Exercise Multimedia Technology WS 2003/2004

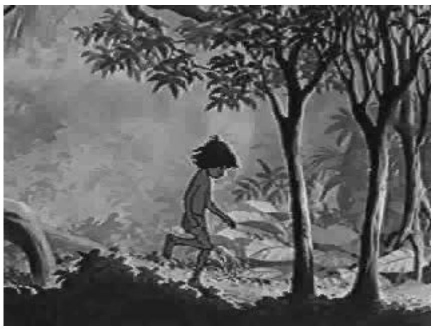
Sheet 4 (November 13th 2003)

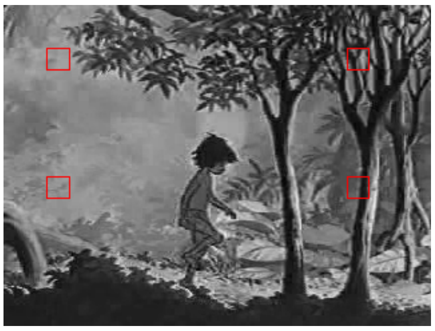
4.1 Video compression using MPEG

4.1.1 Calculating motion vectors

The gray-scale images *Mogli I* and *Mogli II* shall be given at a resolution of 320x240 pixels. You can download both images as plain ASCII files.

www.informatik.uni-mannheim.de/pi4/data/mogli_1.txt
www.informatik.uni-mannheim.de/pi4/data/mogli_2.txt


Mogli I


Mogli II

Each pixel is stored as a single number between 0-255 and the gray values of the images have been ordered in the file column-by-column and row-by-row starting in the upper left corner and ending in the lower right corner. In the image above you can see four blocks of size 16x16 pixels (the borders of the blocks are not visible in the image you can download). The left upper corner of each block is located at (32, 32), (32, 128), (256, 32) and (256, 128).

Find out the appropriate motion vectors for the blocks in *Mogli II* that will best match blocks in *Mogli I*. Use a full-search approach with a radius of 16 pixels.

That means that your search area around each block location will have a range between +/- 15 pixels (for each axis).

As you have to compare each candidate block in *Mogli I* with the corresponding red block in *Mogli II*, you will have to calculate the difference. Use the L_1 norm in this case[1].

In order to find a best matching Block B for a block A in another image, we have to define the distance between two blocks:

$$match(A,B) = \frac{1}{N^2} \sum_{i=0}^{i=N} \sum_{j=0}^{j=N} |A(i,j) - B(i,j)| \tag{1}$$

```
void fullSearch (...)
{
  double minDistance = double(INT_MAX);
  double distance;

  for (int dx = -radius; dx <= radius; dx++) {
    for (int dy = -radius; dy <= radius; dy++) {

      distance = match(previousFrame, blockX+dx,
             blockY+dy, currentFrame, blockX, blockY);

      if(distance < minDistance) {
         minDistance = distance;
         vectorX = dx;
         vectorY = dy;
      } // if
    } // for
  } // for
} // fullSearch
```

[1]The L_1 norm between two values x and y is defined as $|x-y|$

Position of Block (32, 32):	(39, 32)
Position of Block (32, 128):	(38, 128)
Position of Block (256, 32):	(263, 32)
Position of Block (256, 128):	(263, 128)

4.1.2 P Frames

Explain the properties of P frames with regard to the previous exercise.

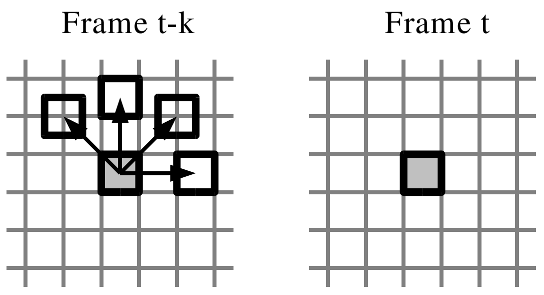
4.1.3 B Frames

Explain the difference between P and B frames.

A P-frame (predictive frame) tries to compose a frame P_t from another preceding frame P_{t-k} by choosing appropriate blocks from P_{t-k} and inserting them into P_t. In the above example *image a* is the original frame. In *image b* the difference between *image a* and the previous frame is displayed. Very light and very dark gray values symbolize big differences between the two succeeding frames. The mean gray value (a value of 128) stands for no difference. As *image b* contains lots of light and dark pixels, the previous frame was obviously not a good predictor for P_t.
Image c was composed from content of the previous frame by using motion compensation. Even though the result was not perfect, it is a far better approximation. The error *image c* after the motion compensation is sometimes referred to as the *residual*. It is also included in P-Frames as a DCT-coded image which has to be added to the initial motion-vector based image by the decoder to compensate the error from an insufficient motion prediction.

4.1.4 Motion vectors and the optical flow

The optical flow shows how the pixels move between two consecutive frames. Critically comment on the usefulness of the MPEG motion vectors as a predictor for the optical flow.



Once again motion compensation tries to find a good match for a block in a preceding frame in a search range round the location of the original block. If the camera e. g., pans from left to right, the image blocks will appear to move from the right to the left-hand side. Many blocks can be expected to follow the motion of the pixels a.k.a. the *optical flow*.
However, it is not sure that the full search, as implemented above, always considers the actual block movement to be the best match. Think of a chessboard pattern. Even under the assumption of a horizontally moving camera, any chessboard field would fit well so there is no guarantee that the best match reflects the actual movement.

4.2 Wavelet coding

You can download an audiostream as textfile audio.txt. Each number is an 8-Bit sample in ASCII format. You can use the Windows Tool Sound.exe in order to play the file. Decompose the samples into high-pass and low-pass coefficients using the filters from the lecture. Afterwards set all high-pass values to zero whose magnitude is less than 80. What happened to the audio signal after the synthesis?

www.informatik.uni-mannheim.de/pi4/data/Sound.exe
www.informatik.uni-mannheim.de/pi4/data/audio.txt

Many high frequency components have been removed from the signal so that it's brilliance is lost to a certain degree.

```
#include <stdlib.h>
#include <stdio.h>
#include <iostream.h>
#include <fstream.h>

const long MAX_LINE_SIZE = 256;

int main(int argc, char** argv)
{
    char   text[MAX_LINE_SIZE];
    double mean, difference;
    double value[2];
    long   index;

    if(argc > 1) {

        ifstream audio_text(argv[1]);
        index = 0;

        while(!audio_text.eof()) {

            audio_text.getline(text, MAX_LINE_SIZE);
            value[index % 2] = atoi(text);

            if((index % 2) == 1) {
                difference = (value[1]-value[0])/2;
                mean = (value[0]+value[1])/2;

                if(fabs(difference) < 80) {
                    difference = 0;
                } // if

                // orig values

                cout << value[0] << endl;
                cout << value[1] << endl;

                // quantized values

                cout << mean-difference << endl;
                cout << mean+difference << endl << endl;
            } // if

            index++;
        } // while
    } // if
    else {
        cout << endl << ‘‘no filename provided’’ << endl << endl;
    } // else

    return 0;
} // main
```