CSCI 576 Multimedia Project

Instructor: Parag Havaldar

Demo date: May 3rd and May 4th 2017

The course project is meant to give you an in depth understanding of some of the areas in multimedia technology. Since this is a broad field, there can be a variety interesting projects that can be done depending on your interests which can also extend to related and complementary topics that are taught in class.

Also, I have often found that a large project can be successfully accomplished via collaboration. Additionally, working together to design and integrate code can be a rewarding exercise and you will frequently need to work in teams when you set out to work in the industry. Accordingly, please form groups of **at least two but utmost four** students. We have started a discussion board to help you make groups, where you may post your preferred language of implementation, availability etc.

This time I want to suggest a topic that real problems that virtual reality and related industries are facing today. On the next page is a complete description giving the motivation and the requirements of your project. It should be a comprehensive project that aims to increase your fundamental understanding of aspects of multimedia.

Segmented and Gaze Controlled decompression for streaming displays such as VR

Motivation and Problem Statement

While traditional multimedia applications such as games and video are still very popular, an emerging area generating significant interest in immersion around today's entertainment landscape requires the capture, streaming and rendering of high bandwidth audio/visual content. For example applications such as Virtual Reality, Augmented Reality, Telepresence etc – aim to work with high resolution omnidirectional video. Furthermore these videos may take the form of monoscopic (one) or stereoscopic(two) streams that need to be streamed to a VR headset. Immersion is possible only when the video is of high resolution with realtime and interactive feedback while browsing. This puts stringent requirements on computational platforms as well as the available streaming bandwidth in devices such as VR headsets. While processing/computing power of such devices is on the rise, and the bandwidths of communication are also on the rise, successful systems that seamless interact with high resolution are still challenge. One smart way to delivering such content is by adaptively encoding/decoding the content based on semantics in the video and using view dependent gaze to control the decoding

This project is an interesting application of block based motion detection and compression in video based communication industries. Here you will be given a video as input that you should preprocess into semantic layers, encode each layer separately and store into a file for reading/streaming later. You will also design a player that will read this compressed file, selectively decode layers to display depending on which parts of the video you are gazing at (controlled by mouse pointer) and create the final displayed frames.

To formally describe the project –

1. Sematic Layering of video: there is a pre processing step where you will analyze the input video, break it into foreground/background layers. This is not required to work in real time, and semantic accuracy may vary.

2. Compressing of layers: Each of the foreground/background layers will need to be stored in a compressible manner so that later access can decide what needs to be read/streamed to the player and displayed. This is not required to work in real time, and file format for storing the data is up to to you to decide – though an example file format has been suggested.

3. Displaying of video: Your player should read the compressed video file and display the video per quantization inputs to simulate bandwidth distribution among layers. Foreground layers should be ideally more clearly visible than background layers .

4. Applying gaze control: We don't have Head mounted VR displays to work with to see the value of this effect, but we can certainly simulate gaze based control by using a mouse pointer. With your mouse location acting as "gaze direction" you want decode/display a local area around the mouse location will the best clarity (no quantization) compared to other layered areas.

You will be writing two programs that logically share the same functionality – a layer based encoder, that will take as input a video file to produce a compressed file and a decoder that will take as input the compressed file along with other parameters to generate a rendering of the final output. The specifics of the input arguments to your two applications are described below.

1) myencoder.exe input_video.rgb

input_video.rgb is the input file to your encoder.

the output of this is a "input_video.cmp" layer wise compressed video file

2) mydecoder.exe input_video.cmp n1 n2 gazeControl

decodes the compressed file as renders layers according to a quantized values

n1 represents quantization step for foreground macroblocks

n2 represents quantization step for background macroblocks

gazeControl - boolean value 0 or 1 to evaluate gaze control

The four parts of the whole project are described below.

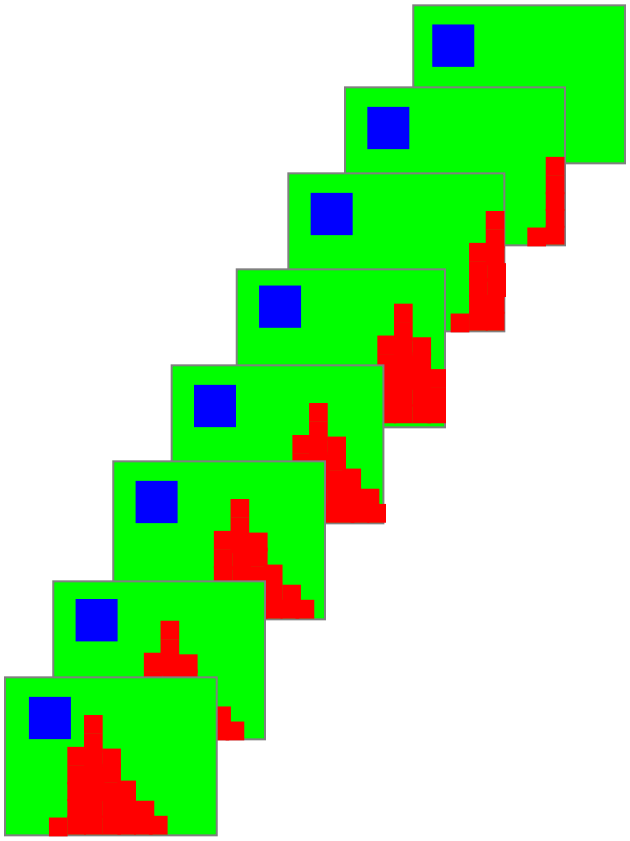
1) Semantic Layering of Video (in myEncoder.exe)

You are required to divide each frame of the video into **background** and **foreground** macroblocks objects. Note that your macroblocks may not be precise, especially at the boundaries but nevertheless you should able to segment out contiguous blocks of foreground elements and background elements. These are the guidelines you should follow for this part:

1. The first frame may be assumed to be an I frame, but for every successive frame divide the image frame into blocks of size 16x16 pixels
2. Compute the Motion Vectors based on the previous frame – this is a imilar to the block based SAD (sum of absolute difference brute force search) or even better any fast motion estimation (FME) technique. At the end of this step, each macroblock of the frame should have a motion vector.
3. Organize the blocks into background and foreground based on the similarity of the directions of their motion vectors. Background macro blocks either have a close to zero motion vectors (if camera is not moving) or a similar shared direction motion vector (when the camera is moving). On the other hand, foreground macroblocks have similar motion vectors with macroblocks of a moving region normally being connected or contiguous. Thus, the main criteria for grouping regions is
    - Contiguous or adjacent

- The motion vectors are all consistent – *important!* The consistency of the motion vector direction gives you an indication that all the macroblocks probably belong to the same object and are moving in a certain direction.

Note - you can have different regions moving in your video sequence. By the end of this first part you should know for each frame for each macroblock – whether they represent a foreground or a background macroblock. The general purpose solution of this problem is not easy, but for the example videos provided your program should perform well and demarcate foreground/background with good relative accuracy. An example of such an output shown below where the green macroblocks represent the background and the red/blue macroblocks represent the foreground elements.



2) Compression of Layers (in myEncoder.exe)

After the segmentation you are required to compress the background and foreground macro blocks. In a typical streaming architecture, you would convert input components into its *quantized* DCT coefficients per bandwidth/gaze requirements and buffer this compressed data for streaming. And in such architectures, the continuous bandwidth sensing would control quantization of the DCT coefficients of the layers (background macroblocks will generally be more quantized whereas foreground macroblocks will have lesser quantization). Also in such architecture, the gaze requirements would decide which of the macroblock areas need to be clear and unquantized (macroblocks in the gaze

areas will be less quantized than the macroblocks outside the gaze areas). In such cases both properties of bandwidth and gaze control are communicated by the client to the streaming encoder.
In our case, for simplicity, we will only emulate this streaming feature and so the encoder will create and store all the transform coefficients, but the decoder will decide the quantized output to display for foreground/background layers and gaze dependent decodings. So the requirements at the encoder become simplified enough to -

1. Divide each frame into 8x8 blocks (similar to MPEG). Each block is either part of a foreground macroblock or a background macroblock.
2. Scan frame 8x8 blocks in scan line order, for each block compute the DCT values and save them into a compressed file. This file will be used by the decoder to decode and render the results depending on the quantization of layers. While you are free to choose any file format for your compressed file, here is a simple suggested format that you can make use of.

Fore a given input file – input_file.rgb – produce a compressed file input_file.cmp that contains the following -

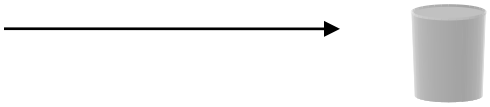
block_type coeff1 coeff2 ... coeff64 coeff1 coeff2 ... coeff64 coeff1 coeff2 ... coeff64
block_type coeff1 coeff2 ... coeff64 coeff1 coeff2 ... coeff64 coeff1 coeff2 ... coeff64
block_type coeff1 coeff2 ... coeff64 coeff1 coeff2 ... coeff64 coeff1 coeff2 ... coeff64
block_type coeff1 coeff2 ... coeff64 coeff1 coeff2 ... coeff64 coeff1 coeff2 ... coeff64
block_type coeff1 coeff2 ... coeff64 coeff1 coeff2 ... coeff64 coeff1 coeff2 ... coeff64
...
...

Where block type is a number to indicate state to say whether each 8x8 block is a background (0) or foreground (1,2,…) block and the coefficients are the quantized DCT coefficients for each block. For each block there are three sets of sixty four coefficients, a set for each of the R, G and B components **. Note here you are NOT expected to perform zig zag ordering, intermediary representations and entropy coding as per the JPEG standard. We are more interested in using the layered data blocks to emulate qualitative decoding in the player.**

3) Decompression and Decoding (in myDecoder.exe)

The decoder (and display) will take the input_file.cmp containing the layered frequency transforms. Additionally, it will also take quantization parameters *n1* and *n2* for the *foreground* and *background* layers. This is to emulate the bandwidth scalability. So depending on whether a block belongs to the foreground or background (which is represented in the file), divide the DCT coefficients by *n1* (or *n2*), round off quantized values, and do the reverse to get the dequantized DCT coeffects. Compute the IDCT for all the quantized blocks and display the results on each frame.
Here is an example workflow with qualitative results. Note that in the decoder the foreground macroblocks (macroblocks in motion) have less compression artifacts than the background macroblocks. The effect would have been reversed if the values of *n1* and *n2* were interchanged.

At the Encoder:

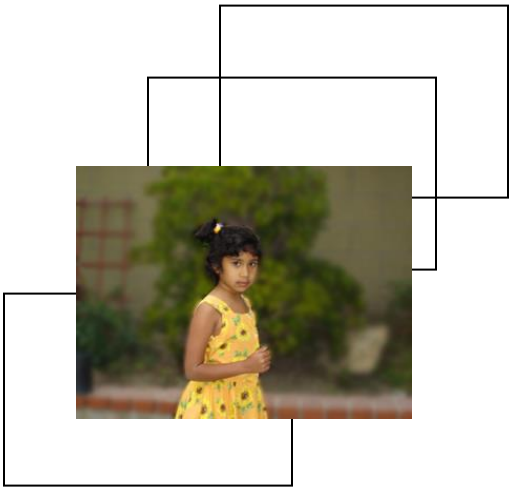


Compressed file

At the Decoder :

n1 + n2

Compressed file



4) Applying gaze control

The last parameter to your decoder is to control the decoded quality based on gaze. Normally if you had the "area of interest" that the person is viewing, then this could be used to additionally optimize the bandwidth of streaming. The area of interest could be defined by the gaze of where the eyes are looking at, or even refer to the display area of a panoramic video. In our case, we don't have Head Mounted Displays to work with, but we could emulate the same effect by using a mouse pointer.
To emulate gaze control, your decoder should analyze the mouse location on the screen. Based on location the decoder will NOT quantize any of the of the blocks in the square 64x64 area around the mouse pointer. As the mouse moves around, the unquantized display area should follow the mouse. Make sure you deal with boundary areas appropriately. What is important is that you have a selectively changing area of unquantized display that is dynamically following the mouse pointer.

Extra Credit

There is extra credit reserved for those who can achieve real time decoding and display. Extra credit is not compulsory – it is more important to have the requirements of the project done before you go ahead and implement the extra credit part.

A few implementation notes

- You will have to create your own methods to display the decoded video. This will include playing the video ***in a loop*** and also ***pausing/restarting*** so that we can evaluate your output. We have already provided you with starting code to display an image in your previous assignments, which you may have made use of effectively. The input datasets and a description will be kept on the class websites and the formats should remain constant across all datasets. You may assume that your video width and height are each multiples of 16 and the frame rate is fixed as described on the class project site.
- We will only test the quality of your output, and if extra credit is attempted, then we will analyze the real-time performance of the decoding system. Correspondingly, we are not going to penalize any implementation adversely that does not run fast enough. As long as you can perform the encoding process and display the video with the parameter set, that should be sufficient. We are expecting the whole evaluation process to not take more than 10-15 mins.
- Remember to do macroblock searches only using data of one component (not all three!). Typically you could convert the image format to YUV and use the Y component for all evaluations. If not, you may just use average of all components. However, you will have to do a DCT conversion on all the three RGB channels.
- Note that you are making use of motion prediction to compute motion vectors for video segmentation only – so the accuracy of motion vectors normally required for compression is not needed, but if you have accurate motion vectors it will help you in your segmentation process.