

Question 2 YUV PSNR:

for (i = 0; i < iFrames; i++) {

//------------------------------------

//HW5:

// Read the YUV components of the two input videos.

// Example: ifs1.read((char \*)pcImgBuf1, iImageArea);

// Compute the PSNRs of the three components in each frame, save them in tmp[0] to tmp[2].

// Calculate the average PSNR of each component over all frames.

// calculate how many pixels to read from input

int Y\_Size = iImageArea; // Y frame is the entire frame

int U\_V\_Size = iImageArea / 4; // UV makes up 1/4 of the Y frame

//read in Y for both images, calculate PSNR store into tmp0

ifs1.read((char \*) pcImgBuf1, iImageArea);

ifs2.read((char \*) pcImgBuf2, iImageArea);

tmp[0] = GetPSNR(pcImgBuf1, pcImgBuf2, Y\_Size);

//read in U for both images, calculate PSNR store into tmp1

ifs1.read((char \*) pcImgBuf1, U\_V\_Size);

ifs2.read((char \*) pcImgBuf2, U\_V\_Size);

tmp[1] = GetPSNR(pcImgBuf1, pcImgBuf2, U\_V\_Size);

//read in V for both images, calculate PSNR store into tmp2

ifs1.read((char \*) pcImgBuf1, U\_V\_Size);

ifs2.read((char \*) pcImgBuf2, U\_V\_Size);

tmp[2] = GetPSNR(pcImgBuf1, pcImgBuf2, U\_V\_Size);

//sum up all the psnr's for each frame for YUV, average will be taken outside the for loop

psnr[0] = psnr[0] + tmp[0];

psnr[1] = psnr[1] + tmp[1];

psnr[2] = psnr[2] + tmp[2];

cout << i << ": " << tmp[0] << " " << tmp[1] << " " << tmp[2] << endl;

}

cout << "Average: " << psnr[0] / iFrames << " " << psnr[1] / iFrames << " " << psnr[2] / iFrames << endl;

ifs1.close();

ifs2.close();

delete pcImgBuf1;

delete pcImgBuf2;

return 0;

}

Question 3 Motion estimation:

void IEncoder::MBMotionEst(

float \*\*pfCurrFrame,

float \*\*pfRefFrame,

int y,

int x)

{

int i, j, iSAD0, iSAD = 0;

int iMVy = 0, iMVx = 0;

//Get SAD at MV=(0, 0) first for early termination purpose.

iSAD0 = GetSAD(pfCurrFrame, pfRefFrame, y, x, 0, 0, 65535);

//------------------------

// HW5: full-search motion estimation:

// For each MV candidate, call GetSAD() below to get its prediction error (SAD).

// Only update MV if the new SAD is less than certain ratio of previous minimum SAD (saved in iSAD0).

// Skip invalid MV (any pixel in the reference block is out of the frame boundary).

// Keep track of best candidate with iMVy and iMVx.

//------------------------

//temp variable for holding SAD value, to compare later

int sadTemp = 0;

//flag to determine if MV is within frame boundary

bool mvFlag = false;

// [-18,18] window

for (i = -18; i < 18; i++) {

for (j = -18; j < 18; j++) {

// if y + i, x + j is less than 0, then it's obviously outside reference frame

if (y + i < 0) {

mvFlag = true;

}

else if (x + j < 0) {

mvFlag = true;

}

//the window extends from 0 to 18, so if it is greater than 18 then it falls outside the boundary of the window. The far corner in the x direction and y

//direction should be maximum 18, and from lecture notes the window is (R + N -1) is the edge R = (Y + i) N = MBSize

else if (y + i + MBSIZE-1 > 18) {

mvFlag = true;

}

else if (x + j + MBSIZE-1 > 18) {

mvFlag = true;

}

//so if the flag isn't triggered, we can proceed with calculation

if (mvFlag = false) {

// computing the SAD of two macro blocks

sadTemp = GetSAD(pfCurrFrame, pfRefFrame, y, x, i, j, iSAD0);

//only update MV when it's sad is less than 0.925 of current lowest SAD0

if (sadTemp < 0.925\*iSAD0) {

iSAD0 = sadTemp;

//keep track of best candidate with iMVy and iMVx

iMVx = j;

iMVy = i;

}

}

mvFlag = false; // reset the flag to false for the next iteration

}

}

m\_iMVy[y / MBSIZE][x / MBSIZE] = iMVy;

m\_iMVx[y / MBSIZE][x / MBSIZE] = iMVx;

}

//return the sum of absolute difference (SAD) of two blocks with the given motion vectors

int IEncoder::GetSAD(

float \*\*pfCurrFrame, //pointer to the current frame,

float \*\*pfRefFrame, //pointer to the reference frame,

int y, //(y, x) is the upper-left corner of the current block

int x,

int iMVy, //(iMVy, iMVx) is the given MV

int iMVx,

int iSAD0) //Minimum SAD so far, used for early termination.

{

int iSAD = 0;

//------------------------

//HW5: compute SAD, compare with the input SAD0 for early termination.

//------------------------

int diff = 0;

for (int i = 0; i < MBSIZE; i++) {

for (int j = 0; j < MBSIZE; j++) {

//following lecture notes 8.2

diff = pfCurrFrame[x + i, y + j] - pfRefFrame[(x + iMVx + i), (y + iMVy + j)];

//takes absolute value of diff if it's negative

if (diff < 0) {

diff = diff\*-1;

}

//sums the differenence into iSAD

iSAD = diff + iSAD;

//early termination if SAD is larger than the minimum SAD, early termination

if (iSAD >= iSAD0) {

return iSAD;

}

}

}

return iSAD;

}

Initial calculations before editing the code:

Qstep size 5:

Average bit rate = 3.5422

Average Y-PSNR = 19.2151

Qstep size 10:

Average bit rate = 2.453

Average Y-PSNR = 21.5057

Qstep size 20:

Average bit rate = 1.573

Average Y-PSNR = 17.9172

Qstep size 30:

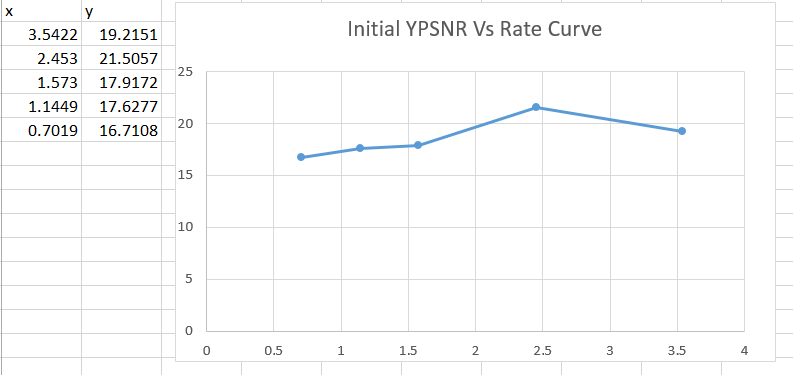
Average bit rate = 1.1449

Average Y-PSNR = 17.6277

Qstep size 40:

Average bit rate = 0.7019

Average Y-PSNR = 16.7108



Calculations after modifying the code:

Qstep size 5:

Average bit rate = 3.5422

Average Y-PSNR = 45.0792

Qstep size 10:

Average bit rate = 2.453

Average Y-PSNR = 39.5492

Qstep size 20:

Average bit rate = 1.573

Average Y-PSNR = 34.1662

Qstep size 30:

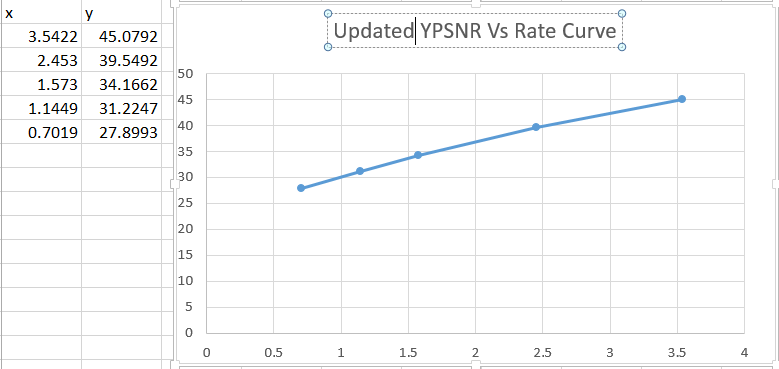
Average bit rate = 1.1449

Average Y-PSNR = 31.2247

Qstep size 40:

Average bit rate = 0.7019

Average Y-PSNR = 27.8993



Which method gives better result?

The updated code, with SAD and motion estimation updated code provides a higher Y-PSNR than the initial code.