CoolCodec
The C++ Class for Compressing and Decompressing Image and Video

Silva, N.M.S
Dept. of Electrical and Electronic Engineering,
Faculty of Engineering, University of Peradeniya
SRI LANKA
malinda.silva@eng.pdn.ac.lk

Abstract - **The Coolcodec is an Image and video encoding codec which can be used to compress and decompress image and video. This is implemented in C++ as a class which can be used by anyone. This is completely an open source project. Anyone can use, modify and re-distribute freely.**

1. INTRODUCTION

The CoolCodec is a c++ implemented class for compress and decompress Image and video files. There exist separate functions for Encode and Decode Image and video, they are public functions which can be used directly. The compressing algorithms are defined private. The source code is open sourced under GPL license and anyone can use, modify or redistribute under GPL license.

2. IMAGE CODING METHODOLOGY

The method used to compress the image is similar to that of JPEG compression. The RAW image is first split into Red, Green, and Blue planes and processed separately.

In each plane the pixels are represented with an 8bit value. So that the space required to store one color plane is (Height x Width x 8) bits.

Therefore, in idea of reducing the size, the approach was to go for the Huffman encoding. But before that getting the Discrete Cosine Transform (DCT) of the image paves the way for further Compression.

Rather than getting the DCT of the entire Image, getting the block-wise DCT gave good results, even when generating the Huffman code.

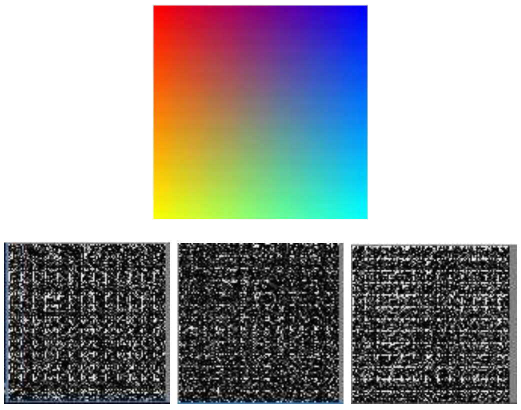


Figure 1: DCT of the Image (R G B separated)

As shown in [figure 1]. The block-wise DCT of the three planes have the DC components (Can be seen with white dots) and High frequency components in black.

In this case the DC components and some High frequency components from DCT will be both positive and negative numbers with larger values. So that they cannot be saved in 8bit chunks. Incase to reduce the size the Image planes has to be down converted by quantizing. But quantization will loss so much of the details, So, for the best results, possible datatype to store the DCT values was signed 16bit integers [figure 2].

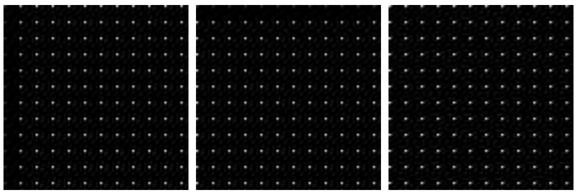
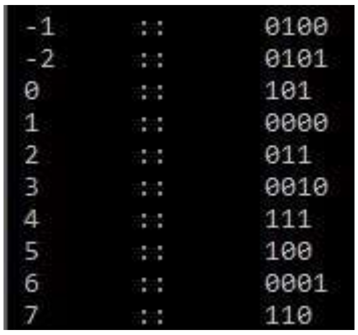


Figure 2: down sampled DCT images (Quantized)

3. HUFFMAN ENCODING

After quantization three planes were encoded with Huffman algorithm. The specialty in Huffman encoding is that it will reserve less number of bits to represent frequently used symbols and quiet larger number of bits to represent less frequently used symbols of an image. The generated Huffman table is shown in [figure 3].



```
-1    ::    0100
-2    ::    0101
0     ::    101
1     ::    0000
2     ::    011
3     ::    0010
4     ::    111
5     ::    100
6     ::    0001
7     ::    110
```

Figure 3: Sample Huffman Table

In implementing the Huffman encoder with c++, the binary tree method was used.

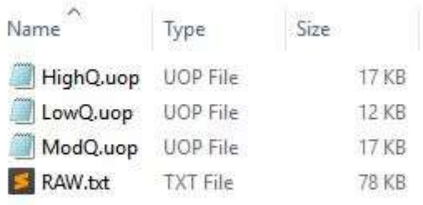
```
struct huff_tree {
      string  symbol;
      double  probability;
      vector<huff_tree> child;
      bool  code;
};

struct code_book {
      int idx;
      map<string, vector<bool>>
code;
      vector<bool> acc;
};
```

The quantized image planes were then encoded with the generated Huffman codes. After that the

```
Header{
        Unsigned int Width;
        Unsigned int Height;
        Uint8_t      BlockSize;
        Uint8_t      Quality;
        [int, bool]  HuffmanDictionary;
        Uint8_t []   bitsequance;
}
```

Figure 3: Image Header

image planes were saved to a file with a defined Image header.



Name	Type	Size
HighQ.uop	UOP File	17 KB
LowQ.uop	UOP File	12 KB
ModQ.uop	UOP File	17 KB
RAW.txt	TXT File	78 KB

Figure 4: Encoded and Raw Image Comparison

4. IMAGE ENCODER

The Image encoder uses a configuration file to read the user defined parameters for the encoding process. The encoded file will be saved with the extension "[*filename.uop*]" as shown above.

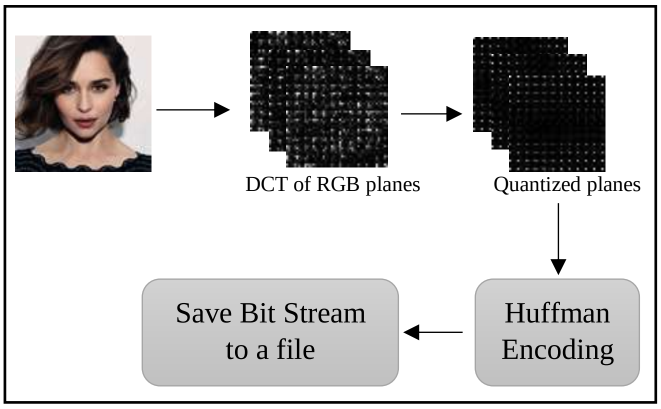
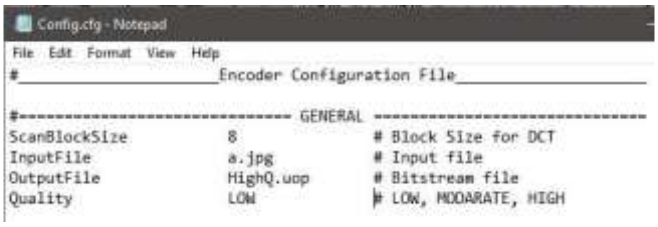


Figure 5: Encoder block diagram



```
Config.cfg - Notepad
File  Edit  Format  View  Help
#____________________________Encoder Configuration File____________________________

#------------------------------ GENERAL ------------------------------
ScanBlockSize              8              # Block Size for DCT
InputFile                  a.jpg          # Input file
OutputFile                 HighQ.uop      # Bitstream file
Quality                    LOW            # LOW, MODARATE, HIGH
```

Figure 6: Configuration file for the Encoder

In the Huffman encoding, the image planes were saved to a file. But saving to a file it has to be done with bitwise. The problem with writing a bit which cannot be done since we only having a least access to a byte only was overcome by using the following methodology.

First an empty byte was taken and then using the pixel values in plane, look for the Huffman code which are already in the Huffman dictionary. Then by checking each bit in the Huffman code, the empty byte was filled accordingly with ‘1’ or ‘0’. After a byte is filled it was written to a file and again empty the byte for remaining Huffman code as shown below

1	0	1	1	1	0	1	0	<< 1011101010

↓

Write to file

5. IMAGE DECODER

In the image decoder the Decoder will read the file with extension ‘*.uop’ and proceed for decoding. First the decoder reads the Header information. The Huffman table is generated thereafter by reading the file. Now as the decoder is ready with image parameters and the Huffman table, the symbols for Huffman code will be replaced. But these symbols are yet quantized. Then in the next step the dequantization will be taken place (Defines the quality of the image).

After the above process the RGB planes will be then subjected into IDCT (Inverse Discrete Cosine Transformation).

Finally, the RGB planes will be merged to have the reconstructed Image with desired quality.

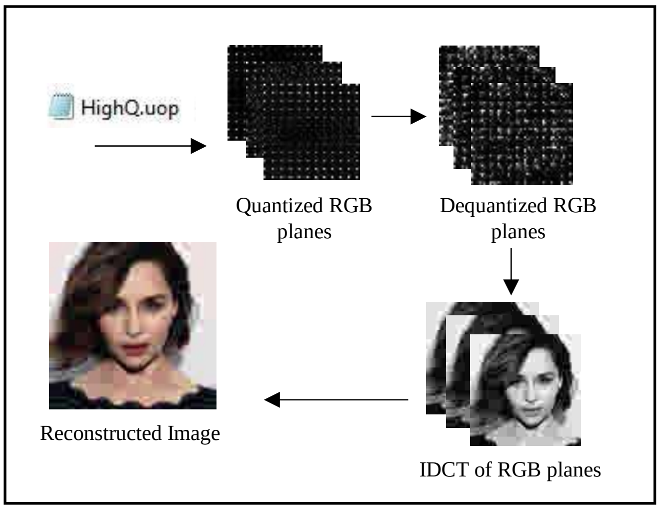


Figure 7: Decoder block diagram

Reading the file for Huffman decode

1	0	1	1	1	0	1	0	Read First byte (temp)
1	0	0	0	0	0	0	0	AND with 0x80
1	0	0	0	0	0	0	0	Result
0	1	1	1	0	1	0		<< 1 (temp)
1	0	0	0	0	0	0	0	AND with 0x80
0	0	0	0	0	0	0	0	Result
1	1	1	0	1	0			<< 1 (temp)
…	…	…	…	…	…	…	…	…

Code accumulation = “10…”.

As shown above the (temp) was read upto 8 bits and if the code is found within the byte pixel value is written to a image matrix referring the Huffman dictionary.

If the Huffman code is not found within 8 bits another byte from file was read to the (temp) byte.

```
104 104 sectn 8 2 sectn -1 000010 -10 001011 -2 000111 -3 001000 -4
10010 -5 01111 -6 001001 -7 001010 0 10100 1 000110 10 000101 11
11011 12 11111 13 000100 14 10111 15 01000 16 11010 17 10110 18
11101 19 01101 2 11001 20 10011 21 11100 22 01011 24 11110 25
01100 26 01010 27 10101 28 000011 29 000000 3 10001 4 01001 5
11000 6 00111 7 000001 8 10000 9 00110 sectn -1 01100 -10 001111 -
2 11011 -3 001011 -4 001100 -5 10110 -6 000010 -7 001101 -8 001110
0 01011 1 01001 10 000011 11 11100 12 001000 13 000111 14 10101
15 10001 16 000110 17 11111 18 11000 19 10100 2 01010 20 11110 21
10011 22 01111 23 01101 24 10010 25 000101 26 10000 27 01110 28
000100 29 10111 3 11010 4 01000 5 000001 6 11101 7 11001 8 000000
9 001001 sectn -1 11010 -10 001101 -2 01001 -3 001010 -4 11001 -5
01010 -6 001011 -7 001100 0 11101 1 01000 10 000111 11 11000 12
01101 13 000110 14 11111 15 10100 16 01111 17 11110 18 10011 19
000010 2 000001 20 10001 21 10010 22 000101 23 01110 24 01011 25
000100 26 10111 27 01100 28 000011 29 10110 3 10000 4 00111 5
000000 6 001001 7 11100 8 10101 9 001000 sectn
”¥)JJR”¥=□Æ¥)CŒ5)JS¥)JQ#R”¥)R”¥)šR”¥)šR”¥()Q¥)JT¥)@)JR” ¥
)JR”¥)JR”¥)JR$
JR”#R”¥”¥)JJR” Ô¥)JR”¥)JR”¥)JS ÁR”a†¥)JR”¥)JR”¥)JR”¥)JR”¥)JP
jR”¥¥)JR”

JR”¥)JR”¤£JR”¡AR”¥(8T¥)JR”¥)J$i

5)JR”¥)JR”¥)JR”¥)@†¥)JP`,¥)JR”¥)J
JR”¥)JR”¥2T¥)A©JR” Ô¥)JR”¥)JRƒ
=)JR”¥)JR”¥)JR”¥)JR”¥)JR”¥)JJR”¥)JR”¥)JR”¥)JR”¥¥)JR”¥)JR”¥)JR
```

Width-104 | Height-104 | blocksize-8 | Quality-2 | Huffman dictionary [symbol, code] | Pixel data (seen as ASCII) | **sectn** (a delimiter).

Figure 7: Sample Encoded file

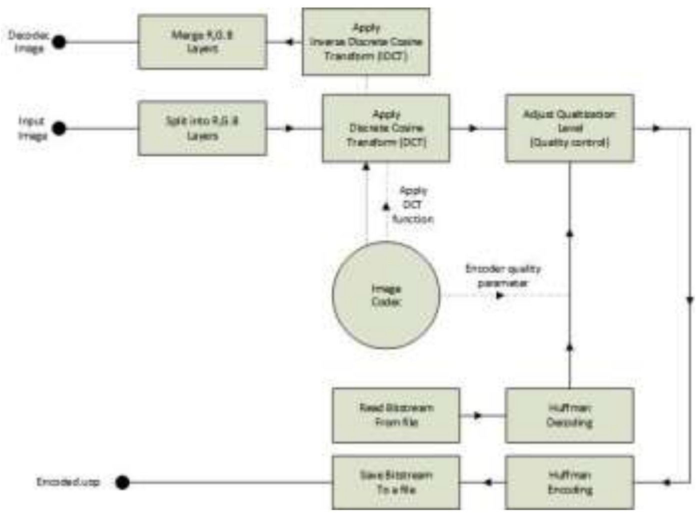


Figure 8: Image codec Block diagram

6. VIDEO CODING METHODOLOGY

The Video was processed as GOPs (Group of Pictures). A GOP contains an I-frame two B-frames a P-frame and again a P-frame followed by another two B-frames.

i.e.: [**I B B P B B P**] [**I B B P B B I**] …

First from the video a GOP is read, then Motion vectors for the first P-frame is calculated using I-frame There after those vectors are saved. Thereafter the P-frame is reconstructed using MV (Motion vectors) which is called as motion compensation and saved.

Then that P-frame is used as the reference frame and B-frame is motion compensated as above and saved accordingly.

Finding the Motion Vectors

Here motion vectors are computed using exhaustive search method.

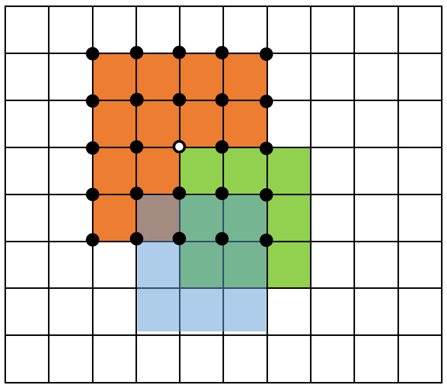


Figure 9: MV search in I-frame for a P-frame

Here the blue box is the searching macroblock and the orange area is the searching area for the block. The matching is done for the green block of the P-frame, if searching is done in I-frame.

The searching kernel goes to each black dot and calculates the sum of all pixels.

Among those sums then the minimum is found. And the MV is the location of the relevant minimum valued block location. From the center of the orange area.

Likewise, all the MVs are obtained for each plane.

7. VIDEO ENCODER

The MVs are generated for the frames. And using the motion compensated image, the original frame was deducted to have the residual image of the relevant frame.

Then the MVs are saved separately and the Residual images are saved after encoding using the Image encoder.

Figure10: I-frame

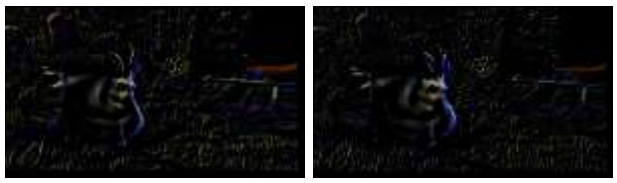


Figure 11: Residual images (Left)P-frame (Right)B-frame

Figure 12: DCT of a Sample Residual Image

8.VIDEO DECODER

In The decoder Using the motion vectors, motion compensation is done with the use of I-frames.

But to get I-frame ready, first thing is to decode the encoded images by Dequantizing>>IDCT.

Although the I-frames are completely recovered after the decoding the images. P-frames and B-frames are still in their residual form.

Therefore, before the reconstruction of the video these residual images are added with the corresponding motion compensated images. After that the decoding of the video is complete.

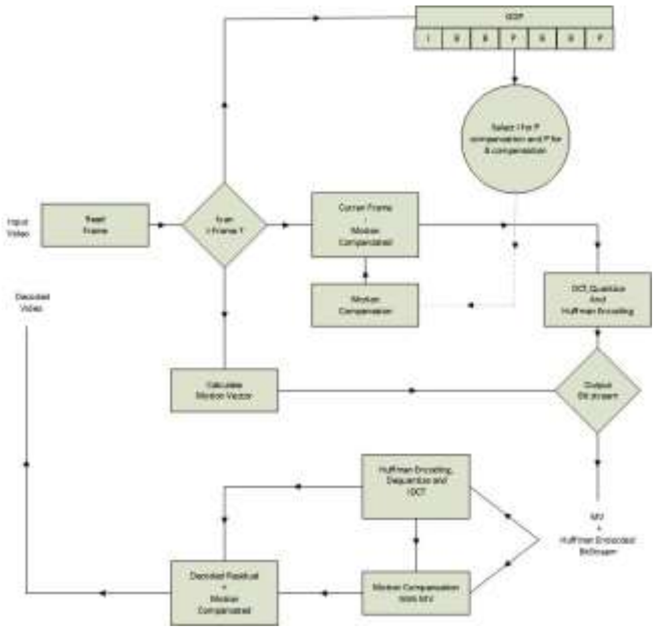
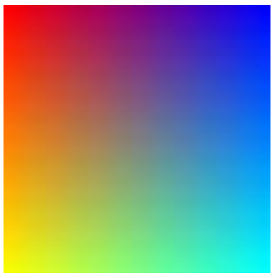
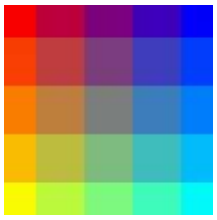

GOP
I
B
B
P
B
B
P
Select I for P compensation and P for B compensation
Input Video
Read Frame
Is an I Frame ?
Current Frame - Motion Compensated
Motion Compensation
DCT, Quantize And Huffman Encoding
Decoded Video
Calculate Motion Vector
Output Bit stream
Huffman Encoding, Dequantize and IDCT
MV + Huffman Encoded Bitstream
Decoded Residual + Motion Compensated
Motion Compensation With MV

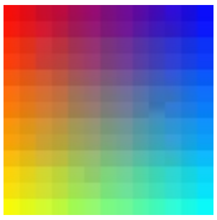
Figure 13: Video codec Block diagram

Decoded Files with 3 Qualities LOW, HIGH, MODARATE.

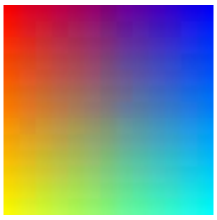




LOW



MODARATE



HIGH