Final Project, ROBT 310: Live Video Streaming

Batyrkhan Orynkul and Alibek Sailanbayev

I. Abstract

In this project we implement live video streaming application: one user is recording a video and simultaneously sending it to another user. Given a high-quality recording video, we apply compression to reduce the size of the video in order to send it fast. First, we lower the resolution of the frames of the video by reducing chrominance components U and V in YUV colorspace to a half of the pixels in vertical and horizontal directions. Then, each frame is divided into 8x8 blocks and the blocks are quantized in the frequency domain using Discrete Cosine Transform. After the quantization the frames represented in the frequency domain are compressed using lossless Run-Length Coding compression algorithm. The compression steps allows us to significantly reduce the size of the frames. The compression ratio is close to 23. Finally, the compressed video frames are being continuously sent through TCP socket from a server to a client, where they are being decompressed.

II. Introduction

Over the past decades, streaming media has developed significantly. Communication through video has become very popular and useful. Famous principal analyst at Forrester Research Dr. James McQuivey estimates that one minute video is equal to 1.8 million words![4] Last few years live video streaming has become very trendy and it is growing rapidly, for example, 81% of internet and mobile audiences watched more live video in 2016 than in 2015. [5] Streaming media is increasingly being paired with use of social media (Instagram, Facebook, Youtube).

For live streaming it is important to transmit video frames very fast, but usually high-quality video has large size. Without compression a two hour 720x480 video of full-color (3 channels) recorded at 30 frames per second requires 224 Gigabytes of space, moreover with 1 Mbps data transfer speed it will be transmitted in 62 hours! But to stream the video it should take about 2 hours. Obviously, the size is too big and the time required to send this video is very long, so live videos must be compressed.

After compression the live video frames or parts should be continuously sent to a client. In this paper we describe our approach of implementation of video streaming.

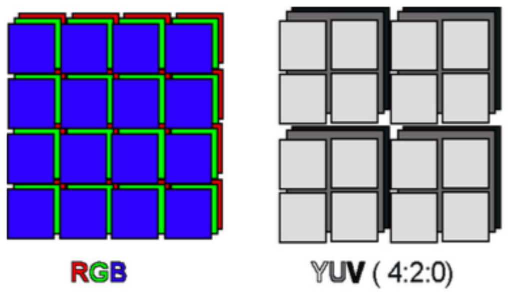
III. JPEG compression

JPEG compression method consists of several steps: the resolution reduction [1], discrete cosine transform quantization, lossless compression of quantized image [2].

A. The resolution reduction

As human eye has low sensibility to color information, an image color components can be slightly reduced. Given an RGB image, it should be changed from RGB colorspace to YUV colorspace. In this colorspace Y stands for the brightness component, U, V are chrominance components. Then U and V components are reduced by grouping pixels in 2x2 blocks, Figure 1. Using this sub-sampling we decrease the image size by 50%. [1]

Fig. 1. Grouping in U, V matrices



B. Discrete Cosine Transform Quantization

Now we divide the image in 8x8 blocks in each component and apply 2D Discrete Cosine Transform to each block. Using DCT (similar to Fourier Transform) we can represent the blocks in frequency domain [7]. The formula for DCT is:

$$F(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{7}\sum_{y=0}^{7} f(x,y)\cos(\frac{(2x+1)u\pi}{16})\cos(\frac{(2y+1)v\pi}{16})$$

, where $C(x) = \frac{1}{\sqrt{2}}$ if $x = 0$ and $C(x) = 1$, otherwise.

The formula for the inverse DCT is:

$$f(x,y) = \frac{1}{4}\sum_{u=0}^{7}\sum_{v=0}^{7} C(u)C(v)F(u,v)\cos(\frac{(2y+1)u\pi}{16})\cos(\frac{(2x+1)v\pi}{16})$$

. The implementation of dividing an $n \times m$ image with applying the 2D DCT to each block has complexity $O(64 * m * n)$.

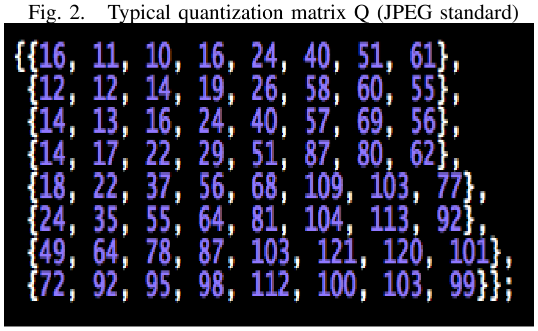
For our purposes we need fast calculations so we optimize it. 2D DCT can be found by calculating 1D DCT for each column and each row. This allows us to reduce the complexity by the factor of 8. [6]

Quantization
The human eye can easily see small differences in brightness over a large area (low frequency), however, in small areas (high frequency) the small differences are negligible for the human eye. This fact allows us to reduce the information in high frequency domain. It can be done by dividing matrix of frequencies F by some quantization matrix Q element wise:

$$F_{quantized}(i, j) = [\frac{F(i, j)}{Q(i, j)}]$$

[x] - integer part of x.
The quantization matrix is responsible for the compression quality. In our project we take Q as in the Original JPEG standard (Figure 2).[2]

Fig. 2. Typical quantization matrix Q (JPEG standard)



```
{{16, 11, 10, 16, 24, 40, 51, 61},
 {12, 12, 14, 19, 26, 58, 60, 55},
 {14, 13, 16, 24, 40, 57, 69, 56},
 {14, 17, 22, 29, 51, 87, 80, 62},
 {18, 22, 37, 56, 68, 109, 103, 77},
 {24, 35, 55, 64, 81, 104, 113, 92},
 {49, 64, 78, 87, 103, 121, 120, 101},
 {72, 92, 95, 98, 112, 100, 103, 99}};
```

C. Lossless compression of quantized image

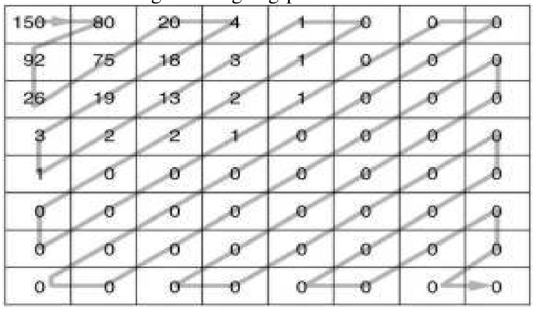
Now we need to compress our quantized image to send it to a receiver. After the quantization the image in the frequency domain has a lot of zeros mostly in right bottom like in Figure 3 (because of the method of quantization and the structure of quantization matrices). Therefore in JPEG method the common way to compress the matrix is to consider the matrix as a string using zig-zag path (Figure 4) and apply entropy coding to the string, usually Run-Length encoding is applied. [2]

Fig. 3. Typical DCT coefficient matrix after quantization

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

Run-Length encoding is simple lossless compression. It runs through a sequence and stores it as pairs (value, count). For example, a sequence [1, 1, 1, 3, 5, 5, 5, 0, 0, 0, 0] is encoded as [(1, 3), (3, 1), (5, 3), (4, 0)]. [3]
In our program we use slightly changed version of RLE, we run through a sequence and for every non-zero number s_n we encode (s_n, r), where r - number of zeros in the sequence between $s_{(n-1)}$ and s_n. For example, we encode a sequence [1, 1, 0, 0, 0, 0, 6, 0, 10, 0, 0, 8] as [(1, 0), (1, 0), (6, 4), (10, 1), (8, 2)].

Fig. 4. Zig-zag path in matrix



150	80	20	4	1	0	0	0
92	75	18	3	1	0	0	0
26	19	13	2	1	0	0	0
3	2	2	1	0	0	0	0
1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

IV. TCP SOCKET

In the project we use TCP socket to send data from a server to a client.
A socket is one endpoint of a two-way communication link between two programs running on the network. A socket is bound to a port number so that the TCP layer can identify the application that data is destined to be sent to (Figure 5). First of all TCP server creates a socket and binds it to the specified port and listens to that port. Then TCP server waits for the connection from the client and blocks the process until the client is connected. After the client is connected, three way handshake is performed by sending SYN (synchronize) and ACK (acknowledgement) packets before starting the transmission of actual data. Then server waits for the messages until the client is closed the connection and sends ACK packets after each received packet of data. At the same time client side works as follows: creates TCP socket and connects with the server using the specified IP address and port of the server. After the three way handshake, client sends the video frame by frame by confirming that packets were delivered without loss by receiving ACK packets.

Fig. 5. TCP server and client

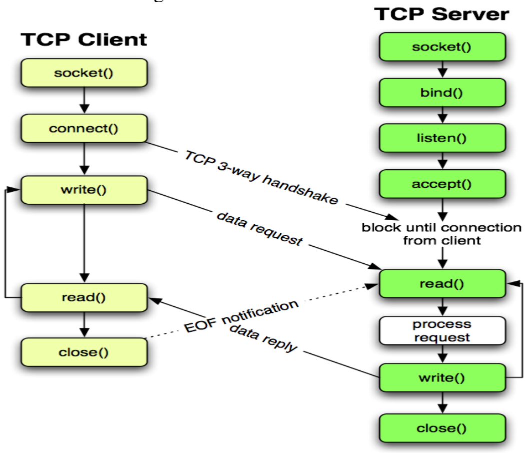
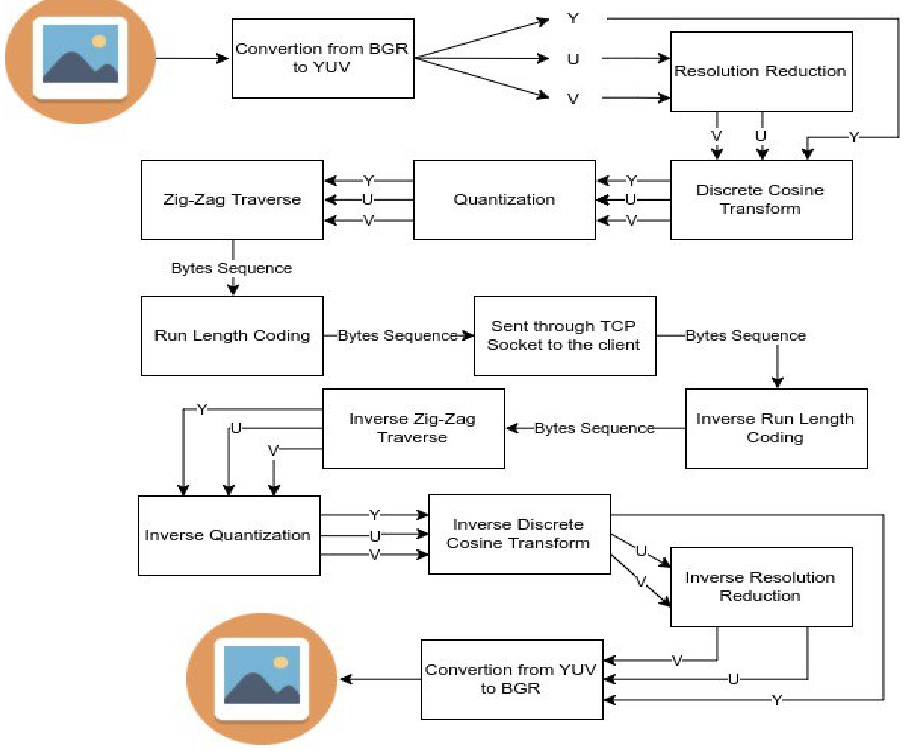


Fig. 6. General Scheme of our Project



Convertion from BGR to YUV
Y
U
V
Resolution Reduction
Discrete Cosine Transform
Quantization
Zig-Zag Traverse
Bytes Sequence
Run Length Coding
Sent through TCP Socket to the client
Inverse Run Length Coding
Inverse Zig-Zag Traverse
Inverse Quantization
Inverse Discrete Cosine Transform
Inverse Resolution Reduction
Convertion from YUV to BGR

TABLE I

RESULTS

	with compression	without compression
compression ratio	23	1
compression/decompression time for one 640x480 frame	0.206 seconds	-
compression/decompression rate for one 640x480 frame	5 frames per second	-
compressiont/decompression time for one 1920x1080 frame	1.23 seconds	-
compression/decompression rate for one 1920x1080 frame	5 frames per 6 seconds	-
required bandwidth for 10 fps video	465.8 kBps	9.2 MBps
required bandwidth for 20 fps video	895.2 kBps	18.4 MBps
transmission rate for NUdormitory Wi-Fi	1 frame per 0.21598 seconds (4.63 fps) with 3 seconds delay	1 frame per 14.34 seconds with 15.48 seconds delay

V. Experiments and Results

First, we implemented the image compression and tested it with an ordinary image.
6.2 Mb 1920x1080 image was chosen. Our program returned compressed file with the size 280 Kb after 1.23 seconds. So the compression ratio of the compression is close to 23.
In the Figure 6 you can see the general scheme of the implementation of live video streaming. The only remarks are that the server is recording video and simultaneously following the scheme by considering the video as sequence of frames and the client is consistently showing received frames on his screen.
We wrote the server program with compression and the client program with decompression. The server program records video, compresses every frame and sends each frame to the client. The client program receives the compressed frames, decompress it and shows them on the screen. In order to evaluate our approach we also wrote another server and client programs without any compression, this server records a video and sends it to this client without any compression, the client receives video and shows it on its screen.
The programs are written in C++ language. The servers are located on Ubuntu 16.04 Linux, the clients are located on OS X Yosemite 10.10.5.
The results are shown in Table 1. Note that with compression the program worked quite good and simple live video stream was created. On other hand without compression one frame was transmitted only in 14 seconds, which is not enough to

stream a video.

References

[1] Djordje Mitrovic: ”Video Compression”, University of Edinburgh.
[2] Wallace, Gregory. The JPEG Still Picture Compression Standard.
[3] CAPON, J. (1959). A probabilistie model for run-length coding of pictures. IRE Trans. On Information Theory, IT-5, (4), pp. 157-163.
[4] `https://blog.hubspot.com/marketing/video-marketing-statistics#sm.0000gcu09ck03de7yuk24dqgixgvn`
[5] `https://livestream.com/blog/62-must-know-stats-live-video-streaming`
[6] `https://unix4lyfe.org/dct/`
[7] Ahmed, N.; Natarajan, T.; Rao, K. R. (January 1974), ”Discrete Cosine Transform”