Video Layer Based Compression

Description

This project is an interesting application of block based motion detection and compression in video based communication industries including entertainment, security, defense etc. Here you are required to separate an input video into different layers based on motion characteristics - those that represent the foreground layers and the background layer. Each layer is then compressed differently according to input parameters provided. The theory taught in class dealt with computing motion vectors to help better compression of video, here we use it to segment a moving region and then use the same for compression.

To formally describe the project –
Given video frames as input, process the video to find out objects in motion or macro blocks that compose objects in motion and macro blocks that comprise the background. Once these macroblocks are detected, each one is compressed using quantization parameters provided as input. You will be writing two programs that logically share the same functionality – a layer based encoder, that will take as input a video file and two quantization parameters to produce a compressed file and a decoder that will take as input the compressed file to and 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 n1 n2
input_video.rgb is the input file to your encoder.
n1 represents quantization step for foreground macroblocks
n2 represents quantization step for background macroblocks
the output of this is a "input_video.cmp" layer wise compressed video file
2) mydecoder.exe input_video.cmp
decodes and renders the compressed file

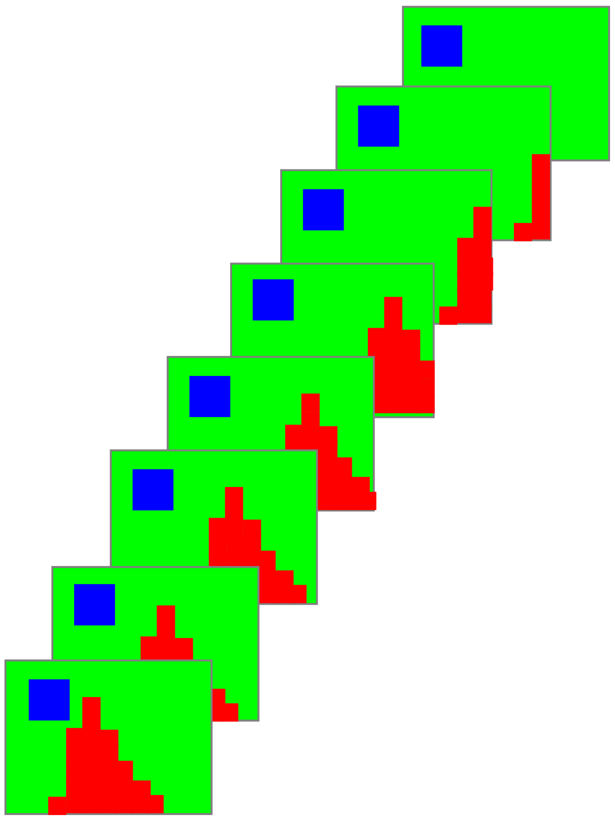
The whole project and the applications may be divided into four main parts described below.

1 Video Segmentation (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 similar to the block based MAD (mean 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 constant same 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
   a) Contiguous or adjacent
   b) 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 that 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 as long as your program comes up with a good demarcation of foreground and background based on motion, that is fine. An example is demarcation is shown below where the green macroblocks represent the background and the red/blue macroblocks represent the foreground elements.



2 Compression (in myEncoder.exe)

After the segmentation you are required to compress the background and foreground macro blocks using different quantization values. The quantization used for quantizing the DCT coefficients of each block is controlled by the input parameters *n1* and *n2*. Background macroblocks will generally be more quantized whereas foreground macroblocks will have lesser quantization. The requirements here are

1. Divide each frame into 8x8 blocks (similar to MPEG). Each block is either part of a foreground macroblock or a background macroblock.
2. Perform a 8x8 DCT, quantize the DC/AC coefficients using a uniform quantization function where each number in the quantization table is 2^n.
3. Scan frame 8x8 blocks in scan line order and save the quantized coefficients of each macroblock into a compressed file. This file will be used by the decoder to decoder and render the results of your compression. While you are free to choose any file format your compressed file, here is a simple suggested format that you can make use of.

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

n1 n2
block_type coeff1 coeff2 … coeff64 coeff1 coeff2 … coeff64 coeff1 coeff2 … coeff64
block_type coeff1 coeff2 … coeff64 coeff1 coeff2 … coeff64 coeff1 coeff2 … coeff64

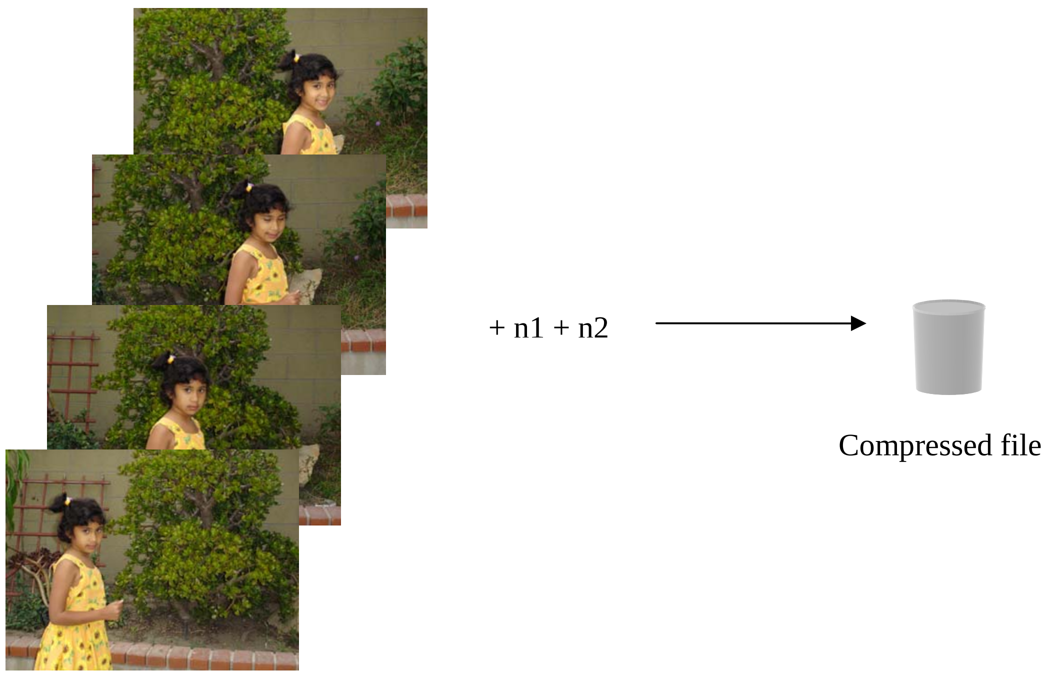
block_type coeff1 coeff2 … coeff64 coeff1 coeff2 … coeff64 coeff1 coeff2 … coeff64
block_type coeff1 coeff2 … coeff64 coeff1 coeff2 … coeff64 coeff1 coeff2 … coeff64
block_type coeff1 coeff2 … coeff64 coeff1 coeff2 … coeff64 coeff1 coeff2 … coeff64
….
…

Where n1, n2 are the quantization inputs, block_type is a binary state to say whether each 8x8 block is a foreground or background block and the coefficients are the quantized DCT coefficients for each block. Note that 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**.

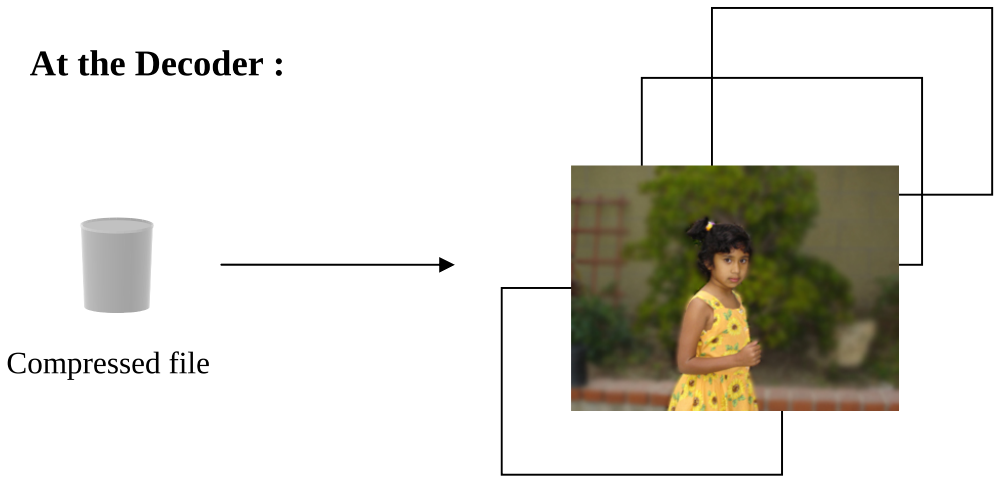
3 Decompression and Decoding (in myDecoder.exe)
The decoder will take the input_file.cmp as input. You should be able to parse the file to get the n1, n2 values and the coefficients of each macro block. Additionally, you should be able to dequantize the blocks using either *n1* or *n2* depending on whether the block_type represents a foreground object or a background object. Decode and display the results of your decompression.

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. Note the effect would have been reversed if the values of *n1* and *n2* were interchanged.

At the Encoder:



At the Decoder :



EXTRA CREDIT

There is extra credit reserved for those who can achieve substantial compression. Extra credit is not compulsory – it is more important to have the required quality of your output before you go ahead and implement the extra credit part.

Although you will not have to implement an entropy encoder, one way to test the quality of your compression rate is to run GZIP on your compressed file. This will entropy code your .cmp file very well and give us an order of how much redundancy you have removed. Correspondingly, we will run your encoder, take your .cmp file and run a ZIP program on it and note the size of your zip file to see if it your compressed image quality and obtained compression is desirous of extra credit.

A few implementation notes

- You will have to create your own methods to display the input and output videos. This will include playing the video at frame rate and also pausing and stepping through the video frames so that we can evaluate your compressed-decompressed output to the original. We have already provided you with starting code to display an image in your previous assignments, which you may have made use of effectively in your previous assignments. The input datasets and a description will be kept on the class websites and the formats should remain constant. 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 amount compression you have achieved. Correspondingly, we are not going to penalize any implementation that does not run fast enough. As long as you can end your encoding and decoding process within 10 mins that is fine (which is roughly the time needed for you to make your presentation).
- 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 the R component. However, you will have to do a DCT compression on all the three channels.
- Note that you are making use of motion prediction to compute motion vectors for video segmentation only. If attempting extra credit, your compression can be substantially higher if you make use of the motion vectors in compression. Ie – you reconstruct a predicted frame based on motion vectors and encode only the difference between the predicted and actual frame. In this case you will have to add a motion vector for each macroblock as well. Additionally, organizing your DCT coefficient scan order can also help in your extra credit.