Design of Multimedia Applications:

Error Concealment in Digital Video

Group 40:
Eveline Hoogstoel, Titouan Vervack and Dries Wijns
1^{st} Master in Computer Science Engineering

1 Explanation of the used algorithms.

Exercise 2.B

For the implementation of exercise 2.B we have first calculated the distances of the pixels left, right, above and below the pixel. Once we know these distances it is easy to get the 2 lowest numbers. The distances were stored in an array, so once we have determined on which index the two lowest values are stored we know exactly which pixel it is. We can indicate which pixels are taken with extra boolean values (`t_taken`, `b_taken`, `r_taken` and `l_taken`), by initialisation of these values we take into account whether the macroblock exists or not. In the formula for the calculation of the luma- or chrominancevalue we multiply each term by it's corresponding boolean, so we know if we have to take the value of that pixel into account or not.

Exercise 2.C

First of all it should be said that we were unable to finish this exercise. We were able to do edge detection but adding actual extrapolation was outside of our capabilities. Nevertheless we'll try to discuss what we were unable to make in pseudocode and try to predict it's results. We also did measurements for this exercise but these are of course not as they should be.
For this assignment we first do edge detection and with the information gained from this we extrapolate the missing macroblock. The technique that we used for edge detection is the Sobel operator. We chose this technique over various others, i.e. Canny, because it gives a good result and is more accurate for corners than Canny is. It is also one of the easier ones to implement. It's disadvantage is that Canny performs better. Sobel is also used in Canny which gave us the ability to use Canny later on.

We started off by looping over all macroblocks in the frame and for those missing we collected the four macroblocks around it. From these macroblocks we take the three rows or columns closest to the missing macroblock, this in order to simplify the calculations and reasoning. On every pixel's lumavalue (because when we purely consider the lumavalues of an image we actually have it's grayscale) of these simplified macroblocks we ran the Sobel operator. The dotproducts needed in Sobel are done by creating a 3x3 block with the current pixel in the middle. Using Sobel we can also calculate the gradient's direction. In order to simplify this we grouped these directions according to the 8 wind directions. We also save the pixels with the same direction in a list so that we can decide where the edge starts later on.

This is about how far we were able to get so we will now discuss what our further plans with this were. From here on we would've checked if there were three pixels (in the three different rows/columns) with the same direction that were aligned in such a way that they'd form an edge. If less than three were found we decide that no edge was found. We do this for all blocks and then we check the found edges with all the others to see if two of them could be connected. Once that is done all we have to do is fill in the planes between the extrapolated edges.

```
1  void conceal_spatial_3(frame)
   {
   for (int i = 0; i < frame->getNumMB(); i++) {
       if(frame->getMacroBlock(i)->isMissing()){
5        // Get the blocks left, right, above and below the missing block
         blocks = getNeighbouringBlocks();
         // Only get the 3 rows or columns that border to the macroblock
```

```
        sides = getSides(blocks);
        edges = sobel(sides);
        extrapolate(edges, sides);
  }
}

short horizontal_sobel[3][3] = {
    { -1, 0, 1 },
    { -2, 0, 2 },
    { -1, 0, 1 }
};

short vertical_sobel[3][3] = {
    { 1, 2, 1 },
    { 0, 0, 0 },
    { -1, -2, -1 }
};

edge* sobel(sides){
  // The 8 wind directions in degrees: 0, 45, 90, 135, ...
  int direction[8] = { 0 };
  // The pixels where we detected the angles
  list<list<pair<int, int>>>  indices = new list(8, list<pair<int, int>>(3 * 16));
  for (pixel in side){
    for (3x3Block)
      if (partOf3x3Block.outOfBounds()){ continue; }

      gx = 3x3Block->luma.dotProduct(horizontal_sobel);
      gy = 3x3Block->luma.dotProduct(vertical_sobel);
    }
    double g = sqrt((gx*gx + gy*gy) / 8);
    double angle = atan2(gy, gx) * 180.0 / PI;

    direction[floor(angle / 45)] += g;
    // Remember which pixel has what angle
    indices[a].add(pair<int, int>(i, j));
    }

  // Return the edge, contains it's angle and the pixels that have this angle
  edge *e = new edge;
  e->direction = direction;
  e->indices = indices;

  return e;
}

void extrapolate(edges, sides){
    for (side in sides) {
        for (otherside in sides) {
            // Loop over de edges that are in side
            // we can determine this from the "indices" we calculated above
            for (edge in side) {
                for (otherEdge in side) {
                    // Check if the direction of the edge is the opposite of the other
                        edge
                    // this means they can be connected
                    if (edge.oppositeWindDirectionFrom(otherEdge)) {
                        extrapolateMacroblock();
                    }
                }
            }
        }
    }
}
```

Since we were unable to complete this assignment our quality and time measurements are incorrect.

Exercise 3.B

To allow for submacroblock concealing, we first build a list of the macroblock movement vectors of neighboring macroblocks. We then build a list of submacroblock movement vectors that span a whole macroblock, f.ex. if submacroblock size is 8 by 8 pixels, then the submacroblock movement vectors are (0,0), (1,0), (0,1) and (1,1).

For a missing macroblock we do the following algorithm:

```
let mmvs be the list of macroblock movement vectors
let smvs be the list of submacroblock movement vectors

initialize best_mmv to none
initialize best_smv to none
initialize smallest_error to +inf

for each submacroblock in the missing macroblock:
  for each mmv in mmvs:
    for each smv in smvs:
      total_movement_vector = 16*mmv + submacroblock_width*smv
      compute error
      if error < smallest_error:
        smallest_error = error
        best_smv = smv
        best_mmv = mmv

replace submacroblock by submacroblock pointed to by best_smv and best_mmv
```

Exercise 3.C

To allow for missing neighboring macroblocks we had to do one small improvement on exercise 3.B: instead of conceiling missing macroblocks in the order they occur in the frame, we select the macroblock that has the most available neighboring macroblocks and conceil him first.

By doing this, clusters of missing macroblocks will be conceiled from the edges to the center.

Exercise 3.D

Adaptive submacroblock conceiling has been done with a recursive algorithm.
For each missing macroblock (in the same order as 3.C) we do:

```
function conceil_submacroblock(submacroblock):
  if submacroblock_width > 2 pixels:
    split the submacroblock into 4 quarters: q0, q1, q2, q3
    recursivly call conceil_submacroblock on each quarter, resulting in 4 error measures
        e0, e1, e2, e3

    calculate the error measure e when conceiling the full submacroblock

    if e < min(e0, e1, e2, e3):
      conceil the full submacroblock
      return e
    else:
      return min(e0, e1, e3, e3)
  else:
    calculate the error measure e when conceiling the full submacroblock
    conceil the full submacroblock
    return e
```

The error measures e, e0, e1, e2 and e3 are the mean error over the boundary pixels.
The first call to this function is done with a submacroblock size 16 by 16 pixel, the full macroblock size.

2 Measurement results

Test platform

All of our test were run on the following hardware and software:

- **CPU:** Intel I7 2600k quad core @3.9Ghz
- **RAM:** Corsair Vengeance 8GB (2x4GB) CL9 @1600Mhz
- **SSD:** Kingston HyperX 3K 240GB, 555MB/s sequential read, 85.000 IOPS (4K random reads) (used for reading)
- **HDD:** Western Digital Blue 1TB 7200RPM 64MB cache (used for writing)
- **OS:** Windows 8.1 Pro 64-bit

2.1 Quality measurements

First of all we have used the PSNR-metric to measure the quality of our video sequences. The results of this quality measurement can be found in figure 1.

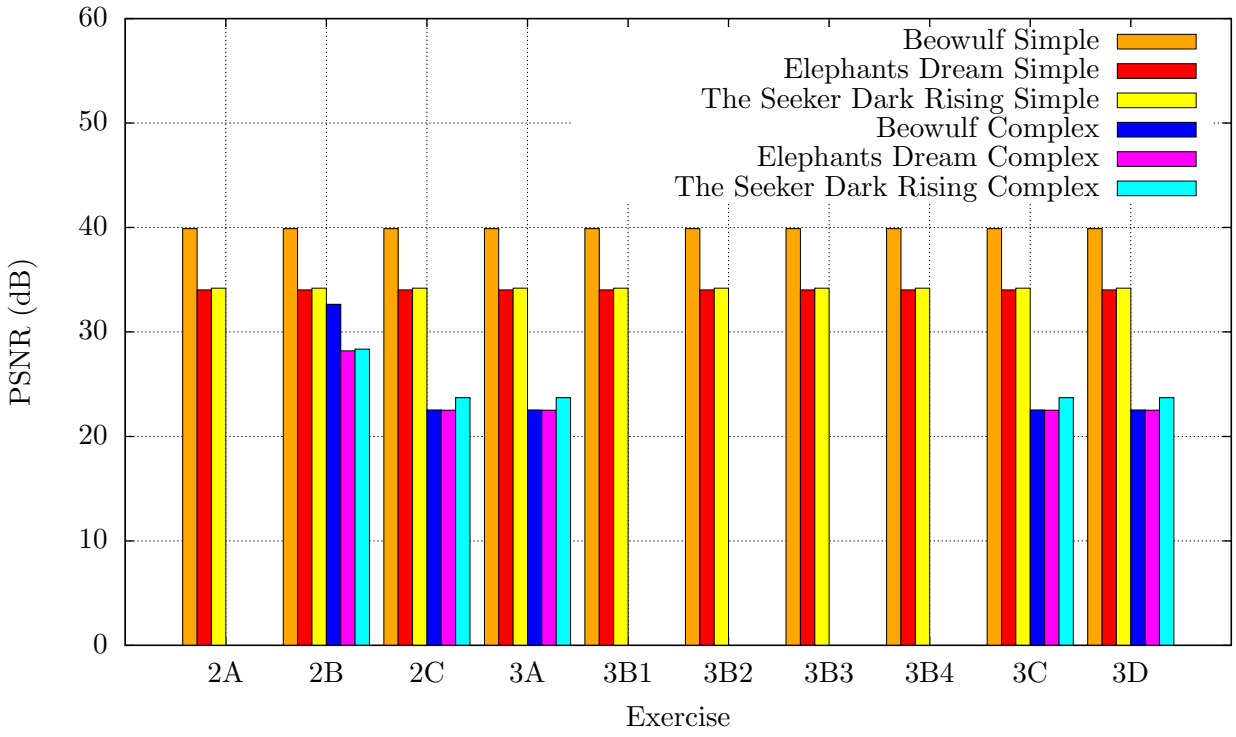

Figure 1: PSNR values for the 3 video sequences, each decoded with their 2 different errorfiles.

The first thing we notice in this graph is the big difference between the PSNR-values of the video sequences decoded with the simple errorpattern and the video sequences decoded with the complex errorpattern. This is rather obvious because the complex errorpattern introduces more errors.
The second thing we notice on this graph is that spacial error concealment implemented in 2B

has a much better quality in comparison with the other methods. This is because our natural fragment is quite static, it doesn't move much. The synthetic fragment and the one unique to our group have a lot of movement which makes the error prediction harder resulting in a lower quality and thus a lower PSNR.

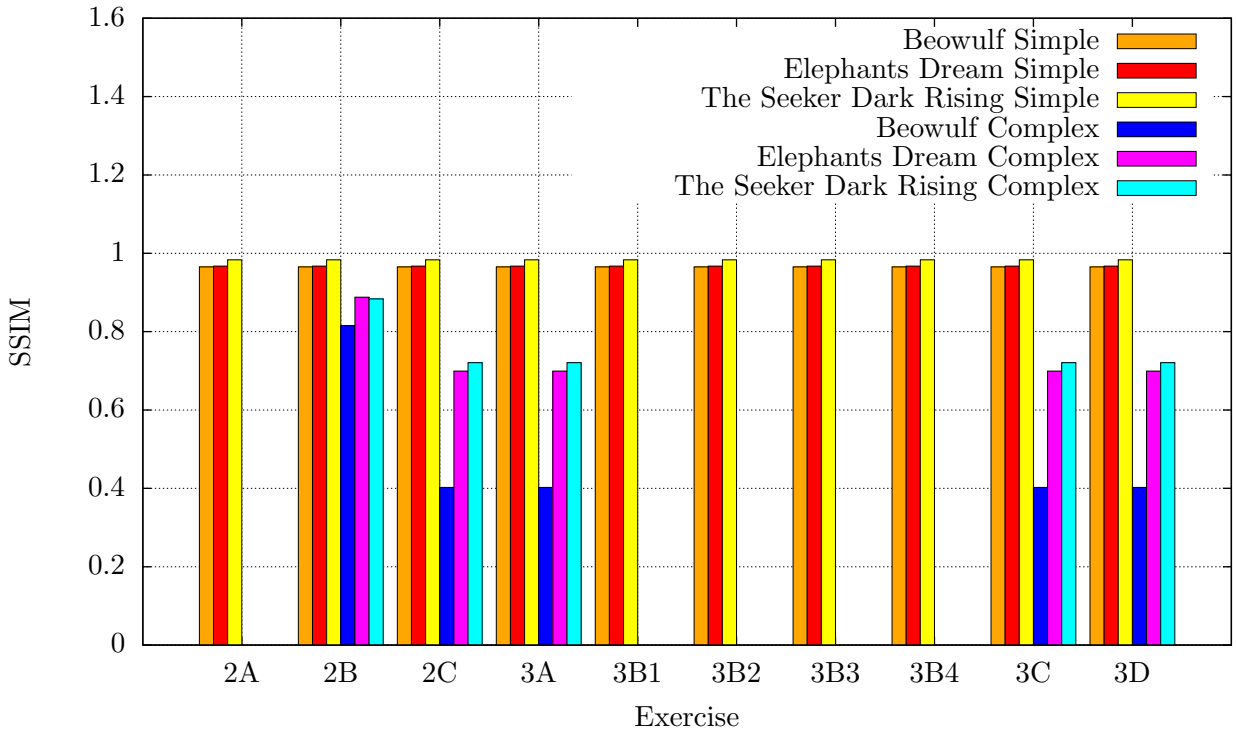

Figure 2: SSIM values for the 3 video sequences, each decoded with their 2 different errorfiles.

2.2 Time measurements

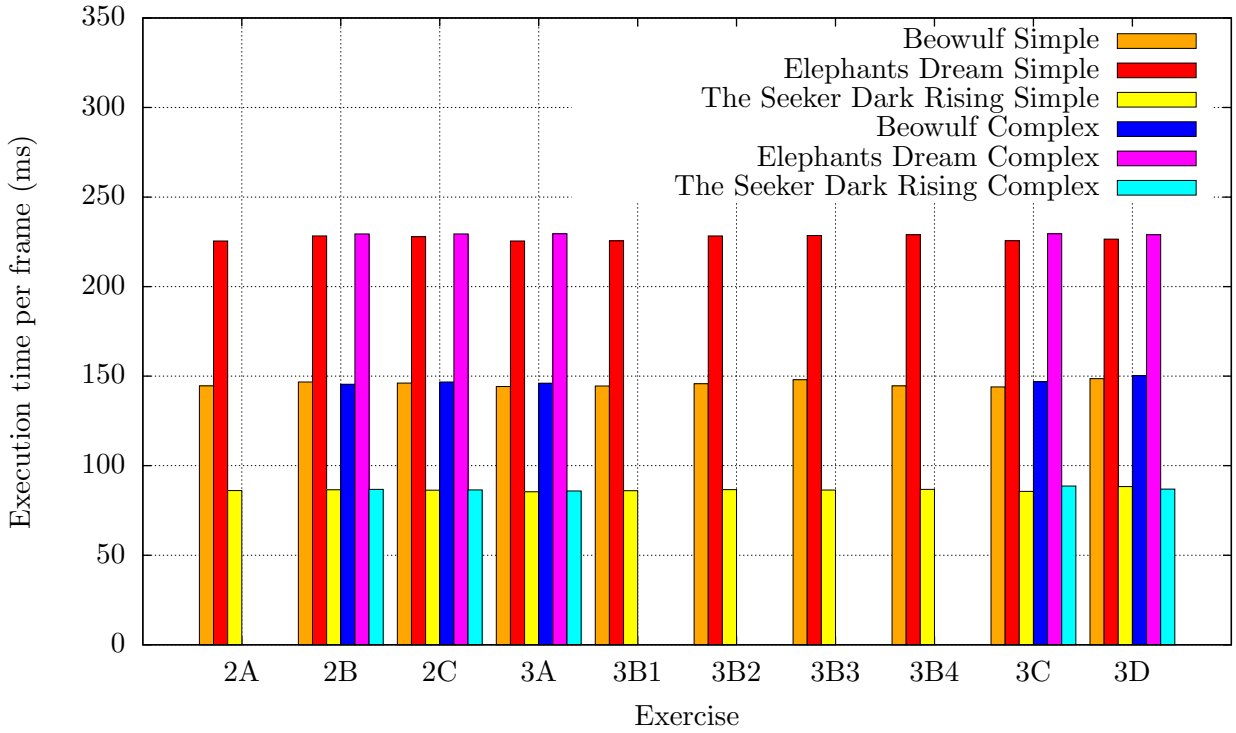

Figure 3: Execution time for decoding the 3 video sequences, each decoded with their 2 different errorfiles.

Complexity reduce in 2A and 2B We have measured the decoding time with concealing method 0 and 1, once without the optimalisation and once with. In tabel you can see the results of this measurement.

		2A		2B	
		4 pixels	2 pixels	4 pixels	2 pixels
Beowulf	simple	145.42	144.59	144.15	145.60
	complex	/	/	145.60	145.66
Elephant	simple	224.51	223.45	224.82	228.30
	complex	/	/	225.22	229.38
Rising times	simple	85.48	86.10	86.05	86.57
	complex	/	/	85.94	86.79

Table 1: Time measurements (ms)

We can see in this table in most cases the computation time goes up for a little bit. Of an algorithmic point of view this is logic because we have to do more checks.

Client-side buffering

The main advantage of client-side buffering is that you can perform (better) temporal error concealment. We have seen in our quality measurement that this will help us to better predict the missing macroblocks, so you will have a better user experience. If you are not doing client-side buffering you can only perform spatial error concealment. The disadvantage is that this buffering requires space.