

2006

LRET : local reference with early termination for H.264 motion estimation

Sweta Singh
San Jose State University

Follow this and additional works at: https://scholarworks.sjsu.edu/etd_theses

Recommended Citation

Singh, Sweta, "LRET : local reference with early termination for H.264 motion estimation" (2006). *Master's Theses*. 3031.
DOI: <https://doi.org/10.31979/etd.bka8-aa6m>
https://scholarworks.sjsu.edu/etd_theses/3031

This Thesis is brought to you for free and open access by the Master's Theses and Graduate Research at SJSU ScholarWorks. It has been accepted for inclusion in Master's Theses by an authorized administrator of SJSU ScholarWorks. For more information, please contact scholarworks@sjsu.edu.

LRET:
LOCAL REFERENCE WITH EARLY TERMINATION
FOR H.264 MOTION ESTIMATION

A Thesis
Presented to
The Faculty of the Department of Computer Engineering
San Jose State University

In Partial Fulfillment
of the Requirements for the Degree
Master of Science

By
Sweta Singh

December 2006

UMI Number: 1441125

INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

UMI[®]

UMI Microform 1441125

Copyright 2007 by ProQuest Information and Learning Company.

All rights reserved. This microform edition is protected against unauthorized copying under Title 17, United States Code.

ProQuest Information and Learning Company
300 North Zeeb Road
P.O. Box 1346
Ann Arbor, MI 48106-1346

©2006

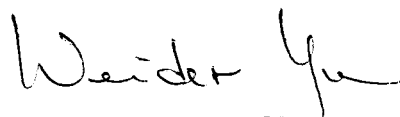
Sweta Singh

ALL RIGHTS RESERVED

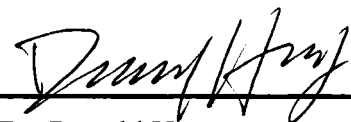
APPROVED FOR THE DEPARTMENT OF COMPUTER ENGINEERING



Dr. Xiao Su



Dr. Weider Yu



Dr. Donald Hung

APPROVED FOR THE UNIVERSITY



ABSTRACT

LRET: LOCAL REFERENCE WITH EARLY TERMINATION FOR H.264 MOTION ESTIMATION

By Sweta Singh

While the support for multiple reference frames and variable block sizes enables H.264 – the latest digital video compression standard – to achieve high data compression, it adversely affects the encoder complexity and motion estimation time. This paper proposes and describes a new algorithm, LRET – Local Reference with Early Termination – that greatly reduces the H.264 motion estimation time. This paper also introduces a novel tool, RFP - Reference Frame Plotter - that graphically shows the reference frames selected by H.264 encoder. Further, the paper presents an analysis of the performance of LRET in terms of motion estimation time, video quality, and compression efficiency. The results show that as compared to H.264 reference software, LRET achieves up to 59% reduction in motion estimation time with negligible effect on the video quality and compression efficiency.

Acknowledgements

It is difficult to overstate my gratitude to my thesis advisor, Dr. Xiao Su. Throughout my thesis-writing period, she provided encouragement, sound advice, and lots of good ideas. This thesis would not have been possible without her persistent support. I cannot thank her enough.

I would also like to thank members of my thesis panel - Dr. Weider Yu and Dr. Donald Hung - for giving me their time and invaluable suggestions.

I am also thankful to my husband, Khem, and friends Rakesh and Anjali for helping me at various stages of my research.

Table of Contents

| | |
|--|----|
| Abbreviations..... | ix |
| 1 Introduction..... | 1 |
| 2 H.264 encoder overview | 4 |
| 2.1 Motion estimation in H.264 | 6 |
| 2.2 Motivation behind research work | 7 |
| 3 Local reference with early termination (LRET) algorithm..... | 11 |
| 4 Reference frame plotter (RFP)..... | 16 |
| 4.1 RFP data structure..... | 17 |
| 4.2 RFP algorithm..... | 17 |
| 5 Experimental results..... | 19 |
| 5.1 Effect of LRET early termination on bit rate, PSNR, and ME time..... | 20 |
| 5.2 Effect of LRET locality search range on bit rate, PSNR, and ME time | 24 |
| 5.3 Effect of partition weights on LRET performance | 27 |
| 5.4 Effect of number of reference frames on LRET performance..... | 29 |
| 5.5 Comparison between LRET, H.264 reference software, and MURF in terms of encoder efficiency..... | 32 |
| 5.6 Comparison based on reference frames selection..... | 35 |
| 6 Conclusion | 39 |
| 7 Future work..... | 40 |
| References..... | 41 |
| Appendix A: Source code of reference frame plotter | 43 |
| Appendix B: Experimental data for locality search range variation..... | 58 |

List of Figures

| | | |
|-------------|--|----|
| Figure 2.1 | Basic coding structure of H.264 for a macroblock..... | 4 |
| Figure 2.2 | Multi-frame prediction | 6 |
| Figure 2.3 | Macroblock partitions..... | 7 |
| Figure 2.4 | Reference frame distribution for frame number 22 of “carphone.qcif” | 9 |
| Figure 3.1 | LRET search range of three..... | 12 |
| Figure 4.1 | RFP data structure | 17 |
| Figure 5.1 | LRET performance - early termination threshold vs ME time..... | 20 |
| Figure 5.2 | LRET performance - early termination threshold vs PSNR..... | 21 |
| Figure 5.3 | LRET performance - early termination threshold vs bit rate | 22 |
| Figure 5.4 | LRET performance - early termination threshold vs bit rate (lesser sequences)..... | 23 |
| Figure 5.5 | LRET performance - ME time vs locality search range..... | 25 |
| Figure 5.6 | LRET performance - PSNR vs locality search range | 26 |
| Figure 5.7 | LRET performance - bit rate vs locality search range..... | 26 |
| Figure 5.8 | Comparison of LRET ME time with and without partition weights | 27 |
| Figure 5.9 | Comparison of LRET PSNR with and without partition weights | 28 |
| Figure 5.10 | Comparison of LRET bit rate with and without partition weights..... | 29 |
| Figure 5.11 | Effect of number of reference frames on LRET ME time..... | 30 |
| Figure 5.12 | Effect of number of reference frames on LRET PSNR..... | 31 |
| Figure 5.13 | Effect of number of reference frames on LRET bit rate | 31 |
| Figure 5.14 | Reference frames selected by H.264 reference software..... | 36 |
| Figure 5.15 | Reference frames selected by LRET | 37 |
| Figure 5.16 | Reference frames selected by LRET with increasing early termination threshold..... | 38 |

List of Tables

| | | |
|-----------|--|----|
| Table 2.1 | Comparison between ME time, bit rate, and video quality for varying number of reference frames | 10 |
| Table 5.1 | Comparison between LRET, MURF, and H.264 reference software based on ME time | 32 |
| Table 5.2 | Comparison between LRET, MURF, and H.264 reference software based on PSNR..... | 33 |
| Table 5.3 | Comparison between LRET, MURF, and H.264 reference software based on bit rate | 34 |

Abbreviations

| | |
|-------|--|
| AVC | Advanced Video Coding |
| CIF | Common Intermediate Format |
| DPB | Decoded Picture Buffer |
| IEC | International Electrotechnical communication |
| ISO | International Organization for Standardization |
| ITU | International Telecommunication Union |
| ITU-T | ITU – Telecommunication Standardization Sector |
| LRET | Local Reference with Early Termination |
| ME | Motion Estimation |
| MPEG | Moving Picture Experts Group |
| PSNR | Peak Signal-to-Noise Ratio |
| QCIF | Quarter Common Intermediate Format |
| RFP | Reference Frame Plotter |
| SIF | Standard Image Format |

1 Introduction

Over the last decade, digital video has become an integral part of visual communication. It is employed widely in applications ranging from television broadcast to video for mobile devices. These applications required that the digital video be optimized to consume less bandwidth, storage, and computing power. Furthermore, the introduction of applications such as video streaming required that this optimization be done in a small time. To meet these requirements, digital video compression was introduced.

H.264, MPEG-4 Part 10, or AVC, for Advanced Video Coding, is the latest digital video compression standard (Wiegand, Sullivan, & Luthra, 2003). It is the result of the collaboration between the ISO/IEC Moving Picture Experts Group and the ITU-T Video Coding Experts Group. The goals of this standardization effort were to provide network friendly video representation and achieve enhanced compression efficiency in terms of better video quality and lower bit rates (“H.264/MPEG-4 AVC,” n.d.). H.264/AVC provides gains in compression efficiency of up to 50% over a wide range of bit rates and video resolutions compared to previous standards such as, MPEG-2 and H.263. However, this improved efficiency and flexibility came at the cost of increased computational complexity in the codec design.

The H.264 standard does not mandate any specific implementation for an Encoder/Decoder pair. It only defines the syntax for the encoded video bit-stream and the method for decoding this bit-stream. Thus, many researches are being carried on each of these functional elements in an attempt to achieve better performance in terms of faster

encoding speed and better video quality with reduced computational complexity.

Some work has been done on efficient multi-frame selection (Chang, Au, & Yeung, 2003; Chen, Chang, Li, & Chi, 2004; Huang, Hsieh, Wang, Chient, Ma, Shen, & Chen, 2003) while other works are based on the efficient exploitation of partition size (Jiang, Li, & Goto, 2004; Yu, 2004; Zhou, Sun, & Hsu 2004). Some research has also been done on efficient search and mode prediction (Li, Chen, Li, & Hsu, 2005). This thesis aims to optimize encoding by exploiting all these areas, i.e. efficient frame selection with consideration to partition size.

The focus of this research is on the process of Motion Estimation (ME). ME in H.264 is far more complicated than previous standards, since it has included support for multiple reference frame, variable block sizes, and quarter pixel precision. The increased complexity in H.264 motion estimation process necessitates a fast search algorithm to improve the encoding performance. The fast algorithm could target different areas of motion estimation e.g. fast mode selection, fast reference frame selection, or fast spatial search-point reduction. Much of the existing work aims at optimizing search pattern in 2D space and finding a good initial motion vector predictor that can be used to skip reference frames. This thesis proposes an algorithm, LRET – Local Reference with Early Termination – that uses the local information to optimize search pattern in the 3D space.

The rest of this paper is organized as follows. Chapter 2 gives a brief overview of H.264 encoder with emphasis on motion estimation. It also mentions the motivation behind this research. Chapter 3 describes the proposed algorithm LRET. Chapter 4 presents the results and analyses of LRET's performance in terms of ME time, bit rate,

and video quality. Chapter 5 presents uniqueness and academic contribution of this research. Chapter 6 presents the conclusion. Chapter 7 addresses the future research that can be performed to further this research work.

2 H.264 encoder overview

H.264 encoding consists of a hybrid of temporal and spatial prediction, in conjunction with transform and entropy encoding. The basic process of encoding consists of the following functional elements: Intra Prediction, Motion Estimation, Transformation, Quantization, and Entropy Coding. Figure 2.1 shows a block diagram of the video encoding structure of H.264.

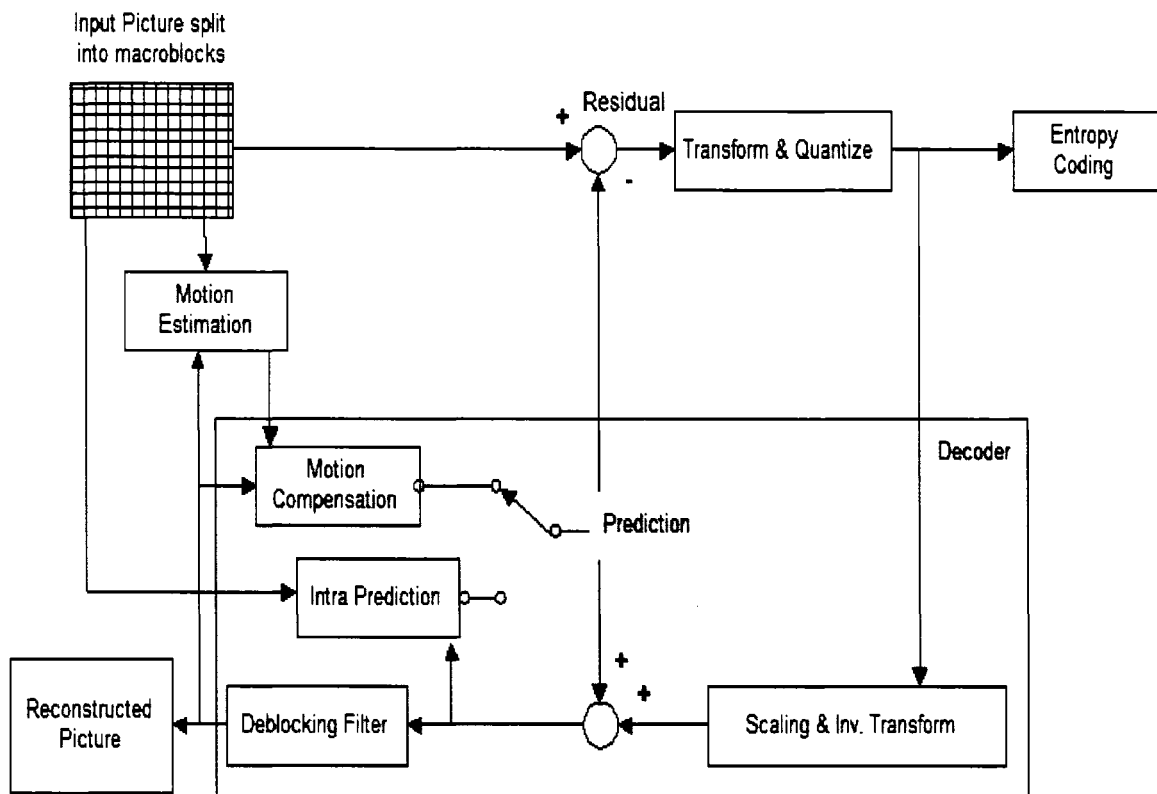


Figure 2.1: Basic coding structure of H.264 for a macroblock

Sullivan and Wiegand explain that in order to encode a video sequence, each picture that comprises the video sequence is encoded individually. To start with, the picture, also called a frame, is split into smaller processing units called Macroblocks.

Each macroblock is 16x16 pixels in dimension.

The first picture of a sequence is typically “Intra” coded, i.e., by only using information contained in the picture itself. Each block in an intra frame is predicted using information from previously-encoded spatially-neighboring blocks.

The rest of the pictures in the sequence are typically “Inter” coded. Inter coding involves prediction from previously decoded pictures (“H.264/MPEG-4 Part 10 Tutorials,” n.d.). The coding comprises of motion estimation and motion compensation. Motion estimation is the process of finding motion vectors, whereas, motion compensation is the application of motion vectors to already-decoded frames. Motion vectors (MV) describe the difference between consecutive frames in terms of where each section of the former frame has moved to. The accuracy of MV is a quarter of a sample distance. If the MV points to an integer-sample position, the prediction block is the corresponding block of the reference picture; otherwise, it is obtained by interpolation at the sub-sample positions.

The residual of the prediction – which is the difference between the original and the predicted block – is transformed. The transform coefficients are scaled, quantized, and entropy encoded. These coefficients, along with the motion vectors, are sent to network abstraction layer for transmission.

The encoder also contains the decoder. For decoding, the quantized transform coefficients are inverse scaled, inverse transformed, and then added to the prediction. The resulting decoded picture is stored in the Decoded Picture Buffer (DPB). DPB is maintained by both receiver and sender, so that the same picture can be used for prediction by both. Sender cannot use the original picture as reference because

compression reduces the video quality, i.e., the decoded picture is not the same as the original picture.

2.1 Motion estimation in H.264

Motion estimation in H.264 is more complex than that of earlier standards. The complexity is introduced due to the support for multiple reference frames and variable block sizes.

Figure 2.2 illustrates the multi-frame motion prediction process. It shows that for predicting a macroblock, more than one previously-encoded frame is used as reference. For each block of current frame, a search for a matching frame is made in all the reference frames. The best match is selected for the predicted frame. For example, in Figure 2.2, the best match for “checkered” block is found in reference frame that is just before the current frame in temporal domain. Similarly, for the “horizontal striped” block, best match is found in reference frame number four (reference frame are numbered in chronologically reverse order) and for “vertical striped” block in reference frame number two.

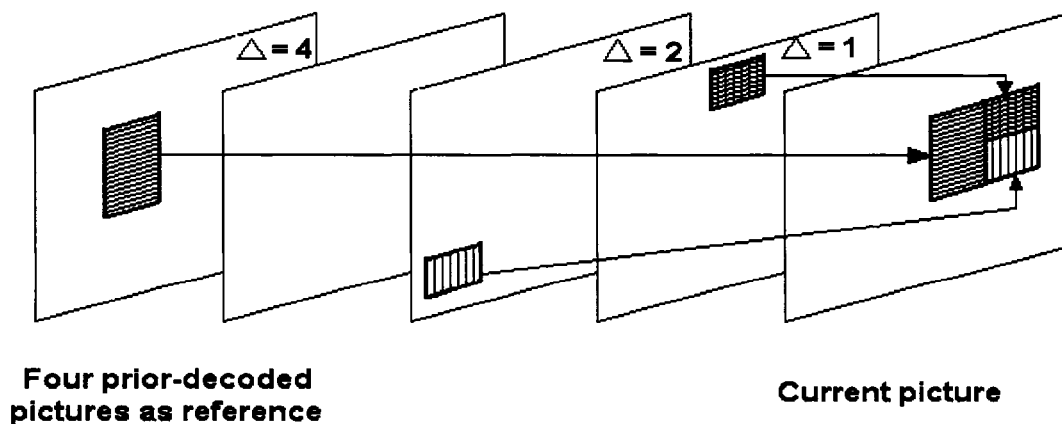


Figure 2.2: Multi-frame prediction

Furthermore, H.264 also supports multiple block sizes. Each 16x16 macroblock can be partitioned into block sizes of 16x16, 16x8, 8x16, or 8x8 pixels. An 8x8 block can be further sub-partitioned into sizes of 8x8, 8x4, 4x8, and 4x4. Figure 2.3 illustrates the partitioning.

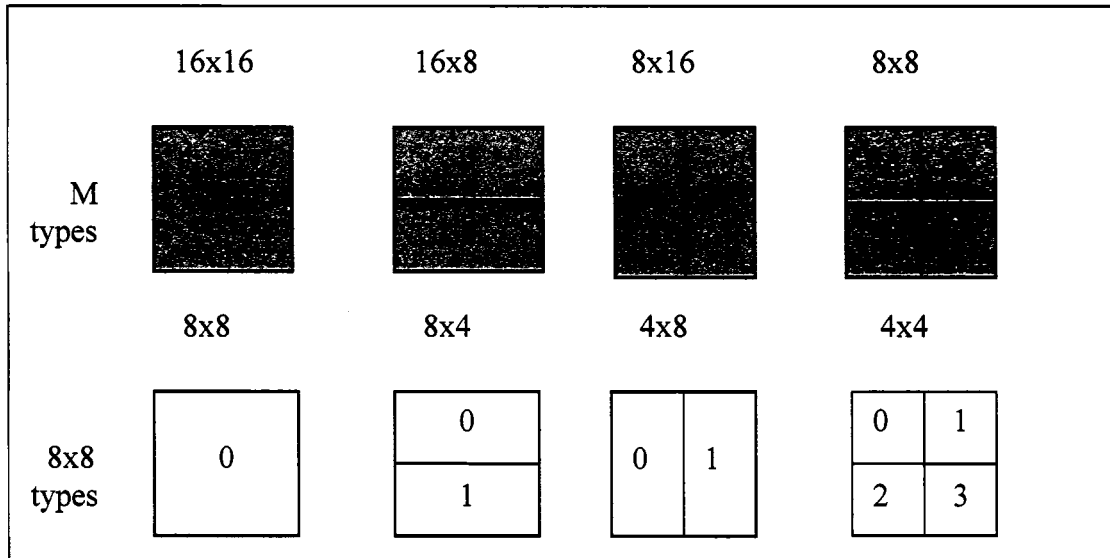


Figure 2.3: Macroblock partitions

Motion estimation is done for each of the MxN blocks. This involves searching for a match for the current block (and all its partitions) in all the reference frames and generating a motion vector corresponding to that. Chances of finding a closer match increases as the block size decreases. While, a closer match leads to a smaller residual, smaller block sizes require higher number of motion vectors to be encoded. Cost of the match is calculated as the sum of bits needed to encode the corresponding motion vector and the residual. The match with the least cost is selected as the best match.

2.2 Motivation behind research work

For motion estimation, H.264 employs various algorithms like Simplified

Hexagonal (SHEX) search and Fast Full search. These algorithms work in the spatial domain. But H.264 reference software does not employ any algorithm for motion estimation in the temporal domain (The reference software simply selects the reference frames in chronologically reverse order). Thus, an algorithm to optimize motion estimation in the temporal domain may be proposed.

Each picture in the temporal domain of a video sequence has a gradual change from its previous one. This change constitutes the video motion. Thus, if a match for a block is found in reference frame number x , it is likely that a match will also be found in reference frame number $x-1$. If more reference frames are searched, a better match may be found. But such optimization comes at the expense of high motion estimation time. For a 16×16 macroblock, there are 45 different partitions for motion search. In each reference frame, candidates are searched within a search window size of $W \times W$ (W is set to 33 by default). For a full search, the number of search points is given by the product of number of partitions, number of reference frames, and search window. For example, with five reference frames in DPB, a full search involves 245025 ($=45 * 5 * 33 * 33$) search points for a single macroblock. This number is prohibitively high for the encoder. Instead, if the search for matching block is terminated after finding a good match (and not carried on to find the best match), then motion estimation time can be saved with little or no effect on the encoded bit rate.

Another property of video motion is that between each consecutive frame, a group of macroblocks might move together. For example, if the motion involves a moving hand, all macroblocks that constitute the hand, will move together. This makes it highly likely that the neighboring blocks of a picture will have the same reference frame. This

feature of video motion can be exploited to prioritize the search order of reference frames.

Experiments were run to verify if the above-mentioned hypotheses held true against the actual implementation of the reference software. Figure 2.4 shows the reference frame selected for each block of a frame using exhaustive search across all reference frames. It can be seen that there is a great correlation between the reference frames of neighboring blocks.

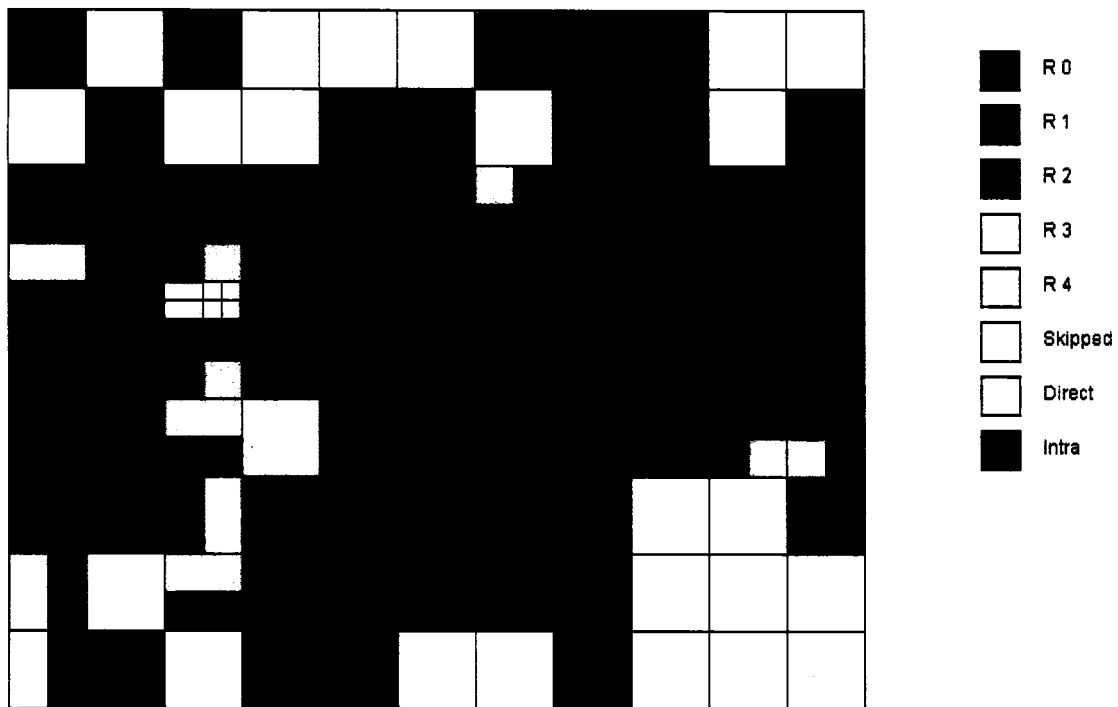


Figure 2.4: Reference frame distribution for frame number 22 of “carphone.qcif”

Table 2.1 tabulates motion estimation time and video quality for varying number of reference frames. ME time has almost ten-fold increase with only negligible degradation in the video quality, or compression (Low bit rates and high PSNR values

signify a better match). This data confirms that if motion estimation is terminated early, substantial gain in ME time can be achieved.

Table 2.1: Comparison between ME time, bit rate, and video quality for varying number of reference frames

(Based on reference software encoding of “carphone.qcif” sequence)

| Number of reference frames searched | Motion Estimation time(sec) | Video Quality (PSNR in db) | Bit Rate (@ 30Hz) |
|-------------------------------------|-----------------------------|----------------------------|-------------------|
| 2 | 156.786 | 36.37 | 135.62 |
| 3 | 217.807 | 36.44 | 134.61 |
| 4 | 282.131 | 36.49 | 133.78 |
| 5 | 351.396 | 36.54 | 133.98 |
| 6 | 426.467 | 36.54 | 133.66 |
| 7 | 490.629 | 36.56 | 133.57 |
| 8 | 560.217 | 36.6 | 133.96 |
| 9 | 628.637 | 36.6 | 133.79 |
| 10 | 700.139 | 36.62 | 134.09 |
| 11 | 768.322 | 36.62 | 133.62 |
| 12 | 830.87 | 36.63 | 133.94 |
| 13 | 899.272 | 36.63 | 134.12 |
| 14 | 970.156 | 36.62 | 134.06 |
| 15 | 1038.691 | 36.65 | 133.91 |

3 Local reference with early termination (LRET) algorithm

Local Reference with Early Termination (LRET) aims at reducing the motion estimation time without affecting the picture quality or compression. The basic concept behind LRET is to reorder the search order of reference frames based on their usage in current locality. Additionally, LRET exploits the redundancy in the temporal domain of a video sequence by terminating the search after a good match is found. Good match is defined as one that takes fewer bits than a threshold.

LRET implementation can be functionally divided into two parts. One part is to store the information about usage of reference frames for each macroblock. This is done after macroblocks are encoded. The other part is to retrieve the reference frame usage information in order to prioritize the search order in temporal domain. This is done when a block is being encoded.

For storing information about reference frames, LRET adds an array of size 16 – since H.264 supports a maximum of 16 reference frames – to each macroblock structure. This array stores information about the reference frames used for encoding each of the macroblock partitions. When a block is encoded, the array element at index corresponding to the selected reference frame is incremented.

When storing information, LRET also allows for partition weights. Since H.264 supports variable block sizes, the reference frame increment can be done in two ways. Either the counters can be incremented by one for each match or the increment can be made proportional to the size of the block that is matched. That is, if a bigger block is matched then the corresponding reference frame may be given more weight. LRET

calculates weight as the sum of height and breadth of the partition. Section 4.3 discusses the effects of partition weights on LRET encoding performance

For retrieving information about the usage of reference frames in the current locality, LRET takes as input the neighborhood range. This range specifies the locality within which the reference frame statistics needs to be collected. Figure 3.1 shows the macroblocks that are considered for statistical information collection, when a search range of three is input. “X” symbolizes the macroblock that is being encoded. Reference frames selected by each of the grayed macroblocks are added together.

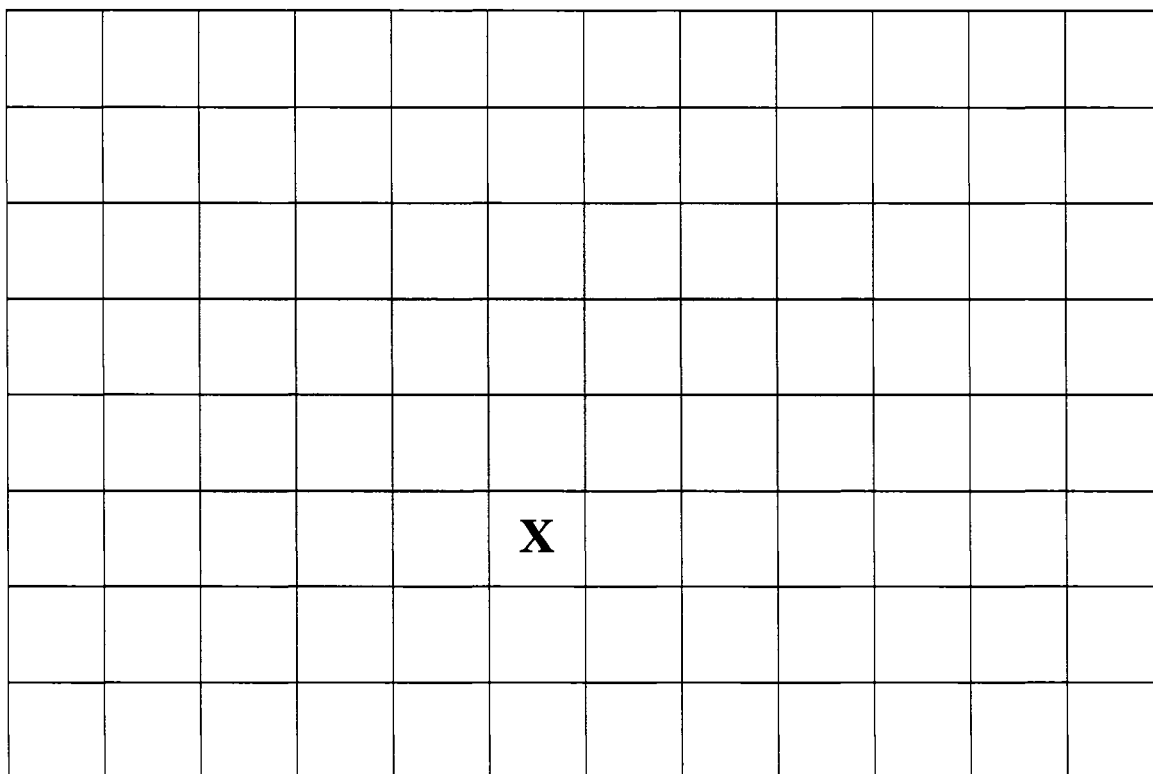


Figure 3.1: LRET search range of three

The reference frames are selected in decreasing order of their usage in the current locality. For each block/reference block pair, encoding cost is compared to a threshold.

If the threshold condition is satisfied, the search is terminated. The LRET algorithm is described below.

1. For each macroblock that is encoded in the current frame
 - a. Declare an array named *macroblock_array* of size 16.
 - b. Initialize all elements of *macroblock_array* to zero.
2. While encoding each block
 - a. Create a search window based on the input search range.
 - b. Create an array named *temp_array* by adding *macroblock_array* of all macroblocks that fall within the search window.
 - c. Set $min_motion_cost = INT_MAX$.
 - d. For $I = 1$ to N , where N is the number of reference frames that the encoder is using,
 - i. Select the index – *ref_index* – that corresponds to the highest value in array *temp_array*.
 - ii. Set the corresponding element in *temp_array* to “-1”.
 - iii. Find the block that is the best match in the reference frame number *ref_index* by calculating its motion cost C .
 - iv. If $C < min_motion_cost$, then
 1. Set $best_ref = ref_index$.
 2. Set $min_motion_cost = C$.
 3. If $min_motion_cost < threshold$ then break loop.
 - e. End-For.
3. After the block is encoded,

- a. Set *increment_factor* to the sum of height and width of the block.
- b. Add *increment_factor* to the *macroblock_array* element that is indexed by *ref_index*.

The optimizations proposed by LRET do not add to the complexity of H.264 encoder. The analysis of code complexity of LRET can be done on per macroblock basis with “n” representing the number of reference frames used for motion estimation.

Initialization of arrays within each macroblock structure

$$= O(n).$$

For a search range of x , total macroblocks that are considered for locality information

$$= 4 * (x + (x-1) + (x-2) + \dots + 1)$$

$$= 4 * (x * (x+1) / 2)$$

$$= 2 * (x^2 + x)$$

$$= O(x^2)$$

Collection of local statistics of reference frame usage

$$= \text{Number of macroblocks in locality} * \text{Number of reference frames}$$

$$= (2 * (x^2 + x)) * n$$

$$= O(nx^2)$$

Updating reference frame statistics

$$= O(1)$$

Overall complexity of LRET

$$= \text{Initialization of arrays}$$

$$+ \text{Collection of local statistics of reference frames}$$

$$+ \text{Updating reference frame statistics}$$

$$= O(n) + O(nx^2) + O(1)$$

$$= O(nx^2)$$

Though the code complexity of LRET is $O(nx^2)$, it is negligible when compared to that of H.264. H.264 motion estimation process involves calculation of the sum of absolute difference of M pixels, where M ranges from 16 (for 4x4 sub-partition) to 256 (for 16x16 macroblock) pixels for each block, and comparisons of motion cost values to $(N * W * W * \text{\#partitions})$ search points (See Section 2.2). Compared to this, the computational overhead introduced by LRET is negligible.

4 Reference frame plotter (RFP)

Reference Frame Plotter (RFP) is a tool that plots the selected reference frames for each block of the encoded picture. RFP is coded in JAVA and needs JAVA run time environment for execution.

H.264 encoder has an option to enable generation of trace file. RFP uses this trace file to collect information regarding the reference frames used by each block. It uses this information to graphically depict the reference frame selection for a given frame. Additionally, RFP allows for the generated image to be saved as a jpeg file.

RFP starts by reading the input file and collecting information related to the encoded macroblock, e.g., the partition size, reference frame used, and if it is further divided into smaller blocks. Then it stores this information into a data structure. Section 4.1 discusses the data structure used by RFP. Once RFP has processed all the macroblocks, it sends the data structure to the image-drawing module which draws square-shaped images. The image-drawing module then fills these images with colors that are selected based on the reference frame used. When the image window is closed, RFP checks to see if the image was requested to be saved. If so, it saves the image with the specified filename in jpeg format.

Appendix A contains the RFP source code.

4.1 RFP data structure

RFP data structure is a three-dimensional vector. It is pictorized in Figure 4.1.

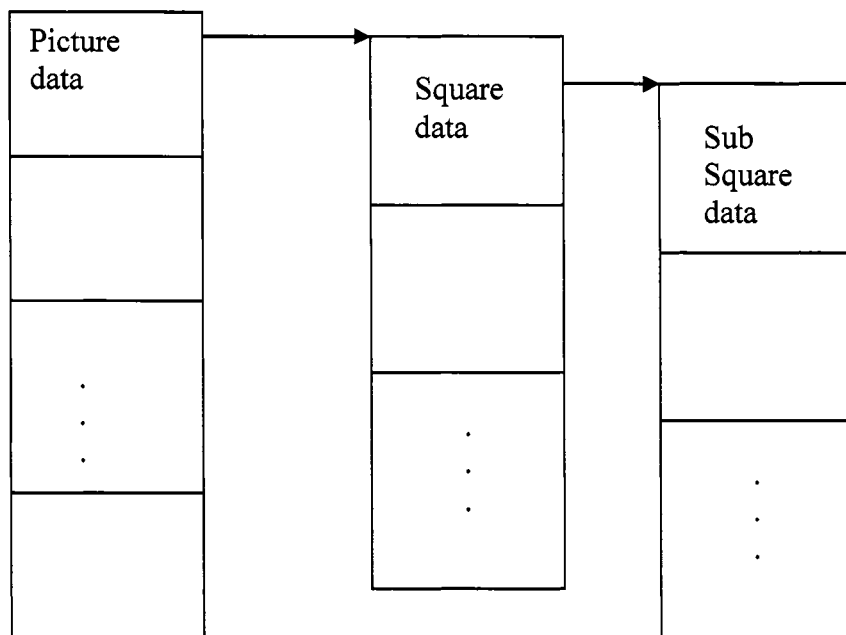


Figure 4.1: RFP data structure

The first vector, called *Picture_data* is a collection of structures that store information about macroblocks. Each *picture_data* element contains a vector named *Square_data* that stores information about each partition of macroblock. Thus, if a macroblock has four 8x8 partitions, then the corresponding *picture_data* element will have four *square_data* elements. *Square_data* can further contain a list named *Sub_square_data* that stores information about sub partitions of 8x8 macroblock partition.

4.2 RFP algorithm

- 1) For each line of encoder trace file

- 2) Read line and pass to tokenizer (delimiter = whitespace).
- 3) If the token = *num_ref_frames*
 - a) Store value in a global variable (this decides how many colors i.e. reference frames, this image is going to use).
- 4) If the token = *pic_width_in_mbs_minus1* or *pic_height_in_map_units_minus1*
 - a) Store values in global variables (this gives the image size).
- 5) For each token = *Pic*
 - a) If picture number = input picture number
 - i) Get the macroblock number and store its offset from image boundaries.
- 6) If token = *mb_type* and picture number = input picture number
 - a) Create and store *square_data* based on *mb_type*.
- 7) If *mb_type* > 8
 - a) Store block as direct block.
- 8) If token = *8x8*
 - a) Create and store *sub_square_data*.
- 9) If token = *ref_idx_l0*
 - a) Update the color values for the *square_data*.
- 10) After all data corresponding to the picture is retrieved, draw the image.
- 11) When image window is closed
 - a) If a filename is provided for saving the file, save image.

5 Experimental results

LRET has been implemented in H.264/AVC reference software JM 11.0 (Tourapis, Suehring, & Sullivan, 2004). LRET's performance was measured on a Linux 2.6.8 machine with Pentium IV 2.4 GHz processor and 512 MB RAM. According to Tourapis, Suehring, and Sullivan, the rate distortion mode should be set to high complexity mode when evaluating algorithmic performance (21; ch.4). Also, Mahajan and Kondayya showed that H.264 gives the best results when Simplified Hexagon (SHEX) search is used (19). Thus, the experiments were run with SHEX and high complexity mode of rate distortion optimization enabled.

The performance of the algorithm was analyzed in terms of compression efficiency (bit rate), motion estimation time, and video quality (PSNR). Experiments were run on nine video sequences. There were three sequences of each of the following types: Common Intermediate Format (CIF), Quarter CIF (QCIF), and Standard Image Format (SIF).

First, experiments were run to study the effect of input parameters on LRET performance. The input parameters that were varied are early termination threshold and LRET search range. Next, experiments to study the effect of partition weights on LRET encoding were performed. This was followed by experiments to analyze performance of LRET for varying number of reference frames. Lastly, experiments were done to compare performance of LRET with that of H.264 reference software and Most Used Reference First (MURF) (Tourapis, Suehring, & Sullivan, 2004; Mahajan & Kondayya, 2006).

5.1 Effect of LRET early termination on bit rate, PSNR, and ME time

Experiments were run with varying early termination threshold. The threshold was varied from zero bits to 1200 bits in steps of 100. Search range was set to span the entire picture. Figure 5.1 through Figure 5.4 depict the effect of early termination on ME time, PSNR values, and bit rate respectively.

Figure 5.1 shows that the ME time reduces with an increase in the early termination threshold. This is expected behavior, since early termination stops further searches for finding a better match.

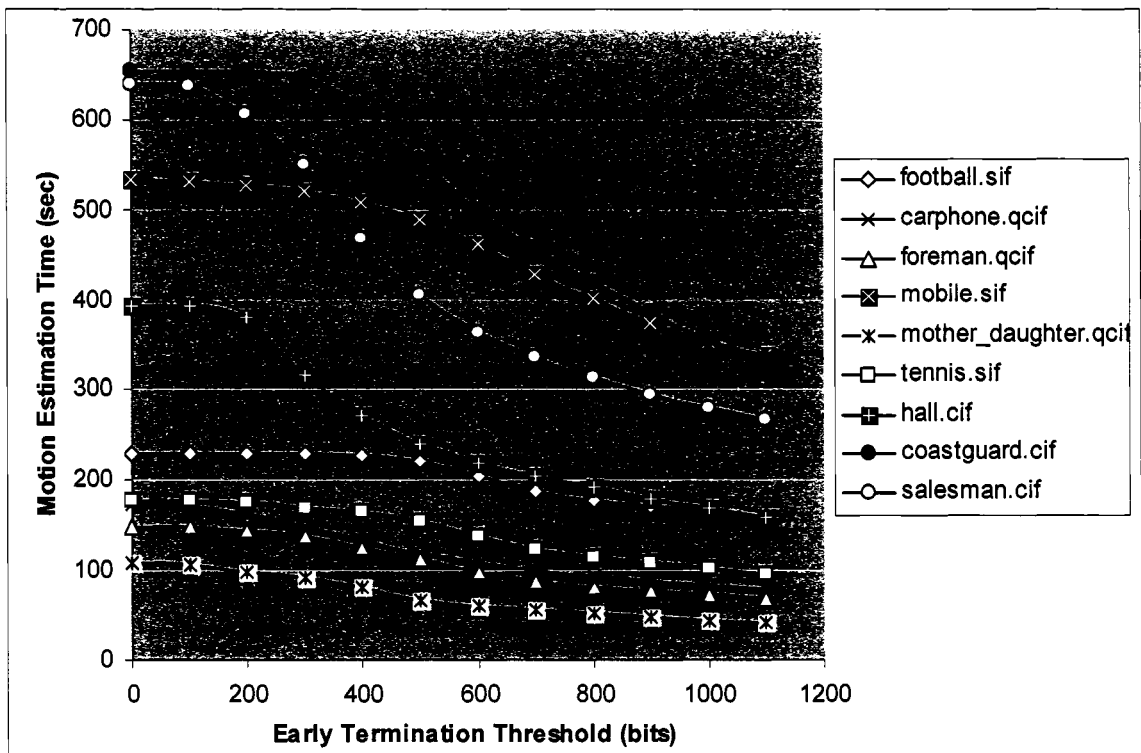


Figure 5.1: LRET performance - early termination threshold vs ME time

Figure 5.2 shows that an increase in the early termination threshold has a very small effect on the PSNR values of the picture. For higher values of early termination

threshold, the PSNR values show a slight decrease. This is because a not-so-good match is found, and the search is terminated. Since worse matches are used for encoding, the video quality decreases.

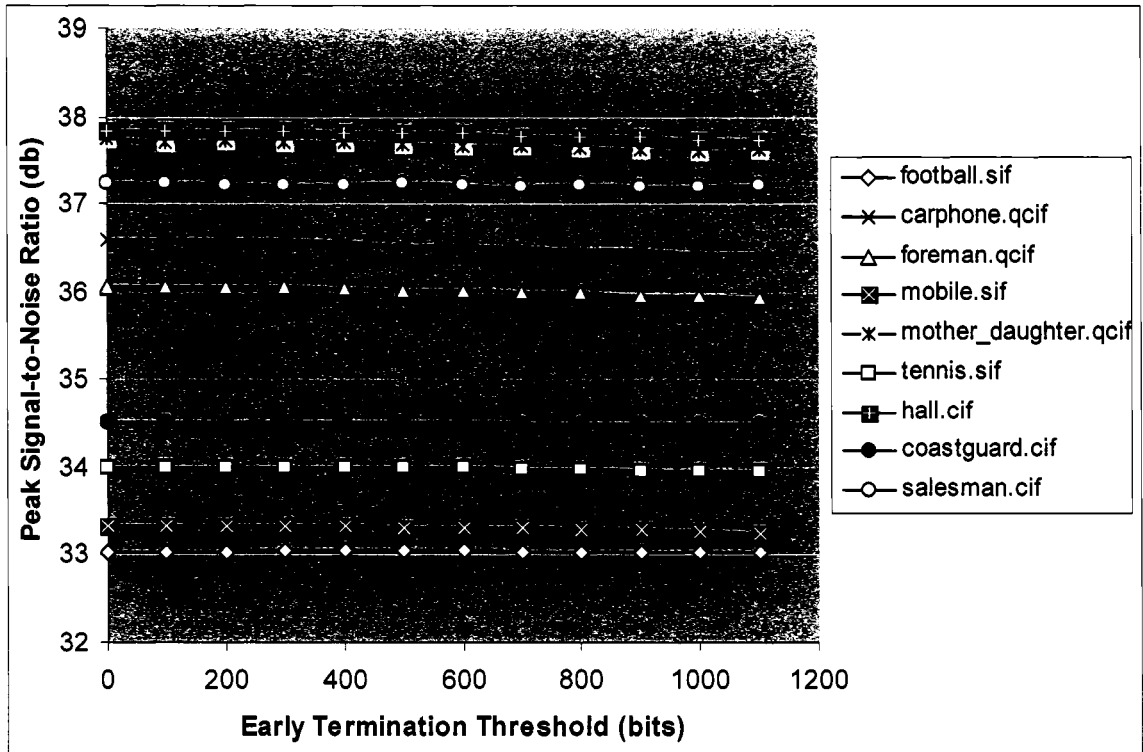


Figure 5.2: LRET performance - early termination threshold vs PSNR

Figure 5.3, shows the bit rates with variable early termination threshold. But, since the bit rates used by SIF and QCIF images vary a lot, not much can be inferred from this figure. Thus, Figure 5.4 is created for a small number of sequences so that the variation can be better observed.

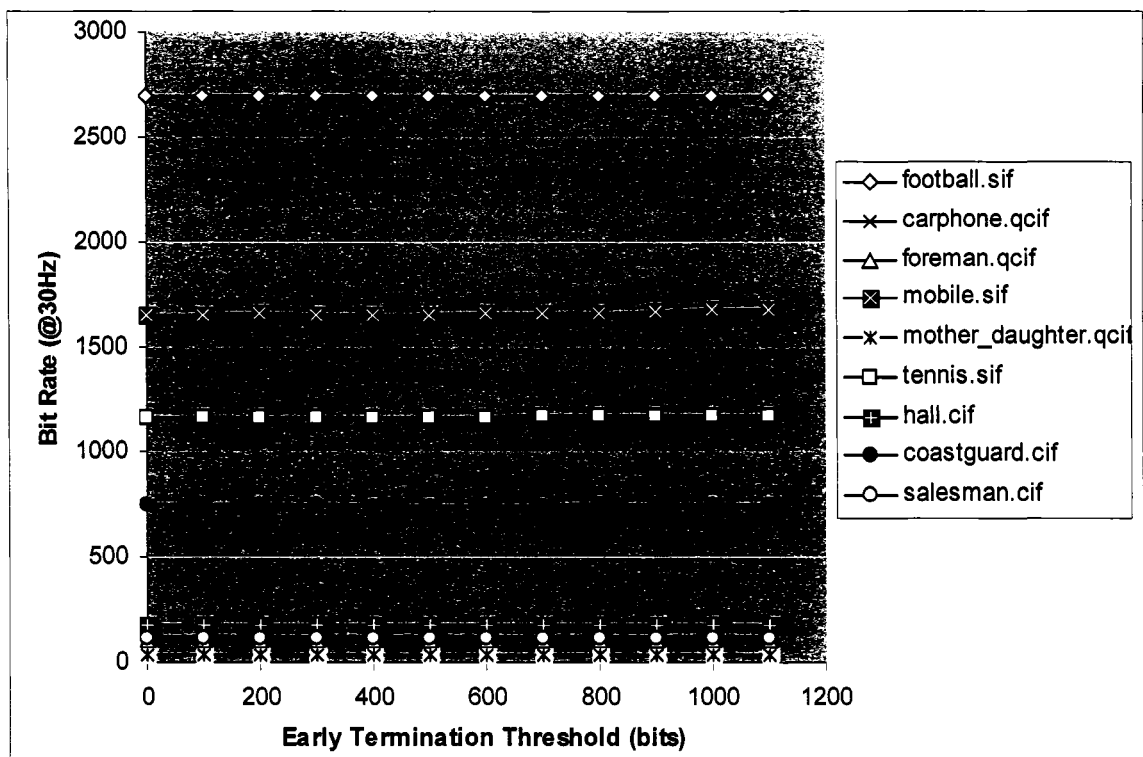


Figure 5.3: LRET performance - early termination threshold vs bit rate

Figure 5.4 shows that the bit rates increase slightly with an increase in the early termination threshold. This too is expected. As the threshold increases, the quality of match selected deteriorates. Worse match results in an increase in “residual” and thereby in an increase in the number of bits required in encoding the match.

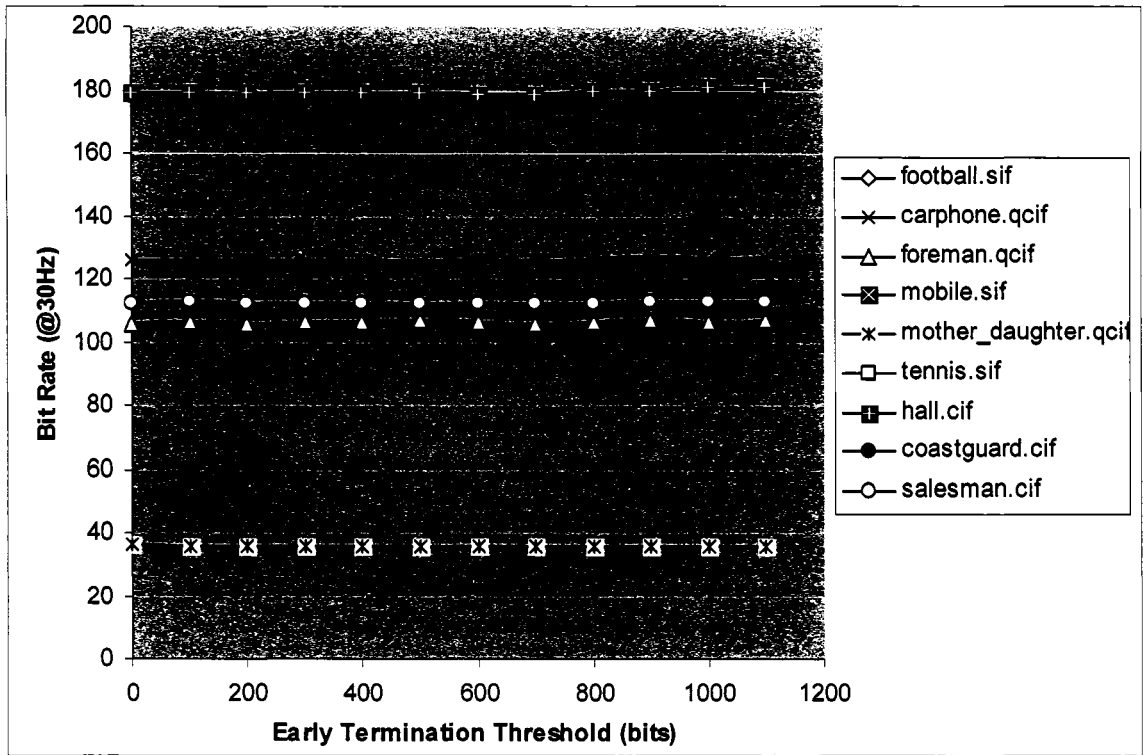


Figure 5.4: LRET performance - early termination threshold vs bit rate (lesser sequences)

Overall performance of LRET with variable early termination threshold shows that increasing the threshold gets major improvements in ME time with only little degradation in compression (bit rate) and video quality (PSNR). For example, in “hall.cif”, ME time reduces by 55% while bit rate increases by 0.006% and PSNR degrades by 0.001%.

Also, LRET performance was seen to vary considerably between early termination threshold values of 200 bits to 600 bits. In terms of percentage gain in ME time for a given degradation in video quality and compression, the best performance is achieved with a threshold of 500.

With a threshold of zero, LRET performance is similar to that of H.264 reference software (Tourapis, Suehring, & Sullivan, 2004). When the early termination threshold is set to zero, the LRET encoder does not terminate the search process early. Instead, it continues to search all reference frames (albeit in a different order than H.264 reference software) for a better match.

5.2 Effect of LRET locality search range on bit rate, PSNR, and ME time

To observe the effects of locality search range on LRET performance, experiments were conducted by varying the search range in steps of one for QCIF, and in steps of two for SIF and CIF formats. Once the search range reaches the maximum dimension of current frame, further increase in search range is immaterial. The early termination threshold was set to 1000 bits. Figure 5.5 through Figure 5.7 illustrate the effect of search range on LRET performance.

Figure 5.5 shows that ME time has initial jumps for increasing search range but almost stabilizes beyond search range of seven. Typically, motion estimation time decreases when a match is found earlier. If all macroblocks within a search range constitute the same moving object, then the correct reference frame will be searched sooner. But if the current block does not come from the same object, then the right reference frame may be picked later. This will increase the motion estimation time.

For a couple of sequences, ME time increases at search range of six. This shows that beyond search range of six, distribution of selected reference frames is such that the right reference frame is not picked early. This leads to an increase in the motion estimation time.

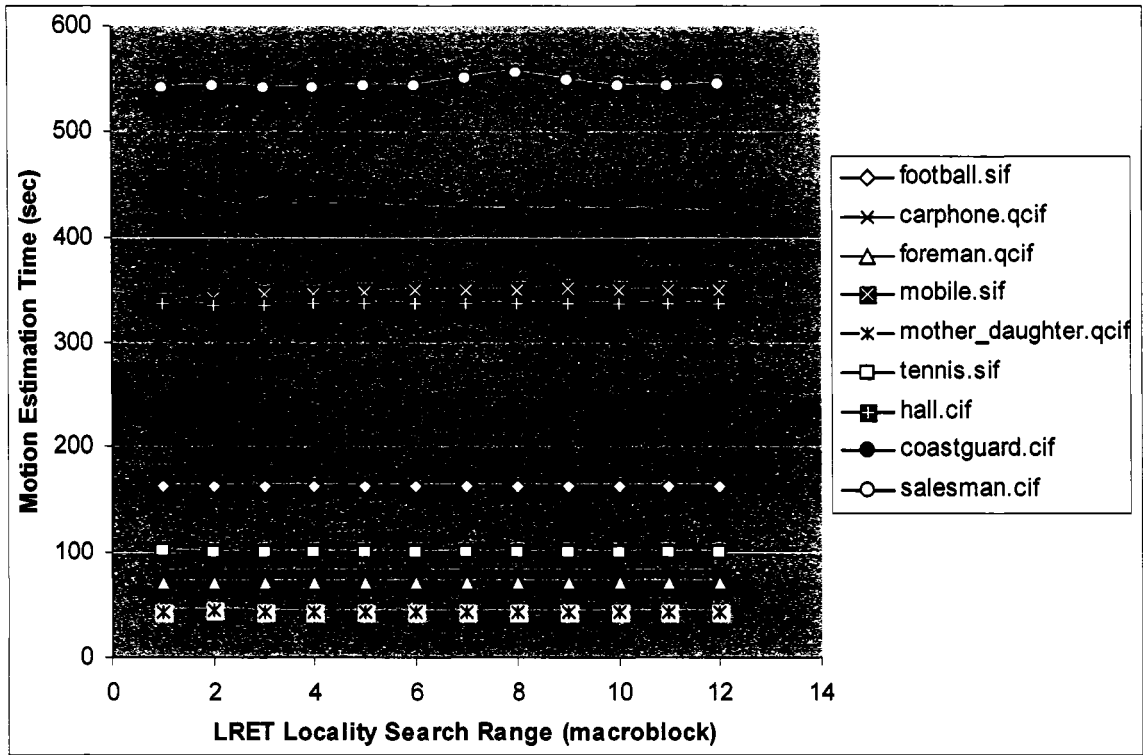


Figure 5.5: LRET performance - ME time vs locality search range

Figures 5.6 and 5.7 show the PSNR and bit rate for increasing search ranges respectively. The variation in values is so low that the charts don't help much in analysis. Instead, the data sets using which these charts were plotted are used for analysis (See Appendix B).

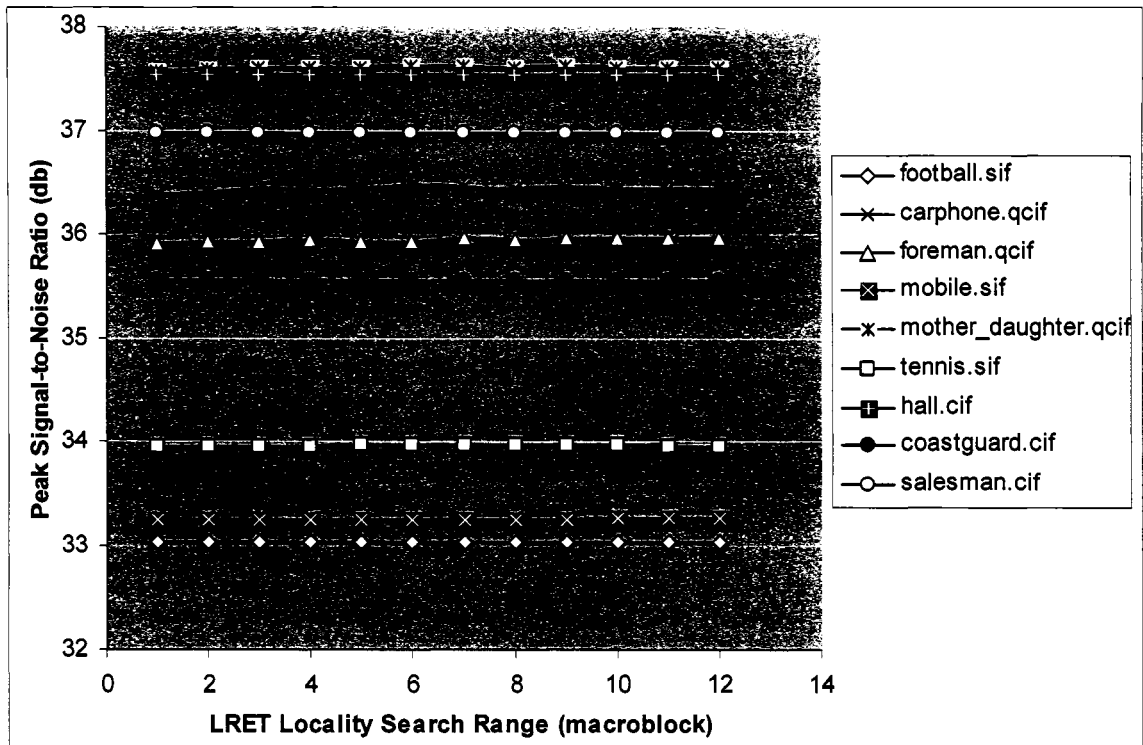


Figure 5.6: LRET performance - PSNR vs locality search range

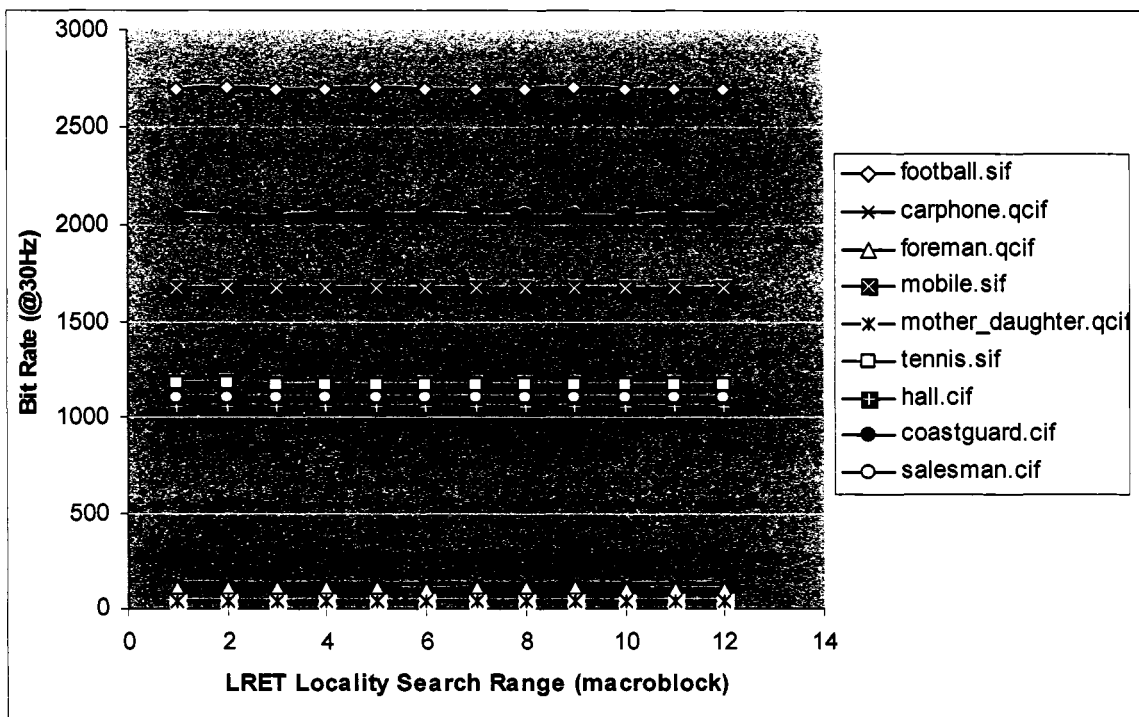


Figure 5.7: LRET performance - bit rate vs locality search range

The data sets illustrate that for most sequences, ME time is approximately the same for varying search ranges, whereas, compression and video quality have minor improvements. The improvements in compression and video quality suggest that bigger search ranges select correct reference frame.

In terms of percentage gain in ME time for a given degradation in video quality and compression, the best performance is achieved for a search range of seven.

5.3 Effect of partition weights on LRET performance

To analyze the effect of partition weights, the “mother-daughter.qcif” video sequence is encoded twice, once with partition weights and once without. Figure 5.8 through Figure 5.10 summarize the results. Samples are collected for increasing search ranges.

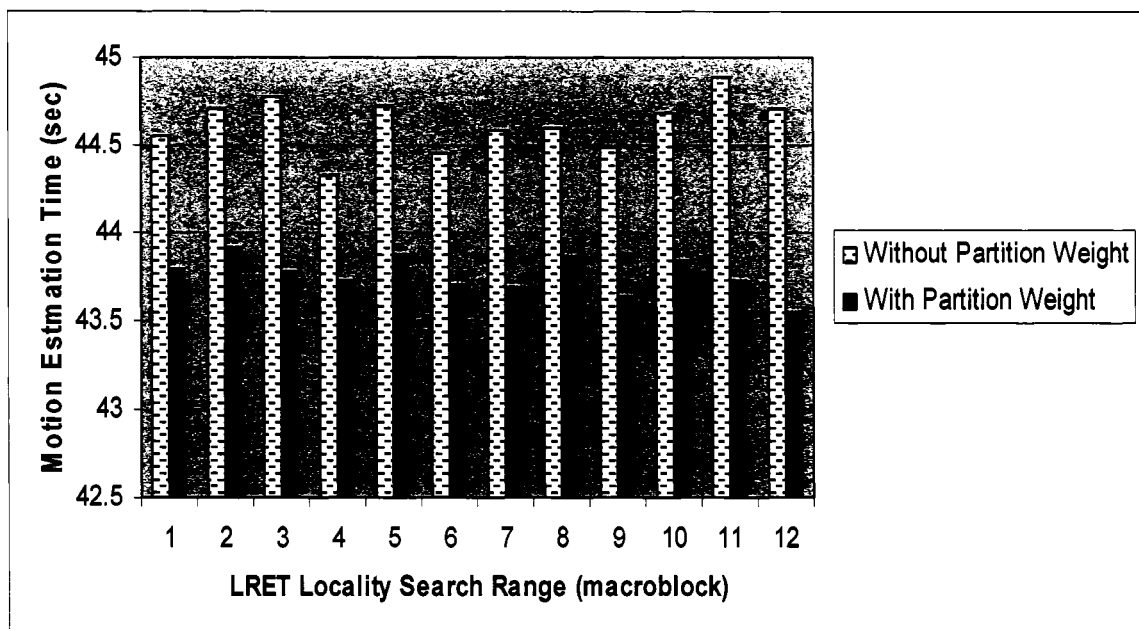


Figure 5.8: Comparison of LRET ME time with and without partition weights

Figure 5.8 shows that the introduction of partition weights reduces the motion estimation time. Figures 5.9 and 5.10 show that the performance of LRET, in terms of bit rate and video quality, does not vary much by the introduction of partition weights. This result suggests that the introduction of partition weights reorders the search in temporal domain such that the correct reference frame is searched first. But since the final selection of match remains unchanged, there is not much difference in the PSNR and bit rate values.

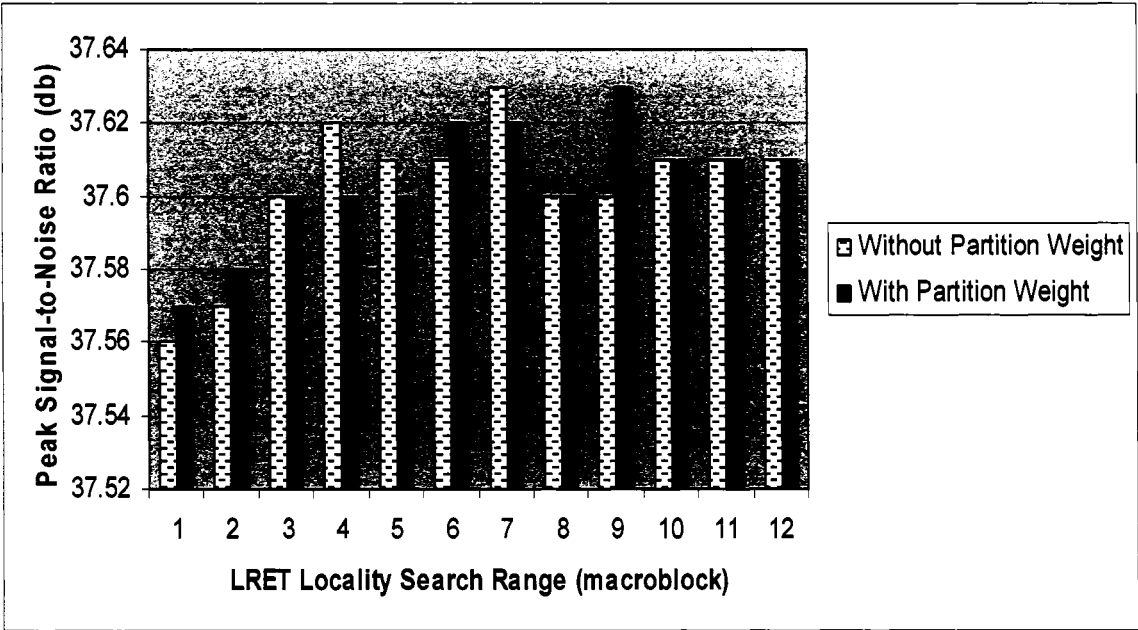


Figure 5.9: Comparison of LRET PSNR with and without partition weights

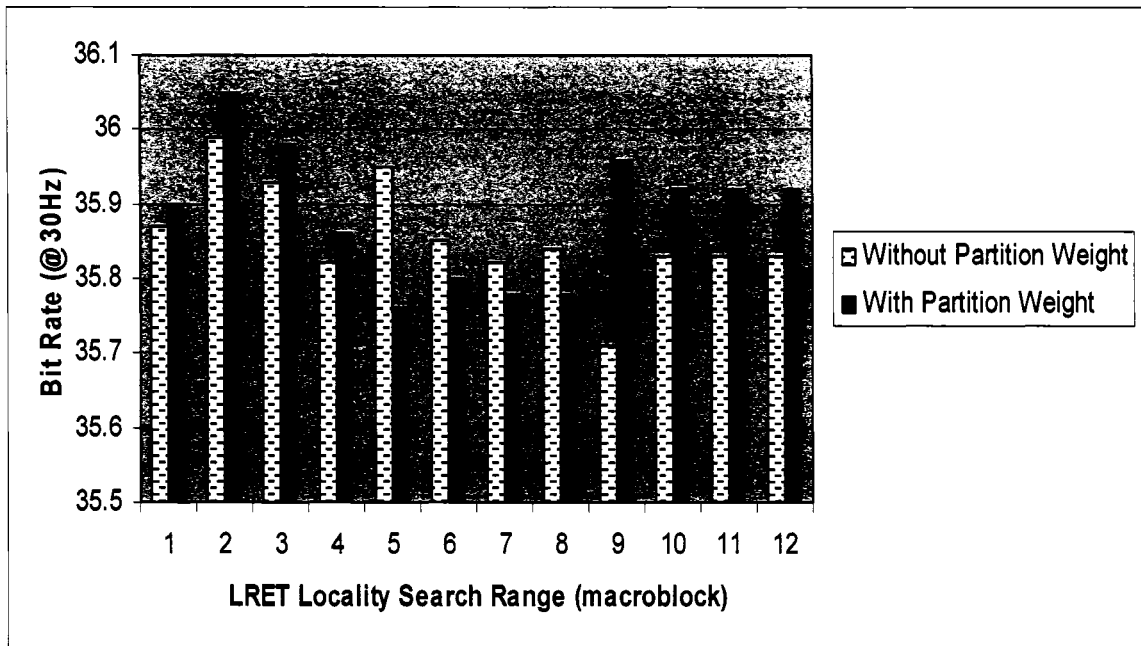


Figure 5.10: Comparison of LRET bit rate with and without partition weights

5.4 Effect of number of reference frames on LRET performance

This section shows the effect of varying number of reference frames on LRET performance. The experiments are run with number of reference frames in DPB being incremented in steps of one, from one to fourteen. The early termination threshold is set to 1000 and the search range is set to span the entire picture. Figure 5.11 through Figure 5.13 summarize the results.

Figure 5.11 shows that as the number of reference frames increases, ME time also increases. This is expected because LRET is $O(nx^2)$, where n is the number of reference frames (See Chapter 3).

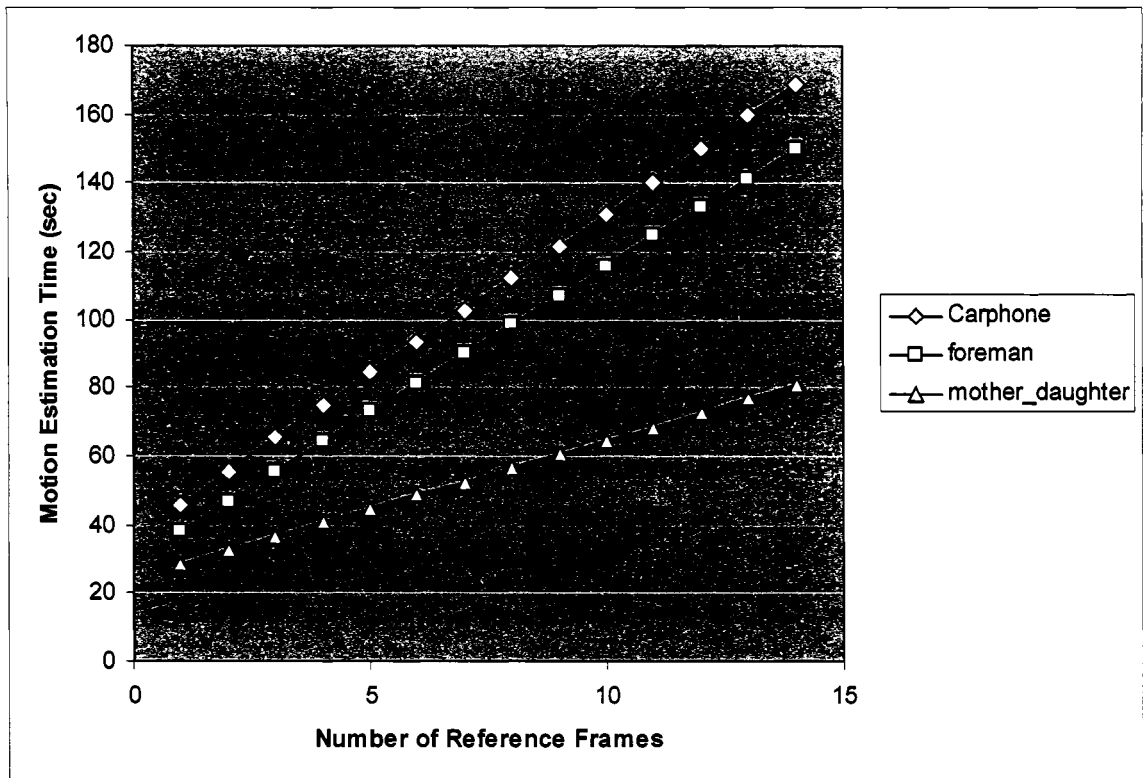


Figure 5.11: Effect of number of reference frames on LRET ME time

Figure 5.12 shows that PSNR values increase with an increase in number of reference frames used. This is expected because if more reference frames are searched then the chances of finding a better match increases. Similarly, Figure 5.13 shows that for higher number of reference frames, lower bit rates are achieved.

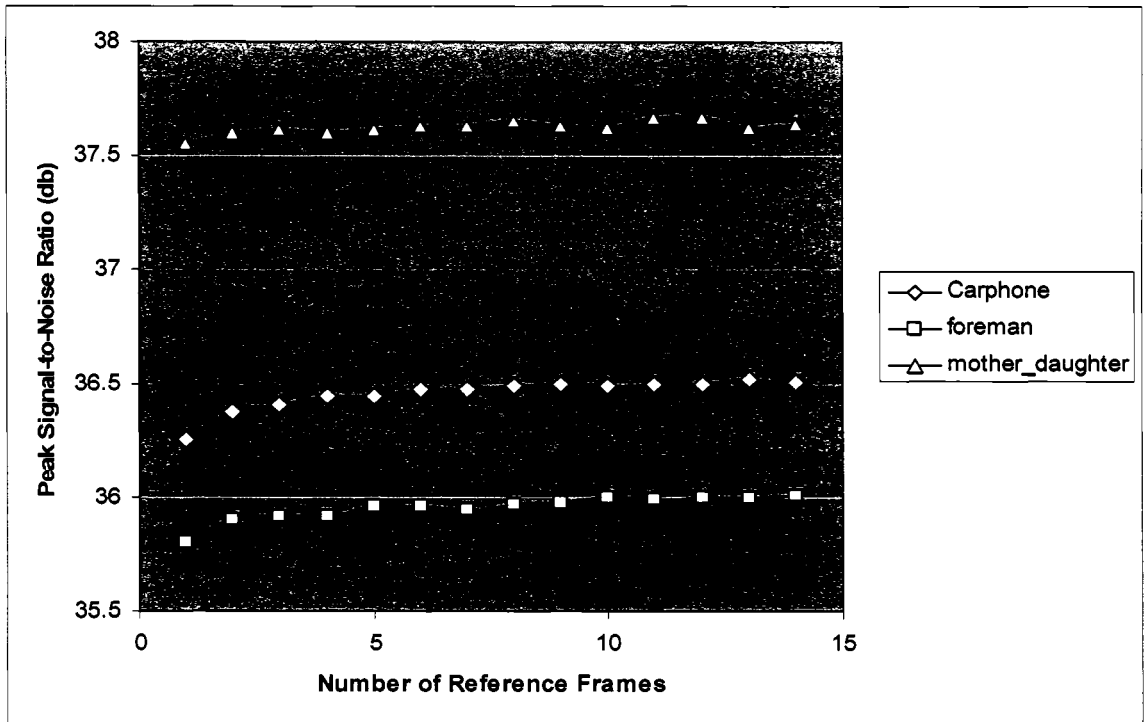


Figure 5.12: Effect of number of reference frames on LRET PSNR

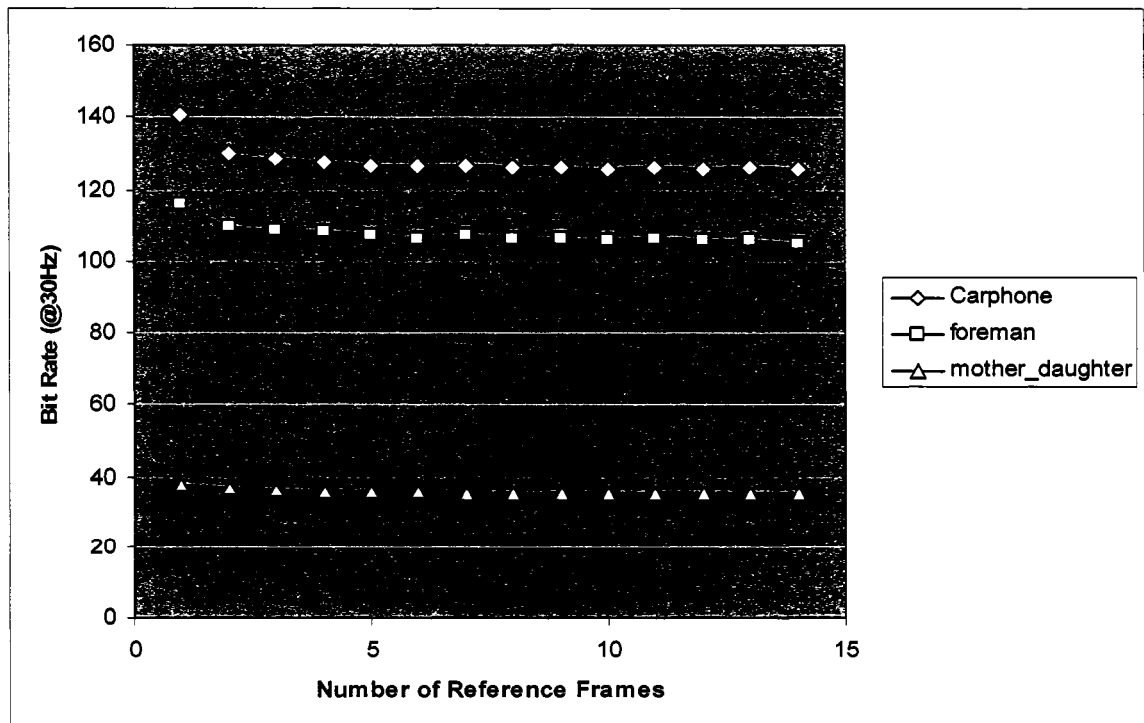


Figure 5.13: Effect of number of reference frames on LRET bit rate

5.5 Comparison between LRET, H.264 reference software, and MURF in terms of encoder efficiency

In this section, LRET is compared to H.264 reference software and MURF (Tourapis, Suehring, & Sullivan, 2004; Mahajan & Kondayya, 2006). The results are summarized in Table 5.1 through Table 5.3.

Table 5.1 shows that compared to the reference software, LRET achieves up to 59% reduction in motion estimation time. But, compared to MURF, the motion estimation times of LRET are slightly higher.

Table 5.1: Comparison between LRET, MURF, and H.264 reference software based on ME time

| | MURF | LRET | H.264 Reference Software | % Gain over Reference Software |
|----------------------|---------|---------|--------------------------------|--------------------------------------|
| Hall.cif | 166.166 | 169.486 | 386.916 | 56.2 |
| Foreman.qcif | 70.444 | 71.09 | 148.81 | 52.23 |
| Carphone.qcif | 82.908 | 83.989 | 171.559 | 51.04 |
| Mother_daughter.qcif | 43.807 | 44.787 | 109.382 | 59.05 |

Gains in ME time as compared to reference software are achieved due to the reduction in number of reference frames searched by LRET. Even when only one reference frame is used, LRET shows gains in ME time. This is due to the “early termination” feature of LRET. While reference software compares all blocks of a frame

to get the lowest cost, LRET only searches till it finds one that costs less than the termination threshold.

As compared to MURF, LRET takes more time for ME. This is because of the difference in the number of operations performed by both the algorithms. LRET, as discussed in Chapter 3, is $O(nx^2)$ while MURF is $O(n \log n)$ (Mahajan & Kondayya, 2006). LRET collects information for each macroblock, whereas, MURF collects information for the entire frame. Thus with an increase in search range, the overhead involved with LRET is considerably higher than MURF.

Table 5.2 shows that compared to the reference software, LRET degrades the video quality of reference software by 0.002% on average. Compared to MURF, LRET is slightly worse in video quality.

Table 5.2: Comparison between LRET, MURF, and H.264 reference software based on PSNR

| | MURF | LRET | H.264 Reference Software | % Decrease in PSNR as compared to H.264 |
|----------------------|-------|-------|--------------------------------|--|
| Hall.cif | 37.75 | 37.74 | 37.84 | 0.002 |
| Foreman.qcif | 35.97 | 35.95 | 36.05 | 0.002 |
| Carphone.qcif | 36.47 | 36.45 | 36.59 | 0.003 |
| Mother_daughter.qcif | 37.64 | 37.61 | 37.75 | 0.003 |

PSNR values of LRET as compared to reference software are as expected. Since LRET terminates search early, the reference block is not the perfect match and thus the video quality deteriorates.

Surprisingly, even MURF gets better video quality than LRET. This suggests that the reference frame order selected by MURF and LRET are different. This difference is introduced by the partition weights in LRET.

Table 5.3 shows that LRET usually utilizes more bits compared to H.264 reference software but gives slightly better performance than MURF.

Table 5.3: Comparison between LRET, MURF, and H.264 reference software based on bit rate

| | MURF | LRET | H.264 Reference Software | % Increase in bit rate as compared to H.264 |
|----------------------|--------|--------|--------------------------------|--|
| Hall.cif | 181.18 | 180.97 | 178.81 | 0.011 |
| Foreman.qcif | 107.67 | 106.56 | 105.86 | 0.006 |
| Carphone.qcif | 126.74 | 126.73 | 126.2 | 0.004 |
| Mother_daughter.qcif | 35.81 | 35.83 | 36.19 | -0.01 |

Again, behavior of LRET as compared to reference software is as expected. Worse matches result in higher residuals and thus require more bits to encode the information.

As compared to MURF, LRET gets better compression. But from Table 5.2, it can be seen that the compression efficiency is achieved at the expense of lower PSNR. This suggests that LRET has more “skipped” blocks.

Interestingly, for the “mother_daughter.qcif” sequence, it is observed that LRET takes less number of bits than reference software. Also, LRET takes much less ME time for the sequence too (See Table 5.1). This result is due to SHEX. SHEX too employs early termination, albeit in the spatial domain. Thus, the reference frame order becomes more important. Consider an example where a block is searched in two reference frames: R1 and R2. Match in R1 has a residual of 999 bits while match in R2 has a residual of 600 bits. If the SHEX termination threshold is set to 1000 bits, SHEX will find a match in either of the reference frames but the order of reference frames’ search will determine the residual left and thereby the compression efficiency. Thus, in this particular sequence, LRET gives a better search order of reference frames.

5.6 Comparison based on reference frames selection

This section uses the RFP tool to compare the choice of reference frame selected by LRET and H.264 reference software.

Figure 5.14 shows the reference frames selected by H.264 reference software. The reference software does full search in both temporal and spatial domain. So the matches found are the best possible ones. The aim of LRET is to achieve the same references for each block as shown in Figure 5.14.

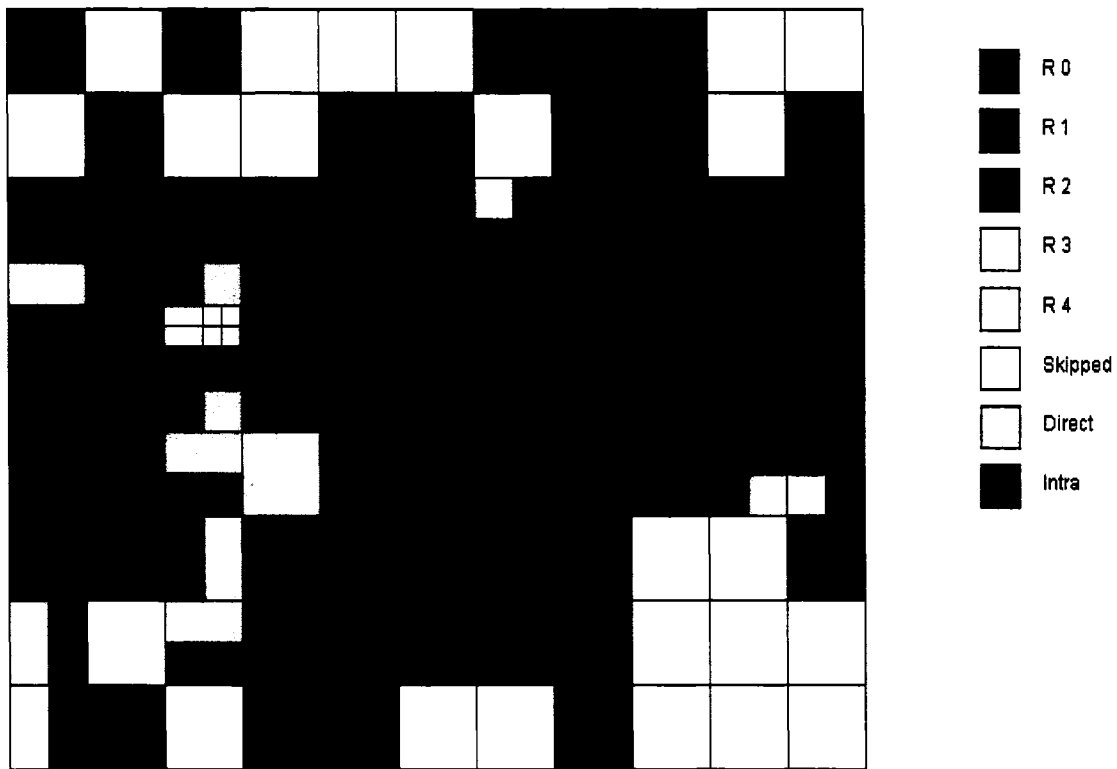


Figure 5.14: Reference frames selected by H.264 reference software

Figure 5.15 shows the reference frames selected by LRET. It can be seen that Figures 5.14 and 5.15 do not match exactly. One reason for this anomaly is the “initial ignorance” of LRET. That is, for the first macroblock, LRET has no history of local usage of reference frames. So it searches in chronologically reverse order of reference frames. The feature of early termination results in a not-so-good match. Information about this match is then used in selecting the reference frame order for the next block. This way, the error gets propagated.

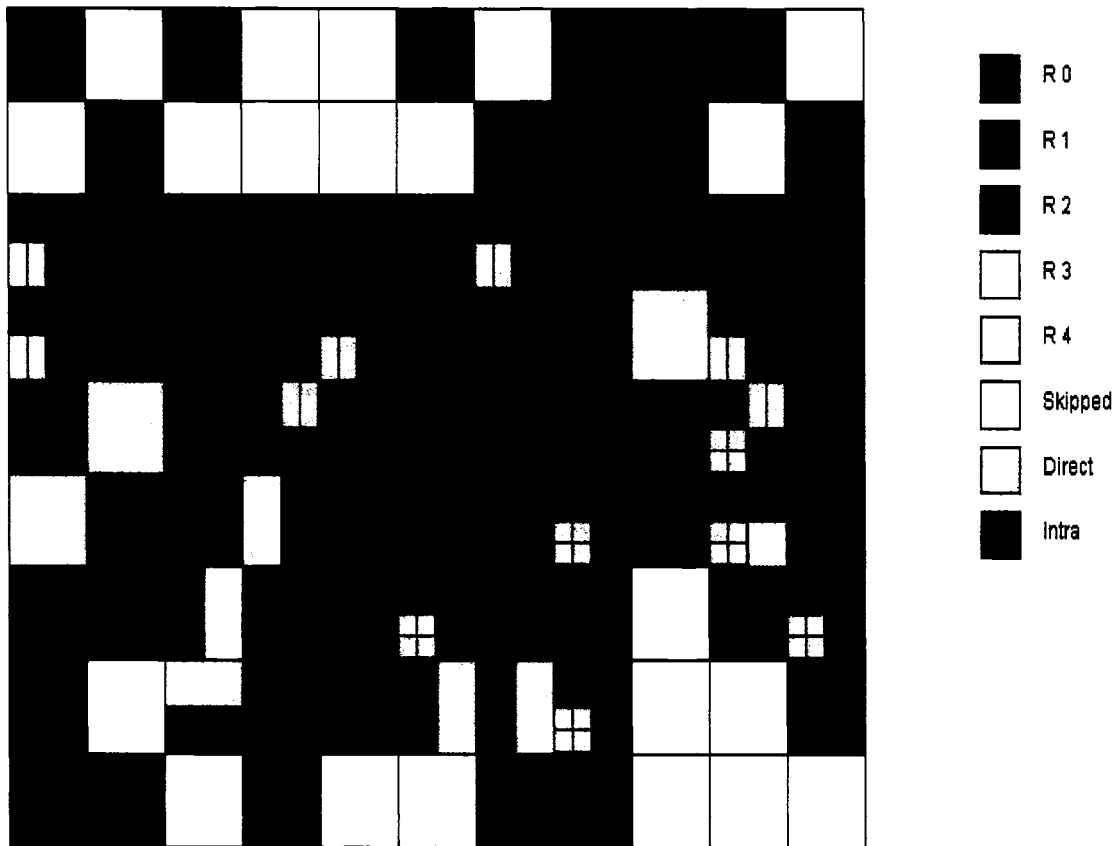


Figure 5.15: Reference frames selected by LRET

In order to remove the initial ignorance problem, the early termination threshold used by LRET is incremented gradually. That is, for the first row, the threshold is set to zero, for the second row to 100, for the third to 200, and so on. This enables LRET to get the best matches in the first row and then build from there. Figure 5.16 shows the reference frames selected when the termination threshold is incremented gradually.

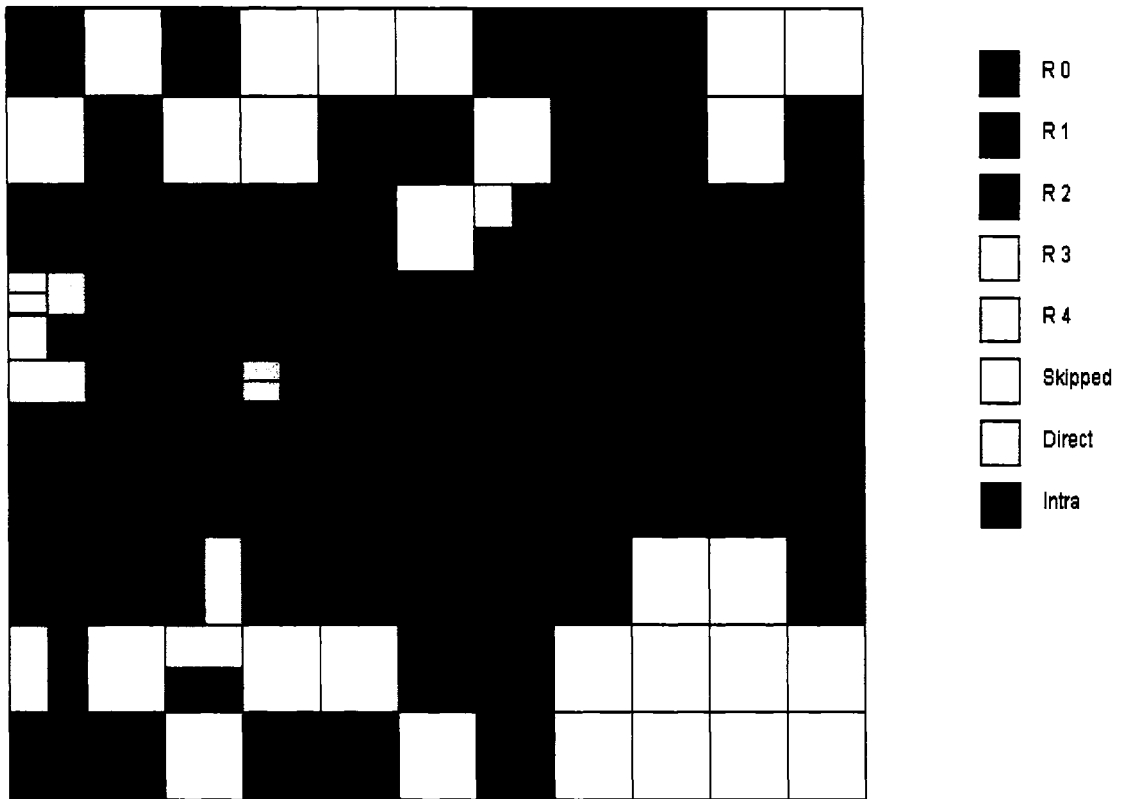


Figure 5.16: Reference frames selected by LRET with increasing early termination threshold

Comparison between Figures 5.14 and 5.16 shows that gradually increasing the early termination threshold gives better matches for selection of reference frames.

6 Conclusion

In this paper, a new algorithm, LRET – Local Reference with Early Termination – is proposed and described in detail. LRET reduces the H.264 motion estimation time by prioritizing the reference frame selection, and terminating the search early. LRET optimizes the search order in the temporal domain while the existing H.264 algorithms reorder the search in the spatial domain (within a frame). Thus, LRET can be used in conjunction with any of the existing H.264 algorithms to get better results.

Also, a tool RFP – Reference Frame Plotter – is created and described. RFP shows the reference frames selected by H.264 encoder in a graphical format and may be used to analyze the performance of H.264 based encoder.

Further, the performance of LRET is analyzed in terms of motion estimation time, video quality, compression efficiency, and selection of reference frames. Although ME time is not affected much by the search range, it is found to decrease with an increase in the early termination threshold, a decrease in the number of reference frames, and the introduction of partition weights. Also, it is seen that the video quality and compression efficiency do not vary much for the various inputs.

Finally, a comparison between performance of LRET, MURF, and H.264 reference software is presented. Though, performance of LRET is found to be similar to that of MURF, as compared to H.264 reference software, LRET is found to achieve up to 59% reduction in motion estimation time with negligible degradation in video quality (0.002%) and compression (0.001%).

7 Future work

Although LRET has greatly improved the motion estimation time, it is not yet as good in video quality, or compression. As seen from section 5.6, the choice of reference frames is not the same as that of H.264 reference software. Further work is required to optimize on the algorithm to get better matches. Some of the areas where improvements can be done are mentioned below.

The weights given for each partition size can be refined, so that they produce better search order of reference frames. Similar weights can be introduced for LRET locality search range.

Also, it is observed (refer to macroblock number 23 in Figures 5.14 and 5.15) that LRET does not get the best match even though the best match came from the reference frame that had highest local usage. This issue needs to be investigated further.

References

- Chang, A., Au, O. C., & Yeung, Y. M. (2003, April). A novel approach to fast multi-frame selection for H.264 video coding. *Proceedings of IEEE International Conference on Acoustics Speech and Signal Processing*, 703-707.
- Chen, M.-J., Chang, Y.-Y., Li, H.-J., & Chi, M.-C. (2004, May). Efficient multi-frame motion estimation algorithms for MPEG-4 AVC/JVT/H.264. *Proceedings of IEEE International Symposium on Circuits and System*, 737-740.
- H.264/MPEG-4 AVC*. (n.d.). Retrieved August 28, 2006, from Wikipedia: The Free Encyclopedia site: <http://en.wikipedia.org/wiki/H.264>
- H.264/MPEG-4 Part 10 Tutorials*. (n.d.). Retrieved August 28, 2006, from <http://www.vcodex.com/h264.html>
- Huang, Y. W., Hsieh, B.-Y., Wang, T.-C., Chient, S.-Y., Ma, S.-Y., Shen, C.-F., & Chen, L.-G. (2003, April). Analysis and Reduction of Reference Frames for Motion Estimation in MPEG-4 AVC/JVT/H.264 [Electronic Version]. *Proceedings of IEEE International Conference on Acoustics Speech and Signal Processing*, 145-148.
- Jiang, Y., Li, S., & Goto S. (2004, July). Low complexity variable block size motion estimation algorithm for video telephony communication. *The 2004 47th Midwest Symposium on Circuits and Systems, Vol. 2*.
- Li, G.-L., Chen, M.-J., Li, H.-J., & Hsu, C.-T. (2005, May). Efficient Search and Mode Prediction Algorithms for Motion Estimation in H.264/AVC [Electronic Version]. *IEEE International Symposium on Circuits and System, 2005*.
- Mahajan, A., & Kondayya, S. (2006, May) Advanced Video Coding. *Project Report submitted to Faculty of the Department of Computer Science, San Jose State University, San Jose, CA*.
- Sullivan, G. J., & Wiegand, T. (2004, December). Video Compression – From Concepts to the H.264/AVC Standard [Electronic Version]. *Proceedings of the IEEE*.
- Tourapis, A. M., Suehring, K., & Sullivan, G. (2004). *H.264/MPEG-4 AVC Reference Software Manual*. Retrieved September 9, 2006, from http://ftp3.itu.ch/av-arch/jvt-site/2004_10_Palma/JVT-M012r0.doc
- Wiegand, T., Sullivan, G., & Luthra, A., *Draft ITU-T Recommendation and Final Draft International Standard of Joint Video Specification*. Retrieved September 12,

2006, from http://ip.hhi.de/imagecom_G1/assets/pdfs/JVT-G050.pdf

Yu, A.C. (2004, May). Efficient block-size selection algorithm for inter-frame coding in H.264/MPEG-4 AVC. *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing, Vol. 3.*

Zhou, Z., Sun, M.-T., & Hsu, Y.-F. (2004, May). Fast Variable Block-Size Motion Estimation Algorithms Based On Merge and Split Procedures for H.264/Mpeg-4 AVC. *Proceedings of the 2004 International Symposium on Circuits and Systems, 2004. Vol. 3.*

Appendix A

Source code of reference frame plotter

```
import java.io.*;
import java.util.*;
import java.lang.reflect.*;
import java.awt.*;
import javax.swing.*;
import java.awt.event.*;
import javax.imageio.*;
import java.awt.image.*;

public class InputFileParser extends JPanel
{
    public static void main(String[] args)
    {
        InputFileParser ifp = new InputFileParser();
        if (Array.getLength(args) < 3)
        {
            System.out.println("USAGE : java InputFileParser
<input file name> <picture number> <square-size (Output
file Name with .jpeg)");
            System.out.println("USAGE : java InputFileParser
optional, if not given image will not be saved, if saved
only jpeg format");
            System.exit(-1);
        }
        else
        {
            ifp.mFileName = args[0];
            ifp.mPicChoice = Integer.parseInt(args[1]);
            ifp.mbHeight = ifp.mbWidth =
Integer.parseInt(args[2]);
            if (Array.getLength(args) == 4)
            {
                ifp.mOutputFile = args[3];
            }
        }

        try
        {
            int line = 0;
            BufferedReader in = new BufferedReader(new
```

```

FileReader(ifp.mFileName));
    String strIN;
    while (!ifp.mDone && ((strIN = in.readLine()) !=
null))
        {
            int ret = ifp.processLine(strIN);
        }
    in.close();
    if(ifp.mDone)
        ifp.startMe();
    else
        System.out.println("Error: Input file must be present");
    }
    catch (IOException e)
    {
        System.out.println("Error: "+e.getMessage());
    }
}

public void startMe()
{
    JFrame frame = new JFrame();

    frame.getContentPane().add(this, java.awt.BorderLayout.CENTE
R);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(mPicWidth + 200 , (mPicHeight+100));
    frame.setVisible(true);
    frame.addWindowListener(new
java.awt.event.WindowAdapter()
    {
        public void windowClosing(WindowEvent winEvt)
        {
            if (mOutputFile != "")
            {
                System.out.println("Saving the output to: " +
mOutputFile);
                try
                {
                    System.out.println("Saving the output to: "
+
"");
                    BufferedImage image = new
BufferedImage(getWidth(), getHeight(),
BufferedImage.TYPE_INT_RGB);

```

```

        Graphics2D g = image.createGraphics();

g.setRenderingHint(RenderingHints.KEY_FRACTIONALMETRICS, Ren
deringHints.VALUE_FRACTIONALMETRICS_ON);

g.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING, Ren
deringHints.VALUE_TEXT_ANTIALIAS_ON);
        paint(g);
        g.dispose();
        System.out.println("Saving to the file
2");
        ImageIO.write(image, "jpeg", new
File(mOutputFile));
        System.out.println("Saving to the file
3");
        image.flush();
        System.out.println("Saving to the file
4");
    }
    catch (IOException e)
    {
        System.out.println("Saving to the file
FAILED:" + e.getMessage());
        e.printStackTrace();
    }
    System.out.println("Saving to the file OK:"
+ mOutputFile);
    }
    System.exit(0);
}
} );
}

public void paintComponent(Graphics g)
{
    ListIterator iterX = mPicdata.listIterator();
    int x = 0, y = 0;
    while (iterX.hasNext())
    {
        MacroBlock mb = (MacroBlock)iterX.next();
        ListIterator iterY = mb.squaresdata.listIterator();
        while (iterY.hasNext())
        {
            SquareData box = (SquareData)iterY.next();
            drawBox(g, mb.offsetX + box.x, mb.offsetY +
box.y, box.wd, box.ht, getColor(box.color));
            ListIterator iterSUB =

```



```

box.subSquaresdata.listIterator();
    while (iterSUB.hasNext())
    {
        SquareData subBox =
(SquareData)iterSUB.next();
        drawBox(g, mb.offsetX + subBox.x,
mb.offsetY + subBox.y, subBox.wd,
subBox.ht,getColor(box.color));
    }
    }
    x++;
}
int h =25;
for (int c = 0; c < mRefFrames; c++,h +=35)
{
    drawBox(g,mPicWidth+75,h,25,25,getColor(c));
    g.setColor(Color.black);
    String str = ""+c;
    g.drawString(str,mPicWidth+75+40,h+15);
}
drawBox(g,mPicWidth+75,h,25,25,getColor(20));
g.setColor(Color.black);
g.drawString("Steps",mPicWidth+75+40,h+15);
drawBox(g,mPicWidth+75,h+35,25,25,getColor(21));
g.drawString("Direct",mPicWidth+75+40,h+50);
g.setColor(Color.black);
drawBox(g,mPicWidth+75,h+70,25,25,getColor(22));
g.setColor(Color.black);
g.drawString("Indirect",mPicWidth+75+40,h+85);

} //close paintComponent
public void drawBox(Graphics g,int mx, int my, int ht,
int wd, Color c)
{
    g.setColor(Color.black); // black to draw the border
    g.drawRect(mx, my, ht,wd);
    g.setColor(c);
    g.fillRect(mx+1,my+1,ht-1,wd-1);
}
public Color getColor(int c)
{
    Color color = Color.gray;

    if (c == 0)
        color = Color.cyan;
}

```

```

if (c == 1)
    color = Color.green;
if (c == 2)
    color = Color.magenta;
if (c == 3)
    color = Color.yellow;
if (c == 4)
    color = Color.pink;
if (c == 5)
    color = Color.blue;
if (c == 6)
    color = Color.red;
if (c == 7)
    color = new Color(100,150,150);
if (c == 8)
    color = new Color(100,150,100);
if (c == 9)
    color = new Color(100,100,150);
if (c == 10)
    color = new Color(150,100,100);
if (c == 11)
    color = new Color(150,100,150);
if (c == 12)
    color = new Color(150,150,100);
if (c == 13)
    color = new Color(200,150,100);
if (c == 14)
    color = new Color(200,100,150);
if (c == 15)
    color = new Color(200,100,100);
if (c == 16)
    color = new Color(200,200,100);
if (c == 22)
    color = Color.darkGray; //intra
if (c == 21) //direct
    color = Color.orange;
if (c == 20)
    color = Color.white; //skipped
return color;
}

int mMBNum;
int mCurrentX;
int mCurrentY;
int mbWidth;
int mbHeight;

```

```

int mPicWidth;
int mPicHeight;
int mRefIdx;
int m8X8Idx;
int mRefFrames;
int i,j;
int mTotalMB;
int mPicChoice;

boolean mPicFound;
boolean mMBFound;
boolean mMBTypeFound;
boolean mDone;

String mFileName;
String mOutputFile;

class SquareData
{
    int x;
    int y;
    int wd;
    int ht;
    int color;
    Vector subSquaresdata; // list of sub SquareData

    SquareData()
    {
        x = 0;
        y = 0;
        wd = 0;
        ht = 0;
        color = 0;
        subSquaresdata = new Vector();
    }
};

class MacroBlock
{
    int offsetX;
    int offsetY;
    int mbType;
    Vector squaresdata; // list of SquareData
    MacroBlock()
    {
        offsetX = 0;

```

```

        offsetY = 0;
        mbType = 0;
        squaresdata = new Vector();
    }
};

Vector mPicdata; // list of MacroBlock
MacroBlock mMacroBlock;
SquareData mSquareData;

InputFileParser()
{
    mbWidth = 100;
    mbHeight = 100;
    mPicWidth = 200;
    mPicHeight = 200;
    mMBNum = 0;
    mCurrentX = 0;
    mCurrentY = 0;
    i = j = 0;
    mRefIdx = 0;
    m8X8Idx = 0;
    mMBFound = false;
    mPicFound = false;
    mMBTypeFound = false;
    mDone = false;
    mTotalMB = 1;
    mPicChoice = 0;
    mPicdata = new Vector();
    mFileName = "";
    mRefFrames = 0;
    mOutputFile = "";
}

public int processLine(String strIN)
{
    String value = "";
    String tempStr = strIN;
    StringTokenizer tokens = new StringTokenizer(tempStr);
    int n=0;
    while (tokens.hasMoreTokens())
    {
        String token = tokens.nextToken();
        if (token.endsWith("num. of frames"))
        {
            tokens.nextToken(); //skip 1 tokens
            tokens.nextToken(); //skip 1 tokens

```

```

        value = removeLastChar(tokens.nextToken());
        mRefFrames = Integer.parseInt(value);
    }

    if (token.endsWith("pic_width_in_mb_min:"))
    {
        tokens.nextToken(); //skip 1 tokens
        tokens.nextToken(); //skip 1 tokens
        value = removeLastChar(tokens.nextToken());
        int xNum = Integer.parseInt(value) + 1;
        mPicWidth = xNum * mbWidth; // 40x40 is the
square size
        mTotalMB *= xNum;
    }
    if
(token.endsWith("pic_height_in_mb_min:"))
    {
        tokens.nextToken(); //skip 1 tokens
        tokens.nextToken(); //skip 1 tokens
        value = removeLastChar(tokens.nextToken());
        int yNum = Integer.parseInt(value) + 1;
        mPicHeight = yNum * mbHeight;
        mTotalMB *= yNum;
    }
    if (token.equals("#pic:"))
    {
        value = tokens.nextToken();
        int picNum = Integer.parseInt(value);
        if(picNum == mPicChoice) //user input
        {
            if (!mPicFound)
                mPicFound = true;
            tokens.nextToken(); //skip one token
            value = tokens.nextToken();
            if ( value.equals("MB:") )
            {
                value = tokens.nextToken();
                int mMBNum = Integer.parseInt(value);
                if ( mMBNum >= 0 )
                {
                    if (mMBFound)
                    {
                        if (!mMBTypeFound)
                        {
                            // this means we did not

```

```

find any mb_type, so this has to be white, skipped
SquareData sd1 = new
SquareData();

sd1.x = sd1.y = 0;
sd1.ht = sd1.wd = mbWidth;
sd1.color = 20; // skipped

mMacroBlock.squaresdata.add(sd1);
}
// we found a new MB so add the
previous active one into the main vector
mPicdata.add(mMacroBlock);
}
mMBFound = true;
mMBTypeFound = false;
j = 0; //initialise j
mRefIdx = 0;
m8X8Idx = 0;
mMacroBlock = new MacroBlock();
if (mPicdata.size() == 0) // this
is a first MB
{
mMacroBlock.offsetX =
mCurrentX;
mMacroBlock.offsetY =
mCurrentY;
}
else
{
if ( ( mMacroBlock.offsetX =
mCurrentX + mbWidth ) > ( mPicWidth - mbWidth ) )
{
mMacroBlock.offsetX = 0;
mMacroBlock.offsetY =
mCurrentY + mbHeight;
}
else
{
mMacroBlock.offsetY =
mCurrentY;
}
}
mCurrentX = mMacroBlock.offsetX;
mCurrentY = mMacroBlock.offsetY;
}
}

```

```

    }
}
else
{
    if (mPicFound)
    {
        //this is the last mb of the pic
        if (mMBFound)
        {
            if (!mMBTypeFound)
            {
                // this means we did not find
                any mb_type, so this has to be white, skipped
                SquareData sd1 = new
SquareData();

                sd1.x = sd1.y = 0;
                sd1.ht = sd1.wd = mbWidth;
                sd1.color = 20; //skipped

mMacroBlock.squaresdata.add(sd1);
            }
            // we found a new MB so add the
previous active one into the main vector
            mPicdata.add(mMacroBlock);
        }

        mMacroBlock = new MacroBlock();
        if ( ( mMacroBlock.offsetX = mCurrentX
+ mbWidth ) > ( mPicWidth - mbWidth ) )
        {
            mMacroBlock.offsetX = 0;
            mMacroBlock.offsetY = mCurrentY +
mbHeight;
        }
        else
        {
            mMacroBlock.offsetY = mCurrentY;
        }
        mCurrentX = mMacroBlock.offsetX;
        mCurrentY = mMacroBlock.offsetY;
        mDone = true; //TODO: cleanup here
    }
    j = 0; //initialise j
    mRefIdx = 0;
    m8X8Idx = 0;

```

```

        mMBFound = false;
        mMBTypeFound = false;
        mPicFound = false;
        // we processed all the MBs, can return
from here
        if (mPicdata.size() == mTotalMB)
        {
            mDone = true;
        }
    }
    if (token.endsWith("mb_type") && mPicFound)
    {
        mMBTypeFound = true;
        value = getValueAfterEquaSign(strIN);
        int mbType = Integer.parseInt(value);
        mMacroBlock.mbType = mbType;
        if (mbType == 1)
        {
            // one full block
            SquareData sd1 = new SquareData();
            sd1.x = sd1.y = 0;
            sd1.ht = sd1.wd = mbWidth;
            mMacroBlock.squaresdata.add(sd1);
        }
        else if (mbType == 2)
        {
            // it has two small blocks devided
horizontally
            SquareData sd1 = new SquareData();
            sd1.x = sd1.y = 0;
            sd1.ht = (int) mbHeight/2;
            sd1.wd = mbWidth;
            SquareData sd2 = new SquareData();
            sd2.y = (int) mbHeight/2;
            sd2.x = 0;
            sd2.ht = (int) mbHeight/2;
            sd2.wd = mbWidth;
            mMacroBlock.squaresdata.add(sd1);
            mMacroBlock.squaresdata.add(sd2);
        }
        else if (mbType == 3)
        {
            SquareData sd1 = new SquareData();
            sd1.x = sd1.y = 0;
            sd1.ht = mbHeight;

```



```

        sd1.wd = (int) mbWidth/2;
        SquareData sd2 = new SquareData();
        sd2.y = 0;
        sd2.x = (int) mbWidth/2;;
        sd2.ht = mbHeight;
        sd2.wd = (int) mbWidth/2;
        mMacroBlock.squaresdata.add(sd1);
        mMacroBlock.squaresdata.add(sd2);
    }
else if (mbType == 8)
    {
        SquareData sd1 = new SquareData();
        sd1.x = sd1.y = 0;
        sd1.ht = (int) mbHeight/2;
        sd1.wd = (int) mbWidth/2;

        SquareData sd2 = new SquareData();
        sd2.y = 0;
        sd2.x = (int) mbWidth/2;;
        sd2.ht = (int) mbHeight/2;
        sd2.wd = (int) mbWidth/2;

        SquareData sd3 = new SquareData();
        sd3.y = (int) mbHeight/2;
        sd3.x = 0;
        sd3.ht = (int) mbHeight/2;
        sd3.wd = (int) mbWidth/2;

        SquareData sd4 = new SquareData();
        sd4.y = (int) mbHeight/2;
        sd4.x = (int) mbWidth/2;;
        sd4.ht = (int) mbHeight/2;
        sd4.wd = (int) mbWidth/2;

        mMacroBlock.squaresdata.add(sd1);
        mMacroBlock.squaresdata.add(sd2);
        mMacroBlock.squaresdata.add(sd3);
        mMacroBlock.squaresdata.add(sd4);
    }
else if (mbType > 8)
    {
        // one full block
        SquareData sd1 = new SquareData();
        sd1.x = sd1.y = 0;
        sd1.ht = sd1.wd = mbWidth;
        sd1.color = 22; // intra

```

```

        mMacroBlock.squaresdata.add(sd1);
    }
}
if (token.endsWith("x8") &&
tokens.nextToken().equals("m8x8; 811") && mPicFound)
{
    String temp = getValueAfterEquaSign(strIN);
    int l = temp.length();
    value = temp.substring(0,l-2);
    int i8x8Type = Integer.parseInt(value);
    SquareData sd = (SquareData)
mMacroBlock.squaresdata.get(m8X8Idx);
    if (i8x8Type == 0) //means it is not divided
but it has be of different color
    {
        sd.color = 21; //direct
    }
    else if (i8x8Type == 5) //means it is divided
in half horizontally
    {
        SquareData sd1 = new SquareData();
        sd1.y = sd.y;
        sd1.x = sd.x;
        sd1.ht = (int) sd.ht/2;
        sd1.wd = sd.wd;
        sd1.color = sd.color;

        SquareData sd2 = new SquareData();
        sd2.x = sd.x;
        sd2.y = sd.y + (int) (sd.ht/2);
        sd2.ht = (int) sd.ht/2;
        sd2.wd = sd.wd;
        sd2.color = sd.color;
        sd.subSquaresdata.add(sd1);
        sd.subSquaresdata.add(sd2);
    }
    else if (i8x8Type == 6) //means it is divided
in half vertically
    {
        SquareData sd1 = new SquareData();
        sd1.y = sd.y;
        sd1.x = sd.x;
        sd1.wd = (int) sd.wd/2;
        sd1.ht = sd.ht;
        sd1.color = sd.color;
    }
}

```

```

        SquareData sd2 = new SquareData();
        sd2.y = sd.y;
        sd2.x = sd.x + (int) (sd.wd/2);
        sd2.ht = sd.ht;
        sd2.wd = (int) sd.wd/2;
        sd2.color = sd.color;
        sd.subSquaresdata.add(sd1);
        sd.subSquaresdata.add(sd2);
    }
    else if (i8x8Type == 7) //means it is divided
in 4 parts
    {
        SquareData sd1 = new SquareData();
        sd1.x = sd.x;
        sd1.y = sd.y;
        sd1.ht = (int) sd.ht/2;
        sd1.wd = (int) sd.wd/2;
        sd1.color = sd.color;

        SquareData sd2 = new SquareData();
        sd2.x = sd.x + (int) sd.wd/2;
        sd2.y = sd.y;
        sd2.ht = (int) sd.ht/2;
        sd2.wd = (int) sd.wd/2;
        sd2.color = sd.color;

        SquareData sd3 = new SquareData();
        sd3.y = sd.y + (int) sd.ht/2;
        sd3.x = sd.x;
        sd3.ht = (int) sd.ht/2;
        sd3.wd = (int) sd.wd/2;
        sd3.color = sd.color;

        SquareData sd4 = new SquareData();
        sd4.y = sd.y + (int) sd.ht/2;
        sd4.x = sd.x + (int) sd.wd/2;
        sd4.ht = (int) sd.ht/2;
        sd4.wd = (int) sd.wd/2;
        sd4.color = sd.color;

        sd.subSquaresdata.add(sd1);
        sd.subSquaresdata.add(sd2);
        sd.subSquaresdata.add(sd3);
        sd.subSquaresdata.add(sd4);
    }
    mMacroBlock.squaresdata.set(m8X8Idx, sd);

```

```

        m8X8Idx++;
    }
    if (token.endsWith("ref_idx_1") && mPicFound)
    {
        value = getValueAfterEquaSign(strIN);
        int color = Integer.parseInt(value);
        SquareData sd = (SquareData)
mMacroBlock.squaresdata.get(mRefIdx);
        sd.color = color;
        mMacroBlock.squaresdata.set(mRefIdx, sd);
        mRefIdx++;
    }
    }
    return 0;
}
public String removeLastChar(String strIN)
{
    int l = strIN.length();
    return strIN.substring(0,l-1);
}
public String removeFirst7Char(String strIN)
{
    int l = strIN.length();
    if (l > 7)
        return strIN.substring(7,l);
    else
        return strIN;
}
public String getValueAfterEquaSign(String strIN)
{
    int l = strIN.length();
    int equalAT = strIN.indexOf('=');
    String sub = strIN.substring(equalAT+1);
    StringTokenizer tokens = new StringTokenizer(sub);
    if (tokens.hasMoreTokens())
    {
        return tokens.nextToken();
    }
    return ".";
}
}
}

```

Appendix B

Experimental data for locality search range variation

Table B.1: ME time vs locality search range for SIF and CIF sequences

| Search Range | Coastguard (CIF) | Salesman (CIF) | Hall (CIF) | Mobile (SIF) | Football (SIF) | Tennis (SIF) |
|--------------|------------------|----------------|------------|--------------|----------------|--------------|
| 1 | 455.1 | 541.066 | 336.139 | 338.676 | 163.067 | 101.21 |
| 3 | 433.529 | 542.488 | 335.75 | 342.783 | 162.834 | 100.546 |
| 5 | 430.744 | 541.699 | 335.636 | 346.432 | 162.796 | 100.345 |
| 7 | 429.245 | 541.451 | 336.483 | 346.38 | 162.76 | 100.313 |
| 9 | 429.061 | 542.157 | 336.35 | 347.989 | 162.847 | 100.237 |
| 11 | 427.458 | 542.372 | 336.635 | 348.694 | 162.765 | 100.258 |
| 13 | 426.324 | 549.619 | 337.48 | 349.946 | 162.915 | 100.939 |
| 15 | 425.591 | 555.162 | 336.829 | 349.63 | 163.36 | 100.392 |
| 17 | 425.925 | 547.704 | 336.283 | 350.444 | 162.569 | 100.275 |
| 19 | 425.687 | 542.899 | 337.054 | 349.938 | 162.871 | 100.181 |
| 21 | 424.843 | 543.197 | 336.694 | 349.723 | 162.944 | 100.643 |
| 22 | 425 | 544.332 | 337.04 | 349.293 | 163.335 | 100.486 |

Table B.2: ME time vs locality search range for QCIF sequences

| Search Range | Carphone (QCIF) | Mother_Daughter (QCIF) | Foreman (QCIF) |
|--------------|--------------------|---------------------------|-------------------|
| 1 | 82.449 | 43.793 | 71.482 |
| 2 | 82.337 | 43.917 | 71.19 |
| 3 | 82.178 | 43.771 | 71.776 |
| 4 | 82.483 | 43.727 | 71.46 |
| 5 | 82.171 | 43.875 | 71.589 |
| 6 | 82.296 | 43.701 | 71.294 |
| 7 | 82.454 | 43.691 | 71.219 |
| 8 | 82.275 | 43.866 | 71.6 |
| 9 | 82.25 | 43.643 | 71.253 |
| 10 | 82.334 | 43.84 | 71.09 |
| 11 | 82.24 | 43.722 | 71.065 |
| 12 | 82.678 | 43.548 | 70.987 |

Table B.3: PSNR vs locality search range for SIF and CIF sequences

| Search Range | Coastguard (CIF) | Salesman (CIF) | Hall (CIF) | Mobile (SIF) | Football (SIF) | Tennis (SIF) |
|--------------|------------------|----------------|------------|--------------|----------------|--------------|
| 1 | 35.55 | 36.98 | 37.54 | 33.25 | 33.03 | 33.96 |
| 3 | 35.55 | 36.98 | 37.54 | 33.26 | 33.03 | 33.96 |
| 5 | 35.55 | 36.98 | 37.54 | 33.26 | 33.03 | 33.96 |
| 7 | 35.54 | 36.98 | 37.54 | 33.26 | 33.03 | 33.96 |
| 9 | 35.54 | 36.98 | 37.54 | 33.26 | 33.03 | 33.97 |
| 11 | 35.54 | 36.98 | 37.54 | 33.26 | 33.03 | 33.97 |
| 13 | 35.54 | 36.98 | 37.54 | 33.26 | 33.03 | 33.97 |
| 15 | 35.54 | 36.98 | 37.54 | 33.26 | 33.03 | 33.97 |
| 17 | 35.54 | 36.98 | 37.54 | 33.26 | 33.03 | 33.97 |
| 19 | 35.54 | 36.98 | 37.54 | 33.27 | 33.03 | 33.97 |
| 21 | 35.55 | 36.98 | 37.54 | 33.27 | 33.03 | 33.96 |
| 22 | 35.55 | 36.98 | 37.54 | 33.27 | 33.03 | 33.96 |

Table B.4: PSNR vs locality search range for QCIF sequences

| Search Range | Carphone (QCIF) | Mother_Daughter (QCIF) | Foreman(QCIF) |
|--------------|--------------------|---------------------------|---------------|
| 1 | 36.38 | 37.57 | 35.9 |
| 2 | 36.41 | 37.58 | 35.93 |
| 3 | 36.44 | 37.6 | 35.93 |
| 4 | 36.43 | 37.6 | 35.94 |
| 5 | 36.44 | 37.6 | 35.93 |
| 6 | 36.47 | 37.62 | 35.93 |
| 7 | 36.46 | 37.62 | 35.95 |
| 8 | 36.45 | 37.6 | 35.94 |
| 9 | 36.46 | 37.63 | 35.95 |
| 10 | 36.45 | 37.61 | 35.95 |
| 11 | 36.45 | 37.61 | 35.95 |
| 12 | 36.45 | 37.61 | 35.95 |

Table B.5: Bit rate vs locality search range for SIF and CIF sequences

| Search Range | Coastguard (CIF) | Salesman (CIF) | Hall (CIF) | Mobile (SIF) | Football (SIF) | Tennis (SIF) |
|--------------|------------------|----------------|------------|--------------|----------------|--------------|
| 1 | 2048.44 | 1105.79 | 1061.13 | 1674.35 | 2697.4 | 1174.96 |
| 3 | 2047.65 | 1105.39 | 1060.82 | 1674.24 | 2698.47 | 1175.58 |
| 5 | 2047.95 | 1105.48 | 1060.39 | 1673.72 | 2696.36 | 1175.78 |
| 7 | 2048.09 | 1105.48 | 1060.39 | 1671.08 | 2698.11 | 1171.08 |
| 9 | 2048.35 | 1105.48 | 1060.39 | 1672.22 | 2699.54 | 1173.23 |
| 11 | 2048 | 1105.48 | 1060.39 | 1671.71 | 2696.05 | 1174.5 |
| 13 | 2047.23 | 1105.48 | 1060.39 | 1670.4 | 2696 | 1171.41 |
| 15 | 2047.94 | 1105.48 | 1060.39 | 1671.47 | 2696.65 | 1172.76 |
| 17 | 2047.18 | 1105.48 | 1060.39 | 1671.62 | 2699.04 | 1173.27 |
| 19 | 2047.2 | 1105.48 | 1060.39 | 1671.31 | 2698.01 | 1174.79 |
| 21 | 2047.99 | 1105.48 | 1060.39 | 1670.29 | 2697.6 | 1171.8 |
| 22 | 2047.99 | 1105.48 | 1060.39 | 1670.29 | 2697.6 | 1171.8 |

Table B.6: Bit rate vs locality search range for QCIF sequences

| Search Range | Carphone (QCIF) | Mother_Daughter (QCIF) | Foreman(QCIF) |
|--------------|--------------------|---------------------------|---------------|
| 1 | 128.26 | 35.9 | 108.24 |
| 2 | 127.58 | 36.05 | 107.43 |
| 3 | 127.41 | 35.98 | 108.27 |
| 4 | 126.97 | 35.86 | 107.51 |
| 5 | 126.99 | 35.76 | 107.76 |
| 6 | 126.77 | 35.8 | 106.8 |
| 7 | 126.74 | 35.78 | 107.89 |
| 8 | 126.75 | 35.78 | 107.39 |
| 9 | 126.91 | 35.96 | 106.92 |
| 10 | 126.82 | 35.92 | 106.56 |
| 11 | 126.82 | 35.92 | 106.56 |
| 12 | 126.82 | 35.92 | 106.56 |