Version 2.4 of ICSP

Codec

ZHENG JUN Master's Student

ICSP Lab. Hanyang Univ.

Jechang Jeong, Prof.

Table of Contents

Introduction

Codecs are compression technologies and have two components, an encoder to compress the files, and a decoder to decompress.
The main purpose of this document is descriptions for encoder and decoder.

General description for ICSP codec design

2.1 Block coding order

In this document, we just deal with 4:2:0 formatted sequence for encoder and decoder descriptions. 4:2:2 and 4:4:4 formats can be generated by 4:2:0 format sequence using interpolation.

4:2:0 format does not mean that there are only Y, Cb, and no Cr components. It means that for each scan line, only one chrominance component is stored at a 2:1 sampling rate. Adjacent scan lines store different chrominance components, that is, if one line is 4:2:0, the next line is 4:0:2

Fig. 1 shows coding order of 4:2:0 format. As you can see, Pack the Y, U, and V components separately and store them in sequence. The YUV data extraction of each pixel follows the extraction method of the YUV420 format, that is, Y components share a set of U , V. In case of 4:2:0 format, sizes of Cb and Cr are half of luma.

Y1	Y2	Y3	Y4	Y5	Y6	Y7	Y8
Y9	Y10	Y11	Y12	Y13	Y14	Y15	Y16
Y17	Y18	Y19	Y20	Y21	Y22	Y23	Y24
Y25	Y26	Y27	Y28	Y29	Y30	Y31	Y32
U1	U2	U3	U4	U5	U6	U7	U8
V1	V2	V3	V4	V5	V6	V7	V8

Fig. 1. Coding order for 4:2:0 format.

2.2 Separation of YUV components

Fig. 2 shows the code needed to separate yuv components. As Figure 1 shows, the amount of information per pixel is width * height * 1.5 in the video.

```
unsigned char* pic = (unsigned char*)malloc(WIDTH * HEIGHT * 3 / 2); //시작위치

int i, j;
int frame_con;

for (frame_con = 1; frame_con <= frame_size; frame_con++ ) {
    //한 프레임씩 읽어오기

    fread(pic, 1, WIDTH * HEIGHT * 3 / 2, fp);

    unsigned char* arr_y = (unsigned char*)malloc(sizeof(unsigned char) * WIDTH * HEIGHT);          //Y
    unsigned char* arr_u = (unsigned char*)malloc(sizeof(unsigned char) * WIDTH * HEIGHT / 4);      //U
    unsigned char* arr_v = (unsigned char*)malloc(sizeof(unsigned char) * WIDTH * HEIGHT / 4);      //V

    for (i = 0; i < HEIGHT; i++)
    {
        for (j = 0; j < WIDTH; j++)
        {
            arr_y[i * WIDTH + j] = pic[i * WIDTH + j];  // Y 성분을 읽어와서 2중 포인터에 저장
        }
    }

    for (i = 0; i < HEIGHT / 2; i++)
    {
        for (j = 0; j < WIDTH / 2; j++)
        {
            arr_u[i * WIDTH / 2 + j] = pic[HEIGHT * WIDTH + i * WIDTH / 2 + j];               //U

            arr_v[i * WIDTH / 2 + j] = pic[HEIGHT * WIDTH * 5 / 4 + i * WIDTH / 2 + j];       //V
        }
    }
```

Fig. 2. Code for YUV component separation.

Fig. 3 shows the result after separating YUV components.

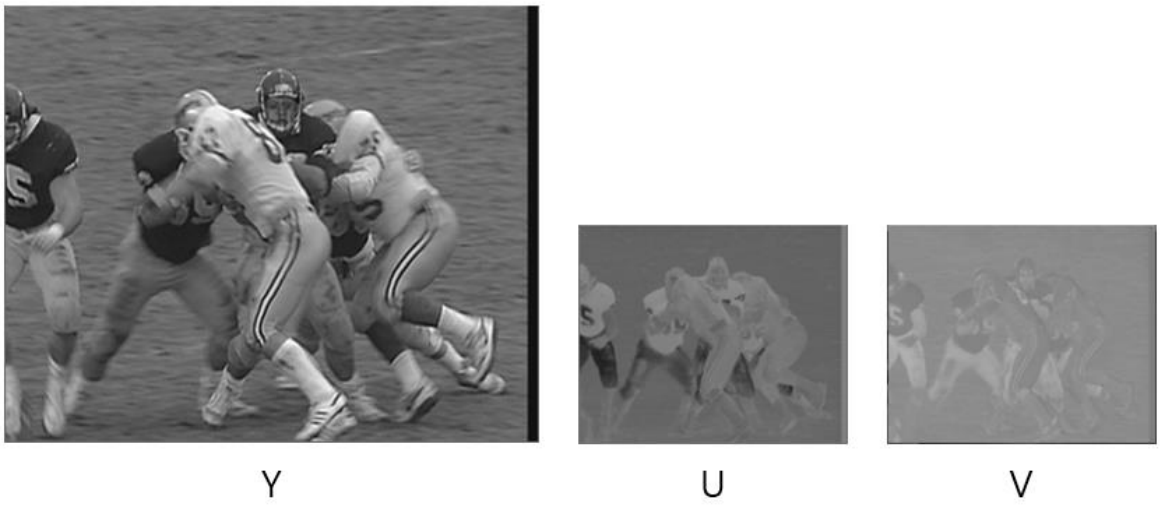


Fig. 3. YUV component

Encoder description

Fig. 4 shows the ICSP encoder structure. T and Q mean transform and quantization each. We use simplified intra 8x8 prediction, Discrete Cosine Transform (DCT) as T and uniform quantization with simple dead zone as Q. Zig-zag scan is used for reordering and entropy coder which is like JPEG's is used. For inter frame, Motion Estimation (ME) and Motion Compensation (MC) are used. Reconstructed frame has to be made using inverse transform and inverse quantization.

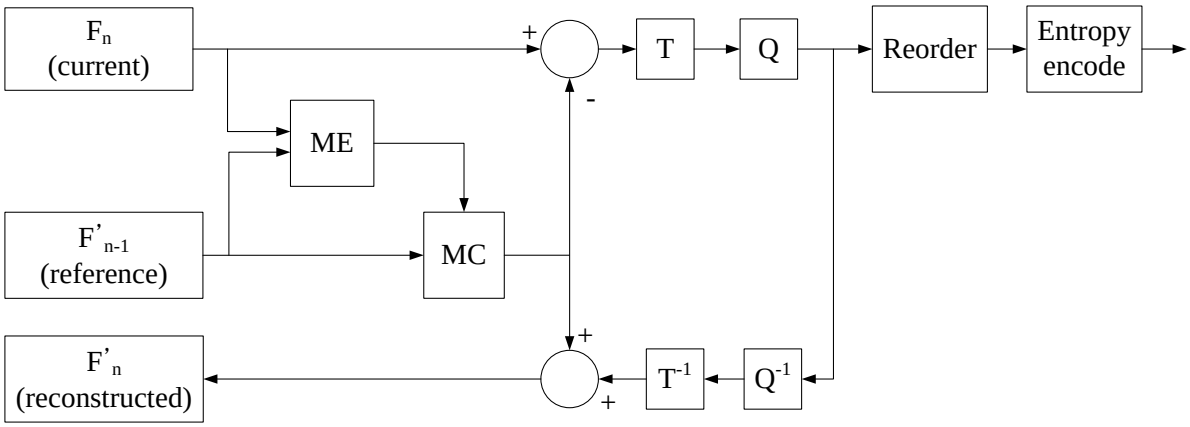


Fig. 4. Block diagram of ICSP encoder.

3.1 Encoding a intra frame

For encoding the intra frame, green units in Fig. 5 are used.

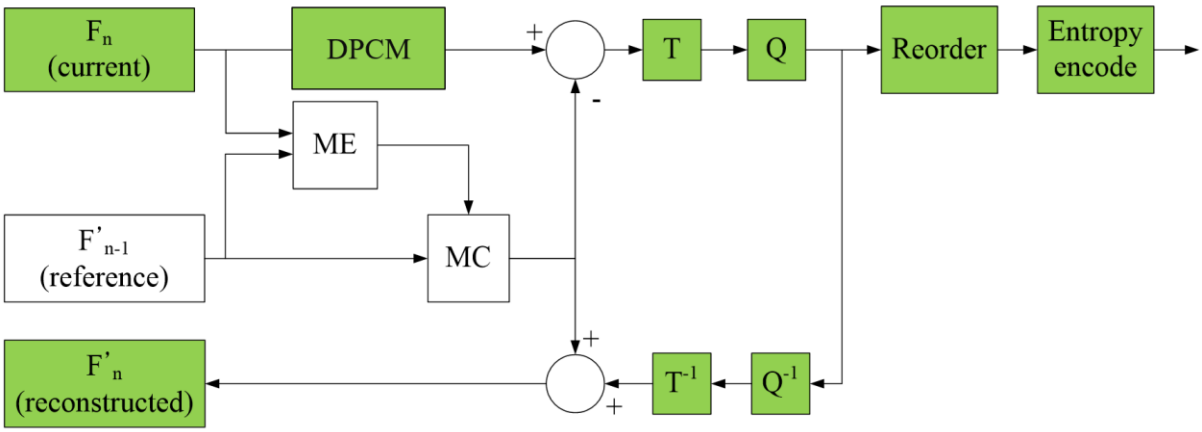


Fig. 5. Block diagram of intra frame.

In case of I frame, first of all, current frame is divided by macro-block. In raster macro-block scan order, after simplified intra 8x8 prediction, pixel based DPCM, Discrete Cosine Transform (DCT), and quantization, we also use DPCM for reconstructed DC component. And then use the reordering (just zig-zag scan) and entropy encoder (like Huffman coding). While doing this forward process, we save the reconstructed data using inverse quantization, inverse transform, inverse DPCM and simplified intra 8x8 prediction for reference of next frame.

3.2 Intra 8x8 prediction

We use simplified intra 8x8 prediction. As shown in Fig. 6, the ICSP codec uses 3 intra prediction modes. Each 8x8 prediction mode generates 64 predicted pixel values using of the upper and left-hand neighboring pixels as show in Fig. 6. Here, if there is no value in the upper and left pixels in the current prediction block, the pixel value has been replaced with 128.

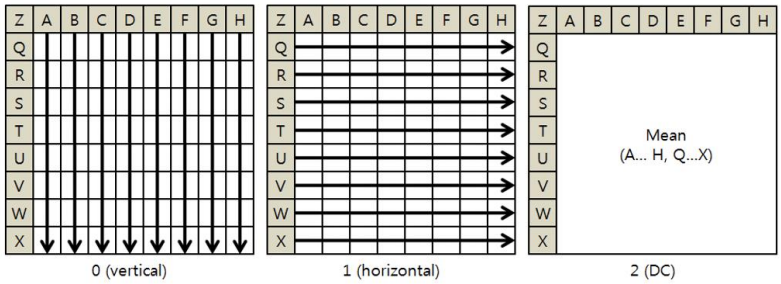


Fig. 6. Three different modes for intra 8x8 prediction

The prediction block created by each of the predictions. The Sum of Absolute Errors (SAE) for each prediction indicates the magnitude of the prediction error. The best matching mode gives the smallest SAE between original block and prediction block. In ICSP codec, only luminance block is predicted.

3.3 Main code of intra prediction

Fig. 7 shows the code that sets the required left and upper pixel values for each block when calculating the intra prediction.

```
for (i = 0; i < h; i += 8) {
    for (j = 0; j < w; j += 8) {
        // 첫번째 블록 위와 좌의 값은 128      if i , j=0
        if ((i == 0) && (j == 0)) {
            for (a = 0; a < 8; a++) {
                temp_left[a] = 128;
                temp_up[a] = 128;
            }
        }

        if ((i == 0) && (j != 0)) {
            for (a = 0; a < 8; a++) {
                temp_up[a] = 128;
                temp_left[a] = input[(i + a) * w + (j - 1)];
            }
        }
        else if ((i != 0) && (j == 0)) {
            for (a = 0; a < 8; a++) {
                temp_left[a] = 128;
                temp_up[a] = input[(i - 1) * w + (j + a)];
            }
        }

        else if ((i != 0) && (j != 0)) {
            for (a = 0; a < 8; a++) { //블록단위
                temp_up[a] = input[(i - 1) * w + (j + a)];
                temp_left[a] = input[(i + a) * w + (j - 1)];
            }
        }
```

Fig. 7. Code for setting the left and upper pixel

Fig. 8 shows the code for calculating the SAE for each mode.

```
// SAE 계산
int sae0 = 0, sae1 = 0, sae2 = 0;

//      intra 0모드
for (a = 0; a < 8; a++) { //블록단위
    for (b = 0; b < 8; b++) {
        intra0[a][b] = temp_up[b] - input[(i + a) * w + (j + b)];
        /*intra0[a][b] = abs(temp_up[b] - input_2d[i + a][j + b]);*/
        sae0 += abs(temp_up[b] - input[(i + a) * w + (j + b)]);
    }
}

//      intra 1모드
for (a = 0; a < 8; a++) { //블록단위
    for (b = 0; b < 8; b++) {
        intra1[a][b] = temp_left[a] - input[(i + a) * w + (j + b)];
        /*intra1[a][b] = abs(temp_left[a] - input_2d[i + a][j + b]);*/
        sae1 += abs(temp_left[a] - input[(i + a) * w + (j + b)]);
    }
}

//      intra 2모드
int sum = 0, ave = 0;

for (x = 0; x < 8; x++) {
    sum += temp_up[x] + temp_left[x];
}
ave = sum / 16;

for (a = 0; a < 8; a++) { //블록단위
    for (b = 0; b < 8; b++) {
        intra2[a][b] = ave - input[(i + a) * w + (j + b)];
        /*intra2[a][b] = abs(ave - input_2d[i + a][j + b]);*/
        sae2 += abs(ave - input[(i + a) * w + (j + b)]);
    }
}
```

Fig. 8. Code for calculating each mode SAE

Then compare the SAE in each mode to find the optimal mode and store the pixel value. Therefore, Fig. 9 shows the result image of intra prediction.

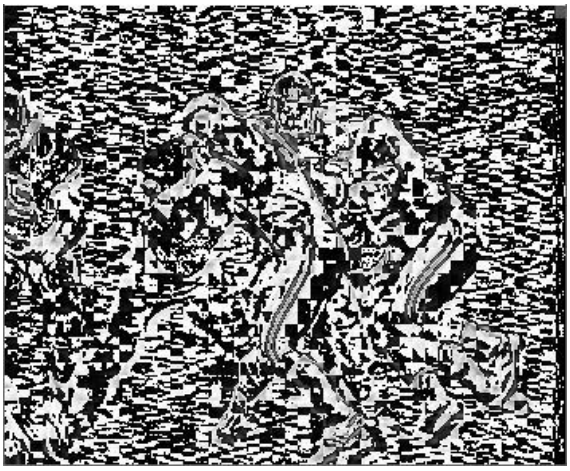


Fig. 9. Result image of intra prediction

3.4 Differential pulse code modulation (DPCM)

Since the current value is often large compared to 0, we predict the current value using neighboring value. We use DPCM for DC prediction and intra pixel prediction. In the two dimensional signal, we can use neighboring pixel values or DC values of neighboring 8x8 blocks. That's why we define the DPCM mode. Fig. 10 shows the neighboring 8x8 block positions or pixel positions which we could use.

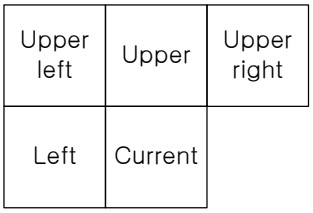


Fig. 10. Positions of the 8x8 block or pixel.

If upper right position is not available, it can be replaced by upper left position. If current DPCM mode is 0, i.e. median prediction mode, predicted value is median value of left, upper, and upper right positions' values (it can be DC values or pixel values). In prediction, if left and upper left positions are not available, then their values are replaced by upper position's value. If upper line is not available, then upper and upper right positions' values are replaced by left position's value. In this report, DPCM mode was used in mode0, mode1, mode2, and mode6.

After DPCM, error value, e, is defined as (1).

$$e = c - p \tag{1}$$

In (1), c and p mean values of current position and predicted value by DPCM mode each.

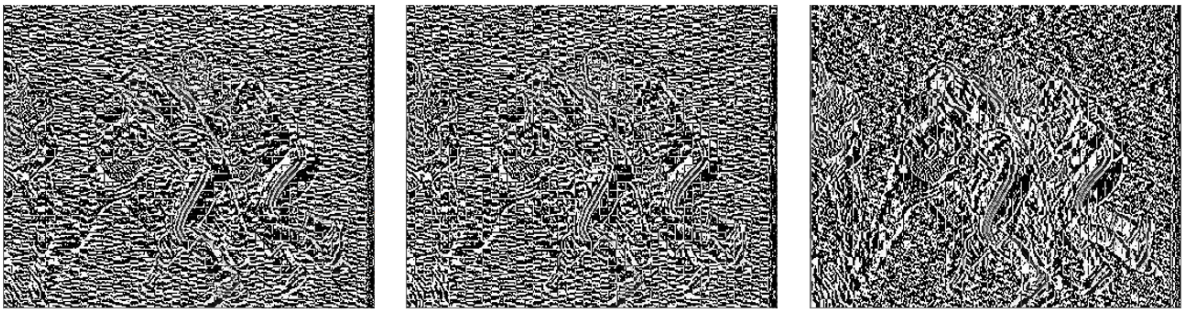
3.5 Main code of pixel DPCM

Pig 11 is the code for calculating the pixel DPCM. where Mode 0 means Median prediction mode, Mode 1 means mean prediction mode, Mode 2 means Left prediction mode, and Mode6 means no prediction. That is, Mode 6 does not use pixel DPCM.

```
if (dpcm_mod == 0) {
    for (i = 0; i < h; i++) {
        for (j = 0; j < w; j++) {
            if ((i == 0) && (j == 0)) {
                median = 128;
            }
            else if (j == 351) {
                median = intra_out_int[i * w + j - 1];
            }
            else if ((i == 0) && (j != 0)) {
                median = intra_out_int[j - 1];
            }
            else if ((i != 0) && (j == 0)) {
                median = intra_out_int[(i - 1) * w + j];
            }
            else if ((i != 0) && (j != 0)) {
                left = intra_out_int[i * w + (j - 1)];
                upper = intra_out_int[(i - 1) * w + j];
                upper_right = intra_out_int[(i - 1) * w + (j + 1)];
                if ((left - upper) * (upper - upper_right) >= 0) {
                    median = upper;
                }
                else if ((upper - left) * (left - upper_right) >= 0) {
                    median = left;
                }
                else if ((upper - upper_right) * (upper_right - left) >= 0) {
                    median = upper_right;
                }
            }
            intra_dpcm_out[i * w + j] = intra_out_int[i * w + j] - median;
            intra_dpcm_int[i * w + j] = intra_out_int[i * w + j] - median;
        }
    }
}
else if (dpcm_mod == 1) {
    for (i = 0; i < h; i++) {
        for (j = 0; j < w; j++) {
            if ((i == 0) && (j == 0)) {
                mean = 128;
            }
            else if ((i == 0) && (j != 0)) {
                mean = intra_out_int[j - 1];
            }
            else if ((i != 0) && (j == 351)) {
                mean = (intra_out_int[i * w + j - 1] * 2 + intra_out_int[(i - 1) * w + j]) / 3;
            }
            else if ((i != 0) && (j == 0)) {
                mean = (intra_out_int[(i - 1) * w] * 2 + intra_out_int[(i - 1) * w + j + 1]) / 3;
            }
            else if ((i != 0) && (j != 0)) {
                left = intra_out_int[i * w + (j - 1)];
                upper = intra_out_int[(i - 1) * w + j];
                upper_right = intra_out_int[(i - 1) * w + (j + 1)];
                mean = (left + upper + upper_right) / 3;
            }
            intra_dpcm_out[i * w + j] = intra_out_int[i * w + j] - mean;
            intra_dpcm_int[i * w + j] = intra_out_int[i * w + j] - mean;
        }
    }
}
else if (dpcm_mod == 2) {
    for (i = 0; i < h; i++) {
        for (j = 0; j < w; j++) {
            if (j == 0) {
                intra_dpcm_out[i * w + j] = intra_out_int[i * w + j] - 128;
                intra_dpcm_int[i * w + j] = intra_out_int[i * w + j] - 128;
            }
            else {
                intra_dpcm_out[i * w + j] = intra_out_int[i * w + j] - intra_out_int[i * w + j - 1];
                intra_dpcm_int[i * w + j] = intra_out_int[i * w + j] - intra_out_int[i * w + j - 1];
            }
        }
    }
}
else if (dpcm_mod == 6) {
    for (i = 0; i < h; i++) {
        for (j = 0; j < w; j++) {
            intra_dpcm_out[i * w + j] = intra_out_int[i * w + j];
            intra_dpcm_int[i * w + j] = intra_out_int[i * w + j];
        }
    }
}
```

Fig. 11. Code for pixel DPCM.

Fig12 is a result image that is output using each mode of the DPCM after intra prediction.



Mode 0 Mode 1 Mode 2

Fig. 12. Result image after intra prediction and pixel DPCM

3.6 Encoding a inter frame

For encoding the inter frame, green units in Fig. 13 are used.

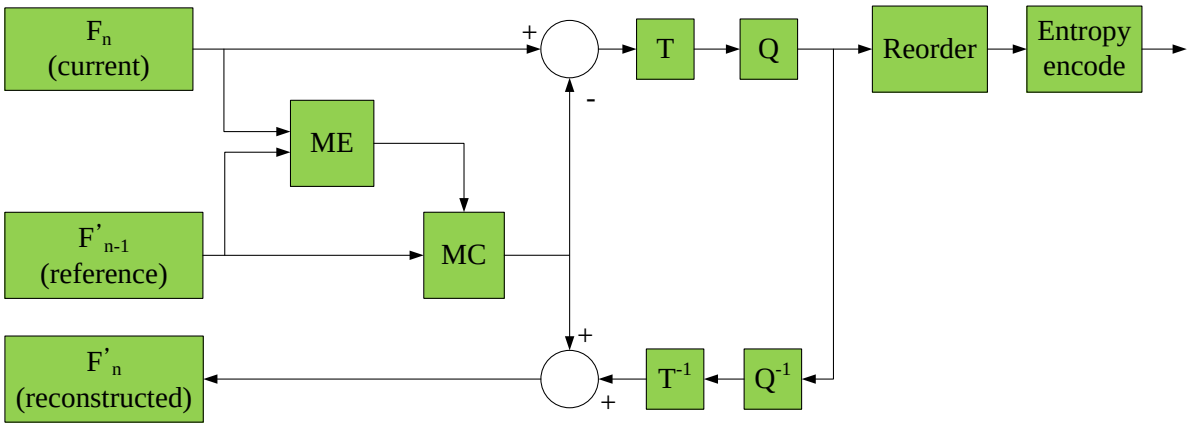


Fig. 13. Block diagram of inter frame.

In case of P frame, using previous reconstructed frame for reference, we use motion estimation to search the best motion vector of current macro-block. After motion estimation, we use DCT, quantization, DPCM for DC, reordering, and entropy coding for error image. We use motion compensation for reconstructed frame. In the inter prediction, we do not use pixel based DPCM.

3.7 Motion estimation and compensation

To reduce the temporal redundancy, the block matching algorithm (BMA) for ME has been widely adopted in many video compression standards, because it is simple to implement and effective at decreasing temporal redundancy. In BMA, the current frame is first divided into macro-blocks, which are fixed-sized square blocks, and the motion vector for each macro-block is estimated by finding the most similar block of pixels within its search range in a reference frame according to some matching criteria. The sum of absolute difference (SAD) is often used as a matching criterion between the current macro-block and the candidate blocks because of its lower complexity compared with other matching criteria such as the sum of squared difference (SSD) or the mean square error (MSE). The SAD between a current macro-block and a block of a candidate position within the search range of the reference frame at a relative coordinate (x, y) is defined as (2).

$$SAD(x, y) = \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \left| f_{cur}(i, j) - f_{ref}(i + x, j + y) \right| \tag{2}$$

where N is 8, $f_{cur}(i, j)$ and $f_{ref}(i, j)$ denote the pixel intensity of the current macro-block and the block of the candidate position, respectively.

In this document, we will implement the FSA for 8x8 modes each. The best-matched block can be obtained by the FSA, which entails a check of all candidate positions in the search range. Namely, mode which has minimum SAD will be selected as best motion estimation mode. Here, set the search range to 17x17.

In the inter prediction process, because there is not enough information about the FSA, it is padded into adjacent pixels. As shown in Fig. 14, the part without information receives a pixel value of the adjacent border part. The remaining pixels without a value equal the pixel value at the adjacent corner position.

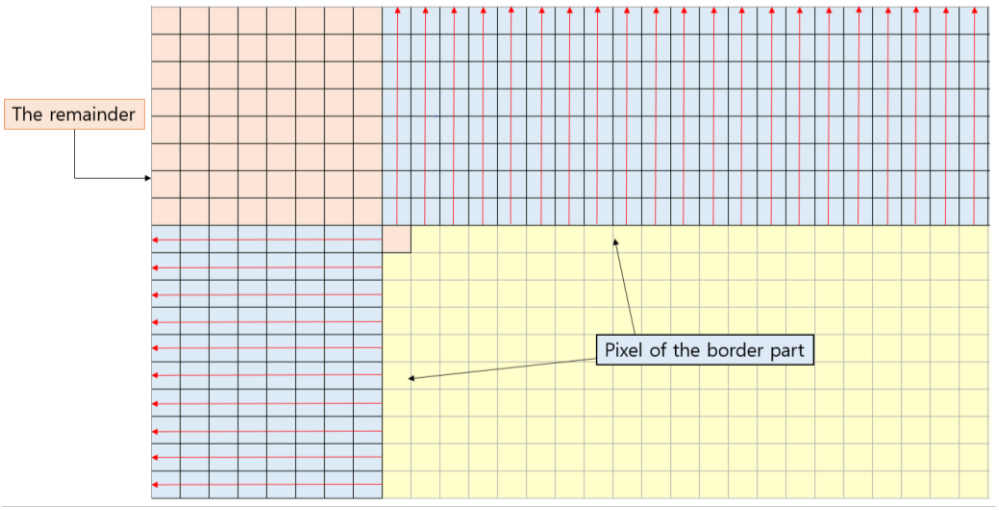


Fig. 14. Padding method

During FSA, matching blocks are found through spiral search. The search direction is shown in Figure 15.

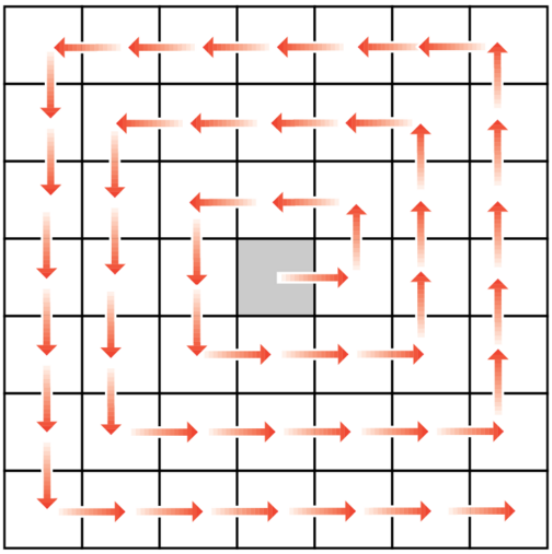


Fig. 15. Spiral search pattern for block matching

3.8 Main code for inter prediction

Fig. 16 shows the main code of inter prediction. Set INTER_BLOCK_SIZE to 8 according to equations (3) and calculate the SAD for each block. Then find the smallest SAD and save motion vectors. cur_arr is the Nth image (i.e, the current frame image), and ref_arr is the N-1 image (i.e, the reference frame image).

```
for (a = a_val; a <= a_max; a++) {
    for (b = b_val; b <= b_max; b++) {

        for (c = 0; c < INTER_BLOCK_SIZE; c++) {
            for (d = 0; d < INTER_BLOCK_SIZE; d++) {
                sad[(a + SEARCH_INDEX) * SEARCH_RANGE + (b + SEARCH_INDEX)] += abs(cur_arr[(i + c) * w + (j + d)] - ref_arr[(i + a + c) * w + (j + b + d)]);
            }
        }
        if ((a == a_val) && (b == b_val)) {
            sad_min = sad[(a_val + SEARCH_INDEX) * SEARCH_RANGE + (b_val + SEARCH_INDEX)];
            MV_x = -b;
            MV_y = -a;
        }

        else if (sad[(a + SEARCH_INDEX) * SEARCH_RANGE + (b + SEARCH_INDEX)] < sad_min) {
            sad_min = sad[(a + SEARCH_INDEX) * SEARCH_RANGE + (b + SEARCH_INDEX)];
            MV_x = -b;
            MV_y = -a;
        }
    }
}
for (c = 0; c < INTER_BLOCK_SIZE; c++) {
    for (d = 0; d < INTER_BLOCK_SIZE; d++) {
        inter_out_int[(i + c) * w + (j + d)] = cur_arr[(i + c) * w + (j + d)] - ref_arr[(i - MV_y + c) * w + (j - MV_x + d)];
        inter_out[(i + c) * w + (j + d)] = ref_arr[(i - MV_y + c) * w + (j - MV_x + d)];
    }
}
MV[((i / INTER_BLOCK_SIZE) * (w / INTER_BLOCK_SIZE) + j / INTER_BLOCK_SIZE) * 2] = MV_x;
MV[((i / INTER_BLOCK_SIZE) * (w / INTER_BLOCK_SIZE) + j / INTER_BLOCK_SIZE) * 2 + 1] = MV_y;
```

Fig. 16. Main code for inter prediction.

Fig. 17 is a motion compensation frame. Motion compensation frames were obtained by performing motion estimation for all macro blocks of the current frame.

Fig. 17. Motion compensation frame.

Fig. 18 shows the image of the inter prediction error. That is, the difference between the current frame and the motion compensation frame.

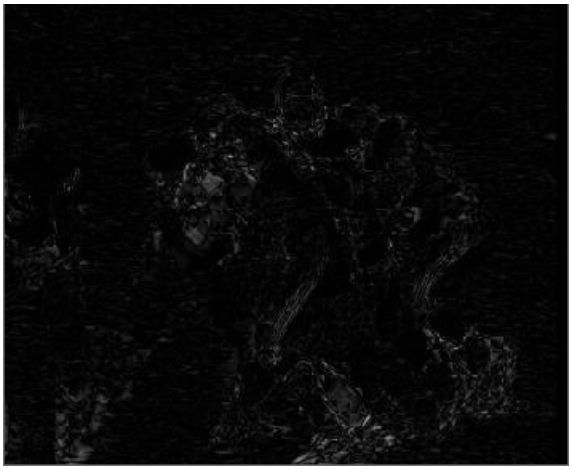


Fig. 18. The inter prediction error.

3.9 Transform

We use 8x8 block size DCT as transform. The following equations (3) and (4) specify the ideal forward DCT and backward DCT each. But these equations contain cosine terms. In our ICSP codec, we use "double" type instead of "float" type for cosine accuracy.

$$S(v,u) = \frac{1}{4} C_u C_v \sum_{x=0}^{7} \sum_{y=0}^{7} s(y,x) \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16} \tag{3}$$

$$s(y,x) = \frac{1}{4} \sum_{u=0}^{7} \sum_{v=0}^{7} C_u C_v S(v,u) \cos\frac{(2x+1)u\pi}{16} \cos\frac{(2y+1)v\pi}{16} \tag{4}$$

where

$C_u, C_v = 1/\sqrt{2}$ for $u, v = 0$

$C_u, C_v = 1$ otherwise.

For one 8x8 block, we need 4 overlapping loops. It can be very heavy. We can implement equations (3) and (4) as separable two 3 overlapping loops each. In other words, it uses two 1d-dcts.

3.10 Main code for DCT

Fig. 19 sets the transformation matrix of the 8x8 block size DCT. Fig. 20 shows a code that implem ents DCT using two 3 overlapping loops.

```
//  DCT_A[i][j]
for (i = 0; i < 8; i++)
{
    for (j = 0; j < 8; j++)
    {
        if (i == 0)
        {
            cm = sqrt((double)1 / 8); // cu * 2 / 1
        }
        else
        {
            cm = sqrt((double)2 / 8);
        }
        DCT_A[i][j] = cm * cos((2 * j + 1) * PI * i / 16);
    }
}
//  DCT_AT[i][j]
for (i = 0; i < 8; i++)
{
    for (j = 0; j < 8; j++)
    {
        DCT_AT[i][j] = DCT_A[j][i];
    }
}
```

Fig. 19. Code for the transformation matrix

```
for (i = 0; i < h; i += 8)
{
    for (j = 0; j < w; j += 8)
    {
        for (a = 0; a < 8; a++)
        {
            for (b = 0; b < 8; b++)
            {
                temp = 0;
                for (k = 0; k < 8; k++) {
                    temp += DCT_A[a][k] * arr1[(k + i) * w + b + j];
                }
                DCT_Temp[a][b] = temp;
            }
        }

        for (a = 0; a < 8; a++)
        {
            for (b = 0; b < 8; b++)
            {
                temp = 0;
                for (k = 0; k < 8; k++) {
                    temp += DCT_Temp[a][k] * DCT_AT[k][b];
                }

                if ((a == 0) && (b == 0)) {
                    temp = temp / qp_dc;
                    dct_out_double[(a + i) * w + b + j] = temp;
                }
                else {
                    temp = temp / qp_ac;
                    dct_out_double[(a + i) * w + b + j] = temp;
                }
                dct_out_int[(a + i) * w + b + j] = (int)(temp < 0 ? temp - 0.5 : temp + 0.5);
                dct_out[(a + i) * w + b + j] = temp;
            }
        }
```

Fig. 20. Main code for DCT

Fig. 21 shows the result image after DCT.

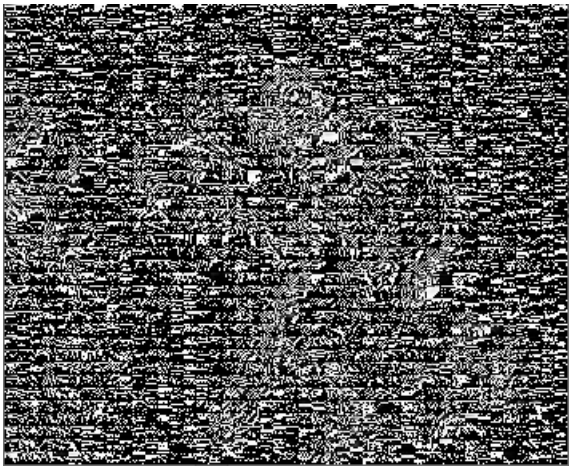


Fig. 21. Result image after DCT.

After DCT, ACs mostly become zero. Therefore, AC flag is used to increase the compression effect. If ACs in current 8x8 block are all zero for intra and inter predictions, then AC flag is '1', and we skip encoding ACs in current 8x8 block. If ACs in current 8x8 block are not all zero, then AC flag is '0', and encoding all ACs. Fig. 22 shows the code implementing AC flag.

```
int count = 0;
AC_Flag[(i / 8) * (w / 8) + (j / 8)] = 0;
for (a = 0; a < 8; a++)
{
    for (b = 0; b < 8; b++)
    {
        if (!((a == 0) && (b == 0)) && (dct_out_int[(a + i) * w + b + j] == 0)) {
            count++;
        }
        if (count == 63) {
            AC_Flag[(i / 8) * (w / 8) + (j / 8)] = 1;
        }
    }
}
```

Fig. 22. Code implementing AC flag

3.11 Quantization

We just use general quantization. There is only one thing. Be careful with round off. Forward quantization is defined as (5).

$$Sq(v,u) = round\left(\frac{S(v,u)}{Qstep}\right) \tag{5}$$

In (5), $Sq(v,u)$ is the quantized DCT coefficient, and $Qstep$ is the quantization step or quantization parameter. $S(v,u)$ is the DCT coefficient. Inverse quantization is defined as (6).

$$R(v,u) = Sq(v,u) \times Qstep \tag{6}$$

In (6), $R(v,u)$ is the inverse quantized DCT coefficients. We use DC $Qstep$ for DC coefficient and AC $Qstep$ for AC coefficients as $Qstep$.

Be sure that if 0.5 is rounded off, it becomes 1. But if -0.5 is rounded off, it becomes 0 not -1. As shown in Fig. 23, the quantization uses rounding, which inevitably leads to losses. Fig. 24 shows an image that only quantization.

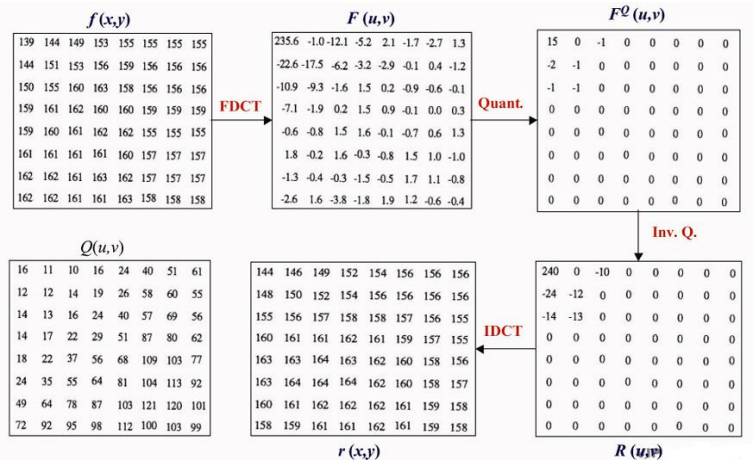


Fig. 23 Example of quantization

Fig. 24. Only Quantization (QP=16)

3.12 Reordering

We use zig-zag scan to increase a 0 probability sequencely. Zig-zag order of quantized DCT coefficients is defined as Fig. 25.

0	1	5	6	14	15	27	28
2	4	7	13	16	26	29	42
3	8	12	17	25	30	41	43
9	11	18	24	31	40	44	53
10	19	23	32	39	45	52	54
20	22	33	38	46	51	55	60
21	34	37	47	50	56	59	61
35	36	48	49	57	58	62	63

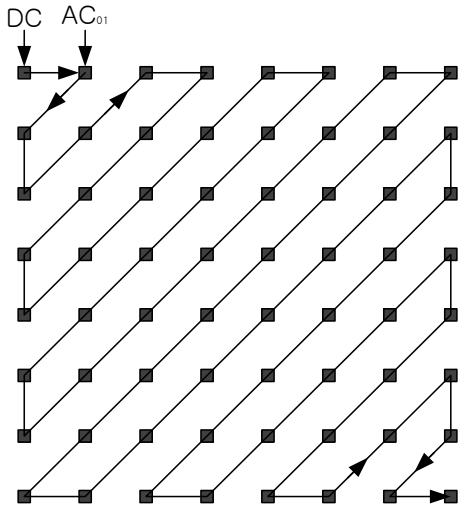


Fig. 25. Zig-zag order of quantized DCT coefficients.

3.13 Main code for Reorder

Fig. 26 shows the code implementing the zig-zag order.

```
for (int idx = 0, d = 1; idx <= 2 * (N - 1); idx++, d *= (-1))
{
    if (idx <= N - 1)
    {
        if (d == 1) // 짝수 대각선
        {
            y = idx;
            x = idx - y;
        }
        else // 홀수 대각선
        {
            x = idx;
            y = idx - x;
        }
    }
    else
    {
        if (d == 1) // 짝수 대각선
        {
            x = idx - (N - 1);
            y = idx - x;
        }
        else // 홀수 대각선
        {
            y = idx - (N - 1);
            x = idx - y;
        }
    }

    for (; x >= 0 && y >= 0 && x < N && y < N; x = x + d, y = y - d)
    {
        if (ind < w * h) {
            temp[ind++] = zigzag_input[(i + y) * w + (j + x)];
        }
    }
}

for (a = 0; a < N; a++) {
    for (b = 0; b < N; b++) {
        zigzag[(i + a) * w + (j + b)] = temp[a * N + b];
        zigzag_int[(i + a) * w + (j + b)] = temp[a * N + b];
    }
}
```

Fig. 26. Main code for Reorder

Fig. 27 shows the image printed using only the reorder.

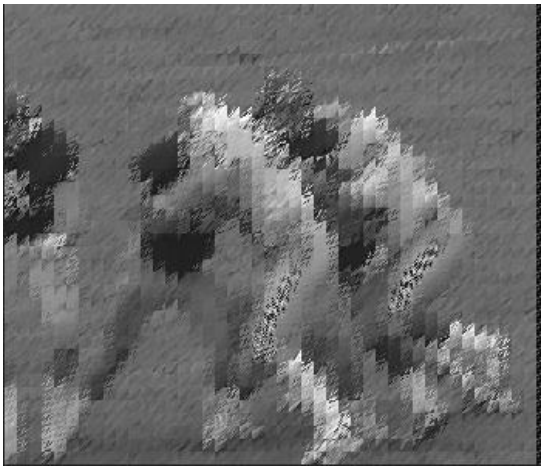


Fig. 27. Reorder Only

3.14 Entropy coding

With table I, we implement the entropy coding for motion vector, quantized DC after DPCM, and quantized ACs. It is very simple and looks like JPEG's table. Sign bit is 0 in case of negative value and 1 in case of positive value.

Table. 1. Entropy code for ICSP codec.

Category	Code word	Range	Bit form Code word + Sign + Range	Total data length
0	00	0	00	2
1	010	1	010 x	4
2	011	2,3	011 x 0	5
3	100	4,…,7	100 x 00	6
4	101	8,…,15	101 x 000	7
5	110	16,…,31	110 x 0000	8
6	1110	32,…,63	1110 x 00000	10
7	11110	64,…,127	11110 x 000000	12
8	111110	128,…,255	111110 x 0000000	14
9	1111110	256,…,511	1111110 x 00000000	16
10	11111110	512,…,1023	11111110 x 000000000	18
11	111111110	1024,…,2047	111111110 x 0000000000	20

Assume that the value to be encoded is -3. The value -3 belongs to the category 2. So code word of -3 is '011'. Sign bit is '0' because the sign of -3 is negative. Finally, range of 3 becomes '1' because it is after 2. Consequently, -3 can be represented by '01101'. Another example is 18. The value 18 belongs to the category 5. And code word of 18 is '110'. Sign bit is '1'. Range of 18 is '0010'. So 18 can be represented by '11010010'.

3.15 Main code of Entropy coding

As shown in Fig. 28, the size variable means the number of bits required to display the input value in binary (i.e, the Category in Table 1). Then, set the code word for each step.

```
do {

    if (value == 0) {
        size = 0;
    }
    else {
        value = value / 2;
        size++;
    }
} while (value != 0); // 2진수 생성

switch (size) {
case 0:
    code_word = 0 << 1;
    data_length = 2;
    break; // 00
case 1:
    code_word = 2 << size;
    data_length = 4;
    break; // 010
case 2:
    code_word = 3 << size;
    data_length = 5;
    break; // 011
case 3:
    code_word = 4 << size;
    data_length = 6;
    break; // 100
case 4:
```

Fig. 28. Code word setting code

Fig. 29 shows entropy codes of each number. Using the bit operator, the front part of the entropy is code word. If the input is a positive number, put its bit behind the code word, if it is negative number, put its bit behind the code word using bitwise NOT

```
if (temp[con] > 0) { // temp -> zigzag후 배열
    temp_cur = code_word ^ temp[con];
}
else {
    ne_code = (0xffffffff << size) ^ (~abs(temp[con])); // 정수 반대
    temp_cur = code_word ^ ne_code;
}
```

Fig. 29. entropy codes of each number

If you set each number of entropy codes, you need to connect them. In this report, we saved the entropy code in the integer data type. The code being implemented is shown in Fig. 30. Data_length is the total length of data that each number requires. If you have saved all 32 bits, you must divide by the last entropy code and save it in the next integer data type.

```
if (total_length > 32) { // 쌓인 비트수가 32비트를 넘는 경우
    back_f = 0;
    data_length = data_length - (total_length - 32); // 마지막 출력해야 할 bit 수
    entropy_cd = (entropy_cd << data_length) ^ (temp_cur >> (total_length - 32));
    entropy_result[entropy_index++] = entropy_cd;
    for (int a = 0; a < total_length - 32; a++) {
        back_f += pow(2.0, a);
    }
    entropy_cd = back_f & temp_cur; // 32bit를 저장하고 마지막 entorpy_cd의 뒤에 있는 bit 값
    total_length = total_length - 32;
    entropy_result[entropy_index] = entropy_cd; // 남는 부분을 다음 배열(entropy_result) 값에 저장
}
else { // 32비트 이내의 경우
    entropy_cd = (entropy_cd << data_length) ^ temp_cur;
    entropy_result[entropy_index] = entropy_cd;

}
```

Fig. 29. Code for storing entropy code

Decoder description

Generally, encoder contains decoder. Therefore, decoder is very simple. The code is almost the sa me. This chapter just shows that there are many different parts.

4.1 Decoder form

Fig. 30 shows the structure of ICSP decoder.

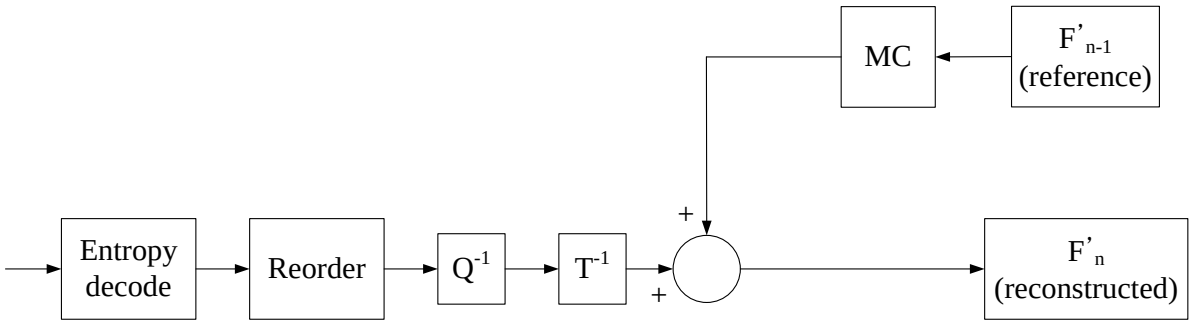


Fig. 30. Block diagram of ICSP decoder.

As you know, Fig. 4 contains Fig. 30.

4.2 Code for Entropy Decoding

Figure 31 shows Code for Entropy Decoding. Recognizing Code word means the beginning of a number. And use the bit right behind the code word to determine whether the number is positive or negative. The results can then be printed using the remaining bits.

The entropy code may be cut off , Because the entropy code uses an integer type when encoding. In such cases, add the code of the next data type and perform a decode.

```
if (((aa >> (31 - total_length)) & 1) == 0) {
    if (((aa >> (31 - total_length - 1)) & 1) == 0) { // 00
        data_length = 2;
        category = 0;
        out[index++] = 0;
    }
    else if (((aa >> (31 - total_length - 1)) & 1) == 1) {
        if (((aa >> (31 - total_length - 2)) & 1) == 0) { // 010
            category = 1;
            for (int a = category - 1; a >= 0; a--) {
                temp += pow(2.0, a);
            }

            if (((aa >> (31 - total_length - 3)) & 1) == 1) {
                out[index++] = (aa >> (32 - total_length - 3 - category)) & temp;
            }
            else {
                out[index++] = -(~((aa >> (32 - total_length - 3 - category)) & temp) & temp);
            }
            data_length = 4;
        }
```

Fig. 31. Code for Entropy Decode.

4.3 Syntax

Stream header is defined as Fig. 32. In intra period, 0 means all predicted frames use intra prediction, and intra prediction flag indicates whether or not an intra prediction is used (Mode 0/1: Disable / Enable). If intra prediction flag is Disable, it means you only use DPCM.

8 (256)	8 (256)	8 (256)	8 (256)	8 (256)
00000000	I	C	S	P
16 (65536)	16 (65536)	8 (256)	8 (256)	4 (16)
Height	Width	QP_DC, 2~32	QP_AC, 2~64	Pixel DPCM mode, 0~6
4 (16)	3 (8)	6 (64)	1(1)	6 (64)
DC DPCM mode, 0~6	MV_Pred Type, 0~5	Intra Period, 0~31	Intra prediction flag, 0 or 1	000000

Fig. 32. Stream header for ICSP codec.

In Fig. 33, (a) and (b) mean intra and inter macro-block header each.

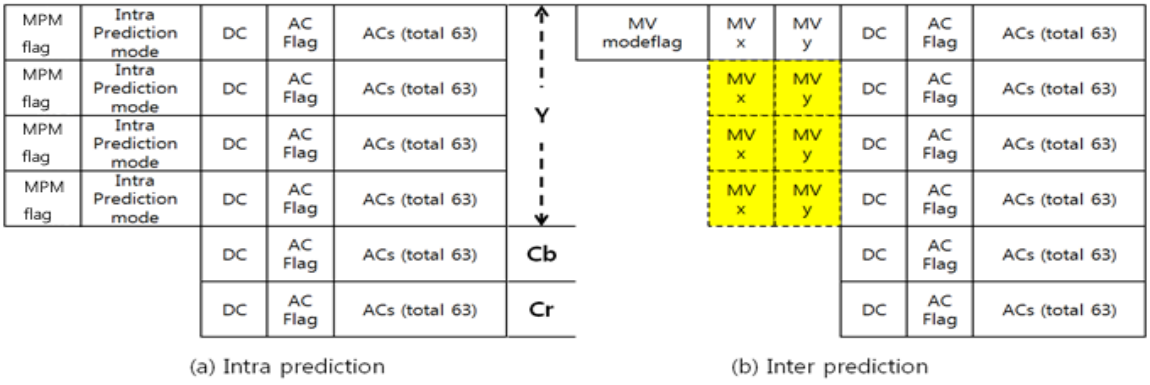


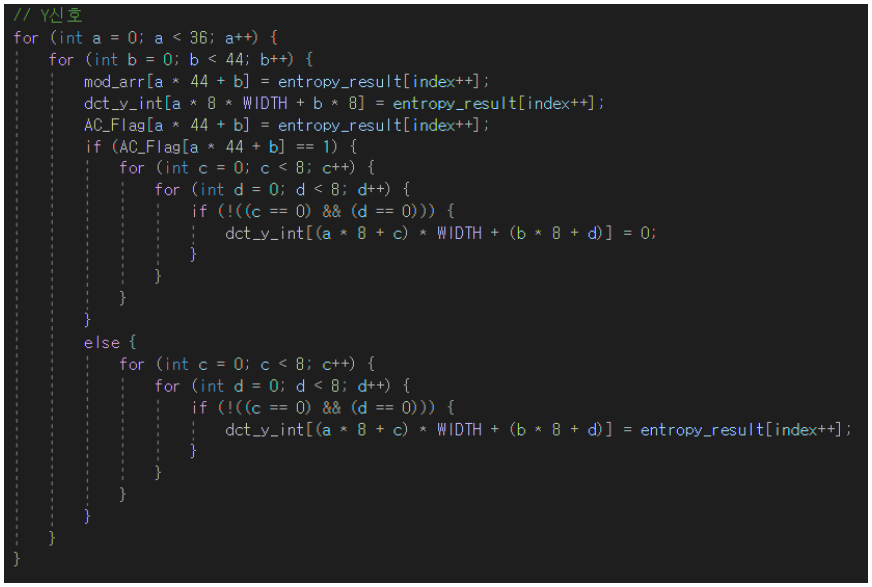
Fig. 33. Macro-block header for intra and inter prediction.

Intra prediction mode is saved during intra prediction. (only luminance block is predicted.)

Store DC and AC separately. If ACs in current 8x8 block are all zero for intra and inter predictions, then AC flag is '1', otherwise AC flag is '0'.

MV means the motion vector used in inter prediction. MV for chroma is just half of luma's MV.

Figure 34 is a code implementation as described above.



```
// Y신호
for (int a = 0; a < 36; a++) {
    for (int b = 0; b < 44; b++) {
        mod_arr[a * 44 + b] = entropy_result[index++];
        dct_y_int[a * 8 * WIDTH + b * 8] = entropy_result[index++];
        AC_Flag[a * 44 + b] = entropy_result[index++];
        if (AC_Flag[a * 44 + b] == 1) {
            for (int c = 0; c < 8; c++) {
                for (int d = 0; d < 8; d++) {
                    if (!((c == 0) && (d == 0))) {
                        dct_y_int[(a * 8 + c) * WIDTH + (b * 8 + d)] = 0;
                    }
                }
            }
        }
        else {
            for (int c = 0; c < 8; c++) {
                for (int d = 0; d < 8; d++) {
                    if (!((c == 0) && (d == 0))) {
                        dct_y_int[(a * 8 + c) * WIDTH + (b * 8 + d)] = entropy_result[index++];
                    }
                }
            }
        }
    }
}
```

Fig. 34. Code for Syntax.

YUV video PSNR calculation method

$$MSE = \frac{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I_Y(i,j)-K_Y(i,j)]^2+\sum_{i=0}^{\frac{m}{2}-1}\sum_{j=0}^{\frac{n}{2}-1}[I_U(i,j)-K_U(i,j)]^2+\sum_{i=0}^{\frac{m}{2}-1}\sum_{j=0}^{\frac{n}{2}-1}[I_V(i,j)-K_V(i,j)]^2}{m\times n\times 1.5} \quad (7)$$

$$MSE = \frac{\sum_{i=0}^{m-1}\sum_{j=0}^{n-1}[I_Y(i,j)-K_Y(i,j)]^2}{m\times n} + \frac{\sum_{i=0}^{\frac{m}{2}-1}\sum_{j=0}^{\frac{n}{2}-1}[I_U(i,j)-K_U(i,j)]^2}{\frac{m}{2}\times\frac{n}{2}} + \frac{\sum_{i=0}^{\frac{m}{2}-1}\sum_{j=0}^{\frac{n}{2}-1}[I_V(i,j)-K_V(i,j)]^2}{\frac{m}{2}\times\frac{n}{2}} \quad (8)$$

$$PSNR = 10 \cdot log_{10}(\frac{{MAX_I}^2}{MSE}) \quad (9)$$

PSNR is most easily defined via the mean squared error (MSE). Equation (7) and Equation (8) are computational expressions of MSE. Equation (7) is averaged by adding the squared error of the three components Y, U and V, and equation (8) is the sum of the MSEs of the three components Y, U and V. It seems similar, but Equation (7) does not mean that the image size is $m \times n$, it means $m \times n \times 1.5$. Therefore, I think it is correct to use equation (8). In addition, the mean PSNR is not calculated by summing the MSEs of all frames and applying to Equation (9). I think it is correct to calculate the MSE for each frame and apply it to equation (9) to find the PSNR and then calculate the mean PSNR.

Experiment Results

Table. 2. Results using quantization only

Sequence (frames)	QP_DC	QP_AC	Pixel DPCM Mode	Intra Period	Intra Prediction flag	Compressibility(%)	PSNR(dB)
Akiyo(300)	1	1	6	0	0	51.836	53.986
	8	1				49.521	46.897
	16	1				48.792	42.449
	1	16				21.684	38.414
	8	16				19.369	38.155
	16	16				18.640	37.237
Foreman(300)	1	1				61.378	53.943
	8	1				59.051	47.140
	16	1				58.282	42.796
	1	16				27.266	35.891
	8	16				24.939	35.761
	16	16				24.169	35.246
Football(90)	1	1				56.372	53.980
	8	1				54.034	47.087
	16	1				53.262	42.819
	1	16				27.770	36.887
	8	16				25.432	36.693
	16	16				24.660	36.055

Table. 2 shows the output results according to the different quantization step after using DCT. According to the analysis of Table 2, the larger the quantization step, the better the compression effect. But the restoration video breaks a lot.

Table. 3. Results of using quantization only for DC

Sequence (frames)	QP_DC	QP_AC	Pixel DPCM Mode	Intra Period	Intra Prediction flag	Compressibility(%)	PSNR(dB)
Akiyo(300)	1	1	6	0	0	51.836	53.986
	8					49.521	46.897
	16					48.792	42.449
Foreman(300)	1					61.378	53.943
	8					59.051	47.140
	16					58.282	42.796
Football(90)	1					56.372	53.980
	8					54.034	47.087
	16					53.262	42.819

Table 3 is the result after quantization only the DC part. As shown in Table 3, the three images have the best compression efficiency of "Akiyo" under the same conditions, and the second is the "football", followed by the "Foreman". The reason is that when comparing the three videos, the "Akiyo" is overall dark and the "Football" is relatively dark, but the "Foreman" is overall bright.

Therefore, a bright image has a higher Y value and a dark image has a smaller Y value. So, the brightest "Foreman" has the lowest compression efficiency, and the "Akiyo" has the best compression efficiency.

Table. 4. Results of using quantization only for AC

Sequence (frames)	QP_DC	QP_AC	Pixel DPCM Mode	Intra Period	Intra Prediction flag	Compressibility(%)	PSNR(dB)
Akiyo(300)	1	1	6	0	0	51.836	53.986
		16				21.684	38.414
Foreman(300)		1				61.378	53.945
		16				24.169	35.246
Football(90)		1				56.372	53.980
		16				24.660	36.055

Table 4 is the result after quantization only the AC part. As shown here, the use of quantization in the ac part greatly affects the compression rate.

Table. 5. Results using Pixel DPCM only

Sequence (frames)	QP_DC	QP_AC	Pixel DPCM Mode	Intra Period	Intra Prediction flag	Compressibility(%)	PSNR(dB)
Akiyo(300)	1	1	0	0	0	48.768	48.645
Foreman(300)						58.170	48.562
Football(90)						53.586	48.517

Table. 5 shows the analysis of each pixel DPCM Mode using only the pixel DPCM. The best mode for PSNR is mode 6 (i.e. no DPCM). The compression effect of mode0 and mode2 is good, but the restoration image is broken. The reason for this is that data loss occurs even if the quantification step is 1 when executing IDCT of Decode. And more loss occurs because it does inverse DPCM with lost data.

Table. 6. Results using Inter prediction

Sequence (frames)	QP_DC	QP_AC	Pixel DPCM Mode	Intra Period	Intra Prediction flag	Compressibility(%)	PSNR(dB)
Akiyo(300)	1	1	6	0	1	48.075	53.959
				10		29.948	54.344
Foreman(300)				0		57.440	53.943
				10		53.662	53.933
Football(90)				0		52.543	53.980
				10		54.816	53.950

In Table 6, Intra Period means the number of frames between intra frames. When Intra Period is 0, all frames are intra frame. And when Intra Period is 10, the frame between intra frames is inter frame.

According to Table 6, using inter prediction is advantageous for restoring videos. And it will have a better compression effect on videos with almost unchanged backgrounds.

Therefore, the compression effect of 'Akiyo' using the inter is the best. The cause of this is that the background of the 'Akiyo' image is stopped and only the face moves, so it compresses a lot during the inter prediction process.

As shown here, if you use inter prediction, the compression effect of "Akiyo" is best, the second is "Foreman", and the last is "Football". However, looking at the results in Tables 5 and 6, when using intra prediction or Pixel DPCM, the compression effect of football is better than that of foreman. This is because both intra prediction and Pixel DPCM perform prediction within a frame. In other words, the value of the part where there is a lot of color difference increases and thus affects the compression rate.

Fig. 36, Fig. 37, and Fig. 38 are distribution diagrams of the number of entropy codes used in each frame when three videos use only intra prediction. As shown in the picture, the "Akiyo" image had about 70,000 entropy codes used by all frames. So, it has the lowest code compared to other videos.

When comparing "Foreman" and "Football" videos, most of the two videos have about 80,000 entropy codes. However, in the remaining part, "Foreman" video occupies 27% of entropy code, which is about 90,000, and "Football" video occupies only 4%. So, the compression effect of "Football" video is better

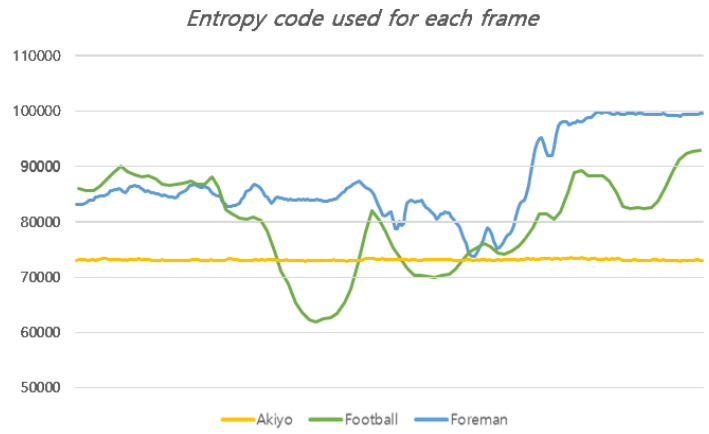


Fig. 35 Distribution of entropy code size used by each frame

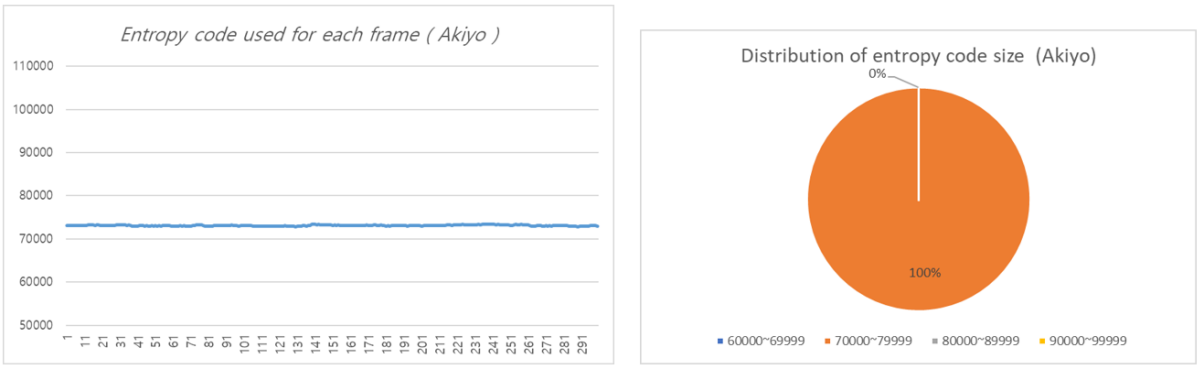


Fig. 36 Distribution of entropy code size used by each frame (Akiyo)

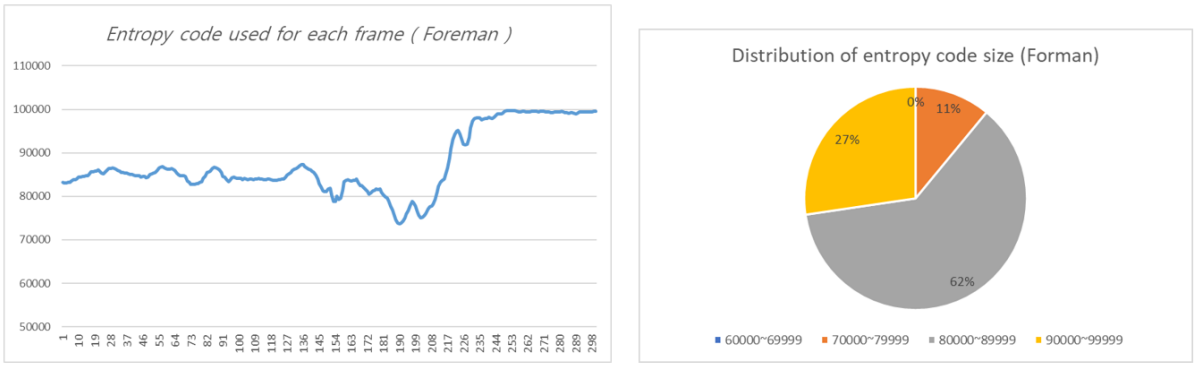


Fig. 37 Distribution of entropy code size used by each frame (Foreman)

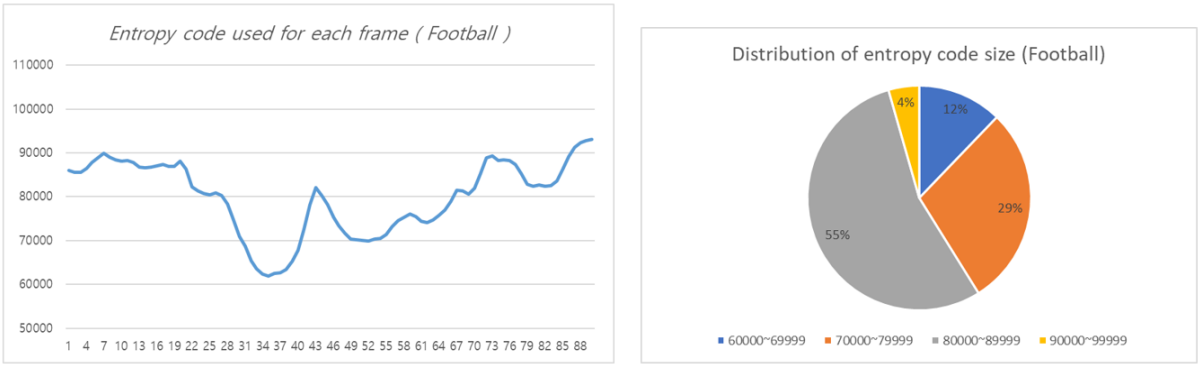


Fig. 38 Distribution of entropy code size used by each frame (Football)

Since the "Akiyo" video hardly moves, the number of entropy codes does not change significantly. The reason why there are many frames with more than 90,000 entropy codes in the "Foreman" video is that there are many frames with many different colors at the end, as shown in Figure 39. And "Football" video is often blurred due to camera shake, as shown in Figure 40. So there is not much color difference, so the number of entropy codes is small.

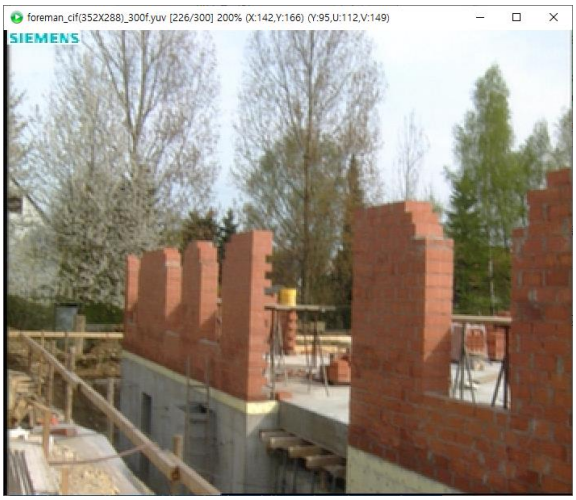


Fig. 39 Forman video

Fig. 40 Football video

Table. 7. According to the result of intra use or not

Sequence (frames)	QP_DC	QP_AC	Pixel DPCM Mode	Intra Period	Intra Prediction flag	Compressibility(%)	PSNR(dB)
Akiyo(300)	1	1	6	0	0	51.836	53.986
					1	48.075	53.959
Foreman(300)					0	61.378	53.945
					1	57.440	53.943
Football(90)					0	56.372	53.9801
					1	52.543	53.9800

Table. 7 shows the results based on the use of Intra prediction. Intra prediction affects compression efficiency a lot. However, since intra prediction stores the difference between the neighboring pixel values, PSNR has little effect.

Table. 8. Results of using Quantization and Pixel DPCM

Sequence (frames)	QP_DC	QP_AC	Pixel DPCM Mode	Intra Period	Intra Prediction flag	Compressibility(%)	PSNR(dB)
Akiyo(300)	1	1	0	0	0	48.768	48.645
	8					49.114	39.060
	16					50.810	33.031
	1	16				13.511	31.268
	8					12.900	30.506
	16					15.837	28.811

As shown in Table. 8, when used with quantization and pixel DPCM, the larger the QP_DC, the worse the compression rate. The reason is that the restored value differs greatly from the original value according to the quantization step. Therefore, pixel DPCM with a large difference increases the value. That is why the compression ratio has increased.

Description of Pixel DPCM Error

Fig. 41 Image with incorrect pixel DPCM (Akiyo)

Fig. 41 shows an image using the wrong pixel DPCM method. As shown in Fig. 41, in the restored image, the Y value increases while restoring to the right. The reason is that when encoding, the DPCM prediction value is not calculated from the reconstructed block in the pixel DPCM part, but the original image is predicted as it is. I reconstructed the image with the prediction value obtained from the original image when encoding. However, when restoring images, the difference values are accumulated more and more because the restored DPCM prediction value is calculated. That is why the value of Y component on the right has increased.