H.264 MOTION ESTIMATION AND FLEXIBLE TRIANGLE SEARCH

by

Raymond Ngun
B.A.Sc., Simon Fraser University, 2002

PROJECT SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF ENGINEERING SCIENCE

In the
School
of
Engineering Science

SIMON FRASER UNIVERSITY

Fall 2006

APPROVAL

Name:	**Raymond Ngun**
Degree:	**Master of Engineering Science**
Title of Project:	**H.264 Motion Estimation and Flexible Triangle Search**

Supervisory Committee:

Chair: **Dr. Ivan Bajic**
Assistant Professor of Engineering Science

Dr. Jie Liang
Senior Supervisor
Assistant Professor of Engineering Science

Dr. Mohamed M. Rehan
Supervisor
Principal Scientist, Broadcom Canada

Date Defended/Approved: December 1, 2006

ABSTRACT

Motion estimation (ME) in H.264 can account for 90% of the encoding time. As such, ME optimization techniques are extensively researched. In this project, we studied and implemented ME technique called Flexible Triangle Search (FTS). Its performance is compared to other ME techniques found in the Joint Video Team (JVT) reference software. Results show the FTS rate-distortion (R-D) curve is very close to the R-D curves of other ME techniques and in turn is close to the optimum Full Search (FS) R-D curve. The benefit of the FTS technique is its complexity which is shown to be significantly less than FS and up to 30% savings from other techniques. FTS is then implemented as a quarter-pixel ME technique while full-pixel ME is completely bypassed. Experimental results show 41% savings in complexity is possible over other sub-pixel ME techniques. The results make FTS an attractive ME technique.

ACKNOWLEDGEMENTS

The author would like to thank the senior supervisor, Dr. Jie Liang and supervisor, Dr. Mohamed M. Rehan for their helpful suggestions, guidance, and support throughout the course of the project. Both of their participation is instrumental in completing the project. In addition, the author thanks his manager, Philip Houghton, for supporting the research.

Finally, the author thanks his parents and fiancée for support and encouragement.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

GLOSSARY

DCT	Discrete Cosine Transform
EFTS	Enhanced Flexible Triangle Search
EPZS	Enhanced Predictive Zonal Search
FP-FTS	Full-Pixel Flexible Triangle Search
FTS	Flexible Triangle Search
HEX	Hexagon Search
HP-FTS	Half-Pixel Flexible Triangle Search
iDCT	Inverse Discrete Cosine Transform
ITU-T	International Telecommunication Union Standardization Sector
JM	Joint Model
JVT	Joint Video Team
MB	Macroblock
ME	Motion Estimation
MPEG	Moving Picture Experts Group
MV	Motion Vector
PFTS	Predictive Flexible Triangle Search
PSNR	Peak Signal-to-Noise Ratio
QP	Quantization Parameter
QP-FTS	Quarter-Pixel Flexible Triangle Search
R-D	Rate-Distortion
SAD	Sum of Accumulated Differences
SHEX	Simplified Hexagon Search
SP-FTS	Sub-Pixel Flexible Triangle Search
UMHexagonS	Unsymmetrical Multi-Hexagon Search

CHAPTER 1
INTRODUCTION

H.264 is the latest video coding standard developed by the Moving Picture Experts Group and the ITU-T Video Coding Experts Group in the Joint Video Team (JVT). The main goal of this standard is to improve the rate-distortion (R-D) efficiency as compared to the available standards such as H.263 and MPEG-2. By improving R-D efficiency, better quality video is made possible under bandwidth limitations. As such, H.264 is beneficial to applications like video telephony over the internet.

The basic building blocks of the H.264 encoder are motion estimation (ME), discrete-cosine transform (DCT), quantization, and entropy coding. The building blocks of the decoder include de-quantization, inverse discrete-cosine transform (iDCT), and reconstruction of the picture. Of the basic building blocks, ME consumes up to 70% of the processing time and possibly even 90% when multiple reference frames are used. ME in H.264 is quite complex due to the number of features that is used in an attempt to reduce bit-rate while maintaining good picture quality (i.e. R-D efficiency). As such, much research has been completed on this topic.

The purpose of motion estimation is to remove temporal redundancy between frames resulting in better compression. H.264 uses block-based motion estimation whereby each frame is divided into a group of macroblocks. Certain frames in a video sequence are compressed using discrete cosine transform, quantization, and variable length coding. These frames or I-frames are reconstructed and used as reference frames for subsequent frames. ME is used in the subsequent frames or P-frames to remove temporal redundancy between the I and P frames. Instead of encoding actual data in the P-frames, motion vectors (MV) are encoded instead along with its error relative to the I-frame. Based on DCT and variable length coding, these MV and its errors can be compressed more efficiently than the original data resulting in overall better compression.

The JVT team developed a reference H.264 software, namely the Joint Model (JM), and is currently on revision 10.2. The software implements the recommendations that they have set forth for the standard. In regards to ME, JVT has implemented four ME techniques in their software namely the Full Search (FS), Unsymmetrical-cross Multi-Hexagon-Grid Search (UMHexagonS), Simplified UMHexagonS, and the Enhanced Predictive Zonal Search (EPZS).

This paper describes the ME techniques available in the JM and outlines another ME technique called the Flexible Triangle Search (FTS) [4]. The FTS ME technique is implemented and its performance is compared to the techniques in JM. Several enhancements are then made to FTS to further improve its complexity and the results of these enhancements are analyzed [5][7]. Finally, being a full-pixel algorithm, the FTS algorithm is extended as a sub-pixel ME where it is used as a quarter-pixel ME technique. Here, full-pixel ME is completely bypassed.

The FTS algorithm has been implemented in H.263 and analyzed in [4]. Based on the results in [4], the authors further enhanced it in enhanced FTS in [5] and predictive FTS in [7]. Both the enhanced FTS and predictive FTS were completed in H.263 as well. Finally, the authors extended FTS to a half-pixel implementation in [6]. This document extends the concepts in [4], [5], and [7] into H.264. Some of the concepts in H.264 are used with FTS to further improve FTS performance over its H.263 counterpart. Also, this document further extended FTS to a quarter-pixel implementation in H.264.

In this document, Chapter 2 provides a brief overview of H.264 and its special features. Chapter 3 discusses ME and briefly describes the ME techniques in JM. Finally, Chapter 4 discusses and analyzes the performance of the FTS algorithm, enhancements to the algorithm, and its extension as a sub-pixel ME technique.

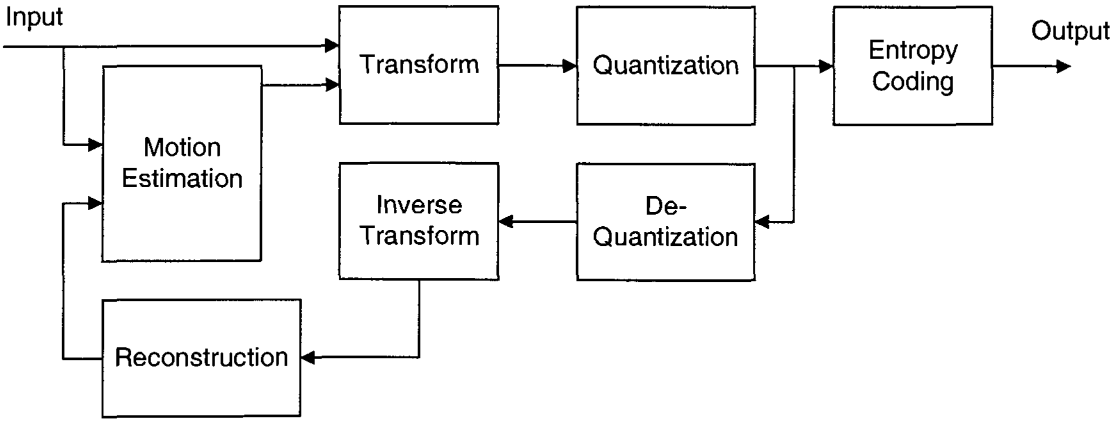
CHAPTER 2
H.264 CONCEPTS

There are a growing number of applications begging for video capabilities that carry its data over media like cable modems, DSL, and WiFi. Some of the older standards like MPEG-2 worked great for high bandwidth applications like high definition TV but falls short for bandwidth limited applications. The Moving Pictures Experts Group (MPEG) and ITU-T Video Coding Experts Group in a coalition as the Joint Video Team (JVT) developed a new standard, H.264, which aims to improve coding efficiency and hence reduce bit-rate. In addition, special attention is paid to ensure that quality is maintained. This is, in other words, known as improving on rate-distortion (R-D) efficiency. In order to do this, JVT introduced many features in H.264 aimed to accomplish just this.

It should be noted that the standard itself only standardizes the decoder by imposing restrictions on the bit stream and syntax. Thus, all encoders must adhere to the bit stream defined in the standard. As a result, there is freedom for the developers to implement the encoder in any way they desire. This allows the developer to make the complexity, time to market, and quality tradeoffs that is necessary for their application.

Figure 2-1 depicts the basic building blocks of the H.264 encoder.

Figure 2-1. H.264 Encoder



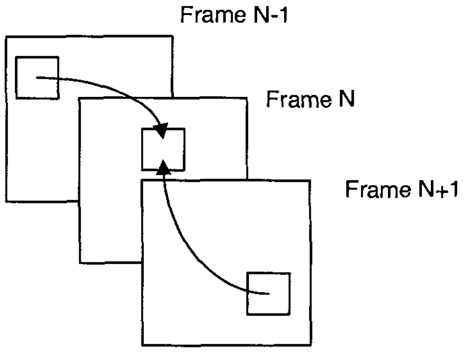
Note that the encoder contains the basic building blocks of the decoder as well since the encoder needs to decode the signal in order to generate reference frames used for ME. The Motion Estimation/Compensation block is the block of interest. Based on a reference frame which can be a frame in the past or a frame in the future, it determines motion vectors (MV) representing the movement of the objects in the picture. The de-blocking filter is common in a block-based codec like H.264 to remove blocking artifacts. The transform block is used to remove spatial redundancy. And finally, the data is quantized and entropy coded.

The following sub-sections list just some of the features of H.264 that help improve the coding efficiency and picture quality.

2-1 B-Frames

Bi-directional motion vector (MV) prediction is not a new idea but it is included in the standard because of it's usefulness in providing better compression. The concept of bi-directional prediction is to use both the past and future frames as reference. The frame coded using this prediction technique is known as a B-frame. Figure 2-2 depicts bi-directional prediction.

Figure 2-2. Illustration of a B-Frame

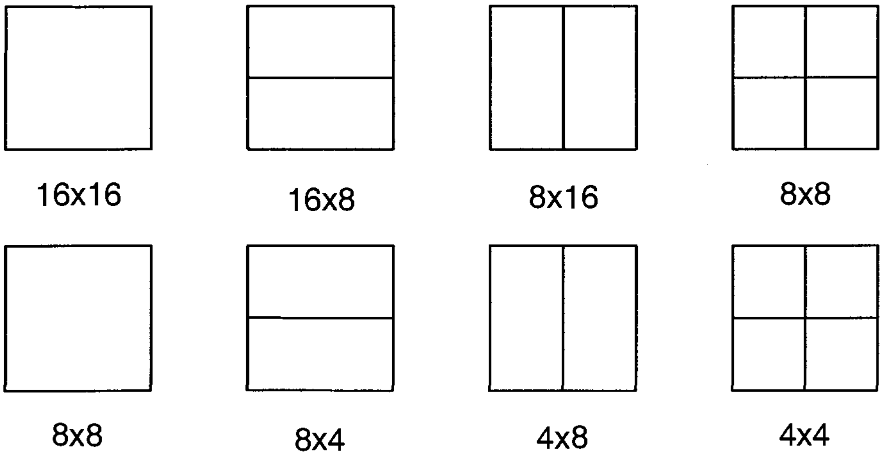


Note that B-frames are never used as reference frames.

2-2 Variable Block-Size ME and Smaller Block Sizes

A standard macroblock (MB) size is 16x16 pixels but the standard allows the use of differing block sizes in ME. The differing block sizes allow for more precise and accurate motion vectors. Figure 2-3 depicts the available block sizes used in the standard and they are 16x16, 16x8, 8x16, 8x8, 8x4, 4x8, and 4x4 pixels.

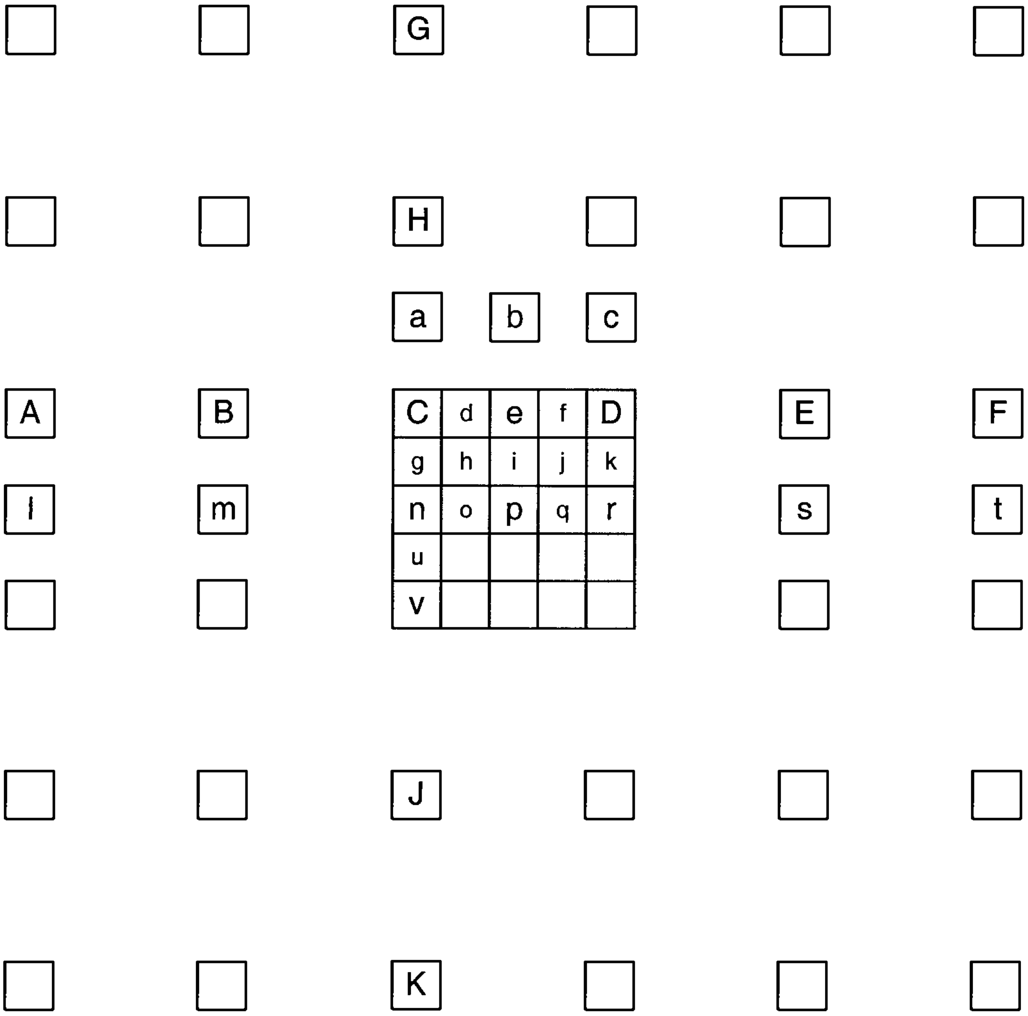
Figure 2-3. Variable Block-Sizes



2-3 Quarter-Pixel ME and Improved Interpolation

In previous standards, the introduction of half-pixel ME dramatically improved the picture quality by allowing for a better match between the current and reference frames. In H.264, this concept has been pushed a step further to include quarter-pixel ME. In addition, an improved interpolator has been introduced to calculate the quarter-pixels. Figure 2-4 depicts half-pixel and quarter-pixel locations as lower-case letters and integer-pixel locations as capital letters.

Figure 2-4. Quarter-Pixel and Half-Pixel Grid



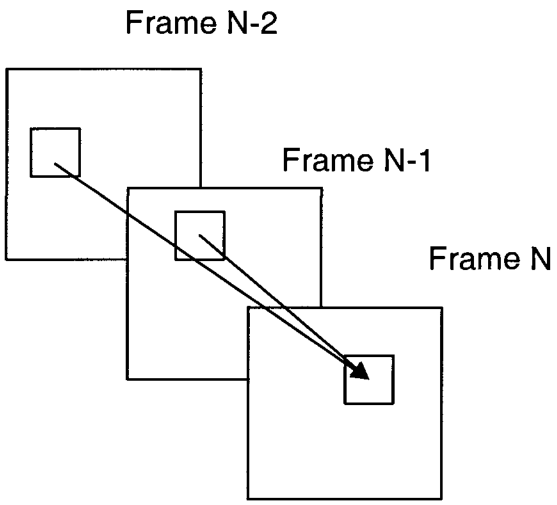
2-4 Motion Vectors Outside Picture Boundaries

Because an object can be moving outside of the picture, H.264 allows motion vectors to point outside the picture boundaries. The pixel information outside picture boundaries is deduced by the pixels at the edge of the picture. Again, this allowed for better and more accurate motion vectors and as a result, reduced bit-rate.

2-5 Multiple Reference Frames

The standard introduces the ability to utilize multiple reference frames to improve compression at the cost of much higher complexity and memory requirements. The concept of multiple reference frames is extremely useful for sequences where repetition is common. Figure 2-5 depicts the use of multiple reference frames to find the best (least cost) motion vector.

Figure 2-5. Multiple Reference Frames



2-6 Weighted Prediction

Weighted prediction is an interesting concept introduced by the standard that allows for the motion compensated picture to be weighted and offset by certain amounts. This greatly helps scenes where fading can occur.

2-7 In-loop De-blocking Filter

It is quite common for block-based video coding to produce artifacts at block edges and this is commonly known as blocking artifacts. Using a de-blocking filter to remove these artifacts is not a new idea. In H.264, the de-blocking filter is included in the motion compensation process. This allows for inter-prediction to perform better as subsequent frames can be predicted better.

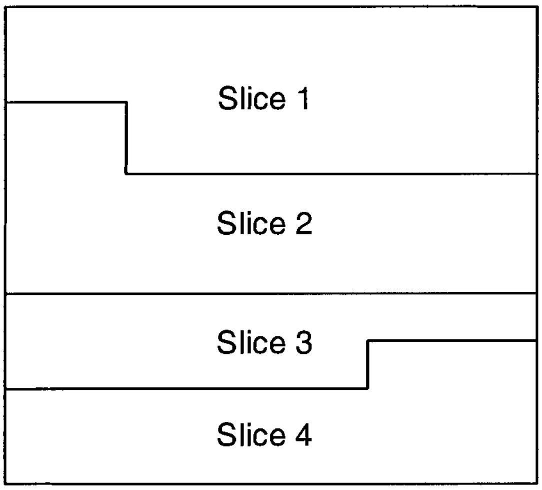
2-8 Entropy Coding

H.264 supports context-adaptive entropy coding in context-adaptive binary arithmetic coding (CABAC) and context-adaptive variable-length coding (CAVLC). Context adaptation greatly improves performance of the codec.

2-9 Slices

A slice is a collection of MBs that can be independently decoded without information from other blocks. Figure 2-6 shows a frame split into 3 slices.

Figure 2-6. Slices

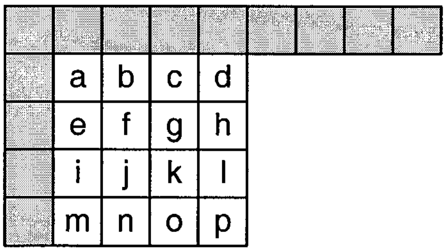


Slices are useful in separating the contents of the picture such that each slice has very little correlation with other slices.

2-10 Intra Prediction

A novel idea in H.264 is intra prediction where surrounding pixels are used to estimate a 4x4 luma frame. Depicted in Figure 2-7, the 4x4 block of pixels shown in lower case letters are predicted by the neighboring pixels in shaded boxes.

Figure 2-7. Intra-Prediction



▨	▨	▨	▨	▨	▨	▨	▨	▨
▨	a	b	c	d				
▨	e	f	g	h				
▨	i	j	k	l				
▨	m	n	o	p				

Intra-prediction can be performed in 9 different modes whereby differing sets of the neighboring pixels are used to capture the direction of movement.

CHAPTER 3
JVT MOTION ESTIMATION TECHNIQUES

The purpose of motion estimation is to remove temporal redundancy between frames. The basis of motion estimation is that an object from one frame has moved slightly in the picture in the next frame. Since the object has already been encoded, a motion vector describing the objects motion can be encoded instead. By doing this, much compression can be attained. Also, sub-pixel ME can be used to improve the performance.

Motion vectors are determined by calculating the distortion between a block in the frame being encoded and the reference frame. Typically, a search range exists in the reference frame to find the best match motion vector. Because the cost of ME is related to both the ME residual and the motion vector, typically a Lagrangian cost function is used. The Lagrangian cost function is shown in Equation 3-1.

$$J\left(\overrightarrow{mv}, \lambda_{MOTION}\right) = D\left(s, c\left(\overrightarrow{mv}\right)\right) + \lambda_{MOTION} \bullet R\left(\overrightarrow{mv} - \overrightarrow{mv}_p\right) \qquad \textbf{(3-1)}$$

In Equation 3-1, mv_p is the prediction for the MV, and λ being the Lagrange multiplier. The symbol *mv* represents a motion vector with horizontal and vertical components. The distortion, *D*, is a function of original signal, *s*, and the coded signal, *c*. It is based on the sum-of-accumulated differences between the reference frame and the current frame and is given in Equation 3-2.

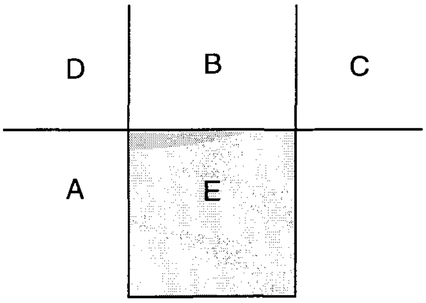
$$SAD(V_i) = \sum_{x=0}^{M}\sum_{y=0}^{N}\left|S_l(x,y) - S_{l-1}(x+dx, y+dy)\right| \qquad \textbf{(3-2)}$$

In Equation 3-1, the size of the motion vector is reduced by a parameter called mv_p. This parameter is a prediction of the motion vector and is described in the following section.

3-1 Median Prediction

To reduce the size of the encoded motion vectors, the motion vector of the block is first predicted. And instead of encoding the full motion vector, the difference between the full motion vector and the predicted motion vector is encoded. One such prediction which has proven to be very useful and effective is the median predictor. Referring to Figure 3-1, the motion vector of block E is estimated based on the spatially adjacent blocks A, B, and C.

Figure 3-1. Median Prediction Neighbors



The formula used in the median predictor is given in Equation 3-3.

$$\overrightarrow{MV} = median\left(\overrightarrow{MV\_A}, \overrightarrow{MV\_B}, \overrightarrow{MV\_C}\right) \qquad \textbf{(3-3)}$$

There are certain rules to follow if any of blocks A, B, or C do not exist. If A does not exist, the motion vector of A is assumed to be (0,0). If block C does not exist, the motion vector of block C is assigned to the motion vector of block D. If both blocks B and C do not exist, then the motion vector of A is used.

Armed with a very effective MV predictor that is used in all the motion estimation techniques in JM, the ME techniques are analyzed next.

3-2 Full Search

The full search is an exhaustive search of the search grid on the reference frame. If the search range is ± 16, then there are 1089 search points. Such an exhaustive search is computationally expensive but this algorithm yields the best R-D efficiency.

Some very simple optimizations have been done by JVT in the JM. The first is a form of early termination whereby if an intermediate SAD value is greater than a previously calculated SAD value, then the calculation can stop. A second optimization is useful in reducing the complexity when using variable block sizes and involves calculating SAD values of larger blocks by summing the SAD values of the smallest 4x4 blocks.

3-3 Unsymmetrical-Cross Multi-Hexagon-Grid Search

The Unsummetrical-Cross Multi-Hexagon-Grid Search (UMHexagonS) is a complete solution that involves MV prediction, search algorithms, and SAD prediction.

3-3.1 UMHexagonS MV Prediction

UMHexagonS adds three more MV predictors in addition to the median predictor. The first predictor added is the upper layer (UpLayer) predictor whereby the MV of a larger block size is used as the estimate of a smaller block size.

The second predictor added is the Corresponding-block predictor whereby the MV of the collocated block in the previous frame is used as an estimate. Finally, the third predictor added is the Neighboring Reference-Frame predictor whereby the multiple reference frame feature is taken advantage of.

3-3.2 SAD Prediction for Early Termination

In addition to MV predictors, the algorithm further adds an early termination technique based on SAD prediction. SAD prediction is very similar to MV prediction and thus includes the median predictor, uplayer predictor, corresponding-block predictor, and the neighboring reference-frame predictor. The SAD prediction is used as an early termination condition and is described in Equation 3-4 and Equation 3-5.

$$SAD_{thrh} = SAD_{pred}(1+\beta) \qquad \textbf{(3-4)}$$

$$SAD_{exit} \leq SAD_{thrh} \qquad \textbf{(3-5)}$$

The idea behind SAD prediction is that the best possible SAD value is predicted and if a SAD value is calculated to be close enough to the predicted SAD, it is assumed the most optimal SAD value and hence MV is found. Hence, during the search process, if a calculated SAD is less than the predicted SAD multiplied by (1+β), the whole search is terminated assuming the best MV has been found. It is clear that the selection of β affects the speed and the quality of the algorithm.

Armed with MV and SAD predictors, UMHexagonS implements a complex search pattern.

3-3.3 UMHexagonS Patterns

The UMHexagonS involves the following steps.

1. MV prediction as described in the previous sections
2. Unsymmetrical-cross search – A cross search is performed whereby there are more horizontal test points than there are vertical test points. Hence, the term unsymmetrical. This is done with the belief that horizontal movement is heavier than vertical movement. Also, the algorithm believes that the cross-search will effectively locate the area of minimum distortion. The minimum cost from this search is used as the search center for the next step.
3. Uneven Multi-Hexagon-grid search – In this step, a full search with range 2 is done about the search center to fine tune the MV. In case this traps the result in a local minimum or there is irregular motion, a growing 16 point hexagon search pattern is done. The 16 point hexagon is again weighted for horizontal motion.
4. Extended hexagon based search – This step is typically done if the big hexagon search is successful. This indicates that the search resulted in MV outside of search center. As such, the MV is further fine tuned with a small hexagon and the search is completed only when the minimum is found in the center of the hexagon.

3-4 Simplified Unsymmetrical Hexagon Search

As shown in the UMHexagonS, the search pattern is complex and contains many search points. The temporal MV predictors (corresponding block and neighboring reference frame predictors) are expensive to implement. Also, the calculation of β is a challenge and expensive as well. The simplified UMHexagonS serves to alleviate the listed deficiencies with UMHexagonS. This is accomplished in three ways. First, the simplified UMHexagonS removes the temporal predictors. Second, a faster sub-pixel ME is implemented. And third, a faster integer pixel ME is implemented by removing the local full search and by implementing additional early termination techniques.

The early termination techniques are based on convergence/intensive conditions. Convergence condition indicates global minima. As a result of meeting the convergence condition, the cross and big hexagon searches are no longer needed. Also, an intensive condition attempts to avoid local minima.

It turns out that with this simplified model, a bit rate savings of up to 18% was seen at little or no cost to quality. Also, the ME time was reduced by as much as 55% compared to UMHexagonS.

3-5 Enhanced Predictive Zonal Search

The Enhanced Predictive Zonal Search (EPZS) is primarily based on effective methods of predicting the MV. Several predictors are used and classified under predictor sets. The following are the predictor sets.

- Set 1: Median predictor
- Set 2: MV of previous frame (collocated lock), spatially adjacent blocks (used in median predictor), and (0,0)
- Set 3: Accelerator motion vector (Calculated based on previous 2 frames) and adjacent blocks in previous frame

EPZS involves first testing the first predictor set and terminates the test based on a threshold T_1 that is set to number of pixels in current block. If this test fails, the other predictor sets are checked and early terminated against an adaptive early termination threshold T_2 shown in Equation 3-5 is used.

$$T_2 = a \bullet \min(MinJ_1, MinJ_2, ...MinJ_n) + b \tag{3-5}$$

In Equation 3-5, *a* and *b* are fixed values and MinJi are minimum distortion values calculated in the search. In order to maintain stability, the following term in Equation 3-6 is added to prevent against inadequate and incorrect early termination.

$$MinJ_i = 3 \bullet Np \tag{3-6}$$

In Equation 3-6, Np is defined to be the number of pixels in the frame.

Finally, EPZS employs simple search patterns to fine tune the search. Namely, it uses the diamond search, square search, and the Extended EPZS pattern.

CHAPTER 4
FLEXIBLE TRIANGLE SEARCH

The search shape in the Flexible Triangle Search (FTS) is the triangle [4]. It is an interesting shape in that there are three test points (triangle vertices) compared to 4 for diamond. Immediately, it seems like the search will result in less test points, so a natural question is how effective this method is. In the FTS algorithm, the triangle is quickly moved from areas of high error to areas of low error by performing certain operations. Small to large triangles are used to allow for fine to coarse movements. Expansion is used to move the triangle quickly away from areas of high error and contraction is used to fine tune a search.

FTS is in fact based on the simplex algorithm for ME but the simplex algorithm is used in the continuous domain. As such, to use the simplex algorithm in an integer search is difficult since estimates are needed to map the continuous domain into the integer domain. It is shown [4] that this may result in the collapse of the triangle into one or two vertices. In addition, floating-point calculations are used and are very computationally expensive. FTS allows the simplex algorithm to be used in an integer grid by defining a finite set of triangles to perform the search. The vertices of these triangles will always lie on the integer grid. Certain operations can be performed on the triangle including reflection, translation, contraction, and expansion. Since the triangles are typically predetermined, the operations are easily performed using look-up tables.

The operations that are performed on the triangle are as follows:

- Reflection – reflecting away the vertex with the highest cost about the other two vertices. If the new vertex has lower cost, then the reflection is successful.
- Expansion – increase the size of the triangle by increasing the level. The purpose of the expansion is to move a particular vertex further in the particular direction of lower cost.
- Contraction – When reflection fails, it is expected that the triangle is in an area of lowest cost (hopefully the global minima). As such, contraction is used by reducing levels to fine tune the MV.
- Translation – On a successful expansion, it may seem the area of lowest cost is further in the direction of the expansion. Hence, translation is used to move the whole triangle in the general direction.

Figure 4-1 and Figure 4-2 depicts some of the valid operations on the triangle. The smallest triangle that is 1 pixel by 1 pixel in size is assigned level 0. The triangles in each level represent the possible reflections of the triangles in the level. Translation is not shown since it is simply a shift of the whole triangle. A triangle is defined by an identifying number and its level. For example, a T24 triangle is the fourth triangle in level 2. The vertices of the triangle are denoted V_O, V_A, and V_B where V_O is the origin of the triangle, V_A is the vertex counterclockwise from V_O and V_B is the last vertex.

Figure 4-1. FTS Triangles and Reflections

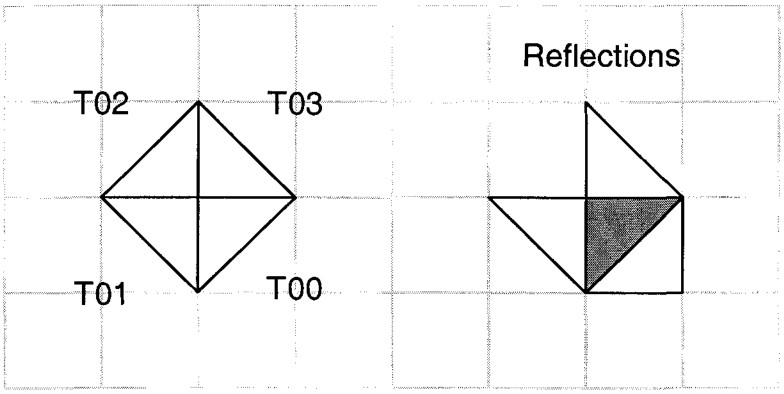
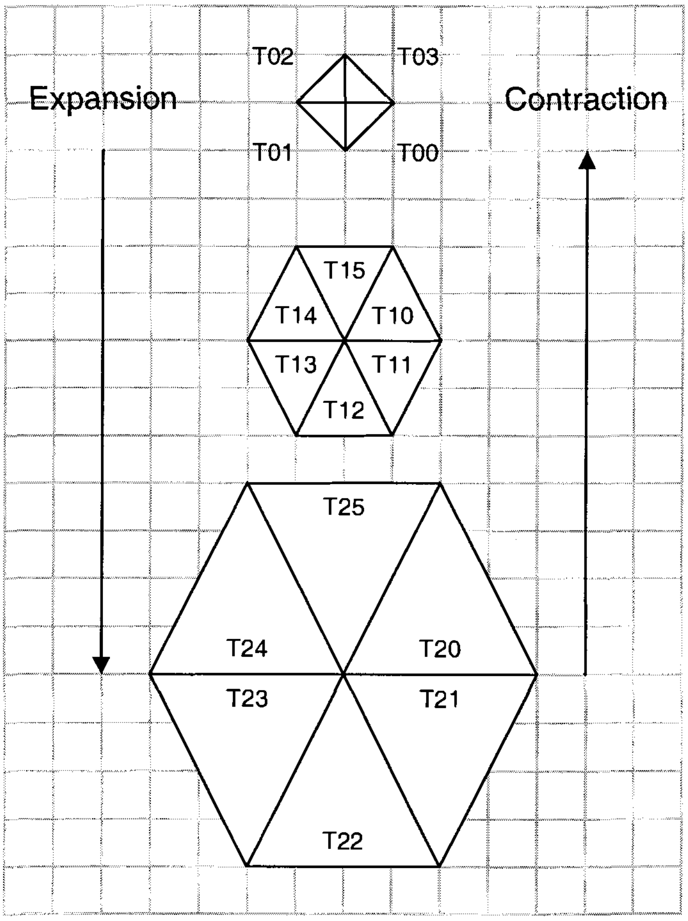


Figure 4-2. FTS Triangles Levels 0 through 2



Note that in Figure 4-2, three levels of triangles are used. More levels can be added but simulations showed [4] that 3 levels are sufficient. Typically, predetermined triangles are used so that it can be easily referenced via tables in software. An example of a level 0 lookup table around the V_O vertex is included in Table 4-1.

Table 4-1: FTS Level 0 Look-up Table

Current Triangle	V_O Reflection		V_O Reflection	
	New Triangle	Origin Shift	New Triangle	Test Point
T00	T02	(1,1)	T14	(2,2)
T01	T03	(-1,1)	T10	(-2,2)
T02	T00	(-1,-1)	T11	(-2,-2)
T03	T01	(1,-1)	T13	(2,-2)

The following is a detailed step-by-step FTS algorithm.

1. Initialization

   - Initialize the triangle to level 0 and initialize the vertices V_0, V_A, and V_B with V_0 chosen as the initial search point generated by MV prediction.

   - Initialize K to 0 and a translation vector V_d to 0.

     Also initialize V_{min} to V_0.

2. Determine costs

   - Calculate the cost using the Lagrangian cost function of the three vertices. Assign the most expensive vertex as V_h and the least expensive as V_l.

   - If this step is reached after a successful expansion or translation, go to step 6. Otherwise, go to step 3.

3. Reflection

- Reflect the triangle away from vertex with largest cost (V_h) and hence obtain a new vertex Vr. Calculate the cost of the new vertex V_r.
- If the new vertex results in a smaller cost, the reflection is successful. And if successful, go to step 4. Otherwise, if the reflection is unsuccessful, proceed to step 5.

4. Expansion
   - Locate an expansion vertex V_e based on the appropriate table for the current level and calculate the cost of V_e.
   - If the cost of V_e is less than the cost of V_r, then expansion was successful. If successful, increase the triangle level and calculate the translation vector to be $V_d = V_e - V_r$.
   - If expansion is not successful, replace V_h by V_r.
   - Update V_{min} if necessary.
   - Go back to step 2 after updating K = K + 1.
5. Contraction
   - Reduce triangle level for fine tuning and go back to step 2 after updating K = K + 1.
6. Translation
   - Test a new vertex V_t by translating V_l by V_d (i.e $V_t = V_l + V_d$).
   - If the cost of Vt is less than Vl, then translation was successful. Hence, replace V_l by V_t and update V_{min} if necessary.

- Go back to step 2 after updating K = K + 1.

The exit conditions of the algorithm are the following.

1. No more contractions are possible.
2. Search iterations reached a limit *KMax.*
3. If the calculated cost is less than a predetermined exit SAD. The exit SAD condition could be similar to that used in UMHexagonS.

Note that via simulations, it was determined that KMax of 8 is sufficient and that any greater value yield negligible to no return on quality. Unfortunately, there is no clear method to determine KMax except by trial and error. KMax can be a function of the search window and the value of 8 is determined with a search window of ±16.

4-1 Full-Pixel FTS

We first implement the FTS algorithm in the H.264 JM reference software to work in the integer grid or as a full-pixel ME algorithm. It is compared against the search algorithms already available in JM but with their respective sub-pixel refinements disabled. The following are the parameters used for these tests.

- QCIF
- CABAC
- Only 1st frame is I-Frame and no B-Frames
- 100 encoded frames
- 1 reference frame (no multiple reference frames)

- No sub-pixel ME
- Search range of +/-16
- Quantization Parameters of 8, 18, 28, and 38
- Variable size macroblocks is not supported. Only 16x16 macroblocks are used in motion estimation.

Detailed analysis of the Foreman and Carphone video sequences are done but results are obtained for many of the available video sequences. Analysis of the Foreman sequences are found in the subsequent sections and analysis of the Carphone sequence can be found in Appendix A.

4-2 Full-Pixel FTS Simulation Results

In evaluating the performance of FTS, we compare both the PSNRs of the reconstructed video sequences and the complexities of different ME algorithms. Typically PSNR is observed as a function of bit rate which produce the rate-distortion (R-D) curve. Such a graph indicates the performance of the video encoder. In other words, the graph shows the PSNR achievable by any search algorithm at any particular bit rate. This can be very important since many applications are bandwidth constrained and ideally the search algorithm exhibiting the best PSNR for the available bandwidth is chosen. Equation 4-1 outlines the calculation of PSNR.

$$PSNR_{dB} = 10 \bullet \log_{10}\left(\frac{(2^n - 1)^2}{MSE}\right) \qquad \textbf{(4-1)}$$

MSE in Equation 4-1 is the mean-squared-error which is the mean square of the difference between the reference frame and the degraded frame and *n* is the number of bits used to represent a video sample. Note that although PSNR allows for an automated and consistent method of evaluating quality, it may not represent subjective quality.

Figure 4-3 and Figure 4-4 show graphs of the luma PSNR vs. bit rate or the R-D curve for the Foreman video sequence.

Figure 4-3. Foreman Luma R-D Curve

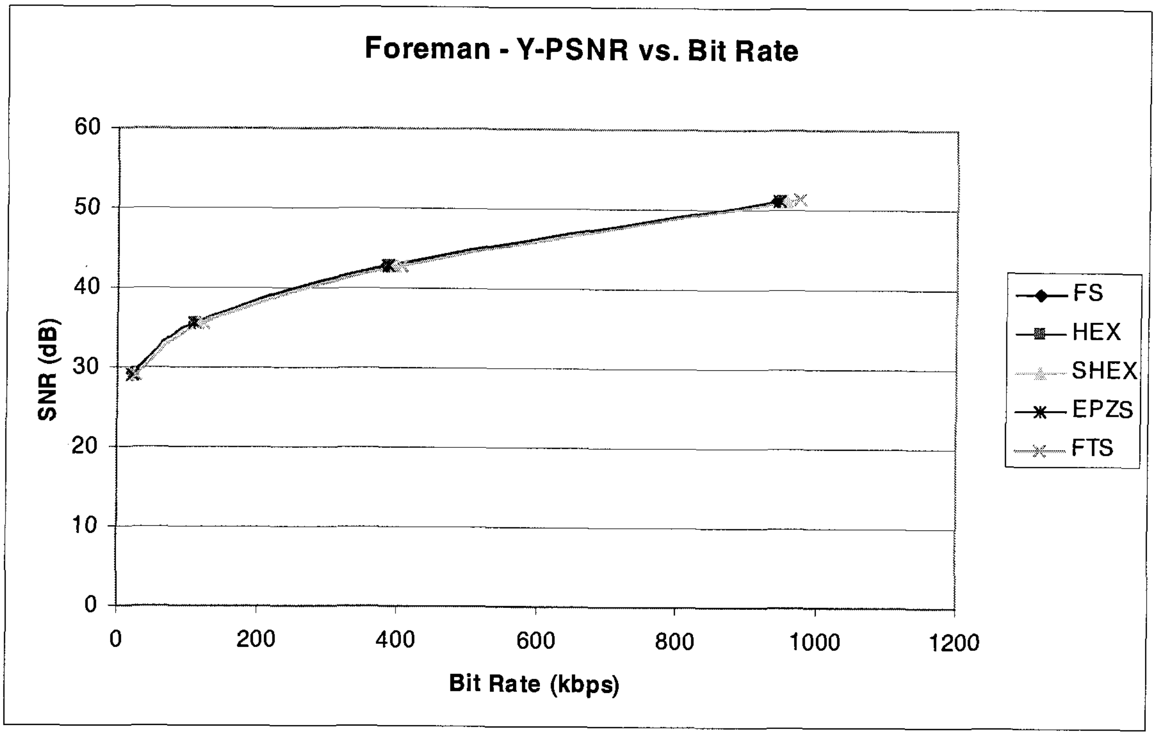
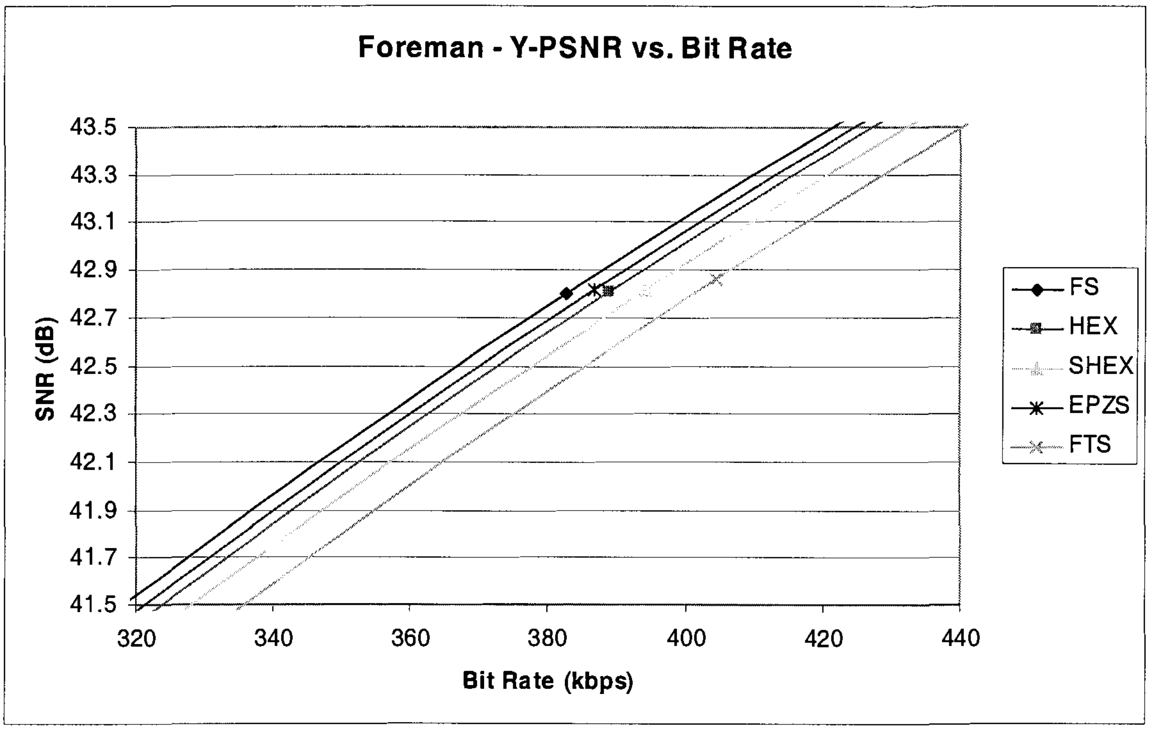


Figure 4-4. Foreman Luma R-D Curve (Close-Up)



It can be seen that the Full Search (FS) algorithm exhibits the best PSNR at any particular bit rate and is known as the optimum. Additionally, the performance of all the search algorithms is very close to that of FS. Upon closer inspection, FTS does exhibit the poorest performance compared to the other search algorithm. Specifically, FTS is 0.3 dB worst than FS but only 0.1 dB worst than simplified UMHexagonS. In other words, at any particular bit-rate, FTS is 0.1 dB to 0.3 dB worst than the other search algorithms. These figures are quite insignificant and if there are benefits elsewhere, it is an acceptable trade-off. Figure 4-5 and Figure 4-6 show the chroma U and chroma V R-D curves of the search algorithms.

Figure 4-5. Foreman Chroma U R-D Curve

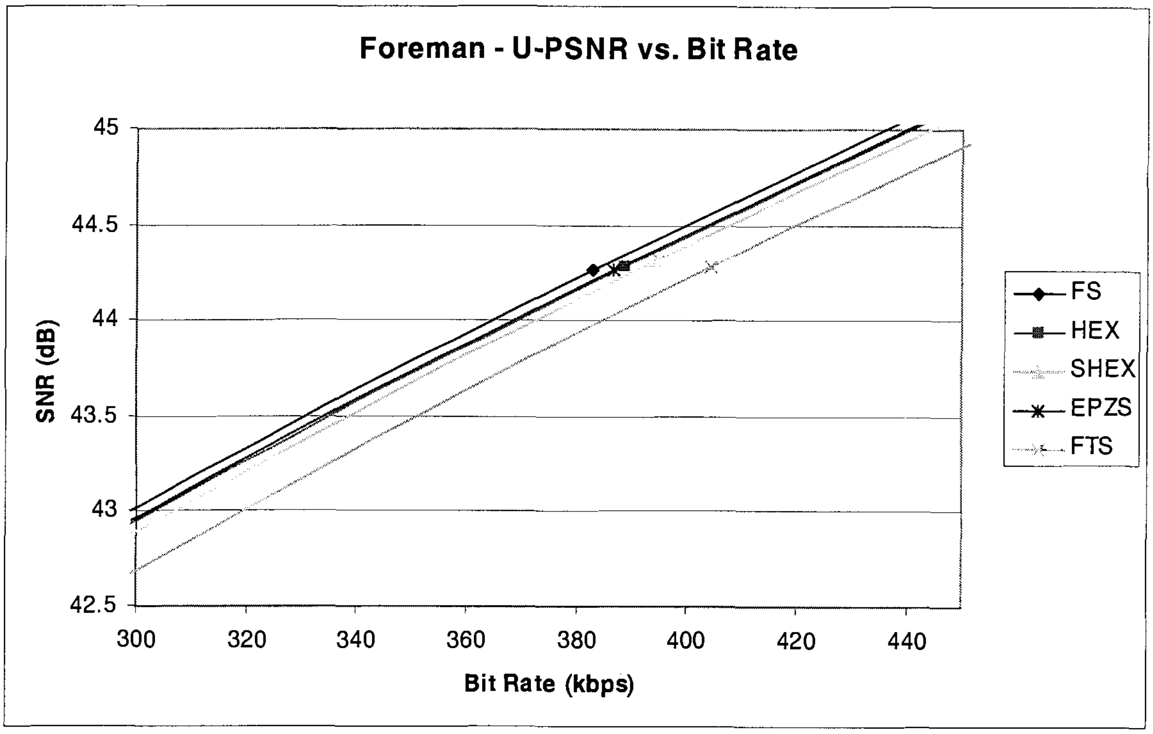


Figure 4-6. Foreman Chroma V R-D Curve

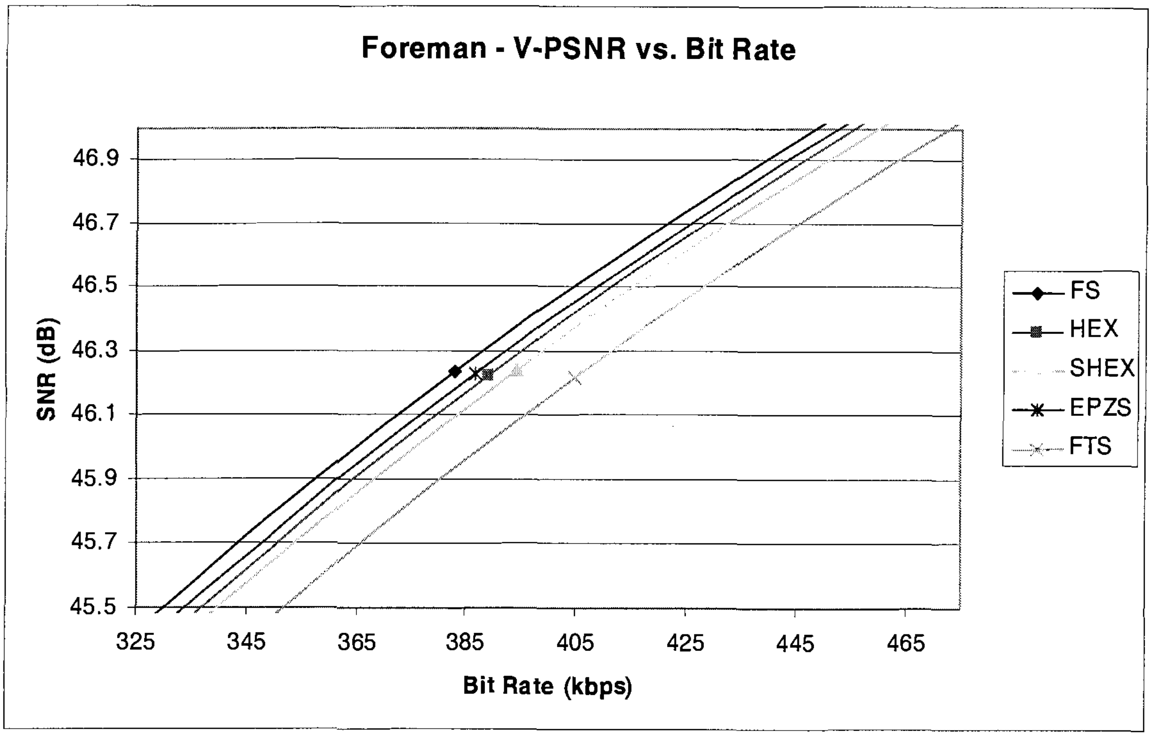


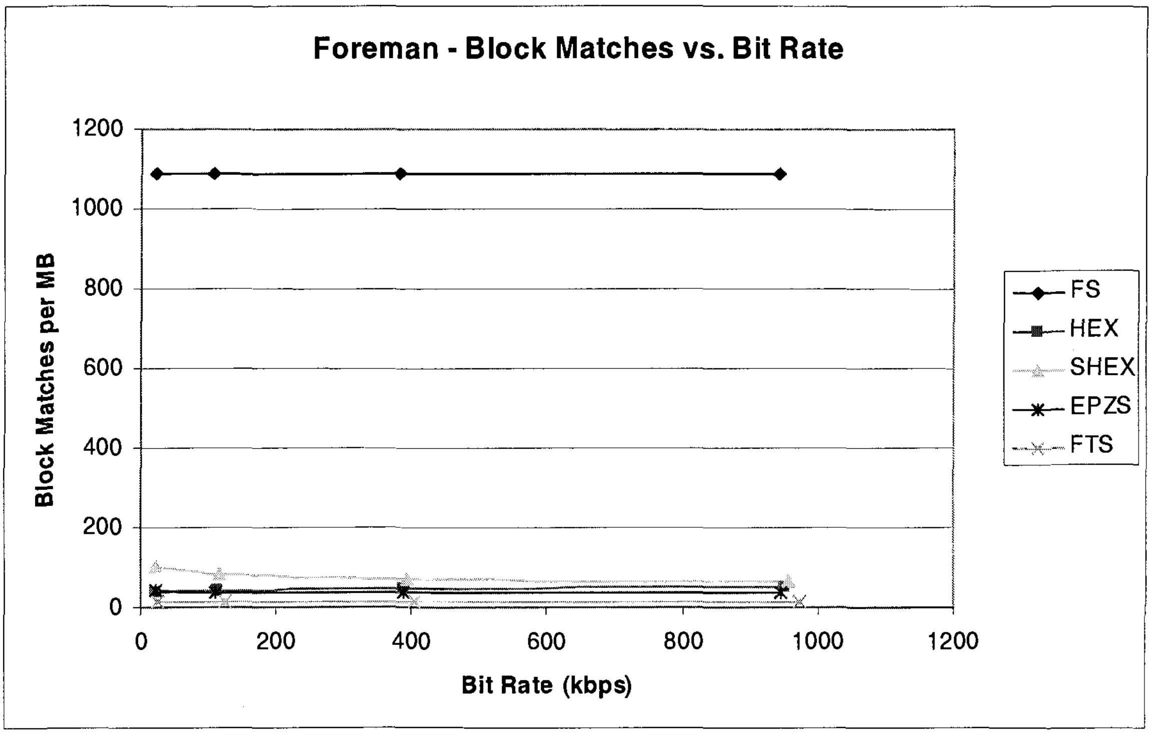
Figure 4-5 and Figure 4-6 show that the chroma R-D performance is comparable to that of the luma R-D performance.

In addition to observing the R-D performance of a search algorithm, the complexity must be looked at as well. If complexity is not an issue, the full search algorithm can be used yielding the best R-D performance. Unfortunately, the complexity of the FS algorithm is high for real world applications. Therefore, in choosing a search algorithm, one must look at its complexity as well as its R-D curve.

As mentioned, motion estimation is performed using the SAD operation. And since the SAD operation is performed for every pixel at every search position for the entire macroblock, the number of SAD operations used is a good indication of complexity. For example, to compute the SAD value at one search location requires 16x16 = 256 SAD operations. A SAD operation may further expand to one add, one subtract, and one absolute difference operations. Thus, the total number of operations required is 3x256 = 768. It should be clear that the most complex algorithm is full search since it performs an exhaustive search at all search locations. Another method of measuring complexity is the number of search positions or block matches that are performed. A block match is defined as the calculation of a SAD value between a macroblock and the reference frame. At first thought, the number of block matches may simply be the number of SAD operations divided by 256 but this is not necessarily true. It may be possible that less than 256 operations are required if early termination techniques are available.

Starting with block matches, Figure 4-7 show the number of block matches required for each of the search algorithms.

Figure 4-7. Foreman Block Matches Required



As suspected, the full search algorithm requires the most block matches. Specifically in a search area of ±16, 33x33 = 1089 block match operations are required. Figure 4-8 show the block matches required for the search algorithms ignoring the complex FS algorithm.

Figure 4-8. Foreman Block Matches Required Ignoring FS

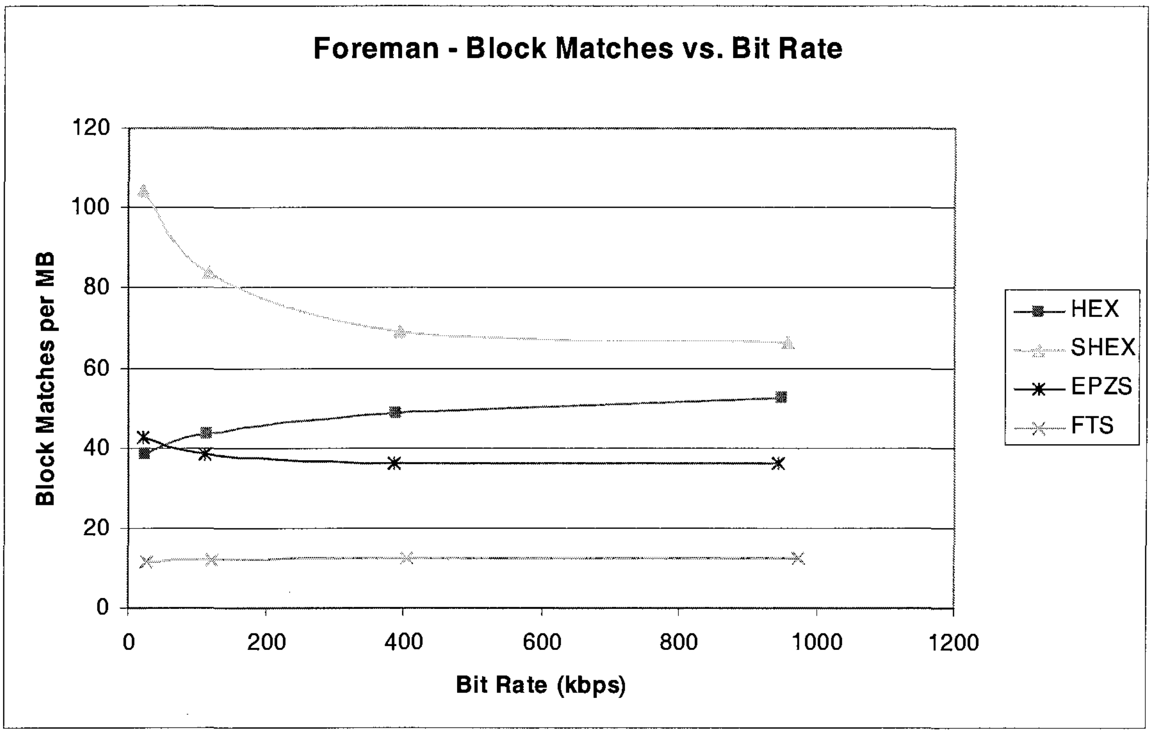


Figure 4-8 show that the EPZS algorithm performed the best of the available search algorithms in the JM software. However, the FTS algorithm was able to consistently use 65% fewer blocks matches than EPZS. This complexity reduction is the highlight of the FTS algorithm. The 0.1 dB to 0.3 dB reduction in R-D performance is a good trade-off for a 65% reduction in complexity.

Although FTS performed well for the Foreman sequence, it must be verified against other sequences to ensure FTS performs well with varying scenarios in the video sequence. Table 4-2 show the block matches required for other available video test sequences.

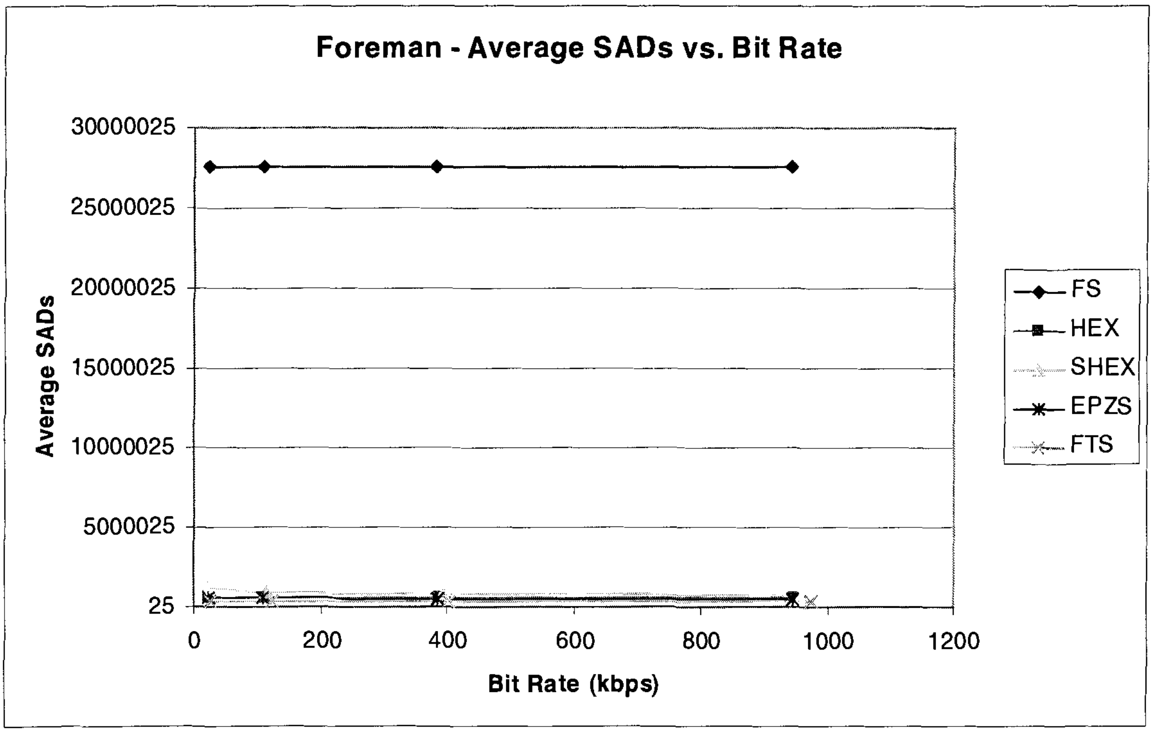
Table 4-2: Average Number of Block Match Operations per Macroblock

Test Sequence	HEX	SHEX	EPZS	FTS	FTS Savings over EPZS
akiyo	14.86	22.66	13.89	6.85	50.69%
news	25.11	39.40	22.45	7.14	68.20%
silent	27.23	45.29	24.44	8.92	63.51%
coastguard	40.86	103.47	38.07	8.33	78.11%
foreman	43.66	83.96	38.23	12.14	68.24%
carphone	36.88	63.58	31.83	10.32	67.57%
car	47.56	93.96	47.19	15.99	66.12%
claire	16.75	21.75	14.87	7.75	47.89%
miss america	15.79	22.23	15.89	9.24	41.83%
Average	**29.85**	**55.14**	**27.43**	**9.63**	**61.35%**

Table 4-2 show that the FTS sequence does indeed perform on average 61% better than the EPZS algorithm. Specifically, FTS can be 42% to 78% better than EPZS depending on the video sequence.

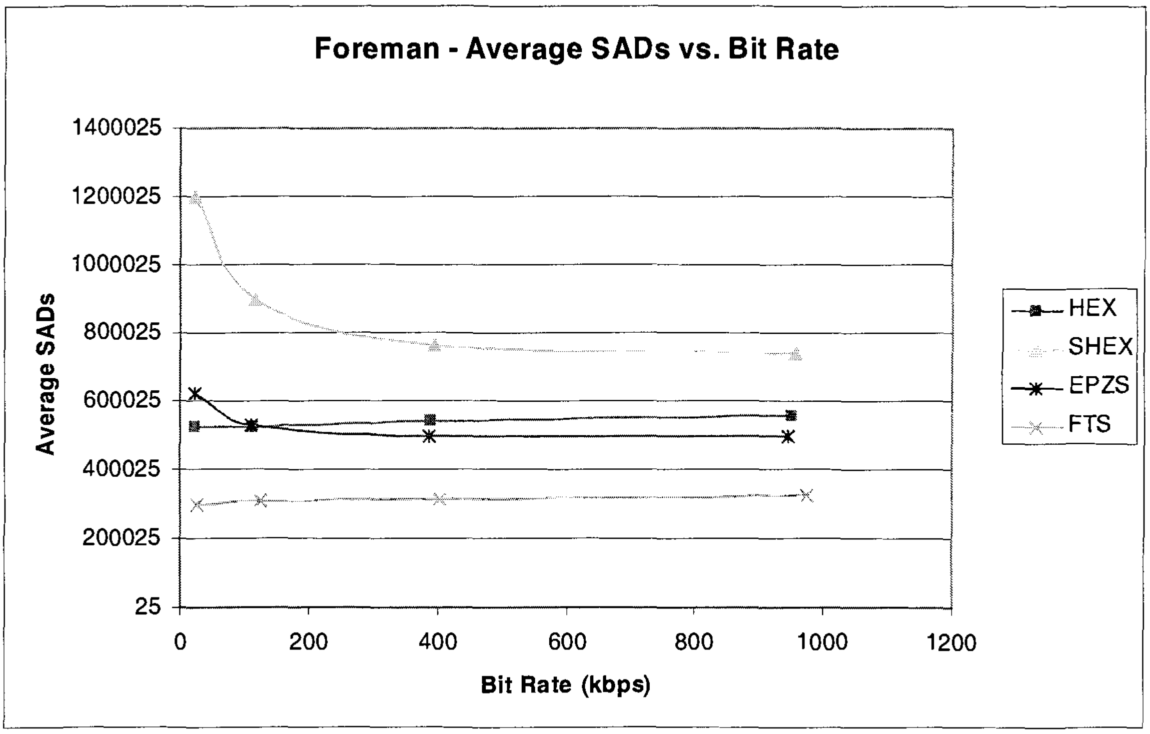
Observing block matches has its deficiencies since a block match operation may require a variable number of SAD operations. Some algorithms have early termination techniques such that not all 256 SAD operations need to be performed on a 16x16 macroblock. For example, if an intermediate SAD value is greater than a known minimum SAD value, the calculation can stop. Hence, measuring the number of SAD operations is a better indication of complexity. Figure 4-9 show the average number of SAD operations required for each search algorithm.

Figure 4-9. Foreman Average SAD Operations



Again, notice that FS is the most complex of the search algorithms. For a clearer picture, Figure 4-10 show the average number of SAD operations without FS.

Figure 4-10. Foreman Average SAD Operations ignoring FS



Again, EPZS performed the best but FTS is able to perform 35% to 52% better. Notice here that although FTS performed better among the H.264 ME implementations, it did not see the same percentage savings as seen with block matches. This would suggest that EPZS performs other early termination techniques during SAD calculations. In fact, in the FTS algorithm, the optimization allowing for early termination of the SAD calculation cannot be implemented. Finally, Table 4-3 show the average number of SAD operations for other video sequences.

Table 4-3: Average Number of SAD Operations per Macroblock

Test Sequence	HEX	SHEX	EPZS	FTS	FTS Savings over EPZS
Akiyo	125237	187605	180695	173710	3.87%
News	274621	401760	314535	181147	42.41%
Silent	313126	488540	356599	226272	36.55%
Coastguard	505120	1177764	529383	211202	60.10%
Foreman	525808	895676	528774	307799	41.79%
Carphone	421371	663875	426066	261696	38.58%
Car	761958	1325480	805052	405309	49.65%
Claire	187968	218486	185226	196439	-6.05%
miss America	200902	252597	206848	234434	-13.34%
Average	**368457**	**623531**	**392575**	**244223**	**28.17%**

Although FTS performed worst in terms of the average SAD operations for a couple of the video sequences, it still performed 28% better on average than the other search algorithms.

In addition to average SAD operations, it is important to take note of peak resource requirements. Hence, the peak number of SAD operations used is observed. Figure 4-11 and Table 4-4 show the maximum SAD operations.

Figure 4-11. Foreman Maximum SAD Operations

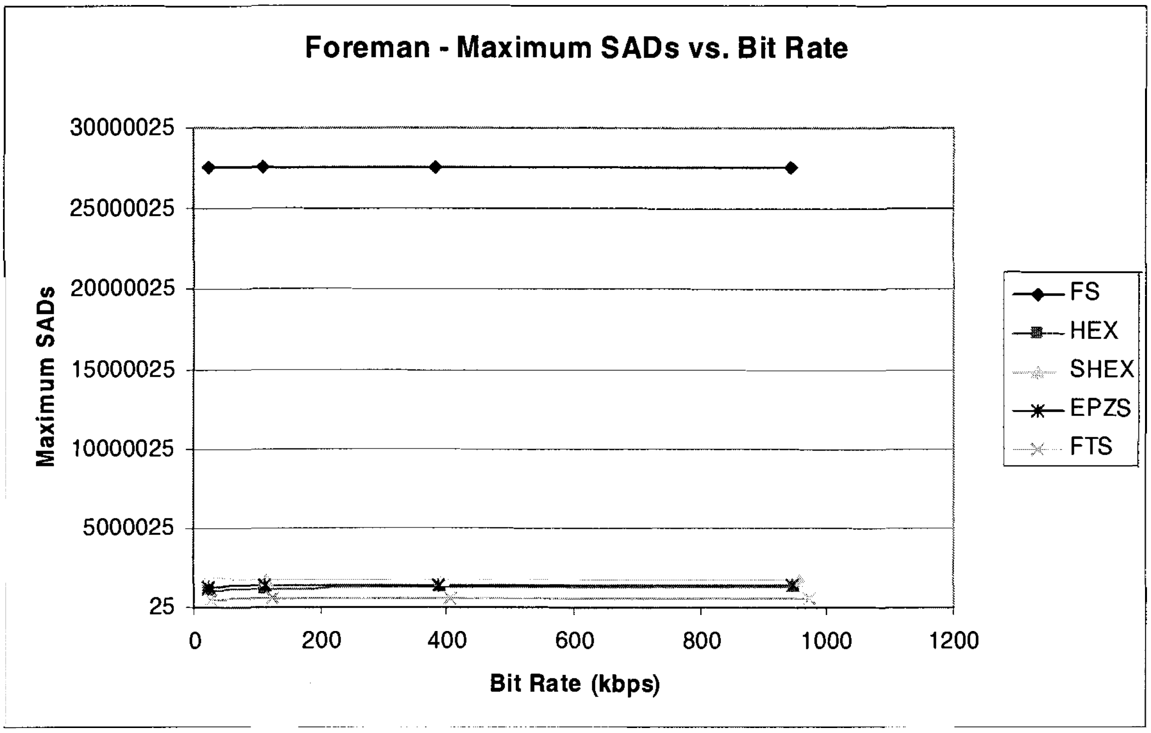


Table 4-4: Maximum Number of SAD Operations per Macroblock

Test Sequence	HEX	SHEX	EPZS	FTS	FTS Savings over EPZS
Akiyo	189280	298800	228384	201216	11.90%
News	498544	777040	547104	228608	58.21%
Silent	581648	886960	601392	289536	51.86%
Coastguard	845168	1644512	799824	352000	55.99%
Foreman	1172032	1776496	1385040	606976	56.18%
Carphone	756864	1223488	732688	385792	47.35%
Car	1407312	2046688	1638608	622336	62.02%
Claire	264864	367712	260512	251904	3.30%
miss America	252944	389088	282432	295168	-4.51%
Average	**663184**	**1045643**	**719554**	**359282**	**38.03%**

Peak resource usage in terms of maximum number of SAD operations is reduced by using FTS. In fact, FTS is on average 38% better than other search algorithms.

The FTS algorithm has shown to suffer insignificantly in terms of its R-D performance. But the FTS algorithm has showed to save quite significantly in complexity as shown by measuring block matches, average SAD operations, and maximum SAD operations. This reduced complexity makes FTS attractive over other search algorithms.

4-3 Enhanced and Predictive FTS

In Enhanced Flexible Triangle Search (EFTS), the FTS algorithm is further reduced in complexity. Compared to the FTS algorithm, EFTS requires less SAD calculations and less block matches. In addition, EFTS offers no degradation in video quality compared to FTS.

During the reflection, expansion, contraction, and translation operations, the vertices of the triangle may lie on a particular search position more than once. If this occurs, the FTS algorithm is doing unnecessary work recalculating the SAD value for that position. It is more efficient to use the SAD value calculated when the position was originally searched and this forms the enhancement to FTS. This enhancement minimizes the number of SAD operations and block matches at the expense of data memory to store previously calculated results. Table 4-5 shows the number of unnecessary block match operations for the foreman and car phone video sequences.

Table 4-5: Average Number of Duplicate Block Match Operations per Macroblock

Sequence	Quantization Parameter	Average Duplicated Block Match Operations per MB
Foreman	8	4.45
	18	4.36
	28	4.26
	38	4.14
Car Phone	8	3.82
	18	3.74
	28	3.70
	38	3.64

As shown in Table 4-5, each macroblock in the Foreman sequence performs about 4.45 unnecessary block match operations. Since the FTS algorithm requires 12.7 block match operations on average in the Forman sequence, 35% of the operations are redundant and can be eliminated. In other words, this enhancement alone can save 35% of FTS complexity making it an even more attractive ME technique.

The modification to the FTS algorithm involves creating a data buffer to store SAD values. Once a SAD value is calculated by FTS at a particular search position, the resultant value is stored. When a new SAD value needs to be calculated, the buffer is checked first to see if it has been previously calculated. If it has, the value from the buffer is used and the additional SAD operations are eliminated. The amount of computation required to perform these checks are significantly less than the number of computations required to perform a SAD operation. Specifically, a SAD operation on one macroblock will require 16x16x3 or 768 subtract, add, and absolute value operations since a SAD operation requires one add, subtract, and absolute operations. In the proposed solution, only two operations are needed to check if the SAD values have already been calculated and to load or store the SAD value.

In predictive FTS (PFTS), a technique is introduced to predict the initial search direction of FTS. Note that in PFTS, the enhancement in EFTS is included. It is shown [7] that by choosing the initial triangle correctly, the number of block matches and SAD operations are reduced. This occurs since the search is directed toward the direction of the minimum earlier. In addition, it is shown that this predictive algorithm will not affect the PSNR performance of FTS.

PFTS chooses a search direction by choosing one of the four triangles found in level 0 the search shall start with. Refer to Figure 4-2 for the four triangles available in level 0. Each triangle represents a search in the direction of the quadrant it lives in. In predicting the starting triangle, the SAD values of the four positions surrounding the search centre which lie as a vertex to a triangle is evaluated. The two vertices of each triangle ignoring the origin are then added together. Finally, the triangle with the two vertices that add to the smallest value is chosen as the starting triangle and search direction.

4-4 Enhanced and Predictive FTS Simulation Results

In the analysis of EFTS and PFTS, simulations are done with the EFTS enhancement by itself and then with EFTS and PFTS together. From this point forward, PFTS implies EFTS is included. Similar to the previous analysis of the FTS algorithm, the R-D performance of EFTS and PFTS are observed. Figure 4-12 show the luma R-D performance of EFTS and PFTS.

Figure 4-12. Foreman Luma R-D Curve Using EFTS and PFTS

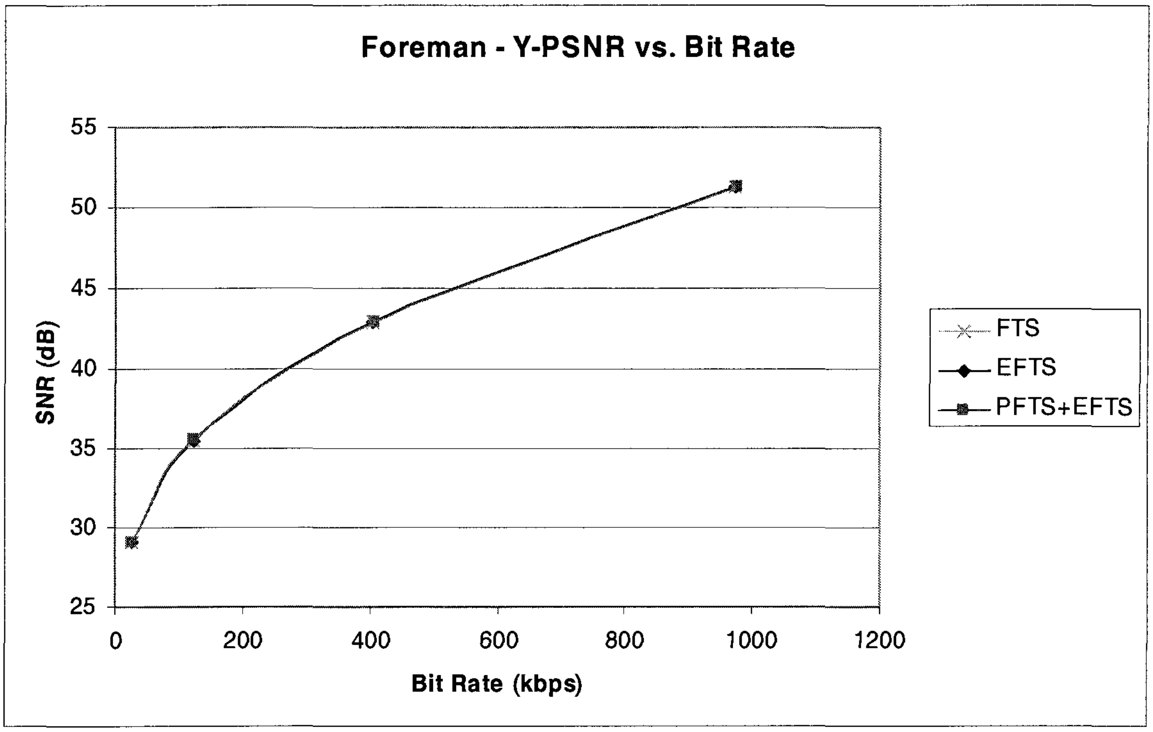


Figure 4-12 show that EFTS and PFTS exhibit virtually the same R-D performance. In fact, it is expected that EFTS perform the same as FTS since the only change is to store calculated SAD values for later use and so the actual performance remains unchanged. In addition, no significant R-D performance change was seen when PFTS is added to EFTS. Figure 4-13 and Figure 4-14 show the chroma R-D performance of EFTS and PFTS, which show similar results as the luma frame.

Figure 4-13. Foreman Chroma U R-D Curve Using EFTS and PFTS

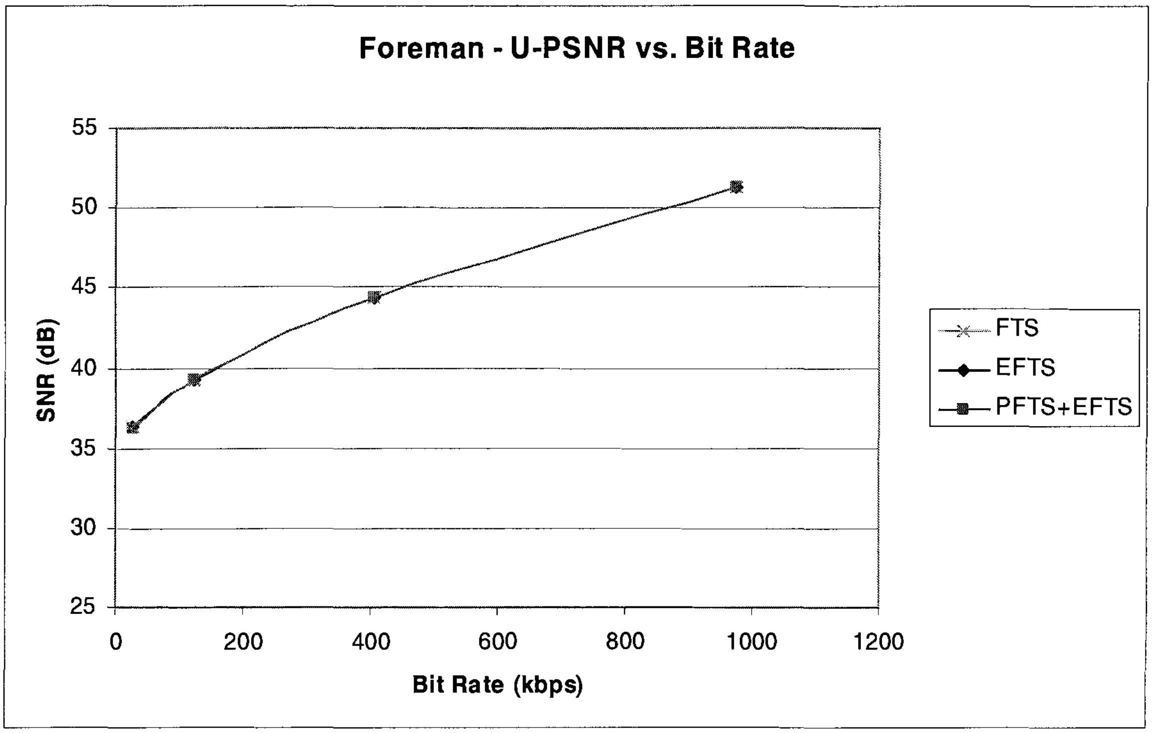
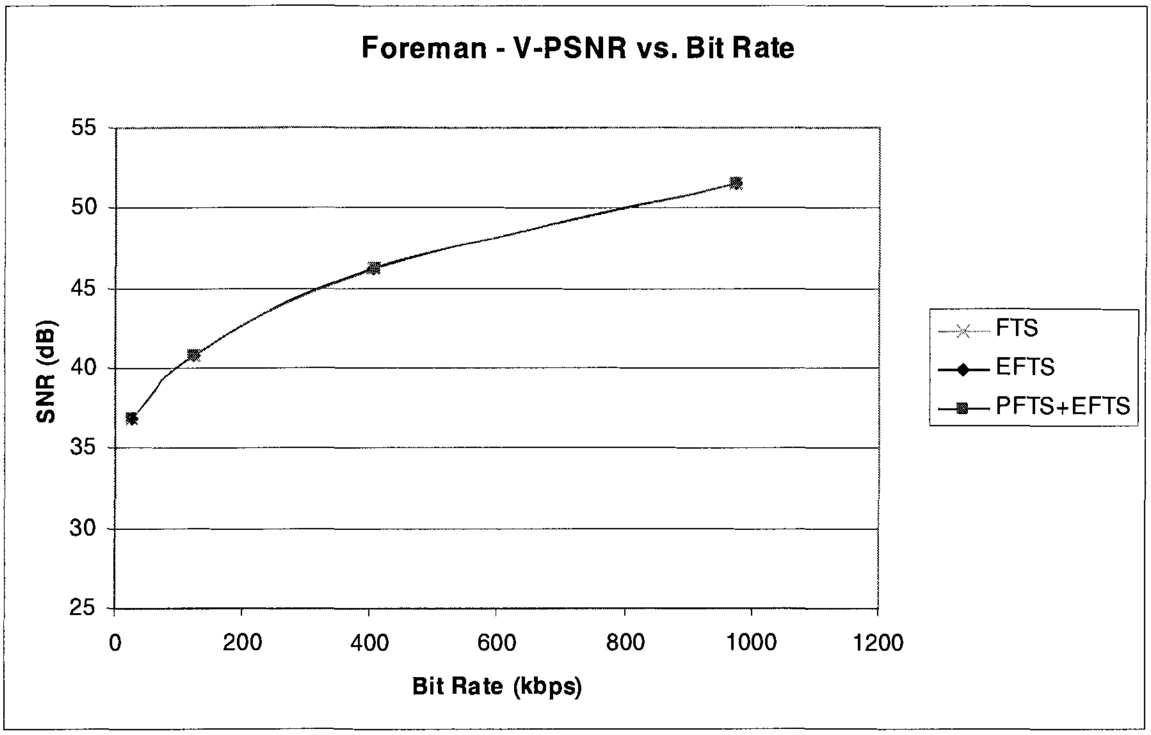
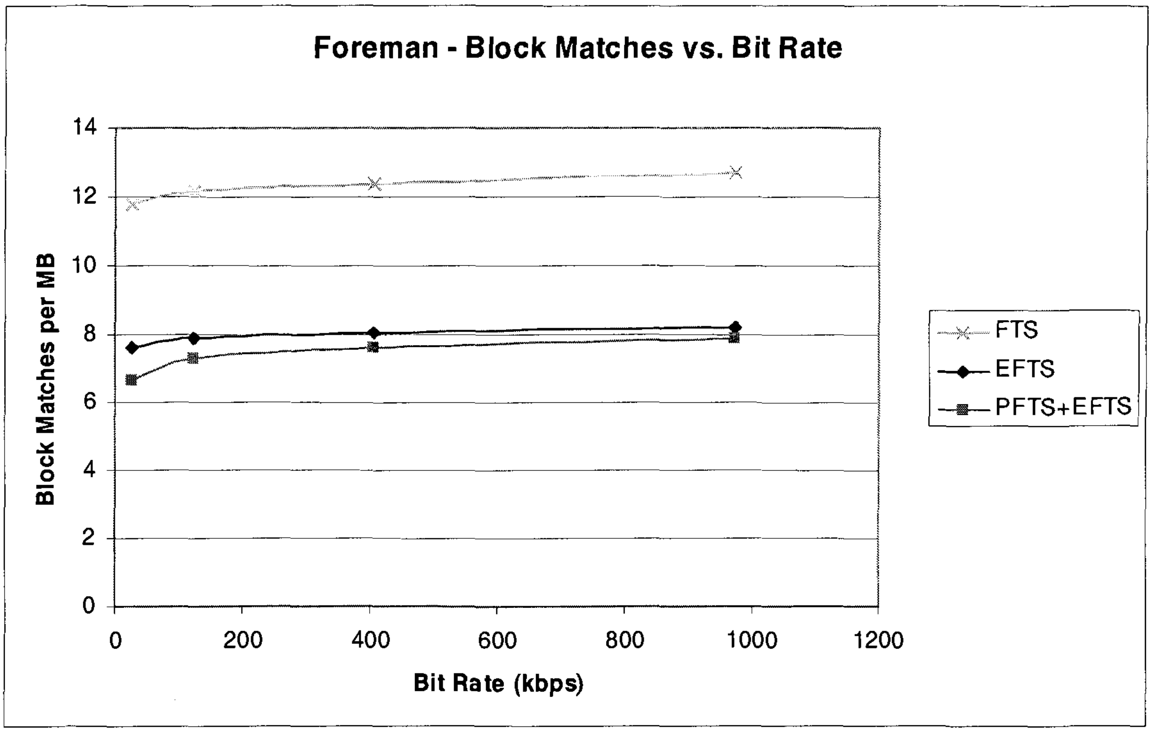


Figure 4-14. Foreman Chroma V R-D Curve Using EFTS and PFTS



The block matches is observed to show the affect of EFTS and PFTS on complexity. Figure 4-15 show the block matches of the EFTS and PFTS algorithms in comparison to the original FTS algorithm.

Figure 4-15. Foreman Block Matches Using EFTS and PFTS



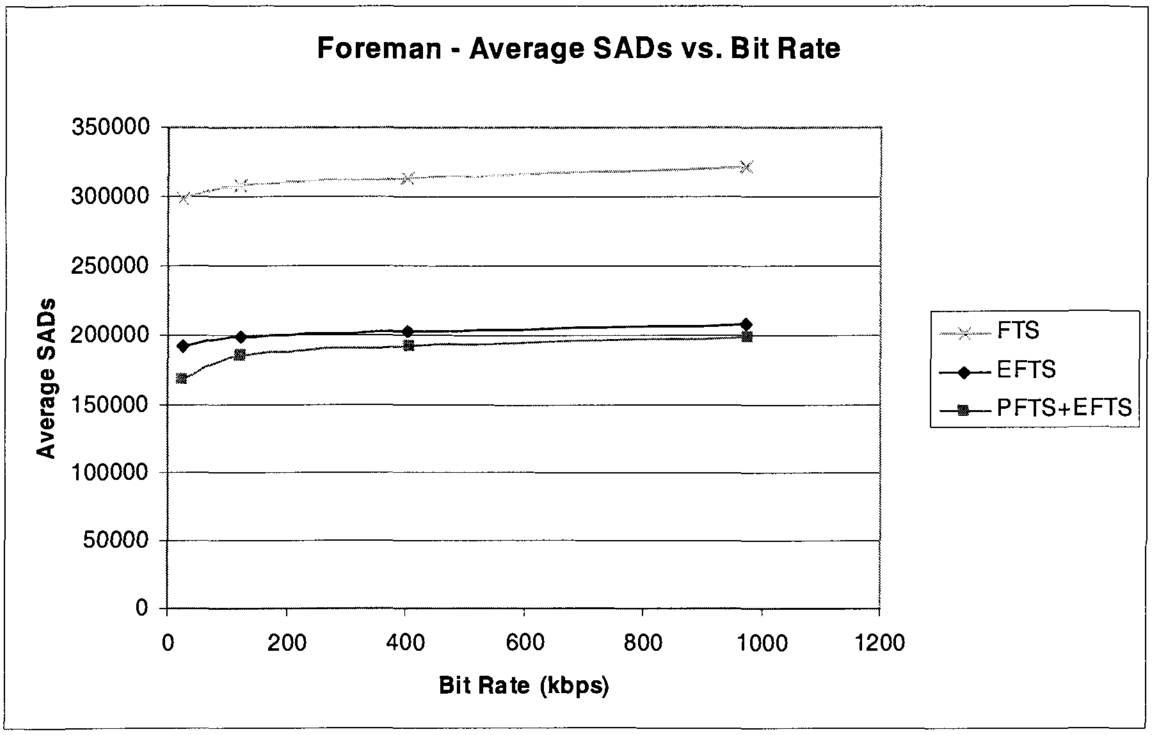
For the Foreman video sequence, Figure 4-15 show that EFTS exhibits an improvement of 35% over FTS which is expected and shown earlier. In addition, by adding the PFTS algorithm to FTS, another 5% to 13% savings can be obtained. Table 4-6 shows the effect of PFTS and EFTS over FTS for all the available video sequences.

Table 4-6: Average Number of Block Match Operations per Macroblock Using EFTS and PFTS

Test Sequence	FTS	EFTS	EFTS Savings over FTS	EFTS + PFTS	PFTS Savings over EFTS
akiyo			31.27%	5.02	-6.65%
			28.43%	5.36	-4.94%
silent	8.92	5.82	34.77%	5.80	0.35%
coastguard	8.33	5.61	32.73%	5.41	3.42%
foreman	12.14	7.87	35.19%	7.32	6.93%
carphone	10.32	6.62	35.91%	6.07	8.24%
car	15.99	10.70	33.10%	10.02	6.33%
claire	7.75	5.32	31.29%	5.19	2.47%
miss america	9.24	5.76	37.70%	5.15	10.53%
Average	**9.63**	**6.39**	**33.38%**	**6.15**	**2.96%**

Table 4-6 show that the EFTS algorithm is capable of achieving a 33% savings on average over the FTS algorithm. In addition, the PFTS algorithm adds another 3% on average of savings on top of the 33%. Notice that PFTS did perform slightly worse for two of the sequences. Keeping in mind that these savings are on top of the 61% savings FTS saw on top of the other search algorithms, EFTS and PFTS are very powerful. Figure 4-16 show the average SAD operations required for EFTS and PFTS in comparison to FTS.

Figure 4-16. Foreman Average SAD Operations Using EFTS and PFTS



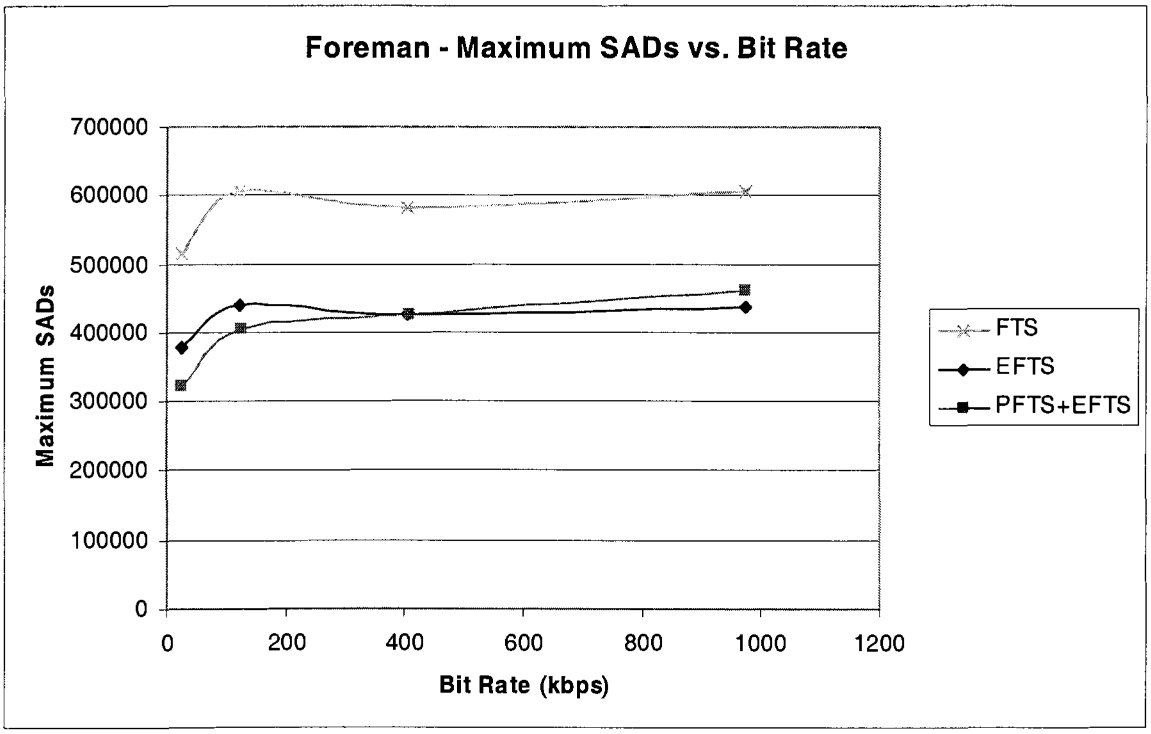
Again, EFTS performed 35% better than FTS and PFTS performed an additional 4% to 13% better. Table 4-7 show the results of other video sequences.

Table 4-7: **Average Number of SAD Operations per Macroblock Using EFTS and PFTS**

Test Sequence	FTS	EFTS	EFTS Savings over FTS	EFTS + PFTS	PFTS Savings over EFTS
akiyo	173710	119360	31.29%	127356	-6.70%
news	181147	129750	28.37%	135964	-4.79%
silent	226272	147618	34.76%	147148	0.32%
coastguard	211202	142307	32.62%	137417	3.44%
foreman	307799	199651	35.14%	185731	6.97%
carphone	261696	167917	35.84%	153938	8.32%
car	405309	271308	33.06%	253982	6.39%
claire	196439	135116	31.22%	131664	2.55%
miss america	234434	146155	37.66%	130585	10.65%
Average	**244223**	**162131**	**33.33%**	**155976**	**3.02%**

Similar to the results obtained from measuring block matches, average SAD operation measurements show the same 33% improvement by EFTS and an additional 3% improvement by adding PFTS. Finally, Figure 4-17 show the maximum SAD operations for the Foreman video sequence using EFTS and PFTS. Again, this is important for some systems to know its peak resource requirements.

Figure 4-17. Foreman Maximum SAD Operations Using EFTS and PFTS



EFTS showed a lower but significant 27% improvement over FTS in peak resource usage. The addition of PFTS on the other hand had a relatively small negative effect at higher bit rates of 5% but up to 14% savings at lower bit rates. Table 4-8 show the maximum SAD operations for other video sequences using EFTS and PFTS.

Table 4-8: **Maximum SAD Operations per Macroblock Using EFTS and PFTS**

Test Sequence	FTS	EFTS	EFTS Savings over FTS	EFTS + PFTS	PFTS Savings over EFTS
akiyo	201216	128256	36.26%	133376	-3.99%
news	228608	164096	28.22%	172800	-5.30%
silent	289536	185856	35.81%	188160	-1.24%
coastguard	352000	219136	37.75%	280576	-28.04%
foreman	606976	440576	27.41%	404480	8.19%
carphone	385792	261120	32.32%	212992	18.43%
car	622336	457472	26.49%	416512	8.95%
claire	251904	162048	35.67%	148992	8.06%
miss america	295168	180992	38.68%	152064	15.98%
Average	**359282**	**244395**	**33.18%**	**234439**	**2.34%**

Notice that although for the coastguard sequence and some of the other sequences the peak resources usage has gone up, the peak usage is still on average 33% better for EFTS and an additional 2% better when PFTS is added. Also, acknowledge these values are savings on top of what FTS already provides.

Significant improvements of around 33% were seen with EFTS over FTS and a smaller improvement of 3% was seen with PFTS. The effects of PFTS was shown to be negative on some video sequences but was still able to obtain a 3% improvement on average. Finally, it was seen that these improvements came with little cost in terms of R-D performance.

4-5 Sub-Pixel FTS

In Sub-Pixel Flexible Triangle Search (SP-FTS), FTS is extended from full-pixel motion estimation to sub-pixel motion estimation. A new FTS technique performed in quarter-pixel accuracy is presented here. Full-pixel FTS, as presented in previous sections, has sub-pixel ME disabled and shall be referred to as FP-FTS. In SP-FTS, full-pixel and sub-pixel motion estimations are combined into a single step by eliminating full-pixel ME. Half-Pixel FTS (HP-FTS) has shown in H.263 [6] to have lower complexity than a two-stage full-pixel and half-pixel motion estimation technique. In this section, SP-FTS is extended to H.264 where quarter-pixel motion estimation is available. H.264 motion estimation can be viewed as a three step process whereby full-pixel, half-pixel, and quarter-pixel refinements are done. The proposal in this section is Quarter-Pixel FTS (QP-FTS), whereby these three steps are combined into one by eliminating full-pixel and half-pixel refinements. In addition to QP-FTS, for comparison purposes, the original FP-FTS algorithm is combined with full sub-pixel search (SP-FS). FP-FTS and SP-FS shall be referred to as (FP-FTS+SP-FS). The idea of SP-FS will be discussed later in more detail.

In sub-pixel motion estimation, intermediate pixels are obtained from integer pixels using interpolation. Half-pixel values are obtained using a one dimension 6-tap finite-impulse-response (FIR) filter. The interpolation filter is applied horizontally and vertically to obtain all the half-pixel points. The quarter-pixel values are then obtained by averaging integer and half-pixel values. Refer to Figure 2-4 for the half and quarter-pixel grid.

In Figure 2-4, pixel positions marked by upper-case letters are integer pixel locations. Lower-case letters mark half-pixel positions and quarter-pixel locations. But note that positions half-pixel positions are also quarter pixel positions. The positions *n* and *e* are calculated using Equation 4-2.

$$\begin{aligned} e_1 &= (A - 5B + 20C + 20D - 5E + F) \\ n_1 &= (G - 5H + 20C + 20I - 5J + K) \\ e &= (e_1 + 16) >> 5 \\ n &= (n_1 + 16) >> 5 \end{aligned} \qquad \textbf{(4-2)}$$

The values *n* and *e* are further clipped to the range of 0 to 255. Half-pixel location *p* is calculated using the same FIR filter as described in Equation 4-3.

$$\begin{aligned} p_1 &= (l - 5m + 20n_1 + 20r_1 - 5s + t) \\ p &= (p_1 + 512) >> 10 \end{aligned} \qquad \textbf{(4-3)}$$

Again, the final value of *p* is clipped to the range of 0 to 255.

The quarter-pixel locations at *e*, *f*, *g*, *u*, etc are calculated by averaging the two nearest integer and half-pixel locations. The value is also rounded up by adding 1 before averaging. For example, *d* is calculated in Equation 4-4.

$$d = (C + e + 1)/2 \qquad \textbf{(4-4)}$$

On the other hand, quarter sample locations at *h* and *j* are calculated using the diagonal half pixel neighbours as shown in equation 4-5.

$$h = (e + n + 1)/2 \qquad \textbf{(4-5)}$$

Note that the discussed interpolation method is used in the luma frames where motion estimation is performed. The benefits of sub-pixel motion estimation and specifically quarter-pixel motion estimation in H.264 over half-pixel motion estimation in H.263 is to provide more accurate estimation of motion and thus increase PSNR and improve the R-D curve. The trade-off for the improvement in R-D is the increase in complexity required to calculate the interpolated frames.

Typically, in motion estimation, a full-pixel search is completed first in a specified search area. Once the minimum is found, half-pixel motion estimation is used to test surrounding half-pixel locations to fine tune the minimum. In the case of H.264, further refinement is done by testing the quarter-pixel locations around the half-pixel locations. Depending on the technique, all or a subset of these half-pixel and quarter-pixel locations are tested. If all of these positions are tested, it is known as full half-pixel and full quarter-pixel search.

In the JVT H.264 reference software, a quarter-pixel interpolated frame is always computed before any motion estimation is done. Instead of electing to interpolate the whole frame, a technique can be used to only interpolate the samples that are necessary during motion estimation. However, such a method is likely to be inefficient since samples may be calculated more than once because samples may be used in the motion estimation of several macroblocks. Also, if half and quarter motion vectors exist, the encoder is required to compute these anyway for its motion compensation block. As such, the quarter-pixel interpolated frame is used in FTS. The QP-FTS algorithm is completely executed in the quarter-pixel interpolated frame. The QP-FTS algorithm is the same as that described in FP-FTS.

The QP-FTS is implemented in the H.264 JM software and its performance is compared to that of the other search algorithms. In previous simulations, the respective sub-pixel refinement for each search algorithm was disabled. For these simulations, the sub-pixel refinements are re-enabled. In addition, the integer FTS technique is paired with the full search sub-pixel motion estimation (FP-FTS+SP-FS). Whereby the full search sub-pixel motion estimation performs an exhaustive search of neighbouring eight half-pixel positions and then the neighbouring eight quarter-pixel positions.

4-6 Sub-Pixel FTS Simulation Results

Starting with the R-D performance of QP-FTS, the R-D performance is compared to FP-FTS and FP-FTS+SP-FS. It is expected that since ME is done in quarter-pixel accuracy, the PSNR performance is better. Figure 4-18 compares the R-D performance of QP-FTS against FP-FTS and FP-FTS+SP-FS.

Figure 4-18. Foreman Luma R-D Curve Using QP-FTS Compared to FP-FTS

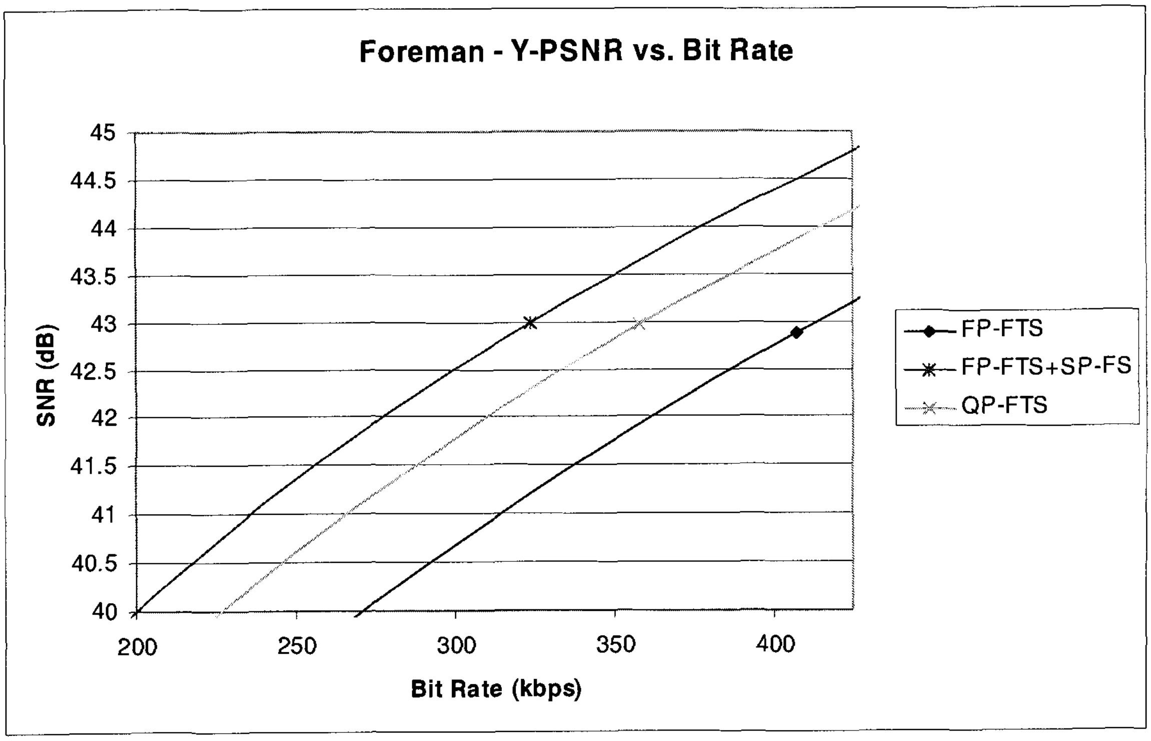


Figure 4-18 shows that QP-FTS performs better than FP-FTS by 1 dB and FP-FTS+SP-FS is 1.75 dB better than FP-FTS. Figure 4-19 and Figure 4-20 show the performance of QP-FTS against other available ME technieques.

Figure 4-19. Foreman Luma R-D Curve Using QP-FTS

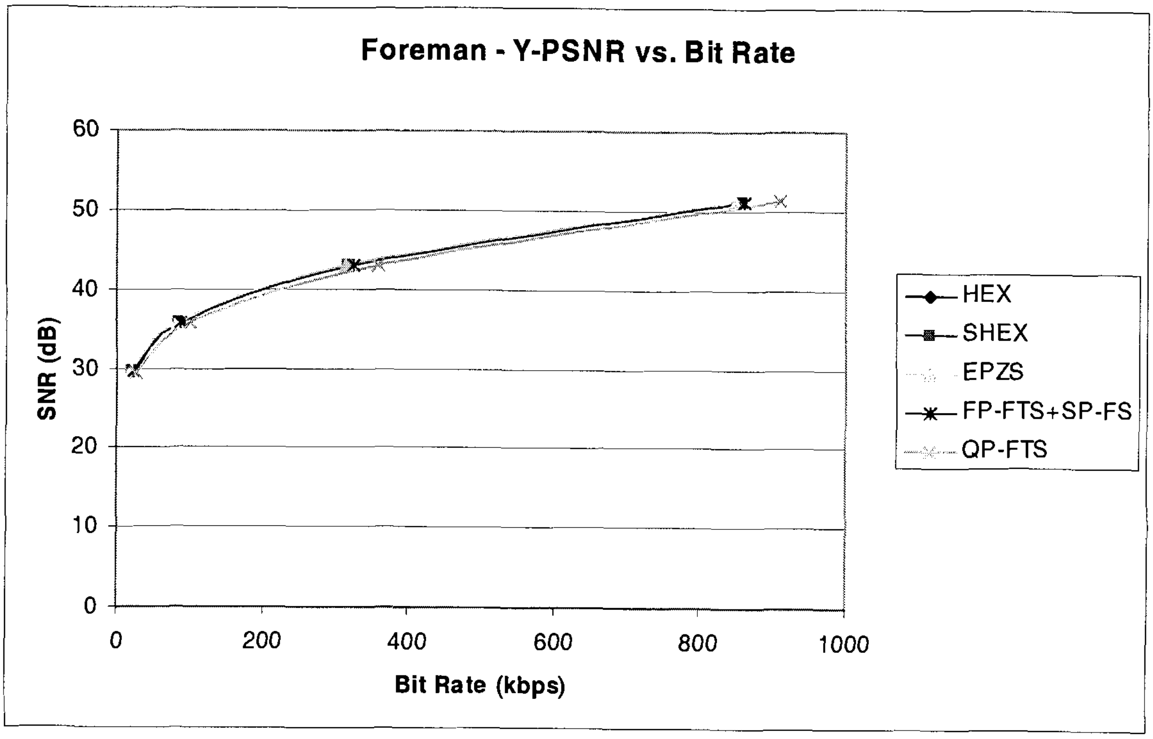
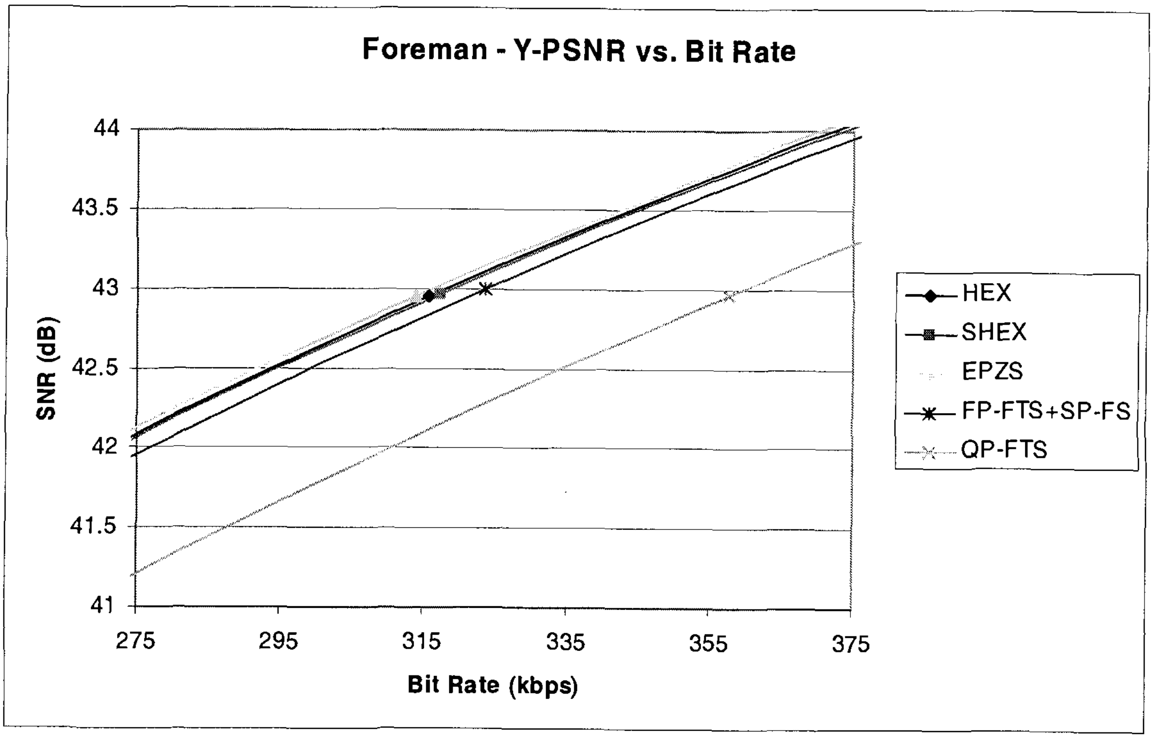


Figure 4-20. Foreman Luma R-D Curve Using QP-FTS (Close-Up)



Instead of improving R-D, QP-FTS exhibited worst performance by 0.8 dB compared to FP-FTS+SP-FS. This is not too surprising since the combination of a full pixel FTS search and a full sub-pixel search can exhibit the ideal SP-FTS performance. Figure 4-21 and Figure 4-22 show the R-D performance of the chroma frames which exhibit the same results as the luma frame.

Figure 4-21. Foreman Chroma U R-D Curve Using QP-FTS

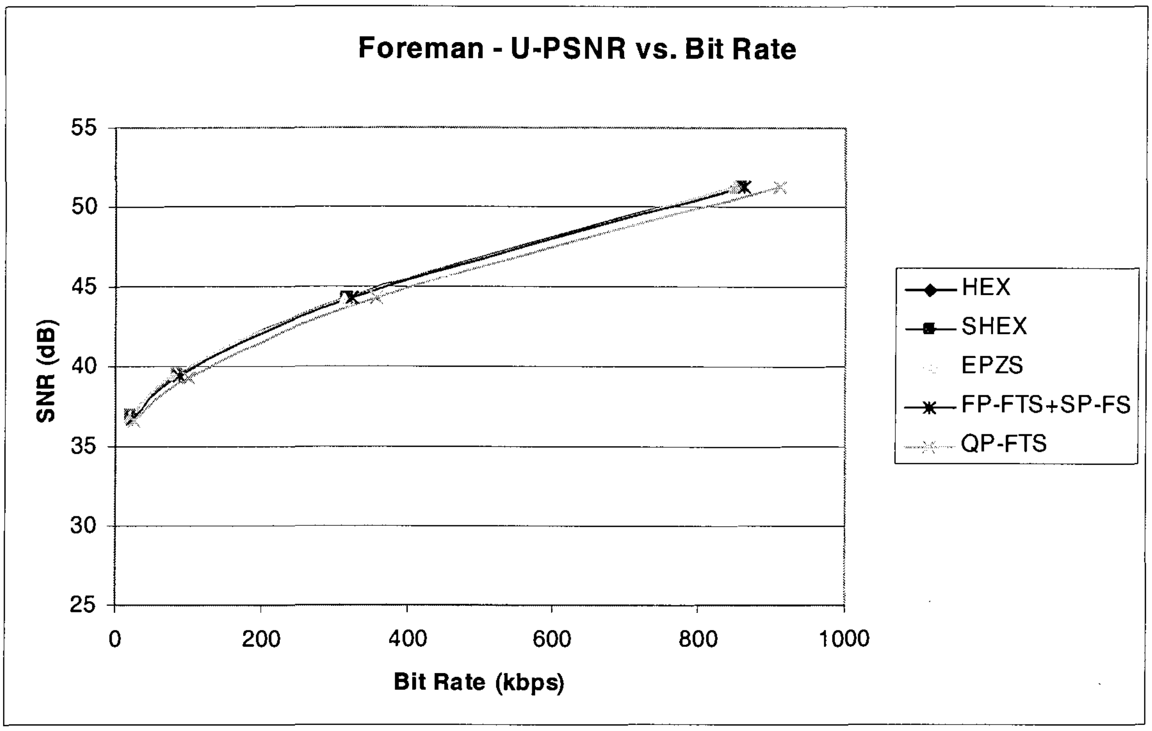


Figure 4-22. Foreman Chroma V R-D Curve Using QP-FTS

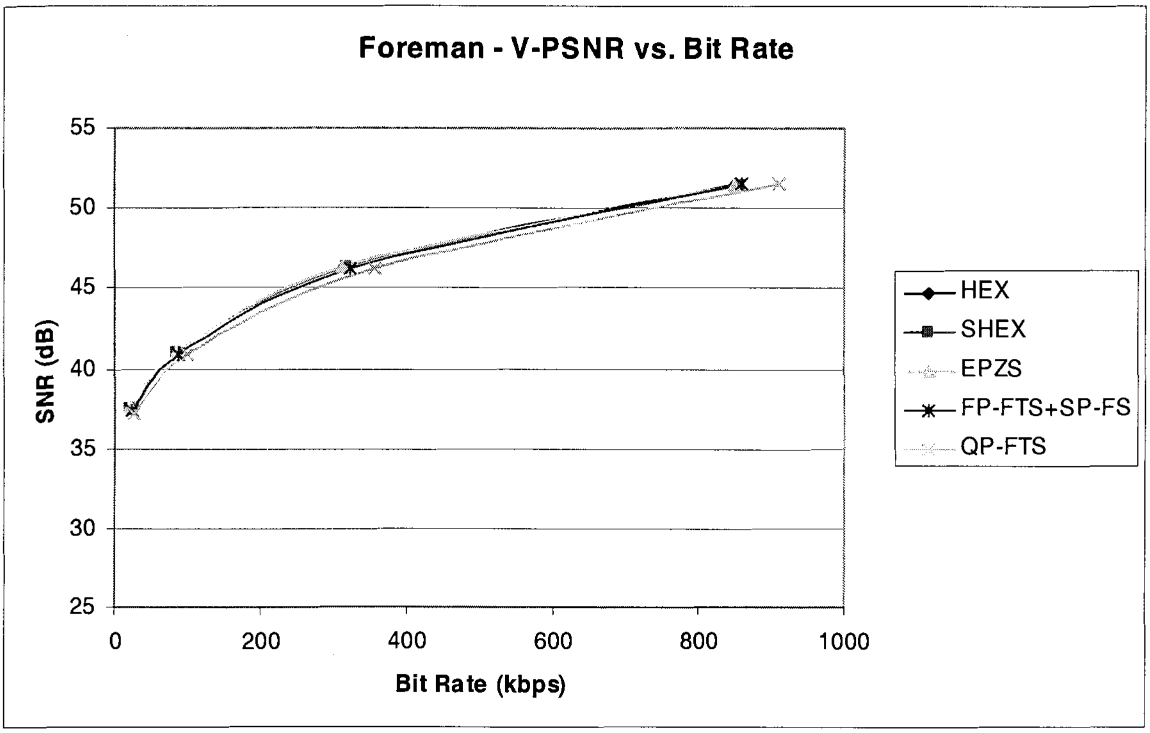


Figure 4-23 show the complexity of the search algorithms based on the number of block matches. It is expected that the complexity of sub-pixel search algorithms are significantly more complex than the full-pixel search results presented in earlier sections. For example, FP-FTS exhibited an average of 6.15 block matches per macro-block. As discussed, full sub-pixel search requires 16 block matches. Hence, it is expected that FP-FTS+SP-FS will require 16+6.15 = 22.15 block matches per macro-block. Indeed, the value of 22.15 block matches per macro-block was measured.

Figure 4-23. Foreman Block Matches Using QP-FTS

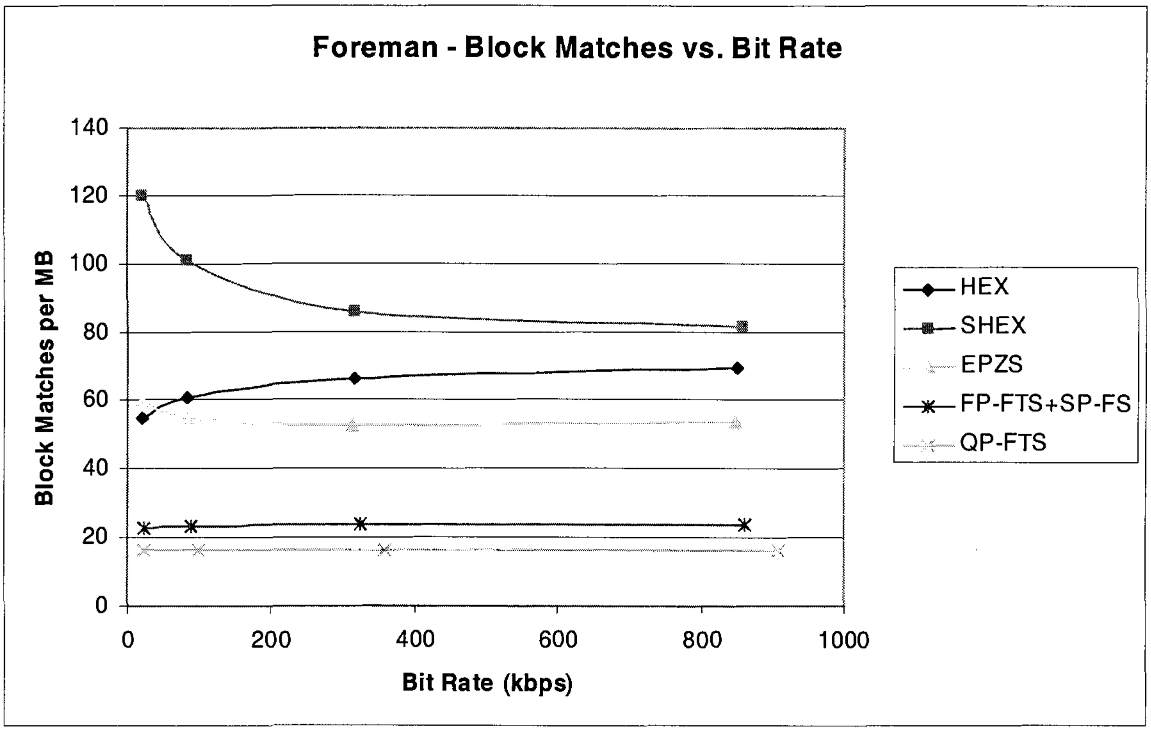


Figure 4-23 show that FP-FTS+SP-FS performed better than the JM techniques. Specifically, FP-FTS+SP-FS performed 55% to 60% better than EPZS. This improvement is similar to that seen when only full-pixel ME was used. Also, QP-FTS performed better than FP-FTS+SP-FS by 29% to 33%. Hence, although QP-FTS exhibited slightly worst performance in terms of R-D, the savings it exhibits in complexity may be an attractable trade-off. Table 4-9 shows the performance of FP-FTS+SP-FS and QP-FTS over other video sequences.

Table 4-9: Average Number of Block Match Operations per Macroblock

Test Sequence	HEX	SHEX	EPZS	FP-FTS+SP-FS	FTS Savings over EPZS
akiyo	29.75	26.01	29.60	21.01	29.01%
news	41.46	45.30	37.31	21.33	42.83%
silent	44.47	53.44	40.25	21.83	45.77%
coastguard	58.72	119.23	54.25	21.30	60.73%
foreman	60.52	100.95	54.97	23.28	57.64%
carphone	54.28	79.10	48.17	22.22	53.87%
car	62.92	109.30	62.03	25.38	59.08%
claire	32.42	27.68	30.74	21.16	31.15%
miss america	31.36	30.14	31.99	20.87	34.76%
Average	**46.21**	**65.68**	**43.26**	**22.04**	**46.09%**

Table 4-10: Average Number of Block Match Operations per Macroblock Using QP-FTS

Test Sequence	QP-FTS	QP-FTS Savings over EPZS	QP-FTS Savings over FTS
akiyo	12.64	57.30%	39.86%
news	13.06	65.00%	38.78%
silent	13.42	66.65%	38.50%
coastguard	15.39	71.63%	27.74%
foreman	16.08	70.75%	30.93%
carphone	15.63	67.56%	29.68%
car	15.54	74.96%	38.80%
claire	14.70	52.19%	30.55%
miss america	16.02	49.92%	23.23%
Average	**14.72**	**63.99%**	**33.12%**

FP-FTS+SP-FS performed very well across the board averaging a 46% savings in complexity. QP-FTS performed on average 33% better than FP-FTS+SP-FS and more significantly, it performed on average 64% better than EPZS. Again, the savings in complexity may be worth the trade-off in R-D performance. Figure 4-24 shows the average SAD operations required by the sub-pixel search algorithms.

Figure 4-24. Foreman Average SAD Operations Using QP-FTS

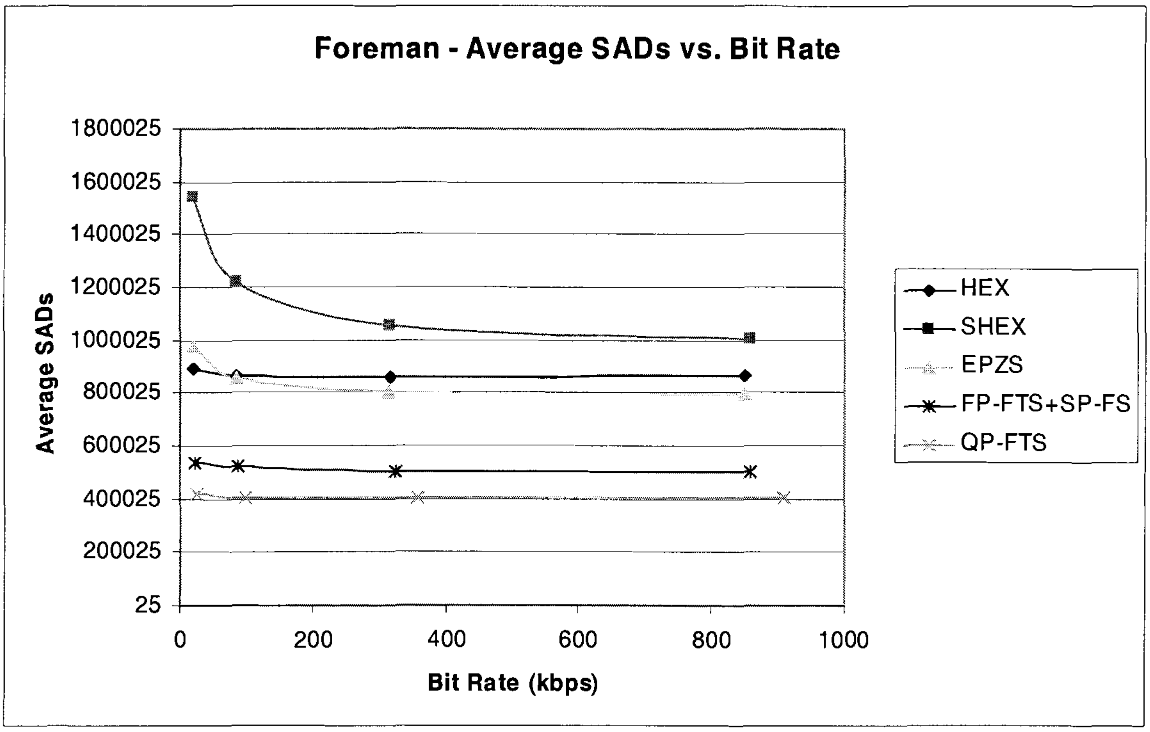


Table 4-11 summarizes the results of average SAD operations for all the video sequences.

Table 4-11: Average Number of SAD Operations per Macroblock

Test Sequence	HEX	SHEX	EPZS	FP-FTS+SP-FS	FTS Savings over EPZS
akiyo	315357	261810	374188	325721	12.95%
news	537944	524673	555985	397694	28.47%
silent	570693	661989	601776	396370	34.13%
coastguard	853112	1505507	848816	470646	44.55%
foreman	867516	1223824	859728	520656	39.44%
carphone	765198	959058	756505	489579	35.28%
car	1113797	1668049	1141074	609253	46.61%
claire	489543	340227	489464	438339	10.45%
miss america	550718	415290	559312	477420	14.64%
Average	**673764**	**840047**	**687428**	**458409**	**29.61%**

Table 4-12: Average Number of SAD Operations per Macroblock Using QP-FTS

Test Sequence	EPZS	QP-FTS	QP-FTS Savings over EPZS	QP-FTS Savings over FTS
akiyo	374188	320276	14.41%	1.67%
news	555985	331214	40.43%	16.72%
silent	601776	340454	43.43%	14.11%
coastguard	848816	390330	54.01%	17.07%
foreman	859728	407663	52.58%	21.70%
carphone	756505	396052	47.65%	19.10%
car	1141074	393758	65.49%	35.37%
claire	489464	372523	23.89%	15.01%
miss america	559312	406072	27.40%	14.94%
Average	**687428**	**373149**	**41.03%**	**17.30%**

Looking at average SAD operations, the FTS algorithms did not perform as well as what was seen in block matches. Still, FTS with full sub-pixel search still performed 30% better than EPZS and QP-FTS added another 17% savings on top of that or 41%

savings over EPZS. Figure 4-25 shows the maximum SAD operations for the search algorithms.

Figure 4-25. Foreman Maximum SAD Operations Using QP-FTS

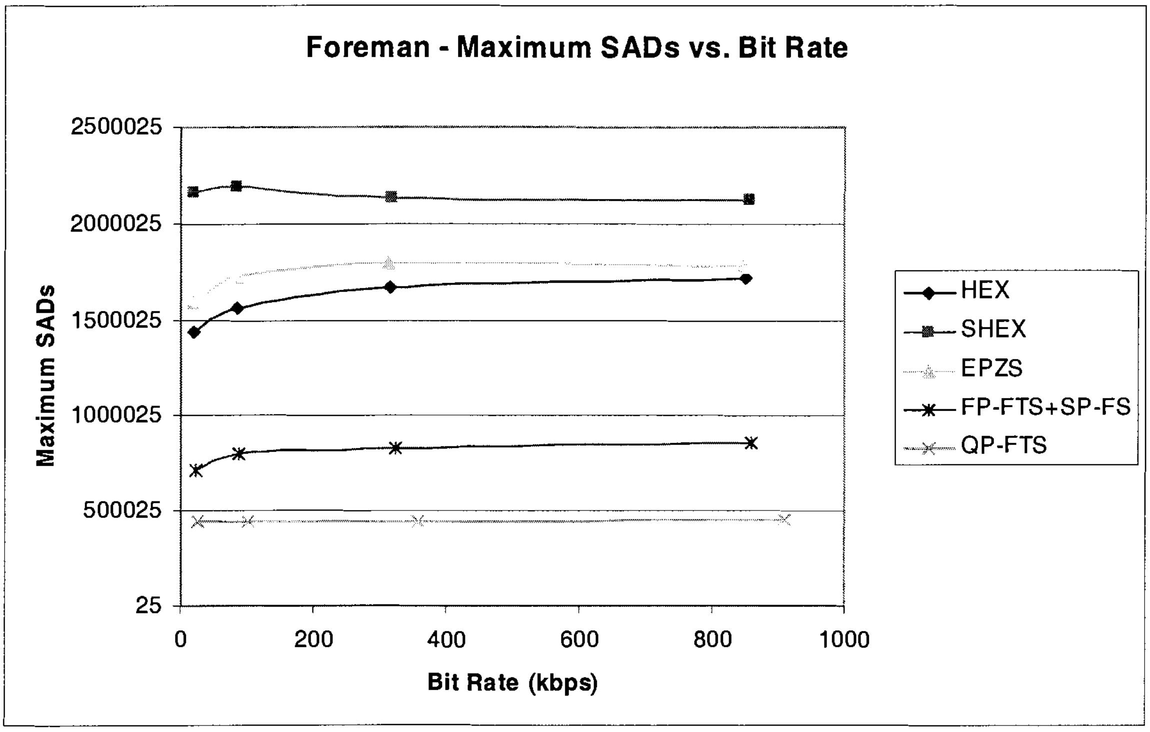


Table 4-13 show the results of measuring maximum SAD operations for the rest of the video sequences.

Table 4-13: **Maximum SAD Operations per Macroblock**

Test Sequence	HEX	SHEX	EPZS	FP-FTS+SP-FS	FTS Savings over EPZS
akiyo	391408	405856	426256	346656	18.67%
news	779600	986992	806720	456400	43.43%
silent	888896	1099168	867680	468000	46.06%
coastguard	1061216	1978448	1070816	704224	34.23%
foreman	1568064	2188736	1729488	802752	53.58%
carphone	1118032	1594576	1088896	565584	48.06%
car	1820256	2474656	2034976	827648	59.33%
claire	589616	522080	563168	467648	16.96%
miss america	624208	605504	643680	509456	20.85%
Average	**982366**	**1317335**	**1025742**	**572041**	**37.91%**

Table 4-14: **Maximum SAD Operations per Macroblock Using QP-FTS**

Test Sequence	EPZS	QP-FTS	QP-FTS Savings over EPZS	QP-FTS Savings over FTS
akiyo	426256	348160	18.32%	-0.43%
news	806720	362240	55.10%	20.63%
silent	867680	361984	58.28%	22.65%
coastguard	1070816	460288	57.02%	34.64%
foreman	1729488	441088	74.50%	45.05%
carphone	1088896	434176	60.13%	23.23%
car	2034976	435456	78.60%	47.39%
claire	563168	408064	27.54%	12.74%
miss america	643680	442880	31.20%	13.07%
Average	**1025742**	**410482**	**51.19%**	**24.33%**

The results are consistent with measuring block matches and average SAD operations. FTS with full sub-pixel search performed on average 38% better and QP-FTS added another 24% savings or 51% savings over EPZS.

FTS was extended as a SP-FTS algorithm in two ways. FTS paired with a full half-pixel and quarter-pixel searches showed significant improvement in complexity over other JM sub-pixel ME techniques. But, FTS showed little effect to the R-D performance. Also, FTS was implemented in the quarter-pixel domain completely bypassing full-pixel and half-pixel searches. Here results showed further improvement over FTS but came at a cost of slightly reduced R-D performance as compared to other sub-pixel ME techniques. However, SP-FTS showed to produce better R-D performance than any of the full-pixel ME techniques.

CHAPTER 5
CONCLUSIONS

This document discussed the H.264 standard produced by the Joint Video Team and some of its features that are introduced in the standard. All these features of H.264 serve the purpose of maintaining high picture quality while reducing bit rate. In other words, the rate-distortion efficiency is improved. The objective of improving the rate-distortion efficiency is to better enable video applications on more bandwidth limited applications that carry its data over cable, DSL, and WiFi. Of course, rate-distortion efficiency does not come free as it is a trade-off with complexity.

It has been shown that the ME component of H.264 consumes 70% of encoder processing time when a single reference frame is used. In fact, ME will consume up to 90% when multiple reference frames are used. These numbers make ME the heaviest component in H.264 and as such have been a popular research topic and have spurred this document as well. The JVT group implemented the JM, an H.264 reference software, that supports 4 ME techniques. These techniques are Full Search, Unsymmetrical-cross Multi-Hexagon-Grid Search, Simplified UMHexagonS, and Enhanced Predictive Zonal Search. Via tests, it was shown that Full Search resulted in the best R-D efficiency but at the cost of very high complexity. The UMHexagonS introduced many MV predictors, early termination with SAD prediction, and a complex search pattern. Unfortunately, UMHexagonS was still too complex and a simplified version was added that removed some MV predictors, added more early termination conditions, and simplified the search pattern. As a result, ME time was reduced significantly and it also resulted in greater R-D efficiency. EPZS is an algorithm that concentrates on MV predictors and an adaptive early termination condition. As such, EPZS has very simple search patterns to refine the MV. Through simulations, it was shown that the simplified EPZS performed the best of all the ME techniques in JM other than FS.

In this project we implement the Flexible Triangle Search that uses the triangle as the search pattern with three test points at its vertices. Starting from the smallest triangle, the triangle is reflected, expanded, translated, and contracted. These operations allow the triangle to quickly move from areas of high error to areas of low error. In order to implement this algorithm, three levels of pre-determined triangles were used that allowed easy implementation via look-up tables.

In the implementation of FTS, the median MV predictor was re-used and the Lagrangian cost function was used. Results show that FTS only dropped by 0.3 dB and 0.1 dB from the FS and EPZS respectively. On the other hand, the FTS algorithm showed improvement on ME time of up to 30% over EPZS. Hence, FTS affected R-D efficiency little but was able to greatly reduce the complexity of ME.

Noticing an inefficiency in the algorithm, the Enhanced FTS modification is presented that save intermediate SAD results so that SAD values for a particular search location isn't calculated twice. In addition, an optimization is made to predict the direction of the minimum in Predictive FTS. This is done by directing the triangle in the particular direction by choosing the correct starting triangle. Both these modifications are shown to not affect R-D while EFTS reduced complexity by another 33% over the original FTS algorithm and PFTS added another 3% savings.

Finally, the FTS algorithm is extended as a sub-pixel ME in two ways. First the PFTS algorithm is paired with a half and quarter-pixel full search. Results show that this alone provided a savings of 30% over other sub-pixel ME. Second, the FTS algorithm was executed directly in the quarter-pixel interpolated frame and any ME in the full-pixel and half-pixel frames are skipped. Although this exhibited a 0.8 dB reduction in R-D curve, it provided a savings of 41% over other sub-pixel ME techniques and 17% savings over FTS paired with half and quarter-pixel full search.

Significant savings alone can be achieved by using full-pixel FTS that can be further paired with full half and quarter-pixel searches for increased R-D. Even more savings are possible when QP-FTS is considered but at a cost of slightly reduced R-D.

5-1 Ongoing Research

On forward looking, much more work needs to be done in validating FTS. Tests need to be completed that verifies if the algorithm will not be trapped in a local minimum since it does not perform any special searches specifically far from the search center. The UMHexagonS accomplishes this by the use of the big hexagon search.

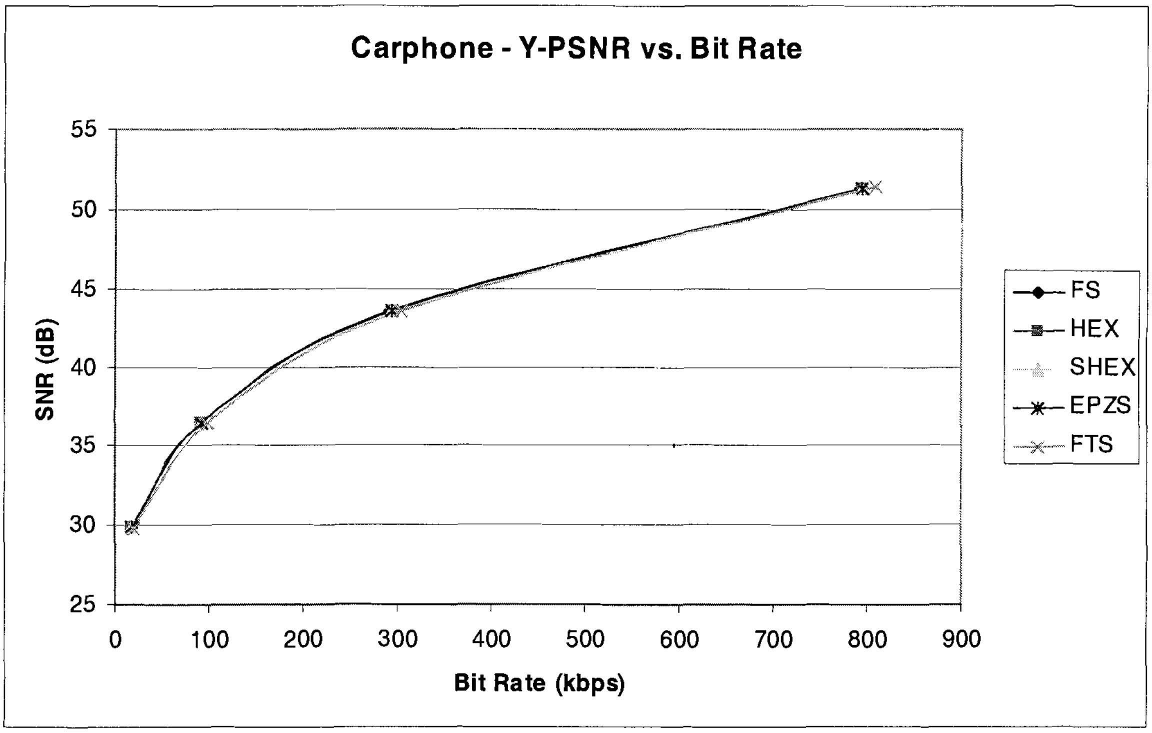
Also, several enhancements can be added to compliment the FTS algorithm. For example, we can take advantage of the benefits of some MV and SAD predictors used in UMHexagonS, Simplified UMHexagonS, and EPZS. Also, in sub-pixel ME, the FTS algorithm can be paired with more efficient sub-pixel searches rather than full half and full quarter-pixel searches.

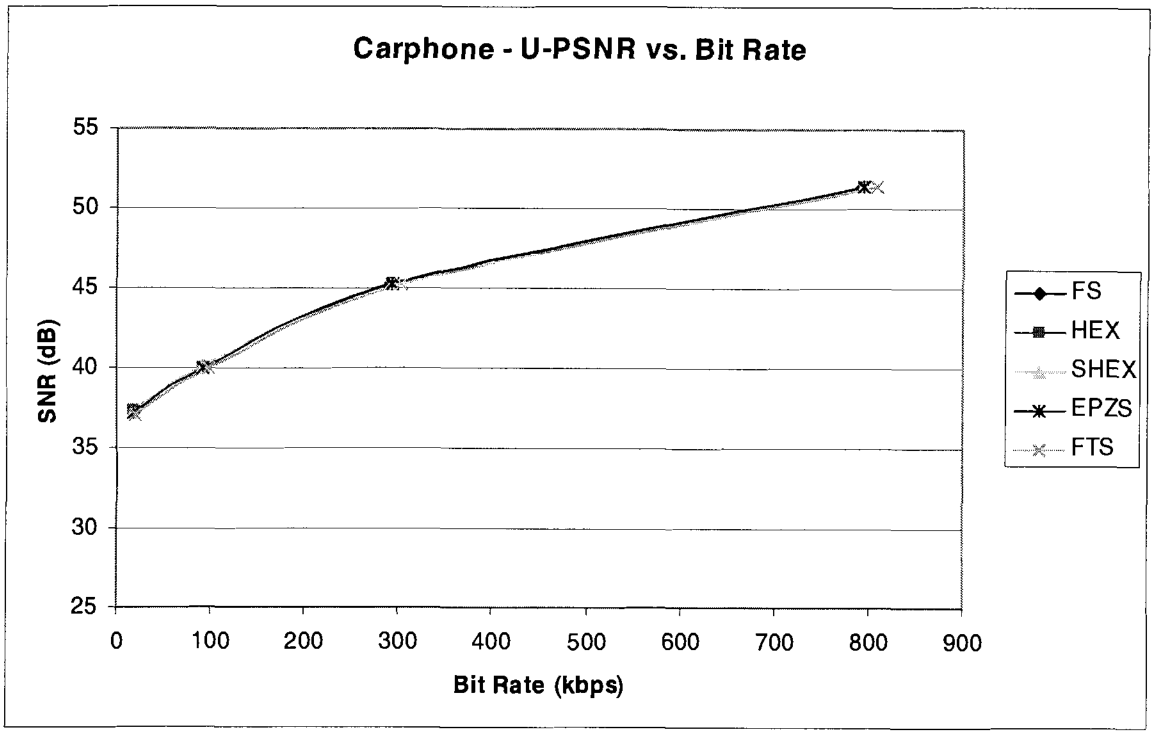
APPENDICES

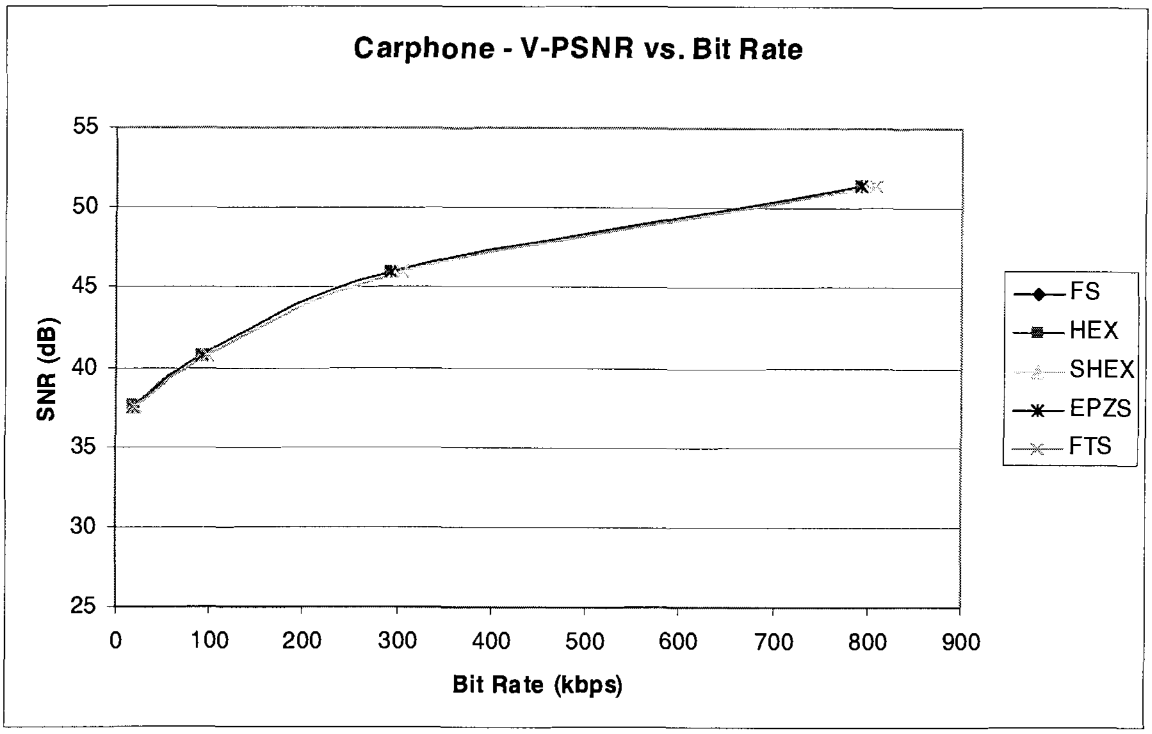
Appendix A: Detailed Analysis of the Carphone Video Sequence

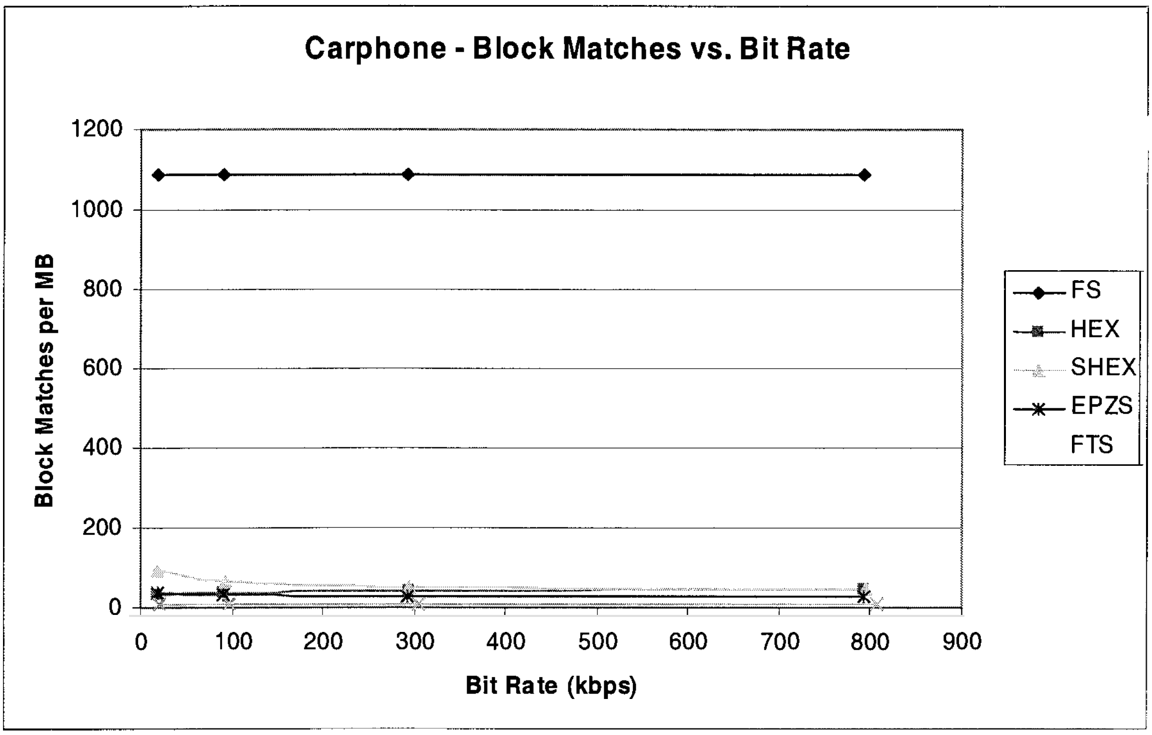
This appendix contains the detailed analysis of the carphone video sequence using the various versions of FTS discussed in this document. This appendix presents the various graphs for carphone that were done for the foreman video sequence.

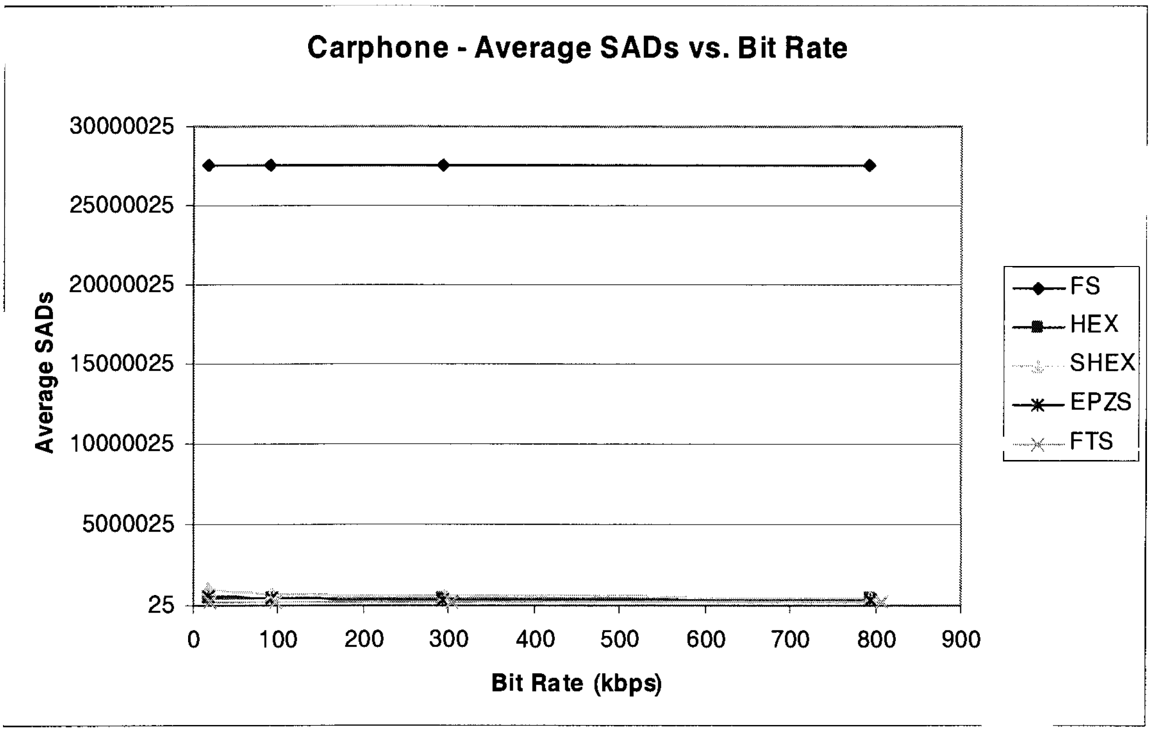
Car Phone Video Sequence Using FP-FTS

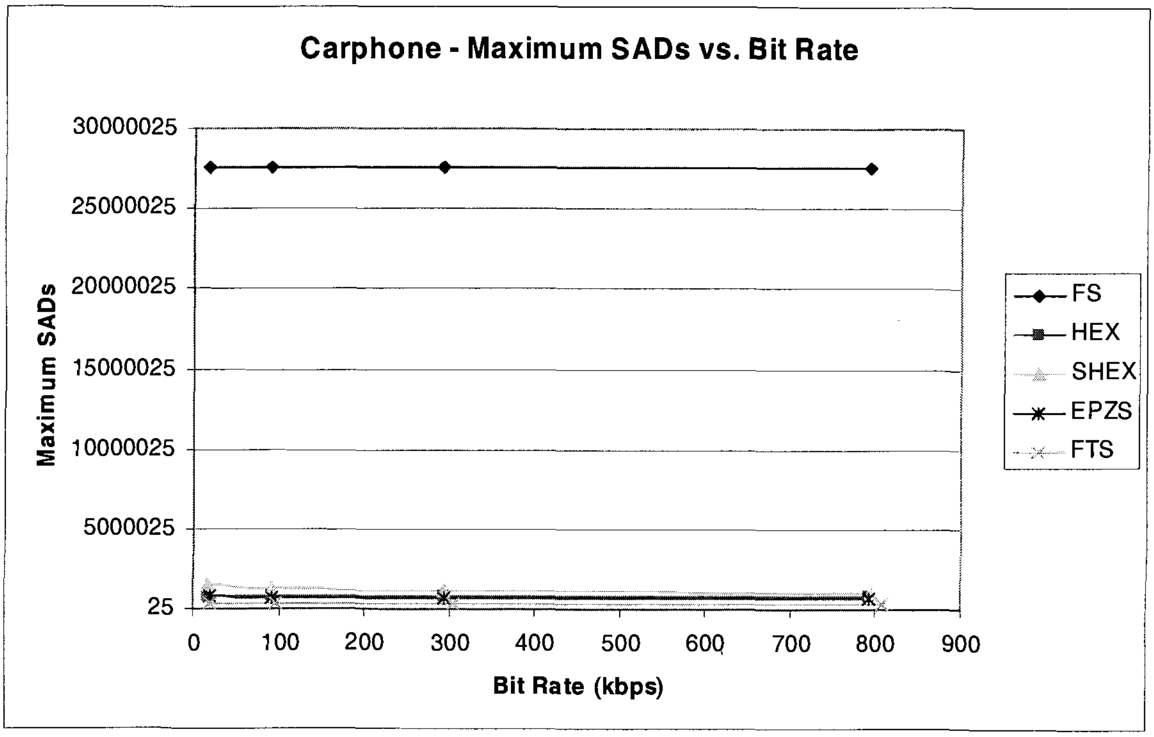



Carphone - U-PSNR vs. Bit Rate
SNR (dB)
55
50
45
40
35
30
25
0
100
200
300
400
500
600
700
800
900
Bit Rate (kbps)
FS
HEX
SHEX
EPZS
FTS

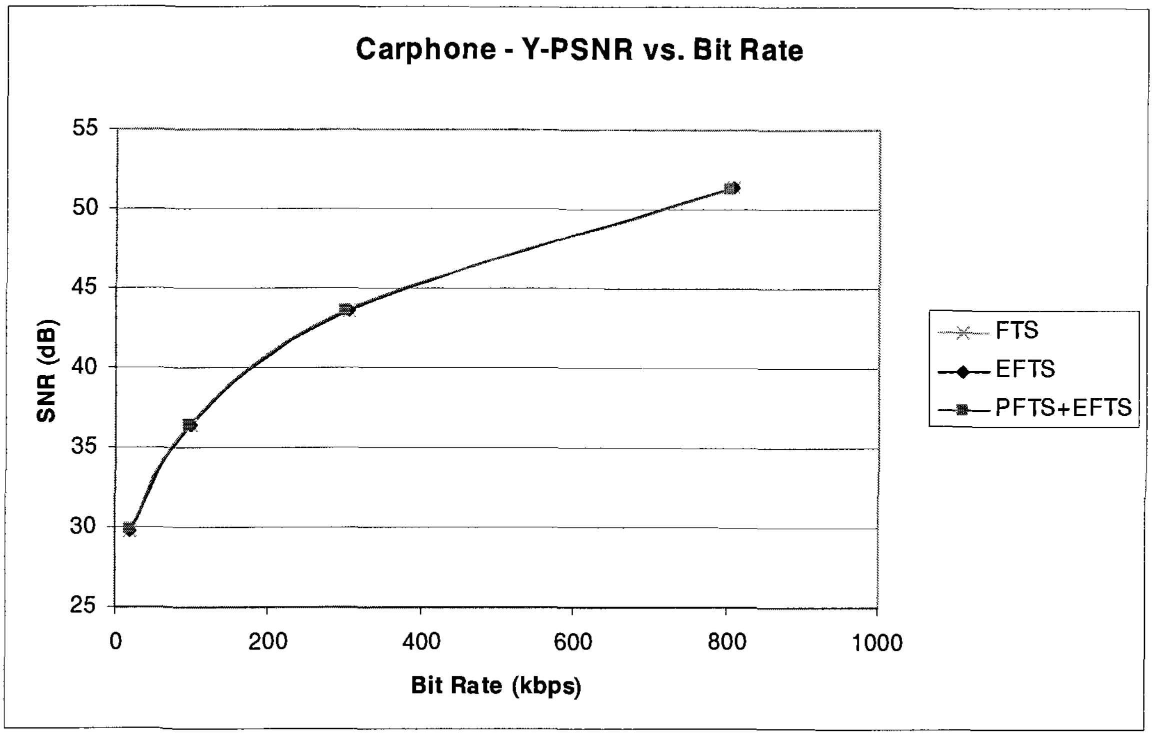

Carphone - V-PSNR vs. Bit Rate
SNR (dB)
55
50
45
40
35
30
25
0
100
200
300
400
500
600
700
800
900
Bit Rate (kbps)
FS
HEX
SHEX
EPZS
FTS

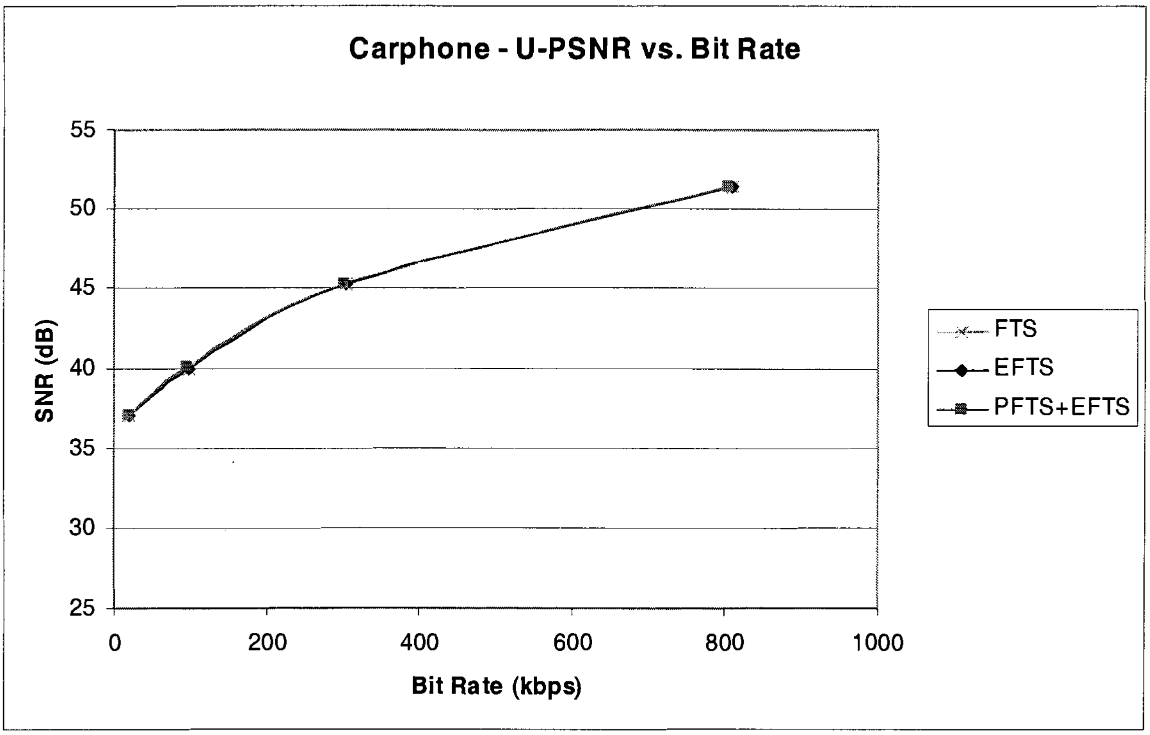

Carphone - Block Matches vs. Bit Rate
Block Matches per MB
1200
1000
800
600
400
200
0
0
100
200
300
400
500
600
700
800
900
Bit Rate (kbps)
FS
HEX
SHEX
EPZS
FTS

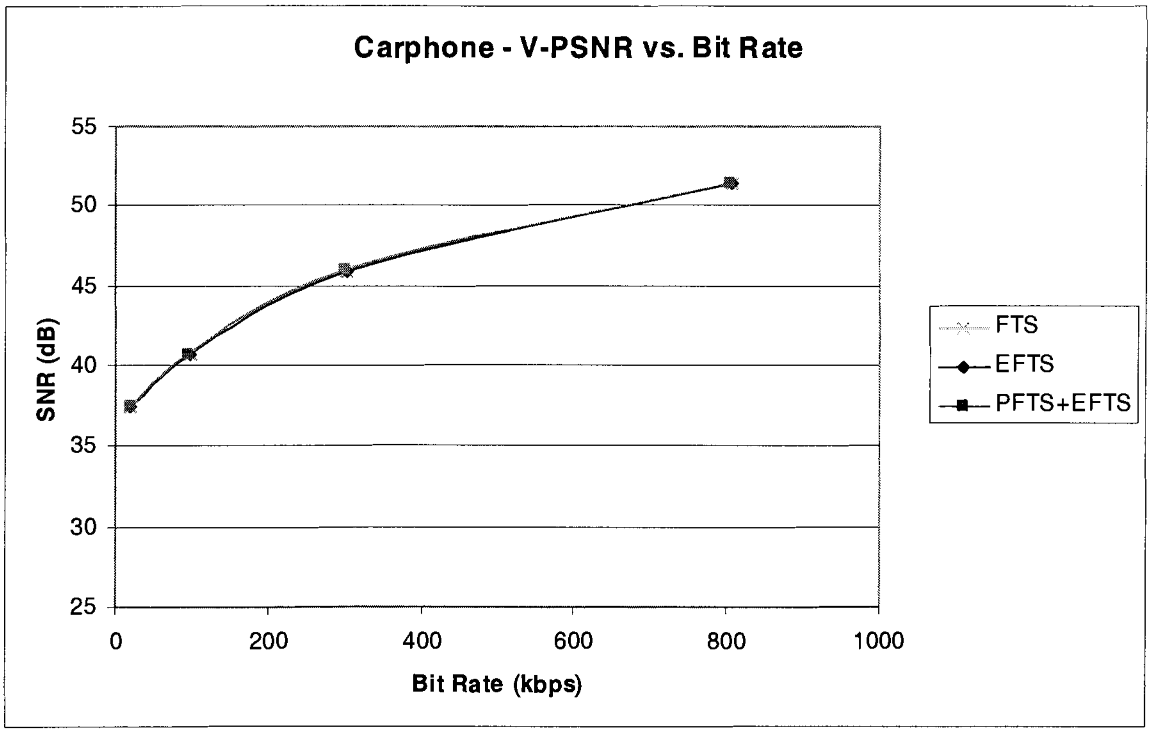

Carphone - Average SADs vs. Bit Rate
Average SADs
30000025
25000025
20000025
15000025
10000025
5000025
25
0
100
200
300
400
500
600
700
800
900
Bit Rate (kbps)
FS
HEX
SHEX
EPZS
FTS

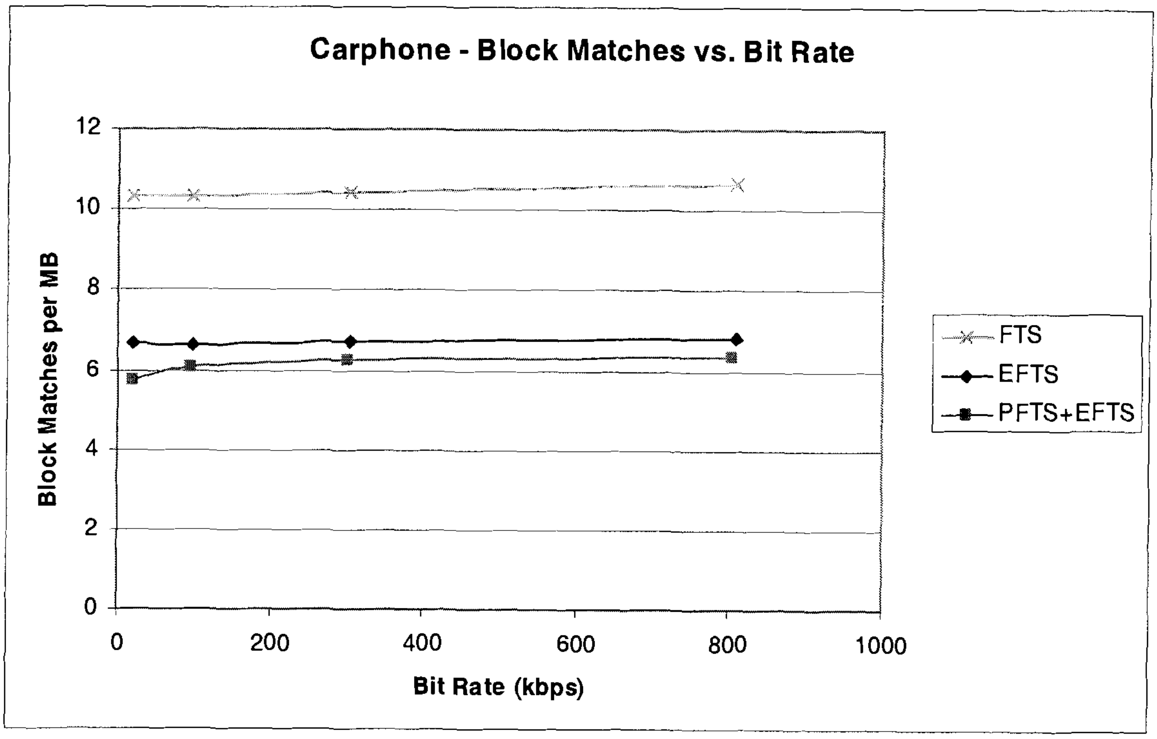


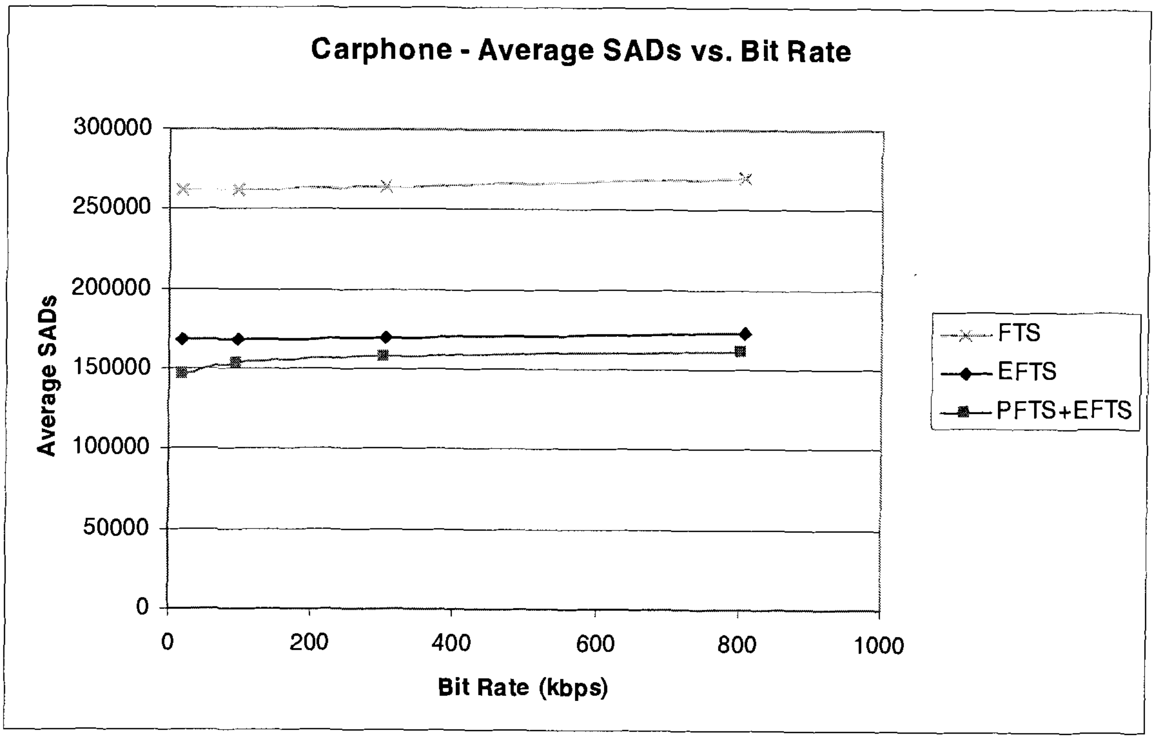
Car Phone Video Sequence Using EFTS and PFTS

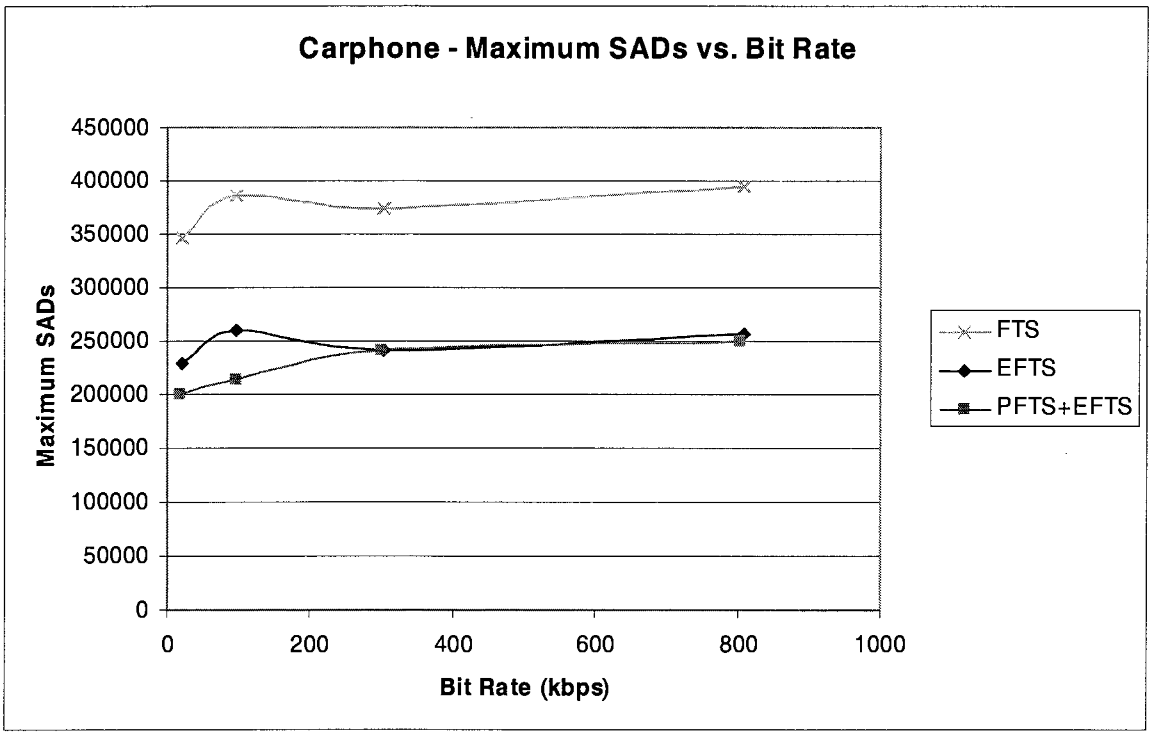



Carphone - U-PSNR vs. Bit Rate
55
50
45
40
35
30
25
SNR (dB)
0
200
400
600
800
1000
Bit Rate (kbps)
FTS
EFTS
PFTS+EFTS

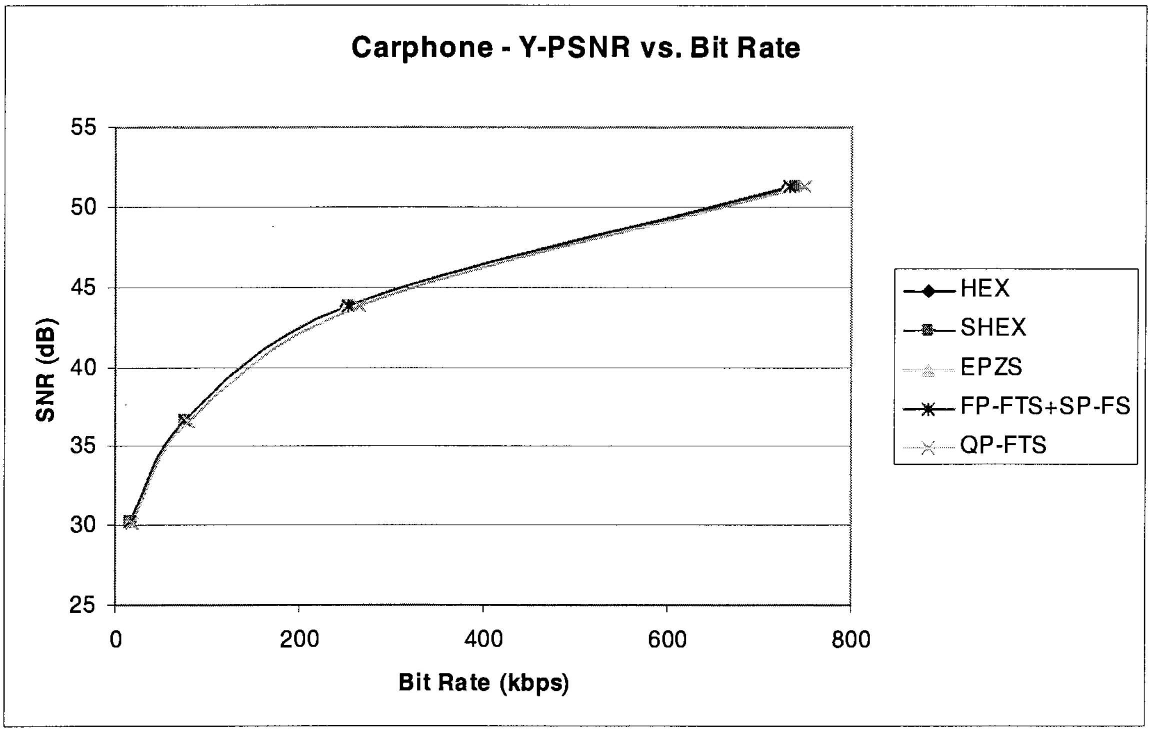

Carphone - V-PSNR vs. Bit Rate
55
50
45
40
35
30
25
SNR (dB)
0
200
400
600
800
1000
Bit Rate (kbps)
FTS
EFTS
PFTS+EFTS

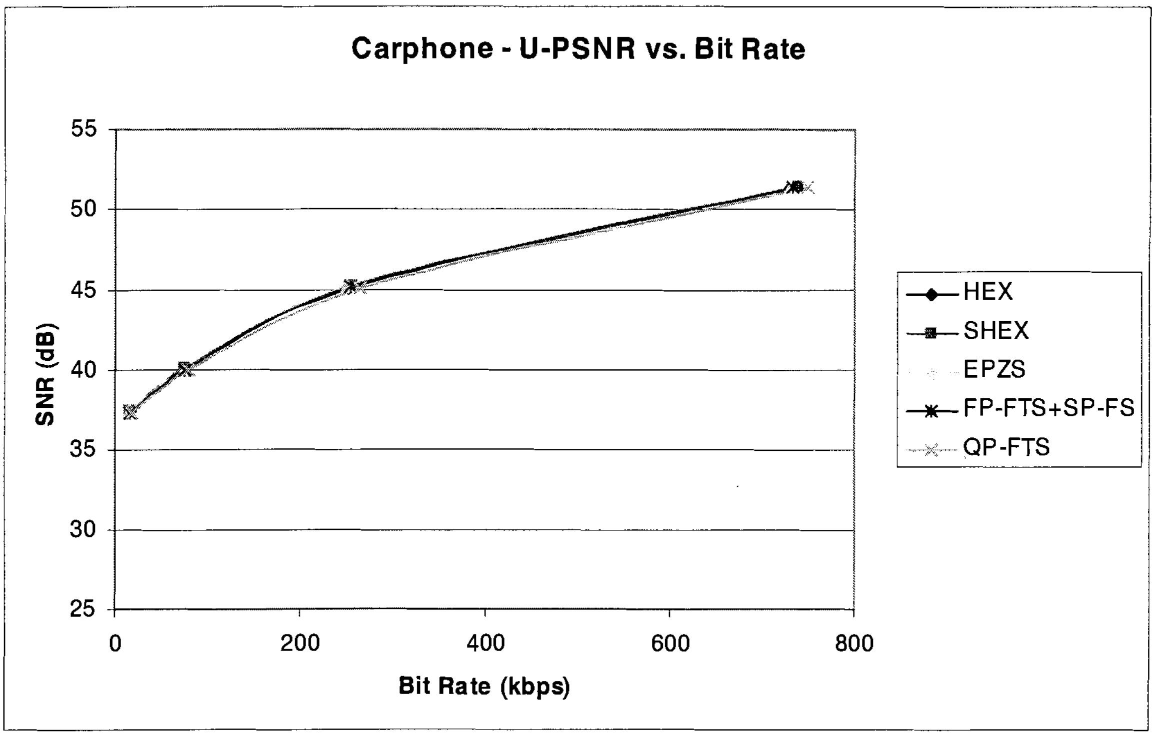

Carphone - Block Matches vs. Bit Rate
12
10
8
6
4
2
0
Block Matches per MB
0
200
400
600
800
1000
Bit Rate (kbps)
FTS
EFTS
PFTS+EFTS

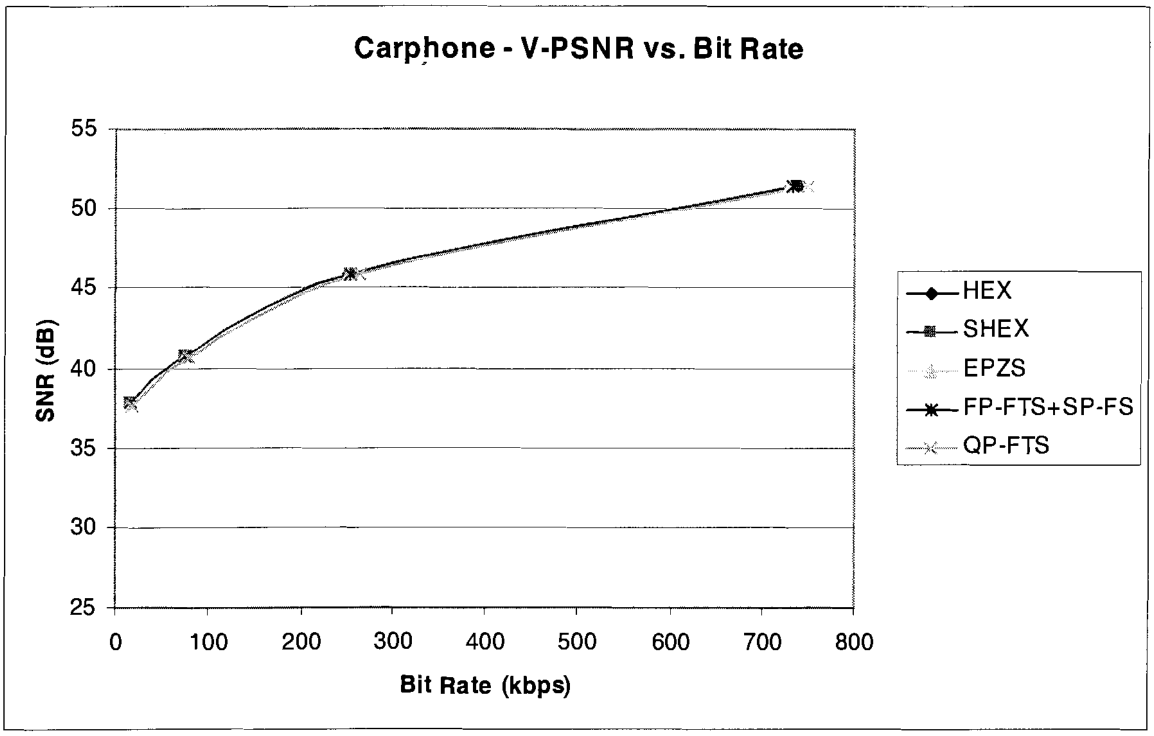

Carphone - Average SADs vs. Bit Rate
300000
250000
200000
150000
100000
50000
0
Average SADs
0
200
400
600
800
1000
Bit Rate (kbps)
FTS
EFTS
PFTS+EFTS

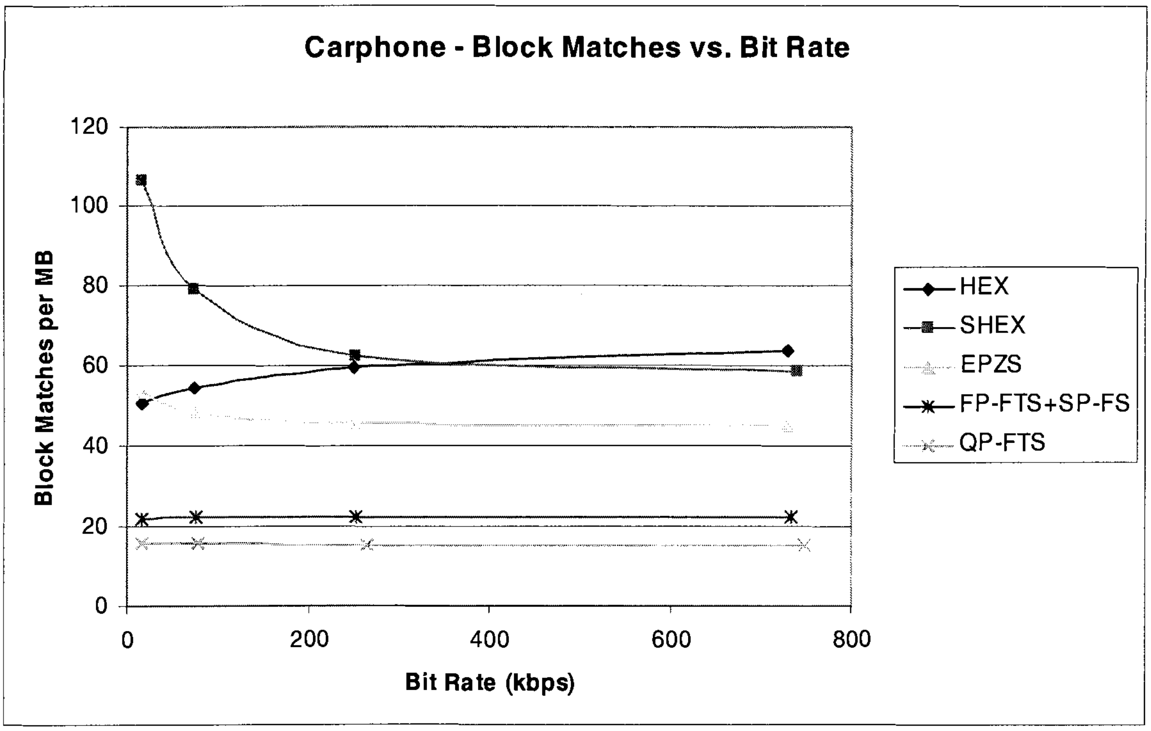


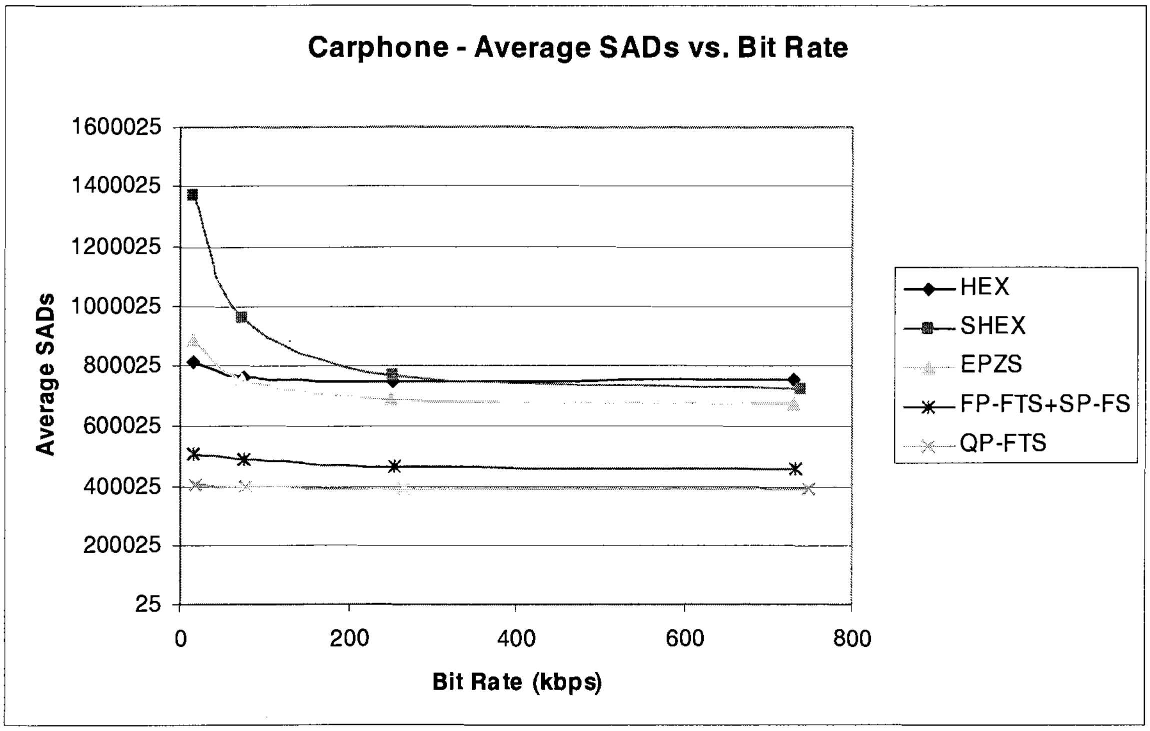
Car Phone Video Sequence Using QP-FTS

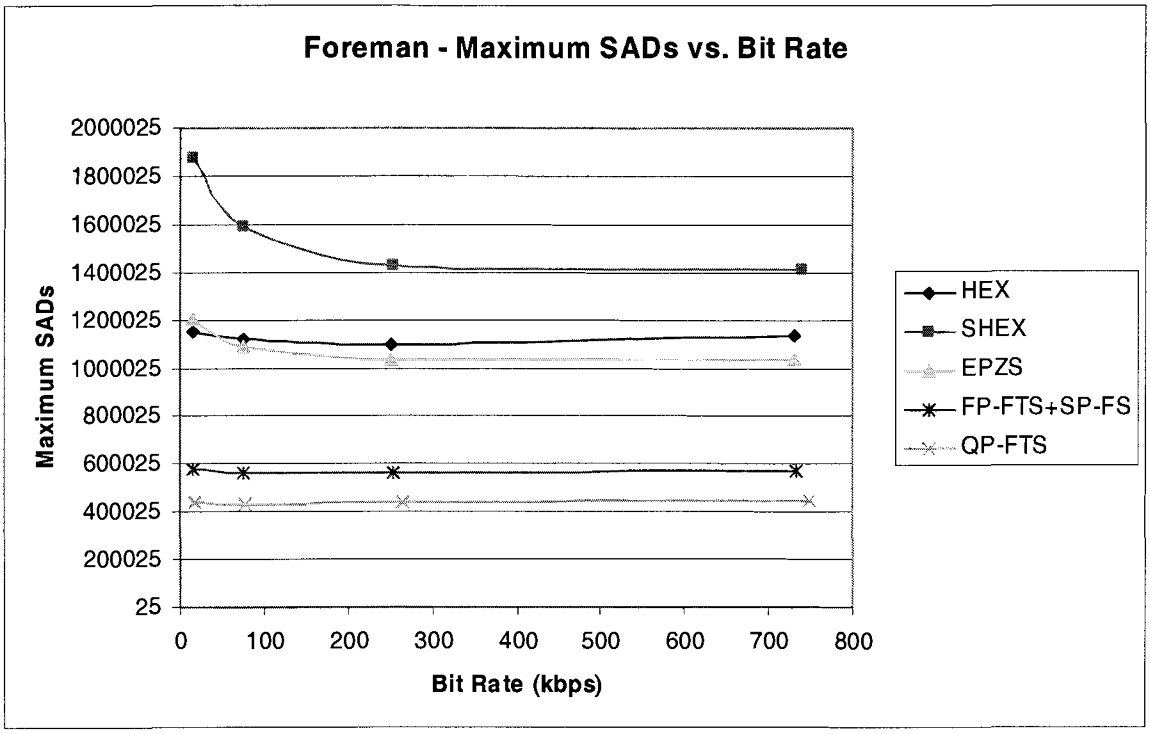



Carphone - U-PSNR vs. Bit Rate
55
50
45
40
35
30
25
SNR (dB)
0
200
400
600
800
Bit Rate (kbps)
HEX
SHEX
EPZS
FP-FTS+SP-FS
QP-FTS


Carphone - V-PSNR vs. Bit Rate
55
50
45
40
35
30
25
SNR (dB)
0
100
200
300
400
500
600
700
800
Bit Rate (kbps)
HEX
SHEX
EPZS
FP-FTS+SP-FS
QP-FTS


Carphone - Block Matches vs. Bit Rate
Block Matches per MB
120
100
80
60
40
20
0
0
200
400
600
800
Bit Rate (kbps)
HEX
SHEX
EPZS
FP-FTS+SP-FS
QP-FTS


Carphone - Average SADs vs. Bit Rate
Average SADs
1600025
1400025
1200025
1000025
800025
600025
400025
200025
25
0
200
400
600
800
Bit Rate (kbps)
HEX
SHEX
EPZS
FP-FTS+SP-FS
QP-FTS


Foreman - Maximum SADs vs. Bit Rate
2000025
1800025
1600025
1400025
1200025
1000025
800025
600025
400025
200025
25
Maximum SADs
0
100
200
300
400
500
600
700
800
Bit Rate (kbps)
HEX
SHEX
EPZS
FP-FTS+SP-FS
QP-FTS

REFERENCE LIST

[1] Hye-Yeon Cheong Tourapis, etc. "Fast Motion Estimation within the JVT codec", JVT-E023.doc, 5th Meeting: Geneva, Switzerland, 9-17 October, 2002

[2] "ITU-T Recommendation H.264, Advanced video coding for generic audiovisual services ", March 2005. http://www.itu.int/rec/T-REC-H.264-200503-I/en

[3] JM10.2, Reference Software of JVT, http://iphome.hhi.de/suehring/tml/index.htm.

[4] Mohamed M. Rehan, Pan Agathoklis, and Andreas Antoniou, "Flexible Triangle Search Algorithm for Block-Based Motion Estimation", Electrical and Computer Engineering, 2005, Canadian Conference on, May 1-4, 2005, Pages 269-272

[5] ----------, "Block-Based Motion Estimation Using An Enhanced Flexible Triangle Search Algorithm", Proceedings of Canadian Conference on Electrical and Computer Engineering (CCECE05), May 2005, pp. 259–262.

[6] Mohamed M. Rehan and Pan Agathoklis, "Half-Pixel Accurate Motion-Estimation Using A Flexible Triangle Search", Proceedings of IEEE Pacific Rim Conference on Communications, Computers and Signal processing (PACRIM'05), Aug. 2005, pp.233–236.

[7] ----------, "Prediction-Based Flexible Triangle Search Algorithm For Block Based Motion Estimation", Proceedings of Canadian Conference on Electrical and Computer Engineering (CCECE06), May 2006, pp. 2067-2070.

[8] Thomas Wiegand, Gary J. Sullivan, Gisle Bjøntegaard, and Ajay Luthra, "Overview of the H.264/AVC Video Coding Standard", IEEE Transactions on Circuits and Systems for Video Technology, Vol.13, No. 7, July 2003

[9] Xiaoquan Yi, Jum Zhang, etc. "Improved and simplified fast motion estimation for JM", JVT-P021.doc, 16th Meeting: Poznan, Poland, 24-29 July, 2005

[10] Zhibo Chen, Peng Zhou, etc. "Fast Motion Estimation for JVT", JVT-G016.doc, 7th Meeting: Pattaya II, Thailand, 7-14 March, 2003

[11] Zhibo Chen, Peng Zhou, etc. "Fast Integer Pel and Fractional Pel Motion Estimation for JVT", JVT-F017.doc, 6th Meeting: Awaji, Island, 5-13 December, 2002