H.264-Bringer:

Dokumentacija

# Predgovor

U ovoj dokumentaciji opisani su postupci programiranja kodeka H.264 isključivo s tehničkog stajališta. Prethodno čitanju ove dokumentacije, kao teorijsku pozadinu preporuča se proučiti (ne nužno pročitati) diplomske radove koji opisuju ovaj kodek te u njima spomenutu literaturu.

# Važne globalne varijable

typedef struct {

int Lwidth,Lheight;

int Cwidth,Cheight;

unsigned char \*L, \*C[2];

} frame\_type;

**frame\_type**: Sadrži najvažnije podatke o slici. Članovi strukture su:

* **\*L** – niz luminantnih uzoraka slike (Y). Kardinalnost niza jednaka je broju piksela u slici, odnosno W\*H, gdje su W i H redom širina i visina slike u pikselima.
* **\*C[2]**–dva niza krominantnih uzoraka slike (Cb i Cr). C[0] odgovara komponenti Cb, a C[1] komponenti Cr. Kardinalnost svakog niza odgovara kardinalnosti krominantnih uzoraka u slici prema poduzorkovanju 4:2:0, odnosno (W/2)\*(H/2) , gdje su W i H redom širina i visina slike u pikselima.
* **Lwidth, Lheight, Cwidht, Cheight** – redom broj luminantnih uzoraka po širini i visini slike te broj krominantnih uzoraka po širini i visini slike. Vrijede jednakosti Cwidth = Lwidth / 2 i Cheight = Lheight / 2. Lwidth i Lheight su uvijek višekratnici broja 16, a Cwidth i Cheight su uvijek višekratnici broja 8. Ukoliko ulazne slike u koder nisu ovakvih dimenzija, one se obrezuju pri učitavanju.

**frame\_type frame**:Sadrži najvažnije podatke o slici koja se trenutno obrađuje.

**int CurrMbAddr** – indeks makrobloka koji se trenutno obrađuje. Raspon vrijednosti koje ova varijabla može poprimiti je [0, N–1], gdje je N broj makroblokova u slici.

# Komponente programa

## fer\_h264.cpp

Ova datoteka sadrži ulaznu točku u program. U njoj postoje samo dvije funkcije – decode() i encode() koje pokreću redom dekoder odnosno koder.

### void decode()

Upravlja dekodiranjem ulaznog toka sintakse H.264. Otvara ulazni i izlazni tok. Pokreće funkcije za upravljanje dinamičkim kadrovima (resursima), što uglavnom uključuje inicijaliziranje i oslobađanje dinamički alocirane memorije. Najvažnija zadaća ove funkcije je dohvaćanje po jedne NAL-jedinice i njezino predavanje funkciji RBSP\_decode u kojoj se odvija dekodiranje same sintakse H.264.

## nal.cpp

Upravljanje NAL-jedinicama u koderu i dekoderu. Komunikacija s tokom podataka koji sadrži sintaksu H.264. U jednoj se NAL-jedinici nalazi jedan skup parametara ili jedna čitava slika – nije podržana podjela slika na odsječke.

### Globalne varijable i strukture podataka

typedef struct

{

bool forbidden\_zero\_bit;

unsigned int nal\_ref\_idc, nal\_unit\_type, NumBytesInRBSP;

unsigned char \*rbsp\_byte;

}

NALunit;

**NALunit**: Struktura koja sadrži sve značajne podatke o NAL-jedinici. Značenja i nazivi varijabli jednaki su onima u Normi (vidi poglavlje 7.4 u Normi).

**NALunit \*NALbytes**: niz koji sadrži oktete trenutne NAL-jedinice.

### void getNAL(unsigned long \*fPtr, NALunit &nu)

**void fPtr**: kazaljka na trenutni oktet unutar ulaznog toka. Ova se kazaljka koristi kao parametar offset u pozivu STL funkcije fseek. fPtr se predaje kao referenca.

**NALunit &nu**: primjerak strukture NALunit, sadrži izdvojene podatke iz NAL jedinice koja će se sljedeća dekodirati. nu se predaje kao referenca.

Funkcija pronalazi veličinu trenutne NAL-jedinice u ulaznome toku pozivom funkcija findNALstart odnosno findNALend. Učitava NAL-jedinicu u niz NALbytes te ju predaje funkciji parseNAL koja podatke razvrstava u strukturu nu. Po završetku funkcije getNAL u strukturi nu se nalaze značajni podatci iz trenutne NAL-jedinice. Ova se funkcija poziva isključivo u dekokderu.

### void parseNAL(unsigned int NumBytesInNALunit,

### NALunit &nal\_unit);

**unsigned int NumBytesInNALunit**: broj učitanih okteta sadržanih u globalnome nizu NALbytes.

**NALunit nal\_unit**: struktura u koju se popunjavaju interpretirani podatci u kontekstu NAL-jedinice pročitani iz niza NALbytes. nal\_unit se predaje kao referenca.

Odvaja podatke o NAL-jedinici od podataka RBSP-a i upisuje ih u strukturu nal\_unit. Pronađeni kodovi za izbjegavanje emulacije (0x000003) se izbacuju prije prijepisa podataka u niz nal\_unit.rbsp\_byte. Ova se funkcija poziva isključivo u dekokderu.

### void writeNAL(NALunit nu)

**NALunit nu**: struktura koja predstavlja kodiranu NAL-jedinicu koja će se zapisati u izlazni tok.

nu je ulazna struktura te su svi podatci već u njoj zapisani. Ova funkcija ispisuje te podatke u odgovarajućemu formatu u izlazni tok. Kako bi se smanjila učestalost pristupanja datoteci, NAL-jedinice se privremeno zapisuju u međuspremnik streamBuffer veličine streamBufferSize pozivom funkcije writeToBuffer. Za inicijalizaciju međuspremnika streamBuffer potrebno je jednom, pri pokretanju programa pozvati funkciju InitNAL. Ova se funkcija poziva isključivo u koderu.

## rbsp\_decoding.cpp

Upravljanje najvažnijim postupcima dekodera: entropijskim dekodiranjem, predviđanjem, inverznom transformacijom i dekvantizacijom, učitavanjem ostataka, itd. Ova se komponenta koristi samo u dekoderu.

### void RBSP\_decode(NALunit nal\_unit)

**NALunit nal\_unit**: struktura u koju se popunjavaju interpretirani podatci u kontekstu NAL-jedinice pročitani iz niza NALbytes. nal\_unit se predaje kao referenca.

U ovisnosti o tipu NAL-jedinice (varijabla nal\_unit.nal\_unit\_type) dekodira podatke iz RBSP-a. Podržani tipovi NAL-jedinica su 1, 5, 7 i 8, koji redom odgovaraju tipovima ne-IDR, IDR, skup parametara niza i skup parametara slike. Nakon učitavanja skupova parametara inicijaliziraju se potrebni podatci poput dimenzija slike, potrebne memorije i sličnoga. Izvedeno dekodiranje za svaki od navedenih tipova izravno odgovara pseudokodovima opisanima u poglavlju 7.3 Norme. Svako odstupanje od Norme komentirano je u kôdu. U većini slučajeva odstupanja se pojavljuju zbog ograničenja profila *baseline*.

Osim učitavanja sintaksnih elemenata opisanog u poglavlju 7.3, u ovoj se funkciji pozivaju i funkcije za interpretaciju učitanih podataka. Prvo se izvodi predviđanje (intra – funkcija intraPrediction ili inter – funkcije DeriveMVs i Decode, ovisno o tipu makrobloka), zatim inverzna transformacija i dekvantizacija te rekonstrukcija slike (funkcije transformDecodingIntra\_16x16Luma, transformDecoding4x4Luma-Residual i transformDecodingChroma) i njen ispis u izlazni, nekomprimirani format Y4M[[1]](#footnote-1) (funkcija writeToY4M). Funkcije initialisationProcess i modificationProcess zaslužne su za upravljanje nizom referentnih slika kao što je opisano u poglavlju 8.2.4 Norme. Trenutno je podržano samo predviđanje u odnosu na neposredno prethodnu sliku pa se ovim funkcijama samo sprema posljednje dekodirana slika kao referentna za sljedeću P-sliku.

## rbsp\_encoding.cpp

Upravljanje najvažnijim postupcima kodera: entropijskim dekodiranjem, predviđanjem, transformacijom i kvantizacijom, upisivanjem ostataka, itd. Ova se komponenta koristi samo u koderu.

### void RBSP\_encode(NALunit &nal\_unit)

**NALunit &nal\_unit**: struktura u koju se upisuje (u član nal\_unit.rbsp\_byte) RBSP jedinica koja će se kodirati.

Kodiranje pojedine RBSP jedinice. Funkcija razlikuje tri načina kodiranja za tri vrste NAL-jedinica: skup parametara niza (SPS), skup parametara slike (PPS) i odsječak slike – I ili P. Svaka slika se kodira kao jedan odsječak. Odluka o tipu NAL-jedinice donosi se izvan ove funkcije. Nakon kodiranja skupa parametara niza (funkcija sps\_write), inicijaliziraju se potrebne varijable i alocira memorija.

Svaka IDR NAL-jedinica je I-slika i svaka ne-IDR NAL-jedinica je P-slika i sastoji se isključivo od P-makroblokova (iako norma dopušta i I-makroblokove unutar P-odsječaka). Ukoliko je trenutna NAL-jedinica I-slika, prije kodiranja zaglavlja poziva se funkcija IntraCL, koja pokreće intra kodiranje na GPU, ukoliko je u sustavu pronađena GPU koja podržava OpenCL.

Kodiranje slika odvija se po makroblokovima, stoga je glavna petlja unutar kodiranja odsječaka petlja po svim makroblokovima slike. U svakom se koraku petlje provodi predviđanje intra (funkcija intraPredictionEncoding) ili inter (funkcija interEncoding), transformacija i kvantizacija, entropijsko kodiranje parametara predviđanja i ostataka. Nakon transformacije i kvantizacije, poziva se funkcija setCodedBlockPattern koja postavlja varijable CodedBlockPatternLuma i CodedBlockPatternChroma. Ove dvije varijable govore u kojoj četvrtini makrobloka postoje ne-nul transformacijski koeficijenti. Za detaljnije objašnjenje ovih varijabli, vidi poglavlje 7.4 Norme.

Funkcije initialisationProcess i modificationProcess zaslužne su za upravljanje nizom referentnih slika kao što je opisano u poglavlju 8.2.4 Norme. Trenutno je podržano samo predviđanje u odnosu na neposredno prethodnu sliku pa se ovim funkcijama samo posljednje dekodirana slika sprema kao referentna za sljedeću P-sliku.

Na kraju kodiranja pojedine RBSP jedinice, poziva se funkcija RBSP\_trailing\_bits, kako je opisano u pseudokodu u poglavlju 7.3.2.11 Norme.

### unsigned int coded\_mb\_size(int intra16x16PredMode, int predL[16][16], int predCb[8][8], int predCr[8][8])

**int intra16x16PredMode**: odabrani način unutarslikovnog predviđanja 16×16, ako je vrijednost ove varijable -1, odabrano je predviđanje 4×4 te su načini za pojedine podblokove zapisani u globalnome nizu Intra4x4PredMode.

**int predL[16][16]**: predviđeni uzorci luminantne komponente.

**int predCb[8][8], int predCr[8][8]**: predviđeni uzorci krominantnih komponenata.

**povratna vrijednost**: broj bitova potrebnih za zapis trenutnog makrobloka uz odabrani način predviđanja.

Funkcija emulira dio funkcije RBSP\_encode, no umjesto zapisa bitova nakon entropijskog kodiranja, ona samo broji koliko je bitova potrebno za zapis te ukupan zbroj vraća kao povratnu vrijednost. Ova se funkcija koristi pri odabiru tipa makrobloka (4×4 ili 16×16) u unutarslikovnome predviđanju.

## intra.cpp

Skup svih postupaka korištenih za unutarslikovno predviđanje. Uključuje funkcije dekodera i kodera.

### void intraPrediction(int predL[16][16], int predCr[8][8], int predCb[8][8])

**int predL[16][16]**: predviđeni uzorci luminantne komponente.

**int predCb[8][8], int predCr[8][8]**: predviđeni uzorci krominantnih komponenata.

Funkcija obavlja unutarslikovno predviđanje za trenutni makroblok na osnovi učitanih parametara predviđanja i dosada dekodiranih makroblokova. Funkcija strogo poštuje postupke opisane u poglavlju 8.3 Norme, te su u kodu naznačena odgovarajuća potpoglavlja za svaki postupak.

Za predviđanje intra 4×4 potrebni su načini predviđanja iz susjednih podmakroblokova. Zbog toga se oni spremaju u globalno polje Intra4x4PredMode koje je kardinalnosti jednakoj ukupnom broju mogućih podmakroblokova 4×4 u slici. Indeks pojedinog načina predviđanja unutar tog polja jednak je CurrMbAddr \* 16 + luma4x4BlkIdx. Za makroblokove tipa intra 16×16 te vrijednosti nisu postavljene. Tipovi makroblokova nalaze se u globalnome polju mb\_type\_array, indeks tipa pojedinog makrobloka jednak je indeksu toga makrobloka u slici (npr. za trenutni makroblok to je CurrMbAddr). Budući da makroblokovi tipa intra 16×16 ne postavljaju vrijednosti u polju Intra4x4PredMode, za dohvat načina predviđanja susjednih podblokova u predviđanju načina predviđanja intra 4×4 prvo se provjerava je li makroblok kojemu pripada susjedni podblok tipa intra 4×4.

Kod izvedbe predviđanja intra 4×4, svaki se podblok mora odmah rekonstruirati (unutar postupka predviđanja za trenutni makroblok) jer rekonstruirani uzorci moraju biti dostupni kao uzorci za predviđanje u narednim podmakroblokovima. Zbog toga se na kraju izvedbe predviđanja za svaki podblok 4×4 poziva funkcija transformDecoding4x4LumaResidual.

Ako je trenutni makroblok tipa intra 16×16, u ovoj se funkciji izvodi samo izračun predviđenih uzoraka koji se kasnije predaju funkciji za transformaciju i kvantizaciju, koji inverzno transformirane i dekvantizirane ostatke pribrajaju ovim uzorcima i time rekonstruiraju makroblok.

Ova se funkcija poziva isključivo u dekoderu.

### int intraPredictionEncoding(int predL[16][16],

### int predCr[8][8], int predCb[8][8])

**int predL[16][16]**: predviđeni uzorci luminantne komponente.

**int predCb[8][8], int predCr[8][8]**: predviđeni uzorci krominantnih komponenata.

Funkcija za kodiranje i izvedbu načina predviđanja za trenutni makroblok. Razlozi odabranih postupaka opisani su u radu „Izvedba i optimiranje intra predikcije za normu MPEG-4 part 10“. Za svaki se makroblok odmah provodi i dekodiranje te se rezultantnim makroblokom prepisuje izvorni makroblok u izvornoj slici (varijabla frame). U funkciji postoje dva rješenja kodiranja u ovisnosti o tome je li u sustavu na kojemu se program izvodi pronađena GPU sukladna s OpenCL-om.

**Ako se ne koristi OpenCL**, provodi se svaki mogući način predviđanja 16×16 za luminantne uzorke. Najbolji način odabire se funkcijom SATD. Ukoliko za neki način nisu dostupni svi uzorci za predviđanje način se preskače. Funkcijom coded\_mb\_size provjerava se broj bitova potrebnih za zapis makrobloka za odabrano predviđanje intra 16×16.

Nakon odabira najboljeg načina intra 16×16, odabiru se najbolji načini predviđanja intra 4×4 za 16 podblokova trenutnoga makrobloka. Zatim se postavljaju parametri za predviđanje 4×4 funkcijom setIntra4x4PredMode i provodi se samo predviđanje. Budući da su za predviđanje podmakroblokova potrebni rekonstruirani uzorci iz susjednih podmakroblokova, u svakome se podmakrobloku za najbolji pronađeni način odmah provodi i rekonstrukcija podmakrobloka. Za dobivene se uzorke predviđanja ponovno poziva funkcija coded\_mb\_size te ukoliko je za predviđanje intra 4×4 potrebno manje bitova nego za predviđanje 16×16, odabire se ovo predviđanje te nije potrebno više provoditi rekonstrukciju kodiranog makrobloka, budući da je ona sastavni dio provedbe predviđanja 4×4 kako je napomenuto prethodno. S druge strane, ako je manje bitova potrebno za predviđanje 16×16, ponovno se mora izvesti predviđanje za odabrani način i rekonstruirati makroblok. Međutim, provedba predviđanja 4×4 zbog nužne rekonstrukcije prepisala je uzorke izvornoga makrobloka dekodiranima. Zbog toga se prije provedbe predviđanja 4×4 izvorni makroblok sprema u privremeno polje original[16][16], prema kojemu se ponovno provodi predviđanje 16×16 ukoliko je odabrano kao bolje.

Za krominantne komponente odabire se način predviđanja ekvivalentan najboljem pronađenom načinu 16×16 (ne izvodi se posebna pretraga najboljeg načina).

**Ako se koristi OpenCL**, u ovoj se funkciji preskače postupak pronalaska najboljeg načina predviđanja, budući da se on provodi na GPU. Odabrani načini predviđanja zapisani su u nizovima predModes16x16 odnosno predModes4x4. Za svaki se makroblok uspoređuje samo broj bitova potrebnih za zapis makrobloka najboljim načinom 16×16 i najboljim načinom 4×4.

Ova se funkcija poziva isključivo u koderu.

## openCL\_functions.cpp

Sadrži sve funkcije za upravljanje OpenCL-om. Ako nije pronađena GPU koja podržava OpenCL, globalna varijabla OpenCLEnabled postavlja se u false i program ne koristi OpenCL (ne koristi se emulacija na CPU, iako je moguća).

### void InitCL()

Funkcija za inicijalizaciju sučelja OpenCL. Potrebno ju je pokrenuti samo jednom pri pokretanju programa. Funkcija pronalazi platformu OpenCL na sustavu, traži GPU koja podržava OpenCL te ako ju ne pronalazi, varijabla OpenCLEnabled se postavlja u false, funkcija završava i program ne koristi OpenCL. Ako je pronađena adekvatna GPU, stvara se kontekst i učitavaju se jezgrice iz datoteka h264\_kernels.cl i intra\_kernels.cl te se prevode za pronađenu karticu. Za svaku jezgricu postavlja se globalna kazaljka kojom se pokreće pojedina jezgrica kada je to potrebno.

### void IntraCL()

Pokreće odabir načina predviđanja za svaki makroblok u trenutnoj slici na GPU. Luminantna komponenta trenutne slike (varijabla frame.L) preslikava se u memoriju GPU. Postavljaju se ulazni parametri jezgrica i pokreću se jezgrice. Pokreće se i učitavanje rezultata (načini predviđanja 16×16 i 4×4), ali u neblokirajućem načinu, što znači da po povratku iz funkcije podatci možda još nisu učitani u izlazne nizove (predModes4x4 i predModes16x16), dapače, u ovome slučaju sigurno nisu. Pri pozivu funkcije za učitavanje rezultata, predaju se parametri eventReadPredModes4x4 i eventReadPredModes16x16, koji predstavljaju događaje učitavanja. Rezultati stoga unutar ove funkcije (IntraCL) nisu učitani, već se neposredno prije njihovog korištenja (u funkciji intraPredictionEncoding) poziva funkcija clWaitForEvents[[2]](#footnote-2) koja blokira izvođenje programa dok rezultati nisu u potpunosti učitani s GPU.

## intra\_kernels.cl

Funkcije u ovoj datoteci većinom su identične onima u datoteci intra.cpp, uz neke nužne preinake zbog SIMD arhitekture GPGPU.

Komunikacija s globalnom memorijom na GPU vrlo je skupa, stoga ju treba što je više moguće smanjiti. Iz toga se razloga, na početku svake od dviju jezgrica (GetIntra4x4PredModes i GetIntra16x16PredModes) uzorci izvornog makrobloka odnosno podbloka dohvaćaju u lokalno polje original[16][16], odnosno original[4][4].

Budući da su računske jedinice na GPU SIMD strojevi, preskakanje izračuna predviđanja i SATD-a za makroblokove ili podblokove za načine za koje nisu dostupni svi potrebni uzorci za predviđanje nema smisla, jer se ionako vrijeme troši na dretvama u kojima su mogući svi načini predviđanja (vidi rad „Izvedba i optimiranje intra predikcije za normu MPEG-4 part 10“). Dapače, zato što se u tokovnim jezgrama mora izvesti svako grananje, ovo znatno usporava prevođenje jezgrica, pa se grananja moraju što više izbjegavati. Stoga se za nemoguće načine predviđanja ocjena funkcije SATD jednostavno postavlja na INT\_MAX.

Najznačajnija promjena u odnosu na funkcije korištene u datoteci intra.cpp jest vektorizacija transformacije i kvantizacije. Umjesto polja 4x4, matrice korištene za ove funkcije predstavljene su kao nizovi od četiri vektora int4. Odluka o tome predstavljaju li vektori retke ili stupce matrica donesena je u ovisnosti o smjeru paralelnosti operacija, te je komentirana u kodu.

## mode\_pred.cpp

Manipuliranje vektorima pomaka. Sadrži sve potrebne funkcije za rekonstrukciju vektora pomaka za trenutni frame. Sastoji se od funkcija potrebnih za predviđanje vektora pomaka te funkcije za konačnu rekonstrukciju na temelju predviđenih vektora pomaka i predane razlike MVd.

### Globalne varijable i strukture podataka

int \*\*\*mvL0x;

int \*\*\*mvL0y;

int \*subMvCnt;

int \*refIdxL0;

**\*\*\*mvL0x i \*\*\*mvL0y**: rekonstruirani vektori pomaka za sve blokove trenutne slike. Varijable su globalne jer se koriste i za vrijeme određivanja predviđanja u koderu, gdje je moguće direktno postaviti određeni vektor pomaka.

**\*subMvCnt**: broj vektora pomaka za svaki makroblok.

**\*refIdxL0**: referentna slika za svaki makroblok. U baseline profilu H.264 je ova vrijednost 0 (prethodna slika) za sve blokove.

### void AllocateMemory()

Funkcija priprema memoriju za globalne varijable mode\_pred-a. Potrebno ju je pozvati prilikom inicijalizacije kodera i dekodera.

### void get\_neighbour\_mv(int mbAddrN, int mbPartIdx,

### int \* mvNx, int \* mvNy, int \* refIdxL0N)

**mbAddrN**: makroblok od kojeg je potrebno dobiti vektor pomaka.

**mbPartIdx**: makroblok od kojeg je potrebno dobiti vektor pomaka.

Funkcija preko mvNx i mvNy parametara vraća vektor pomaka za podblok mbPartIdx makrobloka mbAddrN. Dodatno, vraća i koja je referentna slika (uvijek 0).

### void DeriveNeighbourLocation(int xN, int yN,

### int \* mbAddrN, int \* xW, int \* yW, bool \* validN)

**xN**: udaljenost po x koordinati od gornjeg lijevog kuta trenutnog makrobloka.

**yN**: udaljenost po y koordinati od gornjeg lijevog kuta trenutnog makrobloka

**\*mbAddrN**: povratna vrijednost koja određuju rezultantni makroblok.

**\*xW, \*yW**: pozicija unutar rezultantnog makrobloka.

**\*validN**: povratna vrijednost koja određuju je li rezultatni makroblok valjan.

Funkcija vraća unutar kojeg makrobloka (ako je valjan) i na kojem mjestu tog makrobloka se nalazi točka koja je udaljena za [xN, yN] od gornje-lijevog kuta trenutnog makrobloka.

### void DeriveNeighbourPartitions(int mbPartIdx,

### int subMbPartIdx,

### int \* mbAddrA, int \* mbPartIdxA, int \* subMbPartIdxA,

### bool \* validA, int \* mbAddrB, int \* mbPartIdxB,

### int \* subMbPartIdxB, bool \* validB, int \* mbAddrC,

### int \* mbPartIdxC, int \* subMbPartIdxC, bool \* validC,

### int \* mbAddrD, int \* mbPartIdxD, int \* subMbPartIdxD,

### bool \* validD)

**mbPartIdx:** redni broj podbloka unutar trenutnog makrobloka.

**subMbPartIdx**: redni broj dijela podbloka

**mbAddrN, mbPartIdxN, subMbPartIdxN, validN:** identifikacija susjednog dijela (N se zamjenjuje sa A, B, C ili D) s rednim brojem makrobloka unutar slike (mbAddr), rednim brojem podbloka (mbPartIdx), rednim brojem podjele na 4x4 (subMbPartIdx), jeli taj dio valjan (valid).

Funkcija identificira susjede za dio trenutnog bloka (koji može biti bilo koji 8x8 dio, ili 4x4). Ukoliko neki susjedni blok nije dostupan (nalazi se izvan okvira slike), označava se flag Valid sa false. Ova funkcija se koristi kod rekonstrukcije vektora pomaka jer se predviđeni vektori pomaka dobivaju na temelju vektora pomaka od susjeda. Kako je dekoder sastavni dio kodera, funkcija se koristi u oba.

### void PredictMV\_LumaSubMB(int mbPartIdx, int subMbPartIdx)

**mbPartIdx:** određuje podblok trenutnog makrobloka (veličine 8x8).

**subMbPartIdx:** određuje dodatnu podjelu podbloka.

Funkcija predviđa vektor pomaka za određeni dio trenutnog makrobloka i zapisuje ga u polje mvL0x, odnosno mvL0y. Za predviđanje vektora pomaka koriste se vektori pomaka susjednih blokova dohvaćeni funkcijom DeriveNeighbourPartitions. Ukoliko nije dostupan susjed C, tada se umjesto njegovih vektora pomaka koriste vektori pomaka od susjeda D. Nadalje, ako je dostupno manje od tri vektora pomaka i nakon moguće potrebne zamjene susjeda C sa D, tada se svim nedostupnim susjedima postavljaju vektori pomaka prvog dostupnog (to je obično A ili B). Nakon ovog koraka, konačni predviđeni vektor pomaka za dio trenutnog makrobloka, određenog sa mbPartIdx i subMbPartIdx, se postavlja na srednju vrijednost vektora pomaka susjeda A, B i C.

### void PredictMV\_Luma(int mbPartIdx)

**mbPartIdx:** dio trenutnog makrobloka.

Funkcija predviđa vektor pomaka za dio trenutnog makrobloka te se zapisuje u [mvL0x, mvL0y]. Ako je neki dio tipa P\_L0\_8x8, tada se dodatno za njegove dijelove poziva funkcijaPredictMV\_LumaSubMBkako bi se predvidjeli vektori pomaka i za sitnije dijelove.

### void PredictMV()

Funkcija predviđa vektore pomaka za trenutni makroblok pozivima funkcije PredictMV\_Luma**.** Ako je trenutni makroblok predviđan P\_Skip predikcijom, tada se dodatno postavlja i vrijednost razlike vektora pomaka na 0 (mvd\_l0 = 0). Kako se vektori pomaka računaju na temelju susjednih blokova, koji mogu biti i unutar trenutnog makrobloka, nakon predviđanja pokreta na pojedini blok, automatski se pribraja i razlika mvd\_l0.

### void DeriveMVs()

Funkcija poziva funkciju za predviđanje vektora pomaka te ako je riječ o većoj podjeli trenutnog makrobloka, prepisuje vektore pomaka na podblokove (kako bi se kasnije olakšala rekonstrukcija slike i drugih vektora pomaka).

## mocomp.cpp

Rekonstrukcija slike na temelju referentne i vektora pomaka. Ovdje se nalaze funkcije za interpoliranje vrijednosti na necjelobrojnim lokacijama slike, određivanje predviđenih makroblokova i konačne rekonstrukcije (predviđeno + ostatak).

### void FillTemp\_4x4\_refPart(frame\_type \*ref, int org\_Lx, int org\_Ly, int org\_Cx, int org\_Cy)

**\*ref:** referentna slika.

**org\_Lx, org\_Ly:** lokacija luminancije u referentnoj slici.

**org\_Cx, org\_Cy:** lokacija boje u referentnoj slici.

Funkcija iz referentne slike dohvaća podatke potrebne za interpoliranje vrijednosti na necjelobrojnim pozicijama, a koji se nalaze unutar 4x4 bloka s gornjim lijevim kutom na poziciji org\_Lx, org\_Ly. Potrebni podaci se spremaju u nizove L\_Temp\_4x4\_refPart i C\_Temp\_4x4\_refPart.

### Tap6Filter(E,F,G,H,I,J) Border(((E)-5\*(F)+20\*(G)+20\*(H)- 5\*(I)+(J)+16)>>5)

Ova funkcija interpolira vrijednost na sredini pravca koji je redom definiran točkama E, F, G, H, I i J. Točnije, interpolira vrijednosti na sredini dužine GH.

### int L\_MC\_frac\_interpol(int \*data, int frac)

**data:** pokazivač na niz L\_Temp\_4x4\_refPart.

**frac:** oznaka za pomak do četvrtine piksela po x i y koordinati.

Funkcija na temelju podataka L\_Temp\_4x4\_refPart i funkcije za interpolaciju (Tap6Filter) određuje vrijednost određene pozicije unutar bloka 4x4. Funkcija kad izračuna potrebnu vrijednost odmah prekida iz razloga što je kod rekonstruiranja za sve elemente bloka 4x4 potrebno odrediti samo određenu interpolaciju.

### void FillInterpolSubMBPart(int predL[16][16][16], int sx,

### int sy, int \*data)

**predL:** parametar za povratnu vrijednost.

**sx, sy:** piksel kod kojeg je potrebno obaviti interpolaciju svih 16 vrijednosti.

**data:** pripremljeni podaci za moguću interpolaciju vrijednosti bloka 4x4.

Funkcija radi isto što i L\_MC\_frac\_interpol, samo za razliku od nje ne prekida pri dolasku do određene pozicije za interpoliranje, već redom interpolira vrijednost za 4x4 međupikselne pozicije kojima je gornji lijevi kut određen koordinatama **sx, sy.**

### void FillInterpolatedMB(int predL[16][16][16],

### int predCr[16][8][8], int predCb[16][8][8],

### frame\_type \*refPic)

**predL, predCr, predCb:** izlazni parametri za 16 interpolirajućih vrijednosti kod svakog piksela trenutnog makrobloka (veličine 16x16).

**\*refPic:** referentna slika.

Funkcija se koristi kod kodera za potrebe prvotnog punjenja niza svih interpolirajućih vrijednosti referentne slike, a različita je od standarne takve funkcije kod dekodera iz razloga što puni za sve vektore pomaka, tj. određuju se interpolirajuće vrijednosti na svim mogućim lokacijama unutar referentne slike.

### void MotionCompensateSubMBPart(int predL[16][16],

### int predCr[8][8], int predCb[8][8], frame\_type \*refPic)

**predL, predCr, predCb:** izlazni parametri za predviđene vrijednosti kod svakog piksela trenutnog makrobloka (veličine 16x16) na temelju mvL0x i mvL0y.

**\*refPic:** referentna slika.

Funkcija koristi globalna polja mvL0x i mvL0y, te na temelju vektora pomaka iz tih polja predviđa vrijednosti trenutnog makrobloka. Ako je potrebno, obavlja interpolaciju luminancije prema funkciji L\_MC\_frac\_interpol**,** a za boju obavlja linearnu interpolaciju (linearna interpolacija između dva susjedna piksela).

### void Decode(int predL[16][16], int predCr[8][8],

### int predCb[8][8])

**predL, predCr, predCb:** izlazni parametri za rezultantne predviđene vrijednosti kod svakog piksela trenutnog makrobloka (veličine 16x16).

Funkcija pozivima MotionCompensateSubMBPart za sve blokove veličine 4x4 unutar trenutnog makrobloka puni konačne predviđene vrijednosti za trenutni makroblok. Ova funkcija se poziva i kod kodera i dekodera, a namjerno je postavljeno posebno kako bi program kodiranja/dekodiranja bio čitljiviji pozivom samo funkcije Decode, a MotionCompensateSubMBPartza svaki dio. Stvarne vrijednosti piksela trenutnog makrobloka se dobivaju tek kasnije kad se predviđenim vrijednostima pribroji i ostatak (globalno **residual**).

## moestimation.cpp

Ovdje se nalazi skup funkcija kodera H.264 za međuslikovno predviđanje.

### Globalne varijable i strukture podataka

frame\_type refFrameInterpolated[16];

int MAXDIFF = 2;

int \*\*refFrameKar[5][16];

int \*sortedSuma0[5];

**refFrameInterpolated[16]**: sve potrebne interpolirajuće vrijednosti referentne slike za predviđanje trenutne. Po svakom pikselu postoji 16 interpolirajućih vrijednosti jer su mogući pomaci za četvrtinu piksela.

**MAXDIFF**: dopuštena razlika predviđenih piksela u odnosu na stvarne. Ova varijabla se uglavnom koristi za određivanje može li se trenutni makroblok predviđati P\_Skip tipom predikcije. Ova varijabla ima izuzetnu važnost jer dok povećava broj makroblokova koji su P\_Skip predviđeni i time povećava kompresiju, degradira kvalitetu kasnije rekonstrukcije.

**\*\*refFrameKar[5][16]**: pet karakteristika potrebnih za određivanje potencijalno najboljih vektora pomaka. Karakteristike se računaju za sve moguće referente pozicije, dakle ima ih 16 puta više nego piksela u slici.

**\*sortedSuma0[5]**: nizovi potrebni za prvo filtriranje velikih vektora pomaka za trenutni blok veličine 8x8. U ovom nizu se nalaze vektori pomaka sortirani prema sumi luminantnih vrijednosti.

### void InitializeInterpolatedRefFrame()

Funkcija alocira potrebnu memoriju za globalna polja koja koristi koder za međuslikovno predviđanje. Koristi se samo u koderu i potrebnu ju je pozvati prilikom inicijalizacije kodera.

### void FillInterpolatedRefFrame()

Ova funkcija se koristi za punjenje nizova refFrameInterpolated**,** refFrameKar**,** sortedSuma0**.** Poziva se nakon obrade pojedine slike kako bi se pripremili svi potrebni podaci za predviđanje iduće slike. refFrameInterpolated se puni pozivanjem FillInterpolSubMBPart za svaki makroblok slike. Unutar ove funkcije su implementirane dvije optimizacije – linearno punjenje refFrameKarpolja te linearno sortiranje pozicija u polje sortedSuma0. Linearno punjenje refFrameKar polja je izvedeno dvama prolazima kroz polje refFrameInterpolated pri čemu se u prvom prolazu na svaku poziciju zapiše suma luminantnih komponenti svih piksela dolje-desno od trenutnog, a drugom prolazu se na temelju tih vrijednosti formulama određuju vrijednosti pojedinih karakteristika. Linearno sortiranje u polju sortedSuma0 je napravljeno u odnosu na sumu luminantnih vrijednosti piksela u 8x8 bloku korištenjem counting sorta. Counting sort je moguć u ovom slučaju jer suma luminantnih vrijednosti za pojedini blok može poprimiti vrijednosti u intervalu [0, 16384].

### int sadLuma8x8MVs(int mvx, int mvy, int luma8x8BlkIdx)

**mvx, mvy**: pomak za koji je potrebno izračunati SAD (*Sum of Absolute Differences*) u odnosu na trenutni blok veličine 8x8.

**luma8x8BlkIdx:** redni broj 8x8 bloka unutar trenutnog makrobloka za koji se računa funkcija SAD.

Funkcija vraća SAD između trenutnog bloka 8x8 i predviđenog vektorom pomaka mvx**,** mvy. Funkcija za određivanje SAD-a za predviđeni blok koristi ranije napunjen niz refFrameInterpolated**.**

### int ExactPixels(int predL[16][16])

**predL:** predviđeni makroblok.

Funkcija vraća broj piksela koji su jednaki ili skoro jednaki u trenutnom makrobloku i predviđenom ukoliko se pozove Decode() s definiranim predviđanjem P\_Skip. Ova funkcija se koristi kod određivanja hoće li se za trenutni makroblok koristiti P\_Skip predviđanje i na taj način maksimalno povećati kompresija.

### void PopraviPSkip(int predL[16][16])

**predL:** predviđeni makroblok.

Funkcija popravlja trenutni blok na temelju predviđenog ukoliko se koristi P\_Skip predviđanje. To je nužno jer se kod P\_Skip predviđanja najčešće dopuštaju odmaci od stvarnog makrobloka, a kako greška ne bi propagirala na naredne slike, potrebno je prepraviti trenutni makroblok kako bi izgledao identično onom predviđenom kasnije u dekoderu.

### void MEstimation(int sx, int sy, int granica, int stepMV, int stepFrac, int px, int py)

**sx, sy:** početna pozicija bloka 8x8.

**px, py:** predviđeni vektor pomaka.

**granica:** mogući odmak vektora pomaka od predviđenog.

**stepMV:** svakih koliko piksela unutar granice provjeriti dobrotu vektora pomaka.

**stepFrac:** svakih koliko interpolirajućih vektora pomaka isprobati. Moguće je isprobati samo 16 po svakom pikselu, pa se obično ovaj parametar postavlja na 1 ili na 16 (isprobati sve interpolirajuće, isprobati samo cjelobrojni pomak).

Ovo je funkcija koja u globalno polje potencijalno najboljih vektora pomaka (bmin, bxs, bys) postavlja N najboljih iz skupa određenog parametrima pri pozivu funkcije. Funkcija za odabir njih N koristi ranije definirane i izračunate karakteristike blokova 8x8.

### void basicInterEncoding(int predL[16][16],

### int predCr[8][8], int predCb[8][8])

**predL, predCr, predCb:** povratne vrijednosti predstavljaju predviđeni makroblok.

Funkcija koja obavlja osnovno međuslikovno predviđanje korištenjem potpunog pretraživanja (isprobava redom sve pomake unutar zadanog prozora pretraživanja). Svaki od vektora pomaka se isprobava na način da se prvo on direktno postavi kao konačan u mvL0x i mvL0y te zatim pozove funkcijaDecode(). Po izvršavanju funkcijeDecode(),predL se uspoređuje s trenutnim makroblokom i ono predviđanje koje ima najmanje razlike (najmanja **SAD** funkcija) se postavlja kao odabrano. Dodatno, na početku se provjerava može li trenutni blok proći kao predviđanje P\_Skip. Taj dio je jako bitan za međuslikovno predviđanje, pa se ni u osnovnoj izvedbi ne izostavlja.

### void interEncoding(int predL[16][16], int predCr[8][8], int predCb[8][8])

**predL, predCr, predCb:** povratne vrijednosti predstavljaju predviđeni makroblok.

Kao i basicInterEncoding, funkcija započinje provjerom može li trenutni blok proći kao P\_Skip predviđanje. Ako može, tada se tip predviđanja postavlja na P\_Skip i poziva PopraviPSkipčime završava određivanje međuslikovnog predviđanja za trenutni makroblok. Ako to nije slučaj, nastavlja se odabir predviđanjima za svaki 8x8 podblok. Određuju se potencijalno najbolji pomaci po tri karakteristična tipa pomaka: jako kratki neovisni o susjedima, mali odmaci od susjednih pomaka, daleki pomaci različiti od susjednih. Prva dva tipa se rješavaju vrlo jednostavno pozivanjem MEstimation funkcije s time da se kao ciljni vektor pomaka postavlja MVp u jednom ili 0 drugom slučaju pri čemu se dobiva određen broj potencijalno najboljih pomaka. Zadnji tip pomaka se ne može efikasno riješiti jednostavnim pozivom funkcije MEstimation jer je potrebno provjeriti velik broj pomaka (najbolji pomak može biti prema bilo kojoj poziciji na referentnoj slici). U tu svrhu se koristi polje sortedSuma0 za prvotno filtriranje svih pozicija na referentnoj slici po ukupnoj sumi luminantnih vrijednosti, na način da se dopušta prosječno odstupanje za 3 po pikselima. Odnosno, dozvoljeno odstupanje ukupne sume luminantnih vrijednosti za trenutni blok je 180. Dodatno, ne uzimaju sve svi ti pomaci kao potencijalni nego samo oni kojima lijeva i desna polovica odnosno gornja i donja polovica imaju prosječno odstupanje po pikselu za luminantne komponente također do 3. Time se filtrira na predviđene blokove i po obliku slične trenutnom. Ovo filtriranje se radi po cjelobrojnim pomacima te se za određen broj tako najboljih cjelobrojnih pomaka poziva funkcija MEstimation malim prozorom pretraživanja za finije pomake. Od potencijalnih pomaka za sva tri tipa predviđanja se na temelju funkcije sadLuma8x8MVsodređuje najbolji pomak. Jednom određeni pomaci za svaki 8x8 blok, se po potrebi spajaju u veće cjeline čime se dobiva i konačna podjela trenutnog makrobloka na veličine 16x16, 16x8, 8x16 ili 8x8.

## Inttransform.cpp

Upravljanje transformacijom i skaliranjem blokova ostataka dobivenih kao razlika stvarnih i previđenih vrijednosti.

### void transformDecoding4x4LumaResidual

### (int LumaLevel[16][16], int predL[16][16],

### int luma4x4BlkIdx, int QPy)

**LumaLevel**: Blokovi ostataka za cijeli makroblok

**predL**: predviđene vrijednosti za cijeli makroblok

**luma4x4BlkIdx**: položaj bloka unutar makrobloka

**QPy**: koeficijent kvantizacije

Provodi transformaciju i skaliranje blokova ostataka. Svi podaci obrađuju se unutar ove funkcije bez obzira jesu li dobiveni unutarslikovnim ili međuslikovnim predviđanjem i pripadaju li komponenti osvjetljenja ili boje. Nakon provođenja transformacije i skaliranja (funkcija scaleAndTransform4x4Residual), dobivene je vrijednosti potrebno rekonstruirati za što se koristi funkcija pictureConstruction4x4Luma.

### void transformDecodingIntra\_16x16Luma

### (int Intra16x16DCLevel[16], int Intra16x16ACLevel[16][16],int predL[16][16], int QPy)

**Intra16x16DCLevel**: DC koeficijenti

**Intra16x16ACLevel**: AC koeficijenti za trenutačni makroblok

**predL**: predviđene vrijednosti za trenutačni makroblok

**QPy**: koeficijent kvantizacije

Implementira transformaciju i skaliranje DC koeficijenata dobivenih unutarslikovnim predviđanjem pozivanjem funkcije InverseDCLumaIntra. Nakon obrade, koeficijenti se vraćaju u originalne blokove i provodi se transformacija i skaliranje blokova ostataka dopunjenih s DC koeficijentima (scaleAndTransform4x4Residual)

### void transformDecodingChroma

### (int ChromaDCLevel[4], int ChromaACLevel[4][16], int predC[8][8], int QPy, bool Cb)

**ChromaDCLevel**: DC koeficijenti komponente boje

**ChromaACLevel**: AC koeficijenti komponente boje

**predC**: predviđene vrijednosti

**QPy**: koeficijent kvantizacije

**Cb**: oznaka komponente boje

Funkcija obavlja transformaciju DC koeficijenata i nakon toga blokova ostataka. Sve je identično kao i u funkciji koja obavlja transformaciju DC koeficijenata makroblokova dobivenih unutarslikovnim predviđanjem.

## scaleTransform.cpp

Implementacija funkcija koje stvarno obrađuju podatke, za razliku od prethodno opisanih funkcija koje zapravo samo pozivaju funkcije implementirane u scaleTransform.cpp. Potrebno je napomenuti da su sve funkcije napisane u skladu s normom. Drugim riječima, svaka funkcija implementira jedno od poglavlja transformacije i kvantizacije opisanih u normi.

### int LevelScale[6][4][4]

Globalno polje (pregledna tablica) koje sadrži unaprijed izračunate faktore koji se koriste prilikom skaliranja.

### void inverseTransform4x4(int d[4][4], int r[4][4])

**d**: blok ulaznih podataka

**r**: blok izlaznih podataka (rekonstruirani)

Inverzna transformacija za blokove ostataka implementirana točno onako kako je definirana u normi.

### void inverseTransformDCLumaIntraFast(int c[4][4],

### int f[4][4])

**c**: blok ulaznih (kodiranih) DC koeficijenata

**f**: izlani podaci (dekodirani) DC koeficijenti

Dekodira DC koeficijente koji se nakon toga vraćaju u blokove ostataka i dalje obrađuju.

### void transformDCChromaFast (int c[2][2], int f[2][2])

**c**: blok ulazni DC koeficijenata

**f**: izlani (dekodirani) DC koeficijenti

Dekodira DC koeficijente komponente boje koji se vraćaju u blokove ostataka.

### void scaleResidualBlock(int c[4][4], int d[4][4],

### int qP, bool intra16x16OrChroma)

**c**: ulazni podatci (kvantizirani)

**d**: izlazni podatci (skalirani)

**qP**: kvantizacijski koeficijent

**intra16x16OrChroma**: oznaka radi li se o komponenti osvjetljenja ili unutarslikovnom predviđanju

Funkcija ostvaruje skaliranje blokova ostataka i u slučaju da se radi o unutarslikovnom bloku ili o bloku komponente boje prvi se element ne skalira.

### void scaleLumaDCIntra(int f[4][4], int qP, int dcY[4][4])

**f**: ulazni podaci

**qP**: kvantizacijski koeficijent

**dcY**: skalirani DC koeficijenti

Ova funkcija implementira skaliranje DC koeficijenata blokova dobivenih unutarslikovnim predviđanjem.

### void scaleChromaDC(int f[2][2], int qP, int dcC[2][2])

**f**: ulazni podaci (dobiveni inverznom transformacijom)

**qP**: kvantizacijski koeficijent

**dcC**: skalirani DC koeficijenti

Implementacija skaliranja DC koeficijenata koji pripadaju komponentama boje.

## quantizationTransform.cpp

Ova datoteka sadrži implementaciju kompletne transformacije i kvantizacije potrebne za realizaciju kodera.

### int LevelQuantize[6][4][4]

Globalno polje (pregledna tablica) koje sadrži unaprijed izračunate faktore potrebne za efikasnu implementaciju kvantizacije.

### void forwardTransform4x4(int r[4][4], int d[4][4])

**r**: blok ostataka kojeg je potrebno kodirati

**d**: transformirani blok

Transformacija bloka ostataka dobivenih kao razlika originalnih i predviđenih vrijednosti. Ova transformacija obrađuje svaki blok veličine 4x4 elemenata unutar videa.

### void forwardTransformDCLumaIntra(int f[4][4], int c[4][4])

**f**: polje ulaznih DC koeficijenata (unutarslikovno predviđanje)

**c**: transformirani DC koeficijenti

### void forwardTransformDCChroma (int f[2][2], int c[2][2])

**f**: polje ulazni DC koeficijenata (komponenta boje)

**c**: transformirani DC koeficijenti

### void quantisationResidualBlock(int d[4][4], int c[4][4], int qP, bool Intra16x16OrChroma)

**d**: ulazni (transformirani) podaci

**c**: izlazni (potpuno kodirani) podaci

**qP**: kvantizacijski koeficijent

**Intra16x16OrChroma**: oznaka radi li se o unutarslikovnom predviđanju ili komponenti boje

### void quantisationLumaDCIntra(int f[4][4], int qP,

### int c[4][4])

**f**: ulazni (transformirani) DC koeficijenti – unutarslikovno predviđanje

**qP**: kvantizacijski koeficijent

**c**: izlazni (kvantizirani, u potpunosti kodirani) podaci

### void quantisationChromaDC(int f[2][2], int qP,

### int c[2][2])

**f**: ulazni (transformirani) DC koeficijenti – komponenta boje

**qP**: kvantizacijski koeficijent

**c**: izlazni (kvantizirani, u potpunosti kodirani) podaci

### void transformScan(int c[4][4], int list[16],

### bool Intra16x16AC)

**C**: ulazno dvodimenzionalno polje dimenzija 4x4

**list**: izlazna lista (jednodimenzionalno polje)

**Intra16x16AC**: oznaka radi li se o AC koeficijentima za unutarslikovno predviđanje (u tom se slučaju ne popunjava prvi element liste jer je to DC koeficijent)

Ova funkcija provodi zig-zag preraspodjelu podataka iz dvodimenzionalnog polja u listu koja se šalje u entropijski koder.

### void quantizationTransform(int predL[16][16],

### int predCb[8][8],int predCr[8][8], bool reconstruct)

**predL**: predviđene vrijednosti za komponentu osvjetljenja za trenutačni makroblok

**predCb**: predviđene vrijednosti za prvu komponentu boje

**predCr**: predviđene vrijednosti za drugu komponentu boje

**reconstruct**: oznaka radi li se odmah i rekonstrukcija kodiranih blokova

Ova funkcija je zapravo omotač (engl. *wrapper*) svih procesa transformacije i kvantizacije u koderu, tj. sučelje prema ostalim modulima kodera. Unutar ove funkcije pozivaju se sve ostale funkcije ovisno o vrsti podataka, tj. predviđanja koje se koristi za trenutačni makroblok.

1. Više o formatu Y4M na <http://wiki.multimedia.cx/index.php?title=YUV4MPEG2> [↑](#footnote-ref-1)
2. Točnije, it funkcije intraPredictionEncoding poziva se funkcija WaitIntraCL koja je omotač za funkciju clWaitForEvents. [↑](#footnote-ref-2)