Circuits and Systems, 2010, 1, 12-17
doi:10.4236/cs.2010.11003 Published Online July 2010 (http://www.SciRP.org/journal/cs)

Fast Implementation of VC-1 with Modified Motion Estimation and Adaptive Block Transform

Michael Tammen[1], Mohamed El-Sharkawy[2], Hisham Sliman[1], Maher Rizkalla[1]
[1]*Purdue School of Engineering and Technology*, *Indianapolis*, *USA*
[2]*Egypt Japan University of Science and Technology*, *Borg Elarab*, *Alexandria*, *Egypt*
E-mail: *melshark@iupui.edu*
Received March 18, 2010; *revised April* 20, 2010; *accepted April* 25, 2010

Abstract

The Society of Motion Picture and Television Engineers (SMPTE) Standard 421M, commonly known as VC-1, is a state-of-the-art video compression format that provides highly competitive video quality, from very low through very high bit rates, at a reasonable computational complexity. First, this paper presents fast motion compensation methods. The four motion estimation methods examined are fast, three step search, varying diamond, and 2D logarithmic. These methods use less search points than the full spiral scan used in the VC-1 reference software, which allows for faster motion estimation. Second, this paper presents a residual texture based choice of the block size for the Discrete Cosine Transform (DCT). To determine the block size, data is examined after the residual texture has been calculated. This is in contrast to the VC-1 reference software, which uses calculations at the block level to determine the block size. The residual texture of each block is small and uniform, allowing for simplified block choices.

Keywords: VC-1, Motion Estimation, Discrete Cosine Transform, Video Compression

1. Introduction

VC-1 is a state-of-the-art video compression format that provides highly competitive video quality, from very low through very high bit rates, at a reasonable computational complexity [1-4]. At a high level, VC-1 is similar to other popular video standards since the First Moving Picture Expert Group Standard (MPEG-1). They have similarities in many different areas, one of which is block-by-block motion compensation. The motion compensation is done using a motion vector from a previously reconstructed frame to determine the displacement. On the decoder side, quantized transform coefficients are entropy-decoded, dequantized, and inverse-transformed to produce an approximation of the residual error, which is then added to the motion-compensated prediction to generate the reconstruction [5].

An important feature of VC-1 is adaptive block size transforms for inter-frame coding. Instead of using a fixed block size for every transform, like many previous standards, it has the ability to choose a transform size based on the information in the block or macroblock. VC-1 can choose 8 × 8, 8 × 4, 4 × 8, or 4 × 4 transforms based on the information in the block. The ability to choose a transform size allows VC-1 to deal with areas of continuity and discontinuity with more accuracy and less ringing artifacts than any other video standard [1]. An 8 × 8 block may be transformed by either one 8 × 8 block, two horizontally stacked 8 × 4 s, two vertically stacked 4 × 8 s, or four 4 × 4 blocks. The block sizes can be seen in **Figure 1**.

Another key innovation in VC-1 is the handling of an entire zero block or macroblock. When it happens, it does not send any transform information along, since the inverse transform of a zero block is going to be zero. Intra frames and intra blocks in predicted frames use 8 × 8 transform by default, meaning no calculations need to be preformed [5].

The way VC-1 signals the transform size is new and unique. It can either be signaled at the frame, macroblock, or block level. For frame level signaling, every block within the frame will use the same block size. Frame level signaling is useful for low-rate situations to keep the overhead low. If macroblock level signaling is used, then

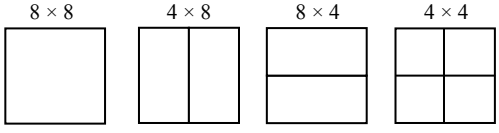

Figure 1. Transform sizes.

every block in the macroblock (six 8 × 8 blocks) is transformed using the same size. Block level signaling is used only for one 8 × 8 block. These different signaling types allow VC-1 to handle both nonstationary data (macroblock and block level signaling) and low-rate situations (frame level signaling) better than previous standards.

Currently, VC-1 uses half difference and half sum energies for the transform size decision for a block. It splits the block into four quadrants: left, right, top, and bottom. The left-right half difference, the left-right half sum, the top-bottom half difference, and the top-bottom half sum are then computed. Each of the above values is a running total for all the values in the quadrant.

To get the block size, it compares the left-right sum to the left-right difference and determines to chop vertically or not. Then it compares the top-bottom sum and top-bottom difference to determine if it should chop horizontally. If both conditions are met a 4 × 4 is used, while if none are met an 8 × 8 is used.

Motion estimation (ME) is the process of comparing the macroblock to be coded with all macroblocks within the search area in the reference frame [6-10]. The macroblock in the reference frame which is closest to the macroblock to be coded is chosen as the reference macroblock. Once it is chosen, a motion vector (MV) is assigned. The motion vector provides the coordinates of the best match in an (x, y) form, using the center of the search area as (0, 0). An illustration can be seen below in **Figure 2**.

2. Motion Estimation Techniques

The VC-1 reference software currently uses the full spiral search shown in **Figure 3** (a search size of 4 is used in the figure) to find the best match. A best match is found by calculating the cost at each location. The spot with the lowest cost is chosen as the best match. This is currently the only method for motion estimation within VC-1 reference software. It does allow for the search size to be modified but that is all.

The first implemented method was the fast motion estimation. It is based on the H.264 fast motion estimation but is not completely similar. It calculates the cost of each position in a diamond shape. The size of the diamond grows or shrinks based on the search size. The pattern is shown below in **Figure 4** with a search size of 4. Also, the cost for the center is calculated before this search starts and if no cost in the fast search is lower than the center, then the center is chosen.

Second, we implemented a three or four step search. **Figure 5** shows the search pattern for this search. First,

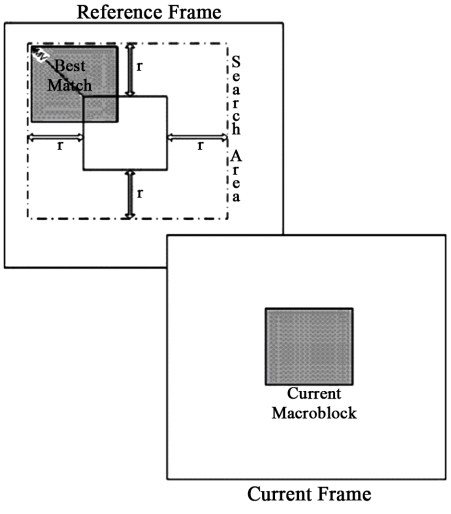

Figure 2. Motion estimation [11].

	-4	-3	-2	-1	0	1	2	3	4
4	74	75	76	77	78	79	80	81	**50**
3	73	44	45	46	47	48	49	**26**	51
2	72	43	22	23	24	25	**10**	27	52
1	71	42	21	8	9	**2**	11	28	53
0	70	41	20	7	**1**	3	12	29	54
-1	69	40	19	6	5	4	13	30	55
-2	68	39	18	17	16	15	14	31	56
-3	67	38	37	36	35	34	33	32	57
-4	66	65	64	63	62	61	60	59	58

Figure 3. Full spiral search.

	-4	-3	-2	-1	0	1	2	3	4
4					×				
3				×		×			
2			×				×		
1		×						×	
0	×								×
-1		×						×	
-2			×				×		
-3				×		×			
-4					×				

Figure 4. Fast motion estimation.

	-6	-5	-4	-3	-2	-1	0	1	2	3	4	5	6
6									2		2		2
5													
4			1				1		2		**1**		2
3								3	3	**3**			
2								3	**2**	3	2		2
1								3	3	3			
0			1				0				1		
-1													
-2													
-3													
-4			1				1				1		
-5													
-6													

Figure 5. Three step search.

the cost is computed at the center (0) and eight positions (1) around the center. The positions distance from the center is calculated by 2^{N-1}, where N is 3 if the search size is less than or equal to 12, or 4 if the search size is greater than 12. This allows for better accuracy when the search size is large, because the normal three step search will only be able to search about half of the search area if the search size is greater than 12. The least cost position is chosen (**1**) and N is decremented. The cost is then calculated at eight positions (2) around **1**. This again allows for another least cost position to be picked (**2**) and N is decremented again. Lastly, eight positions (3) have their cost calculated around **2** and the least cost position is chosen as the best match, **3**. A four step search would simply have one more step and works the same way. Obviously, the best match is not always going to require three or four steps, so if at the end of any step the best cost belongs to the center of the search (0, **1**, and **2** for reference) the search terminates. For example, if the cost of 0 is less than the cost at each of the eight positions (1) then the search terminates and 0 is chosen as the best match.

Third, a 2D log search was implemented. **Figure 6** is an illustration of how the log search works. To determine the distance from the center in which the search is started, d = floor $(2 \times (\log_2 S - 1))$, is computed, where S is the search size. The cost is then calculated at the center (0) and four other positions (1) surrounding it. The position with the least cost is chosen (**1**) and d remains the same. Next, the cost is computed at the three surrounding positions (2) and if their cost is not lower than the center then d is halved. If d is an odd number, 1 is subtracted from it and then it is halved.

The last search we implemented was the varying diamond search. This borrows ideas from the three step search, but instead of using eight positions it uses four. The option of a three step or a four step is allowed in this search as well. **Figure 7** illustrates what a varying diamond search will look like when it uses three steps. The bold positions are those with the lowest cost, and the lower the number, the earlier it is searched.

	-6	**-5**	**-4**	**-3**	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**
6													
5													
4													
3							1						
2													
1													
0				1			0			1			
-1													
-2							3	**4**					
-3				2		3	**1**	**3**	**4**	**2**			
-4							3	**4**					
-5													
-6							2						

Figure 6. 2D log search.

	-6	**-5**	**-4**	**-3**	**-2**	**-1**	**0**	**1**	**2**	**3**	**4**	**5**	**6**
6													
5													
4							1						
3													
2			2										
1													
0	2		**1**		2		0				1		
-1			3										
-2		3	**2**	3									
-3			**3**										
-4							1						
-5													
-6													

Figure 7. Varying diamond search.

3. Modified Adaptive Block Size Transform

To go along with the motion estimation, a modified adaptive block size transform algorithm is implemented. The current method for choosing a transform size has some drawbacks. Multiple calculations are performed at the block level, which is very costly.

Through research, we discovered that the block size is dominated by the residual, which is the prediction error [12]. Using (1), the residual data is computed. The texture that results is small in magnitude and uniform. Compared to the original data, there is a large difference since the original data may not be uniform. Since the data is uniform the block size selection is simplified.

We assume that motion estimation has been performed. The algorithm is as follows:

1) Initialize residual texture threshold T_8 and T_4 for 8 × 8 and 4 × 4 blocks. We perform motion estimation for the current macroblock to get the residual. We assign T_8 and T_4 to 250 and 50 respectively.

2) Partition the macroblock into 24 (4 × 4) blocks.

3) Calculate the residual block texture of each 4 × 4 block using Equation (1).

$$C_{m,n} = \frac{1}{16}\sum_{i=0}^{3}\sum_{j=0}^{3}\left(x_{i,j} - \bar{x}\right)^2 \quad (1)$$

where m is the number of 8 × 8 blocks and n is 4 × 4 block number in the 8 × 8 block, beginning in the upper-left and moving right.

4) Sum all the 4 × 4 blocks' texture in an 8 × 8 block as the 8 × 8 block's texture. The macroblock texture is the sum of all 6 of the 8 × 8 blocks' texture.

$$C_m = \sum_{n=0}^{3} C_{m,n} \quad (2)$$

$$C_{MB} = \sum_{m=0}^{3} C_m \quad (3)$$

5) Create a count of each 4 × 4 residual block whose texture is greater than the threshold T_4 and record it in

*count*8. Also track the position of each block that is over the threshold in the variable *Track_b*4. *Track_b*4 contains the value of 1-4 based on which is the most recent block to be over the threshold value.

6) The block size selection is done through the following method:

a) If *count*4 = 0, 8 × 8 is chosen for the block size.

b) If *count*4 > 1 and $C_m > T_8$, 4 × 4 is chosen as the block size.

c) If *track_b*4 = 0 or 3 and $C_1 >= C_2$ choose 4 × 8, otherwise choose 8 × 4.

d) If *track_b*4 = 1 or 2 and $C_3 <= C_0$ choose 8 × 4, otherwise choose 4 × 8.

The above procedure is carried out in two different steps in the same subroutine. First, a subroutine is called that partitions the macroblock and carries out all calculations. This includes calculating the residual texture for the macroblock, as well as the sum of all the blocks' texture. The code then enters a loop, which is executed 6 times, once for each block in the macroblock, where the transform size is selected and the block is transformed. The subroutine where the transform size is selected does not involve any calculations like the standard. It simply gets a pointer to the residual texture results and performs step 6. Not performing any calculations at the block level allows for greater speed.

4. Results

The motion estimation techniques and modified adaptive block size transform were implemented on the VC-1 sample encoder. A variety of video clips were chosen to make sure that both changes work on different types of video sequences. Some of the sequences have low detail and low movement, while others have medium detail and low movement or medium movement and low detail. All sequences are run 30 fps, and can be seen in **Table 1**.

The cases covered in **Table 1** give varieties of video clips that range from low to high movement and low to high details. More frames are needed for higher movement to cover information depth between frames. **Table 2** shows the key configurations we used for the encoder. It is important to note that the modified adaptive block size transform will run only if the option 'BlockType' is set to 'Any'. If 'random' or 'iterated' is set in the option file, then the subroutine is never called by the encoder due to the transform size being set. We tested all sequences with a search size of 8 and 14, hence both numbers below. These were two separate tests and will be noted later. All of the other important parameters were set to 'random' or 'iterated'.

All sequences were tested to determine the time taken for the motion estimation and adaptive block size transform, the signal-to-noise ratio (SNR) for the luminance and chrominance components, and the bitrate. The results of the motion estimation changes are compared against those of the standard. **Table 3** summarizes the results with a search size of 8, while **Table 4** summarizes the results with a search size of 14. Both tables use percentages to show how much faster the proposed algorithms are than the standard. It is noted that a speed improvement of up to 38% for search size equals to 14 and 18% for search size equals to 8.

Table 1. Video sequences tested.

Index	Sequence	Frames
1	Akiyo (CIF)	300
2	Coastguard (CIF)	300
3	Foreman (QCIF)	300
4	Stefan (CIF)	300
5	Flower (CIF)	250
6	Carphone (QCIF)	90
7	Football (CIF)	90
8	Erik (CIF)	50

Table 2. Key encoder options.

Parameter	Value
Profile:	Advanced
Level:	L0
Frame Pattern:	I:P:P:P
SearchSize:	8 or 14
DQuant:	1
MVRange:	128 × 64
BlockType:	Any
Quantizer:	Explicit

Table 3. Time results with a search size of 8.

Sequence	Percent Saving (%)			
	Fast	TSS	Diamond	2DLog
1	12.31	12.38	13.46	13.04
2	23.88	24.55	25.73	24.68
3	15.04	15.42	16.11	15.48
4	21.78	22.22	22.88	21.72
5	19.75	20.80	21.97	21.08
6	10.03	11.24	10.67	10.97
7	21.81	22.84	23.82	21.95
8	13.86	13.77	14.54	14.02
Average	**17.31**	**17.90**	**18.64**	**17.87**

Table 4. Time results with a search size of 14.

Sequence	Percent Saving (%)			
	Fast	TSS	Diamond	2DLog
1	32.75	33.53	34.45	34.39
2	44.10	45.60	46.68	46.35
3	33.17	34.23	35.04	34.63
4	42.05	43.30	44.23	43.56
5	40.67	42.68	43.58	43.07
6	27.64	28.55	28.87	29.30
7	43.28	44.74	45.87	45.03
8	32.12	32.85	33.98	33.53
Average	**36.97**	**38.18**	**39.09**	**38.73**

Tables 3 and **4** show that the diamond search is the strongest performer followed closely by the 2D log search and three step searches. The fast motion estimation is the worst performer of the group. So strictly from a speed aspect, the diamond search is the best choice.

Next, we will examine the signal-to-noise ratio (SNR) results for search sizes of 8 and 14. **Table 5** shows the SNR for the reference software. **Tables 6** and **7** show the percent degradation in the implemented algorithm for search sizes of 8 and 14 respectively. The negative values in the tables show that the SNR is better in the implemented algorithm.

The VC-1 encoder in its present state has an overflow issue with some of the sequences. The most reliable data comes from sequences 1, 3, and 6 due to little or no overflow occurring. Sequence 1 is the most reliable, as no overflow occurs during the encoding of this sequence. Therefore, using sequence 1 as the baseline we see that there is a very minimal SNR degradation. But, on average all searches see an improvement in SNR.

Lastly, we will look at the bitrates for all of the sequences. The bitrate numbers can get rather large so we examine the percent degradation numbers. Once again if the number is negative that means the proposed algorithm is better than the standard. **Tables 8** and **9** show the results below.

Aside from sequence 6 there is a very minimal effect on the bitrate. There is a range of both small increases from the standard as well as small decreases, neither of which are above 1% (excluding the 1.79% in sequence 4). As for sequence 6, the bitrate is 9% larger than the standard. The reason of such result will be studied in future work.

5. Conclusions

In this paper, we presented four different motion estimation techniques and a fast block size selection tool for

Table 5. SNR results for the standard with both search sizes.

Sequence	SNR (dB)					
	Search Size 8			Search Size 14		
	Y	U	V	Y	U	V
1	21.31	20.78	17.07	21.31	20.78	17.07
2	19.30	25.72	24.88	19.29	25.71	24.90
3	15.39	23.81	24.10	15.39	23.81	24.10
4	24.89	35.04	36.05	24.84	35.07	35.91
5	28.11	22.78	21.37	28.09	23.45	22.65
6	10.06	18.51	16.24	10.06	18.52	16.25
7	22.34	26.75	26.06	22.34	26.95	26.08
8	18.64	20.18	21.50	18.64	20.18	21.52

Table 6. SNR results with a search size of 8.

Sequence	Percent Degradation (%)											
	Fast			TSS			Diamond			2DLog		
	Y	U	V	Y	U	V	Y	U	V	Y	U	V
1	0.28	–0.27	–0.16	0.16	–0.11	–0.09	0.20	–0.17	–0.16	0.19	–0.16	–0.23
2	–2.48	–2.10	0.76	–2.56	–1.04	1.36	–3.30	–1.20	1.59	–3.52	–1.71	–1.01
3	0.48	–0.71	–1.17	0.20	–0.46	–0.83	0.34	–0.48	–0.85	0.33	–0.52	–0.90
4	–5.70	–4.51	–3.79	–6.77	–3.92	–3.93	–6.13	–5.02	–4.30	–5.83	–4.54	–2.91
5	–2.82	0.50	–2.00	–2.92	0.68	0.75	–2.94	0.25	0.73	–2.91	0.80	0.83
6	0.55	0.31	–0.98	0.04	0.70	–0.60	0.13	1.04	–0.96	0.17	0.52	–0.53
7	–0.58	2.15	2.02	–0.88	3.11	–1.43	–0.65	1.74	–1.61	–0.24	0.96	0.27
8	–4.08	–1.46	–3.91	–3.99	–1.47	–3.56	–3.74	–0.37	–2.69	–3.81	0.11	–2.46
Average	**–1.79**	**–0.76**	**–1.15**	**–2.09**	**–0.31**	**–1.04**	**–2.01**	**–0.53**	**–1.03**	**–1.95**	**–0.57**	**–0.87**

Table 7. SNR results with a search size of 14.

Sequence	Percent Degradation (%)											
	Fast			TSS			Diamond			2DLog		
	Y	U	V	Y	U	V	Y	U	V	Y	U	V
1	0.27	–0.13	–0.28	0.17	–0.02	0.15	0.20	–0.20	–0.20	0.20	–0.47	–0.28
2	–4.63	–0.98	0.82	–2.91	–1.37	1.40	–2.91	–1.07	1.44	–2.86	–1.18	1.10
3	0.42	–0.52	36.36	0.21	–0.50	–0.83	0.33	–0.53	–0.85	0.42	–0.52	36.36
4	–7.45	–3.90	26.66	–6.87	–3.75	–4.28	–4.94	–4.44	–3.56	–7.45	–3.90	26.66
5	–3.24	3.08	–27.7	–2.99	3.66	6.33	–3.01	3.32	6.17	–3.24	3.08	–27.7
6	0.29	0.58	38.17	0.21	0.65	–0.76	0.32	0.55	–0.90	0.29	0.58	38.17
7	–0.11	1.81	14.13	–0.50	1.83	0.43	–0.76	2.68	–1.20	–0.11	1.81	14.13
8	–3.87	–0.08	10.08	–4.30	–0.88	–3.54	–3.93	–1.55	–3.32	–3.87	–0.08	10.08
Average	**–2.29**	**–0.02**	**12.28**	**–2.12**	**–0.05**	**–0.18**	**–1.84**	**–0.16**	**–0.30**	**–2.08**	**–0.09**	**12.32**

Table 8. Bitrate results with a search size of 8.

Sequence	Percent Degradation (%)			
	Fast	**TSS**	**Diamond**	**2DLog**
1	0.34	0.20	0.09	0.11
2	–0.24	–0.25	–0.25	–0.29
3	–0.01	–0.01	0	0.01
4	0.01	–0.07	0	0
5	–0.27	–1.47	–0.03	0.26
6	9.68	10.15	8.81	8.37
7	0.09	0.05	0.26	0.06
8	0.49	0.49	0.37	0.48
Average	**1.26**	**1.14**	**1.16**	**1.13**

Table 9. Bitrate results with a search size of 14.

Sequence	Percent Degradation (%)			
	Fast	**TSS**	**Diamond**	**2DLog**
1	0.33	0.22	0.12	0.13
2	–0.06	–0.08	–0.08	0.16
3	–0.01	0	–0.01	–0.01
4	0.21	–0.47	–0.48	1.79
5	–0.51	–1.70	0.02	0.02
6	9.90	10.11	8.77	8.02
7	–0.28	–0.31	–0.29	–0.28
8	0.40	0.40	0.20	0.18
Average	**1.25**	**1.02**	**1.03**	**1.25**

VC-1. Finding an accurate and fast set of motion vectors is very important to any video codec. Also, adaptive block size transforms are a vital part of any next generation video standard. Overall, the best performer was the diamond search which was followed closely by the three and four step search. Both were fast with the diamond having an 18.64% and 39.09% increase in speed with a 0.98% increase in the SNR. The three and four step search had a 17.90% and 38.18% increase in speed with a 0.96% increase in the SNR. Both exhibited an increase in bitrate around 1.10%, which is mainly due to sequence 6. Otherwise they would be much closer to 0.

6. References

[1] S. Srinivasan and S. L. Regunathan, "An Overview of VC-1", *Proceeedings of SPIE*, *VCIP*, Beijing, July 2005, pp. 720-728.

[2] Proposed SMPTE 421M, "VC-1 Compressed Video Bitstream Format for Decoding Process," 2006. http://www.smpte.org

[3] Proposed SMPTE RP 227, "VC-1 Bitstreams Transport Encoding," 2007. http://www.smpte.org

[4] Proposed SMPTE RP 228, "VC-1 Decoder and Bitstreams Conformance," 2008. http://www.smpte.org

[5] S. L. Regunathan, A. M. Rohaly, R. Crinon and P. Griffis, "Quality and Compression: The Proposed SMPTE Video Compression Standard VC-1," *SMPTE Motion Imaging Journal*, Vol. 114, No. 5-6, 2005, pp. 194-201.

[6] M. Ghanbari, "The Cross-Search Algorithm for Motion Estimation," *IEEE Transactions on Communications*, Vol. 38, No. 7, 1990, pp. 950-953.

[7] P. I. Hosur and K. K. Ma, "Motion Vector Field Adaptive Fast Motion Estimation," *Presented at the* 2*nd International Conference on Information*, *Communications and Signal Processing*, Singapore, December 1999, pp. 7-10.

[8] R. Li, B. Zeng and M. Liou, "A New Three-Step Search Algorithm for Block Motion Estimation," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 4, No. 4, 1994, pp. 438-442.

[9] L. K. Liu and E. Feig, "A Block-Based Gradient Descent Search Algorithm for Block Motion Estimation in Video Coding," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 6, No. 4, 1996, pp. 419-422.

[10] L.-M. Po and W.-C. Ma, "A Novel Four-Step Search Algorithm for Fast Block Matching," *IEEE Transactions on Circuits and Systems for Video Technology*, Vol. 6, No. 3, 1996, pp. 313-317.

[11] S. T. Samant and M. El-Sharkawy, "Modified Motion Vector Searches for H.264/AVC," *International Conference on Computer Engineering & Systems* (ICCES'06), Cairo, Egypt, November 2006, pp. 331-336.

[12] Z. Wang, Q. Peng and Y. Zeng, "Residual Texture Based Fast Block-Size Selection for Inter-Frame Coding in H.264/AVC," *IEEE Proceedings of the* 6*th International Conference on Parallel and Distributed Computing Applications and Technologies*, Dalian, 2005, pp. 853-855.

