamise jaoks hiljem lisatavaid lisabitte, mis infohulka uuesti kasvatavad.

- Valida kvantimiseks kasutatavat tabelit:
- Määrata kvantimise meetod:
 - lihtne jagamistehe (valemi (2.12) alusel)
 - MPEG-2 standardile vastav alaliskomponenti ja kõrgemaid sagedusi erinevalt käsitlev meetod (valemi (2.13) alusel).
- Kui kvantimise meetodiks on MPEG-2, siis saab muuta
 - alaliskomponenti edastustäpsust 8 – 11 bitti.
 - vahelduvkomponentide kvaliteeti, muutes valemis (2.13) toodud parameetrit *Kvaliteet*.

Kvantimistabelit on võimalik valida järgnevate hulgast:

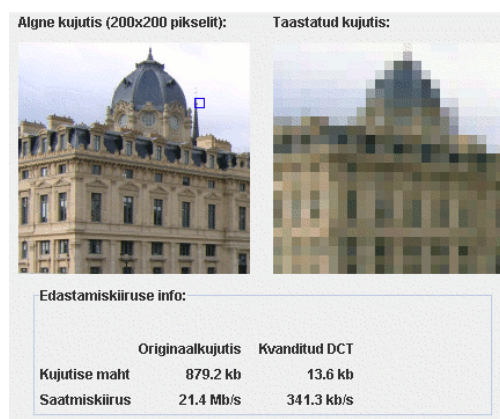
- ilma kvantimiseta – ühtedest koosnev tabel
- MPEG-2 standardis kehtestatud põhitabel
- vastupidine tabel – see tabel on standardtabeli abidiagonaali ümber transponeeritud variant, st madalaid sagedusi kvanditakse suurema sammuga ja kõrgemaid sagedusi väiksema sammuga.
- Suureastmeline tabel – terve tabeli ulatuses võrdselt suur kvantimise samm
- muudetav tabel – selle tabeli algväärtuseks on ühiktabel, kuid igat väärtust saab eraldi muuta. Peale tabeli koostamist tuleb vajutada nuppu „Arvuta“, et näha tulemust.

Kui kvantimise meetodiks on valitud MPEG-2, siis ühiktabeliga kvantimisel valemi (2.13) kujust lähtuvalt saadakse esmalt hoopis suurenenud koefitsiendid, kuna kvantimise sammu ja kvaliteedi korrutis peab olema suurem kui 16, et kvantimine toimuks „õiges suunas“.

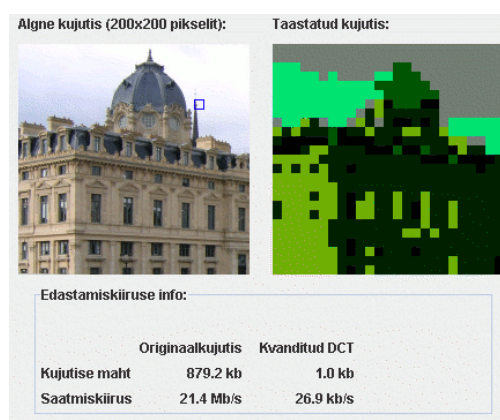
Kvantimisest parema ülevaate saamiseks on akna allservas kuvatud sikk-sakk esitus (joonisel 20 märgitud lühendiga ZZ-esitus) ühest heleduse blokist nii kvantimata kui ka kvanditud kujul. Uuritava bloki asukoha määramiseks on originaalpildi peale projekteeritud 8x8 ruuduke ja pildil hiirega klikkides saab bloki asukohta muuta, muutes sellega ka sikk-sakk esitust. Siinkohal on segaduste vältimiseks vajalik märkida, et bloki valikuruudu värv (punane või sinine) ei edasta mingit infot, vaid see muutub, et ruuduke alati vähemalt mingil määral nähtav oleks. Ruudu värv valitakse sõltuvalt uue asukoha ülemise vasaku nurga alla jääva pikseli punase-sisalduse hulgast – kui punase-sisaldus on suur, siis joonistatakse sinine ruut, muidu punane.

Sikk-sakk esituse puhul on võimalik samuti muuta graafiku mõõtkava, et ka pisemad koefitsiendid tuleksid nähtavale. Joonisel 20 seatud kvantimise parameetrite puhul on enamus koefitsiente väga väikesteks (või nulliks) kvanditud.

Joonisel 21 on toodud näide liiga tugevalt kvanditud pildi kohta (kasutati MPEG-2 meetodit). Info on küll tihendatud ligi 65 korda, kuid sellise kvaliteediga pildilt tunneb objekti ära ilmselt ainult seetõttu, et originaal on kõrval. See pilt aga iseloomustab hästi MPEG-2 kvantimise põhimõtet – kui kõik muud lugemid peale alaliskomponendi (aga see edastati ilma kvantimiseta) nulliks võtta, siis taastatud pildis on tulemuseks ühtlased 8x8 blokid, mille heledus ja värvus on selle bloki aritmeetilised keskmised. Kui kvantimise meetodiks määrata lihtsalt läbi jagamine, siis on tulemuseks pilt joonisel 22, mida ei saa kuidagi originaali sarnaseks lugeda. See tõendab, et MPEG-2 meetodil alaliskomponendi kvantimata jätmine (muutes vaid esitustäpsust) on pildi kvaliteedi seisukohast olulise tähtsusega, kuna alaliskomponent kannab endas põhilist infot. Ka joonisel 20 toodud kvantimisparameetritega puhul MPEG-2 kvantimise välja lülitamisel oleksid värvid väga valesti esitatud, kuna värvide kvantimiseks kasutati suureastmelist tabelit. See tõestab, et inimese silm on värvusele vähem tundlik, kui heledusele, kuna värvuse puhul piisas ühest keskmistatud väärtusest kogu 8x8 pikselilise bloki kohta, samas kus heleduse jaoks on vaja iga piksel eraldi esitada.

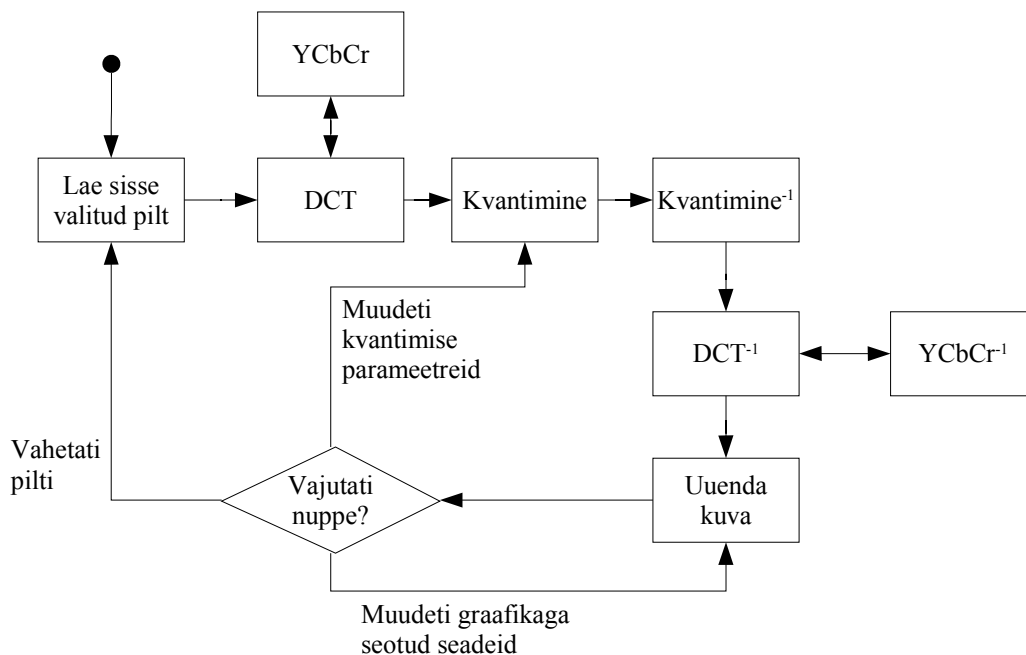


Joonis 21: Ekraanipilt: MPEG-2 meetodil maksimaalselt kvanditud pilt



Joonis 22: Ekraanipilt: joonisega 21 sama tabeliga, kuid valemi (2.12) abil kvanditud pilt.

Mooduli tööpõhimõtet iseloomustab skeem 23.



Joonis 23: Kvantimise mooduli blokk skeem

Selles moodulis demonstreeritakse lisaks kvantimisele ka inimsilma tundlikust heleduse ja värvuse suhtes. Seetõttu on vajalik RGB formaadis sisendpilt esmalt teisendada YCbCr formaati, mida teostab lähtetekst 3:

```
public int[] encodeYCbCr(int rgb){
    // FCC parameetritele vastav kodeerimine
    double r = ((double)((rgb >> 16) & 0xff) / 255.0);
    double g = ((double)((rgb >> 8) & 0xff) / 255.0);
    double b = ((double)(rgb & 0xff) / 255.0);
    int Y = (int)(219 * (0.3*r + 0.59*g + 0.11*b) + 16);
    int Cb = (int)(224 * (-0.169*r - 0.331*g + 0.5*b) + 128);
    int Cr = (int)(224 * (0.5*r - 0.421*g - 0.079*b) + 128);
    return new int[]{Y,Cb,Cr};
}
```

Lähtetekst 3: meetod teisendamaks RGB infot YCbCr formaati

Meetod saab sisendiks ühe 32bitise täisarvu, milles on kodeeritud RGB 3x8 bitti infot (ja sisaldab lisaks ruumi 8bitise läbipaistvuse info salvestamiseks), millest tuleb esmalt eraldada kõik kolm komponenti. Kuna kasutatav teisenduse valem eeldab, et RGB info on antud vahemikus [0;1], siis ühtlasi teisendatakse värvid ka ujukomaarvudeks. Vastupidist teisendust

teostav kood on analoogiline, seega siin välja ei tooda.

Skeemil 23 toodud kuva uuendamise blokis genereeritakse muuhulgas ka info pildi mahtude ja andmeedastuskiiruste kohta. Selleks loendatakse iga piksli esitamiseks kuluva kolme komponendi väärtustes kasutatavad bitid meetodiga lähtetekstist 4.

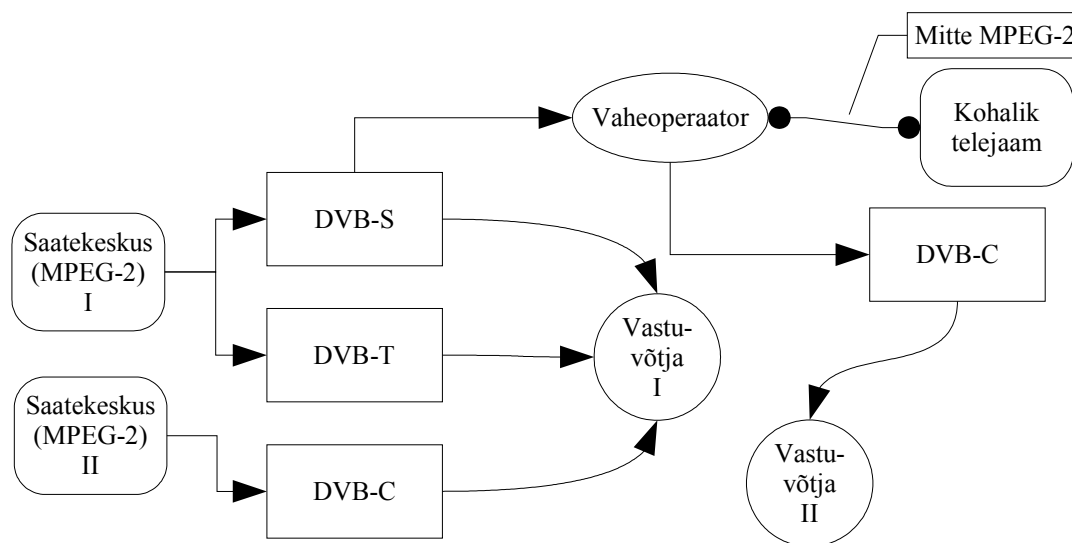
```
private int countBits(int x){
    int count = 0;
    while (x != 0){
        x = x >>> 1;
        count++;
    }
    return count;
}
```

Lähtetekst 4: Bittide loendur

4 DIGITAALSE VIDEOEDASTUSE PÕHIMÕTTED

4.1 DVB süsteemi ülesehitus

DVB (*Digital Video Broadcasting*) süsteem jaguneb mitmeks alamstandardiks: DVB-S, DVB-C, DVB-T jt [21]. Nimetatud standardid reguleerivad digitaalpildi edastamist erinevates meediumides (satelliitside, kaabellevivõrk ja maismaa-antennid). Kõikides DVB süsteemides on edastatava pildi kodeerimiseks kasutatud MPEG-2 standardit [22][6][23], mille suhtes on rakendatud teatud piiranguid, nagu näiteks ainult transportvoo edastamine [24].



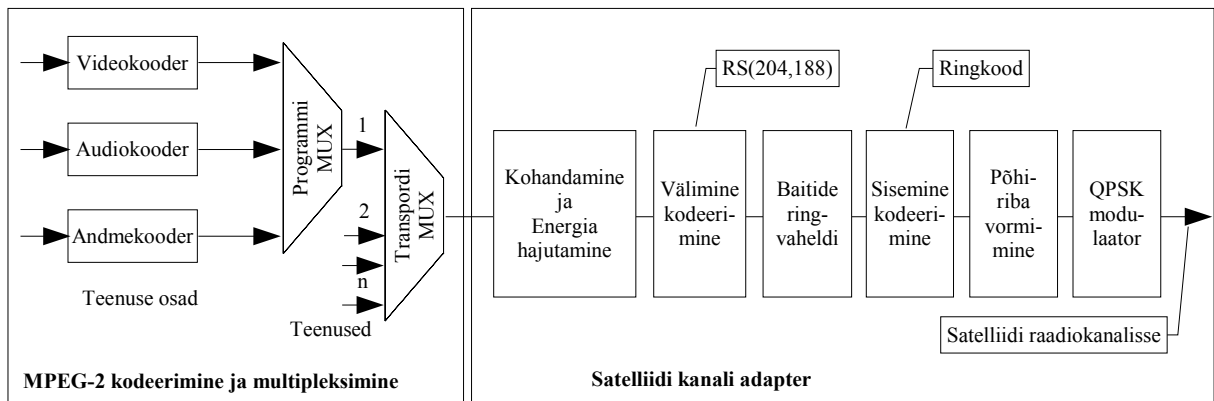
Joonis 24: DVB süsteemi võimalik ülesehitus: mitu MPEG-2 saatekeskust, DVB edastus, vaheoperaator ja vastuvõtjad. Vaheoperaator edastab ka kohalikku telejaama.

Joonisel 24 toodud skeemis on DVB struktuuri võimalik ülesehitus. Kujutatud olukorras on „vastuvõtjal I“ võimalik näha „Saatekeskus I“ programmi nii satelliidi kui ka maapealsete antennide vahendusel. Kaabli kaudu jõuab temani ka „Saatekeskus II“ programm. „Vastuvõtja II“ aga näeb ainult kaabellevi vahendusel „Saatekeskus I“ programmi, kuid tema kaablioperaator edastab ka kohaliku telejaama saateid, mida operaator teisendab MPEG-2 formaati mingist muust edastusvormist.

Ülalnimetatud standarditest vanim ja hetkel levinuim on DVB-S ehk satelliittelevisiooni standard (kuigi viimasel ajal on ka teiste standardite kasutuselevõtt hoogustunud, näiteks käivitus Eestis 2004. aasta alguses DVB-T edastus). Seetõttu esitame siinkohal DVB süsteemi blokk skeemid ja kirjeldused, tuginedes DVB-S standardile.

Joonisel 25 on toodud põhiosad DVB süsteemis, mille hulka on arvestatud ka MPEG-2

kodeerimist teostav osa [25].



Joonis 25: DVB-S süsteemi blokk skeem [25]

Kuna MPEG-2 kodeerimisest tuleb juttu eraldi peatükis, siis siinkohal esitame kirjeldused vaid ülejäänud signaalitöötlustrakti blokkide kohta.

4.1.1 Kohandamine ja energia hajutamine

DVB standardit võib kasutada nii FSS (*Fixed Satellite Service*) kui ka BSS (*Broadcasting Satellite Service*) satelliitidel, mistõttu on vajalik andmevoo kohandamine konkreetse transpordri ribalaiuse ja parameetrite jaoks. Vastavad meetodid sõltuvad kasutatavast satelliidist ja edastatava teenuse vajadustest, näited kohandamiste kohta on toodud allika [25] lisas C.

Et süsteem oleks kooskõlas ITU raadioside regulatsioonidega, tuleb satelliidist väljakiiritava info energiajaotust ühtlustada. Selleks kasutatakse PRBS generaatorit, polünoomiga:

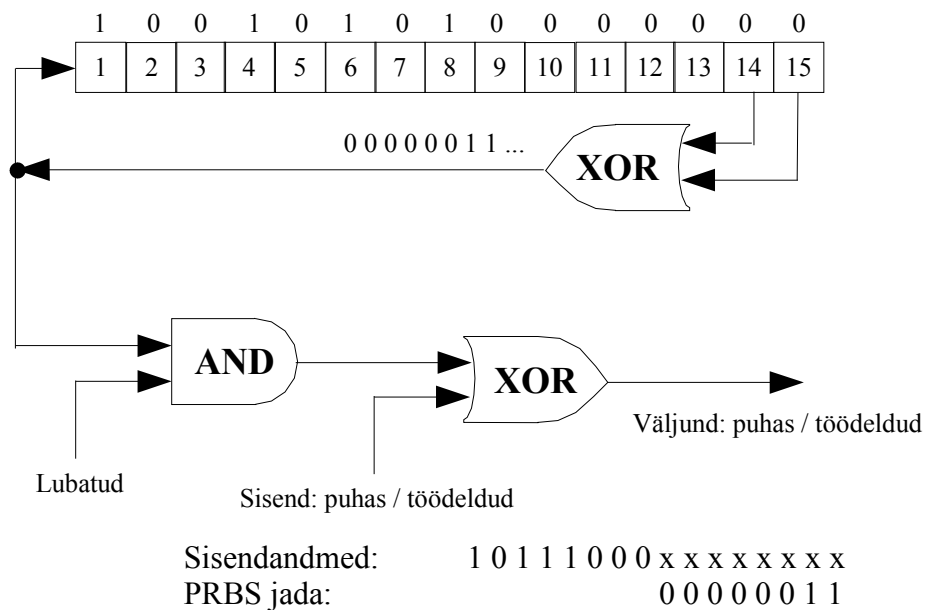
$$1 + X^{14} + X^{15} \quad (4.16)$$

MPEG-2 transportvoo paketid on 188 baidi pikkused, milles sisaldub ka paketi sünkrobait (47_{HEX})[22], mis märgistab uue transportpaketi algust. Töötlemisel alustatakse alati suurima tähtsusega bitist (MSB) ehk sünkrobaidi kahendesituse 01000111 puhul „0“.

Iga kaheksanda transportpaketi alguses laetakse generaatorpolünoom (4.16) kahendjadaga „100101010000000“. Vastuvõtja dekodeeris generaatori taaskäivitamise jaoks invertteeritakse iga kaheksas sünkrobait (saadakse $B8_{\text{HEX}}$) ja generaatori esimene väljundbitt rakendatakse invertteeritud sünkrobaidile järgneva baidi MSB'le.

Jooniselt 26 toodud skeemil on kujutatud olukorda, kus invertteeritud sünkrobait on juba saadetud

ja generaatori väljundit rakendatakse järgnevale kaheksale bitile.



Joonis 26: PRBS generaatori skeem [25]. Sama generaatorit kasutatakse nii saatjass kui vastuvõtjas.

Edasiste transportpakettide sünkrobaitide läbimise ajaks keelatakse generaatori väljund, kuid generaator jäetakse tööle (st sünkrobaidi bitid lastakse muutmata kujul läbi). Kuna iga kaheksanda sünkrobaidi korral generaator seisab, siis on generaatori perioodiks 1503 baiti.

4.1.2 Välimine kodeerimine

Edastamisvigade kõrvaldamiseks lisatakse igale transportpakatile ennetava veaparanduse (FEC) info. Nimetus *väline kodeerimine* tulenebki asjaolust, et infobaitidele lisatakse veakorrektsiooni baidid juurde.

DVB-S süsteemis on FEC meetodina kasutusel Reed-Solomoni kodeering RS(204,188), st 188-le infobaidile lisatakse 16 paarsusbaiti ja saadakse 204-baidine koodisõna.

$$g(x) = (x + \lambda^0) \cdot (x + \lambda^1) \cdot (x + \lambda^2) \cdot \dots \cdot (x + \lambda^{15}) \quad , \text{ kus } \lambda = 02_{HEX} \quad (4.17)$$

$$p(x) = x^8 + x^4 + x^3 + x^2 + 1 \quad (4.18)$$

Ülal (4.17) on RS koodi generaatorpolünoom ja (4.18) on paarsusbaitide generaator.

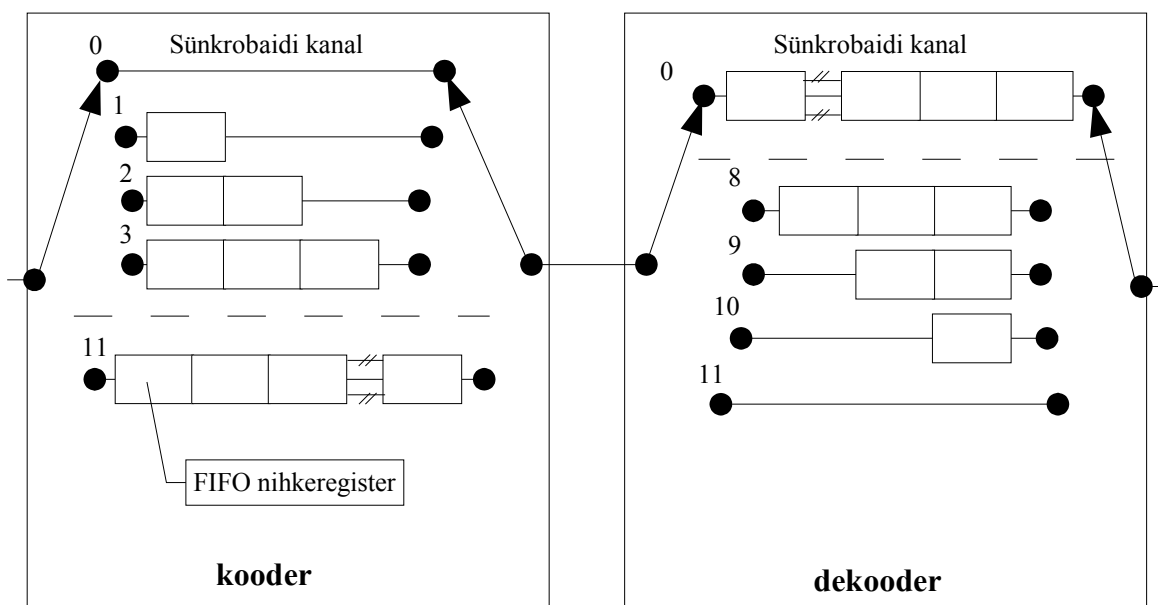
Lisatud baitide abil on vastuvõtja võimeline korrigeerima $\frac{204-188}{2} = 8$ baidiviga või taastama

16 kustutatud baiti, kui nende asukohad on teada [26]. RS kood sobib eriti hästi nn purskvigade (ingl. k. *burst error*) parandamiseks, kuna maksimaalse tihedusega bitivigade korral parandab RS

kood 64 vigast bitti, hajutatud vigade korral aga ainult 8 bitiviga.

4.1.3 Baitide ringvaheldi

Peale RS-kodeerimist läbivad transportpaketid tsüklilise baitide vahetamise protsessi (ingl. k. *convolutional interleaving*), mille põhimõtteskeem on toodud joonisel 27.



Joonis 27: Baitide ringvaheldi põhimõtteskeem [25]

Antud protsessis on kasutusel FIFO-tüüpi ühebaidised nihkeregistrid ja need on organiseeritud selliselt, et iga järgnev kanal on ühe registri võrra pikem, kui eelmine, alustades nullist. Kanaleid on kokku 12. Kanalite sisendid-väljundid on omavahel sünkroonis (st korraga sisestatakse ja loetakse samast kanalist).

Iga sünkrobait (nii invertteeritud kui invertteerimata) suunatakse *alati* nullindasse kanalisse, sellele järgnevad baidid järgmistesse kanalitesse, kuni ring täis ja alustatakse uuesti nullindast kanalist.

Vastuvõtja poolel suunatakse sünkrobaidid samuti alati nullindasse kanalisse, kuid vastupidiselt saatjale, tähendab nullis kanal suurimat viivitust.

Sellise baitide ümbertõstmise eesmärgiks on vähendada veelgi tundlikust purskvigade suhtes, mis tüüpi vigu tuleb eriti ette just satelliitsides.

4.1.4 Sisemine kodeerimine

Erinevalt RS koodist, kus info jäeti puutumata ja paarsusbitid lisati lõppu, modifitseerib sisemine kodeerimine just nimelt infobitte endid (sellest ka nimi). Selleks on kasutusel ringkodeering ja lubatud on tekitada 1/2 emakoodi asemel veel teisigi punkteeritud ringkoode kiirustega 2/3, 3/4, 5/6 ja 7/8. Koodikiirus siinkohal tähendab murdarvu, mille lugeja määrab ära sisendisse antud infobittide arvu ja nimetaja väljundist saadava koodibittide arvu, seega 1/2 kiirus tähendab kahekordset bittide arvu kasvu, 7/8 kiirus aga et iga seitsme infobiti kohta tuleb üks lisabitt juurde.

Punkteeritud koodide tekitamise tabel on toodud allikas [25]. Koodi dekodeerimine toimub Viterbi dekodeeri abil.

4.1.5 Põhiriba vormimine

Enne moduleerimist vormitakse edastatav signaal ruutjuur-tõstetud koosinusfiltriga, mille teoreetiline definitsioon on antud valemiga (4.19).

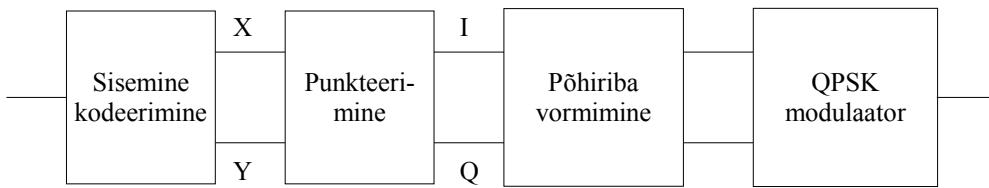
$$H(f) = \begin{cases} 1 & kui |f| < (f_N - \alpha) \\ \left[\frac{1}{2} + \frac{1}{2} \cdot \sin \frac{\pi}{2 \cdot f_N} \cdot \left(\frac{f_N - |f|}{\alpha} \right) \right]^{\frac{1}{2}} & kui (f_N - \alpha) \leq |f| \leq (f_N + \alpha) \\ 0 & kui |f| > (f_N + \alpha) \end{cases} \quad (4.19)$$

kus :

$$f_N = \frac{1}{2 \cdot T_s} \text{ on Nyquisti sagedus}$$
$$T_s \text{ on sümboli kestvus}$$
$$\alpha \text{ on langustegur, } \alpha = 0,35$$

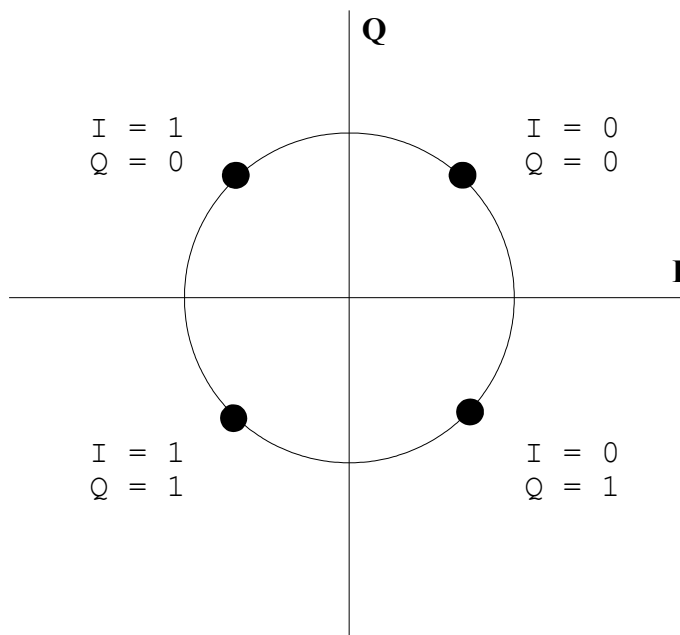
4.1.6 QPSK modulaator

QPSK modulaator kasutab sisenditena põhiriba vormimisest läbi käinud sisemise kodeerimise punkteeritud väljundeid (joonis 28).



Joonis 28: Sisemine kodeerimine ja QPSK moduleerimine [25]

QPSK moduleerimisel kasutatakse „tähtkuju“, mis on toodud joonisel 29.



Joonis 29: QPSK modulatsiooni tähtkuju [25]

5 KOKKUVÕTE

Antud töö eesmärgiks oli valmistada vahend telekommunikatsiooni aluste lektorile loengu esitamise ajal visuaalsete näidete toomiseks. Vajalik on demonstreerida diskreetse koosinusteisenduse omadusi nii ühe- kui kahemõõtmelisel juhul, samuti pakkuda võimalust kvantimise põhimõtete uurimiseks.

Töö käigus valmis tarkvara, mille abil on võimalik uurida koosinusteisenduse omadusi, sõltuvalt signaali kujust ja võrrelda signaali ühekaupa iga koosinusteisenduse baasiga. Sisendsignaali on võimalik tekitada nii hiire abil käsitsi väärtusi ette andes, kui ka lasta arvutil genereerida.

Samuti on võimalik uurida koosinusteisenduse kahemõõtmelist juhtu, valides erineva kujuga algsignaali maatrikseid ja võrreldes tulemuseks saadud koosinusteisendusi. Esitatakse ka vastuvõtjas kujutise taastamise põhimõte, näidates ära vajalikele koefitsientidele vastavad baasifunktsioonid ja taastatud pildid, mis saadaks, kui edastataks vaid väike hulk koefitsiente. Võrdluseks on toodud ka kõikide lugemite edastamisel saadav kadudeta taastatud pilt.

Kvantimist tutvustavas moodulis on programmi kasutajal võimalik valida erinevate mustritega piltide vahel, et võrrelda kvantimise sõltuvust sisendsignaali eripäradest. Samuti saab määrata kõiki kvantimises rakenduvaid parameetreid, sealhulgas muuta käsitsi kvantimise tabelit. Ühtlasi arvutab programm pidevalt välja hinnangulise andmeedastuskiiruse vajaduse selliste kvantimise parameetritega video edastamiseks (ei arvestata edasises signaalitöötluses lisatavat veakorrektsiooniinfot ega satelliitside jaoks vajalike signaale, vaid ainult konkreetse pildi edastamiseks kuluvat andmemahutu).

Kogu programmi koostamisel on arvestatud, et seda kasutatakse loengus näitliku õppevahendina. Et programm oleks tervikuna nähtav ka väiksemamõõtmeliste projektorite puhul, on üritatud kasutajaliides teha võimalikult kompaktsena ja ei ole lisatud kogu infot, mida on vaja valdkonna iseseisval omandamisel. Siiski, vajalike töölehtede või abimaterjalide olemasolul, on võimalik ka tudengitel iseseisvalt selle programmi abil praktilisi kogemusi omandada.

Seega on vajalike võimalustega programm loodud ja selle töö eesmärk täidetud.

Käesolevaga luban juhendajal kasutada selle töö tulemusi telekommunikatsiooni aluste loengus ja praktikumis õppevahendina.

6 KASUTATUD KIRJANDUS

1. *Digitaalringhäälingu kontseptsioon*, Majandus- ja Kommunikatsiooniministeerium, 2004
2. Ots, A. *Telekommunikatsiooni aluste praktikumi tööjuhendid*. Tartu Ülikool, 2002
3. *Simulating the Performance of Communication Links with Satellite Transponders*, http://www.applicationstrategy.com/Communications_simulation.htm, (06.05.2005)
4. Long, M. E. *The Digital Satellite TV Handbook*. Newnes, 1999
5. Ely, S. R. *MPEG video coding a simple introduction*. European Broadcasting Union, 1995
6. *Information Technology - Generic Coding of Moving Pictures and Associated Audio Information. Part 2: Video*, ISO/IEC International Standard IS 13818, 1994
7. *DigitaalTV*, <http://www.levira.ee/fldtv.html>, (29.05.2005)
8. Tucker, J. D. *The eyes, the ears and the brain - and how to cheat them*. European Broadcasting Union, 1997
9. *Sensitivity of the human eye*, <http://www.giangrandi.ch/optics/eye/eye.shtml>, (15.05.2005)
10. *The Human Eye*, <http://home.wanadoo.nl/paulschils/05.00.html>, (15.05.2005)
11. Hsu, H. P. *Schaum's outline of theory and problems of signals and systems*. McGraw-Hill, 1995
12. Ackenhusen, John G. *Real-time signal processing: design and implementation of signal processing systems*. Prentice Hall PTR, 1999
13. *Lecture Notes in Digital Image Processing*, <http://eeweb.poly.edu/~onur/lectures/lectures.html>, (29.05.2005)
14. Lyon, D. A. *Image processing in Java*. Prentice Hall PTR, 1999
15. *An efficient content-adaptive MC 3D_DWT with enhanced spatial and temporal scalability*, http://www.ee.unsw.edu.au/~taubman/publications_files/scalable-fully-adaptive-icip04.pdf, (2004)
16. Kingsbury, N. *The Discrete Cosine Transform (DCT)*. The Connexions Project, 2003
17. *A Beginners Guide for MPEG-2 Standard*, <http://www.fh-friedberg.de/fachbereiche/e2/telekom-labor/zinke/mk/mpeg2beg/beginnzi.htm>, (30.05.2005)
18. *XAPP615 "Quantization"*, <http://www.xilinx.com/bvdocs/appnotes/xapp615.pdf>, (09.05.2005)
20. Kiho, J. *Väike Java leksikon*. Tartu Ülikooli Kirjastus, 2000
19. *The Java Tutorial Continued: The Rest of the JDK*, <http://java.sun.com/docs/books/tutorial/2d/display/index.html>, (1998)
21. *Guideline for use of DVB standards ("Cookbook")*, <http://portal.etsi.org/broadcast/cookbook.asp>, (04.05.2005)
22. *Information Technology - Generic Coding of Moving Pictures and Associated Audio Information. Part 1: Systems*, ISO/IEC International Standard IS 13818, 1994
23. *Information Technology - Generic Coding of Moving Pictures and Associated Audio*

Information. Part 3: Audio, ISO/IEC International Standard IS 13818, 1994

24. *Implementation guidelines for the use of MPEG-2 Systems, Video and Audio in satellite, cable and terrestrial broadcasting applications*, European Telecommunications Standards Institute, 1997

25. *Framing structure, channel coding and modulation for 11/12 GHz satellite services*, European Telecommunications Standards Institute, 1997

26. *Reed-Solomon Codes*, http://www.4i2i.com/reed_solomon_codes.htm, (05.05.2005)

27. *E-teatmik*, <http://www.vallaste.ee/>, (05.05.2005)

7 SUMMARY

Computer simulation of DCT processing of DVB video signal

Today's fast-growing telecommunications market demands for better quality while not willing to pay for extra bandwidth, thus requiring advanced compression techniques. The University of Tartu teaches the principles of telecommunications to the students, mainly concentrating on the television and radio broadcasting, covering also the Digital Video Broadcasting (DVB).

The main goal of this thesis was to offer means to the lecturer to provide visual examples to the audience regarding the properties and basics of one and two-dimensional discrete cosine transform (DCT) as well as the quantization process, as these are the most vital parts of DVB and MPEG system.

The developed software offers possibilities to explore the properties of cosine transform depending on the form of the input signal. It is possible to compare the signal to each basis functions of the DCT. Different input signals may be created by either letting the computer to generate it or by drawing the signal by hand.

This principles of one dimensional DCT is also expanded to the domain of two dimensional images and again it is possible to demonstrate the DCT of different input signals. As the creating of two dimensional signal by hand is too complex for being usable live demonstrations, this module only offers different predefined signals and fully random signals. The program also demonstrates qualitative method of recreating the image at the receiving end by adding (or subtracting) the values of discrete cosine transform multiplied by the corresponding basis functions of DCT.

Quantization module demonstrates how the image can be compressed by knowingly modifying the values of DCT before transmission. In this module it is possible to choose between different input images and to set many parameters of quantization including modifying the entire quantization table and switching between conventional dividing and MPEG-2 quantization.

Because of the limited time of the lecture that is devoted to DVB, this software does not cover the coding of video, using interlacing and prediction.

8 KASUTATAVAD LÜHENDID

BSS	<i>Broadcasting Satellite Service</i> – ringhäälingu-satelliiditeenus, mille all mõistetakse sagedusribasid, mille kasutamine on reguleeritud rahvusvaheliste lepetega [27]. vt ka FSS.
DVB	<i>Digital Video Broadcasting</i> , digitaalse televisiooni edastamise standardite kogum
DVB-C	Kaabeltelevisiooni võrgus DVB pildi edastamise standard
DVB-S	Satelliidi abil DVB pildi edastamise standard
DVB-T	Maapealseid (<i>Terrestrial</i>) antennid DVB pildi edastamiseks kasutatav standard
FCC	<i>Federal Communications Commission</i> – USA telekommunikatsiooni ja side riigisest sektorit ja ka rahvusvahelist koostööd reguleeriv valitsusorganisatsioon.
FEC	<i>Forward Error Correction</i> – ennetav veaparandus
FIFO	<i>First In First Out</i> – registritüüp, kus esimesena sisestatud bait ka esimesena väljub
FSS	<i>Fixed Satellite Service</i> – fikssatelliiditeenus; sageli kasutatakse seda terminit ka sagedusribade tähistamiseks, mille kohta puuduvad rahvusvahelised satelliitringhäälingu lepped [27]. vt ka BSS
ISO	<i>International Organization for Standardization</i> – rahvusvaheline standardite organisatsioon.
ITU	<i>International Telecommunication Union</i> - rahvusvaheline valitsuste ja eraõiguslike ettevõtete organisatsioon, mis reguleerib telekommunikatsiooni võrkude ja teenuste kasutamist ja arendamist.
JPEG	<i>Joint Photographic Experts Group</i> – ISO töögrupi ja selle poolt väljatöötatud standardi nimetus piltide tihendamiseks [27]. vt ka MPEG
JRE	<i>Java Runtime Environment</i> – keskkond Java programmide käivitamiseks
JVM	<i>Java Virtual Machine</i> – Java virtuaalmasin on tarkvaraline arvutiemulaator, mis peidab Java programmide eest erinevate operatsioonisüsteemide iseärasused ja võimaldab neid programmeerida ühtsete vahenditega. Kõiki Java programme käivitatakse tegelikult JVM'i peal, mis tõlgib saadud käsud tegelikus protsessoris kasutatavasse keelde.
MPEG	<i>Motion Picture Experts Group</i> – ISO töögrupi ja selle poolt väljatöötatud videopakimise standardi nimi [27]. vt ka JPEG
MSB	<i>Most Significant Bit</i> – suurima tähtsusega bitt – arvu kahendesituses bitt, mille kaal on suurim; vastand LSB (<i>Least Significant Bit</i>) ehk vähimtähtis bitt – bitt, mille ümberpööramine muudab arvu kõige vähem.
PAL	<i>Phase Alternation Line</i> – üks levinuimatest televisioonistandarditest, kasutusel põhiliselt Euroopas (ka Eestis).

- PRBS** *Pseudo Random Binary Sequence* – kahendarvude jada, mis üritab jäljendada juhuslikku protsessi, kuid jäädes ise ikkagi ettearvatavaks (et saatja ja vastuvõtja generaatorid annaks samal ajahetkel välja sama tulemuse).
- QPSK** *Quadrature Phase Shift Keying* – kvadratuur-faasinihkemodulatsioon [27]
- RGB** *Red, Green, Blue* – värviruum, kus piksli info on esitatud kolme põhivärvi (punane, roheline, sinine) lugemite kogumina, igaüks neist 8-bitine, kokku 24-bitine värvisügavus.
- YCbCr** Värviruum, kus piksel info on esitatud kolme lugemina: Y – heleduse lugem, Cb ja Cr vastavalt sinisest ja punasest heleduse suhtes arvatud diferentslugemid.

9 LISA 1: PROGRAMMI KOOD

9.1 Pakett dvbsim

9.1.1 DVBSim.java

```
package dvbsim;

/*
 * Created on 3.04.2005
 *
 */

/**
 * @author Laas Toom
 *
 */
public class DVBSim {

    /**
     * Põhimeetod, mis käivitab kogu programmi
     * @param args
     */
    public static void main(String[] args) {
        MainProgram main = new MainProgram();
        main.setVisible(true);
    }
}
```

9.1.2 MainProgram.java

```
/*
 * Created on 9.04.2005
 *
 */
package dvbsim;

import javax.swing.JFrame;
import javax.swing.JScrollPane;
import javax.swing.JTabbedPane;

import dvbsim.dct1d.DCT1D;
import dvbsim.kvantimine.Kvantimine;
import dvbsim.lihtnedct.LihtneDCT;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class MainProgram extends JFrame {

    JTabbedPane tabbedPane;

    /**
     * konstruktor
     */
    public MainProgram() {
        this.initialize();
    }

    /**
     * Teeb põhiprogrammi algväärtuste seadmise
     */
}
```

```

    */
private void initialize() {
    this.setTitle("DVBSim");
    this.setSize(1024, 768);
    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    this.tabbedPane = new JTabbedPane();
    this.getContentPane().add(this.tabbedPane);

    //////////// DCT1D demo
    DCT1D dct1d = new DCT1D();
    JScrollPane dct1dScroll = new JScrollPane(dct1d.getDisplay());
    tabbedPane.addTab("Ühemõõtmeline DCT", dct1dScroll);

    //////////// Lihtne DCT
    LihtneDCT ldct = new LihtneDCT();

    JScrollPane LDCTScroll = new JScrollPane(ldct.getDisplay());
    tabbedPane.addTab("Kahemõõtmelise DCT näide", LDCTScroll);

    // Kvantimine
    Kvantimine kvant = new Kvantimine();
    JScrollPane kvantScroll = new JScrollPane(kvant.getDisplay());
    tabbedPane.addTab("Kvantimine", kvantScroll);

    this.pack();

}
}

```

9.1.3 DCT.java

```

/*
 * Created on 9.04.2005
 *
 */
package dvbsim;

/**
 * Klass diskreetse koosinusteisenduse tegemiseks.
 *
 * @author Laas Toom (laaz@laaz.org)
 */
public class DCT {

    private final double sqrt_2 = 1.4142135623730951;
    public final double[] C = new double[]{1.0/sqrt_2, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0};

    private double idct_out;
    private double dct_out;

    public double[][] cos_table = new double[8][8];
    public final int N = 8;

    public DCT(){
        this.init();
    }

    private void init(){
        for (int u = 0; u < N; u++){
            for (int x = 0; x < N; x++){

```

```

        cos_table[u][x] = Math.cos((double) (2*x+1) * (double) (u) *
(Math.PI/(2*N)));
    }
}

/**
 * Arvutab pildiinfo baasil lugemi DCT maatrixi punktis (u,v)
 *
 * @param u      - rea loendur
 * @param v      - veeru loendur
 * @param data   - 8x8 maatriks pildiinfoga
 * @return      tagastab DCT koefitsiendi (u,v);
 */
private final int dctLugem(int u, int v, int[][] data){
    // väljundi hoidja
    dct_out = 0;
    int i,j;
    for (i = 0; i < N; i++){
        for (j = 0; j < N; j++){
            dct_out += data[i][j] * cos_table[u][i] * cos_table[v][j];
        }
    }
    // eemaldatai Math.round()
    return (int)((dct_out * C[u] * C[v] * 2)/N);
}

private final int dctLugem(int u,int[] data){
    // väljundi hoidja
    dct_out = 0;
    int i,j;
    for (i = 0; i < N; i++){
        dct_out += data[i] * cos_table[u][i];
    }
    return (int)((dct_out * C[u] * 2)/N);
}

/**
 * Arvutab inversse DCT lugemi (ehk siis taastab DCT maatriksist pildiinfo)
 *
 * @param i      - pildi rea indeks
 * @param j      - pildi veeru indeks
 * @param data   - DCT lugemid
 * @return      tagastab pildiinfo punktis (i,j)
 */
private final int idctLugem(int i, int j, int[][] data){
    // väljundi hoidja
    idct_out = 0;
    int u,v;
    for (u = 0; u < N; u++){
        for (v = 0; v < N; v++){
            idct_out += C[u] * C[v] * data[u][v] * cos_table[u][i] *
cos_table[v][j];
        }
    }
    // eemaldatai Math.floor()
    return (int)(2 * idct_out /N);
}

/**
 * meetod DCT bloki arvutamiseks

```

```

    *
    * @param in
    * @return
    */
public int[][] getDCT(int[][] in){
    int[][] out = new int[in.length][in[0].length];
    for (int i = 0; i < in.length; i++){
        for (int j = 0; j < in[0].length; j++){
            out[i][j] = dctLugem(i,j,in);
        }
    }
    return out;
}

/**
 * meetod DCT vektori arvutamiseks
 *
 * @param in
 * @return
 */
public int[] getDCT(int[] in){
    int[] out = new int[in.length];
    for (int i = 0; i < in.length; i++){
        out[i] = dctLugem(i,in);
    }
    return out;
}

/**
 * meetod DCT blokist pildiinfo taastamiseks
 * @param in
 * @return
 */
public int[][] getIDCT(int[][] in){
    int[][] out = new int[in.length][in[0].length];
    for (int i = 0; i < in.length; i++){
        for (int j = 0; j < in[0].length; j++){
            out[i][j] = idctLugem(i,j,in);
        }
    }
    return out;
}
}

```

9.1.4 MatrixPanel.java

```

/*
 * Created on 9.04.2005
 *
 */
package dvbsim;

import java.awt.AlphaComposite;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.GradientPaint;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Polygon;
import java.awt.geom.AffineTransform;

import javax.swing.JPanel;

/**

```

```

* @author Laas Toom (laaz@laaz.org)
*/
public class MatrixPanel extends JPanel {

    // pilt, mida näidata (maatriks)
    int[][] image;

    // parameetrid
    int laius = 12;
    int maxVal = 255;
    int maxVal_title = maxVal;
    int realMax = 255; // skaleerimisel vahetatakse selle ja maxVal väärtusi
    int MAX_H = 40;
    double nurk = 40*Math.PI/180; // teisendame kraadid radiaanideks
    int samm = laius/2; // horis. samm
    int samm2 = laius*3/4; // vert. samm
    int laius2 = (int) (Math.cos(nurk)*laius*2/3);
    int korgus2 = (int) (Math.sin(nurk)*laius*2/3);

    int taheLaius = 5;
    int taheKorgus = 12;
    int mootKavaLaius = 10*taheLaius;

    // Värvid:
    public Color grad_varv1 = new Color(83,194,255);
    public Color grad_varv2 = new Color(223,231,244);
    public Color taustavarv = new Color(232,255,227);
    public GradientPaint grad_pos = new GradientPaint(-10,-
10,grad_varv1,20,20,grad_varv2,true);
    public GradientPaint grad_neg = new GradientPaint(-10,-
10,Color.GREEN,20,20,Color.YELLOW,true);

    // miinimum-mõõdud
    Dimension minSize;

    public MatrixPanel(int[][] im){
        this.image = im;
        this.minSize = new Dimension((laius+samm)*image.length + image[0].length
* (laius2+samm) + 10 + mootKavaLaius,2*MAX_H + (samm2) * image[0].length +
2*samm2 + 10 + taheKorgus);
        this.setMinimumSize(this.minSize);
        this.setPreferredSize(this.minSize);
    }

    public MatrixPanel(int[][] im, int max){
        this(im);
        setMaxVal(max);
        this.realMax = max;
    }

    public void setMatrix(int[][] mat){
        this.image = mat;
        this.updateUI();
    }

    public void paintComponent(Graphics g){
        super.paintComponent(g);
        dispMatrix(image,g);
    }

    public int[][] getMatrix(){
        return image;
    }
}

```

```

public void mahutaEkraanile(boolean b){
    if (b) this.maxVal = realMax;
    else this.maxVal = 255;
    maxVal_title = maxVal;
    updateUI();
}

private void dispMatrix(int[][] matrix, Graphics g){
    Graphics2D g2d = (Graphics2D)g;

    // joonistamise alguspunkt (arvestama peab laius ja sammuga)
    g2d.translate((laius+samm)*matrix[0].length - laius,0);

    // nihutame algset punkti veel, et ka mõõtkava peale mahuks
    g2d.translate(mootKavaLaius,taheKorgus);

    AffineTransform algTrans = g2d.getTransform(); // algne
transformatsioon, lõpus taastame

    // Joonistame esmalt negatiivsed väärtused, seejärel tasandi
// ning viimasena positiivsed väärtused

    // negatiivsed
    for (int i =0; i<matrix.length; i++){
        for (int j = matrix[i].length -1; j >= 0; j--){
            int value = matrix[i][j];
            if (value < 0)
                drawBar3D(value, grad_neg,g2d);
            g2d.translate(-(laius + samm), 0);
        }
        g2d.translate((laius+samm)*matrix[i].length+laius,samm2);
    }
    // taastame algse asukoha:
    g2d.setTransform(algTrans);

    // joonistame aluse
    Polygon alus = new Polygon();
    int l,k;
    alus.addPoint(l = laius, k = MAX_H);
    alus.addPoint(l -= matrix[0].length*(laius+samm) - samm ,k);
    alus.addPoint(l += matrix.length*laius - laius + laius2,k +=
matrix.length*(samm2)-samm2 + korgus2);
    alus.addPoint(l += matrix[0].length*(laius+samm)-samm,k);

    algTrans = g2d.getTransform(); // algne transformatsioon, lõpus
taastame
    g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
(float)0.7));
    g2d.setPaint(taustavarv);
    g2d.fill(alus);
    g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
(float)1));
    g2d.setPaint(Color.BLACK);
    g2d.draw(alus);
    // taastame algse transformatsiooni
    g2d.setTransform(algTrans);

    // positiivsed
    for (int i =0; i<matrix.length; i++){
        for (int j = matrix[i].length -1; j >= 0; j--){
            int value = matrix[i][j];

```

```

        if (value >= 0)
            drawBar3D(value, grad_pos,g2d);
        g2d.translate(-(laius + samm), 0);
    }
    g2d.translate((laius+samm)*matrix[i].length+laius,samm2);
}

// taastame algse transformatsiooni
g2d.setTransform(algTrans);
g2d.translate(-matrix[0].length*(laius+samm) + laius +samm,matrix[0].
length*(samm2)+samm2);
drawBar3D((int) (0.2*maxVal), grad_pos,g2d);
g2d.drawString("Positiivsed",laius + 2*samm,MAX_H + korgus2);
g2d.translate(0,10+samm2);
drawBar3D((int) (0.2*maxVal), grad_neg,g2d);
g2d.drawString("Negatiivsed",laius + 2*samm,MAX_H + korgus2);

// Joonistame mõõtkava:
// g2d.translate(matrix.length*(laius+samm) - samm,0);
g2d.setTransform(algTrans);
g2d.translate(-matrix[0].length*(laius+samm)+ laius,0);
g2d.drawLine(0,0,-5,0);
g2d.drawLine(0,0,0,MAX_H);
g2d.translate(0,MAX_H);
g2d.drawLine(0,0,-5,0);
g2d.drawLine(0,0,0,MAX_H);
g2d.translate(0,MAX_H);
g2d.drawLine(0,0,-5,0);
g2d.setTransform(algTrans);
g2d.drawString(String.valueOf(maxVal_title),-matrix[0].length*
(laius+samm) - String.valueOf(maxVal_title).length()*taheLaius,taheKorgus/2);
g2d.drawString("0",-matrix[0].length*(laius+samm) ,taheKorgus/2 +
MAX_H);
g2d.drawString(String.valueOf(-maxVal_title),-matrix[0].length*
(laius+samm) - String.valueOf(-maxVal_title).length()*taheLaius,taheKorgus/2 +
2*MAX_H);
}

private void drawBar3D(int value,GradientPaint grad,Graphics2D g2d){
    //----- JOONISTAMISE ALGUS
    -----
    //algne transformatsioon, lõpus taastame
    AffineTransform algTrans = g2d.getTransform();

    // defineerime baasristkülikud:
    int korgus = (Math.abs(value) * MAX_H) / maxVal;
    boolean neg = (value < 0)? true:false;

    // polügon, mis kujutab endast risttahuka 3D-vaadet
    Polygon piirjooned = new Polygon();
    piirjooned.addPoint(0,0);
    piirjooned.addPoint(laius,0);
    piirjooned.addPoint(laius + laius2,korgus2);
    piirjooned.addPoint(laius + laius2,korgus2+korgus);
    piirjooned.addPoint(laius2,korgus2+korgus);
    piirjooned.addPoint(0,korgus);

    // teine polügon puuduvate joonte tõmbamiseks
    Polygon sisejooned = new Polygon();
    sisejooned.addPoint(0,0);
    sisejooned.addPoint(laius2,korgus2);
    sisejooned.addPoint(laius2+laius,korgus2);
    sisejooned.addPoint(laius2,korgus2); // läheme tagasi eelmisse punkti
    sisejooned.addPoint(laius2,korgus2+korgus);

```

```

        sisejooned.addPoint(laius2,korgus2);    // jälle tagasi

        // liigume kõrguses õigesse kohta:
        if (neg){
            g2d.translate(0,MAX_H);
            neg = false;
        }
        else
            g2d.translate(0,MAX_H-korgus);

        // joonistame kujundid
        g2d.setPaint(grad);
        g2d.fill(piiirjooned);
        g2d.setPaint(Color.BLACK);
        g2d.draw(piiirjooned);
        g2d.draw(sisejooned);

        // taastame algse transformatsiooni
        g2d.setTransform(algTrans);

        //----- JOONISTAMISE LÕPP
-----
    }

    /**
     * @return Returns the image.
     */
    public int[][] getImage() {
        return image;
    }
    /**
     * @param image The image to set.
     */
    public void setImage(int[][] image) {
        this.image = image;
    }
    /**
     * @return Returns the maxVal.
     */
    public int getMaxVal() {
        return maxVal;
    }
    /**
     * @param maxVal The maxVal to set.
     */
    public void setMaxVal(int maxVal) {
        this.maxVal = maxVal;
    }

    public void setMaxValTitle(int mvt){
        this.maxVal_title = mvt;
    }
}

```

9.1.5 TablePanel.java

```

/*
 * Created on 13.04.2005
 *
 */
package dvbsim;

```

```

import java.awt.BorderLayout;
import java.awt.Dimension;

import javax.swing.JPanel;
import javax.swing.JTable;
import javax.swing.table.TableColumn;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class TablePanel extends JPanel {

    int[][] matrix;
    JTable table;
    Dimension size = new Dimension(300,155);
    boolean enabled = false;
    IntegerEditor editor = new IntegerEditor(1,2047);

    public TablePanel(int[][] m){
        super(new BorderLayout());
        this.setMinimumSize(size);
        this.setPreferredSize(size);
        this.setMaximumSize(size);
        this.setMatrix(m);
    }

    public void setMatrix(int[][] t){
        this.matrix = t;
        // kui tabel puudub või mõõtmed erinevad, siis tuleb uus tabel teha.
        if (this.table == null || this.table.getRowCount() !=
this.matrix.length || this.table.getColumnCount() != this.matrix[0].length){
            this.table = new JTable(this.matrix.length, this.matrix[0].
length);
        }
        this.removeAll();
        this.add(table);

        createUI();
    }

    public int[][] getMatrix(){
        int rows = table.getRowCount();
        int cols = table.getColumnCount();
        int[][] out = new int[rows][cols];
        for (int i = 0; i < rows; i++){
            for (int j = 0; j < cols; j++){
                out[i][j] = ((Integer)table.getValueAt(i,j)).intValue();
            }
        }
        return out;
    }

    public void setTableEnabled(boolean enabled){
        this.table.setEnabled(enabled);
        this.enabled = enabled;
        // kui tabel keelatakse, tühistame valikud ja editorid
        if (! enabled){
            this.table.clearSelection();
            if (table.isEditing())
                table.removeEditor();
        }
        else{
            table.changeSelection(0,0,false,false);
        }
    }
}

```

```

    }

    void createUI(){
        TableColumn c;
        for (int i = 0; i < matrix.length; i++) {
            for (int j = 0; j < matrix[0].length; j++) {
                this.table.setValueAt(new Integer(matrix[i][j]),i,j);
                c = this.table.getColumnModel().getColumn(j);
                // seame lahtrile kontrolliva editori
                c.setCellEditor(editor);
            }
        }
        this.table.setEnabled(enabled);
    }
}

```

9.1.6 IntegerEditor.java

```

/*
 * IntegerEditor is a 1.4 class used by TableFTFEditDemo.java.
 */
package dvbsim;

import java.awt.Component;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
import java.text.NumberFormat;
import java.text.ParseException;

import javax.swing.AbstractAction;
import javax.swing.DefaultCellEditor;
import javax.swing.JFormattedTextField;
import javax.swing.JTable;
import javax.swing.JTextField;
import javax.swing.KeyStroke;
import javax.swing.text.DefaultFormatterFactory;
import javax.swing.text.NumberFormatter;

/**
 * Võetud Sun'i õpetusest "How to Use Tables"
 * (http://java.sun.com/docs/books/tutorial/uiswing/components/table.html).<br>
 * Tõlgitud ja kohandatud.
 *
 * Implements a cell editor that uses a formatted text field
 * to edit Integer values.
 */
public class IntegerEditor extends DefaultCellEditor {
    JFormattedTextField ftf;
    NumberFormat integerFormat;
    private Integer minimum, maximum;
    private boolean DEBUG = false;

    public IntegerEditor(int min, int max) {
        super(new JFormattedTextField());
        ftf = (JFormattedTextField) getComponent();
        minimum = new Integer(min);
        maximum = new Integer(max);

        //Set up the editor for the integer cells.
        integerFormat = NumberFormat.getIntegerInstance();
        NumberFormatter intFormatter = new NumberFormatter(integerFormat);

```

```

intFormatter.setFormat(integerFormat);
intFormatter.setMinimum(minimum);
intFormatter.setMaximum(maximum);

ftf.setFormatterFactory(
    new DefaultFormatterFactory(intFormatter));
ftf.setValue(minimum);
ftf.setHorizontalAlignment(JTextField.TRAILING);
ftf.setFocusLostBehavior(JFormattedTextField.PERSIST);

//React when the user presses Enter while the editor is
//active. (Tab is handled as specified by
//JFormattedTextField's focusLostBehavior property.)
ftf.getInputMap().put(KeyStroke.getKeyStroke(
    KeyEvent.VK_ENTER, 0),
    "check");
ftf.getActionMap().put("check", new AbstractAction() {
    public void actionPerformed(ActionEvent e) {
        if (!ftf.isEditValid()) { //The text is invalid.
            if (userSaysRevert()) { //reverted
                ftf.postActionEvent(); //inform the editor
            }
        } else try { //The text is valid,
            ftf.commitEdit(); //so use it.
            ftf.postActionEvent(); //stop editing
        } catch (java.text.ParseException exc) { }
    }
});
}

//Override to invoke setValue on the formatted text field.
public Component getTableCellEditorComponent(JTable table,
    Object value, boolean isSelected,
    int row, int column) {
    JFormattedTextField ftf =
        (JFormattedTextField)super.getTableCellEditorComponent(
            table, value, isSelected, row, column);
    ftf.setValue(value);
    return ftf;
}

//Override to ensure that the value remains an Integer.
public Object getCellEditorValue() {
    JFormattedTextField ftf = (JFormattedTextField)getComponent();
    Object o = ftf.getValue();
    if (o instanceof Integer) {
        return o;
    } else if (o instanceof Number) {
        return new Integer(((Number)o).intValue());
    } else {
        if (DEBUG) {
            System.out.println("getCellEditorValue: o isn't a Number");
        }
        try {
            return integerFormat.parseObject(o.toString());
        } catch (ParseException exc) {
            System.err.println("getCellEditorValue: can't parse o: " + o);
            return null;
        }
    }
}

//Override to check whether the edit is valid,
//setting the value if it is and complaining if
//it isn't. If it's OK for the editor to go
//away, we need to invoke the superclass's version

```

```

//of this method so that everything gets cleaned up.
public boolean stopCellEditing() {
    JFormattedTextField ftf = (JFormattedTextField)getComponent();
    if (ftf.isEditValid()) {
        try {
            ftf.commitEdit();
        } catch (java.text.ParseException exc) { }

    } else { //text is invalid
        if (!userSaysRevert()) { //user wants to edit
            return false; //don't let the editor go away
        }
    }
    return super.stopCellEditing();
}

/**
 * Muudetud: ei küsi kasutajalt, vaid alati taastab vana väärtuse.
 */
protected boolean userSaysRevert() {
    Toolkit.getDefaultToolkit().beep();
    // me ei taha dialoogi, vaid lihtsalt vana väärtuse taastamist
    ftf.setValue(ftf.getValue());
    return true;
}
}

```

9.2 Pakett dvbsim.dct1d

9.2.1 DCT1D.java

```

/*
 * Created on 10.05.2005
 *
 */
package dvbsim.dct1d;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.ButtonGroup;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTable;

import dvbsim.DCT;

```

```

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class DCT1D extends DCT{

    int maxVal = 255;
    int[] sisend = new int[8];
    int[] dct = new int[8];

    JPanel komponendiValikAlus = new JPanel();
    ButtonGroup komponendiValik = new ButtonGroup();
    VectorPanel sisendiKuva;
    VectorPanel komponendiKuva;
    VectorPanel dctKuva;
    DoubleVectorPanel dctJaSisendiKuva;
    BigPixelPanel sisendPixels;
    BigPixelPanel dctPixels;

    int defWidth = 250;
    int defHeight = 150;

    int[][] dct_table = new int[8][8];
    protected int valitudKomponent = 0;

    boolean sisendHasNeg = true;

    JPanel panel;
    String dctTekst = "<html>DCT kujutis (DCT<sub>i</sub></html>";
    JLabel dctSilt;
    JTable dctTabel;
    JLabel dctValem;

    Color sisendColor = Color.BLUE;
    Color baasColor = Color.RED;
    Color dctColor = Color.GREEN;

    /**
     * Vaikekonstruktor
     */
    public DCT1D(){
        super();
        this.init();
    }

    /**
     * Tekitab isendi tööks vajalikud algväärtused.
     */
    private void init(){
        komponendiValikAlus.setLayout(new GridLayout(0,4));

        JRadioButton radio;
        for (int i = 0; i<8; i++){
            // seame komponendi algväärtuse
            this.sisend[i] = maxVal;
            for (int j = 0; j < 8; j++)
                dct_table[i][j] = (int) (Math.cos((double)(2*j+1) * (double)(i
* (Math.PI/(2*8))) * 255));

            radio = new JRadioButton(String.valueOf(i));
            radio.setToolTipText("Vali DCT baas number " + i);
            if (i == valitudKomponent)

```

```

        radio.setSelected(true);
        radio.setActionCommand(String.valueOf(i));
        radio.addActionListener(new KomponendiValikListener(this));
        komponendiValik.add(radio);
        komponendiValikAlus.add(radio);

    }
    this.dct = getDCT(this.sisend);
    sisendPixels = new BigPixelPanel(sisend, defWidth, true, 255);
    dctPixels = new BigPixelPanel(dct, defWidth, false, 360);
}

public void setSisendHasNeg(boolean b){
    this.sisendHasNeg = b;
    if (! b){
        for (int i = 0; i < sisend.length; i++) {
            if (sisend[i] < 0)
                sisend[i] = 0;
        }
    }
    sisendiKuva.setHasNeg(sisendHasNeg);
    sisendiKuva.setVector(sisend);
    dctJaSisendiKuva.setVector(sisend);
    sisendPixels.setHasNeg(b);
    sisendPixels.setVector(sisend);
}

public void updateDCT(){
    dct = getDCT(sisend);
    dctKuva.highlightItem = valitudKomponent;
    dctKuva.setVector(dct);
    dctJaSisendiKuva.setVector(sisend,1);
    komponendiKuva.setVector(dct_table[valitudKomponent]);
    dctJaSisendiKuva.setVector(dct_table[valitudKomponent],2);
    dctPixels.setVector(dct);
    for (int i = 0; i < dct.length; i++) {
        dctTabel.setValueAt(String.valueOf(dct[i]),0,i);
    }
}

public void setDCTValem(){
    if (dctValem == null){
        dctValem = new JLabel(new ImageIcon()); // kui näidata siin, et
teksti ei tule, paigutatakse ikoon keskele
        dctValem.setBorder(BorderFactory.createTitledBorder("DCT baasi
valem"));
    }
    dctValem.setIcon(new ImageIcon(loadImage("images/dct-" +
valitudKomponent + ".png")));
    dctValem.setToolTipText("<html>DCT koefitsiendi DCT<sub>" +
valitudKomponent + "</sub> arvutamise valem");
}

/**
 * Tagastab JPanel objekti, millele on lisatud kogu selle mooduli sisu
 * @return JPanel objekt sisuga
 */
public JPanel getDisplay(){
    panel = new JPanel();
    panel.setLayout(new GridBagLayout());
    GridBagConstraints c = new GridBagConstraints();

    JLabel sisendiSilt = new JLabel("<html>Sisendpikslite lugemid

```

```

<i>x<sub>i</sub></i>:"");
    sisendiKuva = new VectorPanel(sisend,255,sisendHasNeg, defWidth,
defHeight);
    sisendiKuva.setColor(sisendColor);
    SisendiHiireAdapter sha = new SisendiHiireAdapter(this,sisendiKuva);
    sisendiKuva.addMouseListener(sha);
    sisendiKuva.addMouseListener(sha);

    JLabel komponendiSilt = new JLabel("Valitud DCT baas:");
    komponendiKuva = new VectorPanel(dct_table[valitudKomponent],255,true,
defWidth, defHeight);
    komponendiKuva.setColor(baasColor);
    komponendiKuva.setTransparency((float)0.7);

    komponendiValikAlus.setBorder(BorderFactory.createTitledBorder("Vali DCT
baas:"));
    komponendiValikAlus.setPreferredSize(new Dimension(defWidth,
defHeight/2));

    dctSilt = new JLabel(dctTekst);
    dctKuva = new VectorPanel(dct,360,true, defWidth, defHeight);
    dctKuva.setColor(dctColor);
    dctKuva.setHighLightColor(sisendColor);

    JLabel dctJaSisendSilt = new JLabel("Sisend ja DCT baas koos:");
    dctJaSisendiKuva = new DoubleVectorPanel(sisend,255,true, defWidth,
defHeight);
    dctJaSisendiKuva.setColors(sisendColor,baasColor);
    dctJaSisendiKuva.setTransparency((float)0.7);
    dctJaSisendiKuva.setToolTipText("<html>Graafikul on mustaga kujutatud
sisendi <b>kaalutud väärtust</b></html>");
    dctJaSisendiKuva.setVector(dct_table[valitudKomponent],2);
    dctJaSisendiKuva.addMouseListener(sha);
    dctJaSisendiKuva.addMouseMotionListener(sha);

    dctTabel = new JTable(1,8);
    dctTabel.setPreferredSize(new Dimension(defWidth,15));
    dctTabel.setEnabled(false);
    updateDCT();

    JPanel controls = createControls();

    JLabel info = new JLabel(
        "<html><p>Sisendi graafikutel saab hiire<br>" +
        "abil väärtusi muuta</p><br>" +
        "<p>Sisendi ja DCT graafikul kujutatud <br>" +
        "värvide tähendused:" +
        "<ul><li><font color='blue'>Sisendpikslite lugemid</font></li>"
+
        "<li><font color='red'>DCT baas</font></li>" +
        "<li>Must joon - sisendi kaalutud väärtus</li>" +
        "</ul></p>" +
        "<p>Arusaadavuse huvides on punane värv<br>" +
        "pisut läbipaistev.</p></html>");
    info.setBorder(BorderFactory.createTitledBorder("Abiinfo:"));

    setDCTValem();

    // Sildid
    c.gridx = 0;
    c.gridy = 0;
    c.anchor = GridBagConstraints.LAST_LINE_START;
    panel.add(sisendiSilt,c);

```

```

c.gridy = 5;
panel.add(dctSilt,c);

c.gridx = 2;
c.gridy = 0;
panel.add(komponendiSilt,c);

c.gridy = 5;
panel.add(dctJaSisendSilt,c);

// graafikud
c.anchor = GridBagConstraints.FIRST_LINE_START;
c.gridx = 0;
c.gridy = 4;
panel.add(sisendPixels,c);

c.gridy = 9;
panel.add(dctPixels,c);

c.gridy = 10;
panel.add(dctTabel,c);

c.gridx = 0;
c.gridy = 1;
c.gridheight = 3;
panel.add(sisendiKuva,c);

c.gridy = 6;
panel.add(dctKuva,c);

c.gridx = 2;
c.gridy = 1;
panel.add(komponendiKuva,c);

c.gridy = 6;
panel.add(dctJaSisendiKuva,c);

// kontrollinupud
c.fill = GridBagConstraints.HORIZONTAL;

c.gridx=4;
c.gridy=1;
c.gridheight = 4;
panel.add(controls,c);

c.gridx = 4;
c.gridy = 5;
c.gridheight = 2;
panel.add(dctValem,c);

c.gridy = 7;
c.gridheight=5;
panel.add(info,c);

// vahed
c.gridx = 1;
c.gridy = 0;

```

```

panel.add(Box.createHorizontalStrut(20),c);

c.gridx = 3;
panel.add(Box.createHorizontalStrut(20),c);

return panel;
}

private JPanel createControls(){
    JPanel alus = new JPanel(new GridBagLayout());
    JPanel controls = new JPanel(new BorderLayout());
    GridBagConstraints gbc = new GridBagConstraints();
    gbc.gridx = 0;
    gbc.gridy = 0;
    gbc.fill = GridBagConstraints.HORIZONTAL;

    ControlListener cl = new ControlListener(this);

    JCheckBox cb = new JCheckBox("Negatiivsed väärtused lubatud");
    cb.setToolTipText("<html>Sisendis on lubatud negatiivsete väärtuste
kasutamine</html>");
    cb.setActionCommand("sisendi-neg");
    cb.addItemListener(new CheckBoxAdapter(this));
    cb.setSelected(sisendHasNeg);
    alus.add(cb,gbc);

    JButton nupp = new JButton("Pikseli lugemite inverteerimine");
    nupp.setActionCommand("peegel-ver");
    nupp.addActionListener(cl);
    nupp.setToolTipText("<html>Kui negatiivsed väärtused lubatud, siis
x<sub>i</sub> = -x<sub>i</sub><br>" +
        "Kui lubatud ainult positiivsed, siis x<sub>i</sub> = " +
maxVal + " - x<sub>i</sub>");
    gbc.gridy++;
    alus.add(nupp,gbc);

    /*
    nupp = new JButton("Peegelda horisontaalselt");
    nupp.setActionCommand("peegel-hor");
    nupp.addActionListener(cl);
    gbc.gridy++;
    alus.add(nupp,gbc);
    */

    nupp = new JButton("Genereeri signaal");
    nupp.setToolTipText("Seab sisendiks arvuti genereeritud juhusliku
signaali");
    nupp.setActionCommand("gen");
    nupp.addActionListener(cl);
    gbc.gridy++;
    alus.add(nupp,gbc);

    nupp = new JButton("Samasta DCT baasiga");
    nupp.setToolTipText("Seab sisendiks signaali, mille kuju on sarnane
valitud DCT baasile");
    nupp.setActionCommand("dct");
    nupp.addActionListener(cl);
    gbc.gridy++;
    alus.add(nupp,gbc);

    alus.setBorder(BorderFactory.createTitledBorder("Sisendi muutmine:"));

```

```

        controls.add(alus, BorderLayout.CENTER);
        controls.add(komponendiValikAlus, BorderLayout.SOUTH);
        return controls;
    }

    /**
     * Muutuste toimumisel uuenda vajalike komponentide vaateid.
     */
    public void updateUI(){
        sisendiKuva.setVector(sisend);
        sisendPixels.setVector(sisend);
        updateDCT();
    }

    public BufferedImage loadImage(String pildiFail){
        BufferedImage tempIm = new BufferedImage
(100,100,BufferedImage.TYPE_INT_ARGB);
        try {
            tempIm = ImageIO.read(new File(pildiFail));
        }
        catch(IOException ioe){
            System.err.println(ioe.getMessage());
            JOptionPane.showMessageDialog(panel,
                "Programmi tööks vajalikku pildifaili (" + pildiFail + ") ei
leitud.\n" +
                "Vajuta OK, et programm sulgeda.",
                "VIGA: Faili ei leitud!",
                JOptionPane.ERROR_MESSAGE);
            System.exit(1);
        }
        return tempIm;
    }
}

```

9.2.2 VectorPanel.java

```

/*
 * Created on 10.05.2005
 *
 */
package dvbsim.dct1d;

import java.awt.AlphaComposite;
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.Shape;
import java.awt.Stroke;
import java.awt.geom.AffineTransform;

import javax.swing.JPanel;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class VectorPanel extends JPanel {

```

```

int[] vector;

Dimension size = new Dimension(200,150);

int maxVal = 255;
int bar_step = 10;
int barW = (int) ((size.getWidth()/8) - bar_step);
int barH = (int) (size.getHeight()/2);

boolean hasNeg = true;

Color barColor = Color.BLUE;
Color highlightColor = Color.GREEN;
Rectangle bar;
Rectangle highlightBar;

int alg_x = bar_step/2;
int alg_y = barH;

int highlightItem = -1;
float transparency = 1;

boolean firstTime = true;

/**
 * Vaikekonstruktor, mis seab vektori nullideks.
 */
public VectorPanel(){
    this(new int[]{0,0,0,0,0,0,0,0});
}

/**
 * konstruktor, mis seab vektori, mida kuvada.
 * @param v vektor, mida kuvada
 */
public VectorPanel(int[] v){
    this.vector = v;
    this.setSize();
}

/**
 * Konstruktor, mis seab vektori ja selles esineva maksimumväärtuse.
 * maxVal ei pea olema tegelik maksimumväärtus, vaid seab mõõtkava (maxVal
<-> tulba kõrgus)
 *
 * @param v vektor, mida kuvatakse
 * @param maxVal maksimumväärtus, mis võib vektoris esineda. Selle järgi
skaleeritakse vektor
 */
public VectorPanel(int[] v, int maxVal){
    this(v);
    this.maxVal = maxVal;
}

/**
 * Konstruktor, mis seab vektori, maksimumväärtuse ja määrab, kas
negatiivsed väärtused on lubatud.
 * Negatiivsete väärtuste puudumisel ei joonistata graafikul alumist poolt -
hoiab ruumi kokku
 *
 * @param v vektor, mida kuvatakse
 * @param maxVal maksimumväärtus
 * @param hasNeg kas negatiivseid väärtused on lubatud

```

```

    */
public VectorPanel(int[] v, int maxVal, boolean hasNeg){
    this(v,maxVal);
    this.hasNeg = hasNeg;
    // vaja uuesti välja kutsuda, kuna hasNeg võis muutuda
    this.setSize();
}

/**
 * Konstruktor, mis seab lisaks muule ka veel paneeli mõõdud
 * @param v
 * @param maxVal
 * @param hasNeg
 * @param w
 * @param h
 */
public VectorPanel(int[] v, int maxVal, boolean hasNeg, int w, int h){
    this(v,maxVal,hasNeg);
    this.setSize(w,h);
}

public void setColor(Color c){
    this.barColor = c;
}

public void setHighLightColor(Color c){
    this.highlightColor = c;
}

public void setTransparency(float transp){
    this.transparency = transp;
}

protected void setSize(){
    this.setPreferredSize(size);
    this.setMinimumSize(size);

    barW = (int) ((size.getWidth()/8) - bar_step);

    this.barH = (int) size.getHeight()/2;
}

public void setSize(int w, int h){
    this.size = new Dimension(w,h);
    this.setSize();
}

/**
 * Seab kuvamiseks uue vektori
 * @param v vektor, mida kuvada
 */
public void setVector(int[] v){
    this.vector = v;
    updateUI();
}

public void setHasNeg(boolean b){
    this.hasNeg = b;
}

/**
 * Kontrollib, kas punkt (x,y) on mingi tulba peal.
 * Kui on, tagastab positsiooni ja klikkimise y koordinaadi alusel tulba
väärtuse, muidu (-1,0)

```

```

    * @return komponendi indeks või -1
    */
    public int[] getIndexAndValue(int x, int y){
        int[] out = new int[]{-1,0};
        int neg = 0;
        if (hasNeg)
            neg=1;
        if ((x >= alg_x && x < (vector.length * (barW + bar_step))) && (y >= 0
&& y <= (barH + neg * barH) )){
            out[0] = (x-alg_x)/(barW + bar_step);
            if(y < alg_y)
                out[1] = alg_y - y;
            else
                out[1] = (y - alg_y) * -1;
        }
        out[1] = out[1]*maxVal/barH;
        return out;
    }

    public void paintComponent(Graphics g){
        Graphics2D g2d = (Graphics2D) g;

        if(firstTime){
            bar = new Rectangle(0,0,barW,barH);
            highlightBar = new Rectangle(-bar_step/2,-
bar_step/2,barW+bar_step,barH+bar_step);
            firstTime = false;
        }

        // kustutame vana sisu
        g2d.setColor(g2d.getBackground());
        g2d.fillRect(0,0,this.getWidth(),this.getHeight());

        g2d.translate(alg_x,alg_y);

        AffineTransform algTrans = g2d.getTransform();
        g2d.setColor(barColor);
        for (int i = 0; i < vector.length; i++) {
            int h = (int)((Math.abs(vector[i]) * barH)/maxVal);
            bar.setSize(barW,h);
            AffineTransform trans = g2d.getTransform();
            if (vector[i] > 0)
                g2d.translate(0,-1*h);

            g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
transparency));
            g2d.fill(bar);
            g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
1));

            if (i == highlightItem){
                //highlightBar.setSize(barW+bar_step,h+ bar_step);
                Stroke algStroke = g2d.getStroke();
                BasicStroke stroke = new BasicStroke(1);
                Shape s = stroke.createStrokedShape(bar);
                g2d.setColor(highlightColor);
                g2d.setStroke(stroke);
                g2d.draw(s);
                g2d.setColor(barColor);
                g2d.setStroke(algStroke);
            }

            g2d.setTransform(trans);
            g2d.translate(bar_step + barW, 0);
        }
        g2d.setTransform(algTrans);
    }

```

```

        g2d.setColor(Color.BLACK);
        g2d.drawLine(0,0,vector.length * (barW + bar_step) + bar_step,0);
    }
}

```

9.2.3 DoubleVectorPanel.java

```

/*
 * Created on 11.05.2005
 *
 */
package dvbsim.dct1d;

import java.awt.AlphaComposite;
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.Stroke;
import java.awt.geom.AffineTransform;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class DoubleVectorPanel extends VectorPanel {

    //algselt on teine vektor nullitud
    int[] vector2 = new int[]{0,0,0,0,0,0,0,0};

    Color bar2Color = Color.RED;

    /**
     * Vaikekonstruktor, mis seab vektori nullideks.
     */
    public DoubleVectorPanel(){
        this(new int[]{0,0,0,0,0,0,0,0});
    }

    /**
     * konstruktor, mis seab vektori, mida kuvada.
     * @param v vektor, mida kuvada
     */
    public DoubleVectorPanel(int[] v){
        this.vector = v;
        this.setSize();
    }

    /**
     * Konstruktor, mis seab vektori ja selles esineva maksimumväärtuse.
     * maxVal ei pea olema tegelik maksimumväärtus, vaid seab mõõtkava (maxVal <-
     > tulba kõrgus)
     *
     * @param v vektor, mida kuvatakse
     * @param maxVal maksimumväärtus, mis võib vektoris esineda. Selle järgi

```

```

skaleeritakse vektor
*/
public DoubleVectorPanel(int[] v, int maxVal){
    this(v);
    this.maxVal = maxVal;
}

/**
 * Konstruktor, mis seab vektori, maksimumväärtuse ja määrab, kas negatiivsed
väärtused on lubatud.
 * Negatiivsete väärtuste puudumisel ei joonistata graafikul alumist poolt -
hoiab ruumi kokku
 *
 * @param v vektor, mida kuvatakse
 * @param maxVal maksimumväärtus
 * @param hasNeg kas negatiivseid väärtused on lubatud
 */
public DoubleVectorPanel(int[] v, int maxVal, boolean hasNeg){
    this(v,maxVal);
    this.hasNeg = hasNeg;
    // vaja uuesti välja kutsuda, kuna hasNeg võis muutuda
    this.setSize();
}

/**
 * Konstruktor, mis seab lisaks muule ka veel paneeli mõõdud
 * @param v
 * @param maxVal
 * @param hasNeg
 * @param w
 * @param h
 */
public DoubleVectorPanel(int[] v, int maxVal, boolean hasNeg, int w, int h){
    this(v,maxVal,hasNeg);
    this.setSize(w,h);
}

public void setColors(Color c1, Color c2){
    this.barColor = c1;
    this.bar2Color = c2;
}

public void setVector(int[] v, int num){
    if (num == 1)
        setVector(v);
    else
        vector2 = v;
    this.updateUI();
}

public void paintComponent(Graphics g){
    Graphics2D g2d = (Graphics2D) g;

    if(firstTime){
        bar = new Rectangle(0,0,barW,barH);

        firstTime = false;
    }

//    kustutame vana sisu
g2d.setColor(g2d.getBackground());
g2d.fillRect(0,0,this.getWidth(),this.getHeight());

g2d.translate(alg_x,alg_y);

```

```

AffineTransform algTrans = g2d.getTransform();
for (int i = 0; i < Math.min(vector.length,vector2.length); i++){
    g2d.setColor(barColor);
    int h = (int)((Math.abs(vector[i]) * barH)/maxVal);
    bar.setSize(barW,h);
    AffineTransform trans = g2d.getTransform();
    if (vector[i] > 0)
        g2d.translate(0,-1*h);
    g2d.fill(bar);

    // teine tulp
    g2d.setTransform(trans);
    h = (int)((Math.abs(vector2[i]) * barH)/maxVal);
    bar.setSize(barW,h);
    g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
transparency));
    g2d.setColor(bar2Color);
    if (vector2[i] > 0)
        g2d.translate(0,-1*h);
    g2d.fill(bar);
    g2d.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
(float)1));

    g2d.setTransform(trans);
    g2d.translate(barW/2, 0);
    g2d.setColor(Color.BLACK);
    h = (int)((vector[i]*(double)((double)vector2[i]/255) * barH)/
maxVal);

    Stroke stroke = g2d.getStroke();
    g2d.setStroke(new BasicStroke(2));
    g2d.drawLine(0,0,0,-1*h);
    g2d.drawLine(-bar_step/2,-1*h,bar_step/2,-1*h);
    g2d.setStroke(stroke);

    g2d.setTransform(trans);
    g2d.translate(bar_step + barW, 0);
}
g2d.setTransform(algTrans);
g2d.setColor(Color.BLACK);
g2d.drawLine(0,0,vector.length * (barW + bar_step) + bar_step,0);
}
}

```

9.2.4 SisendiHiireAdapter.java

```

/*
 * Created on 10.05.2005
 *
 */
package dvbsim.dct1d;

import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionAdapter;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class SisendiHiireAdapter extends MouseMotionAdapter implements
MouseListener {

    DCT1D vanem;

```

```

VectorPanel source;

public SisendiHiireAdapter(DCT1D vanem, VectorPanel vp){
    this.vanem = vanem;
    this.source = vp;
}

public void mouseDragged(MouseEvent me) {
    int x = me.getX();
    int y = me.getY();
    int[] iv = source.getIndexAndValue(x,y);
    if (iv[0] >= 0){
        vanem.sisend[iv[0]] = iv[1];
        vanem.updateUI();
    }
    source.highlightItem = iv[0];
}

public void mouseMoved(MouseEvent me){
    int x = me.getX();
    int y = me.getY();
    int[] iv = source.getIndexAndValue(x,y);
    source.highlightItem = iv[0];
    source.updateUI();
    if (iv[0] >=0)
        source.setToolTipText("<html>Väärtus = <b>" + vanem.sisend[iv[0]] +
"</b></html>");
}

/* (non-Javadoc)
 * @see java.awt.event.MouseListener#mouseExited(java.awt.event.MouseEvent)
 */
public void mouseExited(MouseEvent me) {
    source.highlightItem = -1;
    source.updateUI();
}

/* (non-Javadoc)
 * @see java.awt.event.MouseListener#mouseClicked(java.awt.event.MouseEvent)
 */
public void mouseClicked(MouseEvent me) {
}

/* (non-Javadoc)
 * @see java.awt.event.MouseListener#mousePressed(java.awt.event.MouseEvent)
 */
public void mousePressed(MouseEvent me) {
    int x = me.getX();
    int y = me.getY();
    int[] iv = source.getIndexAndValue(x,y);
    if (iv[0] >= 0){
        vanem.sisend[iv[0]] = iv[1];
        vanem.updateUI();
    }
}

/* (non-Javadoc)
 * @see java.awt.event.MouseListener#mouseReleased
(java.awt.event.MouseEvent)
 */
public void mouseReleased(MouseEvent arg0) {
    // ei kasutata
}

```

```

    }

    /* (non-Javadoc)
     * @see java.awt.event.MouseListener#mouseEntered(java.awt.event.MouseEvent)
     */
    public void mouseEntered(MouseEvent me) {
        int x = me.getX();
        int y = me.getY();
        int[] iv = source.getIndexAndValue(x,y);
        source.highlightItem = iv[0];
        source.updateUI();
    }

}

```

9.2.5 KomponendiValikListener.java

```

/*
 * Created on 10.05.2005
 *
 */
package dvbsim.dctlD;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class KomponendiValikListener implements ActionListener {

    DCTL1D vanem;

    public KomponendiValikListener(DCTL1D vanem){
        this.vanem = vanem;
    }

    /* (non-Javadoc)
     * @see java.awt.event.ActionListener#actionPerformed
     (java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent ae) {
        int komp = Integer.valueOf(ae.getActionCommand()).intValue();
        vanem.valitudKomponent = komp;
        vanem.updateDCT();
        vanem.setDCTValem();
    }
}

```

9.2.6 ControlListener.java

```

/*
 * Created on 10.05.2005
 *
 */
package dvbsim.dctlD;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**

```

```

* @author Laas Toom (laaz@laaz.org)
*/
public class ControlListener implements ActionListener {

    DCT1D vanem;

    public ControlListener(DCT1D vanem){
        this.vanem = vanem;
    }

    /* (non-Javadoc)
    * @see java.awt.event.ActionListener#actionPerformed
    (java.awt.event.ActionEvent)
    */
    public void actionPerformed(ActionEvent ae) {
        String cmd = ae.getActionCommand();
        if (cmd.equalsIgnoreCase("peegel-ver")){
            for (int i = 0; i < vanem.sisend.length; i++) {
                if (vanem.sisendHasNeg)
                    vanem.sisend[i] = -1 * vanem.sisend[i];
                else
                    vanem.sisend[i] = vanem.maxVal - vanem.sisend[i];
            }
        }
        else if (cmd.equalsIgnoreCase("peegel-hor")){
            int[] temp = new int[vanem.sisend.length];
            for (int i = 0; i < vanem.sisend.length; i++) {
                temp[i] = vanem.sisend[vanem.sisend.length -1 - i];
            }
            vanem.sisend = temp;
        }
        else if (cmd.equalsIgnoreCase("dct")){
            for (int i = 0; i < vanem.sisend.length; i++) {
                if (vanem.sisendHasNeg)
                    vanem.sisend[i] = vanem.dct_table[vanem.valitudKomponent]
[i];
                else
                    vanem.sisend[i] = vanem.dct_table[vanem.valitudKomponent]
[i]/2 + 128;
            }
        }
        else if (cmd.equalsIgnoreCase("gen")){
            int x=0;
            for (int i = 0; i < vanem.sisend.length; i++) {
                if (vanem.sisendHasNeg){
                    vanem.sisend[i] = (int) (Math.random()*512-256);
                }
                else{
                    vanem.sisend[i] = (int) (Math.random()*255);
                }
            }
        }
        vanem.updateUI();
    }
}

```

9.2.7 CheckBoxAdapter.java

```

/*
* Created on 11.05.2005
*
*/

```

```

package dvbsim.dct1d;

import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import javax.swing.JCheckBox;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class CheckBoxAdapter implements ItemListener {

    DCT1D vanem;

    public CheckBoxAdapter(DCT1D vanem){
        this.vanem = vanem;
    }

    /* (non-Javadoc)
     * @see java.awt.event.ItemListener#itemStateChanged
     (java.awt.event.ItemEvent)
     */
    public void itemStateChanged(ItemEvent ie) {
        String cmd = ((JCheckBox)(ie.getItem())).getActionCommand();
        if(cmd.equalsIgnoreCase("sisendi-neg")){
            vanem.setSisendHasNeg(((JCheckBox)(ie.getItem())).isSelected());
        }
    }
}

```

9.2.8 BigPixelPanel.java

```

/*
 * Created on 21.05.2005
 *
 */
package dvbsim.dct1d;

import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;

import javax.swing.JPanel;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class BigPixelPanel extends JPanel {

    int[] vector;
    int width;
    int rectW;
    int step = 10;
    int maxVal = 255;
    Dimension size;
    Rectangle rect;

    boolean has_neg = true;
}

```

```

public BigPixelPanel(int[] vector, int width){
    this.vector = vector;
    this.width = width;
    this.setSize();
}

public BigPixelPanel(int[] v, int w, boolean neg){
    this(v,w);
    this.has_neg = neg;
}

public BigPixelPanel(int[] v, int w, boolean neg, int max){
    this(v,w,neg);
    this.maxVal = max;
}

public void setSize(){
    size = new Dimension(width, width/vector.length);
    rectW = width/8 - step;
    rect = new Rectangle(rectW, rectW);
    this.setPreferredSize(size);
}

public void setVector(int[] v){
    vector = v;
    this.setSize();
    updateUI();
}

public void setHasNeg(boolean neg){
    this.has_neg = neg;
    updateUI();
}

public void paintComponent(Graphics g){
    Graphics2D g2d = (Graphics2D) g;
    int value = 0;

    // kustutame vana sisu
    g2d.setColor(g2d.getBackground());
    g2d.fillRect(0,0,this.getWidth(),this.getHeight());

    g2d.translate(step/2,step/2);
    for (int i = 0; i < vector.length; i++) {
        if (has_neg){
            value = vector[i]*127 / maxVal + 128;
        }
        else {
            value = Math.abs(vector[i])*255/maxVal;
        }
        g2d.setColor(new Color(value, value, value));
        g2d.fill(rect);
        g2d.setColor(Color.BLACK);
        g2d.draw(rect);
        g2d.translate(rectW + step,0);
    }
}

/**
 * @return Returns the maxVal.
 */
public int getMaxVal() {
    return maxVal;
}

```

```

/**
 * @param maxVal The maxVal to set.
 */
public void setMaxVal(int maxVal) {
    this.maxVal = maxVal;
}
}

```

9.3 Paketti dvbsim.lihtnedct

9.3.1 LihtneDCT.java

```

/*
 * Created on 9.04.2005
 *
 */
package dvbsim.lihtnedct;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Component;
import java.awt.Dimension;
import java.awt.Graphics2D;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.Insets;
import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;
import java.awt.event.KeyEvent;
import java.awt.geom.AffineTransform;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JDialog;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.SwingConstants;

import dvbsim.DCT;
import dvbsim.MatrixPanel;
import dvbsim.TablePanel;
import dvbsim.kvantimine.ImagePanel;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class LihtneDCT extends DCT implements ActionListener, ItemListener{

    int[][] whiteImage;
    int[][] customImage;

```

```

int[][] rndImage;
int[][] sujuvImage = new int[8][8];
int[][] sujuvImage2d = new int[8][8];
int[][] asteImage = new int[8][8];
int[][] veerandidImage = new int[8][8];
int[][] diagonaalImage = new int[8][8];
int[][] ristImage = new int[8][8];

int[][] dct;

JPanel panel;

Object[] pildid;
int rndPos = 0;

String[] piltideNimed = {
    "Ühtlane",
    "Sujuv tõus",
    "2-suunaline sujuv tõus",
    "Aste",
    "Veerandid",
    "Diagonaal",
    "Rist",
    "Suvalised väärtused"
};

int MAX_Value = 255;

MatrixPanel originaal_pilt;
MatrixPanel dct_pilt;
JComboBox pildiValik;
TablePanel originaal_tabel;
TablePanel dct_tabel;

BufferedImage[][] im;
BufferedImage blackIm;
final int img_kylg = 32;
final int nupu_kylg = img_kylg + 6;
final int kylg = img_kylg / 8;
Rectangle rect = new Rectangle(0,0,kylg,kylg);
int[][][][] nupu_dct_tabel;
JDialog infoDialog;
JPanel button_panel = new JPanel(new GridBagLayout());
JButton[][] buttons;

JDialog vvDialog;
int[] zu =
{0,0,1,2,1,0,0,1,2,3,4,3,2,1,0,0,1,2,3,4,5,6,5,4,3,2,1,0,0,1,2,3,4,5,6,7,7,6,5,4
,3,2,1,2,3,4,5,6,7,7,6,5,4,3,4,5,6,7,7,6,4,6,7,7};
int[] zv =
{0,1,0,0,1,2,3,2,1,0,0,1,2,3,4,5,4,3,2,1,0,0,1,2,3,4,5,6,7,6,5,4,3,2,1,0,1,2,3,4
,5,6,7,7,6,5,4,3,2,3,4,5,6,7,7,6,5,4,5,6,7,7,6,7};

int[][] vvDCT;

/**
 * Konstruktor
 */
public LihtneDCT(){
    createImages();
    this.customImage = rndImage;
}

/**
 * Konstruktor

```

```

*
* @param im - etteantud pilt
*/
public LihtneDCT(int[][] im){
    this();
    this.customImage = im;
}

private void createImages(){
    this.whiteImage = new int[][]{
        {255,255,255,255,255,255,255,255},
        {255,255,255,255,255,255,255,255},
        {255,255,255,255,255,255,255,255},
        {255,255,255,255,255,255,255,255},
        {255,255,255,255,255,255,255,255},
        {255,255,255,255,255,255,255,255},
        {255,255,255,255,255,255,255,255},
        {255,255,255,255,255,255,255,255}
    };

    for (int i = 0; i < 8; i++){
        for (int j = 0; j < 8; j++){
            sujuvImage[i][j] = (int) (j * 36.4);
            sujuvImage2d[i][j] = (int) (i*j*5.2);
            if (j < 4)
                asteImage[i][j] = 0;
            else
                asteImage[i][j] = 255;
            if (i == j)
                diagonaalImage[i][j] = 255;
            else
                diagonaalImage[i][j] = 0;
            if ((i < 4 && j < 4) || (i >= 4 && j >= 4))
                veerandidImage[i][j] = 0;
            else
                veerandidImage[i][j] = 255;
            if (i == 4 || j == 4)
                ristImage[i][j] = 255;
            else
                ristImage[i][j] = 0;
        }
    }

    generateRndImage(false);

    pildid = new Object[]{
        whiteImage,
        sujuvImage,
        sujuvImage2d,
        asteImage,
        veerandidImage,
        diagonaalImage,
        ristImage,
        rndImage
    };
}

void generateRndImage(boolean toArray){
    this.rndImage = new int[N][N];
    for (int i = 0; i < N; i++){

```

```

        for (int j =0; j < N; j++){
            rndImage[i][j] = (int) (Math.random()*MAX_Value);
        }
    }

    if(toArray) pildid[pildid.length-1] = rndImage;
}

public JPanel getDisplay(){
    panel = new JPanel(new GridBagLayout());
    GridBagConstraints c = new GridBagConstraints();
    c.insets = new Insets(10,10,0,0);

    initButtons(button_panel);
    dct = getDCT((int[][]pildid[0]);
    updateButtons();
    button_panel.setBorder(BorderFactory.createTitledBorder("DCT
baasifunksioonid"));

    // Originaalpilt
    originaal_pilt = new MatrixPanel((int[][]pildid[0]);
    // originaalpildi peakiri
    JLabel originaal_silt = new JLabel("Originaalkujutis:");
    // piltide valimise nupp
    pildiValik = new JComboBox(piltideNimed);
    pildiValik.setActionCommand("pildivalik");
    pildiValik.addActionListener(this);
    originaal_tabel = new TablePanel(originaal_pilt.getImage());
    originaal_tabel.setBorder(BorderFactory.createTitledBorder
("Originaalkujutise heledused:"));

    // DCT kujutis
    dct_pilt = new MatrixPanel(this.getDCT((int[][]pildid[0]),2047);
    JLabel dct_silt = new JLabel("DCT kujutis:");
    dct_tabel = new TablePanel(dct_pilt.getImage());
    dct_tabel.setBorder(BorderFactory.createTitledBorder("DCT
koefitsiendid:"));
    JCheckBox dct_skaleerida = new JCheckBox("Vähendatud mõõtkava");
    dct_skaleerida.setToolTipText("<html>DCT kujutise originaaliga samas
skaalas nägemiseks<br>eemalda linnuke</html>");
    dct_skaleerida.addItemListener(this);
    dct_skaleerida.setMnemonic(KeyEvent.VK_M);
    dct_skaleerida.setSelected(true);

    JButton vvButton = new JButton("Näita kujutise koostamist vastuvõtjas");
    vvButton.setActionCommand("vastuvotja");
    vvButton.addActionListener(this);

    ImageIcon zzsagedused = new ImageIcon(loadImage("images/zz-
sagedused.png"));
    JLabel zzSagedusedSilt = new JLabel("<html>DCT sagedused kasvavad<br>" +
        "vasakult ülalt alla paremale<br>" +
        "(joonisel roheliselt punaseks).<br>" +
        "Nool näitab edastamisel<br>" +
        "kasutatavat sikk-sakk esituse<br>" +
        "koefitsientide
järjekorda.</html>", zzsagedused, JLabel.CENTER);
    zzSagedusedSilt.setBorder(BorderFactory.createTitledBorder("Saatmisel
kasutatav sikk-sakk esitus"));

    JLabel dctValem = new JLabel(new ImageIcon(loadImage("images/2d-

```

```

dct.png"));
    dctValem.setBorder(BorderFactory.createTitledBorder("DCT valem"));

    c.anchor = GridBagConstraints.PAGE_START;
    //c.fill = GridBagConstraints.BOTH;
    c.fill = GridBagConstraints.HORIZONTAL;

    c.gridx = 0;
    c.gridy = 0;
    c.gridwidth = 1;
    c.gridheight = 1;
    panel.add(pildiValik,c);
    c.gridy = 1;
    panel.add(originaal_silt,c);
    c.gridy = 2;
    c.gridwidth = 2; // kaks tulpa lai
    c.gridheight = 1;
    panel.add(originaal_pilt,c);

    c.gridx = 0;
    c.gridy = 3;
    c.gridwidth = 2;
    c.gridheight = 2;
    panel.add(originaal_tabel,c);

    c.gridx = 2;
    c.gridy = 0;
    c.gridwidth = 1;
    c.gridheight = 1;
    panel.add(dct_skaleerida,c);
    c.gridy=1;
    panel.add(dct_silt,c);
    c.gridy=2;
    c.gridwidth = 2;
    c.gridheight = 1;
    panel.add(dct_pilt,c);

    c.gridwidth = 2;
    c.gridheight = 2;
    c.gridy = 3;
    panel.add(dct_tabel,c);

    c.gridx = 4;
    c.gridy = 1;
    c.gridheight = 4;
    c.gridwidth = 1;
    panel.add(button_panel,c);

    //c.fill = GridBagConstraints.HORIZONTAL;

    c.gridx = 4;
    c.gridy = 0;
    c.gridheight = 1;
    c.gridwidth = 1;
    panel.add(vvButton,c);

    c.gridx = 4;
    c.gridy = 5;
    c.gridwidth = 1;
    c.gridheight = 1;
    panel.add(zzSagedusedSilt,c);

    c.gridx = 0;
    c.gridy = 5;

```

```

        c.gridwidth = 4;
        c.gridheight = 1;
        panel.add(dctValem,c);

        //c.gridy = 9;
        //panel.add(Box.createVerticalStrut(20),c);

        //panel.setBorder(BorderFactory.createTitledBorder("Lihtne DCT
näide:"));
        return panel;
    }

    /**
     * Reageerib nupuvajutusele
     * @see java.awt.event.ActionListener#actionPerformed
     (java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent ae) {
        String cmd = ae.getActionCommand();
        if (cmd.equalsIgnoreCase("pildivalik")){
            if (pildiValik.getSelectedIndex() == this.pildid.length-1){
                generateRndImage(true);
            }
            dct = getDCT((int[][]pildid[pildiValik.getSelectedIndex()]);
            originaal_pilt.setMatrix((int[][]pildid[pildiValik.getSelectedIndex
()]);
            originaal_tabel.setMatrix((int[][]pildid
[pildiValik.getSelectedIndex()]);
            dct_pilt.setMatrix(dct);
            dct_tabel.setMatrix(dct);
            updateButtons();
        }
        else if (cmd.equalsIgnoreCase("vastuvotja")){
            displayVVDialog();
        }
        else {
            displayDialog(cmd);
        }
    }

    /* (non-Javadoc)
     * @see java.awt.event.ItemListener#itemStateChanged
     (java.awt.event.ItemEvent)
     */
    public void itemStateChanged(ItemEvent arg0) {
        // kuna meil checkboxe üks, siis ei pea uurima, kellega tegu
        if (arg0.getStateChange() == ItemEvent.DESELECTED)
            dct_pilt.mahutaEkraanile(false);
        else
            dct_pilt.mahutaEkraanile(true);
    }

    /**
     * initsialiseerib nupud
     */
    private void initButtons(JPanel panel){
        panel.setLayout(new GridLayout(8,8));
        buttons = new JButton[8][8];
        im = new BufferedImage[8][8];
        nupu_dct_tabel = new int[8][8][8][8];
    }

```

```

        for (int u = 0; u < 8; u++) {
            for (int v = 0; v < 8; v++){
                im[u][v] = new BufferedImage
(img_kylg, img_kylg, BufferedImage.TYPE_INT_RGB);
                Graphics2D g2d = im[u][v].createGraphics();
                AffineTransform alg = g2d.getTransform();
                for (int i = 0; i < 8; i++){
                    g2d.translate(0, i * kylg);
                    for (int j = 0; j < 8; j++){
                        nupu_dct_tabel[u][v][i][j] = (int) (Math.cos((double)
(2*i+1) * (double)(u) * (Math.PI/(2*N)))
                        * Math.cos((double)(2*j+1) * (double)(v) *
(Math.PI/(2*N))) * 255);
                        int color = nupu_dct_tabel[u][v][i][j]/2 + 128;
                        g2d.setPaint(new Color(getRGBInt(color, color, color)));
                        g2d.fill(rect);
                        g2d.translate(kylg, 0);
                    }
                    g2d.setTransform(alg);
                }
                buttons[u][v] = new JButton(new ImageIcon(im[u][v]));
                buttons[u][v].setPreferredSize(new Dimension
(nupu_kylg, nupu_kylg));
                buttons[u][v].setToolTipText("<html>Baasfunksioon: <b>(" + u +
", " + v + ")</b><br>" +
                "Kliki, et avada aken infoga.</html>");
                buttons[u][v].setActionCommand(u + "-" + v);
                buttons[u][v].addActionListener(this);
                panel.add(buttons[u][v]);
            }
        }
        blackIm = new BufferedImage
(img_kylg, img_kylg, BufferedImage.TYPE_INT_RGB);
        Graphics2D g2d = blackIm.createGraphics();
        g2d.setColor(Color.BLACK);
        g2d.fillRect(0, 0, img_kylg, img_kylg);
    }

    private void updateButtons(){
        for (int i = 0; i < buttons.length; i++) {
            for (int j = 0; j < buttons[i].length; j++) {
                if(dct[i][j] == 0){
                    buttons[i][j].setIcon(null);
                }
                else{
                    buttons[i][j].setIcon(new ImageIcon(im[i][j]));
                }
            }
        }
    }
}

/**
 * Tagastab 32-bitise ARGB täisarvu, kus Alpha on 255.
 * @param red
 * @param green
 * @param blue
 * @return 32-bitine ARGB täisarv
 */
public int getRGBInt(int red, int green, int blue){
    return getRGBInt(red, green, blue, 0);
}

/**

```

```

* Tagastab 32-bitise ARGB täisarvu
*
* @param red
* @param green
* @param bblue
* @param alpha
* @return
*/
public int getRGBInt(int red, int green, int blue, int alpha){
    return (alpha << 24) | (red << 16) | (green << 8) | blue;
}

public int[][] korrutaNumbriliselt(int[][] a, int[][] b){
    int[][] out = new int[8][8];
    if (a.length != b.length || a[0].length != b[0].length || a.length != 8)
        throw new IllegalArgumentException("Korrutatavad matriksid ei ole
samade mõõtmetega!");
    for (int i = 0; i < a.length; i++) {
        for (int j = 0; j < a[i].length; j++) {
            out[i][j] = (int) (((double)a[i][j]/255.0) * b[i][j]);
        }
    }
    return out;
}

public void displayDialog(String s){
    String[] cmd = s.split("-");
    int u = Integer.valueOf(cmd[0]).intValue();
    int v = Integer.valueOf(cmd[1]).intValue();

    if (infoDialog == null)
        infoDialog = new JDialog((JFrame)panel.getTopLevelAncestor(), "DCT
Baasfunktsioon (" + u + ", " + v + ")");
    else
        infoDialog.getContentPane().removeAll();
    infoDialog.setTitle("DCT Baasfunktsioon (" + u + ", " + v + ")");
    infoDialog.getContentPane().setLayout(new GridBagLayout());
    GridBagConstraints c2 = new GridBagConstraints();

    MatrixPanel dct = new MatrixPanel(nupu_dct_tabel[u][v], 255);
    dct.setMaxValTitle(1);
    JLabel silt1 = new JLabel("DCT Baasfunktsioon (" + u + ", " + v + ")");

    MatrixPanel orig = new MatrixPanel(originaal_pilt.getImage(), 255);
    JLabel silt2 = new JLabel("Algne kujutis:");

    MatrixPanel dct_ja_sisend = new MatrixPanel(korrutaNumbriliselt
(nupu_dct_tabel[u][v], originaal_pilt.getImage()), 255);
    JLabel silt3 = new JLabel("Korrutis elementide kaupa:");

    JLabel text = new JLabel("<html>Baasfunktsioon: <b>(" + u + ", " + v +
")</b><br>" +
        "'korrelatsioon' kujutisega: <b>" + dct_pilt.getMatrix()[u][v]
+ "</b></html>");
    text.setHorizontalAlignment(SwingConstants.CENTER);
    text.setBorder(BorderFactory.createLineBorder(Color.BLACK));

    c2.gridx = 0;
    c2.gridy=0;
    c2.anchor = GridBagConstraints.FIRST_LINE_START;
    infoDialog.getContentPane().add(silt2, c2);

```

```

c2.gridx = 2;
infoDialog.getContentPane().add(silt1,c2);

c2.gridx = 0;
c2.gridy = 3;
infoDialog.getContentPane().add(silt3,c2);

c2.fill = GridBagConstraints.BOTH;
c2.gridx = 2;
infoDialog.getContentPane().add(text,c2);

c2.gridheight = 2;
c2.gridx = 0;
c2.gridy = 1;
infoDialog.getContentPane().add(orig,c2);

c2.gridx = 2;
infoDialog.getContentPane().add(dct,c2);

c2.gridx = 0;
c2.gridy = 4;
infoDialog.getContentPane().add(dct_ja_sisend,c2);

infoDialog.pack();
infoDialog.setVisible(true);
}

private BufferedImage createImage(int[][] a){
    int[][] b = getIDCT(a);
    int w = 5;
    BufferedImage im = new BufferedImage(8*w, 8*w,
BufferedImage.TYPE_INT_RGB);
    Graphics2D g2d = im.createGraphics();
    Rectangle rect = new Rectangle(w,w);
    for (int i = 0; i < 8; i++){
        for (int j = 0; j < 8; j++){
            int value = fitBounds(b[i][j],0,255);
            g2d.setColor(new Color(value,value,value));
            g2d.fill(rect);
            g2d.translate(w,0);
        }
        g2d.translate(-8*w,w);
    }
    return im;
}

private int fitBounds(int x, int min, int max){
    if (x < min) return min;
    else if (x > max) return max;
    else return x;
}

private BufferedImage createImage(int u, int v){
    vvDCT[u][v] = dct[u][v];
    return createImage(vvDCT);
}

private BufferedImage createFullImage(){
    return createImage(dct);
}

private String getSign(int x){
    return (x < 0) ? "-" : "+";
}

```

```

private int[][] skaleeri(int[][] a, double b){
    int[][] out = new int[8][8];
    for (int i = 0; i < a.length; i++) {
        for (int j = 0; j < a[i].length; j++) {
            out[i][j] = (int) (a[i][j] * b);
        }
    }
    return out;
}

private void displayVVDialog(){
    if (vvDialog == null){
        vvDialog = new JDialog((JFrame)panel.getTopLevelAncestor(),
"Vastuvõtjas kujutise taastamine");
        vvDialog.getContentPane().setLayout(new BorderLayout());
    }
    else {
        vvDialog.getContentPane().removeAll();
    }

    int elemente = 8;
    JPanel alus1 = new JPanel(new GridLayout(0,8));
    Box alus2;
    JPanel alus3 = new JPanel(new GridLayout(0,8,5,5));

    vvDCT = new int[8][8];
    int u, v;
    // Võtame vaid mõned esimesed baasid
    int count = 0;
    int i = -1;
    while (++i < 64){
        u = zu[i];
        v = zv[i];
        // näitame välja ainult nullist erinevaid lugemeid
        if (dct[u][v] == 0 )
            continue; // läheme uuele ringile

        // piirame näidatavate lugemite arvu
        if(count >= elemente){
            // kui ka see element on nullist erinev, siis lisame punktiiri
            if (dct[u][v] != 0 )
                count++;
            break; // lõpetame while tsükli
        }
        count++;
        alus2 = Box.createVerticalBox();
        if (i > 0){
            JLabel l = new JLabel("<html><font size=+2>" + getSign(dct[u]
[v]) + "</font></html>",JLabel.CENTER);
            l.setVerticalAlignment(SwingConstants.CENTER);
            alus1.add(l);

            // teise paneeli noolekesed:
            l = new JLabel("<html><font
size=+1>&gt;</font></html>",JLabel.CENTER);
            l.setVerticalAlignment(SwingConstants.CENTER);
            alus3.add(l);
        }
        JLabel l = new JLabel("<html>" + Math.abs(dct[u][v]),JLabel.CENTER);
        l.setAlignmentX(Component.CENTER_ALIGNMENT);
        l.setAlignmentY(Component.CENTER_ALIGNMENT);
        alus2.add(l);
        //im = createImage(u,v);
        ImagePanel ip = new ImagePanel(im[u][v]);
        ip.setAlignmentX(Component.CENTER_ALIGNMENT);

```

```

        alus2.add(ip);

        //alus2.setBorder(BorderFactory.createLoweredBevelBorder());
        alus2.add(Box.createVerticalStrut(10));
        alus1.add(alus2);

        // teisele paneelile pildikesed
        alus3.add(new ImagePanel(createImage(u,v)));
    }
    if (count > elemente){
        alus1.add(new JLabel("<html><font size=+2>+ ...
</font></html>",JLabel.CENTER));
        alus3.add(new JLabel("<html><font
size=+1>...</font></html>",JLabel.CENTER));
    }

    alus3.setBorder(BorderFactory.createLoweredBevelBorder());

    JPanel alus4 = new JPanel(new GridBagLayout());
    GridBagConstraints c = new GridBagConstraints();

    c.gridx = 0;
    c.gridy = 0;
    alus4.add(alus3,c);

    ImagePanel fullImage = new ImagePanel(createFullImage());
    c.gridx = 1;
    c.anchor = GridBagConstraints.CENTER;
    alus4.add(Box.createHorizontalStrut(10),c);
    c.gridx = 2;
    c.gridheight = 2;
    alus4.add(new JLabel("<html><font
size=+1>=</font></html>",JLabel.CENTER));
    c.gridx = 3;
    alus4.add(Box.createHorizontalStrut(10),c);
    c.gridx = 4;
    alus4.add(fullImage,c);

    alus1.setBorder(BorderFactory.createTitledBorder("Nullist erinevad
baasifunktsioonid ja nende kaalud"));
    alus4.setBorder(BorderFactory.createTitledBorder("Iga koefitsiendi
lisandumise vahetulemus ja lõplik pilt"));

    vvDialog.getContentPane().add(alus1, BorderLayout.CENTER);
    vvDialog.getContentPane().add(alus4, BorderLayout.SOUTH);

    vvDialog.pack();
    vvDialog.setVisible(true);
}

public BufferedImage loadImage(String pildiFail){
    BufferedImage tempIm = new BufferedImage
(100,100,BufferedImage.TYPE_INT_ARGB);
    try {
        tempIm = ImageIO.read(new File(pildiFail));
    }
    catch(IOException ioe){
        System.err.println(ioe.getMessage());
        JOptionPane.showMessageDialog(panel,
            "Programmi tööks vajalikku pildifaili (" + pildiFail + ") ei
leitud.\n" +
            "Vajuta OK, et programm sulgeda.",
            "VIGA: Faili ei leitud!",
            JOptionPane.ERROR_MESSAGE);
    }
}

```

```

        System.exit(1);
    }
    return tempIm;
}
}

```

9.4 Pakett dvbsim.kvantimine

9.4.1 Kvantimine.java

```

/*
 * Created on 14.04.2005
 *
 */
package dvbsim.kvantimine;

import java.awt.Cursor;
import java.awt.GridBagConstraints;
import java.awt.GridBagLayout;
import java.awt.GridLayout;
import java.awt.Insets;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Hashtable;

import javax.imageio.ImageIO;
import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JSlider;

import dvbsim.DCT;
import dvbsim.TablePanel;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class Kvantimine extends DCT {

    String imageDir = "images";
    String[] pildid = new String[]{"ruudud.png", "triibud-horiz.png", "triibud-vert.png", "triibud-varv.png", "loss.png"};
    int pildiIndex = 0;
    BufferedImage originaal_pilt;
    ClickableImagePanel originaal_paneel;
    BufferedImage tulemus_pilt;
    ImagePanel tulemus_paneel;
    JPanel panel;
    JComboBox pildiValik;
    TablePanel kvandiTabel;
    JComboBox kvandiValik;
    JSlider kvaliteet;
    JSlider dcKvaliteet;
    JPanel bittide_info_panel;

```

```

ZigZagPanel qzz_panel;
ZigZagPanel orig_zz_panel;
JButton arvuta;

boolean mpegQuant = false;
JCheckBox mpegQValik;

final String heledusText = "Heledus";
final String varvusText = "Värvus";

Cursor wait_cursor = new Cursor(Cursor.WAIT_CURSOR);
Cursor default_cursor = new Cursor(Cursor.DEFAULT_CURSOR);

/* maatriks, mis hoiab endas ühe bloki kolme värvikomponendi DCT lugemeid
 * Ehitus:
 * Object[x][y] -> Object[] = int[][] Y
 *                                     = int[][] Cr
 *                                     = int[][] Cb
 */
Object[][] dct_tabel;

// kvanditud dct lugemite tabel. Sama põhimõte, mis dct_tabeli puhul
Object[][] qdct_tabel;
Object[][] deqdct_tabel;

// kiiruse huvides meetoditest välja toodud
private int[][] qb_out;
private int[][] deqb_out;

private int[][][] blokk;
private int[] ycbcr;

final int[][] def_q_matrix = new int[][]{
    {8,16,19,22,26,27,29,34},
    {16,16,22,24,27,29,34,37},
    {19,22,26,27,29,34,34,38},
    {22,22,26,27,29,34,37,40},
    {22,26,27,29,32,34,40,48},
    {26,27,29,32,35,40,48,58},
    {26,27,29,34,38,46,56,69},
    {27,29,35,38,46,56,69,83}
};

final int[][] halb_q_matrix = new int[][]{
    {83,69,56,46,38,35,29,27},
    {69,56,46,38,34,29,27,26},
    {58,48,40,35,32,29,27,26},
    {48,40,34,32,29,27,26,22},
    {40,37,34,29,27,26,22,22},
    {38,34,34,29,27,26,22,19},
    {37,34,29,27,24,22,16,16},
    {34,29,27,26,22,19,16,8}
};

final int[][] no_q_matrix = new int[][]{
    {1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1},
    {1,1,1,1,1,1,1,1}
};

```

```

};

final int[][] aste_q_matrix = new int[][]{
    {1000,1000,1000,1000,1000,1000,1000,1000},
    {1000,1000,1000,1000,1000,1000,1000,1000},
    {1000,1000,1000,1000,1000,1000,1000,1000},
    {1000,1000,1000,1000,1000,1000,1000,1000},
    {1000,1000,1000,1000,1000,1000,1000,1000},
    {1000,1000,1000,1000,1000,1000,1000,1000},
    {1000,1000,1000,1000,1000,1000,1000,1000},
    {1000,1000,1000,1000,1000,1000,1000,1000}
};

int muudetavIndex = 4;
int[][][] q_matrix = new int[][][] {no_q_matrix, def_q_matrix, halb_q_matrix,
aste_q_matrix, no_q_matrix}; // algselt on ka omatehtud tabel standardne
String[] kvandiTabelid = new String[]{"Ilma kvantimiseta", "Standardne
tabel", "Vastupidine tabel", "Suureastmeline tabel", "Muudetav tabel"};
// indeks, mis määrab kvantimise maatriksi
//int q_index = 0;
// indeks, mis määrab kromaatsuse kvantimistabeli
//int kroma_q_index = 0;

// määrab, millise komponendi kvantimisomadusi muudetakse
int selected_komponent = 0;
// määrab iga komponendi kvantimistabeli indexi
int[] q_index = {1,1};

// skaleerimine: 2..63
int[] quant_scale = {2,2};
int scale_min = 1;
int scale_max = 31;
// DCT aläliskomponendi täpsus
int dc_min = 0;
int dc_max = 3;
//int dc_precision = (int) Math.pow(2,dc_min);
int[] dc_precision = {(int) Math.pow(2,dc_min), (int) Math.pow(2,dc_min)};

/**
 * konstruktor
 */
public Kvantimine(){
    init();
}

/**
 * initsialiseerib isendi
 */
private void init(){
    loadImage();
    tulemus_pilt = new BufferedImage(this.originaal_pilt.getWidth(),
this.originaal_pilt.getHeight(),BufferedImage.TYPE_INT_RGB);

    quantize();
    dequantize();
    generateIDCT();
}

/**
 * Mingi muutuse ilmnemisel kutsutakse see meetod välja
 */
public void updateUI(){

```

```

// seame kursori kellaks:
panel.setCursor(wait_cursor);
tulemus_pilt = new BufferedImage(this.originaal_pilt.getWidth(),
    this.originaal_pilt.getHeight(),BufferedImage.TYPE_INT_RGB);
//generateDCT();
quantize();
dequantize();
generateIDCT();
tulemus_paneel.setImage(tulemus_pilt);
generateBitInfo();
generateZZPanel();
panel.setCursor(default_cursor);
}

/**
 * Tagastab massiivi, mille elementideks on sisendist arvutatud Y, Cb ja Cr
väärtused.
 * @param rgb piksli RGB kood 32-bitises täisarvesituses.
 * @return
 */
public int[] encodeYCbCr(int rgb){
    // FCC parameetritele vastav kodeerimine
    double r = ((double)((rgb >> 16) & 0xff) / 255.0);
    double g = ((double)((rgb >> 8) & 0xff) / 255.0);
    double b = ((double)(rgb & 0xff) / 255.0);
    int Y = (int)(219 * (0.3*r + 0.59*g + 0.11*b) + 16);
    int Cb = (int)(224 * (-0.169*r - 0.331*g + 0.5*b) + 128);
    int Cr = (int)(224 * (0.5*r - 0.421*g - 0.079*b) + 128);

    /*
    int r = ((rgb >> 16) & 0xff);
    int g = ((rgb >> 8) & 0xff);
    int b = (rgb & 0xff);

    int Y = (int)(0.257*r + 0.504*g + 0.098*b + 16);
    int Cb = (int)(-0.148*r - 0.291*g + 0.439*b + 128);
    int Cr = (int)(0.439*r - 0.368*g - 0.071*b + 128);
    */
    return new int[]{Y,Cb,Cr};
}

public int[] getRGBFromInt(int rgb){
    int r = ((rgb >> 16) & 0xff);
    int g = ((rgb >> 8) & 0xff);
    int b = (rgb & 0xff);
    return new int[]{r,g,b};
}

public int decodeYCbCr(int[] ycbcr){
    int Y = ycbcr[0] - 16;
    int Cb = ycbcr[1] - 128;
    int Cr = ycbcr[2] - 128;
    int r = (int) (0.999* Y + 1.4 * Cr)*255/219;
    int g = (int) (1.0*Y - 0.333 * Cb - 0.712 * Cr)*255/224;
    int b = (int) (0.999 * Y + 1.78 * Cb)*255/224;

    /*
    int Y = ycbcr[0];
    int Cb = ycbcr[1];
    int Cr = ycbcr[2];
    int r = (int)(1.164*(Y-16) + 1.596*(Cr - 128));
    int g = (int)(1.164*(Y-16) - 0.813*(Cr - 128) - 0.392*(Cb-128));
    int b = (int)(1.164*(Y-16) + 2.017*(Cb - 128));
    */
}

```

```

        */
        return getRGBInt(fitBounds(r, 0, 255),fitBounds(g, 0, 255),fitBounds(b,
0, 255));
    }

    /**
     * Kontrollib, et number oleks vahemikus 0..255, kui pole, väljastab lähema
    piiri.
     * @param x kontrollitav arv
     * @param max maksimaalväärtus
     * @param min minimaalväärtus
     * @return number vahemikus 0..255
     */
    private final int fitBounds(int x, int min, int max){
        if (x < min) return min;
        else if (x > max) return max;
        else return x;
    }

    /**
     * Tagastab 32-bitise ARGB täisarvu, kus Alpha on 255.
     * @param red
     * @param green
     * @param blue
     * @return 32-bitine ARGB täisarv
     */
    public int getRGBInt(int red, int green, int blue){
        return getRGBInt(red,green,blue,0);
    }

    /**
     * Tagastab 32-bitise ARGB täisarvu
     *
     * @param red
     * @param green
     * @param bblue
     * @param alpha
     * @return
     */
    public int getRGBInt(int red, int green, int blue, int alpha){
        return (alpha << 24) | (red << 16) | (green << 8) | blue;
    }

    /**
     * laeb pildi
     */
    public void loadImage(){
        String pildiFail = imageDir + File.separator + pildid[pildiIndex];
        try {
            BufferedImage tempIm = ImageIO.read(new File(pildiFail));
            if (((tempIm.getWidth() % 8) != 0) || ((tempIm.getHeight() % 8) !=
0)){
                throw new IllegalArgumentException("Pildi (" + pildiFail + ")
mõõtmed ei jagu 8-ga: " +
                    tempIm.getWidth() + "x" +
                    tempIm.getHeight());
            }
            this.originaal_pilt = tempIm;
        }
        catch(IOException ioe){
            System.err.println(ioe.getMessage());
            JOptionPane.showMessageDialog(panel,
                "Programmi tööks vajalikku pildifaili (" + pildiFail + ") ei
leitud.\n" +

```

```

        "Vajuta OK, et programm sulgeda.",
        "VIGA: Faili ei leitud!",
        JOptionPane.ERROR_MESSAGE);
    System.exit(1);
}
catch(IllegalArgumentException iae){
    System.err.println(iae.getMessage());
    boolean valjuda = false;
    if (this.originaal_pilt == null)
        valjuda = true;
    JOptionPane.showMessageDialog(panel,
        "Programmi tööks vajalikku pildifaili (" + pildiFail + ")
mõõtmed ei jagu 8-ga.\n" +
        ((valjuda) ? "Vajuta OK, et programm sulgeda." : "Jätkamiseks
vajuta OK"),
        "VIGA: Pildi mõõtmed ebasobivad!",
        JOptionPane.ERROR_MESSAGE);

    if (valjuda)
        System.exit(1);
}

generateDCT();

if (this.originaal_paneel != null){
    this.originaal_paneel.setImage(this.originaal_pilt);
    updateUI();
}
}

/**
 * Loendab kokku bitid ühes blokis.
 * Bittide arvuks loetakse bittide arvu, mis on minimaalselt tarvis,
 * et edastada see blokk terviklikult
 */
private int countBits(int[][] bl){
    int bits=0;
    for (int i = 0; i < bl.length; i++) {
        for (int j = 0; j < bl[i].length; j++) {
            bits += countBits(bl[i][j]);
        }
    }
    return bits;
}

/**
 * Loendab kokku bitid täisarvus
 * @param x arv, mille bitte loendatakse
 * @return bittide arv, mis vaja x-i edastamiseks
 */
private int countBits(int x){
    int count = 0;
    while (x != 0){
        x = x >>> 1;
        count++;
    }
    return count;
}

/**
 * Tekitab graafilise objekti pildi edastuseks kuluva andmemahu infoga.
 */
public void generateBitInfo(){
    int kaadreid_sek = 25;

```

```

        if (bittide_info_panel == null){
            bittide_info_panel = new JPanel();
            bittide_info_panel.setLayout(new GridLayout(1,0));
        }
        bittide_info_panel.removeAll();
        bittide_info_panel.setToolTipText("Saatmiskiiruse puhul kaadrisagedus: "
+ kaadreid_sek + " kaadrit/s");

        int orig_bitte_pildis = 0;
        int quant_bitte_pildis = 0;

        for (int i = 0; i < qdct_tabel.length; i++) {
            for (int j = 0; j < qdct_tabel[i].length; j++) {
                int[][][] orig_blokk = getBlock(originaal_pilt,i*8,j*8);
                Object[] quant_blokk = (Object[])qdct_tabel[i][j];
                for (int k = 0; k < 3; k++) {
                    orig_bitte_pildis += countBits(orig_blokk[k]);
                    quant_bitte_pildis += countBits((int[][])quant_blokk[k]);
                }
            }
        }

        JLabel tekst = new JLabel("<html><table align='right'><tr><td>" +
            "<tr><td>&nbsp;</td><td>Originaalkujutis</td><td>&nbsp;</td><td>"
Kvanditud DCT</td></tr>" +
            "<tr><td>Kujutise maht</td><td align='right'>" + bitsToString
(orig_bitte_pildis,"b") + "</td><td>&nbsp;</td><td align='right'>" +
            bitsToString(quant_bitte_pildis,"b") + "</td></tr>" +
            "<tr><td>Saatmiskiirus</td><td align='right'>" + bitsToString
(orig_bitte_pildis * kaadreid_sek,"b/s") + "</td><td>&nbsp;</td><td
align='right'>" +
            bitsToString(quant_bitte_pildis * kaadreid_sek,"b/s") +
"</td></tr></html>");

        bittide_info_panel.add(tekst);
    }

    private String bitsToString(float bits, String yhik){
        if (bits < 1024)
            return bits + " " +yhik;
        String[] liited = new String[]{"","k", "M","G"};
        int i;
        for (i = 1; i < liited.length; i++) {
            bits /= 1024;
            if (bits < 1024)
                break;
        }
        if (i >= liited.length)
            i = liited.length - 1;
        return "" + ((float)((int)(bits*10))/10) + " " + liited[i] + yhik;
    }

    /**
     * koostab mooduli pildi ning tagastab selle
     * @return paneel sisuga
     */
    public JPanel getDisplay(){
        panel = new JPanel();
        panel.setLayout(new GridBagLayout());
        GridBagConstraints c = new GridBagConstraints();
        c.insets = new Insets(5,5,5,5);

```

```

        originaal_paneel = new ClickableImagePanel(this.originaal_pilt);
        originaal_paneel.addMouseListener(new ClickableImageMouseListener
(this));
        originaal_paneel.setToolTipText("Kliki pildil, et muuta näidisbloki
asukohta");
        JLabel originaal_silt = new JLabel("Algne kujutis (" +
originaal_pilt.getWidth() + "x" + originaal_pilt.getHeight() + " pikselit:");

        tulemus_paneel = new ImagePanel(tulemus_pilt);
        JLabel tulemus_silt = new JLabel("Taastatud kujutis:");

        JPanel infopaneel = new JPanel();
        infopaneel.setLayout(new GridLayout(0,2));

        pildiValik = new JComboBox(pildid);
        pildiValik.setBorder(BorderFactory.createTitledBorder("Vali algpilt:"));
        pildiValik.setSelectedIndex(pildiIndex);
        pildiValik.addActionListener(new PildiValikListener(this));
        infopaneel.add(pildiValik);

        JRadioButton heledus_komponent = new JRadioButton(heledusText);
        heledus_komponent.setActionCommand(heledus_komponent.getText());
        heledus_komponent.addActionListener(new KomponentValikListener(this));
        heledus_komponent.setSelected(true);

        JRadioButton varvus_komponent = new JRadioButton(varvusText);
        varvus_komponent.setActionCommand(varvus_komponent.getText());
        varvus_komponent.addActionListener(new KomponentValikListener(this));

        ButtonGroup komponent_valik = new ButtonGroup();
        komponent_valik.add(heledus_komponent);
        komponent_valik.add(varvus_komponent);

        JPanel komponent_valik_panel = new JPanel();
        komponent_valik_panel.setLayout(new GridLayout(1,1));
        komponent_valik_panel.add(heledus_komponent);
        komponent_valik_panel.add(varvus_komponent);
        komponent_valik_panel.setBorder(BorderFactory.createTitledBorder
("Komponent seadistamiseks"));
        komponent_valik_panel.setToolTipText("Vali komponent, mille
kvantimistabeli seadeid soovid muuta");

        infopaneel.add(komponent_valik_panel);

        kvandiValik = new JComboBox(kvandiTabelid);
        kvandiValik.setBorder(BorderFactory.createTitledBorder("Kvantimistabeli
valik:"));
        kvandiValik.setSelectedIndex(q_index[selected_komponent]);
        kvandiValik.addActionListener(new KvandiValikListener(this));
        infopaneel.add(kvandiValik);

        mpegQValik = new JCheckBox("MPEG-2 kvantimine");
        mpegQValik.setToolTipText("<html>Kasutada kvantimisel MPEG-2 standardile
vastavaid valemeid.<br>" +
                "Sel juhul kvanditakse alaliskomponent eraldi ja arvestatakse
kvaliteeti määravaid liugureid.");
        mpegQValik.setActionCommand("mpeg-q-valik");
        mpegQValik.setSelected(mpegQuant);
        mpegQValik.addItemListener(new CheckBoxAdapter(this));
        infopaneel.add(mpegQValik);

```

```

        dcKvaliteet = new JSlider(dc_min, dc_max, dc_min);
        dcKvaliteet.setToolTipText("<html>Kui kvantimine on MPEG-2,<br>siis see
määrab alaliskomponendi edastustäpsuse.");
        dcKvaliteet.setBorder(BorderFactory.createTitledBorder("MPEG-2 DC
täpsus:"));
        dcKvaliteet.addChangeListener(new DCKvaliteetListener(this));
        Hashtable dcKvaliteetSildid = new Hashtable();
        dcKvaliteetSildid.put(new Integer(dc_min),new JLabel("Hea"));
        dcKvaliteetSildid.put(new Integer(dc_max),new JLabel("Halb"));
        dcKvaliteet.setLabelTable(dcKvaliteetSildid);
        dcKvaliteet.setPaintLabels(true);
        dcKvaliteet.setMajorTickSpacing(1);
        dcKvaliteet.setPaintTicks(true);
        dcKvaliteet.setSnapToTicks(true);
        dcKvaliteet.setEnabled(mpegQuant);
        infopaneel.add(dcKvaliteet);

        kvaliteet = new JSlider(scale_min, scale_max, (int) (quant_scale
[selected_komponent]/2));
        kvaliteet.addChangeListener(new KvaliteetListener(this));
        kvaliteet.setBorder(BorderFactory.createTitledBorder("MPEG-2 AC
kvaliteet:"));
        kvaliteet.setToolTipText("<html>Kui kvantimise meetodiks on MPEG-
2,<br>siis see määrab mitte-alaliskomponentide kvaliteedi");
        Hashtable kvaliteetSildid = new Hashtable();
        kvaliteetSildid.put(new Integer(scale_min),new JLabel("Hea"));
        kvaliteetSildid.put(new Integer(scale_max),new JLabel("Halb"));
        kvaliteet.setLabelTable(kvaliteetSildid);
        kvaliteet.setPaintLabels(true);
        kvaliteet.setEnabled(mpegQuant);
        infopaneel.add(kvaliteet);

        kvandiTabel = new TablePanel(q_matrix[q_index[selected_komponent]]);
        kvandiTabel.setBorder(BorderFactory.createTitledBorder
("Kvantimistabel:"));

        generateBitInfo();
        bittide_info_panel.setBorder(BorderFactory.createTitledBorder
("Edastamiskiiruse info:"));

        // Sikk-Sakk esitus:
        generateZZPanel();
        JLabel orig_zz_silt = new JLabel("Heleduse kvantimata DCT ZZ-esitus:");
        JLabel qzz_silt = new JLabel("Heleduse kvanditud DCT ZZ-esitus:");

        arvuta = new JButton("Arvuta");
        arvuta.setActionCommand("arvuta");
        arvuta.setEnabled(false);
        arvuta.addActionListener(new ButtonAdapter(this));

        // Lisame kõik paneelile
        c.anchor = GridBagConstraints.FIRST_LINE_START;

        c.gridx = 0;
        c.gridy = 0;
        c.gridwidth = 1;
        c.gridheight = 1;
        panel.add(originaal_silt,c);
        c.gridx = 0;
        c.gridy = 1;
        c.gridwidth = 1;
        c.gridheight = 3;

```

```

panel.add(originaal_paneel,c);

c.gridx = 1;
c.gridy = 0;
c.gridheight = 1;
panel.add(tulemus_silt,c);
c.gridx = 1;
c.gridy = 1;
c.gridwidth = 1;
c.gridheight = 3;
panel.add(tulemus_paneel,c);

c.gridx = 0;
c.gridy = 4;
c.gridwidth = 1;
c.gridheight = 1;
panel.add(orig_zz_silt,c);
c.gridx = 0;
c.gridy = 5;
c.gridwidth = 1;
c.gridheight = 3;
panel.add(orig_zz_panel,c);

c.gridx = 1;
c.gridy = 4;
c.gridwidth = 1;
c.gridheight = 1;
panel.add(qzz_silt,c);
c.gridx = 1;
c.gridy = 5;
c.gridwidth = 1;
c.gridheight = 3;
panel.add(qzz_panel,c);

c.fill = GridBagConstraints.HORIZONTAL;

c.gridx = 2;
c.gridy = 1;
c.gridwidth = 4;
c.gridheight = 3;
panel.add(Infopaneel,c);

c.gridx = 2;
c.gridy = 4;
c.gridwidth = 3;
c.gridheight = 2;
panel.add(kvandiTabel,c);

c.gridx = 5;
c.gridy = 5;
c.gridwidth = 1;
c.gridheight = 1;
panel.add(arvuta,c);

c.gridx = 2;
c.gridy = 6;
c.gridwidth = 4;
c.gridheight = 2;
panel.add(bittide_info_panel,c);

return panel;
}

```

```

public void generateZZPanel(){

```

```

        int[][] blokk = getDCT(getBlock
(originaal_pilt,originaal_paneel.getRect_x(),originaal_paneel.getRect_y())[0]);
        if (orig_zz_panel == null)
            orig_zz_panel = new ZigZagPanel(this,blokk,2048);
        else
            orig_zz_panel.setMatrix(blokk);

        int[][] blokk2 = (int[][])((Object[])qdct_tabel
[originaal_paneel.getRect_x()/8][originaal_paneel.getRect_y()/8])[0];
        if (qzz_panel == null)
            qzz_panel = new ZigZagPanel(this,blokk2,2048);
        else
            qzz_panel.setMatrix(blokk2);
    }

    public void dump(int[][] a){
        for (int i = 0; i < a.length; i++) {
            for (int j = 0; j < a[i].length; j++) {
                System.out.print(a[i][j] + ", ");
            }
            System.out.println();
        }
    }

    /**
     * Loeb sisendpildist x ja y kohapealt 8x8 bloki välja ning tagastab selle
     YCbCr lugemid.
     * @param im
     * @param x
     * @param y
     * @return 3x8x8 maatriks YCbCr väärtustega
     */
    private final int[][][] getBlock(BufferedImage im, int im_x, int im_y){
        blokk = new int[3][8][8];
        for (int i = 0; i < 8; i++){
            for (int j = 0; j < 8; j++){
                ybcr = encodeYCbCr(im.getRGB(im_x+i,im_y+j));
                //int[] ybcr = getRGBFromInt(im.getRGB(im_x+i,im_y+j));
                blokk[0][i][j] = ybcr[0];
                blokk[1][i][j] = ybcr[1];
                blokk[2][i][j] = ybcr[2];
            }
        }
        return blokk;
    }

    /**
     * Seab tulemuse pildis pikslid vastavalt etteantud parameetritele
     * @param im pilt, mille piksleid muudetakse
     * @param im_x pildi bloki alguskoordinaat
     * @param im_y pildi bloki alguskoordinaat
     * @param blokk pikslite blokk (Y,Cr,Cb)
     */
    private void setBlock(BufferedImage im, int im_x, int im_y, int[][][] blokk)
    {
        for (int i = 0; i < 8; i++){
            for (int j = 0; j < 8; j++){
                im.setRGB(im_x + i, im_y + j,
                decodeYCbCr(new int[]{blokk[0][i][j], blokk[1][i][j],
                blokk[2][i][j]}));
                /*im.setRGB(im_x + i, im_y + j,
                getRGBInt(blokk[0][i][j], blokk[1][i][j], blokk[2][i]
                [j]));*/
            }
        }
    }
}

```

```

/**
 * Arvutab pildifailist DCT tulemuse ning salvestab DCT pildi hoidjasse
 */
private void generateDCT(){
    // hakkab hoidma dct-töödeldud pildi blokke
    dct_tabel = new Object[(int)(Math.ceil(originaal_pilt.getHeight()/8))]
[(int)(Math.ceil(originaal_pilt.getHeight()/8))];
    int[][][] blokk;
    for (int i = 0; i < (int)originaal_pilt.getHeight(); i += 8){
        for (int j = 0; j < (int)originaal_pilt.getWidth(); j += 8){
            blokk = getBlock(originaal_pilt,i,j);
            dct_tabel[(int)(i/8)][(int)(j/8)] = new Object[]{getDCT((int[]
[])blokk[0]),
                getDCT((int[][])blokk[1]),
                getDCT((int[][])blokk[2])};
        }
    }
}

/**
 * Arvutab DCT infost taas pildi.
 */
private void generateIDCT(){
    int[][] Y;
    int[][] Cr;
    int[][] Cb;
    Object[] temp;
    for (int i = 0; i < dct_tabel.length; i++){
        for (int j = 0; j < dct_tabel[i].length; j++){
            temp =(Object[])deqdcTabel[i][j];
            Y = getIDCT((int[][])temp[0]);
            Cr = getIDCT((int[][])temp[1]);
            Cb = getIDCT((int[][])temp[2]);
            setBlock(tulemus_pilt, (int)(i*8), (int)(j*8), new int[][][]
{Y,Cr,Cb});
        }
    }
}

/**
 * kvandib DCT lugemid
 */
private void quantize(){
    qdct_tabel = new Object[(int)(Math.ceil(originaal_pilt.getHeight()/8))]
[(int)(Math.ceil(originaal_pilt.getHeight()/8))];
    int[][] Y;
    int[][] Cr;
    int[][] Cb;
    Object[] temp;
    for (int i = 0; i < dct_tabel.length; i++){
        for (int j = 0; j < dct_tabel[i].length; j++){
            temp =(Object[])dct_tabel[i][j];
            if (mpegQuant){
                Y = quantizeBlockMPEG((int[][])temp[0],0);
                Cr = quantizeBlockMPEG((int[][])temp[1],1);
                Cb = quantizeBlockMPEG((int[][])temp[2],1);
            }
            else {
                Y = quantizeBlock((int[][])temp[0],0);
                Cr = quantizeBlock((int[][])temp[1],1);
                Cb = quantizeBlock((int[][])temp[2],1);
            }
            qdct_tabel[i][j] = new Object[]{Y,Cr,Cb};
        }
    }
}

```

```

    }
}

/**
 * taastab kvanditud lugemitest DCT lugemid
 */
private void dequantize(){
    deq_dct_tabel = new Object[(int)(Math.ceil(originaal_pilt.getHeight()/
8))][(int)(Math.ceil(originaal_pilt.getHeight()/8))];
    int[][] Y;
    int[][] Cr;
    int[][] Cb;
    Object[] temp;
    for (int i = 0; i < qdct_tabel.length; i++){
        for (int j = 0; j < qdct_tabel[i].length; j++){
            temp = (Object[])qdct_tabel[i][j];
            if (mpegQuant){
                Y = deQuantizeBlockMPEG((int[])temp[0],0);
                Cr = deQuantizeBlockMPEG((int[])temp[1],1);
                Cb = deQuantizeBlockMPEG((int[])temp[2],1);
            }
            else {
                Y = deQuantizeBlock((int[])temp[0],0);
                Cr = deQuantizeBlock((int[])temp[1],1);
                Cb = deQuantizeBlock((int[])temp[2],1);
            }
            deq_dct_tabel[i][j] = new Object[]{Y,Cr,Cb};
        }
    }
}

/**
 * kvandib ühe 8x8 dct lugemite bloki MPEG-2 valemitele vastavalt
 * @param bl kvanditav blokk
 * @return tagastab kvanditud väärtuste bloki
 */
private final int[][] quantizeBlockMPEG(int[][] bl,int index){
    qb_out = new int[8][8];
    int i,j;
    for (i = 0; i < bl.length; i++){
        for (j =0; j < bl[i].length; j++){
            // eemaldatud Math.round()
            //qb_out[i][j] = (int)Math.round((bl[i][j] * 32 + sign(bl[i][j]))
*q_matrix[index][i][j])/(2*q_matrix[index][i][j] * quant_scale));
            if (i == 0 && j == 0 )
                qb_out[i][j] = bl[i][j] / dc_precision[index];
            else
                qb_out[i][j] = ((32 * bl[i][j])/(q_matrix[q_index[index]][i][j]*quant_scale[index]))/2;
        }
    }
    return qb_out;
}

/**
 * kvandib ühe 8x8 dct lugemite bloki lihtsustatud valemitega
 * @param bl kvanditav blokk
 * @return tagastab kvanditud väärtuste bloki
 */
private final int[][] quantizeBlock(int[][] bl,int index){
    qb_out = new int[8][8];
    int i,j;

```

```

    for (i = 0; i < bl.length; i++){
        for (j =0; j < bl[i].length; j++){
            qb_out[i][j] = bl[i][j]/q_matrix[q_index[index]][i][j];
        }
    }
    return qb_out;
}

private final int[][] deQuantizeBlockMPEG(int[][] bl,int index){
    deqb_out = new int[8][8];
    int i,j;
    int summa=0;
    for (i = 0; i < bl.length; i++){
        for (j =0; j < bl[i].length; j++){
            // eemaldataud Math.round()
            //deqb_out[i][j] = (int)((2 * bl[i][j]*q_matrix[index][i][j] *
quant_scale - sign(bl[i][j])*q_matrix[index][i][j])/32);
            //deqb_out[i][j] = (int)Math.round(bl[i][j]*q_matrix[index][i
[j]));
            /*
            deqb_out[i][j] = fitBounds((int)((2 * bl[i][j] * q_matrix
[index][i][j])/32),-2048,2047);
            summa += deqb_out[i][j];
            */
            if (i == 0 && j == 0)
                deqb_out[i][j] = fitBounds(bl[i][j] * dc_precision[index],-
2048,2047);
            else
                deqb_out[i][j] = fitBounds((int)((2 * bl[i][j] * (q_matrix
[q_index[index]][i][j]*quant_scale[index]))/32),-2048,2047);

        }
    }
    // LSB toggling (kui summa on paaritu, siis ei muutu midagi)
    //if (summa % 2 == 0)
        // deqb_out[7][7] = (deqb_out[7][7] % 2 == 1)? deqb_out[7][7] -1 :
deqb_out[7][7] +1;

    return deqb_out;
}

private final int[][] deQuantizeBlock(int[][] bl,int index){
    deqb_out = new int[8][8];
    int i,j;
    int summa=0;
    for (i = 0; i < bl.length; i++){
        for (j =0; j < bl[i].length; j++){
            deqb_out[i][j] = bl[i][j]*q_matrix[q_index[index]][i][j];
        }
    }
    return deqb_out;
}

/**
 * Kiire meetod arvu märgi saamiseks.
 * Eesmärk pakkuda kiiremat tulemust, kui Math.signum().
 * @param in arv, mille märki kontrollitakse
 * @return -1, kui in < 0, muidu 1
 */
private final int sign(int in){
    return (in < 0)? -1: 1;
}

public void checkKvantEditable(){
    if (q_index[selected_komponent] != muudetavIndex){

```

```

        kvandiTabel.setTableEnabled(false);
        arvuta.setEnabled(false);
    }
    else{
        kvandiTabel.setTableEnabled(true);
        arvuta.setEnabled(true);
    }
}
}

```

9.4.2 ImagePanel.java

```

/*
 * Created on 14.04.2005
 *
 */
package dvbsim.kvantimine;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;

import javax.swing.JPanel;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class ImagePanel extends JPanel {

    BufferedImage image;

    /**
     * konstruktor
     * @param im - pilt, mida näidata
     */
    public ImagePanel(BufferedImage im){
        this.setImage(im);
    }

    public void setImage(BufferedImage im){
        this.image = im;
        Dimension size = new Dimension(this.image.getHeight(),
this.image.getWidth());
        this.setPreferredSize(size);
        this.setMinimumSize(size);
        updateUI();
    }

    /**
     * tegeleb joonistamisega
     * @see javax.swing.JComponent#paintComponent(java.awt.Graphics)
     */
    public void paintComponent(Graphics g){
        Graphics2D g2d = (Graphics2D) g;
        g2d.drawImage(image,0,0,this);
    }

}

```

9.4.3 ClickableImagePanel.java

```

/*

```

```

* Created on 23.04.2005
*
*/
package dvbsim.kvantimine;

import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.image.BufferedImage;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class ClickableImagePanel extends ImagePanel {

    /**
     * 8x8 ristkülik, mis näitab parasjagu aktiivset blokki
     */
    private final Rectangle rect = new Rectangle(0,0,8,8);

    private int rect_x = 40, rect_y = 40;

    /**
     * konstruktor
     * @param im - pilt, mida näidata
     */
    public ClickableImagePanel(BufferedImage im){
        super(im);
    }

    void drawRect(Graphics2D g2d){
        g2d.translate(rect_x,rect_y);
        // Kui taustavärvi on suhteliselt palju puhast, siis teeme kasti
        sinise, muidu punase
        int rgb = ((image.getRGB(rect_x,rect_y) & 0xffffffff) ^ 0xff0000)>>16 < 50
? 0xff:0xff0000;
        g2d.setColor(new Color(rgb));
        g2d.draw(rect);
    }

    /**
     * tegeleb joonistamisega
     * @see javax.swing.JComponent#paintComponent(java.awt.Graphics)
     */
    public void paintComponent(Graphics g){
        Graphics2D g2d = (Graphics2D) g;
        g2d.drawImage(image,0,0,this);
        drawRect(g2d);
    }

    /**
     * @return Tagastab rect_x väärtuse.
     */
    public int getRect_x() {
        return rect_x;
    }

    /**
     * @param rect_x The rect_x to set.
     */
    public void setRect_x(int rect_x) {
        this.rect_x = rect_x;
    }

    /**
     * @return Tagastab rect_y väärtuse.
     */

```

```

public int getRect_y() {
    return rect_y;
}
/**
 * @param rect_y Väärtus, mis rect_y-le seada.
 */
public void setRect_y(int rect_y) {
    this.rect_y = rect_y;
}
}

```

9.4.4 ZigZagPanel.java

```

/*
 * Created on 24.04.2005
 *
 */
package dvbsim.kvantimine;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Polygon;
import java.awt.Rectangle;
import java.awt.font.FontRenderContext;
import java.awt.font.GlyphVector;
import java.awt.geom.AffineTransform;
import java.awt.geom.Rectangle2D;

import javax.swing.JPanel;
import javax.swing.JSlider;
import javax.swing.SwingConstants;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class ZigZagPanel extends JPanel {

    Kvantimine vanem;
    int[][] matrix;
    int maxVal;

    ZigZagImage image;
    JSlider mootkava;

    // määravad paneeli mõõdud
    int height = 160;
    int width = 200;
    int inset = 10;

    Dimension size = new Dimension(width + inset,height + inset);

    public ZigZagPanel(Kvantimine vanem,int[][] m,int maxVal){
        this.vanem = vanem;
        this.matrix = m;
    }
}

```

```

        this.maxVal = maxVal;

        this.setMinimumSize(size);
        this.setPreferredSize(size);

        mootkava = new JSlider(SwingConstants.VERTICAL,1,2048,maxVal);
        mootkava.addChangeListener(new MootkavaListener(this));
        mootkava.setToolTipText("Muuda joonise skaalat");

        image = new ZigZagImage(this,width,height);

        this.setLayout(new BorderLayout());
        this.add(mootkava,BorderLayout.WEST);
        this.add(image,BorderLayout.CENTER);

    }

    public void setMatrix(int[][] m){
        this.matrix = m;
        this.image.setMatrix(this.matrix);
    }

    public void setMaxVal(int v){
        this.maxVal = v;
        this.image.setMaxVal(v);
    }
}

class ZigZagImage extends JPanel{

    ZigZagPanel vanem;
    int[][] matrix;

    Font font = new Font("Monospaced",Font.PLAIN,12);
    FontRenderContext frc;
    double textWidth;
    int axis_w = 5;

    int width,height;

    int maxVal;
    int samm = 2;
    int bar_w;
    int bar_h;
    Rectangle bar;
    Polygon nool;
    Dimension size;

    boolean firstTime = true;

    int[] zi =
{0,0,1,2,1,0,0,1,2,3,4,3,2,1,0,0,1,2,3,4,5,6,5,4,3,2,1,0,0,1,2,3,4,5,6,7,7,6,5,4
,3,2,1,2,3,4,5,6,7,7,6,5,4,3,4,5,6,7,7,6,4,6,7,7};
    int[] zj =
{0,1,0,0,1,2,3,2,1,0,0,1,2,3,4,5,4,3,2,1,0,0,1,2,3,4,5,6,7,6,5,4,3,2,1,0,1,2,3,4
,5,6,7,7,6,5,4,3,2,3,4,5,6,7,7,6,5,4,5,6,7,7,6,7};

    public ZigZagImage(ZigZagPanel vanem,int width, int height){
        this.vanem = vanem;
        this.height = height;
        this.width = width;
        size = new Dimension(width,height);
        this.bar_h = (height / 2);
    }
}

```

```

        this.maxVal = vanem.maxVal;
        this.matrix = vanem.matrix;
        this.setMinimumSize(size);
        this.setPreferredSize(size);
    }

    public void paintComponent(Graphics g){
        Graphics2D g2d = (Graphics2D)g;
        if (firstTime){
            frc = g2d.getFontRenderContext();
            // see on kõige laiem teksti laius (koos ruumiga)
            textWidth = ((font.createGlyphVector(frc,String.valueOf("-2048"))).
getVisualBounds()).getWidth() + 2*axis_w;
            bar_w = (width - (int)textWidth)/64;
            bar = new Rectangle(0,0,bar_w,height);
            nool = new Polygon();
            nool.addPoint(0,-axis_w);
            nool.addPoint(axis_w,0);
            nool.addPoint(0,axis_w);

            firstTime = false;
        }

        g2d.setColor(g2d.getBackground());
        g2d.fillRect(0,0,this.getWidth(),this.getHeight());
        g2d.translate(0,bar_h+ vanem.inset/2);

        g2d.translate(textWidth,0);

        AffineTransform algtrans = g2d.getTransform();

        g2d.setColor(Color.BLACK);
        g2d.setTransform(algtrans);
        GlyphVector gv1 = font.createGlyphVector(frc,String.valueOf(maxVal));
        Rectangle2D textOuter1 = gv1.getVisualBounds();
        GlyphVector gv2 = font.createGlyphVector(frc,"0");
        Rectangle2D textOuter2 = gv2.getVisualBounds();
        GlyphVector gv3 = font.createGlyphVector(frc,String.valueOf(-maxVal));
        Rectangle2D textOuter3 = gv3.getVisualBounds();
        GlyphVector gv4 = font.createGlyphVector(frc,"64");
        Rectangle2D textOuter4 = gv4.getVisualBounds();

        g2d.drawGlyphVector(gv1,-(float)textOuter1.getWidth() - 2*axis_w, -
bar_h + (float)textOuter1.getHeight());
        g2d.drawGlyphVector(gv2,-(float)textOuter2.getWidth() - 2*axis_w,
(float)textOuter2.getHeight()/2);
        g2d.drawGlyphVector(gv3,-(float)textOuter3.getWidth() - 2*axis_w, bar_h
);
        g2d.drawGlyphVector(gv4,64 * bar_w, (float)textOuter4.getHeight() +
2*axis_w);

        g2d.drawLine(0,0,-axis_w,0);
        g2d.drawLine(0,-bar_h,-axis_w,-bar_h);
        g2d.drawLine(0,bar_h,-axis_w,bar_h);
        g2d.drawLine(0,-bar_h,0,bar_h);
        g2d.drawLine(64 * bar_w, 0,64 * bar_w,axis_w);
        g2d.drawLine(0,0,64 * bar_w + 2*axis_w,0);
        g2d.translate(64*bar_w + 2*axis_w,0);
        g2d.fill(nool);
        g2d.setTransform(algtrans);

        g2d.translate((int)(bar_w),0);
        algtrans = g2d.getTransform();
    }

```

```

        for (int i = 0; i < 64; i++) {
            drawBar(matrix[zi[i]][zj[i]],g2d);
            g2d.translate(bar_w,0);
        }
    }

    private void drawBar(int value,Graphics2D g2d){
        int korgus = (int)((Math.abs(value) * bar_h)/maxVal);
        bar.setSize(bar_w,korgus);
        g2d.setColor(Color.BLUE);
        AffineTransform trans = g2d.getTransform();
        if (value > 0)
            g2d.translate(0,-1*korgus);
        g2d.fill(bar);
        g2d.setTransform(trans);
        //System.out.println(value);
    }

    public void setMatrix(int[][] m){
        this.matrix = m;
        updateUI();
    }

    public void setMaxVal(int v){
        this.maxVal = v;
        updateUI();
    }
}

class MootkavaListener implements ChangeListener{

    ZigZagPanel vanem;

    public MootkavaListener(ZigZagPanel vanem){
        this.vanem = vanem;
    }

    /* (non-Javadoc)
     * @see javax.swing.event.ChangeListener#stateChanged
     (javax.swing.event.ChangeEvent)
     */
    public void stateChanged(ChangeEvent arg0) {
        JSlider source = (JSlider)arg0.getSource();
        int value = (int)source.getValue();
        if (! source.getValueIsAdjusting()){
            vanem.setMaxVal(value);
        }
    }
}

```

9.4.5 ClickableImageMouseListener.java

```

/*
 * Created on 23.04.2005
 *
 */

```

```

package dvbsim.kvantimine;

import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class ClickableImageMouseListener extends MouseAdapter {

    Kvantimine vanem;

    /**
     * Konstruktor
     * @param vanem      Kvantimise isend, mille kaudu suheldakse teiste
objektidega.
     */
    public ClickableImageMouseListener(Kvantimine vanem) {
        this.vanem = vanem;
    }

    public void mouseClicked(MouseEvent me) {
        int x = ((int) (me.getX()/8))*8;
        int y = ((int) (me.getY()/8))*8;
        if ((vanem.originaal_paneel.getRect_x() != x ||
vanem.originaal_paneel.getRect_y() != y) && x < vanem.originaal_pilt.getWidth()
&& y < vanem.originaal_pilt.getHeight()) {
            vanem.originaal_paneel.setRect_x(x);
            vanem.originaal_paneel.setRect_y(y);
            vanem.originaal_paneel.updateUI();
            vanem.generateZZPanel();
        }
    }
}

```

9.4.6 ButtonAdapter.java

```

/**
 * Created on 13.05.2005
 *
 */
package dvbsim.kvantimine;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class ButtonAdapter implements ActionListener {

    Kvantimine vanem;

    public ButtonAdapter(Kvantimine vanem) {
        this.vanem = vanem;
    }

    /** (non-Javadoc)
     * @see java.awt.event.ActionListener#actionPerformed
(java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent ae) {

```

```

        String cmd = ae.getActionCommand();
        if (cmd.equalsIgnoreCase("arvuta") && vanem.arvuta.isEnabled()){
            vanem.q_matrix[vanem.muudetavIndex] = vanem.kvandiTabel.getMatrix();
            vanem.updateUI();
        }
    }
}

```

9.4.7 KomponentValikListener.java

```

/*
 * Created on 12.05.2005
 *
 */
package dvbsim.kvantimine;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class KomponentValikListener implements ActionListener{

    Kvantimine vanem;

    public KomponentValikListener(Kvantimine vanem){
        this.vanem = vanem;
    }

    /* (non-Javadoc)
     * @see java.awt.event.ActionListener#actionPerformed
     (java.awt.event.ActionEvent)
     */
    public void actionPerformed(ActionEvent ae) {
        String cmd = ae.getActionCommand();
        if (cmd.equals(vanem.heledusText)){
            doAction(0);
        }
        else if (cmd.equals(vanem.varvusText)){
            doAction(1);
        }
    }

    private void doAction(int komponent){
        vanem.selected_komponent = komponent;
        vanem.kvandiValik.setSelectedIndex(vanem.q_index[komponent]);
        vanem.kvaliteet.setValue((int) (vanem.quant_scale
[vanem.selected_komponent]/2));
        vanem.kvaliteet.updateUI();
        vanem.dcKvaliteet.setValue((int) (Math.log(vanem.dc_precision
[komponent])/Math.log(2)));
        vanem.kvandiTabel.setMatrix(vanem.q_matrix[vanem.q_index
[vanem.selected_komponent]]);
        vanem.checkKvantEditable();
    }
}

```

9.4.8 KvaliteetListener.java

```
/*
 * Created on 12.05.2005
 *
 */
package dvbsim.kvantimine;

import javax.swing.JSlider;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class KvaliteetListener implements ChangeListener{

    Kvantimine vanem;

    public KvaliteetListener(Kvantimine vanem){
        this.vanem = vanem;
    }

    /* (non-Javadoc)
     * @see javax.swing.event.ChangeListener#stateChanged
     (javax.swing.event.ChangeEvent)
     */
    public void stateChanged(ChangeEvent arg0) {
        JSlider source = (JSlider)arg0.getSource();
        int value = (int)(source.getValue()*2);
        if (! source.getValueIsAdjusting() && vanem.quant_scale
[vanem.selected_komponent] != value){
            vanem.quant_scale[vanem.selected_komponent] = value;
            if (vanem.mpegQuant)
                vanem.updateUI();
        }

    }
}
```

9.4.9 KvandiValikListener.java

```
/*
 * Created on 12.05.2005
 *
 */
package dvbsim.kvantimine;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JComboBox;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class KvandiValikListener implements ActionListener{
    Kvantimine vanem;

    public KvandiValikListener(Kvantimine vanem){
        this.vanem = vanem;
    }
}
```

```

    public void actionPerformed(ActionEvent ae) {
        int selected = (int) ((JComboBox) ae.getSource()).getSelectedIndex();
        if (vanem.q_index[vanem.selected_komponent] != selected) {
            vanem.q_index[vanem.selected_komponent] = selected;
            vanem.kvandiTabel.setMatrix(vanem.q_matrix[selected]);
            vanem.checkKvantEditable();
            vanem.updateUI();
        }
    }
}

```

9.4.10 PildiValikListener.java

```

/*
 * Created on 12.05.2005
 *
 */
package dvbsim.kvantimine;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.JComboBox;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class PildiValikListener implements ActionListener {
    Kvantimine vanem;

    public PildiValikListener(Kvantimine vanem) {
        this.vanem = vanem;
    }

    public void actionPerformed(ActionEvent ae) {
        int selected = (int) ((JComboBox) ae.getSource()).getSelectedIndex();
        vanem.pildiIndex = selected;
        vanem.loadImage();
    }
}

```

9.4.11 CheckBoxAdapter.java

```

/*
 * Created on 13.05.2005
 *
 */
package dvbsim.kvantimine;

import java.awt.event.ItemEvent;
import java.awt.event.ItemListener;

import javax.swing.JCheckBox;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class CheckBoxAdapter implements ItemListener {

    Kvantimine vanem;
}

```

```

public CheckBoxAdapter(Kvantimine v){
    this.vanem = v;
}

/* (non-Javadoc)
 * @see java.awt.event.ItemListener#itemStateChanged
(java.awt.event.ItemEvent)
 */
public void itemStateChanged(ItemEvent ie) {
    JCheckBox source = (JCheckBox)ie.getSource();
    String cmd = source.getActionCommand();
    if (cmd.equalsIgnoreCase("mpeg-q-valik")){
        vanem.mpegQuant = source.isSelected();
        vanem.kvaliteet.setEnabled(source.isSelected());
        vanem.dcKvaliteet.setEnabled(source.isSelected());
        vanem.updateUI();
    }
}
}

```

9.4.12 DCKvaliteetListener.java

```

/*
 * Created on 13.05.2005
 *
 */
package dvbsim.kvantimine;

import javax.swing.JSlider;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

/**
 * @author Laas Toom (laaz@laaz.org)
 */
public class DCKvaliteetListener implements ChangeListener {

    Kvantimine vanem;

    public DCKvaliteetListener(Kvantimine v){
        this.vanem = v;
    }

    /* (non-Javadoc)
     * @see javax.swing.event.ChangeListener#stateChanged
(javax.swing.event.ChangeEvent)
     */
    public void stateChanged(ChangeEvent arg0) {
        JSlider source = (JSlider)arg0.getSource();
        int value = (int)Math.pow(2, source.getValue());
        if (! source.getValueIsAdjusting() && vanem.dc_precision
[vanem.selected_komponent] != value){
            vanem.dc_precision[vanem.selected_komponent] = value;
            if (vanem.mpegQuant)
                vanem.updateUI();
        }
    }
}

```

10 LISA 2: CD VALMINUD TARKVARAGA

CD sisaldab:

1. magistritöö (pdf vormingus)
2. valminud tarkvara (koheselt käivitatava JAR-failina)
3. programmi koodi
4. Java käivituskeskkonna versiooni 1.5.0_02 Windows ja Linux operatsioonisüsteemide jaoks