A New Predictive Diamond Search Algorithm for Block Based Motion Estimation

Alexis M. Tourapis^{1†}, Guobin Shen[†], Ming L. Liou[†], Oscar C. Au[†], Ishfaq Ahmad[‡]

[†]Department of Electrical and Electronic Engineering,

[‡]Department of Computer Science,
The Hong Kong University of Science and Technology,
Clear Water Bay, Kowloon, Hong Kong.

ABSTRACT

In this paper a new fast motion estimation algorithm is presented. The algorithm, named as Predictive Diamond Search, is actually based on the Diamond Search (DS) algorithm, which was recently adopted inside the MPEG-4 standard. The DS algorithm, even though faster than most known algorithms, was found not to be very robust in terms of quality for several sequences. By introducing a new predictive criterion and some additional steps in DS, our simulation results show that the proposed algorithm manages to have similar complexity with the DS algorithm, while having superior and more robust quality, similar to that of the Full Search (FS) algorithm.

Keywords: Motion Estimation, Diamond Search, Prediction, Predictive Diamond Search, Block Matching, Video Coding

1. INTRODUCTION

The technique of block matching estimation is used in most video coding systems and standards such as MPEG-1/2/4 and ITU-T H.261/263 due to its efficiency. By performing motion estimation and compensation we are able to exploit the temporal correlation that exists between frames in a video sequence, and thus achieve high compression.

In block matching motion estimation, the current frame is initially partitioned into square blocks of pixels. For each one of these blocks, we try to find a best match inside a previously coded frame by using a predefined criterion. The best match is then used as a predictor for the block in the current frame, where as the displacement between the two blocks actually defines a motion vector (MV), which is associated with the current block. In the encoder it is only necessary to send the motion vector and a residue block, defined as the difference between the current block and the predictor. Both together actually require fewer bits than the original block.

The criterion used is typically the sum of absolute difference (SAD), which for a block A of size $N \times N$ inside the current frame, compared to a block B of distance (α, β) from A in the previous frame, is given as:

$$SAD(\alpha, \beta) = \sum_{x,y=1}^N |I_t(x, y) - I_{t-1}(x + \alpha, y + \beta)| \quad (1)$$

If a maximum displacement of p pixels/frame is allowed, then we will have $(2p+1)^2$ locations to search for the best match of the current block.

The Full Search (FS) algorithm can find the optimal solution (optimal in the sense of the criterion used) by exhaustively searching all possible blocks. It can be seen that this algorithm is rather computational intensive, making it difficult to apply for real-time video compression, particularly for software-based implementations. For this reason, in the upcoming standard of MPEG-4 [1-2], the Diamond Search (DS) [6-7] algorithm was adopted, due to its superiority versus other well-known algorithms such as Three-Step Search (TSS) [3], 2-D log Search [4], and New Three Step Search (NTSS) [5], in both terms of computational complexity and output quality.

¹ Correspondence: Email: alexis@ee.ust.hk; Telephone (+852) 2243 0594, Fax (+852) 2358 1485

Unfortunately it was demonstrated [8] that the DS algorithm, even though can in most cases reduce complexity significantly without much reduction in quality versus FS, does not perform as well for some cases, such as sequences with either significant global motion or scene variations, and can be said not to be robust in general.

In this paper a new algorithm named Predictive Diamond Search (PDS) will be presented which is an improvement of the DS algorithm, and not only manages to have similar complexity as DS, but can also achieve superior and more robust quality.

2. DIAMOND SEARCH

The Diamond Search algorithm (DS) [6-7] is mainly based on the assumption that motion vectors are in general center biased [5]. The algorithm always starts searching from the center (0,0) of the search area, by examining nine (9) checking points as seen in Figure 1a. If the minimum is found at the center, then four (4) additional checking points (Figure 1b) are examined and the search stops. Otherwise, depending on the position of the current minimum (Figure 1c-d), additional points will have to be examined. By considering the current minimum as the new center of this new large diamond that is created, the process iterates until the minimum is found to be in the center, where again the smaller diamond (the four additional checking points) are examined.

Figure 1: Definition of the Diamond Search (DS) algorithm

As was presented in [7] the algorithm might actually have significant problems when coding non center-biased sequences. It is possible, by introducing an additional predictive step in the algorithm to significantly improve its performance [9].

3. PREDICTIVE DIAMOND SEARCH

In most video coding standards, motion vectors are differentially coded versus a predicted MV since adjacent blocks and their motion vectors are usually very highly correlated. This technique can significantly reduce the bit requirements of the motion vectors and thus increase quality. In MPEG-1/2 this predicted MV is taken as the MV from the previous block, where as in more recent coding standards, such as H.263 and MPEG-4, the median value of the MVs of the three neighboring blocks (Figure 2) is used.

Figure 2: Coding of Macroblock motion vectors in MPEG-4 and H.263

It is rather obvious that the DS algorithm does not take in consideration the high correlation of the neighboring MVs. Instead of examining the center (0,0) of the search area in the first step, we simply modify the DS algorithm, to initially consider as the center of the search area the predicted motion vector, and by using the large diamond pattern (Figure 1a) examine whether our motion vector lies again close to the predictor. If the minimum is found to be the same as the prediction, then the small diamond (Figure 1b) is used and the final four points around the prediction are examined to optimize the final motion vector. This technique significantly improves the correlation between motion vectors, reduces the bits required by them, and thus allowing more bits to be used for coding the error.

Unlike other similar algorithms, if the minimum is not found to be the same as the prediction in the first step, the search moves back to the center (0,0) and the original DS algorithm is used (Figure 3a) until the best match is found in the center of the diamond pattern. This can significantly help in cases that the original prediction was incorrect, such as for example the case of blocks containing edges of objects moving in different directions or stationary backgrounds. If though when moving to the center, after examining the first nine points, the minimum candidate up to now is still a point previously examined around the prediction, in our approach we again move back to the prediction, examine four additional points using the small diamond pattern around the current best match and the search stops (Figure 3b).

Figure 3: Definition of the Predictive Diamond Search (PDS) algorithm

4. SIMULATION RESULTS

We embedded the proposed algorithm in the MPEG-4 VM Encoder and selected several different testing conditions to test its efficiency. In our comparisons, we selected not only the diamond search (DS), but also the three step search (TSS) algorithm, New three step search (NTSS), and their variations using four (4) steps sometimes used for larger search areas, FSS and NFSS respectively. For FSS and NFSS distance of 8, 4, 2, and 1 was used for each step. Finally PSNR and speed up results were compared versus the FS algorithm.

In our first test (Table 1) we selected the MPEG-4 VM5 Q2 rate control with the IPPP... scheme (first frame is an I frame, all others are P frames), and simulation was performed at low bitrates (10kbps-112kbps). Search areas (SAs) of $(-16, +15.5)$ and $(-32, +31.5)$ were used for all cases. For this experiment, five sequences were selected, the first three (*Container*, *Hall Monitor*, and *Mother & Daughter*) being class A sequences (sequences with low and relatively small motion), and the other two (*Coastguard* and *Foreman*) class B sequences (sequences with medium motion).

From our results for the first three sequences, it appears that all algorithms perform approximately the same in terms of output quality. These results however are also significantly affected by the rate control used. In terms of speed up, DS appears to be slightly faster than our proposed algorithm, but still this outperforms all others. For the other two sequences, it can be seen that the proposed algorithm has the highest PSNR. Even though DS again is the fastest algorithm, we need to mention that the performance of this algorithm is very poor in coding these sequences, as can be seen from the loss of about 0.6dB for *Coastguard* and 0.5dB-0.7dB for *Foreman*.

Of interest it might also be the observation that TSS and NTSS perform better than the FSS and NFSS algorithms for these cases. This is actually a result of the increased distance between checking points examined in the first step of the four step algorithms (8 for the four step algorithms versus 4 for the three step algorithms), which, considering the relatively small resolution and average motion of the sequences, makes the outer checking points less likely to be near the candidate motion vectors. In addition the SAD values of these blocks tend to be in most cases larger than the SAD of the center, thus resulting

in the algorithm selecting the center or the points around the center most of the time. The difference also between SA16 and SA32 mainly comes from the different amount of bits used for the motion vectors as defined for these search areas.

The actual difference in performance is more obvious when examining the PSNR of all frames in the entire sequence (Figure 2). In this figure we present the per-frame PSNR results for *Coastguard* at 112kbps for the DS, NTSS, TSS, and PDS algorithms and these are compared with the FS algorithm. It is rather obvious that the proposed algorithm performs very close to FS, where as DS has the worse performance with even more than 1dB loss for several frames.

In Table 2 the MPEG-2 TM5 rate control was selected, again using the MPEG-4 VM encoder, with parameters $M=1$ and $N=15$ (IPP...IP... an I frame every 14 P frame). For this simulation we used the sequences *Foreman* and *Table Tennis* and we examined the performance of our algorithm using search areas of $(-16, +15.5)$, $(-32, +31.5)$, and $(-48, +47.5)$. Similarly to the B class sequences at lower bitrates in Table 1, our algorithm again manages to outperform all other algorithms in terms of PSNR performance. TSS, FSS, NTSS, and NFSS have significant loss in quality in all these cases, where as DS works fairly well in *Table Tennis* but has a significant loss in *Foreman*. Our algorithm manages again the best performance, with a worse loss of 0.36dB in *Foreman* versus 0.77dB of DS. On the average, our algorithm has only 0.14dB loss versus 0.33dB of DS, where as in terms of speed up the algorithm again manages to have the same complexity as DS. In Figure 5 we can again see the per-frame PSNR for all algorithms. What is actually not as apparent in the average PSNR results is that DS and the other algorithms examined actually have a rather severe loss for several frames (170-180, 220-230), which can even be in some cases higher than 4dB.

Finally, in Table 3, four CCIR interlaced sequences, *Basket*, *Bus*, *Flower Garden*, and *Stefan*, were simulated, again using the TM5 rate control, and with a search area of $(-64, 63.5)$. We need to mention that these four sequences are all class C sequences, which means that they contain rather fast and complicated motion. Note also that only progressive motion estimation was used instead of the field motion estimation that could significantly improve the quality of the encoded sequence, and thus making the simulation conditions even harder.

Despite the fact that the proposed algorithm could be said to have a significant loss versus the FS algorithm (0.78dB on the average), it is rather obvious that the algorithm performs significantly better than all other tested algorithms (1.45dB loss for DS). Due to the higher resolution and larger motion, in these cases the FSS and NFSS algorithms perform significantly better than TSS and NTSS, unlike the lower resolutions. It is again interesting that the DS algorithm does not perform as well as either of these algorithms, regardless being the faster algorithm. In Figure 6 we again show the per-frame PSNR for the *Bus* sequence, in which the DS algorithm has a rather significant loss of 4dB, where as the proposed algorithm can perform much better. As in our previous results, the complexity of the proposed algorithm is very similar to that of DS.

Even though the proposed algorithm for these cases does have a significant loss in terms of PSNR versus the FS algorithm, we should mention that this problem should be significantly reduced if field motion estimation is used, mainly due to the increased correlation between blocks in field mode.

5. CONCLUSION

In this paper a new block based motion estimation algorithm was presented, which is a significant improvement to the Diamond Search algorithm. The algorithm, named as Predictive Diamond Search (PDS), uses the same concepts as Diamond Search, but also considers some additional predictive criteria, which can significantly improve performance. Extensive simulation shows that the proposed algorithm has similar complexity to the Diamond Search algorithm, where as it is more robust and achieves significantly better quality. It is possible to improve the algorithm further by adding more criteria (i.e. thresholds), or by considering more candidates in our prediction.

6. ACKNOWLEDGEMENTS

This work was supported by the Hongkong Telecom Institute of Information Technology and the Research Council Grant of Hong Kong Government.

7. REFERENCES

1. S. Fukunaga, Y. Nakaya, S. H. Son, and T. Nagumo, "MPEG-4 Video Verification Model version 14.0", in *ISO/IEC JTC1/SC29/WG11 MPEG99/N2932*, Victoria, Australia, Oct. 1999.
2. T. Sikora, "The MPEG-4 video standard verification model," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 7, pp. 19-31, Feb. 1997.
3. T. Koga, K. Iinuma, A. Hirano, Y. Iijima, and T. Ishiguro, "Motion compensated interframe coding for video conferencing," *Proc. Nat. Telecommun. Conf.*, New Orleans, LA, Dec. 1981, pp. G5.3.1-G5.3.5.
4. J. R. Jain and A. K. Jain, "Displacement measurement and its application in interframe image coding," *IEEE Trans. Commun.*, vol. COM-29, Dec. 1981, pp. 1799-1808.
5. R. Li, B. Zeng, and M. L. Liou, "A new three-step search algorithm for block motion estimation," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 4, no. 4, Aug. 1994, pp. 438-42.
6. S. Zhu and K. K. Ma, "A new diamond search algorithm for fast block matching motion estimation," *Proc. of Int. Conf. Information, Communications and Signal Processing*, vol. 1, pp. 292-6, 1997.
7. J. Y. Tham, S. Ranganath, M. Ranganath, and A. A. Kassim, "A Novel Unrestricted Center-Biased Diamond Search Algorithm for Block Motion Estimation," *IEEE Trans. Circuits Syst. Video Technol.*, Vol. 8, Pp. 369-377, Aug. 1998.
8. A. M. Tourapis, O. C. Au, M. L. Liou, and G. Shen, "Status Report of Core Experiment on Fast Block-Matching Motion Estimation using Advanced Diamond Zonal Search with Embedded Radar," in *ISO/IEC JTC1/SC29/WG11 MPEG99/m4980*, Melbourne, Australia, October'99
9. A. M. Tourapis, O. C. Au, and M. L. Liou, "Fast Motion Estimation using Circular Zonal Search", *Proc. of SPIE Sym. of Visual Comm. & Image Processing, VCIP'99*, Vol. 2, pp. 1496-1504, Jan. 25-27, 1999.

Figure 4: Per Frame PSNR comparison versus FS for Coastguard encoded at 112kbps

Table 1: PSNR and algorithmic complexity using the Q2 rate control algorithm at low bitrates

Sequence	Size	FR	BR	SA		FS	PDS	DS	FSS	TSS	NFSS	NTSS	Difference from FS & Speed Up						
													PDS	DS	FSS	TSS	NFSS	NTSS	
Container	QCIF	7.5	10	16	PSNR	29.81	29.76	29.76	29.81	29.79	29.78	29.79	29.79	-0.04	-0.05	0.01	-0.01	-0.03	-0.01
				Points	7501824	97061	96969	241758	183150	129505	135804	77	77	31	41	58	55		
				32	PSNR	29.72	29.78	29.74	29.75	29.72	29.79	29.77	29.77	0.06	0.02	0.04	0.00	0.07	0.06
				Points	27142090	96906	97030	241758	183150	129666	135909	280	280	112	148	209	200		
Hall Monitor	QCIF	7.5	10	16	PSNR	30.35	30.32	30.32	30.31	30.31	30.30	30.36	30.36	-0.03	-0.03	-0.04	-0.04	-0.05	0.01
				Points	7501824	97058	96927	241758	183150	127032	134353	77	77	31	41	59	56		
				32	PSNR	30.29	30.30	30.29	30.26	30.31	30.27	30.24	30.24	0.01	0.00	-0.04	0.02	-0.02	-0.05
				Points	27142090	97168	97209	241758	183150	126994	134443	279	279	112	148	214	202		
Mother	QCIF	10	24	16	PSNR	34.80	34.77	34.78	34.69	34.81	34.71	34.78	187125	73	74	31	41	57	54
				Points	10036224	136929	135676	323433	245025	176879	34.73	34.73	-0.07	-0.08	-0.14	-0.05	-0.12	-0.08	
				32	PSNR	34.81	34.74	34.73	34.67	34.75	34.69	176918	187158	265	268	112	148	205	194
				Points	36311715	137084	135687	323433	245025	176918	27.96	28.75	-0.04	-0.15	-0.53	-0.11	-0.93	-0.13	
Coastguard	QCIF	10	48	16	PSNR	28.88	28.84	28.73	28.36	28.77	27.96	28.75	28.75	51	58	31	41	45	41
				Points	10036224	196906	171767	323433	245025	221318	247452	28.78	-0.04	-0.19	-0.54	-0.12	-0.96	-0.12	
				32	PSNR	28.90	28.85	28.71	28.35	28.78	27.94	28.78	246868	185	212	112	148	165	147
				Points	36311715	196538	171556	323433	245025	219749	26.66	26.66	-0.07	-0.59	-0.30	-0.32	-0.51	-0.37	
	CIF	10	112	16	PSNR	27.03	26.96	26.44	26.73	26.71	26.52	26.66	26.66	44	49	31	41	34	37
				Points	40144896	915517	811384	1293732	980100	1169789	1075830	26.69	-0.06	-0.59	-0.34	-0.33	-0.52	-0.36	
				32	PSNR	27.06	26.99	26.47	26.72	26.72	26.54	26.69	1168292	167	187	118	156	131	142
				Points	152818083	914876	817306	1293732	980100	1168292	29.54	-0.18	-0.46	-0.72	-0.58	-0.67	-0.50		
Foreman	CIF	10	112	16	PSNR	30.04	29.86	29.58	29.32	29.46	29.37	29.54	1041816	43	43	31	41	37	39
				Points	40144896	925917	931373	1293732	980100	1084613	29.58	-0.40	-0.74	-1.01	-0.85	-0.91	-0.79		
				32	PSNR	30.37	29.97	29.63	29.36	29.52	29.46	29.58	1084733	158	158	118	156	141	147
				Points	152818083	969774	968480	1293732	980100	1084733	1041799	-0.07	-0.24	-0.31	-0.20	-0.39	-0.20		
Average Combined PSNR Difference vs. the Full Search (FS) Algorithm																			
Average Combined Speed Up Gain vs. the Full Search (FS) Algorithm																			
Average PSNR Difference vs. the Full Search (FS) Algorithm for SA16																			
Average Speed Up Gain vs. the Full Search (FS) Algorithm for SA16																			
Average PSNR Difference vs. the Full Search (FS) Algorithm for SA32																			
Average Speed Up Gain vs. the Full Search (FS) Algorithm for SA32																			

Table 2: PSNR and algorithmic complexity using the TM5 rate control algorithm for medium to high bitrates

Sequence	Size	FR	BR	SA	PSNR	FS	PDS	DS	FSS	TSS	NFSS	NTSS	Difference from FS & Speed Up					
													PDS	DS	FSS	TSS	NFSS	NTSS
Foreman	CIF	15	512kbps	16	PSNR	34.51	34.38	34.07	33.80	33.88	33.94	34.02	-0.14	-0.44	-0.71	-0.63	-0.57	-0.49
					Points	56770560	1190877	1207842	1829520	1386000	1432633	1409856	48	47	31	41	40	40
				32	PSNR	34.84	34.51	34.09	33.81	33.90	33.95	34.02	-0.33	-0.74	-1.03	-0.94	-0.88	-0.81
					Points	216106380	1223058	1248552	1829520	1386000	1433131	1410394	177	173	118	156	151	153
Foreman	CIF	30	1024Kbps	48	PSNR	34.88	34.51	34.10	33.83	33.90	33.95	34.02	-0.36	-0.77	-1.05	-0.98	-0.92	-0.86
					Points	461637680	1224383	1251236	1829520	1386000	1432551	1409649	377	369	252	333	322	327
				16	PSNR	35.47	35.30	34.97	34.86	34.79	34.98	34.92	-0.17	-0.50	-0.61	-0.69	-0.49	-0.56
					Points	113541120	2002784	2061548	3659040	2772000	2542857	2553693	57	55	31	41	45	44
Foreman	CIF	30	1024Kbps	32	PSNR	35.53	35.33	34.97	34.87	34.79	34.98	34.91	-0.20	-0.56	-0.66	-0.74	-0.56	-0.62
					Points	432212760	2022015	2082887	3659040	2772000	2542957	2554598	214	208	118	156	170	169
				48	PSNR	35.51	35.32	34.96	34.86	34.78	34.97	34.90	-0.19	-0.54	-0.65	-0.73	-0.54	-0.61
					Points	923275360	2021521	2084544	3659040	2772000	2542812	2553921	457	443	252	333	363	362
Table	SIF	30	1024Kbps	16	PSNR	34.98	34.93	34.92	34.53	34.71	34.70	34.82	-0.05	-0.06	-0.46	-0.27	-0.29	-0.17
					Points	94617600	1435171	1383397	3049200	2310000	1785223	1874533	66	68	31	41	53	50
				32	PSNR	35.00	34.93	34.90	34.51	34.72	34.71	34.83	-0.08	-0.10	-0.50	-0.28	-0.30	-0.18
					Points	358185240	1435793	1386257	3049200	2310000	1784816	1874004	249	258	117	155	201	191
Table	SIF	30	2048kbps	48	PSNR	34.97	34.93	34.92	34.50	34.72	34.68	34.80	-0.04	-0.05	-0.48	-0.25	-0.29	-0.17
					Points	760543840	1437849	1387092	3049200	2310000	1785479	1873789	529	548	249	329	426	406
				16	PSNR	37.95	37.91	37.87	37.51	37.70	37.68	37.80	-0.05	-0.08	-0.44	-0.26	-0.27	-0.15
					Points	94617600	1440307	1386772	3049200	2310000	1783906	1877378	66	68	31	41	53	50
Table	SIF	30	2048kbps	32	PSNR	37.95	37.91	37.87	37.51	37.70	37.68	37.80	-0.04	-0.07	-0.44	-0.24	-0.27	-0.15
					Points	358185240	1442758	1389430	3049200	2310000	1783935	1877244	248	258	117	155	201	191
				48	PSNR	37.94	37.90	37.87	37.51	37.70	37.67	37.80	-0.04	-0.07	-0.43	-0.24	-0.27	-0.14
					Points	760543840	1443898	1390283	3049200	2310000	1784297	1876879	527	547	249	329	426	405
Average Combined PSNR Difference vs. the Full Search (FS) Algorithm																		
Average Combined Speed Up Gain vs. the Full Search (FS) Algorithm																		
Average PSNR Difference vs. the Full Search (FS) Algorithm for SA16																		
Average Speed Up Gain vs. the Full Search (FS) Algorithm for SA16																		
Average PSNR Difference vs. the Full Search (FS) Algorithm for SA32																		
Average Speed Up Gain vs. the Full Search (FS) Algorithm for SA32																		
Average PSNR Difference vs. the Full Search (FS) Algorithm for SA48																		
Average Speed Up Gain vs. the Full Search (FS) Algorithm for SA48																		

Table 3: PSNR and algorithmic complexity when coding 4 Interlaced CCIR sequences (without Field Motion Estimation)

Sequence	Size	FR	BR Mbps	SA	FS	PDS	DS	FSS	TSS	NFSS	NTSS	Difference from FS & Speed Up						
												PDS	DS	FSS	TSS	NFSS	NTSS	
Basket	704x576	30	4	64	PSNR	26.71	26.31	26.01	25.85	25.90	25.69	25.91	-0.40	-0.70	-0.86	-0.81	-1.02	-0.80
				Points	5580207144	9506479	9557434	12127104	9187200	10921322	10624209	587	584	460	607	511	525	
			9	64	PSNR	30.78	30.37	29.97	29.86	29.92	29.61	29.91	-0.41	-0.81	-0.92	-0.86	-1.17	-0.87
				Points	5580207144	9447362	9465762	12127104	9187200	10818681	10603983	591	590	460	607	516	526	
Bus	720x480	30	4	64	PSNR	29.78	28.34	27.15	28.25	27.01	28.11	26.96	-1.44	-2.64	-1.53	-2.77	-1.67	-2.83
				Points	2827221219	4772473	4509336	6192450	4691250	6115846	5192358	592	627	457	603	462	544	
			9	64	PSNR	34.03	32.65	31.18	32.54	31.04	32.37	30.99	-1.38	-2.85	-1.49	-2.99	-1.66	-3.04
				Points	2827221219	4720345	4416942	6192450	4691250	6138446	5191933	599	640	457	603	461	545	
Flower Garden	720x480	30	4	64	PSNR	28.33	28.17	27.88	27.16	27.54	27.02	27.83	-0.16	-0.45	-1.17	-0.79	-1.30	-0.50
				Points	2827221219	4107797	3880328	6192450	4691250	4857673	5116374	688	729	457	603	582	553	
			9	64	PSNR	33.17	33.05	32.76	31.85	32.26	31.82	32.68	-0.12	-0.41	-1.32	-0.91	-1.36	-0.49
				Points	2827221219	4101335	3870558	6192450	4691250	4831506	5110170	689	730	457	603	585	553	
Stefan	720x480	30	4	64	PSNR	30.77	29.43	28.71	29.12	28.51	28.97	28.48	-1.33	-2.06	-1.65	-2.26	-1.79	-2.29
				Points	5695121880	8908569	8609388	12474000	9450000	10529637	9724654	639	662	457	603	541	586	
			9	64	PSNR	34.97	33.98	33.26	33.60	33.11	33.50	33.09	-0.99	-1.71	-1.37	-1.86	-1.47	-1.88
				Points	5695121880	8840175	8525897	12474000	9450000	10444285	9682237	644	668	457	603	545	588	
Average PSNR Difference vs. the Full Search (FS) Algorithm													-0.78	-1.45	-1.29	-1.66	-1.43	-1.59
Average Speed Up Gain vs. the Full Search (FS) Algorithm													629	654	457	604	525	553

Figure 5: Per Frame PSNR comparison versus FS for Foreman encoded at 512kbps

Figure 6: Per Frame PSNR comparison versus FS for Bus encoded at 4000kbps