Objective: To consolidate the concept we learned in class for media compression and to learn how to use OpenCV or Matlab for interactive media processing. This assignment is due on Oct. 15.

1 Entropy Coding

1. What is the entropy of the "toy" image below, where numbers 0, 20, 50 and 99 denote the gray-level intensities?

```
20 20 20 20 20 20 20 20
0 0 0 0 0 0 0 0
0 0 50 50 50 50 0 0
0 0 50 50 50 50 0 0
0 0 50 50 50 50 0 0
0 0 50 50 50 50 0 0
0 0 0 0 0 0 0 0
```

2. Construct the Huffman coding tree. What is the average bit rate per pixel when using Huffman coding?

3. Compute the differential image. (Assume there is a “0” at the beginning of each row and the prediction is restricted in each row). Compare the difference image's entropy with the original one.

4. Discuss why differential image can be used to achieve better compression ratio.

2 Quantization

Quantization has been used in different lossy media representation, such as JPEG and MPEG. Here we will experiment with quantization on image qualities. We write a short program to visualize the distortion caused by quantization interactively. The code is based on OpenCV. OpenCV provides a nice interface for simple interactive applications. (Note that a Matlab implementation is also good. I will try to put another Matlab version on the web soon.) The GUI interface of the program is shown as Fig. 1.

This program extracts non-overlapping 8x8 image blocks from the source image, does DCT for each block, quantizes these blocks based on the quality factor set on the sliding bar, does IDCT, and displays the original image and the quantized image side by side. We can also move two cursors to show the difference of the pixel values in the original image and the processed image. In fact we only need to move one cursor. The other cursor always points to the same location in another image (either the original one or the processed one).

An incomplete but runnable program is available at the course website. The given program has some functions missing. You need to complete the function so that the program does the right job.

The first missing function is `get8x8imageblock`. 

```
Figure 1: The look of the running program.

```c
// Extract a 8x8 image block
// Parameters:
// im: the input image
// col: the column of the left-top corner of the block
// row: the row of the left-top corner of the block
// block: the output 8x8 image block
get8x8imageblock(IplImage *im, int col, int row, IplImage *block);
```

which extracts a 8 × 8 image block with the left-top corner at column `col` and row `row`. Pay attention to the image boundaries. The 8 × 8 image block may cross image boundaries. If the boundary crossing happens, for pixels outside of the boundaries, set zeros to the corresponding pixels in the block. Note that this is not a good method for real image compression. JPEG in fact uses techniques such as padding to fill the missing pixels.

The second missing function is similar to the previous one. This time, we set the image block at a specific location instead of reading the block. The interface of the function is as follows.

```c
// set a 8x8 image block
// Parameters:
// im: the input image where the block is inserted
// col: the column of the left-top corner of the block
// row: the row of the left-top corner of the block
// block: the input 8x8 image block
void set8x8imageblock(IplImage *im, int col, int row, IplImage *block);
```

The third missing function is the quantization function. The quantization table for DCT coefficients is given as a 2D array `qtable[8][8]`. The quantization should also be based on the quality factor you set on the sliding bar. Quality factor has a range from 0 to 100.
There are many different ways to adapt the quantization levels. One simple way is

\[ qtable(u, v) = qtable(u, v) \times (101 - qfactor)/16; \]

\[ F_q(u, v) = \text{round}(F(u, v)/qtable(u, v)) \times qtable(u, v). \]

You can experiment with other methods to adapt quantization to quality factors.

If you have completed the above three functions, you should have a working program to show the original image and the quantized image. You can slide the sliding bar to set the quality factor and the result will show instantly (depending on the speed of your computer). Now, we would like to add a bit more on the program to show the pixel value differences at the same location for the original image and the processed image. You need to modify the mouse callback function `onMouse` to implement the functionality. The current mouse callback function is as follows which displays only one cursor on the image.

```c
void onMouse ( int event, int x, int y, int flags, void* param )
{
    switch( event )
    {
    case CV_EVENT_MOUSEMOVE:
        {
            CvPoint center = cvPoint(x, y);
            cvCvtColor(canvas, canvask, CV_GRAY2RGB);
            drawCross(canvask, x, y);
            showGrayLevel(canvask, x, y);
            cvShowImage(wndname, canvask);
        }
        break;
    }
}
```

The modified version will display two cursors on the image. One cursor is controlled by the user and the other one synchronizes with the user cursor and points to the same location on the image.

The skeleton code uses many global variables to simplify the message passing between different functions. Briefly discuss how you can improve the code.