Implementation Of A Java Applet For Demonstration Of Block-Matching Motion-Estimation Algorithms

Holger Peinsipp

Department of Computer Science IV
Prof. Dr. Wolfgang Effelsberg
Faculty of Mathematics and Computer Sciences
University of Mannheim

October 2003
Supervisor: Dirk Farin

Table of content

1. Introduction........2

2. What is motion estimation........3

2.1 The idea of Motion Estimation........3
2.2 How blockmatching works........5
2.3 Approaches for reducing the computation complexity........6

3. The algorithms........7

3.1 Full search algorithms........7
3.2 One-dimensional full search algorithm (1DFS) [1]........8
3.3 Three-step search algorithm (TSS) 10
3.4 New three-step search algorithm (NTSS) [5]........11
3.5 Four-step-search algorithm (FSS) [2]........12
3.6 Cross-search algorithm (CSA) [3]........13
3.7 Gradient descent search algorithm (GDS)........14
3.8 One-dimensional gradient descent search (1DGDS) [7]........15
3.9 Circular-zone-search algorithm (CZS) [6]........18
3.10 Successive elimination algorithms 20
3.11 Evaluation of computational complexity........21

4. The Java applet........24

4.1 “Step-by-step” mode 25
4.2 “All-at-once” mode........28
4.3 Image set selection........29

5. Index........30

6. Bibliography........31

1. Introduction

Comparing digitally stored video sequences and those “stored” on celluloid and considering the fact that data-storage or data-transmission capacity still is restricted in computer technology, this comparison shows the necessity of compressing the video-data: Using the same approach to store videos digitally as they are in the classic way on celluloid would require at least 25 still images per second. A high-quality 90 minute movie with a resolution of 720*576 pixels and a 24-bit colour depth per pixel could require of over 156 GB of storage capacity. Transmitting this amount of data over the Internet is impractical, especially when real-time performance is needed. This uncompressed video needs a transmission bandwidth of over 237 MBit/s. Similar problems occur when storing the data to disc – only very few memory devices have the necessary capacity.

MPEG-encoding uses - among other compressing techniques - block-based motion-compensation to reduce the memory requirements of video files. This method takes advantage of temporal redundancy between two or more frames. Consider for example a video showing a car driving across the image. The frames will all show the same car - only with a slightly changed position between successive frames. Motion estimation tries to detect similar areas within two frames. It calculates a motion vector describing the movement of these areas from one frame to the next. Instead of storing nearly the same image data twice for these areas, it is only stored once and the video decoder then moves the content of the image along the calculated motion vector.

This study presents a survey of various motion estimation algorithms. A Java applet was developed which illustrates the described algorithms with a step-by-step explanation. Additionally, the computational complexity and matching accuracy are measured, which enables a quantitative algorithm comparison.

2. What is motion estimation

2.1 The idea of Motion Estimation

When using motion estimation for compressing video data, this compression technique uses the fact that usually not the whole image content changes from one frame of a video to the next, but only regions. Often these regions do not disappear from the image, they just change their location within the image - see Figure 1 and 2. With a fixed camera position, a scene like this will show a static non-moving background with almost no changes, and an object – the boat – moving in front of this background. The object itself also changes its appearance only slightly on its way through the scene.

The idea of motion estimation is to detect all objects within an image, compute their motion and represent it by motion vectors. The advantage of this technique is, that the image data for the background and for the objects is stored only in one frame - the following frames contain the motion vectors. A static background then is represented by a zero vector because it has not moved whereas the object movement is stored with non-zero motion vectors, pointing to the new locations of the objects. If the camera position is not fixed, the static background also turns into a moving object.

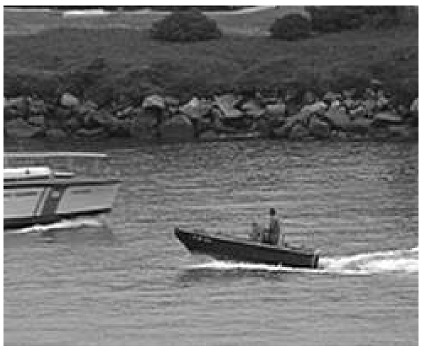


Fig. 1 A boat cruising on a river and a coastguard boat entering the scene.

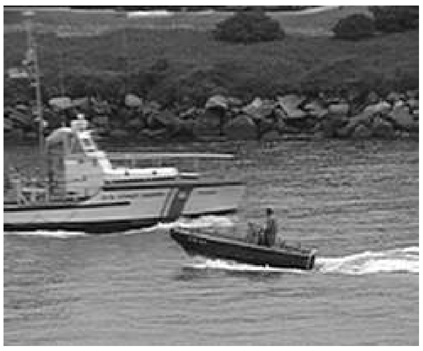


Fig. 2 The coastguard boat has moved towards the centre of the frame. Though the camera is slightly moving, consider the coastguard boat as an object moving in front of an (almost) static background.

As mentioned , the motion of semantic objects in videos should be represented by motion vectors. The problem is to detect the objects and to find their exact boundaries. If you take a car for example: is it a single object? Or do you distinguish between the tires and the chassis? And if you do, what about the driver sitting in the car? It is very hard to define semantic objects automatically. E.g. region growing algorithms only connect regions with similar colour or texture, neglecting the semantic meaning. Typical problems here are scenes with ocean and sky. Since both may have a similar colour. On the other hand, a mountain that is partly covered with snow will be divided into several objects because rough stones and plain white regions do not look very similar. The same applies to trees: why do leaves belong to the branch? Optically they do not have anything in common. These few examples show that it is very difficult – if not impossible – to recognize semantic objects in an image the way the human brain does. Therefore other simple techniques have been invented. MPEG2 uses block matching to relocate a region in another frame. This means that each frame is decomposed into a raster of rectangular blocks. For each of them, motion estimation algorithms try to find a motion vector pointing to its new position in the adjacent frame. Section 2.2 will have a closer look at how this is done. Each block is considered as an object. If many blocks belong to the same semantic object of the scene, they will have the same or very similar motion vectors. While not being as coding efficient as an abject based approach, block based motion compensation does not depend on a robust segment ion algorithm.

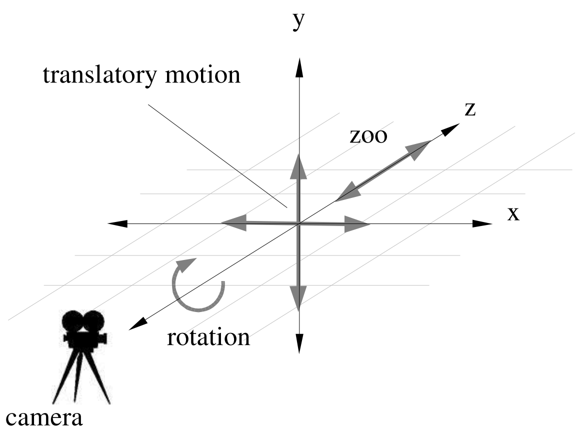


Fig. 3 Different kinds of motion

A further problem is that the motion estimation algorithms do not work properly for all kinds of motion. We distinguish between translatory motion, scaling and rotation (Fig. 3). A translation along the x and y axes also leads to translational motion in the image, while a translation along the z-axis is observed as a change of object size.
Even tough scaling scaling and rotational motion cannot be represented with motion vectors accurately, splitting the object into small blocks with independent motion vectors allows to approximate all kinds of motion.

2.2 How blockmatching works

As explained above, MPEG2 encoders use block matching algorithms to relocate an object in another frame. Therefore the image is segmented into a raster of rectangular blocks of 8 by 8 pixels. Depending on the algorithm used for motion estimation, a block within a certain search range is compared – *matched* – with the source block. In the following, we assume that input images are greyscale only.

Block matching uses a value called "block distortion measure" - *BDM* - to rate the similarity between two blocks. The basic idea is to sum up the differences of the pixel luminances of pixels located at the same position in the two blocks. There are different algorithms to perform this calculation:

Let $c_i(x,y)$ detect the luminance of pixel (x,y) in block i.

The *„Mean Squared Error" (MSE)* uses the sum of squared differences between the two luminance values. The sum is divided by the quantity of the compared pixels to normalize the result:

$$diff=\frac{1}{(n*m)}*\sum_{y=0}^{m-1}\sum_{x=0}^{n-1}(c_1(x,y)-c_2(x,y))^2$$

The *„Sum of Absolute Differences " (SAD)* differs only slightly by using absolute differences instead of squared differences:

$$diff=\frac{1}{(n*m)}*\sum_{y=0}^{m-1}\sum_{x=0}^{n-1}|c_1(x,y)-c_2(x,y)|$$

It is not necessary to normalize the result in the end, since we only need the minimum cost block position and not the absolute cost itself. In fact, it costs an extra mathematical operation per comparison. However, in the applet, we used normalized costs to make the results independent of a specific block size.

Usually, the motion estimation algorithms do not search a whole frame for the best matching block. A search range defines a search area around the coordinates of the source block. It is necessary to understand that the source block is from a frame A, and the search area as well as the resulting best matching block is located in a frame B. Usually these are successive frames in a video.

The most simple search algorithm, *full search* (section 3.1), to find the block with the lowest BDM value within a specified search area, is to match the source block to every existing block in this area. The *search-area* is a rectangular area around the source block. Its size, and therefore the number contained search coordinates, depends on the horizontal search range r_x and the vertical search range r_y,:

$$\text{size of search area} = (2 * r_x + 1) * (2 * r_y + 1)$$

We will consider r_x and r_y always of the same size, therefore we will only use the value r.

Matching all those blocks in the search area to the source block is the computationally most intensive algorithm: For a search range of 10 pixels and also a block size of 8 by 8 pixels, there are $(2*10+1)^2 = 441$ blocks to be matched with 64 operations per block. In total, these are 28,2244 operations for each block.

2.3 Approaches for reducing the computation complexity

As demonstrated in the last paragraph, a full search over the whole search area comes along with a very high computation demand. Though this high demand only occurs once when *en*coding the video, it is essential to reduce it, especially when real-time encoding is required.

Very often objects just move horizontally or vertically or not at all. E.g. in a news broadcast, the anchorman just sits or stands in front of the camera and hardly moves. In this case it would be reasonable to start the block-matching for each block at the same position the block had in the previous frame. Some of the algorithms in section 3 use this assumption of zero movement to increase their performance (e.g. Cross Search Algorithm, New Three Step Search). Prior to searching somewhere else, they try to match the same coordinates again or the area close to it.

Others assume a certain structure of the error surface, which is the surface representing the BDM values over the search area, to locate the minimum. This minimum is the best matching block in the search area.

It is necessary to understand that these algorithms only find blocks that are *similar*, respectively have a low BDM value, to the source block. Very often, it would be impossible to find the exact block again – e.g. due to changes of image brightness. The human eye is not that strict and accepts small blocks which look very similar to the original block. It is a matter of choosing a good threshold value – which defines a block as similar or not – to achieve a good video quality and also a short encoding time.

3. The algorithms

In this section, often a grid is used to present a schematic overview of the search area, as displayed in Fig. 4. The centre of the grid is the coordinate (0,0). Red dots highlight the search coordinates, the respective MEA will compare with the source block. Each grid coordinate represents a distance of one pixel. If new search coordinates are added to the grid, the newer ones are displayed larger than the older ones.

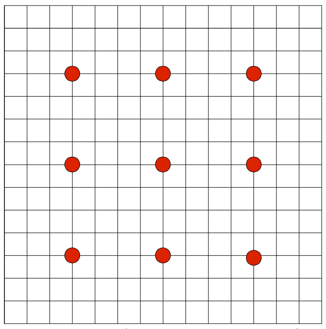


Fig. 4 Grid, representing the search area. Some search coordinates are highlighted

3.1 Full search algorithms

This is a simple MEA that compares the source block with the blocks at every position within the search area. As mentioned above, the calculation costs of this algorithm are very high, but it guarantees to find the optimal block position within the search range.

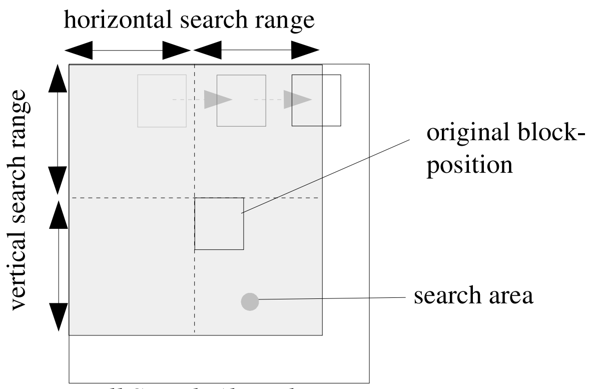


Fig. 5: Full Search Algorithm

The number of comparisons increases quadratically (n^2) with the search range:

The number of comparisons is: $(2r+1)^2 \in O(r^2)$

search range r [pixel]	# compared blocks
5	121
10	441
15	961
20	1681

Table 1 performance of full search

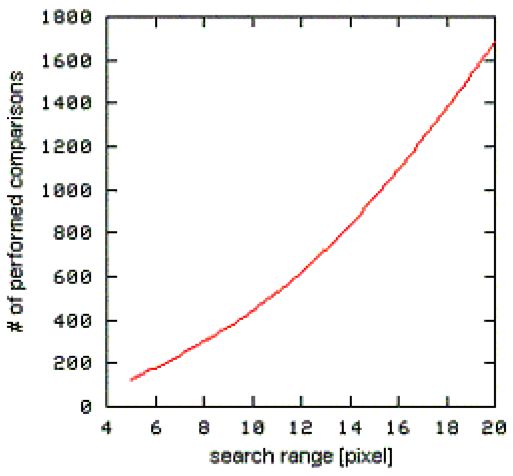


Fig. 6 Performance of full search

3.2 One-dimensional full search algorithm (1DFS) [1]

In contrast to the simple full search algorithm, which through the whole search area, the one-dimensional full search approximates each coordinate in separate steps. Assuming increasing BDM values if the distance from the global minimum increases, some algorithms try to “follow” the gradient downwards to that minimum. By doing this, there is a great risk in being trapped in a local minimum. One dimension full search uses a different approach. Instead of following the gradients, 1DFS starts with searching the minimum for the row (x,0). It continues by searching the whole column of the coordinate with the lowest BDM value. Then these two search processes are repeated with a reduced search range to improve the result.

1. Start with a search pattern of all coordinates (x,0) within the search area and find the coordinate with the lowest BDM value.
2. Continue by creating a new search pattern, consisting of all coordinates that are vertically lined up with the current best matching position. Again search the block with the lowest BDM value.
3. Halve the search range. Proceed like in step 1, this time with the row of the current best matching block.
4. Step 2 is repeated with the reduced search range

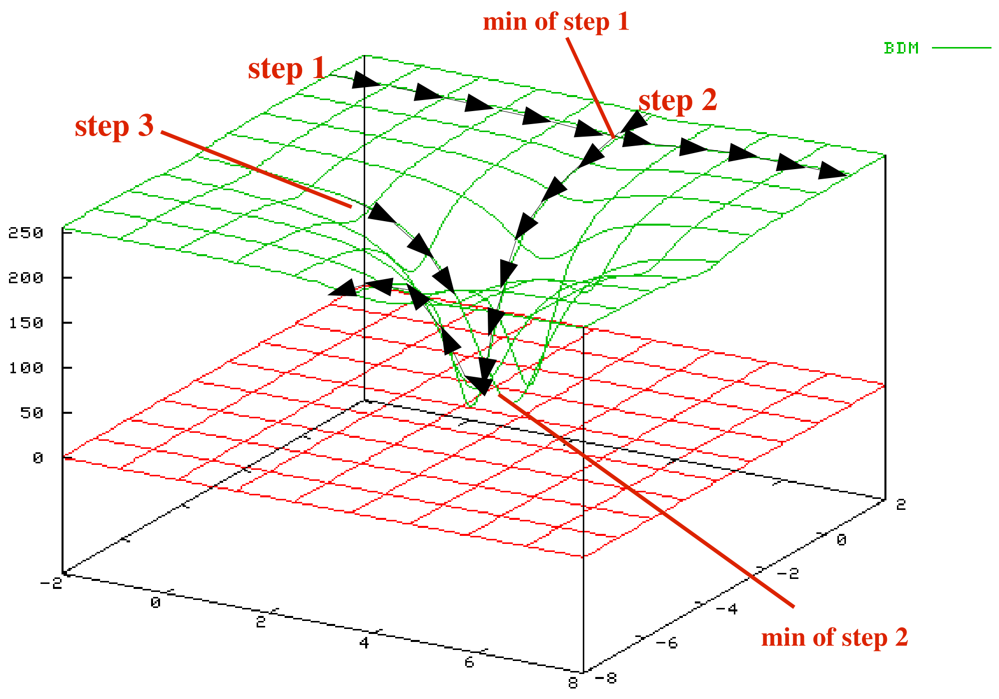


Fig. 7 1DFS in step 3

One-dimensional full search performs faster than simple full search. The number of compared blocks just increases linearly when increasing the search range:

Number of performed comparisons: $2*(2r+1)+2*(\frac{1}{2}r+1)\in O(r)$

search range [pixel]	**# compared blocks**
5	36
10	64
15	96
20	124

Table 2 performance of 1D full search

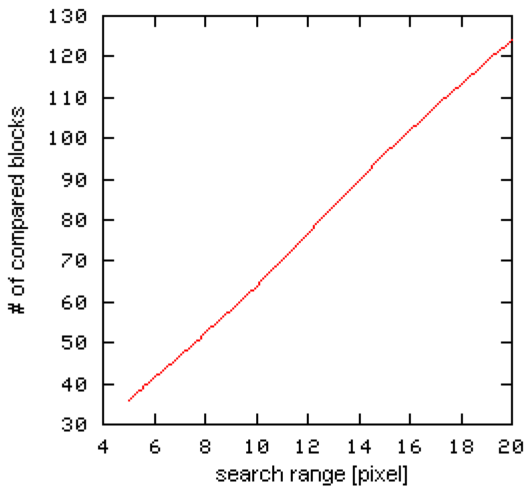


Fig. 8 Performance of 1DFS

3.3 Three-step search algorithm (TSS)

Another fast motion estimation algorithm is the the three-step search. Though powerful in itself, some other algorithms base on TSS and add some enhancements, e.g. NTSS (section 3.4), FSS (section 3.5) and CSA (section 3.6). Like some other algorithms, e.g. the gradient descent algorithms (section 3.7, section 3.8), TSS does assumes increasing BDM values the more the distance to the minimum increases. Starting at the centre, TSS searches for the minimum BDM value using 3x3 search patterns of decreasing size. Each step uses the the current best matching coordinate as centre for its search pattern.

__Note:__ TSS, and all algorithms based on it, will be presented with a fixed search range. In some papers, e.g. [3], the authors use a variable search range, others like [2] use a fixed one. For presentation purposes, a fixed search range is easier to understand.

This is how TSS proceeds:

1. Create a search pattern consisting of 9 coordinates. These are the centre of the search area (0,0) and the 8 coordinates with a horizontal and/or vertical distance of 4 pixels to it. Find the coordinate with the minimum BDM value (Figure 9).
2. Halve the search range to to 2. Create a new search pattern with the eight coordinates surrounding the current best matching coordinate. Again, find the coordinate with the minimum BDM.
3. Reduce the search range again - this time to 1 pixel. Create a search pattern existing of the eight coordinates surrounding the current best matching position. Finally, search for the minimum BDM. This is the coordinate the motion vector will point to.

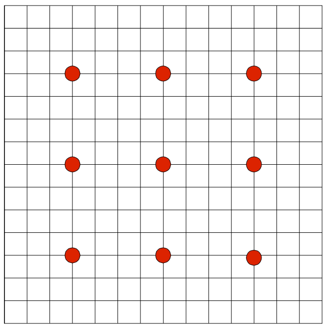


Fig. 9 TSS in step 1

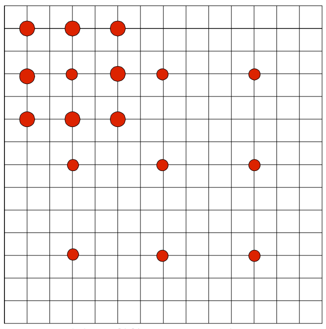


Fig. 10 TSS in step 2

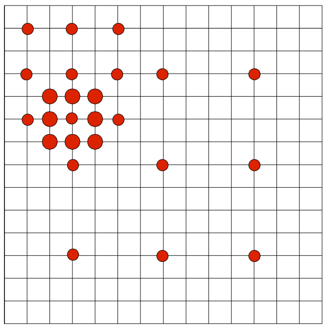


Fig. 11 TSS in step 3

Instead of a global monotonic error surface, TSS assumes a local monotonic surface surrounding the occurring minima. Step by step TSS approaches closer to what it thinks is a global minimum of the search area.

The number of compared blocks is always to 25.

3.4 New three-step search algorithm (NTSS) [5]

The NTSS algorithm is a more centre biassed variant of TSS: in the first step the eight directly adjacent blocks to the centre coordinate are added to the initial search pattern. Two extra features are introduced to enhance TSS. “First-step-stop” will abort the search if after having searched through all initial coordinates, the minimum BDM is still the centre coordinate. “Half-way-stop” aborts the search after step two: if the minimum BDM of the initial 17 coordinates is located at one of the eight coordinates directly surrounding the centre, finally the eight blocks surrounding this coordinate are searched. As a result, NTSS performs faster with stationary or quasi-stationary blocks.

The algorithm in detail:

1. Create the initial search pattern, which consists of the centre coordinate, the eight blocks surrounding it at a distance of one pixel and at a distance of four pixels. (Fig. 12)Search the minimum BDM of this pattern. If the centre coordinate is the minimum BDM, the search is finished (*first-step stop*).
   Otherwise: Go to step 2.
2. If the minimum BDM is one of the eight direct neighbours of the centre coordinate, add the eight direct neighbours of this block to the search pattern and search it again for the minimum BDM. After that, the search is done.
   Otherwise: Continue like TSS: add the four coordinates surrounding the minimum BDM like an “X” at a distance of 2 pixels and search for the new minimum BDM.
3. Search the four coordinates surrounding the current minimum BDM like an “X” at a distance of 1 pixel for the final minimum BDM.

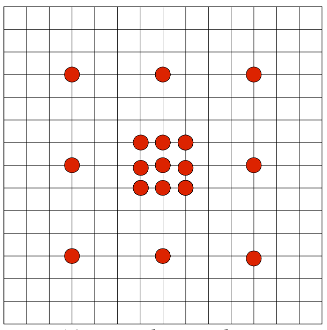


Fig. 12 Initial search pattern of NTSS

NTSS requires 17 comparisons in the first-step-stop condition, aborts the search after 25 comparisons for the half-way-stop, and 33 otherwise.

3.5 Four-step-search algorithm (FSS) [2]

This algorithm is also based on the TSS algorithm. The intention of this algorithm, which the authors Po and Ma describe in [2], was creating an algorithm which achieves better results than the TSS algorithm while having a better worst case performance than the NTSS algorithm. The search range of FSS is fixed to 7 pixels.

Summary of the algorithm:

1) Create a 5x5 search pattern around the centre of the search area, consisting of the nine coordinates surrounding the centre. Find the minimum BDM of these coordinates. If the minimum is located at the centre, proceed with step 4.
   Otherwise: Go to step 2
2) The search pattern still has the size of 5x5. Depending on the location of the current minimum BDM, new search coordinates are added to the pattern.
   a) If the current minimum BDM is located at the corner of the pattern in step 1), add 5 additional coordinates located next to the corner as shown in Figure 13.
   b) If the minimum BDM is located at the centre of a vertical or horizontal axis of the search pattern in step 1), add coordinates to the pattern as shown in Figure 14.

   Search for the new minimum BDM. If its position has not changed after this search, continue with step 4.
3) Repeat step 2 once and proceed with step 4.
4) The search window size is reduced to 3x3 around the minimum BDM. Search all eight new coordinates in this pattern for the final minimum BDM.

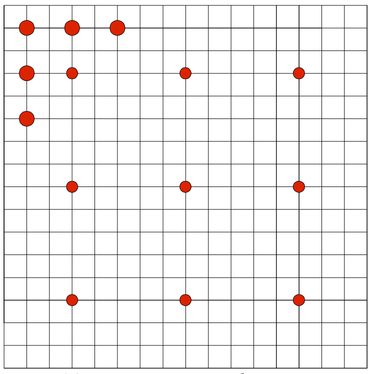


Fig. 13 Min. BDM value was found at the corner of the pattern - FSS adds additional search coordinates surrounding that corner

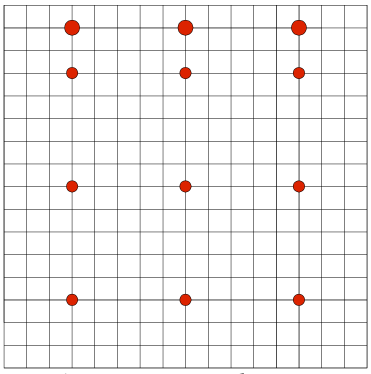


Fig. 14 Min. BDM value was found at the centre of a horizontal axis - FSS adds three additional search coordinates parallel to that axis

The performance of the FSS algorithm varies from 17 (9+8) compared blocks, best case, to 27 (9 + 5 + 5 + 8) compared blocks, worst case. This performance is only slightly worse than the original TSS algorithm with the possibility to perform equal or better in average or best case scenarios. Compared with the NTSS algorithm, FSS performs equal with “first-step stop” (17 compared blocks), which is the best case, it performs equal or better in the average case and definitely performs better in the worst case (NTSS: 33 compared blocks).

3.6 Cross-search algorithm (CSA) [3]

CSA is also one of the algorithms based on TSS though there are some differences. First of all, CSA uses a threshold-based first-step stop. This increases performance for videos or regions of a frame with no motion or plain textures (like plain sky). It also has a reduced amount of initial search coordinates - five instead of nine. These are the centre-coordinate and the four coordinates surrounding the centre like an "X" at a distance of 4 pixels.

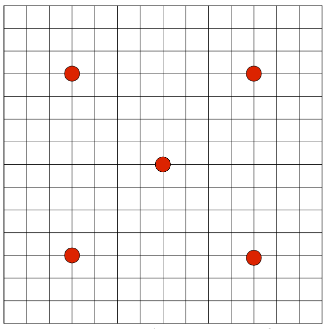


Fig. 15 Initial pattern of

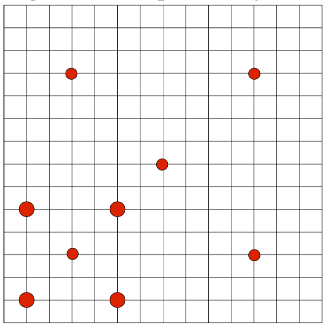


Fig. 16 A new cross is set up around the min. BDM coordinate

1. Match the search block to the centre block (0,0). If the BDM is lower than the predefined threshold, stop the search.
   <u>*Otherwise:*</u> proceed with step 2.
2. Set search range *r = 4*
3. Add the coordinates surrounding the centre at the edges of an "X" at a distance of *r* to the search pattern. Find the minimum BDM.
4. Halve the search range.
5. If the search range is greater than one, go back to step 3.
   <u>*Otherwise:*</u> If the actual minimum BDM position is equal to the previous one or located top-right or low-left to it, add the four coordinates surrounding it like a Greek cross ("+") to the pattern. If it is located top-left or low-right to the previous, add the four coordinates, surrounding it like a St. Andrew's cross ("X") to the pattern.
   Search this pattern for the location of the final minimum BDM.

Cross search needs just one comparison if the bail-out criteria is met. Otherwise it needs 17 comparisons.

3.7 Gradient descent search algorithm (GDS)

The GDS algorithm uses another assumption to perform fast motion estimation. Different from all TSS-based algorithms, GDS assumes that the surface of the function, which represents the BDM values within the search area, contains only a single minimum (Fig. 17). The idea is always to follow the steepest gradient to find the way to the absolute minimum BDM. Therefore, a search pattern consisting of the centre coordinate and the eight direct neighbours are matched to the source block. Afterwards this step is repeated with the minimum BDM coordinate from the previous step as centre. This procedure is repeated until the new minimum BDM value is equal or less than the one from the previous step. A possible variation is that the algorithm also stops searching if the minimum BDM is lower than a certain threshold.

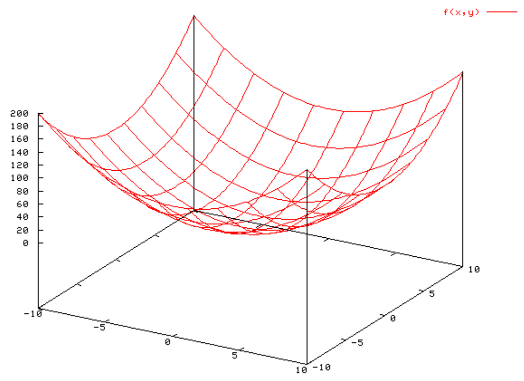


Fig. 17 Error surface containing only a global minimum. GDS will find the minimum

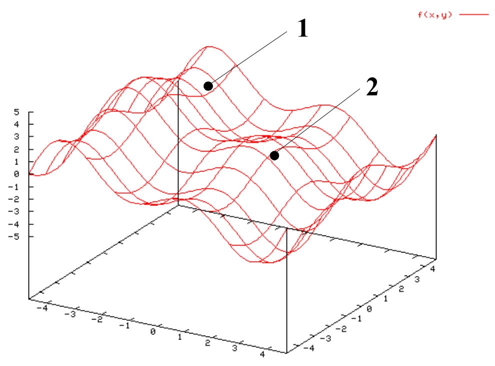


Fig.18 Error surface containing multiple local minima and a global minimum. GDS may find the global minimum

As mentioned above, GDS assumes an error surface with just a global minimum. If there are also local minima, there is a great risk that the algorithm is being trapped in those. GDS can reach the global minimum, but this may depend on the first search coordinate. Referring to Fig. 18, GDS started at position 1 will be trapped in a local minimum. If the initial coordinate is at position 2, GDS will find the global minimum.

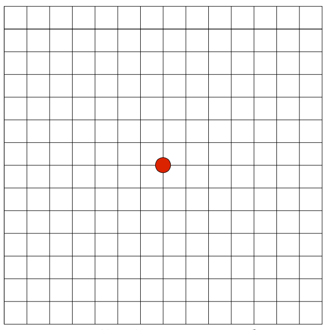


Fig. 19 GDS starts with a initial search pattern containing only one coordinate.

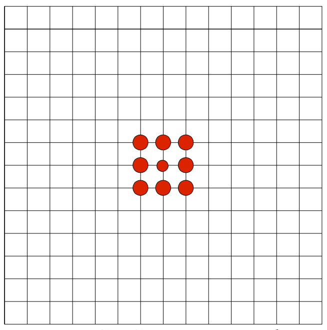


Fig. 20 GDS in step 2: The eight coordinates surrounding the previous min. BDM value are added.

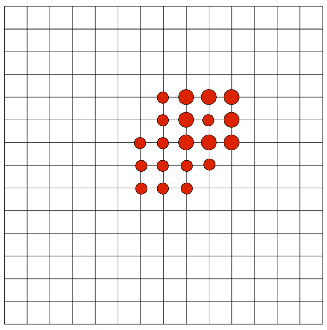


Fig. 21 GDS in step 4.

The performance of GDS cannot be defined clearly because the number of performed comparisons varies very much. The performance depends on the structure of the error surface.

3.8 One-dimensional gradient descent search (1DGDS) [7]

In [7], an improved GDS algorithm is described which reduces the main disadvantage of GDS, the risk of being trapped in local minima. Therefore some changes have been made. First of all, 1DGDS does not have one fixed starting coordinate for the search – GDS always starts at the centre (0,0). 1DGDS uses the up to four motion vectors of prior searches to find a better starting coordinate (Figure 22). In addition to the centre coordinate, the coordinates resulting from those motion vectors are tested for the one with the minimum BDM value. The idea is to start the search closer to the global minimum. This procedure is called initial-point determination and takes advantage of the fact that very often the direction of motion is similar for blocks in the same region.

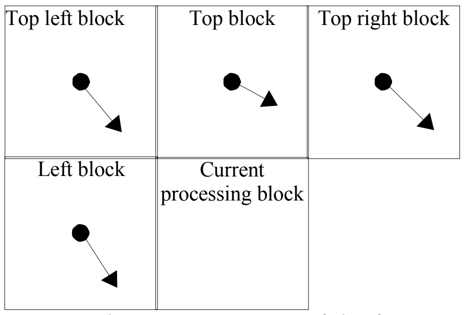


Fig. 22 The motion vectors of the four neighbouring blocks are used to find a better starting coordinate

A further deviation from ordinary GDS is that the search is performed in one direction at a time. Four directions are defined (Figure 23). Each direction provides two new coordinates per search step. Only if the new coordinates in one direction do not lead to decreasing BDM values, the search direction is changed. Also the search range is increased compared to GDS. While GDS only checks the neighbouring coordinates (search range of 1 pixel), 1DGDS makes two search runs, the first one with a search range greater than 1 pixel. During the first search run, the algorithm probably "jumps" over the small valleys of local minima due to its greater search range. In the second run, a more precise search for the global minimum starts in the located area with a search range of 1. The risk of being trapped in local minima is reduced.

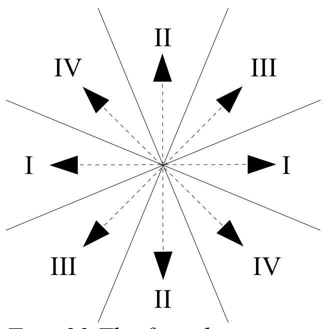


Fig. 23 The four directions of 1DGDS

Respecting the increase in performance that a centre biased orientation may achieve, 1DGDS algorithms stops, if after searching through two directions the coordinate with the minimum BDM value is still the initial one. This reduces the minimum amount of comparisons in case that the initial coordinate is the one with the minimum BDM value.

The following paragraph will show how 1DGDS works in detail:

Let *d=4* be the step size for the first search run.

1. Set search direction = 1 (Figure 23).
2. Match the source block to the coordinates, which the motion vectors of the neighbouring blocks point to, as well as to the centre coordinate (0,0). The one with the lowest BDM value will be the initial coordinate for the search and is the current processing point.
3. Add the two coordinates lined up with the current processing point at distance *d* in the search direction.
4. Calculate the BDM value for the new coordinates. If the coordinates with the lowest BDM value is one of the two new coordinates, this is the new current processing point. Repeat step 3.
   Otherwise: If search direction = 2 and the coordinate with the minimum BDM is still the initial coordinate, stop the search. The final motion vector then points to this coordinate.
   If search direction < 4: increase search direction by 1. Repeat step 3.
   If search direction ≥ 4: set search direction = 1. Set d=1.
5. Do the same thing as in step 3 but this time with d=1.
6. Calculate the BDM value for the new coordinates. If the coordinate with the lowest BDM value is one of the two new coordinates, this is the new current processing point. Repeat step 5.
   Otherwise: If search direction < 4: increase search direction by 1. Repeat step 5.
   If search direction ≥ 4: The final motion vector points to the current processing point.

Since the proceeding of this algorithm is not very easy to understand, it is also illustrated as a flow diagram in Figure 24 on the following page.
Again it is not easy to determine what the performance of 1DGDS will be. The optional search for a good initial search coordinate costs up to four comparisons. The bail-out-criteria may stop the algorithm after another five comparisons. If not, the final number of compared blocks depends on the image content and motion.

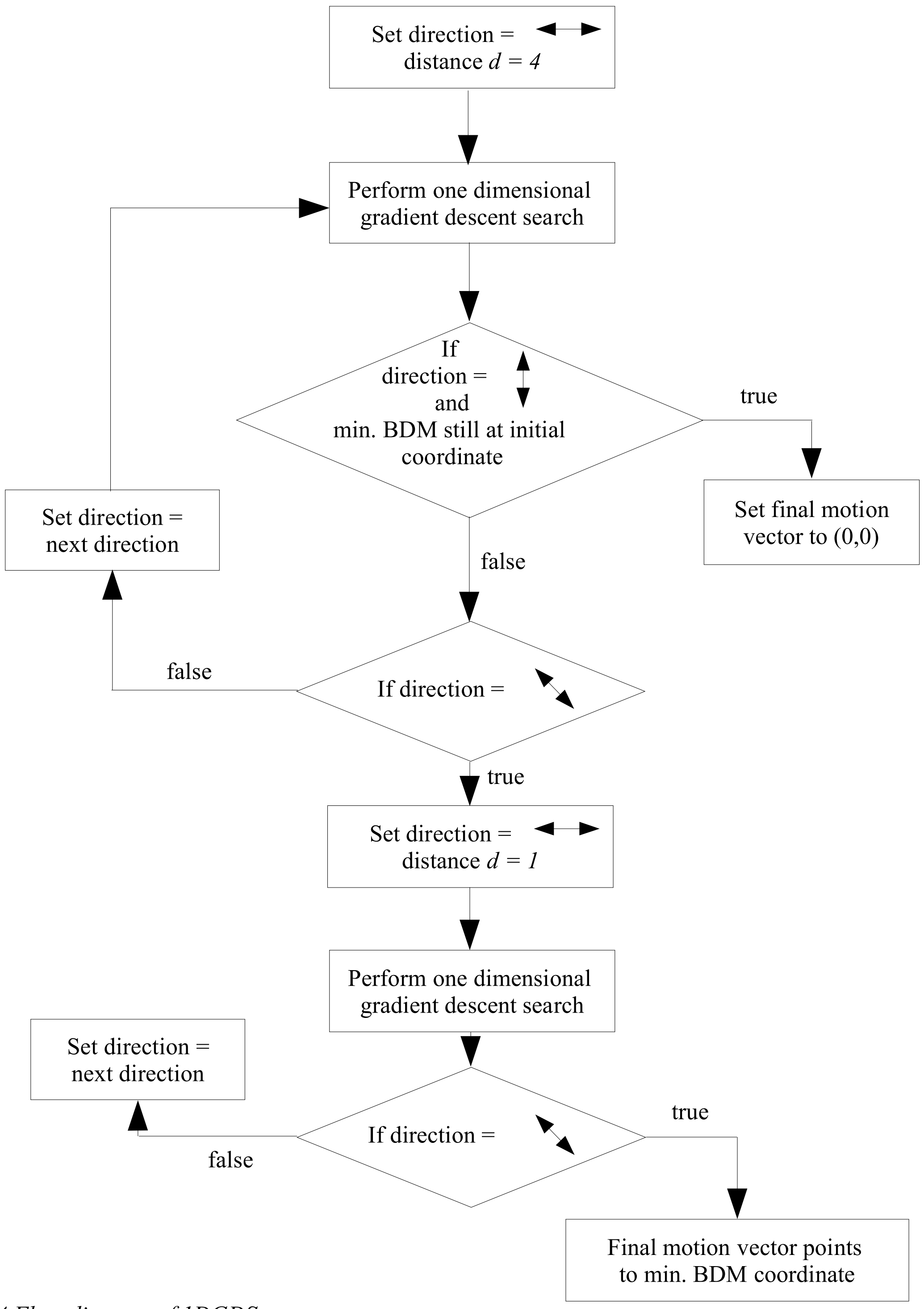

Set direction =
distance d = 4
Perform one dimensional gradient descent search
If direction = and min. BDM still at initial coordinate
true
Set final motion vector to (0,0)
false
Set direction = next direction
false
If direction =
true
Set direction =
distance d = 1
Perform one dimensional gradient descent search
Set direction = next direction
If direction =
true
false
Final motion vector points to min. BDM coordinate

Fig. 24 Flow diagram of 1DGDS

3.9 Circular-zone-search algorithm (CZS) [6]

While the previously presented algorithms only try to find a motion vector to a best matching block, CZS goes one step further. Keeping in mind that data reduction is the main target of motion estimation, the design of CZS favours short motion vectors. The idea is that longer motion vectors, e.g. those pointing to the edges of the search area, need more bits when encoded than short ones. CZS searches circular zones around the centre of the search area, beginning with the innermost (Figure 25). A threshold value defines the desired minimum quality of the block to be searched, by setting a minimum BDM value. The first block found which under-runs this threshold is the one the final motion vector will point to, though there might be blocks with a lower BDM value within the search area. Due to this behaviour, shorter motion vectors are preferred to longer ones.
The setting of this threshold also affects the performance of the algorithm and the quality of the encoded video. A higher threshold leads to better matching blocks but will keep the algorithm searching for a longer time, into the outer regions of the search area.
Motion very often is homogeneous for an area of the frame. Its direction very likely does not change abruptly form one block to the next. Taking advantage of this fact, CZS starts by using the motion vector, $MV_{predicted}$, of the previous block. A small area around the resulting coordinate is searched for a good matching block. Again a threshold BDM value will decide whether a block is good enough or not.
Preferring short motion vectors makes this algorithm clearly centre biased. Though this is not optimal in general, a centre-biased algorithm has its advantages, e.g. in video-phoning or -conferences. In those cases, mostly faces are filmed which very likely do not perform sweeping movements. The best matching block usually will be found near the centre of the search area. Focussing the search on this smaller area leads to a better performance and a faster encoding time, which is needed for real time encoding in video-phoning.

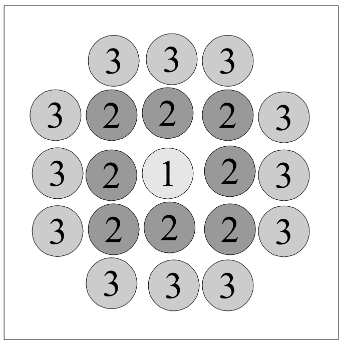


Fig. 25 Definition of circular zones around the centre.

CZS builds up circular search zones around the centre (see Figure 25). The zones are constructed using the following formula:

$$\text{round}\left(\sqrt{MV_h^2+MV_v^2}\right)=r-1$$

where MV_h and MV_v are the distance of the currently examined block to the centre and “r” is the corresponding zone. In order to describe the algorithm, the following symbols are used:

MV_x	motion vector
T_1, T_2, T_3	predefined threshold values. $T_2 < T_3$
M	number of circular zones around the predicted coordinate

N	number of circular zones around the centre of the search area. Usually M < N.
i	counter for the circular zones

1. If the current source block is the first one examined in this frame, set $MV_{predicted}$ to (0,0). Then proceed with step 5.
   <u>*Otherwise:*</u> Set $MV_{predicted}$ to the motion vector of the previous block.

Circular search around predicted motion vector

2. Construct M circular zones around $MV_{predicted}$. Set i = 1.
3. Compute the BDM value for each block within the circular zone i around $MV_{predicted}$.
4. If the minimum BDM found is less than T_1 , proceed with step 10.
   <u>*Otherwise:*</u> If i<M, set i = i+1 and go back to step 3. If not, proceed with step 5.

Circular search around (0,0)

5. Construct N circular zones around (0,0) in the search area. Set i=1 and LAST = false.
6. Compute the BDM value for each block within zone i around (0,0).
7. If the minimum BDM value found so far is less than T_2 or LAST = true, proceed with step 10.
8. If the current minimum BDM is greater than T_2 but less than T_3, set LAST = true.
9. If i<N set i=i+1 and proceed with step 6.

Final step: Use MV found

10. The motion vector of the block with the minimum BDM value is chosen.

The performance of CZS is hard to determine. Many factors affect the number of compared blocks: The search range, which defines the size of search area, the number of circular zones around the predicted motion vector (the M value) and the threshold values, which determine the moment when the algorithm bails out of the search before reaching the last unsearched coordinate.
As mentioned above , this algorithm is highly specialized to encode videos with little motion, fast. The authors point out that CZS works best with “lower bit rates, which are the bit rates of interest for video conferencing” and that it “does not perform well in video sequences with large objects or camera motion” ([6]).

3.10 Successive elimination algorithms

This work has presented only a few of the existing motion estimation algorithms and thereby has focused on those that make heuristic assumptions to reduce the number comparisons. Another approach to increase performance is to optimize the "Full Search" algorithm. Though it has the worst performance (see section 3.11) , the quality of the located blocks still is the best of all presented algorithms.

The idea is to find a quick test to eliminate as many regions as possible from the search area. Multilevel successive elimination algorithms like [8] take this approach. They aggregate the information of the single pixels by summing up the colour values of neighbouring pixels. For example, the colour values of the pixels of 2x2 square are summed up and are represented by a new field. These fields are aggregated again – that is what is meant by "multilevel" (Fig. 26). Each of these fields represents a greater number of fields on the level below. A possible starting point for the search in the multilevel architecture may be using the motion vectors from adjacent, previously searched blocks. This vector points to a region within the search area which also is represented by an aggregated value on a higher level. At this point the algorithm uses the "Schwartz Inequality" to eliminate regions from the search area. Referring to this inequality, the aggregated value on a higher level can be equal or higher than the sum of single values it represents on the levels below. It never can be lower. By this fact, the value resulting from the first guess eliminates all regions with a higher aggregated value. It is necessary to understand that this elimination will never affect a possible better matching block than the one found. This procedure is repeated successively for the different levels. In the end only a few blocks "survive" the elimination process and are matched. The number of compared blocks is drastically reduced though a full search is performed.

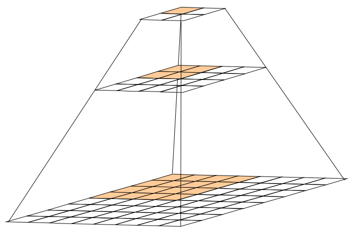


Fig. 26 Pyramid, representing the multiple aggregation-levels for the BDM values in successive elimination algorithms

3.11 Evaluation of computational complexity

This paragraph presents an overview of how the different algorithms perform with the example frames from the applet. When possible, a search range of ten pixels is chosen, and if a threshold value is needed, it is set to 12.

The algorithms have been tested with frames of the video sequences “Stefan” and “Coastguard”. In the “Stefan” sequence the search ranges for some regions are to small to find similar blocks. This is a result of the fast movement of the person in the scene, and 5 missing frames between the two presented. Those frames have been left out for presentation reasons. The higher average BDM values lead to a generally lower performance in the comparison. As a result, all algorithms have a better performance in the “Coastguard” sequence.

It is not possible to compare all algorithms under the same conditions. Most of the TSS-based algorithms have a fixed search range, others have variable search ranges. Even the fixed search ranges differ. For some algorithms, the search range is of great importance, other algorithms depend more on a threshold value. If the threshold value in FTS is set to a high value, e.g. 50 or higher, the search will stop almost immediately,but if set to zero, it will run through all search coordinates - which is the worst case scenario. This is also a difference of the algorithms: running through all coordinates may be common for some algorithms, for others it is the worst case. In spite of these differences, the algorithms perform nearly comparable tasks. The data of table 3 and 4 gives an overview on how well each of them performs.

Algorithm	Avg. min. BDM	Avg. # comparisons per block	Total # of compared blocks
Full Search	17.6230	441	338,688
1DFS	18.6982	61	47,044
TSS [1]	19.9489	27	20,736
NTSS [1]	19.9739	30	23,802
FSS [1]	20.6595	22	17,188
GDS	21.1903	31	24,133
1DGDS	19.7601	19	15,604
CSA 2	20.4977	14	11,368
CZS	18.4462	238	183,002

Table 3 Sequence: Stefan

(1) search range = 7 pixel
(2) search range = 8 pixel

Searching for a value representing both variables, the average quality of the results and the average number of compared blocks I chose one based on the product of quality and performance. The optimum is a minimum of both variables – the more the value of each variable increases, the worse the general performance of the algorithm becomes.

Algorithm	Avg. min. BDM	Avg. # comparisons per block	Total # of compared blocks
Full Search	8.4202	441	432,180
1DFS	8.7377	60	59,695
TSS [1]	12.5873	27	26,460
NTSS [1]	12.1617	30	29,710
FSS [1]	11.9091	21	21,257
GDS	11.6735	28	27,986
1DGDS	9.8181	18	18,141
CSA [2]	12.4502	12	11,960
CZS	10.4940	151	18,141

Table 4 Sequence: Coastguard

(1) search range = 7 pixel
(2) search range = 8 pixel

Due to the different search ranges of some algorithms, the size of the search areas differs and therefore the average number of searched blocks may be affected. To get comparable values, the number of compared blocks has to be normalized. This is done by dividing it by the total amount of blocks within the search area. The resulting value represents the percentage of blocks compared of the search area.

The average BDM value is normalised by dividing it by 255 – the maximum BDM value. This only is done to get a fully normalized indicator with a maximum value of 100. This is the formula of the indicator:

$$i = \frac{BDM}{255} * \frac{c}{(2*s+1)^2} * 100$$

where:

bdm	=	*the average BDM value*
c	=	*the average number of compared blocks*
s	=	*the search range*

See Figure 27 for the resulting performance indicators of the algorithms for the image sets of the applet.

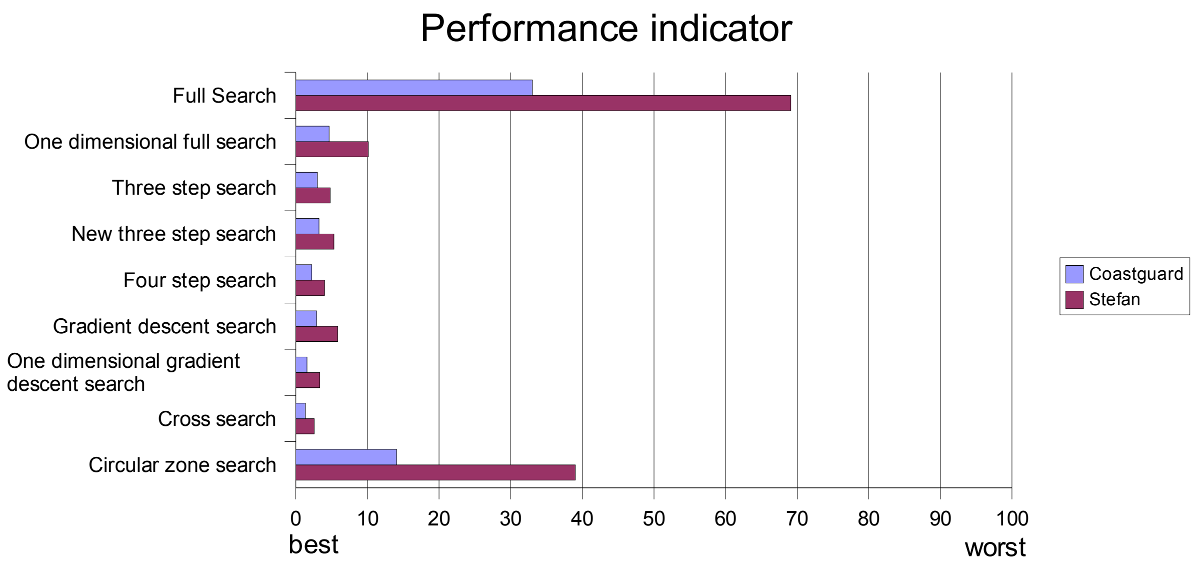


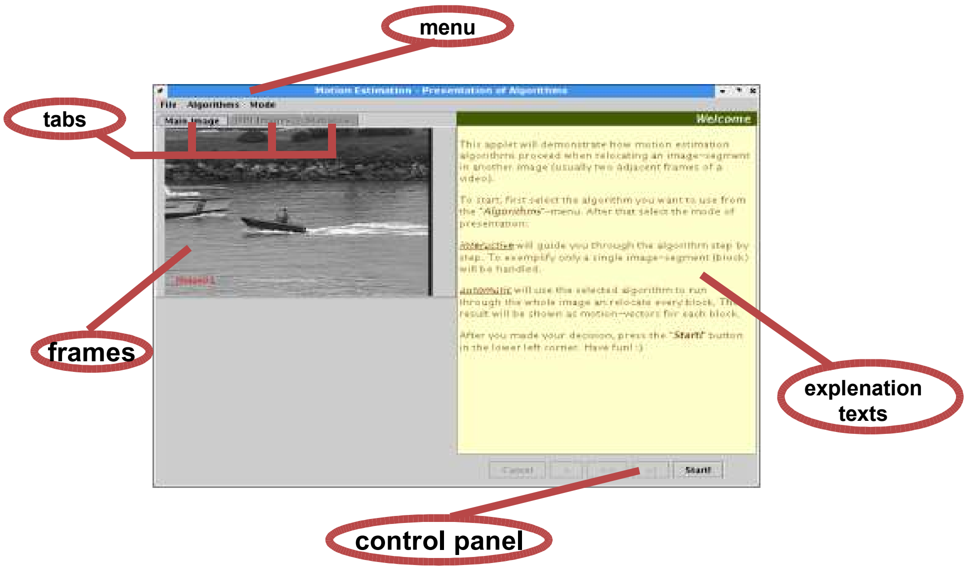
Fig. 27 Performance indicator for MEAs

The 1DGDS algorithm and CSA perform best for these two examples. Full search has a really bad performance. The costs for finding the best average BDM value do not pay off in comparison with the other algorithms. The CZS algorithm also has a bad performance. As mentioned in 3.9 it was designed to perform fast in video-conferencing situations with average slow object movement. Referring to the results of the comparison, using heuristic assumptions to increase performance of motion estimation algorithms pays off. Using the results from full search as a reference, the average BDM values of the other algorithms mostly are only slightly worse though the number of compared blocks is reduced enormously. Though the performance indicator presents “Cross Search” as the best algorithm - the example of CZS shows that there is not one best algorithm, it depends on the kind of video which algorithm performs best.

4. The Java applet

The Java applet belonging to this work is written with the *JDK 1.4.1* from *Sun Microsystems* (http://java.sun.com). Required for the execution of the applet is the JDK 1.3.x.

The applet presents the proceedings of the algorithms in two different modi: "*step by step*" or "*all at once*". Two sets of images from videos, each consisting of two frames, support the presentation. The applet starts when visiting the designated URL and will display a welcome window while loading. Please make sure that one of the recommended JDKs or SDKs is installed and that your browser has the right to execute Java programs.



This applet will demonstrate how motion estimation algorithms proceed when relocating an image-segment in another image (usually two adjacent frames of a video).

To start, first select the algorithm you want to use from the "*Algorithms*"-menu. After that select the mode of presentation:

interactive will guide you through the algorithm step by step. To exemplify only a single image-segment (block) will be handled.

automatic will use the selected algorithm to run through the whole image and relocate every block. The result will be shown as motion-vectors for each block.

After you made your decision, press the "*Start*" button in the lower left corner. Have fun! :)

Fig. 28 The main window

After the loading procedure the main window will appear (Figure 28). In the top left corner, the first frame of the current image set id displayed. The number of the frame is written on it. Switch between the two frames by left-clicking on the image to. The different tabs will provide more detailed information of the comparisons when all required settings are done. In the beginning, only the main window tab will be activated.

On the right, a text area is displayed. Explanations to the algorithm or hints for the settings will be provided here, depending on the selected mode, algorithm and the current step. Open the algorithms menu and select the algorithm of your choice. (Figure 29)

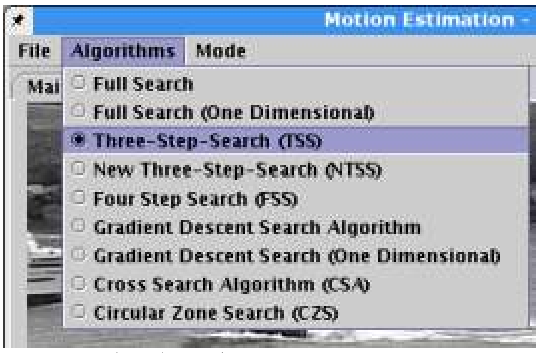

Motion Estimation -
File
Algorithms
Mode
Mai
Full Search
Full Search (One Dimensional)
Three-Step-Search (TSS)
New Three-Step-Search (NTSS)
Four Step Search (FSS)
Gradient Descent Search Algorithm
Gradient Descent Search (One Dimensional)
Cross Search Algorithm (CSA)
Circular Zone Search (CZS)

Fig. 29 The algorithms menu

Changing the algorithm, the selected mode or the image set can only be done before pressing the "*Start*" button or after the selected mode has finished. A mode can always be stopped by pressing the "*Cancel*" button.

4.1 "Step-by-step" mode

Select the "*step-by-step*" mode from the mode menu. This mode will guide you through the settings that need to be done for this algorithm and will show the ongoing search for the block with the minimum BDM in detail. The user will get a good impression of how the single algorithm works. At the same time detailed information is provided in the text area.

4.1.1. Displaying the motion

Press the "*Start*" button to begin the presentation of the selected algorithm. At first, the program will propose to click on the frame presented to switch between the two frames of the selected image set. This provides a good impression of what motion takes place from one frame to the next. This is what the algorithms have to calculate with their different search strategies.

4.1.2. Selecting the search block

The next step orders the user to select a block within frame 1, which the algorithm should try to relocate. The selection is done by simply clicking on the frame in the upper left corner. A red rectangle will mark the selection.

4.1.3. Setting the search range

The following step may ask the user to select a search range. Therefore frame two is presented in the upper left corner. A red cross appears in the frame. The coordinates of the cross are the same as those of the upper left corner of the selected block in the previous frame though it may mark a different content in the frame now. This is because the the region marked in frame 1 may have moved in frame 2. Move the mouse over the frame while keeping the left mouse button pressed to drag a rectangle around the selection which marks the search area. The selected search range will be displayed. Not all algorithms have an adjustable search range. When one of these algorithms is selected, the step is skipped.

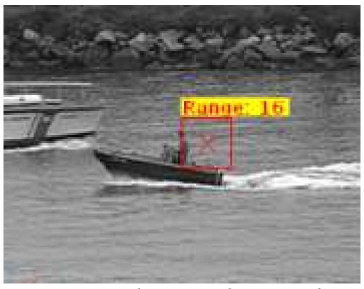


Fig. 30 set the search range by dragging the mouse

4.1.4. Setting a threshold value

In the next step, a threshold value has to be set. Some algorithms need such a value to stop the search before having compared all designated coordinates. Usually this is some sort of bail-out criteria to reduce the search costs. The lower the threshold value is, the more unlikely it is that the BDM value found is lower than the threshold. The maximum value is 255, which would e.g. be a totally white block if you are searching a totally black one. The minimum value is 0, e.g. if the algorithm finds exactly the block it searches for. Blocks that look very similar have a BDM value of up to 15. A value of 50 is bad already. Use the slider appearing in the lower left corner to set a threshold value of your choice. If the selected algorithm does not require a threshold value, this step is skipped.

4.1.5. Presentation of the algorithm

With this step the "Diff image" tab and the "Statistics" tab are activated. The program automatically switches to the "Diff image" tab. Also a raster appears in the lower left corner. In the control panel, the "*Next*" button is disabled temporarily and three new ones are activated:

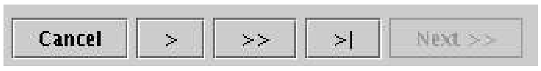


Fig. 31 The control panel with activated forward-, fast-forward- and perform-all-button

">"	Forward button - Press this button to perform the next comparison
">>"	Fast forward button - Press this button to perform the next ten comparisons
">|"	perform-all-button - Press this button to perform all comparisons

The grid (Figure 32) presents a schematic overview over the search area. Each grey dot represents a coordinate within the search area and by that the upper left corner of every possible block to be searched. The red dots represent the blocks designated to be matched to the source block by the algorithm. A black cross marks the position that have already been searched. The current search position is marked by the green dot, the yellow dot marks the block with the minimum BDM value so far. Sometimes the initial set of search coordinates is extended by new ones after some performed comparisons. This will also be displayed in this raster.

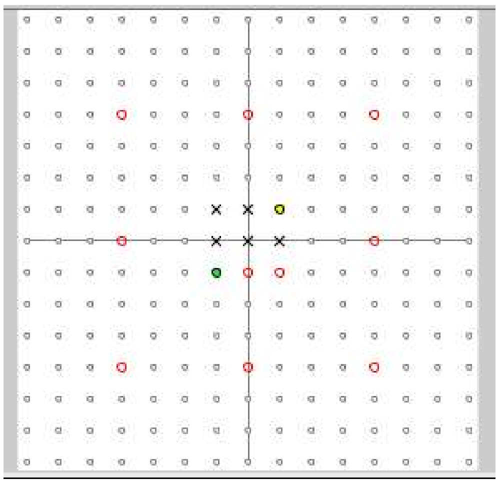


Fig. 32 The grid displaying the new three-step-search

The "Diff Images" tab displays a zoomed region of one of the original frame, showing the search area to present a better overview over the ongoing comparisons. A red rectangle will show the current search position while a yellow rectangle will mark the block with the minimum BDM value so far. The costs for the last performed comparison, which is the BDM value, is displayed below this image. Beneath, the source block which has been selected in step 2 is displayed together with the current search block and a difference image. The difference image displays the differences of the colour value of each pixel of the source block and the current search block., e.g. regarding the pixels at coordinate (0,3) in the two blocks. The pixel in the source at this coordinate block has a value of 225 (a light grey, nearly white) and the one in the current search block a value of 200 (a somewhat darker grey). The difference value is 25. The pixel at coordinate (0,3) in the difference image will have the value of 25, which is a very dark grey. The darker the pixels of the difference image are, the more equal are the two blocks .

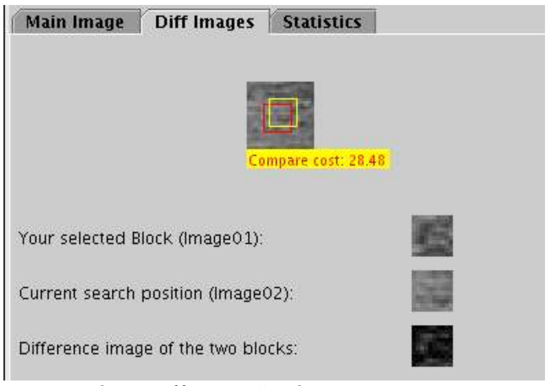

Main Image
Diff Images
Statistics
Compare cost: 28.48
Your selected Block (Image01):
Current search position (Image02):
Difference image of the two blocks:

Fig. 33 the "Diff image" tab

The “Statistics” tab provides more data about the search:

search range:	the selected or predefined search range
No. of compared blocks:	total of compared blocks so far
Avg. no. of compared blocks:	only activated in the “*all-at-once*” mode
min costs occurred:	the minimum BDM value so far
max costs occurred:	the maximum BDM value so far
current optimal position:	the coordinates of the block with minimum BDM value found
Mean distance :	only activated in the “*all-at-once*” mode

Use the buttons of the control panel to perform the search. The next button will be reactivated automatically after the last coordinate has been searched.

4.1.6. The result

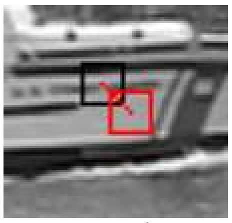


Fig. 34 A red rectangle marks the block located in the displayed frame, a black rectangle marks the coordinate of the other block in the second frame. The red line is the resulting motion vector

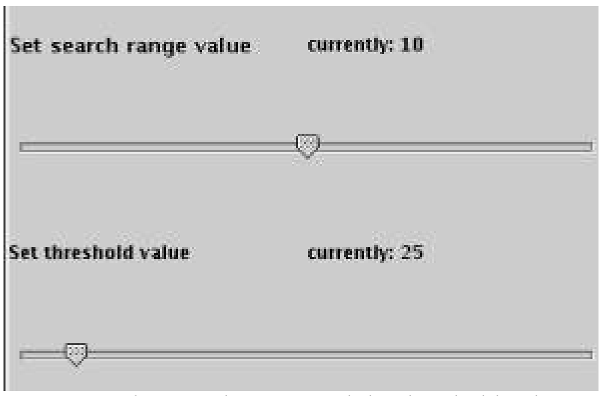
The last step presents the result of the search. The program automatically switches to the “Main image” tab while the other two tabs remain activated. In the frame the selected source block is marked again by a rectangle as well as the block with the lowest BDM value. By clicking on the frame you can still switch between the two frames of the set. As your selected block is located in frame one, the block with the minimum BDM value is in frame two, so the block located in the displayed frame is marked with a red rectangle and the one in the other frame is marked black. A red line represents the motion vector which the algorithm has calculated. This is the vector that is stored in the MPEG-2 file for the source block when encoding the video.

4.2 “All-at-once” mode

This mode was designed to present the work of the algorithms on a whole frame instead of a single block like in the “*step-by-step*” mode. It is less detailed in its description but presents a better overview of the results. For each block in frame 1 a best matching block in frame two is searched. A good video encoder may possibly not store each block as a motion vector with regards to the video quality if the BDM value of the resulting block is too bad. The “*all-at-once*” mode was programmed regardless of this option.

4.2.1. The settings

As explained before some algorithms need the setting of a search range. The selected algorithm will try to relocate each block within a search area resulting from this range. Also some algorithms stop the search if they find a block with a BDM value lower than a certain threshold. These two values, the search range and the threshold can be set in the lower left corner with the two sliders. If the selected algorithm has no variable search range or threshold value, the corresponding slider will be deactivated. Press the “*Next*” button to start the algorithm. Be aware that, depending on the selected algorithm and the processor power of your computer, this action may take up to a few minutes.


Set search range value currently: 10

Set threshold value currently: 25

Fig. 35Set the search range and the threshold value for the algorithm

4.2.2.The results

The results of the search will be presented as motion vectors in the frame in the upper left corner. Each block of the image now has a motion vector, represented by a red line, which points to the location where the best matching block within the search area has been found. If there is no motion vector is displayed for a block, this block has been relocated at the initial coordinate (0,0), which means it has not moved.

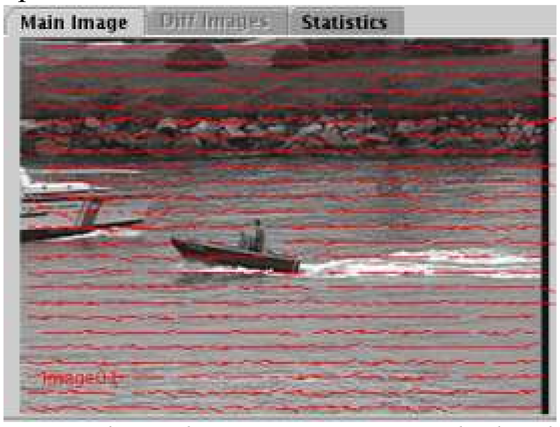


Fig. 36 The resulting motion vectors are displayed in the frame

The “Statistics” tab is also activated. It provides more data about the search:

search range:	the selected or predefined search range
No. of compared blocks:	total of compared blocks so far (for all blocks)
Avg. no. of compared blocks:	the average number of searched coordinates per block
min costs occurred:	the absolute minimum BDM value
max costs occurred:	the absolute maximum BDM value
current optimal position:	only activated in the "*step-by-step*" mode
Mean distance :	the average minimum BDM value found per block

4.3 Image set selection

Before starting the selected algorithm or after finishing it, the set of images can be changed. Select this option from the file menu. A new window will appear which offers different image sets. Select one of the sets and press "Ok".

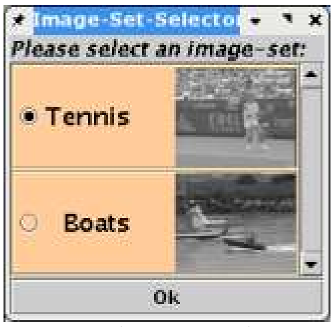


Fig. 37 Selecting another image set

5. Index

CSA	Cross search algorithm
CZS	Circular zone search algorithm
FSS	Four step search algorithm
GDS	Gradient descent search algorithm
MPEG	Motion picture experts group
MPEG2	A video compression standard, defined by MPEG
MSE	Mean squared error - a method of caluclating the distance of two values. For a better comparison with other values, the result is squared, so only positive results occur. -> SAD
NTSS	New three step search algorithm
SAD	Sum of absolute differences - a method of caluclating the distance of two values. For a better comparison with other values, the result is the absolute difference, so only positive results occur. -> MSE
TSS	Three step search algorithm

6. Bibliography

[1]Mei-Juan Chen, Liang-Gee Chen, Tzi-Dar Chiueh: "One-Dimensional Full Search Motion Estimation Algorithm For Video Coding" IEEE Transaction On Circuits Ans Systems for Video Technology, Vol. 4 No. 5, page 504-509, October 1994

[2] Lai-Man Po, Wing-Chung Ma: “A Novel Four-Step Search Algorithm for Fast Block Motion Estimation”, IEEE Transactions On Circuits And Systems For Video Technology, Vol 6, No 3, page 313-317, June 1996

[3] M Ghanbari: "The Cross-Search Algorithm for Motion Estimation" IEEE Transactions On Communications, Vol. 38, no. 7, page 950–953, July 1990

[4] Jianhua Lu and Ming Liou: "A Simple and Efficient Search Algorithm for Block-Matching Motion Estimation", IEEE Transactions On Circuits And Systems For Video Technology, Vol 7, No. 2, page 429-433, April 1997

[5] Renxiang Li, Bing Zeng, Ming l. Liou: "A New Three-Step Search Algorithm for Block Motion estimation", IEEE Transactions On Circuits And Systems For Video Technology, Vol 4, No. 4, page 438 – 442, August 1994

[6] Alexis M. Tourapis, Oscar C. Au, Ming L. Liou: “Fast Motion Estimation using Circular Zone Search”, SPIE Vol. 3653, page 1496-1504

[7] Oscal T.-C. Chen: “Motion Estimation using One Dimensional Gradient Descent Search” EEE Transactions On Circuits And Systems For Video Technology, Vol 10, No. 4, June 2000, page 608-616

[8] Tae Gyoung Ahn, Yong Ho Moon, Jae Ho Kim: “An improved multilevel successive elimination algorithm for full search estimation”, Proc. IEEE Conference on image processing, (ICIP), Sept. 2003