Dynamic Region and Block-Based Motion Estimation for Video Compression

on behalf of
École Polytechnique Fédérale de Lausanne
Signal Processing Laboratory
supervised by
Prof. *Pierre Vandergheynst*

developed in
Visiowave S.A.
under the supervision of
Dr. *Julien Reichel*
Dr. *Francesco Ziliani*

by
Philippe Jost

September 17, 2002

Abstract

In order to reduce temporal redundancies, most video codec make use of motion estimation. The most used approach is called block-matching. It decomposes the current frame into blocks and for each one tries to find its origin in the previous frame by minimizing the mean square error of each block. From these information it creates a matrix of motion vectors called a motion field. This approach does not search the real motion vectors but rather the nearest ones in term of error. Such techniques introduce the so called *blocky* effect at low bit-rate. The Human Visual System (HVS) is very sensitive to regular patterns. Thus, even if block-matching minimizes the errors, the resulting artifacts are visible.

The aim of this project is to find a motion estimation method that works in combination with block matching in order to reduce the visible artifacts. The proposed solution tries to extract the real motions taking place in a sequence. The developed algorithm is a region based motion estimator. We associate to regions of general shape motion parameters which describe an affine transformation on the plane. The traditional block matching is then used for smaller transformations.

The key fact behind the developed algorithm is that the relation between a region and a motion is very strong. The developed algorithm is iterative. It alternatively considers the motion and the region as being constant and refines the other one according to this hypothesis. The results will show that it converges and leads to good results if big objects are moving in a sequence.

In order to be able to work in a efficient manner with both kinds of motions, a framework had to be defined and created. This project also deals with the creation of a structure for region-based and for block-based motions. The block-based one consists in a kind of quad tree decomposition of the image created in the optic of multi scale motion estimation. The region-based motion structure has the property to be very modular, the information about motion is separated from the region. Thus, we can combine many motions and zones together.

Acknowledgements

I would like to thank Visiowave R&D department for allowing me to design and develop this project.
In particular I wish to thank Dr. Julien Reichel and Dr. Francesco Ziliani for their precious advices and help.
I would like to thanks my parents for their support.

. Philippe Jost

Contents

1 Introduction **7**

2 Context **9**
2.1 VisioWave V6 Coder 9
2.2 Motion Estimation in V6 10

3 Structure **13**
3.1 Region-based motion 13
3.1.1 Goals and constraints 13
3.1.2 Architectural principles 13
3.1.3 Generic Motion 14
3.1.4 Generic Zone 15
3.1.5 Motion Description 15
3.1.6 Motion Container 15
3.1.7 Translation and zoom as example 16
3.1.8 Quad-tree zone as example 20
3.2 Block-based motion 21
3.2.1 Goals and constraints 21
3.2.2 Architectural principles 21
3.2.3 Encoding 25
3.2.4 Results 28

4 Region-based Motion Estimation **35**
4.1 Algorithm principles 35
4.1.1 Algorithm overview 35
4.1.2 Information and variables used 36
4.1.3 Initialization phase 37
4.1.4 Update phase 38
4.1.5 Recursive phase 39
4.1.6 Creation of the container 44

5 Annex problems **47**
5.1 Using half-pel 47
5.1.1 Problem definition 47

5.1.2 Optimal prefilter 47
5.1.3 Numerical solution 49
5.1.4 Results 51

6 Results **53**

7 Conclusion **63**

A About VisioWave **65**

B Structure objects models **67**

C The local motion field interface **73**

Chapter 1

Introduction

There are many applications in which a compact representation of a video sequence is used. For storage, it saves space. Due to the fact that the networks have a limited bandwidth, sequences have to be compressed before they are sent.

Video compression techniques rely mainly on the reduction of redundancies in the data. In order to reduce temporal redundancies, most of the coders make use of a technique called *Motion Estimation*. They aim to predict a frame from its predecessor. Compression also tries to work with the limitation of the Human Visual System (HVS).

Traditional motion estimators try to find a motion field by simple matching strategies. The current frame is partitioned into blocks, and each block corresponds to a vector which indicates the best matching block according to a given criterion. In general, this criterion is the Mean Square Error (MSE). These techniques are called *block-matching*. For all the blocks of a frame they try to find the best translation.

Block-matching tries to find an apparent motion field which minimize a given criterion. This report presents a region-based motion estimator which estimates the real motion of bigger regions in a frame according to the past one. The regions do not have a fixed shape. The information about the motion of the regions are made of few parameters which allow to describe complex motions (rotation, zoom, ...). Thus, region-based motion estimation aims to find the real motion instead of an apparent motion like the block-matching does.

This project takes place in the context of the new video coder of VisioWave. From a general point of view, the schema of this coder is near from a MPEG-style coder. Chapter 2 will explain more in detail how it works. From the point of view of motion estimation, it has the ability to do region and block-based motion estimations.

This work mainly treats two different topics. The first one, is the creation of the structure dealing with motions in the new coder of VisioWave. The

second topic is about the region-based motion estimation.

Adding new features to a program is sometimes a difficult task. The region and the block-based structures are created to avoid such problems. Chapter 3 explains more in detail the goals and constraints that are fixed in order to have a flexible and evolutive framework for motions in a video codec.

At low bit-rate, the amount of bits used to encode the block-based motion information is not negligible. Thus, the structure has been created to encode the data about motion in an efficient manner. We will see in chapter 3 that the solution we looked at is a quad-tree decomposition of the frame where only the minimum of needed information is encoded.

For the region-based motion's structure, the key idea is the separation of motion and region. A region-based motion is the combination of the concepts of motion and of region. We will see later on, the responsibilities each of them has to deal with.

In general, a motion in a sequence of images is due to a motion of the camera or to objects moving in the scene. In literature, most of the time, a motion of the first kind is called a global motion and in the second case, we speak about local motion. In this work, we call global motion the association of a region and an affine transformation on the plane.

The key problem, in fact, is not compression but rather the quality we have at low bit-rate! The human visual system is very sensitive to regular patterns such as the one we obtain if we use block-matching technics which creates "blocky" images. The solution we propose to overcome this problem is to create associations between bigger regions of the images and affine transformations on the plane. This association is called Region-Based Motion. Using this method, we are going to reduce the spacial redundancy within video sequences. Keeping a global approach avoids the appearance of blocks at low bit-rate. If the motion is uniform on the whole image, a region-based motion estimation uses only few parameters to describe a complex motion. Thus, it will also lead to a gain in compression even if this was not our main goal. Using region-based motions is not optimal from the point of view of the mathematical error. But, the HVS is less sensitive to this kind of error.

Chapter 4 shows the proposed method to solve the estimation of the regions and their associated motions. It consists in a recursive algorithm. At each scene change, we choose initial regions and estimate their motions. The recursive phase is then to refine the motions parameters and to find for all pixels of the border of a region if it belongs to the current region, the neighbor region or if it is not part of a global motion.

Chapter 5 will be devoted to some annex problems encountered during the project. We will focus on what we called the half-pel problem. Using region-based motion estimation increases the mean square error; we will look at the reasons and also propose a solution that reduces this effect.

Chapter 2

Context

2.1 VisioWave V6 Coder

This section is a fast overview of the coder used for this project. The next one will more precisely deal with the place motion has in this coder.

Figure 2.1 shows the general aspect of the coder. The input of the coder is the current frame to encode. The output is then the data about the motion between the last frame and the current one and the spacial data which is the error between the prediction and the input frame.

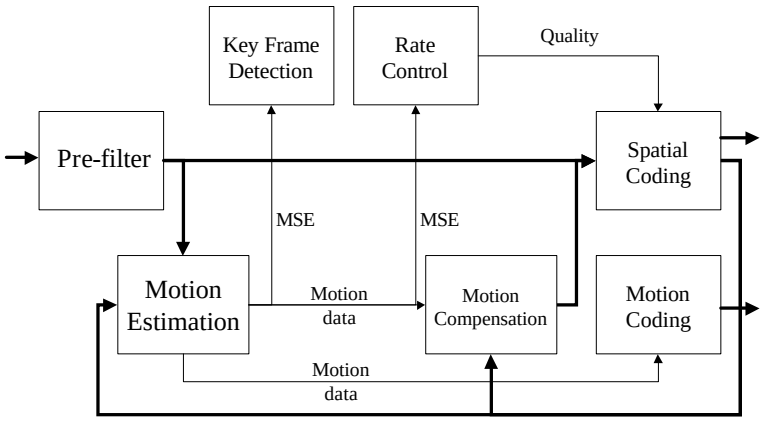


Figure 2.1: General shape of the coder (VisioWave V6).

The encoded information will be used by the decoder to recreate the sequence of image. Figure 2.2 is an illustration of how the decoding is done. The input is made of the data about motion and the spacial information. The motion compensator takes the last frame and the motion information and creates an estimate of the current frame. We add the spacial information to this estimate to create the output frame.

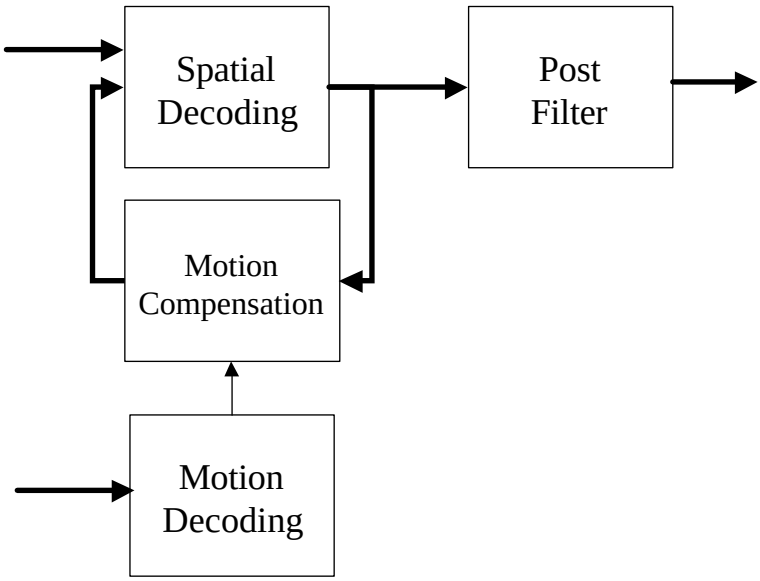


Figure 2.2: General shape of the decoder.

2.2 Motion Estimation in V6

There are two types of motion estimations: region-based and block-based. From the point of view of implementation, there is only one object that does both tasks; it has two main functions namely `estimateGlobal` and `refineLocal`. In addition to the motion estimation task, these functions return values to the coder allowing it to do scene change detection. The first function should return a scene change probability and the second one returns the MSE obtained during the block matching.

There are three possible combinations of motion estimations in VisioWave V6.

1. Only Block-based
2. Only Region-based
3. Region-based followed by Block-based

In the figures describing each of this possibilities, `Frame i` represents the frame to encode, `Frame i-1` is the last encoded image. Most of the time, it is different from the last image to encode. This is due to the quantification step that the coder does when encoding the difference between the prediction and the input frame.

In the first case (figure 2.3), block matching is used to create a local motion field[1] describing from where the blocks come in the last frame. Using

[1] See chapter3

the compensator, the estimated image is created and the difference is sent to a coder. The motion field is also encoded.

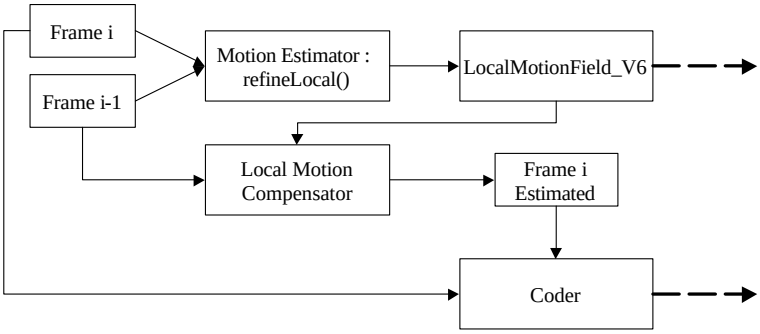


Figure 2.3: Purely block-based motion estimation.

In the second case (figure 2.4), only region-based motion is used to describe the changes between frame $i-1$ and frame i. Choosing only this kind of motion can be a very interesting solution if a transformation is only due to the camera (translation, panning, zooming, rotation,...). The container[2] compensates a frame according to the information it stores. The other steps are the same as before. A coder is used to encode the difference between the prediction and the reference frame.

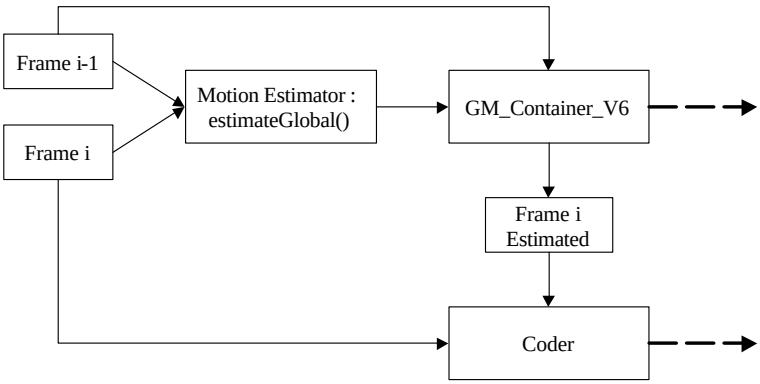


Figure 2.4: Purely region-based motion estimation.

The last case (figure 2.5) where a region-based motion estimation is followed by a block-based one, is the one treated in this project. In this situation, we want to encode image i. First, the frame passes through the region-based motion estimator (`estimateGlobal`) which will give a global motion container. Frame $i-1$ is compensated according to the information of

[2]See chapter3

the container and gives a first estimate of the frame to encode. The second stage of motion estimation consists in a block-based estimation between the found estimate and the reference frame. Using one object doing both estimations allows to use the information found during the region-based step at the block-based estimation. For example, if there is a part of the image we are very confident in the results of the region based procedure, this part of the image is encoded as a null motion in the local motion field.

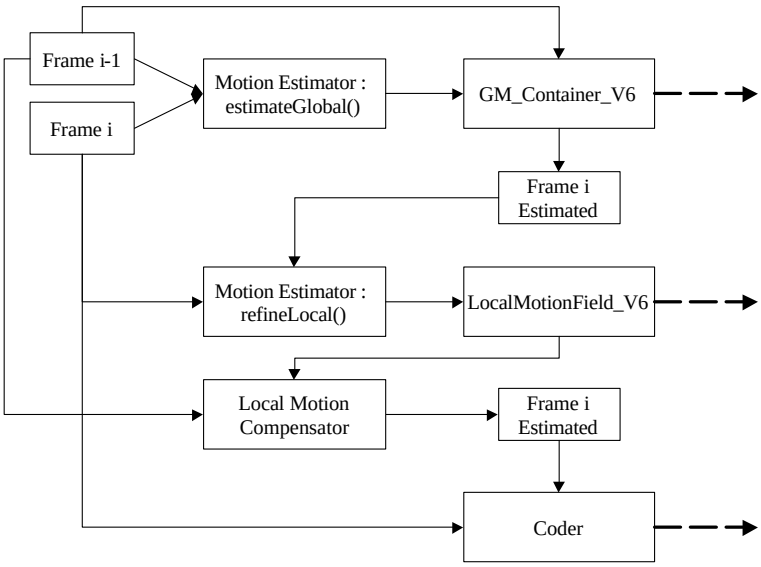


Figure 2.5: Region-based motion estimation followed by a block-based refinement .

Chapter 3

Structure

3.1 Region-based motion

3.1.1 Goals and constraints

The need for an efficient and evolutive structure is important for this kind of motion. One has to be aware that in the future, region-based motion will be an important part of the next generation coder of VisioWave. Thus, a base structure which is able to deal with the kind of information that will be added had to be created.

We defined the notion of region-based motion as a combination of a motion[1] and a zone[2] where the motion is located. Both concepts are clearly separated in the implementation. Thus, the new structure has to deal with the fact that a motion could not know all kind of zones. This lead to the creation of unified interfaces.

The structure should to be capable to handle many kind of zones and motions on the same frame. But, having many motion and zones should not increase the complexity for the developer.

At a first glance, these goals and constraints can look very restrictive. But, it is well known that having a well designed structure is very helpful when one wants to add something new. One just has to create new objects with the defined functions and variables.

The following sections are about the choices made in order to satisfy the constraints and to reach the fixed goals. Some examples will illustrate the working of the created structure.

3.1.2 Architectural principles

In order to be able to add new kind of motions in the future without modifying zones or other things, the responsibility to deal with the compensation is

[1] A motion has a type and parameters.

[2] A zone is also defined by a type and according parameters.

given to the motion. This means that when a developer adds a new type of motion, he also has to implement the corresponding compensation function. This also leads to create a structure that ensures that any motion can deal with any zone. In the worst case, a motion should be able to compensate an image from generic information sent by a zone. This information is a binary mask describing which parts of the image are part of the global motion and which are not.

From a general point of view, a task is given to the object which store the information related to it. This is the reason why a zone is responsible to compute its parameters from a binary mask describing the region to encode. This has also another big advantage, we could for example imagine to have a pool of different zones and to ask each one to encode a given region. The motion estimator then chooses the best one according to some criterions as the number of bits needed to encode the zone or the amount of badly classified pixels (pixels added that were not in the mask or pixels in the mask that have not been encoded).

Annex B shows the interfaces of the implemented objects described in this chapter.

3.1.3 Generic Motion

The generic motion defines the interfaces a motion must have in order to satisfy the fixed constraints. Such a motion can directly be integrated in the motion estimator. Basically, a generic motion is described by a set of parameters and a type[3] giving a semantic meaning to these parameters. Some other responsibilities have been given to a motion. A motion has to be able to store its parameters to a stream and has also to be able to retrieve them. In addition to this, the motion has to be capable to compensate an image according to its parameters and a zone. The reason of this choice is simply due to the fact that we wanted to give this work to the object which knows most about the motion.

The motion has the responsibility to estimate its parameters from a set of vectors and positions. The example about translation and zoom included in this chapter on page 16 explains more in detail how this estimation is done. A model should also be able to refine the values of its parameters given a set of vectors and positions. The estimation and the refinement of the parameters has a great influence on the behavior of the region-based motion estimation algorithm as shown in chapter 4.

A motion should also be capable to compute the distance from another motion to it. We want this function to return a value between 0 (equal) and 1 (completely different).

A motion also has to be able to compute the distance (in term of MSE)

[3]Rotation, translation for example

from a block to itself.

A motion also has a state. For example, in the implementation, the translation and zoom motion first estimate its parameters as if it was a pure translation; when a refinement of the parameters is asked, it tries to evolve to a more complex model.

From an external point of view, it is impossible to say if a motion is null or not according to the parameters without knowing the meaning of these parameters. For this reason, there is also a function (`nullMotionTest()`) returning `true` it the motion is null on the whole image.

3.1.4 Generic Zone

As for motion, a generic version of a zone has been developed. It defines the interfaces that has to be implemented for a zone in order to be compatible with the rest of the system. A zone estimates its parameters from a binary mask describing the region to encode. Most of the time, when a zone estimates its parameters, the resulting solution is not exactly what was the binary mask. Thus, a minimization type attribute has been added; it allows a user to tell to the zone how it has to approximate a mask. For example, in some applications, a zone should not add pixels. Or, in some other cases, a user wants to use as few as possible bytes to encode the zone and thus, does not mind if some pixels are added or removed.

As for the motion object, a zone has to be capable to store and retrieve its parameters. The example on page 20 of a zone using a quad-tree decomposition of the binary plane illustrates how the streaming can be done. It also shows how the estimation of the parameters from a binary mask can be done.

3.1.5 Motion Description

The motion description represents the "real" region-based motion. It represents the association of a zone and a motion. Apart from this, it is used to create a simple interface to work with a region-based motion.

It provides an interface to deal in a very simple manner with more complicated tasks. It is able to store itself to the stream. The compensation of a frame can be directly called trough it. It relays the user's invocations to the right place. Thus, the programmer does not have to deal with complex calls; the motion description does the needed work for him.

3.1.6 Motion Container

We said that there can be more than one region-based motion on a frame. The motion container reflects this. It holds 1 to n motion descriptions.

It provides a simple interface for a developer to deal with many motion descriptions. The calls to this object will be forwarded to the right underlying object. Figure 3.1 shows a general overview of the container and how the region-based motion structure has to work.

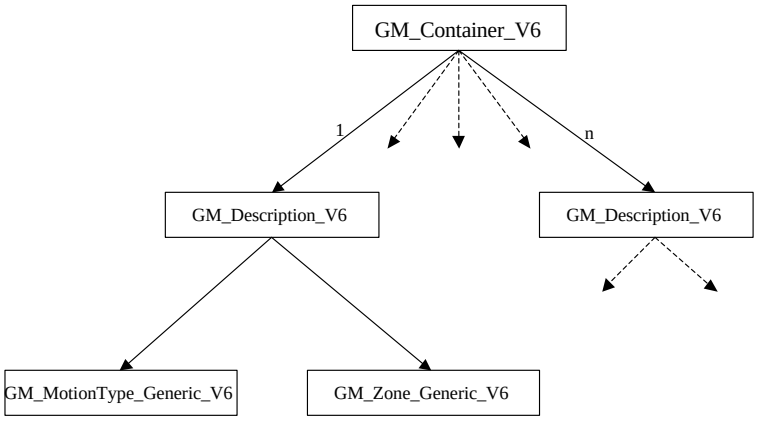


Figure 3.1: Global motion structure overview.

3.1.7 Translation and zoom as example

Introduction to the example

This example passes trough the different phases of the parameters estimation and the model refinement. It also illustrates the streaming abilities.

As we said before, the motion has a state; in the case of translation and zoom, there are three possibilities:

1. Translation.
2. Zoom centered in the middle of the sequence.
3. Zoom and translation.

We will assume that a zoom can only result from a camera-zoom. This kind of zoom has the property that it is centered in the image; so, the most complex motion we could have is a zoom combined with a translation. This hypothesis limits the complexity.

The parameters we want to find are defined by the following equation:

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} c_x \\ c_y \end{pmatrix} + \rho \begin{pmatrix} x - c_x \\ y - c_y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{3.1}$$

(x', y') represent the position of the point (x, y) in the last frame, ρ is the zoom factor, t_x and t_y are the translation parameters. Finally, (c_x, c_y) is the center of the image.

The number of parameters to find depends on the state of the motion. In the first case, we have only a translation, so the zoom factor is equal to 1. The motion has to find t_x and t_y.

In the second state, the translation is null so, the motion only has to find the zoom factor ρ. Even if there is only one parameter to find, this problem is more difficult. The zoom factor has to be found more precisely than a translation. Assume that the difference between the true zoom factor and the current one is 0.01 and that the current point is at position $(0, 0)$ the upper-left corner of our image. The resulting error will then be $0.01c_x$ for the x axis and $0.01c_y$ on the y axis. Typical values for c_x and c_y are 180 and 144. This means that the error corresponds to 1.8 pixels on the x axis and 1.44 pixels on the other axis. Knowing that compensation uses half-pel, the error are now of 2 and 1.5 pixels!

The last case represents the fact that someone zooms with a camera while moving it in a translational manner.

There are many ways to find the parameters. In this project, a genetic algorithm has been used. The working of a genetic algorithm will not be explained in detail. From the point of view of the user, it needs an objective function to minimize. The choice of this function has a big influence on the behavior of the region-based motion estimation. Thus, it will be explained later on in this report.

Estimation of parameters

The meaning of parameters estimation is in fact to find the initial parameters for the motion. The estimation always starts with small regions. It reflects the assumption that locally, even a complex motion, can be represented in a translational manner. The grey rectangle on figure 3.2 shows that locally, a zoom can be seen as a translation.

There are mainly two reasons to create initial regions with only translation as motion. The first reason is related to complexity considerations. Computing the mean of a set vectors is very fast. The second reason follows from the fact that if we have a region-based motion, the regions are small enough to be local approximations of this motion. If we do not have a global motion, the local approximation would be false and the region will never grow and move to a more complex models. They are detected as non region-based motions and are destroyed. Thus, the initial translation approximation can also be a fast test to know if there is a global motion.

For the estimation of parameters, we first have a list of motion vectors and a list of corresponding points. In order to estimate translation, the returned solution is the mean of all the vectors.

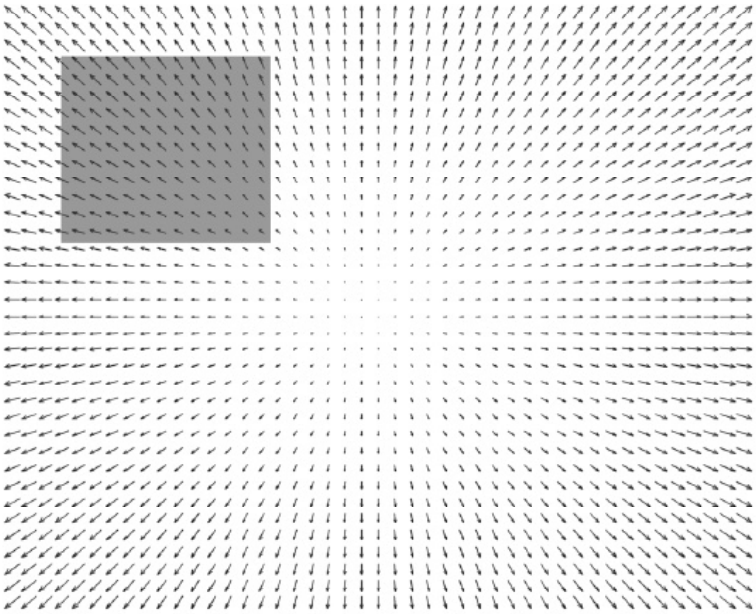


Figure 3.2: A zoom can locally be approximated by a translation.

Refinement of parameters

This is the most important task our motion has to do. It is done once or many times per frame. It can be divided into two steps:

1. Check if the current state is accurate. Try to evolve or to regress.
2. Update the parameters according to the chosen state and positions of vectors.

In the first step, one of the following tasks is done: evolve, regress or do nothing. In order to illustrate the evolution procedure, assume that the motion is in the translation state. From there, it is possible to evolve to a zoom or to a zoom and a translation. To check the different possibilities, a fast evolution testing algorithm has been developed. Figure 3.3 shows an example of this test. The information at disposal are the motion vectors and the associated points. Algorithm 1 tests if the motion could evolve and if it is the case, returns an estimate of the parameters. We will see later on the use of the returned values.

The more complex a model is, the more parameters the motion has to find. Thus, it increases the computational complexity. For this reason, there is also a test to see if it is possible to regress to a simpler model. One can see that there is no regression from zoom to translation. A change from zoom to translation will pass through the zoom and translation state and then regress to translation.

Algorithm 1 Test possible state and find estimate of motion parameters

g_0 = gravity center of received coordinates
v_0 = mean translation of the received motion vectors
for all points on one side of line defined by g_0 and v_0 **do**
  g_1 = gravity center of coordinates of $R1$
  v_1 = mean translation of $R1$
end for
for all points on the other side of line defined by g_0 and v_0 **do**
  g_2 = gravity center of coordinates of $R2$
  v_2 = mean translation of $R2$
end for
i_{01} = intersection of $g_0 + \alpha v_0$ and $g_1 + \beta v_1$
i_{02} = intersection of $g_0 + \alpha v_0$ and $g_2 + \beta v_2$
i_{12} = intersection of $g_1 + \alpha v_1$ and $g_2 + \beta v_2$
if i_{01}, i_{02} and i_{12} not near from each other **then**
  return motion is not a zoom
end if
i the mean of i_{01}, i_{02} and i_{12}
if i near from the center **then**
  knowing i, find ρ the zoom factor
  return state is 1, center i with zoom factor of ρ and translation is null
else
  knowing i, find ρ the zoom factor
  knowing i and ρ, find t_x and t_y the translation
  return state is 2, center i with zoom factor of ρ and translation of t_x and t_y
end if

Experimental results show that the state of a motion remains relatively constant on a large amount of frames.

Once that the current state is found, the motion estimates the corresponding parameters. There are many ways to do this; in the implementation, we choose a genetic algorithm to do this. First, an initial population of genes (corresponds to an array of the parameters) is created then the best is chosen according to an objective function to minimize. Once the best is found, the genetic algorithm randomly creates new genes near from this one. These operations are called mutations and crossovers. The genetic algorithm also constructs randomly chosen new genes.

The evolution test returns an estimate of the parameters. Generally, the values returned are near the real parameters. A genetic algorithm is very good to converge to the minimum if the initial gen is near the optimal value. The initial population will contain the values returned by the evolution test and slightly modifications of this. It also contains the last found gen. An-

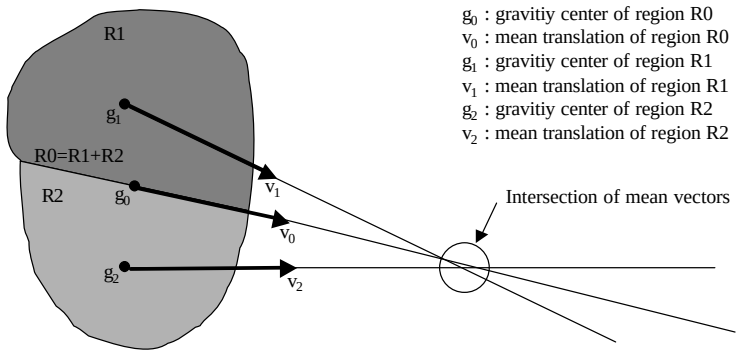


Figure 3.3: Test possible state.

Algorithm 2 Try to regress to a simpler model

```
if not evolved and state is different from translation then
  if state is zoom plus translation then
    if zoom factor is very close to 1 then
      state is translation and zoom factor ρ = 1
    else if translation is very close to 0 then
      state is zoom, t_x = 0 and t_y = 0
    end if
  end if
end if
```

other interesting property of the genetic algorithm is that the computational complexity is not too high.

The translation and zoom motion has the choice between two kind of objective functions to minimize. A fast and less fast one. The first consists in finding the parameters that minimize the distance between the passed motion vectors at each point and the resulting motion vector found at the same point in function of the motion parameters. The second version of the objective function adds to this distance the resulting mean square error obtained when using these parameters.

3.1.8 Quad-tree zone as example

Goal of this zone

The first tests of region-based motion estimation algorithm were made with a zone named multi-rectangles. It approximates the binary mask with a list of rectangles. This solution is not optimal in the general case where zones have complex shapes. However, it is very good in describing the zones which correspond to the whole image (unique global motion for the current frame)

or to a regular inner part of the frame (zoom out motion).

The quad-tree zone decomposes the image into a tree-based structure. The next section describes how the parameters are estimated.

Parameters estimation

A binary mask that describes the zone is received. The zone has to find the parameters corresponding to this zone to encode it. They correspond to the description of the quad-tree decomposition of the received binary mask. Figure 3.4 shows an example of this kind of decomposition. This procedure is applied to the mask. The resulting parameters are the values shown near to the nodes in part *d.* of figure 3.4. They describe in a unique manner the decomposition.

Thus, the encoding is very easy, we simply pass the 0 and 1 bits to the decoder. The example on figure 3.4 would need 30 bits to be encoded. If we had used the multi-rectangles zone, we would have had 16 parameters to encode. Assuming that the size of a rectangle can not be bigger than 1024, the amount of bits needed would have been 160!

3.2 Block-based motion

3.2.1 Goals and constraints

A motion field has to be a natural instrument to deal with multi-scale motion estimation. It should also be able to store a lot of information; we would like, for example, to be able to store more than one motion vector per region. And the motion vectors are not only two dimensional vectors but four dimensional ones. Two parameters for the motion, one to store the mean square error of the block and a confidence value. At low bit-rate, the size of a motion field is no longer a negligible part of the information to encode. Thus, our field should also include an efficient method to store the minimum amount of information used by the decoder to the stream and to retrieve them. Another constraint, is that the complexity should be hidden from the point of view of the user working with this kind of motion field.

In order to satisfy these constraints, we decided to create a kind of modified quad-tree based decomposition of the image where the nodes store information related to the region they are responsible for. The next section describe this more in detail.

3.2.2 Architectural principles

Local Motion Field Node

This section describes how a node behaves and how the tree is built.

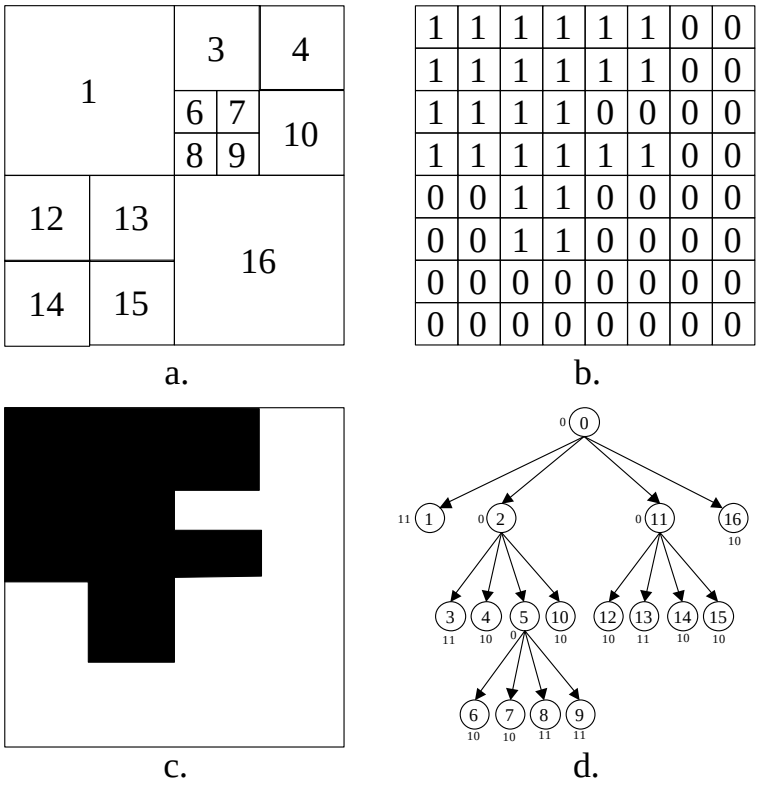


Figure 3.4: a. regions corresponding to a node in the tree b. binary mask c. shape of the region we want to encode d. tree with binary code to pass for each node

The tree structure is built deterministically; this avoids to have to pass information about the decomposition to the decoder and thus to loose precious bits. The tree is built in order to be efficient for multi-scale estimation. The block-based motion estimator implemented in the VisioWave coder V6 is able to deal with blocks having a size equal to a power of two. Typically the dimension of the blocks goes from 32x32 to 4x4 pixels. Knowing this, our goal is to create a maximum of blocks corresponding to this criterion. The coder for which this structure is implemented crops the images to be a multiple of eight in height and in width. The creation of the tree makes use of the knowledge of the size of the biggest divider of the image we are sure to have. Assume this size is `bd` standing for *biggest divider*. It has to be a power of two equal or bigger than the size of the smallest estimable block.

A node is always responsible for a region on the image. The root node is responsible for the whole frame. A node can have two or four children according to its shape.

The algorithm used to create the tree decomposition can be divided into

smaller parts. The first one consists in finding the shape of a node according to the parameters it stores about the region it is responsible for. The shape determines how the children have to be created. The second deals with the creation of the children according to the previously found shape.

Algorithm 3 finds the shape of a region according to the parameters a node stores.

Algorithm 3 Find the shape of a region.

width the width of the current node's region
height the width of the current node's region
$sizeRatio = (width)/(height)$
if $sizeRatio < 0.5$ **then**
  the shape is vertical. It leads to a horizontal division of the region in two parts.
end if
if $sizeRatio > 2.0$ **then**
  the shape is horizontal. It leads to an vertical division of the region in two parts.
end if
if $sizeRatio > 0.5$ AND $sizeRatio > 2.0$ **then**
  the shape is regular. It leads to the division of the region in four sub-regions.
end if

As shown in algorithm 3, there exists three different shapes. Thus we also have three different cases to treat when we want to create new children for a node. Let us have a look at each of these algorithms.

If the shape we obtained is vertical we divide the region horizontally in two parts. This will lead to the creation of two new nodes. Algorithm 4 finds the parameters of those new nodes and creates them. They will be set as the children of the current node. In order to create a new node, we have to find its position on the image and the size of the region it will be responsible for. Assume a new node is then created as follows:

```
node = createNode(ulx, uly, width, height)
```

where `ulx`, `uly` are the coordinates of the upper left corner of the region and `width`, `height` define the size of the region.

The second case is very similar to the first one. Instead of having a vertical shape, we have an horizontal one. Thus the region has to be divided horizontally. This procedure is illustrated by algorithm 5.

The regions are not cut in the middle. They are divided at the multiple of `bd` in the middle. This ensures that the algorithm will never lead to the creation of blocks that are not estimable.

Algorithm 4 Create new nodes if the shape is vertical.

(ulx, uly) and (drx, dry) the coordinates of the upper left and down right corners of the region of the current node.
set the current number of divisions of this node to two
$divY = ((\lfloor uly/bd \rfloor + \lfloor dry/bd \rfloor)/2) * bd$
first child is $createNode(ulx, uly, drx - ulx, divY - uly)$
second child is $createNode(ulx, divY, drx - ulx, dry - divY)$
set parent of the created nodes as the current one.

Algorithm 5 Create new nodes if the shape is horizontal.

(ulx, uly) and (drx, dry) the coordinates of the upper left and down right corners of the region of the current node.
set the current number of divisions of this node to two
$divX = ((\lfloor ulx/bd \rfloor + \lfloor drx/bd \rfloor)/bd) * bd$
first child is $createNode(ulx, uly, divX - ulx, dry - uly)$
second child is $createNode(divX, uly, drx - divX, dry - uly)$
set parent of the created nodes as the current one.

The last case is the most interesting one for use. As we have seen before, one of our goals in this decomposition is to have blocks that divides hierarchically from the biggest estimable size to the smallest one. Thus, we try to make blocks that are as big as possible, square and whose size is a power of two. The last case is divided in four subparts.

Figure 3.5 illustrates in which order the children will be stored in the tree according to the kind of shape we have. The order in the tree is very important and is well known by all participants in the local motion field. This allows a node to know how its children are placed according to its shape. Thus to relay the information and other actions to the right child.

The following example make use of the previously described algorithm in order to create a full tree[4]. Algorithm 7 creates such a tree for a given frame. The first three steps are illustrated by figure 3.6. A tree is created for an image of size 360x288. To make the algorithm simpler, we also assume that we have a function `getNextNode` that returns the next node to divide or `No more nodes` if we are finished. The next node is defined as being the leaf node most left in the tree that is bigger than the smallest estimable block. In typical cases, the smallest estimable block has a size of 8x8 or 4x4. We have to make sure that this size is smaller or equal to the biggest divider of the image we are sure to have. As we have seen before, in our case, the size of this divider is eight.

This algorithm is very simple. The `getNextNode` function is also simple

[4]All leaf nodes have the same size as the smallest estimable block can be created for an image

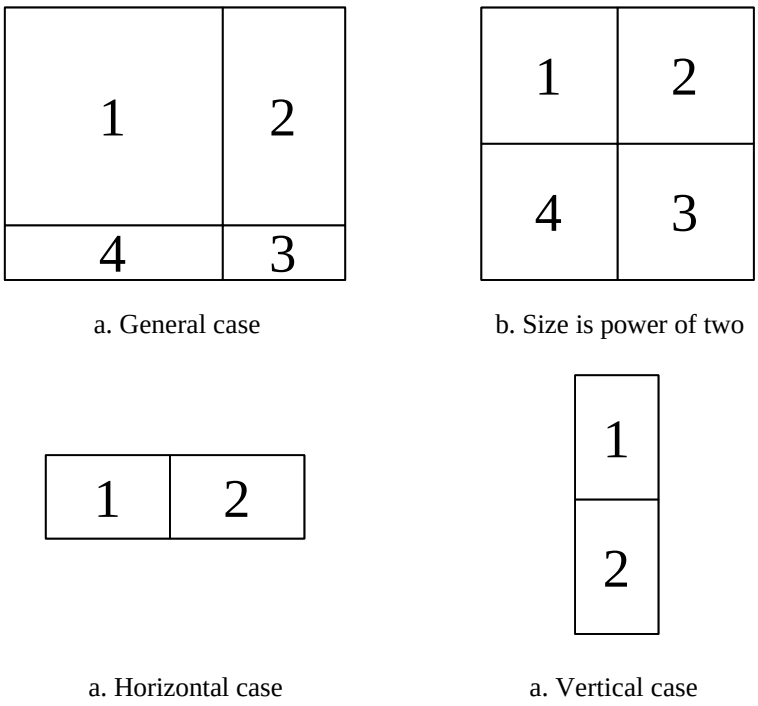


Figure 3.5: Order of children of a node in function of the shape of the parrent.

to obtain. It is a Depth-First Search (DFS) algorithm.

Local Motion Field

One of our goals is to hide complexity from the point of view of the user working with this structure. Most of the information, a user wants to handle, is stored in the tree. The Local Motion Field is an interface that gives the ability to work with the data without having to get worry about how it is internally managed. The Local Motion Field also deals with the information that is common to all the nodes.

The annex C shows the interfaces the local motion field gives to a user in the implementation.

3.2.3 Encoding

One important constraint for the created structure is the efficiency in term of place used to store the information about motion in a compressed sequence. This section describes the `store` and `retrieve` algorithms of the local motion field. Another aim of the created structure is the ability to contain

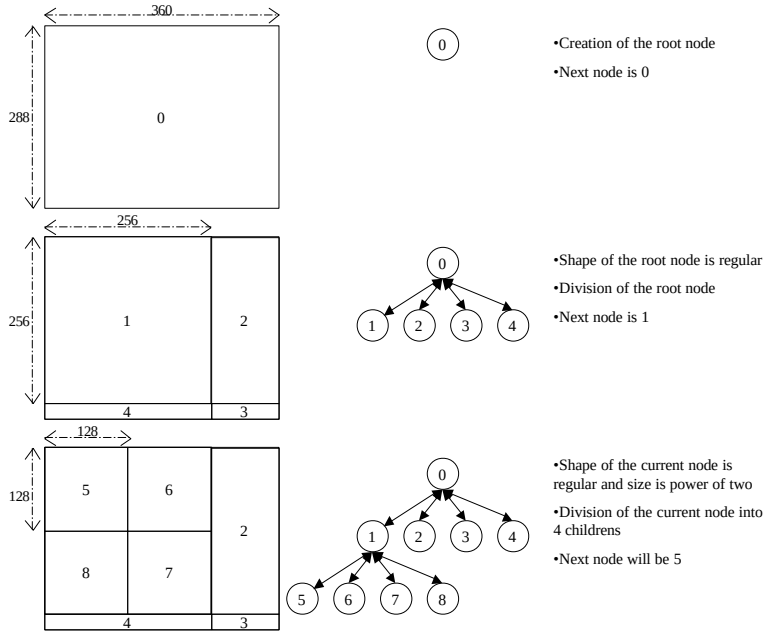


Figure 3.6: Three steps of the creation of the tree structure for the local motion field.

many different kind of information that are used during by the block-based estimator. There is no need to send everything to the decoder. The local motion field tries to send only the minimal amount of information needed by the decoder to compensate the image.

The following data is used at the decoder to compensate the image and is sent:

- the pel divider which gives sense to the values stored in the motion vectors
- the biggest and the smallest estimable block
- the size of the image
- the two first values of the motion vectors at first position of the leaf nodes
- the binary description of the tree

To encode these information, the local motion field makes use of an arithmetic coder. This coder works very well to encode bits given a context that modifies the probability for the current bit to be 0 or 1. For each node, the tree has to send a boolean value indicating if this node has children

or not. The ability an arithmetic coder has to encode bits is thus very interesting. The motion vectors also need to be sent. Other non binary data like the motion vectors also have to be sent. This fact led to the creation of two functions `codeAmplidude` and `decodeAmplitude` that use the arithmetic coder to encode integers values.

The motion vectors are not encoded as they are. The motion field first tries to predict the motion vector and then encodes the difference between the predicated and the real motion vector. This procedure tries to reduce the spacial redundancy contained in the motion field.

The encoding algorithm does a Depth-First Search in the tree and stops at each node to encode the information it contains. Algorithm 8 illustrates the encoding procedure in a simpler manner. Assume for this example that there is a function called `getNextNode` that implements the DFS algorithm and returns the next non-fully treated[5] node it encounters on his walk trough the tree or `No more nodes` if we parsed the whole tree. This function starts from the root node.

Algorithm 8 describes this simplified encoding procedure. Algorithm 9 illustrates the corresponding decoding function.

The simplified version does send the divide/not-divide bit for each node. The following example shows that if we make use of some other knowledge and hypothesis, this bit has not to be sent for all the nodes. Figure 3.7 is the reference tree for this example. It represents a fictive case where the size of the image is 12x12 and the sizes of the estimable blocks are 8x8 and 4x4.

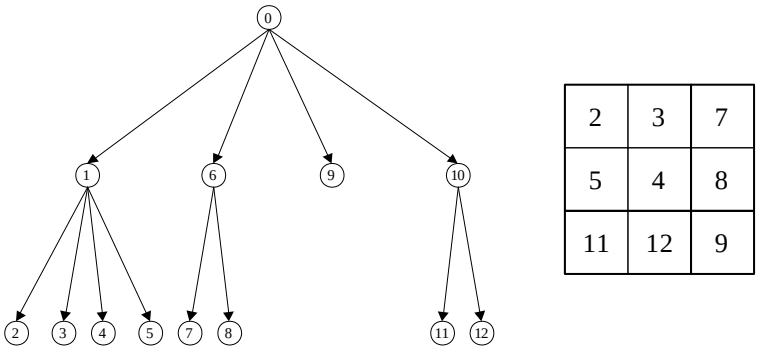


Figure 3.7: Example of a tree we want to encode.

The first case shown in table 3.1 illustrates which nodes are treated by the simple algorithm. For the second case, the algorithm gets rid of encoding the bits for the nodes on its way when going up in the tree. This is due to the fact that the division bit has already been sent for this node. In this case, the algorithm has to walk up to the next non-fully treated node and

[5] A node is defined as fully treated if all its children have been parsed.

Case	Ordered list of nodes where the divide bit is encoded	bits used
1	0 1 2 1 3 1 4 1 5 0 6 7 6 8 0 9 0 10 11 10 12	21
2	0 1 2 3 4 5 6 7 8 9 10 11 12	13
3	0 1 6 10	4
4	1	1

Table 3.1: Encoding of the information about the tree decomposition.

to act as if it received a divide bit for this node. The third case makes use of the information about the size of the smallest estimable block. The algorithm knows that such a node can not be divided. Thus, there is no need to pass the divide bit for such nodes. The last line shows an example that is not efficient in most of the cases. The algorithm assumes that motions are only encoded for block that correspond to an estimable size. Thus, the algorithm encodes the divide/not-divide bit only for the nodes whose region have an estimable size larger than the smallest one. This case leads to the best results for this example. In general, this is not true. The problem is that this methods prevents the algorithm from grouping children with equal motion vectors into their parent node if the size of the parent is bigger than the biggest estimable block.

For the previously explained reasons, the third method is used to encode the information about the tree in the created structure.

In order to reduce the size needed to encode a motion field, predicate of the motion vector is used. This is done by creating a list of neighbor motion vectors that have already been encoded. The predicate value will be the median vector. In a matrix-based structure, obtaining a list of neighbors is very easy. In our tree based structure, this task is no longer trivial. A node does not have the knowledge of the structure over it. A node has to ask its father to give him this list of motion vectors. If a node receives such a call for its child at position n, the children from 1 up to $n-1$ have already been encoded and have neighbors nodes in their subtree. Thus, the node will ask them to find motion vectors that have this property and to put them in a list. Let us cut this problem in three smaller parts. The first algorithm (10) will ask the parent for the list of vectors and find the median one. The second part is the call that goes up in the tree. This will locate potential neighbor subtrees; illustrated by algorithm 11. The last part goes down in these subtrees to find the motion vectors. This procedure is shown in algorithm 12.

3.2.4 Results

In order to analyze the results, the performances of the created block-based motion field used in the VisioWave coder V6 are compared to the motion

field of a MPEG-2/4 coder. This motion field is a matrix whose width and height are the one of the image divided by the size of the blocks ($bSize$). At each point (x, y) of this matrix, we store a motion vector (dx, dy) for the block of size $bSize$ at position $(x \cdot bsize, y \cdot bsize)$.

The encoding of the motion field of a MPEG-2/4 coder is done the following way. The encoding algorithm passes trough the matrix horizontally. For each point, it predicts the current vector according to its encoded neighbors and then passes the difference to a Huffman coder.

To have comparable results, the estimation mode is only block-based. We obtain a tree-based motion field. From this tree, a MPEG-2/4 motion field is created. The size of the matrix depends on the smallest estimable block used in the V6 block-based estimator. The two different motion fields store the same information and are thus comparable.

The measures of the performances are done by encoding both and comparing the size in bytes they need to encode the same data into a video stream.

Figure 3.8 shows the amount of bytes used to encode both motion fields for different sequences at each frame. As one can see, for these test sequences, the structure of V6 uses less bytes than to one from MPEG-2/4.

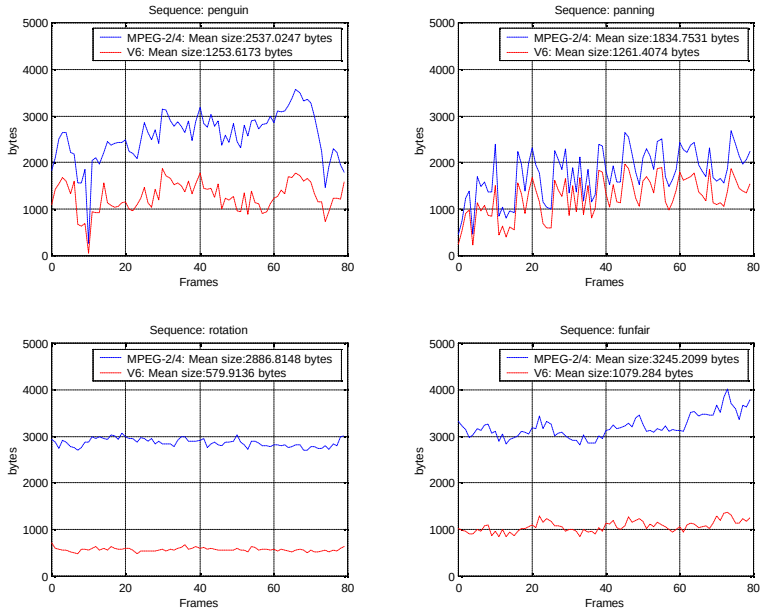


Figure 3.8: Size of encoded motion field per frame for different sequences.

At low bit-rate, the size of the motion field is not a negligible part of the total size of an encoded sequence. Thus we wanted to create a motion field which is small in terms of bytes used to encode. Figure 3.9 shows the same

sequences encoded at different qualities. For each quality, the mean size of the motion field per frame is computed. At low quality, the part of the total size used by the motion fields is high.

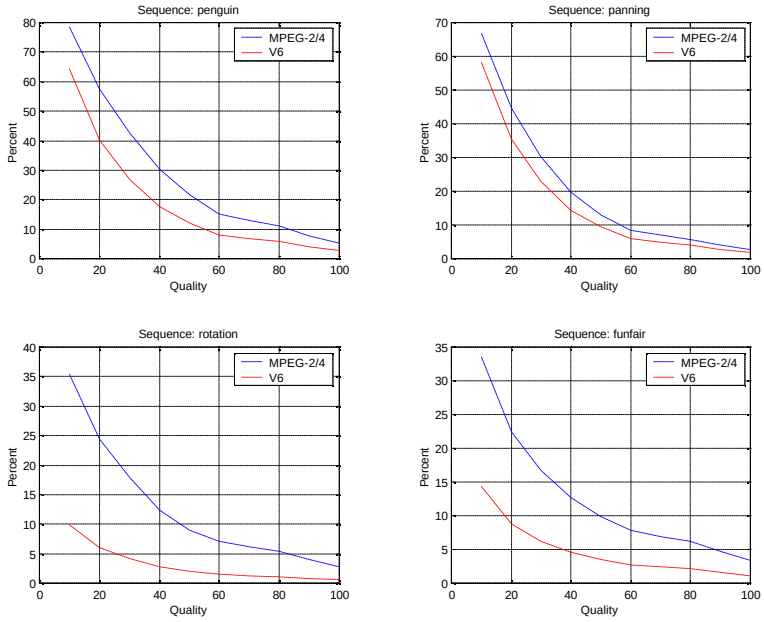


Figure 3.9: Part of the total size used by the motion fields in encoded sequences at different qualities.

Algorithm 6 Create new nodes if the shape is regular.

width the width of the region and *height* its height
$maxPower2$ = biggest power of 2 such that $maxPower2 \leq \min(width, height)$
if *width* equals $maxPower2$ and *height* is different from $maxPower2$ **then**
  set the current number of divisions of this node to two
  $divY = maxPower2 + uly$
  first child is $createNode(ulx, uly, drx - ulx, divY - uly)$
  second child is $createNode(ulx, divY, drx - ulx, dry - divY)$
  set parent of the created nodes as the current one.
end if
if $width != maxPower2$ AND $height == maxPower2$ **then**
  set the current number of divisions of this node to two
  $divX = maxPower2 + ulx$
  first child is $createNode(ulx, uly, divX - ulx, dry - uly)$
  second child is $createNode(divX, uly, drx - divX, dry - uly)$
  set parent of the created nodes as the current one.
end if
if *width* equals $maxPower2$ and *height* equals $maxPower2$ **then**
  set the current number of divisions of this node to four
  $divX = (ulx + drx)/2$ and $divY = (uly + dry)/2$
  first child is $createNode(ulx, uly, divX - ulx, divY - uly)$
  second child is $createNode(divX, uly, drx - divX, divY - uly)$
  third child is $createNode(divX, divY, drx - divX, dry - divY)$
  last child is $createNode(ulx, divY, divX - ulx, dry - divY)$
  set parent of the created nodes as the current one.
end if
if *width* different from $maxPower2$ and *height* different from $maxPower2$ **then**
  set the current number of divisions of this node to four
  $divX = maxPower2 + ulx$ AND $divY = maxPower2 + uly$
  first child is $createNode(ulx, uly, divX - ulx, divY - uly)$
  second child is $createNode(divX, uly, drx - divX, divY - uly)$
  third child is $createNode(divX, divY, drx - divX, dry - divY)$
  last child is $createNode(ulx, divY, divX - ulx, dry - divY)$
  set parent of the created nodes as the current one.
end if

Algorithm 7 Create a full motion field tree.

root node is $createNode(0, 0, widthOfFrame, heightOfFrame)$
find the shape of the root node
create the children given the shape of the root node.
while current node given by `getNextNode`) is different from `No more nodes` **do**
  find the shape of the current node
  create the children given the shape of the current node.
end while

Algorithm 8 Store a local motion field. Simplified version.

encode information contained in the motion field
while current node found by `DFSNextNode` different from `No more nodes` **do**
  if current node is not a leaf node **then**
    encode one bit equal to 1
  else
    encode one bit equal to 0
    get predicate of this motion vector
    encode difference between motion vector and predicate
  end if
end while

Algorithm 9 Retrieve a local motion field. Simplified version.

if root node does not exist **then**
  create root node
end if
set the current node to root node
while not finished **do**
  divide = get next bit from stream
  if *divide* == 1 **then**
    if current node does not have children **then**
      create children of the current node
    end if
  else
    get predicate of this motion vector
    decode difference from stream
    set the motion vector for this node as predicate + difference
  end if
  current node updated from `DFSNextNode`
  if current node is `No more nodes` **then**
    construction is finished
  end if
end while

Algorithm 10 Retrieve a list of motion vector and find the median one.

create an empty list of motion vectors
the number of motion vectors in the list is set to 0
define the numbers of levels we want to go up in the tree
ask the parent node to fill up the list of motion vectors. The current node passes its coordinates and how many times he want this call to go up in the tree.
return the median motion vector contained in the list

Algorithm 11 Go up in the tree to find subtrees that could contain a neighbor to the node that first made this call.

n the child from which the call is issued
for $i = 1$ to $n - 1$ **do**
  ask child i to put motion vectors in list if possible
end for
if I am not the root node and levels left to go up **then**
  ask the parent node to fill up the list of motion vectors. Pass the information received from the node that called my.
end if

Algorithm 12 Go down in a subtree and find leaf nodes that are neighbor of node and put their motion vectors in a list

```
if we are neighbor of node which asked for the vectors then
  if I am a leaf node then
    add my first motion vector to the list of vector
  else
    for all my children do
      ask for motion vector to put in list if possible
    end for
  end if
end if
```

Chapter 4

Region-based Motion Estimation

4.1 Algorithm principles

Region-based motion estimation is seen in this project as an unsupervised classification problem. Our goal is to classify the pixels of a frame in n different classes. The upper bound on the number of motions is fixed.

There exists a lot of papers[1] dealing with region-based motion estimation. In many cases, they start from a given region to find a motion or start from some kind of motion to find the corresponding region. Starting from the point that the relation between motion and its corresponding region is strong, we developed an iterative algorithm that refines the motion according to the region and then slightly modifies the region according to the new motion parameters. Experimental results show that this kind of algorithm converges to a coherent solution and that the non-coherent one will rapidly be detected and removed. We also make use of temporal redundancy by assuming that a region-based motion is constant from one frame to the next in order to create the current initial state for the algorithm.

This chapter is devoted to region-based motion estimation. The first section describes the algorithm from a general point of view. The second section deals with the information and the variables that are used by the algorithm. We will then pass trough the different phases it is made of.

4.1.1 Algorithm overview

A frame is divided into atomic parts. Let us call the smallest element of a region a chunk. Typically, it is a block of 2x2 pixels. Thus, the resolution of the regions is rather high.

[1] Some are listed in the bibliography.

The general shape of the region-based motion estimation algorithm (13) is rather simple. It can be divided in three parts. An initialization phase if we do not have access to information from the past. This happens after each key frame. An update phase to create a new initial state if there was no key frame just before. And finally the recursive phase which will refine the motions and the regions.

Algorithm 13

if Previous frame was a key frame **then**
  INITIALIZATION. Create regions and associated motion.
else
  UPDATE. Assuming motion is constant, displace the old regions according to the motions in order to create the actual estimate of the zones.
end if
for number of recursion **do**
  REFINE MOTIONS AND RANDOM CHECK. According to the actual regions and motions, refine the motions. For randomly chosen chunks, test their adequation to the model and remove them if they are too bad.
  CREATE NEW MODELS. If we have enough place, create new zones and estimate their motion.
  DYNAMIC PARAMETERS UPDATE. There are some variables that depend on the sequence that are dynamically computed.
  for number of border checks **do**
    BORDER CHECKS. A border chunk of a zone can have another region a neighbor or an unassigned region. Find the region where the chunk is more likely to belong to.
  end for
  ELIMINATE AND MERGE. Try to merge regions and eliminate the false or too small ones.
end for
CREATE THE CONTAINER.

The core of the algorithm relies on successive creation of region-based motion candidates and on the elimination of those which do not satisfy enough conditions to be region-based motion. The accepted one are then refined. Their motion and region then evolve. The goal is to have as few models as possible which cover as many pixels as possible.

4.1.2 Information and variables used

First of all, our algorithm needs the ability to estimate motion vectors for blocks. We will simply make use of the same block-matching estimator as the

one used by block-based motion estimation. If the block we want to estimate is located in a region, we will use the motion vector it has according to the motion related to this region as initial state for the block matching.

The region-based motion is the association of a motion and a region. We will from now on call this pair a model. At each time, the algorithm is aware of the number of models it is working with. This value changes each time models are merged, created or deleted.

The current amount of pixels in a model is also known. The algorithm also has access to the history of this information. This permits to the algorithm to know the evolution in term of the size of a zone.

To each model, an age is associated. The elder a model is the more it passed through the algorithm; thus, the algorithm is more confident in such a model. In a way, the age reflects the *experience* a model has.

The models have a unique identifier. Spacial knowledge of the regions is also needed. A mask stores for each chunk the identifier of the model it belongs to. The size of this mask is the one of the frame divided by the size of a chunk. The identifier gives also access to the motion corresponding to a model.

It is known that the block-matching works better if the block we are trying to match contains a border. In order to make use of this fact, we create at the beginning a mask containing interesting point to do block-matching on. Many different filters can lead to such a mask.

The region-based motion estimation algorithm has some fixed constraints. These values mainly correspond to upper bounds. The maximum number of models the algorithm is allowed to work with is fixed. The condition for a model to be considered as a region-based motion relies on the amount of pixels he is responsible for. Thus, this limit is also fixed. If this condition is not satisfied, the model is destroyed.

The algorithm makes use of dynamically computed values. The previous chapter showed that the motions have the ability to compute their distance to a block. The mean distance from the chunks of a region to their corresponding motion is computed. These values are precious to give an indication about the region. An uniform region for example will have a small mean. The mean will thus be used to assign a chunk to a model.

4.1.3 Initialization phase

At this point, we have information neither about the motions nor about the zones. Thus, we have to create an initial state for the motion estimation algorithm. The frame is divided in a given number of initial regions and for each one, a motion is estimated.

To achieve our goal, we will make use of a block-based motion estimator that will create a motion field. The created local motion field contains already a segmentation of the frame that we will use. In addition to spacial

information, the motion vectors also store the mean square error of their node according to their translation.

To find the initial segmentation, a greedy algorithm (14) is used. The nodes with the smallest mean square error are taken. If a node has a small MSE, the probability that a correct motion was found is bigger.

Algorithm 14 Greedy algorithm used to find a list of node with minimal overall MSE.

```
Put the root node at position 0 of the list
while Not Finished do
  Find node with the biggest MSE in list
  n the number of children it has
  if number of nodes in list +i − 1 ≤ wanted number of nodes then
    remove the found node from the list
    add the children of the found node to the list
  else
    We are finished
  end if
end while
return the list and the number of nodes
```

Each node represents a part of the image, the coordinates they contain are used to fill the mask of zones and the array of pixels per region. For each region, we take some motion vectors and positions from the motion field and pass them to the corresponding motion in our list of motions to estimate the parameters. This task is done independently from the motion estimation algorithm; as described in chapter 3, a motion is responsible for the estimation of its parameters.

This procedure created n models. The more models the algorithm has to deal with, the higher the complexity is. Thus, for all pairs of created models, the distance between their motions is computed. If this distance is small, both models are merged together.

4.1.4 Update phase

An update phase occurs if we have access to past estimations of zones and motions. The goal of this part is to find the actual position of the zones. In order to do this, we make the hypothesis that the motion is constant.

First, we create a temporary zone mask that is filled with the non treated value. Then, for each non unassigned chunk at position (x, y) on the zone mask, we ask to the model which chunk at position (x', y') has the following property: $x' + mvx' = x$ and $y' + mvy' = y$ where $(mvx', mvy')^T$ is the motion vector at (x', y') according to the motion of chunk (x, y).

We assign the motion of (x, y) to (x', y') in the temporary mask. If

there is a conflict, the priority is given to the elder model. If the motions have the same age then the priority is given to the one having most pixels. This conflict case is illustrated by figure 4.1. If we still can not decide, the region with the smallest identifier is taken. In case of conflict between an unassigned chunk and a non unassigned, the second one is always the winner.

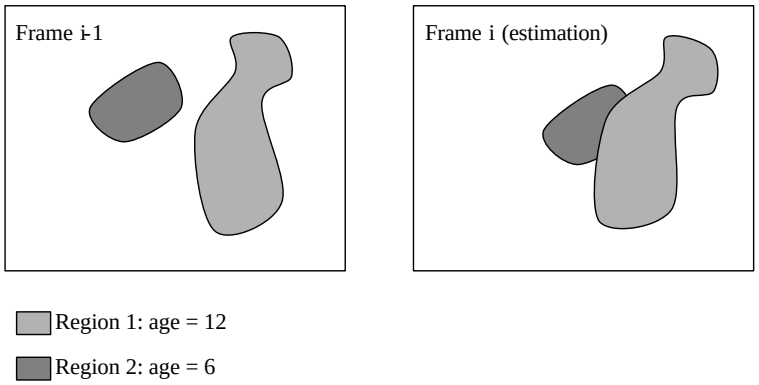


Figure 4.1: An elder region has the priority during the update phase.

An unassigned chunk at position (x, y) leads to an analog chunk at same position on the temporary mask if this position contains the not treated value.

At the end, all chunks that remain not treated move to the unassigned state.

The array that keeps the information about the number of pixels per region is updated when there is a conflict between two chunks.

4.1.5 Recursive phase

Refinement of parameters and random validity check

Three ways can lead to a refinement of the parameters. After an initialization, after an update or if we pass more than once through the recursive phase.

Apart from the first case, the refinement of parameters takes place after changes done on the zones by either the update or some borders check phases.

During the update, our hypothesis was that motion is constant. This hypothesis made it possible to get an estimate of the regions. In order to refine the motions, we will assume that the regions are constant and correct. From a general point or view, this motion estimation algorithm alternatively considers the motions or the zones as being correct and constant and tries

to refine the other ones.

The main part of the motion refinement is to choose the chunks to use. It uses the points defined as interesting for the block-matching. For each zone we fill a list of motion vectors and points as shown is algorithm 15. We take every chunk containing an interesting point and 1 chunk every N. We try to get the motion of uniformly distributed chunks.

Algorithm 15 Fill list of points and vectors for the parameters refinement.

```
for Each chunk in frame do
  counter = counter + 1
  if Contains interesting point OR counter == N then
    if counter == N then
      counter == 0
    end if
    if motion of current chunk is not unassigned then
      Ask motion to give us the vector at this point.
      Block matching with previous vector as initial state.
      Put the result and current point in list corresponding to the current
      model.
    end if
  end if
end for
```

Now, we have list of points and vectors that can be huge. K the number of points and vectors we have for a motion. If K is less than the fixed upper bound, we pass all point to the refinement function of the motion. If K is bigger, the algorithm takes as much points as possible equally distributed in the list. We assume that choosing equally distributed chunks in the list will give us equally distributed chunks in the regions.

The next step is independent from the motion estimation algorithm. The refinement of the parameters is up to the motion. The parameter refinement and estimation have a big influence on the behavior of the motion estimation algorithm. It decides if the priority is given to spacial or temporal persistence of the motion. Spacial persistence is achieved when it is assumed that the region associated to a motion has less chance to suddenly change than the motion. On the other hand, temporal persistence consists in assuming that a motion has less chances to change abruptly than the shape of the region. Spacial persistence has to be chosen. It often happens that the motion changes in a very fast manner from one frame to the next. If temporal persistence is chosen, it results in having false parameters for a region. This leads to the rapid destruction of the corresponding model.

The second task done in this phase consists in a random validity check. This is done for two reasons. The first one is to detect changes in the middle of a region that can not be detected by looking only at the borders

of a region. The second reason is to decrease the time needed to destroy a false model.

N chunks are randomly chosen on the frame. The distance to their model is computed. If this value is over a given threshold depending on the maximal distance allowed and the current mean distance of the region, we remove this chunk from the region.

This procedure creates holes in a region. These holes will be interpreted as borders by the border check phase. A false region contains many holes and is rapidly destroyed during the border check phase. This reflects the general goal of the algorithm which consists in creating candidates, modifying them, deleting the false ones and keeping the good ones. The random check decrease the time needed to delete false models and avoids having objects in a region that are not part of the model.

If a region contains objects having different motions, the parameters used during the update phase will be a mix of the motion vectors of these different objects. Thus, the resulting parameters are wrong and the whole model is destroyed. This is clearly not what we want!

Creation of new models

Moving regions can enter and leave the scene. Thus, the algorithm has to be able to create new models. If something different is entering the scene, the part containing the new object will be unassigned. If there are enough unassigned pixels, this function will try to create convex regions made of these pixels. Once this is done, motions are assigned to these regions and their parameters are estimated.

The purpose of the algorithm is to create many models and to choose the one having a chance to be region-based motions. The chosen ones then evolve and converge.

This phase of the motion estimation is described by algorithm 16.

Algorithm 16 create new models during the region-based motion estimation.

```
while enough unassigned pixels and limit of models not reached do
  find a rectangle of unassigned chunks.
  if size of the rectangle big enough then
    assign the chunks in the rectangle to an unused model.
    create a list of motion vectors and points for the chunks in the rectangle.
    estimate the parameters of the motion from these values.
  end if
end while
```

As for the initialization, the pairwise distance between the motion of all

models is computed and merging is done if possible.

Borders probability check

Changes in the shape of a region can only happen at the borders. This phase tries to expand or to shrink the regions. The border check phase can be seen as a simple classification problem. For each treated chunk the algorithm has to choose the best model in a list of possible models.

A border chunk is defined as being a chunk that has at least one neighbor that belongs to a different model. A chunk that has changed to another model is not treated again in the same border check. Three different cases are treated:

- Region X v.s. Unassigned
- Unassigned v.s. Region X
- Region X v.s. Region Y

It can happen that a chunk has more than one type of neighbor. In this case, the first encountered different model is chosen as neighbor.

For the first case, the current chunk belongs to region X. Its neighbor is unassigned. The algorithm must decide if the current chunk is near enough from model X to stay in it. This decision is taken in function of the distance the chunk has to the model, the mean distance chunks have to this model and the classification of the neighbors. The neighbors are taken in account to include spacial information. This case is illustrated by algorithm 17. If a motion has to search outside the image for a given chunk, the returned distance is an infinity symbol.

Algorithm 17 Border check phase: case 1.

```
compute the distance from current chunk to current model.
if distance is bigger than (maximum allowed distance + mean distance of
model)/2 then
  chunk is unassigned
  if numbers of same class neighbors ≥ 7 AND distance smaller than
  maximum allowed distance then
    chunk belongs to model X
  end if
else
  chunk belongs to model X
end if
```

The condition on the distance depends from the mean of the tested model. This comes from the fact that this mean holds information about the other chunks in a model. A chunk can be over the first computed distance

but stay in the current model if its neighbors are mainly in the same model. This is done to keep local spacial consistency.

The second case (algorithm 18) deals with unassigned chunks that are candidates to enter the motion. The decision is taken in a very similar way as in the first case.

Algorithm 18 Border check phase: case 2.

```
compute the distance from current chunk to the other model.
if distance is bigger than (maximum allowed distance + mean distance of
model)/2 then
  chunk is unassigned
  if numbers of other class neighbors ≥ number of unassigned neighbors
  AND distance smaller than maximum allowed distance then
    chunk belongs to model Y
  end if
else
  chunk belongs to model Y
end if
```

The last case happens when two zones are adjacent. The algorithm has to find the best fitting model for a chunk between two models or to change the chunk to unassigned if non of the models fits. In addition to the values needed to decide in the two first cases, the algorithm also takes in account the ages of the concurrent models. If both models fits well to the chunk, the priority is given to the elder one. The decision procedure is described by algorithm 19.

Model elimination

The previously phases try to create and refine the population of models at our disposal. The model elimination phase has to choose the ones that do not correspond to our expectations.

The first step of the elimination consists in looking for models to merge by checking pairwise the distance between the different motions. The second step is the real elimination. It is done by checking constraints on the models. The algorithm eliminates the models which:

- have a very small region
- are old but never grew up to a given size
- lost a big amount of pixels in one step

Deleting the bad models will permit the correct ones to grow. This step leads to the creation of new unassigned chunks. From these, the algorithm

Algorithm 19 Border check phase: case 3.

```
compute the distance from current chunk to model X.
compute the distance from current chunk to model Y.
if distance to model X is infinity then
  if distance to model Y ≤ maximal allowed distance then
    chunk belongs to model Y
  else
    chunk is unassigned
  end if
else if distance to model Y is infinity then
  if distance to model X ≤ maximal allowed distance then
    chunk belongs to model X
  else
    chunk is unassigned
  end if
else
  if distance to the younger model ≪ distance to the elder model then
    chunk belongs to the younger model
  else
    if distance to the younger model < distance to the elder model then
      if amount of neighbors belonging to the elder model > amount of
      neighbors belonging to the younger model then
        chunk belongs to the elder model
      else
        chunk is belong to the model whose mean distance is nearest to
        the computed distance
      end if
    else
      chunk belongs to the elder model
    end if
  end if
end if
```

will be able, in another recursion, to create new models which will perhaps grow into a new region-based motion.

4.1.6 Creation of the container

The last task of our algorithm is to create the container described in chapter 3 to encode the results. The algorithm may internally work with a huge amount of different models regarding to the number of models it encodes at the end. This step chooses the models that are worth encoding.

The encoding constraints are fixed. A lower bound on the number of

pixels a model has to contain in order to be encoded is defined. Thus, there is a tradeoff between the gain a model adds to the system and the amount of information needed to encode it. An upper bound on the number of models to encode is also fixed. This part will choose the most valuable models to encode according to this limit.

For the encoding, a model X is better than a model Y if the number of pixels in the region of X is bigger than the number of pixels in the region of Y and if the motion of the model X is not corresponding to a null motion.

The algorithm creates the container according to these conditions and to the encoding constraints.

Chapter 5

Annex problems

5.1 Using half-pel

5.1.1 Problem definition

Using half-pel during the estimation leads to more precise motions. However, it also increases the mean square error between the reference frame and the compensated one. Half-pel tries to recreate the missing information. To do this, it interpolates pixels. Thus, the mean square error between the interpolated pixel and the real one is high.

The goal of this section is to find a prefilter that would reduce the impact of interpolation on the error. Assume one has access to an image in full size and wants to encode a downsampled version of it while using half-pel. Assume also that the decoder has access to the original image in full size. Thus, it can compute the mean square error generated by the interpolation.

The key idea of the filter is to insert some information about the missing pixels into the downsampled version to the image. Including information can be done by applying a prefilter H before downsampling the image.

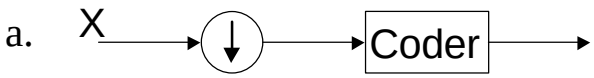
This section shows the existence of an optimal prefilter in the general case. Numerical results are also shown.

Assume we have access to an image X delivered by a given source in full size and that we want to encode it in CIF. Figure 5.1 shows the traditional and the the proposed way to do this.

We want to know the error due to the interpolation. Figure 5.2 shows how this error is computed. Our goal now is to find the filter H which minimizes this error. If the filter H is equal to 1, we are in the first case where no prefiltering is done.

5.1.2 Optimal prefilter

The best filter H is such that the MSE between Y and X is minimal. Let us call this this value $e(H(z))$. Thus, the function to minimize in the z-domain



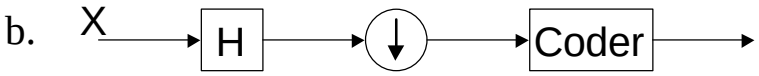


Figure 5.1: Part a. shows the normal downsampling. Part b. shows the proposed solution with prefiltering.

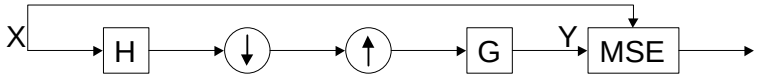


Figure 5.2: Compute the error due to the interpolation represented by the upsampling.

is

$$e(H(z)) = E\left[(Y(z) - X(z))^2\right] \tag{5.1}$$

where $Y(z)$ is

$$Y(z) = \frac{X(z)H(z) + X(-z)H(-z)}{2}G(z) \tag{5.2}$$

The filter used for the interpolation is also known

$$G(z) = \frac{1}{2}z^{-1} + 1 + \frac{1}{2}z \tag{5.3}$$

From this equation, we rapidly deduce that $G(z) = \overline{G(z)}$.

First, by developing the term in the expectation of 5.1, we obtain the following equation

$$\begin{aligned} e(H(z)) &= (Y(z) - X(z))\overline{(Y(z) - X(z))} \\ &= Y(z)\overline{Y(z)} - Y(z)\overline{X(z)} - \overline{Y(z)}X(z) + X(z)\overline{X(z)} \end{aligned} \tag{5.4}$$

To minimize the error, we want to find the derivative of $e(H(z))$ in function of the filter $H(z)$.

$$\frac{\delta e(H(z))}{\delta H(z)} = \frac{\delta Y(z)}{\delta H(z)}\overline{Y(z)} + \frac{\delta \overline{Y(z)}}{\delta H(z)}Y(z) - \frac{\delta Y(z)}{\delta H(z)}\overline{X(z)} - \frac{\delta \overline{x(z)}}{\delta H(z)}Y(z)$$

$$
\begin{aligned}
- & \frac{\delta\overline{Y(z)}}{\delta H(z)}X(z) - \frac{\delta X(z)}{\delta H(z)}\overline{Y(z)} + \frac{\delta X(z)}{\delta H(z)}\overline{X(z)} + \frac{\delta\overline{X(z)}}{\delta H(z)}X(z) \\
= & \frac{\delta Y(z)}{\delta H(z)}\left(\overline{Y(z)} - \overline{X(z)}\right) + \frac{\delta\overline{Y(z)}}{\delta H(z)}\left(Y(z) - X(z)\right) \qquad (5.5)
\end{aligned}
$$

Let us now look at the terms of the right part of the previous equation. For the first term

$$
\begin{aligned}
\frac{\delta Y(z)}{\delta H(z)} &= \frac{\delta H(z)}{\delta H(z)}X(z)G(z)\frac{1}{2} + \frac{\delta H(-z)}{\delta H(z)}X(-z)G(z)\frac{1}{2} \\
&= \frac{1}{2}G(z)\left(X(z) + \frac{\delta H(-z)}{H(z)}X(-z)\right) \qquad (5.6)
\end{aligned}
$$

The second term becomes

$$
\frac{\delta\overline{Y(z)}}{\delta H(z)} = \frac{1}{2}G(z)\left(\frac{\delta\overline{H(z)}}{H(z)}\overline{X(z)} + \frac{\delta\overline{H(-z)}}{H(z)}\overline{X(-z)}\right) \qquad (5.7)
$$

In order to simplify our problem, we can make the following hypothesis on $H(z)$: $\overline{H(z)} = H(z)$. We now have a simpler equation for 5.7.

$$
\frac{\delta\overline{Y(z)}}{\delta H(z)} = \frac{1}{2}G(z)\left(\overline{X(z)} + \frac{\delta H(-z)}{H(z)}\overline{X(-z)}\right) \qquad (5.8)
$$

Even, with these simplification, the analytic solution remains difficult to find. Thus, let us now look in the next part at a numerical solution found.

5.1.3 Numerical solution

In order to solve the problem numerically, we have to fix some parameters for H and make hypothesis on the spectral density $X(z)$ of the input signal. Let us assume from now that X is white noise; so we have that $X(z) = 1$. The desired filter is $H(z) = a0z^{-1} + a1z^{-1} + a2 + a1z + a2z^2$.

Under these conditions, we find that the parameters of the optimal filter are:

- $a0 = -0.11764705882352941176$
- $a1 = 0.26470588235294117646$
- $a1 = 0.70588235294117647060$

Figure 5.3 clearly shows us that this filter is low-pass one. These parameters are very similar to the famous Cohen-Daubechies-Feauveau (5,3) wavelet shown on figure 5.4.

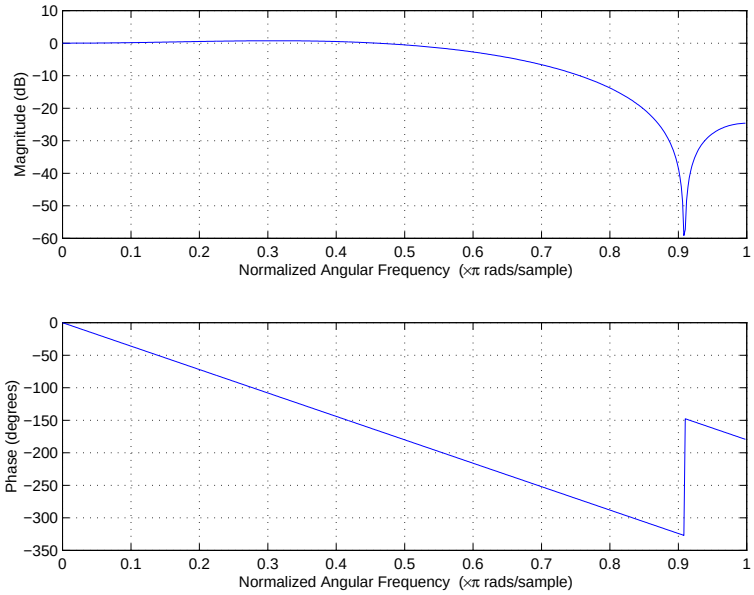


Figure 5.3: Frequency response of H

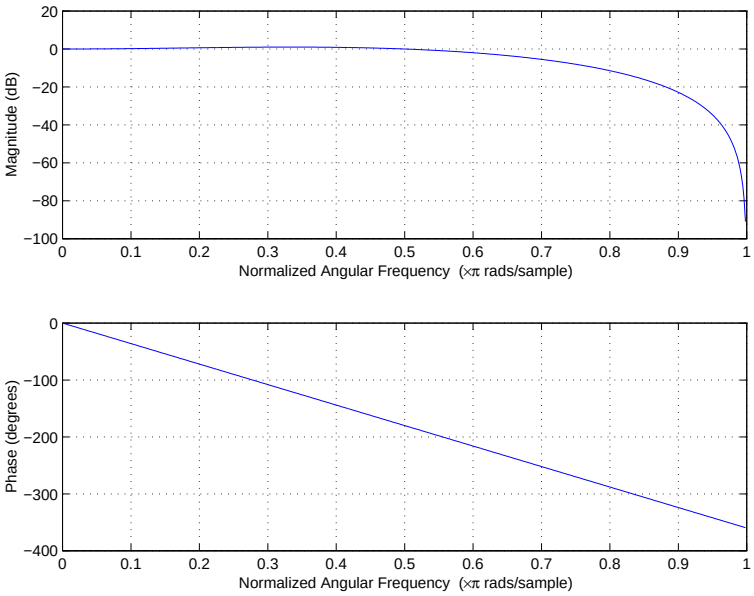


Figure 5.4: Frequency response of Cohen-Daubechies-Feauveau (5,3) wavelet $[-1/81/43/41/4 - 1/8]$.

	Mean Square Error				
Filter	**Image**	**No**	**Horiz**	**Vert**	**Diag**
Found	17.938957	5.326927	13.105896	23.170330	30.152675
[1]	22.353552	0.000000	17.382696	33.541153	38.490359
[1/4 1/2 1/4]	27.051030	13.256355	22.625163	32.476936	39.845666
[1/2 1/2]	45.195176	38.555800	41.376480	48.430684	52.417741

Table 5.1: Mean square errors with different prefilters.

5.1.4 Results

The following table shows different numerical values of the mean square error for different filters H. The image used is lena. The mean square error is computed between X and Y in a YCbCr color space on the Y plane. There are five results per analyzed filter. The first analyzed filter is the one we found. The second corresponds to apply no filter. Two other filters have also been tested. This permits to compare the results to simple low pass filters.

The different possible shifts are shown in figure 5.5. The column Image is the mean square error between X and Y on the whole image. The four other values reflect the possible kind of half-pel interpolation.

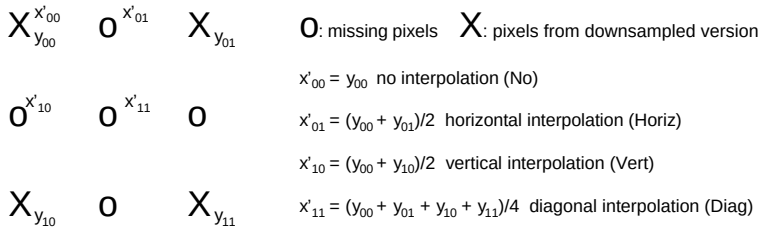


Figure 5.5: Different possible interpolations.

The second line correspond to the previously cited case where no prefiltering is done. As one can see, in this case, the mean square error on the pixels where no interpolation is done is null but it is high in the other cases. The first line stores the results when we used the found filter. One can remark that the MSE is increased on the pixels where no interpolation is done. The other terms are a lot lower. The main point is that the variance of the found filter (89.7410) is smaller than the variance obtained when no prefiltering is done (227.4874). Thus, if different kinds of interpolations are done on an image, the error will be more uniform.

The third filter has a small variance but the mean is higher than the one of the first and of the second filter. The last tested filter has the smallest

variance (30.3244) but its mean is the highest. The two added filters are clearly low-pass ones thus, the resulting downsampled image is blurred. The found filter is also a low pass but the cutting frequency is high. Thus, the resulting downsampled image is not too blurred.

As shown here, the found filter reduces the MSE resulting from a interpolation. The variance of the created error is also smaller. Thus, it is a good idea to use such a filter. In real-time applications, it is important to keep in mind that filtering the images increases the complexity. If one has access to the full images from a camera, it could be possible to implement this filter in the hardware.

Chapter 6

Results

This chapter presents the results for proposed region-based estimation algorithm. They are derived from the measure of the mean square error between the received frame to encode and the encoded one. No pre-filtering is done.

The used block-based motion estimator is the one of VisioWave. The block-matching is done on blocks of size 8x8. The motion field produced is of the same type than the one presented in chapter 3.

The region-based motion estimation is followed by a block-based refinement. The resulting motion field is corrected the following way: the motion vectors of the blocks whose pixels mainly belong to a region-based motion are put to null.

There is no objective measure for the improvement in the visual quality. Thus, for the results I decided to focus more on the aspect of compression. One has to keep in mind that the main goal we wanted to achieve is a better visual quality at a given bit-rate.

This chapter often refers to the quality at which a sequence is encoded. This value determines the quantization step used for the spacial encoding. Each time the quality is increased by 4, the resulting PSNR increases of 1 dB.

As shown in the general flowchart about the coder used in chapter 2, it makes use of a rate control module. This module minimize the following Lagrange multiplier to adapt the quality:

$$L = Rate + \lambda MSE$$

If for all qualities this measure is smaller for a given motion estimation method, one can say that it is better than the other method. For each tested sequence, the rate/distorsion curve, the rate/PSNR and the Lagrange multiplier are shown.

There are natural and artificial testing sequences. The artificial one, also enable to show the quality of the parameters estimation. On the other hand, natural sequences lead to more realistic results.

The region-based motion estimation for translation uses the translation and zoom motion described in chapter 3. Translation should give us less improvement than the other types of motions. This is due to the fact that a translation can be well matched by the block-based motion estimator. In addition to this, as we have seen in the chapter about the local motion field, the node structure has the property to group the nodes if their motion vectors are identical. This means that for a global translation, this structure will use only few bytes. Thus, the gain one can hope by using a region-based approach are not exceptional.

The testing sequence is an artificial one. It is made of a picture (lena) moving from left to right. Figure 6.1 shows the MSE obtained at a given rate for the region-based and for the block-based approach. The obtained gain with the region-based algorithm is high a low-bit rate. The second part of this figure shows the PSNR in function of the rate. The motion is really uniform on the whole frame and detected as so by the algorithm. Thus, the gain is very high. About 6dB at 1000 bits/sec. Figure 6.2 show the function that the internal rate control of the coder tries to minimize. At each different qualities, the region-based motion estimator is better. From the point of view of the coder, the region-based approach is better.

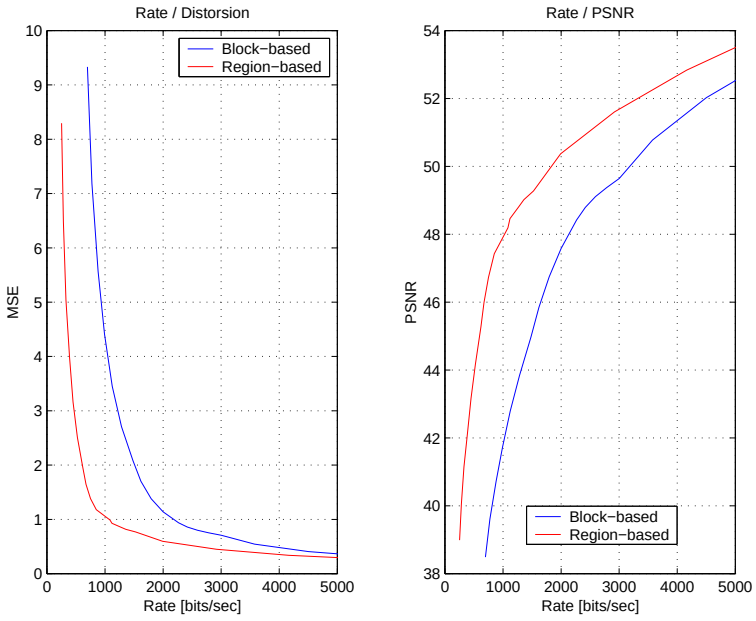


Figure 6.1: Artificial translation.

Figure 6.3 compares a compensated framed for both methods. The region-based compensated image looks more low-passed but has an overall better visual quality. Even is the block-based compensated image is less

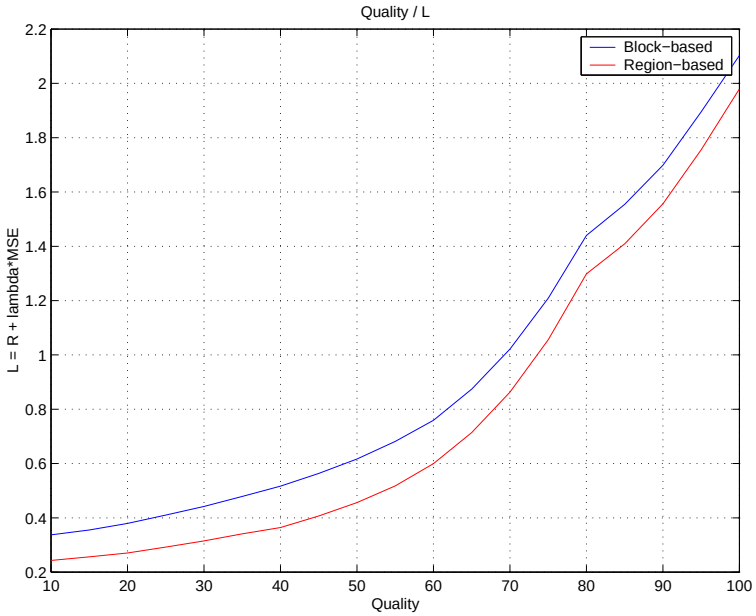


Figure 6.2: Artificial translation.

blurred, the visual quality is bad due to the blocks appearing. At low bit-rate, these artifacts can persist even after the spacial encoding of the error. The Human Visual System is very sensitive to these regular patterns.

Figure 6.3: Block-based and region-based compensated frame.

The improvement in term of compression is achieved by reducing the size needed to pass the information about motion. This is done by describing motions with parameters and regions instead of using vectors for each block of the image. The main cost is the encoding of the regions.

The next results are obtained for more complex motions. The region-based approach enables to describe them. It is not possible to describe these

motions precisely with the block-based motion estimator.

Rotation can not be precisely expressed using block matching without using blocks that have the same size as a pixel. In our case, the blocks have a size 8x8 pixels. As for the translational case, using the region-based motion estimator leads to an improvement in term of compression.

This rotation is made of a static image that rotates. Thus, the real rotation parameters are known and can be compared to the obtained ones. The motion created is uniform on the whole frame. The motion estimator detected a unique motion covering most parts of the image. It also detected the entering and outgoing parts and did not include them in the region.

The rotation is made of three parameters: the center (cx, cy) and the angle δ. The parameter that needs to be very precise is the angle. Figure 6.4 show us the real angle and the found one. An encouraging fact is that at the first frame, the angle is correctly found. The mean is very near from the real angle. The most important fact is that the variance is low! Thus, we can say that the parameters found by the estimation algorithm correct. Figure 6.5 shows the distribution of the center on the image. The real center is in the middle of the frame $(180, 144)$. The found results are good. Three blobs appear on this figure. This is due to the fact that the genetic algorithm has the property to search for the minimum near from its initial state.

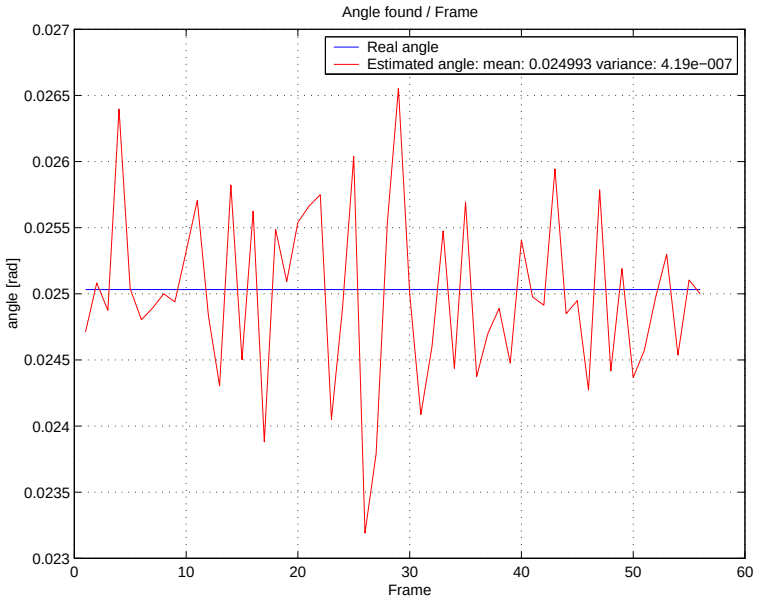


Figure 6.4: Real and found angle of a rotation.

Figure 6.6 shows the rate/distorsion and the rate/PSNR curve. For a given rate, the MSE obtained while using the region-based motion estimation is always smaller. Thus, the PSNR is also better for this approach. The

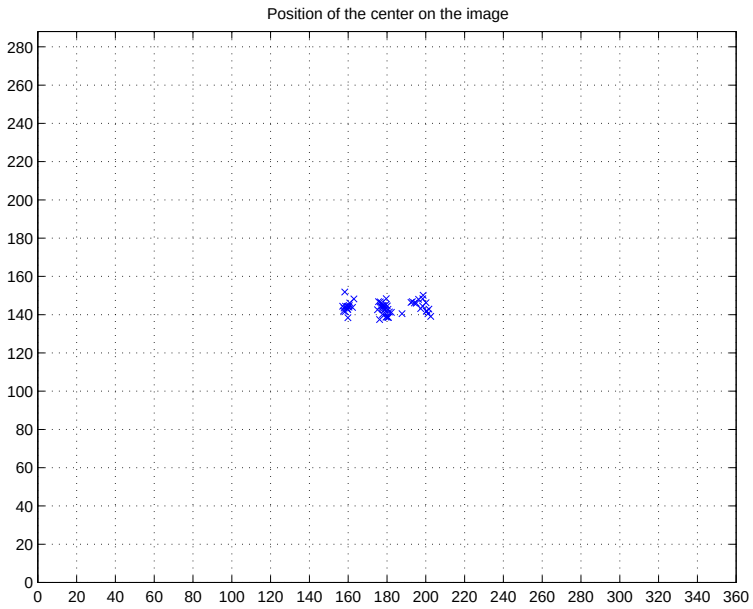


Figure 6.5: Distribution on the image of the center of a rotation.

gain is relatively uniform and is about 2dB. The rotation used is a slow one. This is the reason that block-matching approximates it not too badly.

For the lowest quality, the block-based approach has a bit-rate of 515 bits/sec. For the same quality, the other methods lead to a bit-rate of 414 bit/sec and a gain of about 0.65dB for the PSNR. Thus, the developed method if a good alternative to block-matching at low bit-rate.

The last graphic show the function that is minimized by the rate control function of the codec. As one can see, the L value is always smaller if the region-based motion estimation is enabled (illustrated by figure 6.7). Thus, also for this sequence, the region-based motion estimation leads to better results.

In the implementation, we made the following assumption about the zoom. Its origin is due to the camera. This hypothesis leads to the fact that its center is in the middle of the image. Thus, the zoom consists in only one parameter, the zoom factor. In this artificial sequence, the camera does nothing at the beginning then zooms in the image an then zooms out again. The zoom is not linear. Figure 6.8 shows us the results we obtain. For the zoom, the gain we have is high. It is about 5dB at low bit-rate and about 2dB at high bit-rate. If we take only the zoom in, the gain is

For the biggest quantization step, the bit-rate obtained with the block-based approach is 291 bits/sec. Under the same constraints, the other methods lead to a bit-rate of 141 bit/sec. There is also a gain of about 0.4dB if

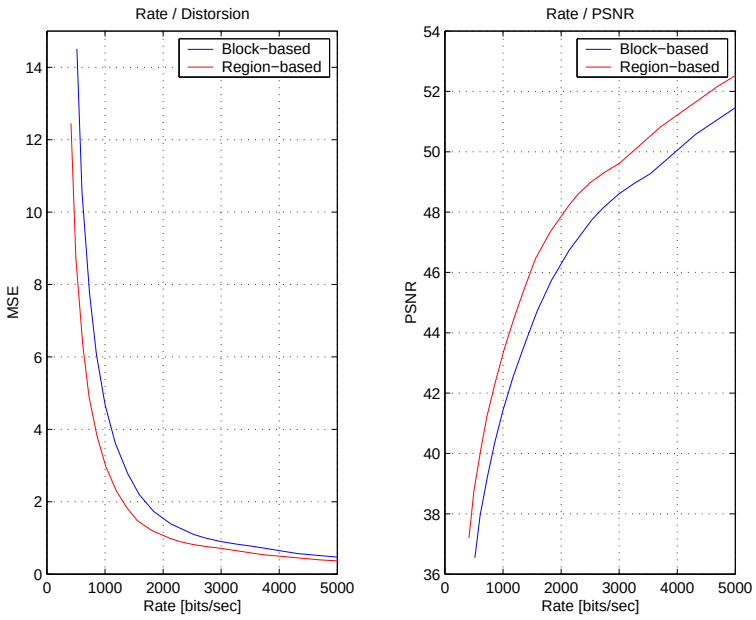


Figure 6.6: Artificial rotation.

the region-based motion estimator is used. At a bit-rate of 291 bits/sec, the developed algorithm lead to a gain of about 5dB.

Let us now look at a motion that is a combination of zoom and translation. This motion is described in detail by an example in chapter 3. A natural sequence was used. Figure 6.10 shows the results we have. The gain is lower than the ones we have on artificial sequences. It is constant and about 2dB for the different bit-rates. The internally minimized Lagrange multiplier is lower for the proposed solution. This shows that this method also works well with natural sequences.

Figure 6.12 shows a frame of the considered sequence. We can see two found regions the big one (red) has a zoom factor bigger that 1 which means that the camera is zooming out and the smaller region has a zoom near from 1 and translation factor of $(-6.0, -3.3)$. The first zone also has a translation factor. The camera does not only zoom out. It has also a translational motion of $(5.4, -4.9)$.

For sequences that contains no or very few motions corresponding to a region-based motion, the results are near from the results of the block-based coder. This is mainly due to the fact that in this case, we will encode no regions or very small ones. Thus, the rest of the estimation is done by the block-matching.

The main improvement of the proposed algorithm is for low bit-rate sequences. This is mainly due to the fact that it uses only few bits to

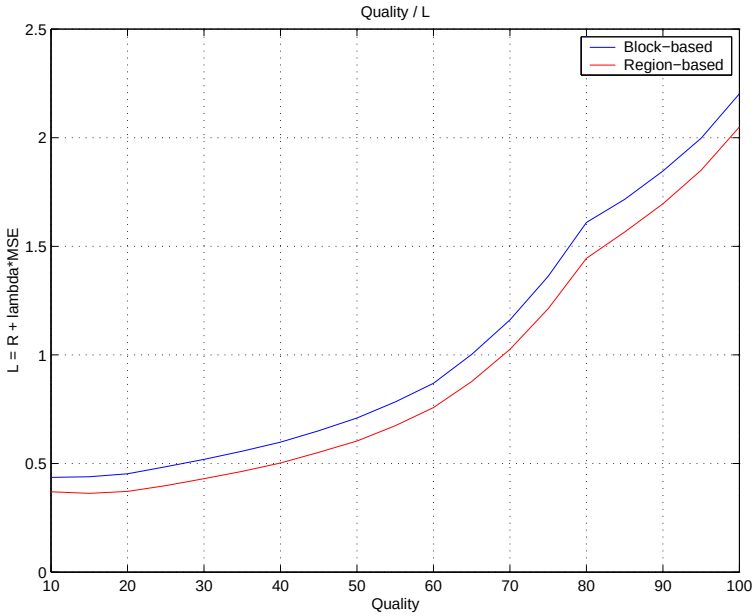


Figure 6.7: Artificial rotation.

encode the motions and the regions. As show in figure 6.3, the proposed solution creates less visible artifacts. Thus, at low bit-rate, the visual quality is improved.

All the results we have are based on the mean square error between the encoded frame and the input. The block-matching tries to minimize this value without taking in account the real motion on a frame. The error produced by the block-matching creates a *blocky* effect on the image, meaning that the error is geometrically located. The human visual system is very sensitive to regular patterns. The region-based approach compensates bigger parts of the image with the same parameters and does not lead to the appearance of geometric artifact. Even if its error is higher, its distribution on the image is not geometric. Thus, the visual quality is better. It has been shown that the MSE is not the best way to measure the quality of a codec.

Nowadays, the networks have a high bandwidth and in the future, it will even be higher. Trying to minimize measure like the mean square error leads to a better compression. Perhaps, in the future bandwidth will no longer be a constraint. Thus, I think more efforts should be put in the visual quality improvement at fixed bit-rate. The mean square error does not take in account the visible impact of the error on a human. In order to quantify the visual improvements, we should make use of psycho-visual tests on an important number of persons with different knowledge in the domain

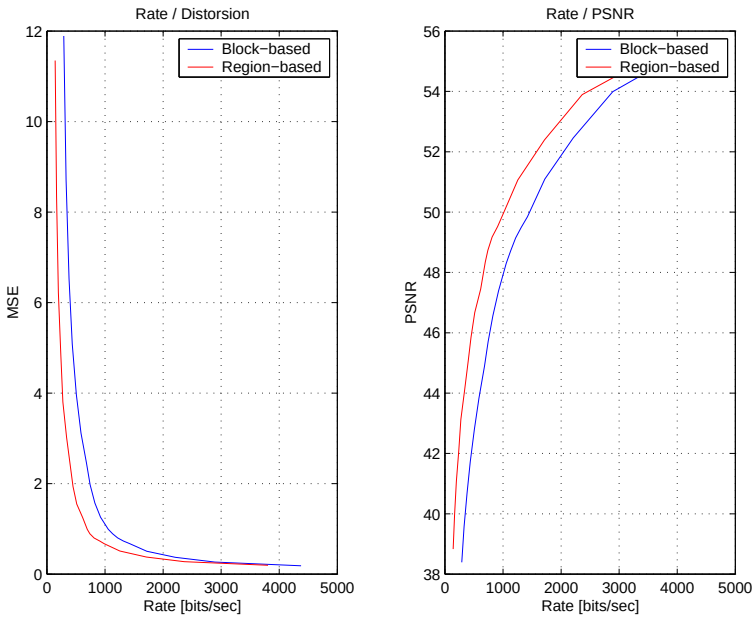


Figure 6.8: Artificial zoom.

of video compression.

The first tested sequences have a unique global motion. The developed algorithm has the ability to deal with this case. It is not limited by the shape a zone could have. This kind of property is a non-negligible one, it shows somehow that the estimator matches very well most of the region-based motions that can exist in a real video sequence.

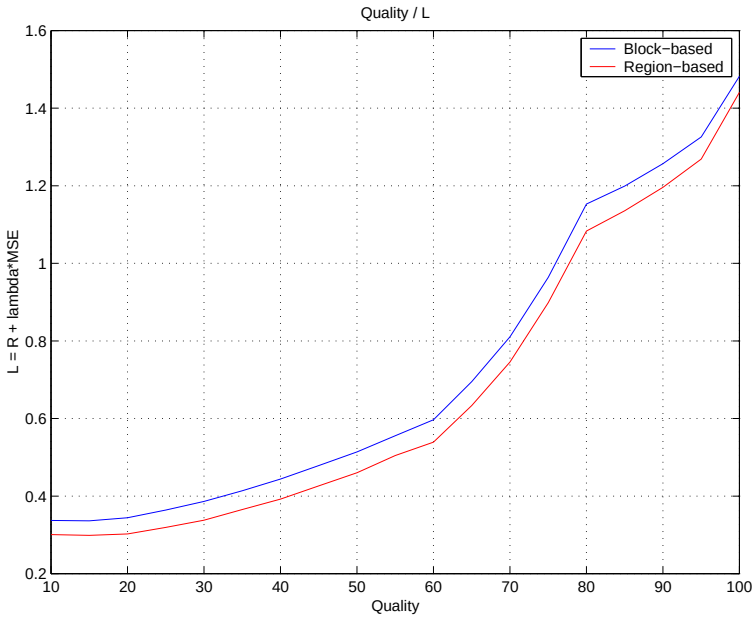


Figure 6.9: Artificial zoom.

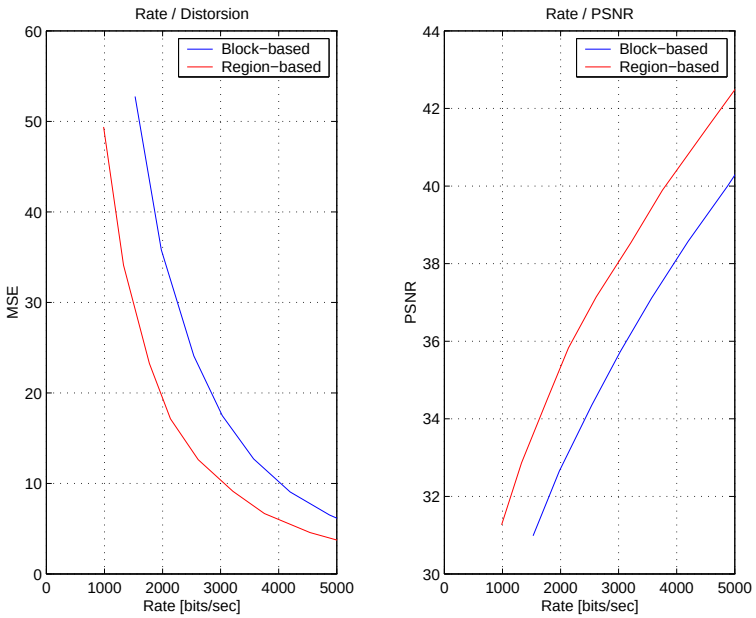


Figure 6.10: Translation and zoom. Natural sequence.

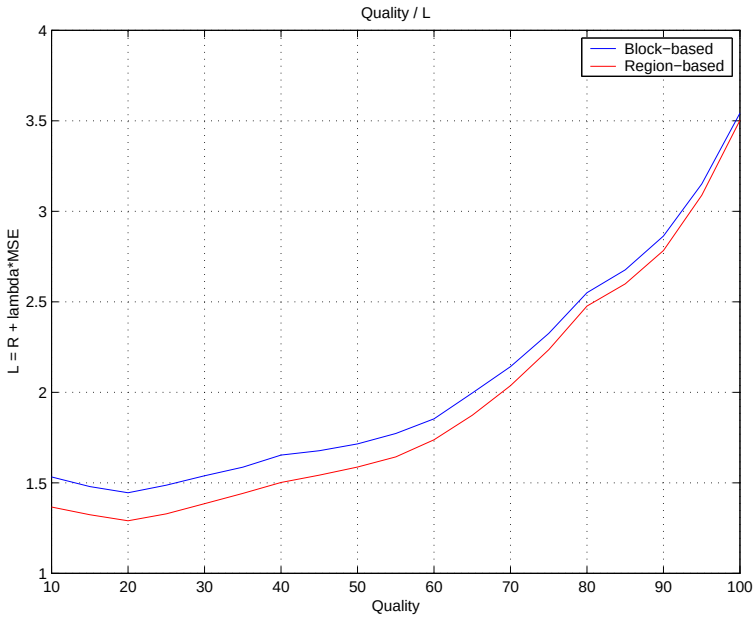


Figure 6.11: Translation and zoom. Natural sequence.

Figure 6.12: Translation and zoom. Example of region and motion found.

Chapter 7

Conclusion

During this work, I realized the importance of spending time on the creation of a structure. The constructed framework for motion enables to add new motions and new zones easily. The structure also simplifies the block-based motion estimation and compensation.

The compression results are good. It has been shown that a more global approach to the problem of motion estimation lead to better visual quality at same quantization step for the spacial encoding.

Results on compression are rather encouraging. The region-based motion estimator whose goal is not only to minimize the mean square error also leads to a gain in compression. One has to remember that this was not our main goal. We wanted to achieve a better visual quality at low bit-rate.

Many perspectives are still open. A lot of improvement are possible. An interesting notion to introduce into this project would be a confidence value for the regions. One could imagine that if we are enough confident in an estimation, we do not encode any error for this part of the image. This kind of feature is not imaginable for block-matching which does not try to achieve spacial consistency.

In this implementation, the algorithm works with elementary parts of the image of size 2x2 pixels. An interesting point for the future would be to extend this algorithm to work with chunks of different sizes regarding the task it does. One could imagine to have a kind of a multi scale approach of the region refinement. To do a preliminary border check phase with big chunks then to od it again with smaller chunks. I think, this would increase the speed of the algorithm and the precision of the regions.

This diploma project gave me the possibility to get more familiar with the different aspects of video coding and motion estimation. It was also a precious professional experience.

Appendix A

About VisioWave

Founded in 1996 around real-time digital video compression and high speed networks, VisioWave designs, produces and markets hardware and software products for digital video networking applied to the :

- **security** market (Surveillance and CCTV)

- **media and entertainment** market (Multimedia intranets, Interactive TV, Corporate TV, Video learning, Video medicine, Video forums).

VisioWave is offering hardware and software solutions for compression, processing and real-time transmission of high quality video over any network (IP, ATM, ADSL, cable, wireless, etc.)

VisioWave is now developing new generation 3D wavelets compression technologies in collaboration with the Laboratory of Signal Processing of the Federal Institute of Technology of Lausanne.

Since July 2000, TF1 group, number one European TV Broadcaster, is the reference shareholder of VisioWave. TF1 provides VisioWave with the necessary vision and market requirements for its development in the area of new interactive TV services over broadband networks and all related possible activities.

The company is incorporated in Switzerland with subsidiaries in France and Canada, and counts about 60 employees.

Digital Video Networking consists in encoding and compressing video streams from a variety of sources (cameras, DVD, VCR, satellite TV, cable TV, etc.) and transmitting them over any kind of digital networks.

This transmission generally takes place alongside voice applications (telephony) and conventional data applications (e-mail, file transfers, database

access, ...). These digital video streams are viewed on workstations or TV sets, using specific applications or standardized interfaces such as web browsers.

Both digital video processing and network technologies are applied to Media and Security Solutions :

VisioWave Security Solutions :

- CCTV and video surveillance
- Video Supervision
- Traffic and flow management

VisioWave Media Solutions :

- Multimedia intranets, corporate TV
- Interactive video forum
- Video learning Video medicine

Appendix B

Structure objects models

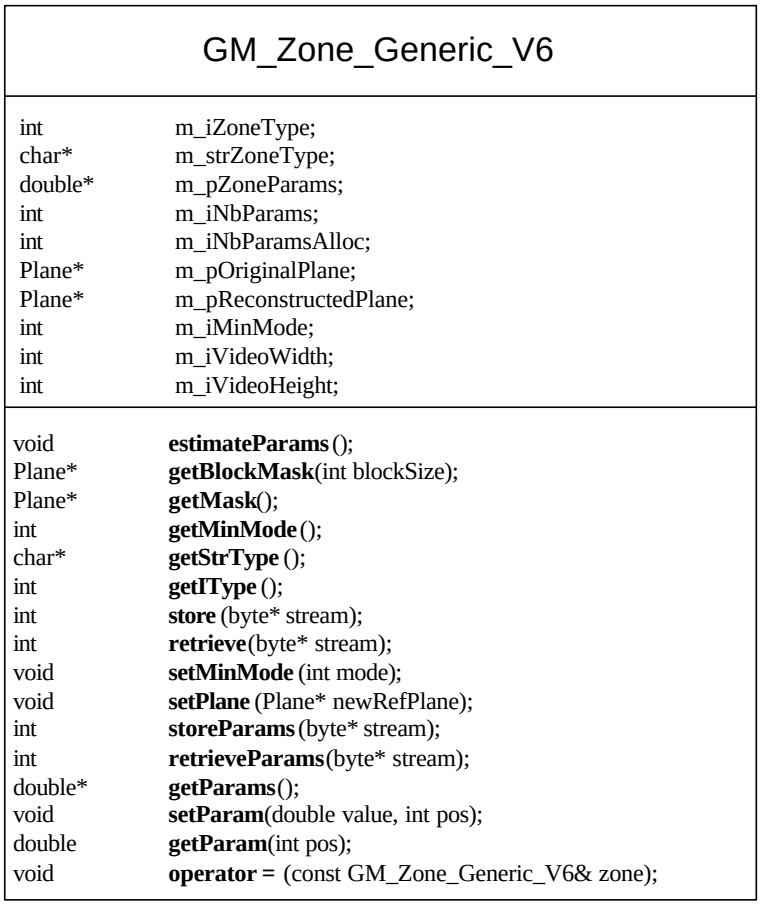


Figure B.1: Interface of a generic zone.

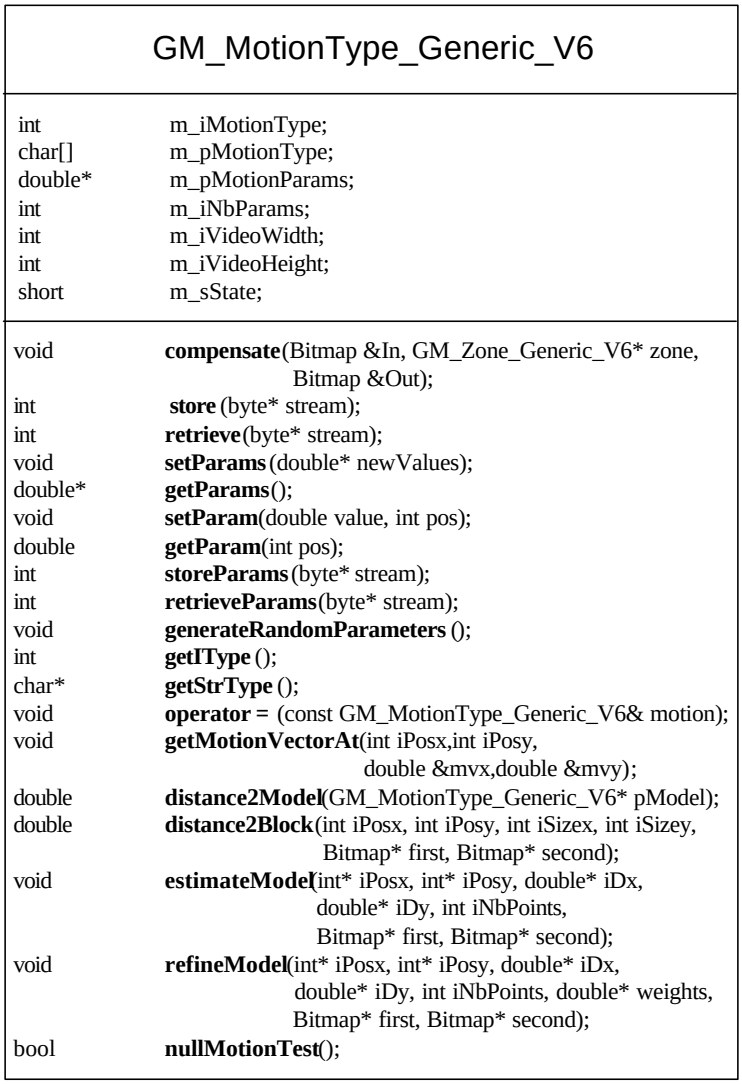


Figure B.2: Interface of a generic motion.

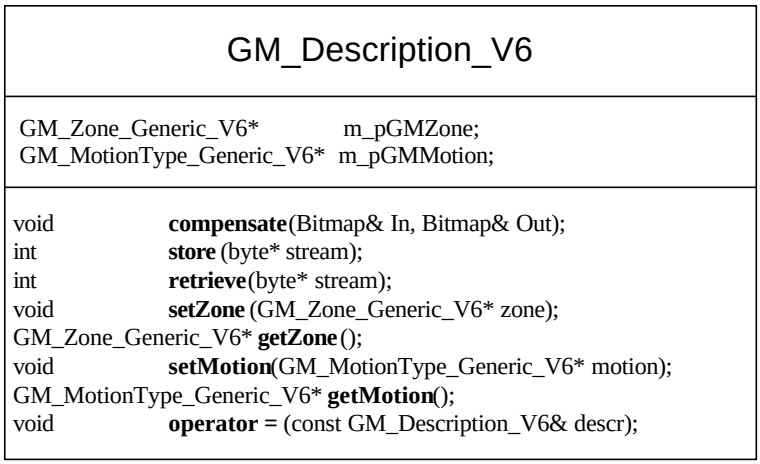


Figure B.3: Interface of the motion description.

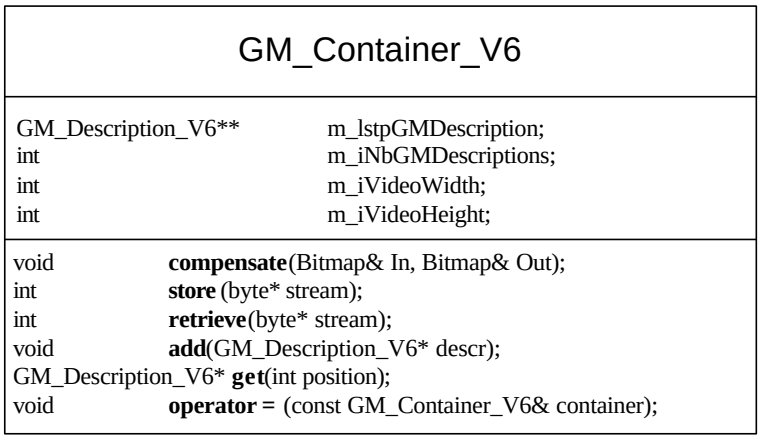


Figure B.4: Interface of the container.

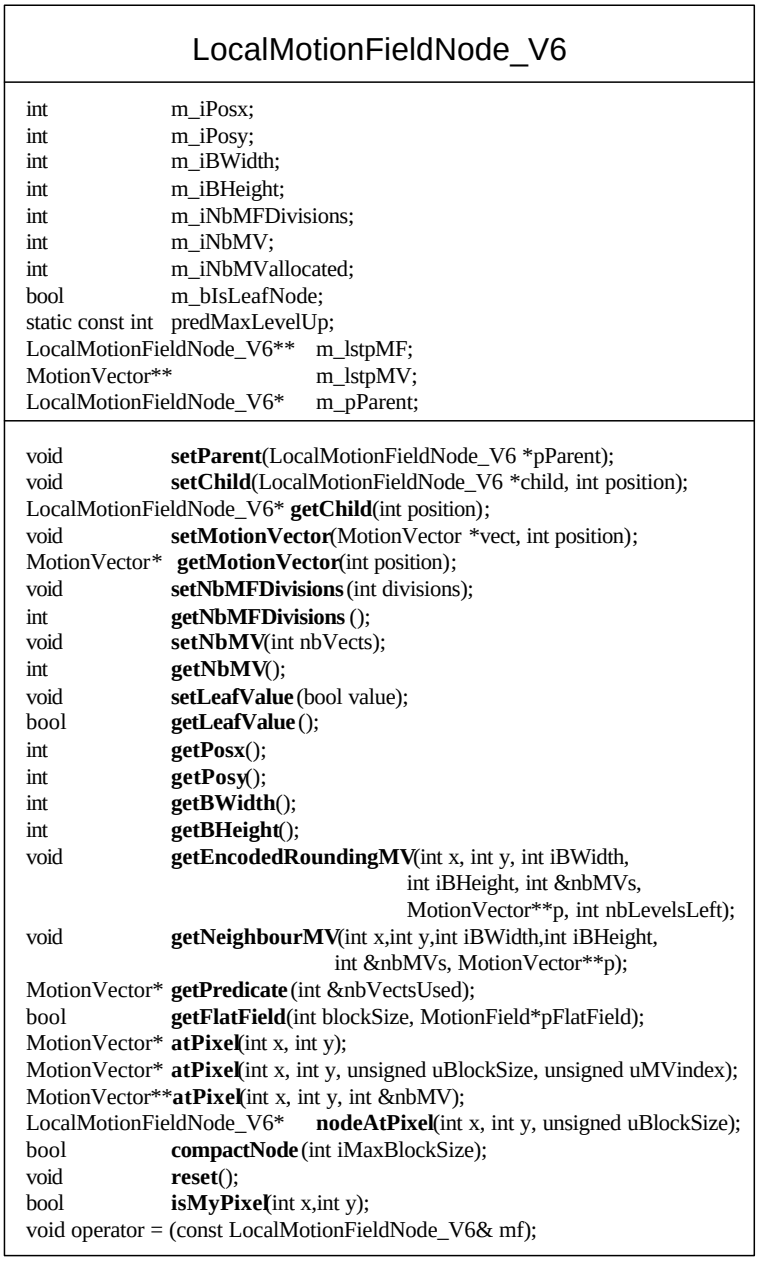


Figure B.5: Interface of the local motion field node.

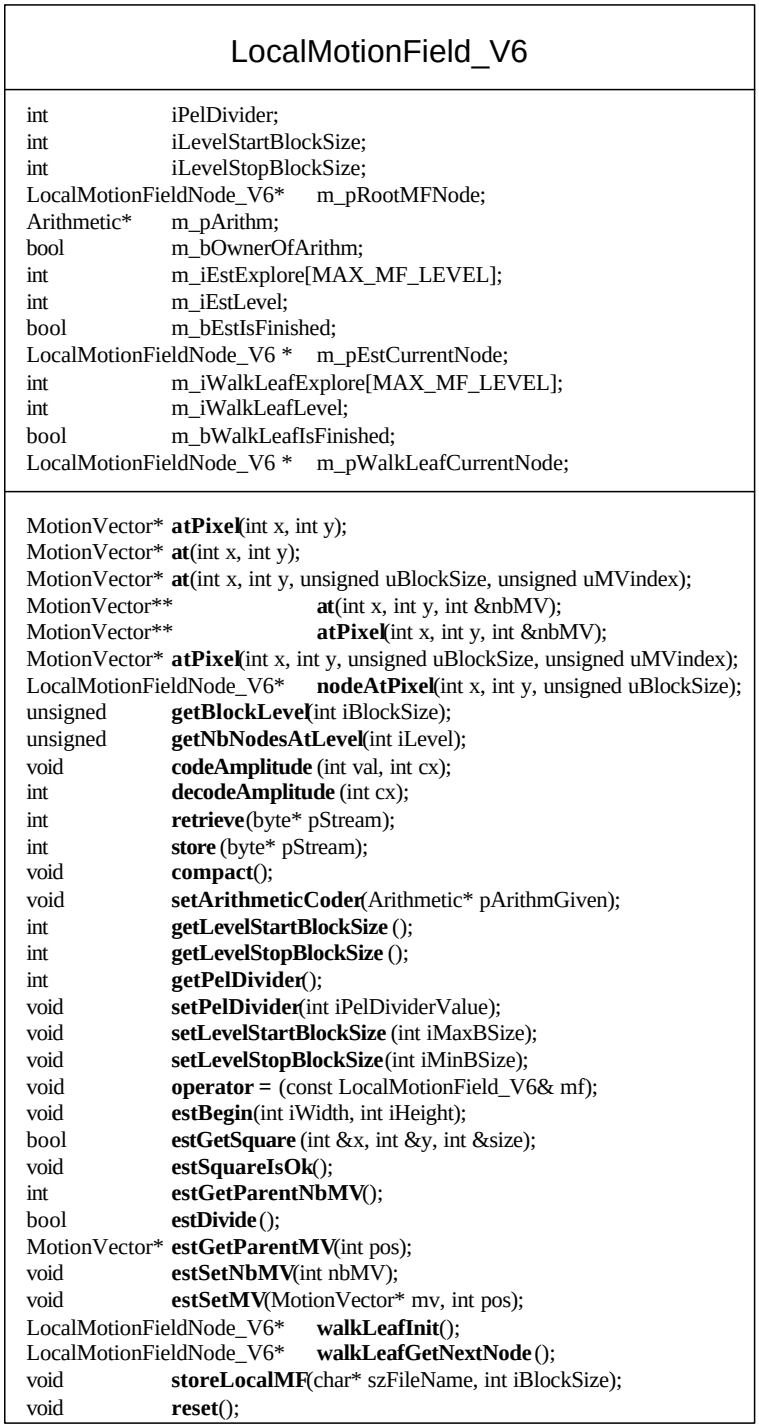


Figure B.6: Interface of the local motion field node.

Appendix C

The local motion field interface

This appendix shows some of the functions and variables of the `LocalMotionField_V6` object described by figure B.6.

Two simple examples included in this chapter illustrate the implementation of the Local Motion Field interface. The first example is a multi-scale motion estimator. The second one is about accessing the data contained in the nodes through this object.

The purpose of the functions containing `Est`in their name is to simplify the task of motion estimation. The simple motion estimator we will look at is described by algorithm 20. The estimator does not need to have the knowledge of the structure. The local motion field is responsible to deliver him estimable blocks. We also see that the estimator accesses in a very simple way to data to get an initial motion vector for example.

The algorithm laying behind the `Est` functions walks through the tree, stops on each node and waits for the user to tell him what it has to do. They share the following global variables:

`m_pEstCurrentNode` pointer to the current node. It is the one currently proposed to the estimator.

`m_iEstLevel` the depth in the tree where the current node is located.

`m_iEstExplore` array of integer containing the amount of remaining subtrees to propose for each node on the path from the root to the current node.

`m_bEstIsFinished` a boolean value set to *true* when the estimation is finished.

The behavior of each `Est` function.

Algorithm 20 Simple motion estimator

create a empty data structure. (`estBegin()`)
while go to the next estimable block (`estGetSquare()`) **do**
  if parent has a motion vector (`estGetParentNbMV() != 0`) **then**
    $initialMotionVector$ =motion vector of parent (`estGetParentMV()`)
  else
    $initialMotionVector = NULL$
  end if
  estimate motion vector for current block (also return the MSE).
  current block will have one motion bector (`estSetNbMV()`).
  set the current motion vector for this block (`estSetMV()`).
  if MSE is small or size of block equals smallest estimable **then**
    do not further divide this block (`estSquareIsOk()`).
  else
    divide this block (`estDivide()`).
  end if
end while

`estBegin(int iWidth, int iHeight)` constructs the tree in an analog manner than algorithm 7 and sets the variables of each node to an initial state. It just does the second part if the tree is already created. Once we have the tree, we go down in it to the first estimable block and update the pointer to the current node, the level and the array containing the number of unexplored child at each node on our path in the tree. `m_pEstCurrentNode` is set to the first estimable block.

`estGetSquare(int &x, int &y, int &size)` just puts the corresponding attribute of the current node to the values passed to the function. Returns `m_bEstIsFinished`.

`estGetParentNbMV()` returns the number of motion vectors stored in the father of the current node.

`estGetParentMV(int pos)` returns the motion vector at position `pos` of the father of the current node.

`estSetNbMV(int nbMV)` tells the current node how many motion vector it has to store.

`estSetMV(MotionVector *mv, int pos)` passes a motion vector to the current node. It is stored at the position passed in parameter.

`estSquareIsOk()` tell to the motion field that the estimator is fine with the current estimation for this node and do not want to further divide. It sets the leaf value boolean to *true* for the current node. The pointer

to the current node is set to the next block to estimate. This means it will go up in the tree to the next node whose children have not all be estimated and go down to the next block that has not yet been estimated and which is estimable. If we explored the whole tree (the estimation is finished) it will put the corresponding boolean to *true*.

`estDivide()` tell to the motion field that we are not fine with the current estimation and thus, want to go more in depth for this region. The leaf node boolean is set to false. The pointer to the current node is set the first child. The level and the array of unexplored nodes is updated.

The second example of this appendix, is related to data retrieval from the tree. It is based on the `atPixel()` function of the local motion field. This function returns the motion vector of the leaf node responsible for a given pixel. The call on the local motion field is relayed to the root node. When a node receives this call, it asks the right child for the result. Algorithm 21 shows the principles of the function `atPixel()` for the nodes.

Algorithm 21 Finding the motion vector of a pixel.

if my leaf attribute is *true* **then**
  return my motion vector at position 0
else
  Knowing how many child I have and my shape, I deduce which child is responsible for the wanted pixel.
  return $atPixel$(found child)
end if