Chapter 3

Decoder Issues

The previous chapter introduced matching pursuit video compression, but focused mainly on encoder issues. In this chapter, we take a closer look at decoder issues. Specifically, we compare both coding efficiency and decoder complexity for the matching pursuit system to that of the DCT-based MPEG-4 standard with and without post-processing. We show that in order to achieve reasonable quality at low bit rates, the DCT-based MPEG-4 decoder requires post-processing, while the matching pursuit scheme does not. Furthermore, we show that DCT post-processing may introduce a prohibitive cost at the decoder. For example, the MPEG-4 post-processing filters are shown to increase decoder complexity by a factor between 3 and 8, depending on sequence and bit rate. Finally, an all-integer matching pursuit implementation is developed and the decoder implications are discussed. Performance is shown to be within 0.05 dB of an otherwise identical floating point matching pursuit system.

3.1 Introduction

As mentioned previously, most commercial and all standardized video compression systems are based on motion-compensated DCT. At low bit rates, such systems show noticeable block edges and ringing artifacts. One solution is to introduce post-filtering to correct the artifacts. This is the method used by the DCT-based MPEG-4 video verification model (VM), which includes optional de-blocking and de-ringing filters [35]. These filters were turned on in the results shown previously in Chapter 2.

A disadvantage of postfiltering is that coded detail may be removed along with the unwanted artifacts. To mitigate this problem, intelligence may be built into the filtering algorithm. In the case of the MPEG-4 filters, decision criteria are used to adapt the filters to the strength of the artifacts and the amount of underlying detail. Filtering performance is in this way improved. However, the decision process and the filtering together may be a significant computational burden to the decoder. Matching pursuit video compression avoids this problem by using an overcomplete dictionary which does not introduce structured artifacts into the coded images. Redundancy in the dictionary allows each coded atom to closely match the original signal, and the structure of the basis set is not imposed on the coded images. Postprocessing is thus unnecessary, and a high level of coded detail can be preserved.

Matching pursuit video compression is thus ideal for applications in which video is encoded once and decoded many times. This includes most internet video content, as well as video server and broadcast applications. Decoder complexity is critical in these cases, since the end user must decode and display the content in real time. Compression efficiency is also important, since the coded content must be stored and may be transmitted over low-bandwidth channels such as telephone lines or wireless channels. This chapter will show that the matching pursuit encoder outperforms the post-processed MPEG-4 system on these two points. However, it is important to note that the matching pursuit encoder complexity is higher than that of a hybrid-DCT encoder [45], and so applications which require real-time encoding may be less appealing. We defer the issue of fast encoder implementations until Chapter 6.

We begin by discussing decoder complexity for both matching pursuit and DCT-based systems in the next section. Complexity and coding efficiency results are then shown in Section 3.3. Finally, an integer matching pursuit implementation is presented in Section 3.6, and the decoder implications are discussed.

3.2 Decoder Complexity

We will consider the complexity of the following three decoders:

1. DCT-based MPEG-4 Verification Model (VM)
2. DCT-based MPEG-4 VM with postfiltering
3. Matching Pursuit

The complexity of each decoder will be presented in terms of operation counts. Each addition, multiplication, division or bit-shift is counted as a single operation. Memory access and program control are ignored. The operation counts are thus theoretical, but are based on statistics from actual coding/decoding runs. The following subsections provide more details on how each portion of decoder complexity is counted.

3.2.1 Matching Pursuit Decoder Complexity

The complexity of the matching pursuit decoder depends on the number of atoms used to code the residual image, and also on the spatial extent of each atom. We use the separable 2-D Gabor dictionary pictured in Figure 2.2. During decoding, atoms are added to the reconstructed image using a direct sum approach. Each significant pixel of each decoded atom is weighted by the quantized modulus value and then added to the image. Thus two operations per atom-pixel are required, with the number of atom-pixels depending on the extent of the atom. Matching pursuit decoding complexity is thus computed by summing this cost over all coded atoms in the given frame.

3.2.2 Inverse DCT Complexity

An optimized Inverse DCT (IDCT) algorithm presented in [19] is capable of performing a full 8×8 inverse DCT in 522 operations. However, even better performance may be achieved for DCT blocks which contain mostly zero coeficients, since for these blocks direct basis summation may be less expensive than a general IDCT algorithm. This method is equivalent to the direct sum approach used in matching pursuit decoding. Each 8×8 DCT basis function with a nonzero coeficient requires two operations per pixel to add the function into the reconstructed image. To compute the IDCT complexity, we thus choose the cheapest IDCT method for each block and sum the resulting complexities across all blocks. The method used for each block may be chosen based on the number of nonzero coefficients, as shown in Table 3.1.

Table 3.1: IDCT complexity as a function of the number of coefficients.

Coefs (N)	IDCT Method	Complexity
0	None	0
1 to 4	Direct Sum	$N \times 128$
5 to 64	Full IDCT	522

3.2.3 Motion Decoder Complexity

The same motion coding and decoding method is used for both the matching pursuit and DCT-based MPEG-4 systems. Details can be found in the MPEG-4 VM specification [35]. Depending on the phase of the decoded motion vector, half-pixel interpolation may be used to decode both luma and chroma motion blocks. In addition, an overlapping block motion model is used for all luma blocks. To determine the complexity related to motion decoding, we count the number of operations needed to compute all decoded prediction blocks in the luma and chroma subimages. These counts are based on statistics from a real decoder implementation.

3.2.4 DCT Post Processing

For a measure of the complexity of post-processing, we add up the operations needed for the de-blocking and de-ringing filters. Operation counts are again based on actual coding/decoding runs. We will now provide a brief summary of the filters. A complete specification of these may be found in Sections 15.3.1 and 15.3.2 of [33].

The de-blocking filter is applied first across the vertical and horizontal DCT block boundaries in both luma and chroma subblocks. For each filter location, a test is applied to classify the location as either smoothly varying or detailed. If the test indicates a smooth region, a strong smoothing filter is applied to four pixels on each side of the DCT block boundary. If the test finds a region with coded detail, then a weaker "default filter" is applied. This filter only affects one pixel on each side of the boundary. The filter thus adapts to the local properties of the coded image so that block artifacts may be removed without destroying coded details.

A separate de-ringing filter is also applied. This filter is a 3×3 smoothing kernal applied adaptively after de-blocking. For each luma and chroma DCT block, a threshhold is computed which is normally the midpoint of the intensity range of the block. Each intensity value in the block is compared to this threshhold and a binary mask is generated. The mask contains a one if the corresponding pixel is above the threshhold, and a zero if the pixel is below threshhold. The filter kernal is then applied to only those pixels whose surrounding 3×3 region in the binary mask contains all zeros or all ones. A clipping rule is invoked so that no pixel may be changed by more than half the quantization stepsize.

3.3 Results

We now show coding results for the matching pursuit and MPEG-4 VM systems. Test sequences are taken from Table 1.2, and test conditions are identical to those used earlier in Section 2.4. However, we now show separate results for the MPEG-4 VM with and without postfiltering. Section 3.4 begins with an examination of the decoder complexity for each system. This allows a complexity comparison between matching pursuit and DCT-based decoders, and also allows the complexity cost of the postfilters to be examined. Section 3.5 shows the coding efficiency of each system in terms of PSNR and visual performance. This allows the benefits of postfiltering to be weighed against the costs.

3.4 Decoder Complexity

Table 3.2 shows a breakdown of the complexity of the two decoders. The first two columns show the operation count per frame needed to perform inverse Matching Pursuit (IMP) and inverse DCT (IDCT) transforms. The numbers show that these two inverse transforms require a similar amount of computation across the range of bit rates tested. The next column shows the complexity required for motion prediction image decoding. Motion decoding is much more complex than either transform inversion; it requires between four and seventy times the operation count of the IDCT, depending on sequence and bit rate. This is due to a combination of overlapping block motion compensation and half-pixel interpolation [35], each of which can be thought of as a filtering operation. We thus expect motion decoding to dominate the decoder complexity for both Matching Pursuit and MPEG-4 systems in the absence of post-processing.

When post-processing is used for the MPEG-4 decoder, the rightmost columns of Table 3.2 become relevant. These represent the complexity of the de-blocking and de-ringing filters specified in the MPEG-4 VM. The filters are a significant computational expense; when combined they require from three to seven times the computation of the motion decoder. Thus when the post-processing filters are included, they can be expected to dominate the computation of the MPEG-4 decoder.

Note that the complexity of the inverse matching pursuit and inverse DCT transforms generally increase with the bit rate, while the remaining columns show more depen-

Table 3.2: Itemized Decoder Complexity (Ops per frame)

Seq	Label	IMP	IDCT	Motion	De-block	De-ring
1	Cont-10	8705	11805	213391	856858	737079
2	Hall-10	8877	10081	202929	832574	698876
3	Mom-10	11744	8178	264768	874968	714539
4	Cont-24	18444	24798	204479	818782	725544
5	Silent-24	27003	19956	264265	721998	607235
6	Mom-24	25853	22685	257842	837196	704067
7	Coast-48	53336	49867	368900	762415	616044
8	Fore-48	60881	40136	396848	776557	671555
9	News-48	51287	42075	1005182	3719816	3139131
10	Coast-112	73109	31605	1495243	3675608	2936685
11	Fore-112	74723	20914	1498977	3850242	3047989
12	News-112	72723	68121	938580	3623375	3103931
13	Mob-1024	175558	339340	1258895	1735694	1841017
14	Stef-1024	351291	319043	1218875	2280845	1976359

dence on image size. This is because Motion, De-block, and De-ring are all block based with a roughly constant complexity per image block. As a result, each of these three columns shows similar per-frame complexity over the set of QCIF sequences (1-8), and much larger complexity for CIF and SIF sequences (9-14).

Table 3.3 shows operation count totals for the complete decoder systems. Note that the matching pursuit and DCT based MPEG-4 VM decoder have nearly identical complexity in the absence of post-processing. The column labelled "VM+PP" shows the complexity of the VM decoder with the de-blocking and de-ringing filters added. The post-processing takes between seventy and ninety percent of the total complexity, as shown in the column labelled "VM %PP." The final column shows the complexity ratio between postprocessed DCT and matching pursuits. The postprocessed DCT is between three and eight times the complexity of the matching pursuit decoder. The operation count totals are

Table 3.3: Total Decoder Complexity Per Frame

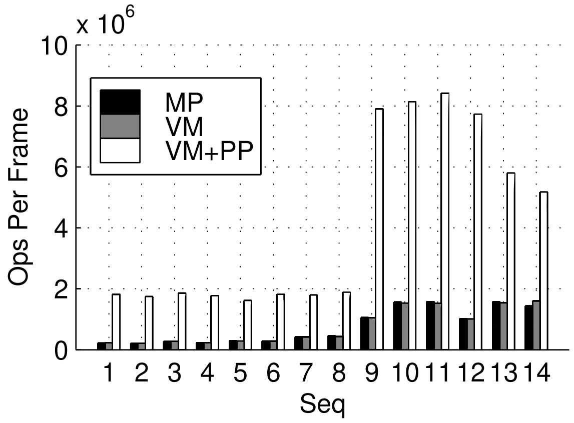
Seq	Label	Millions of Ops MP	VM	VM+PP	VM %PP	(VM+PP)/ MP
1	Cont-10	0.222	0.223	1.819	87.62	8.19
2	Hall-10	0.212	0.213	1.745	87.79	8.24
3	Mom-10	0.277	0.273	1.863	85.34	6.74
4	Cont-24	0.223	0.224	1.774	87.07	7.96
5	Silent-24	0.291	0.284	1.614	82.38	5.54
6	Mom-24	0.284	0.281	1.822	84.60	6.42
7	Coast-48	0.422	0.419	1.797	76.70	4.26
8	Fore-48	0.458	0.437	1.885	76.82	4.12
9	News-48	1.057	1.047	7.906	86.75	7.48
10	Coast-112	1.568	1.527	8.139	81.24	5.19
11	Fore-112	1.574	1.520	8.418	81.95	5.35
12	News-112	1.011	1.007	7.734	86.98	7.65
13	Mob-1024	1.435	1.598	5.175	69.12	3.61
14	Stef-1024	1.570	1.538	5.795	73.46	3.69

shown graphically in Figure 3.1.

The numbers presented thus far count basic arithmetic operations in a way that is both machine and implementation independent. It is also important to consider how the algorithms run on a real machine. To measure this, we use an instruction level profiler to count machine instructions used during actual decoding runs. Such a measure is in some sense more realistic than the operation counts used earlier. However, machine instruction counts depend not only on the algorithm, but on the implementation as well. In particular, C-level optimizations such as loop unwrapping and careful pointer arithmetic may reduce the number of machine instructions used. While it is interesting to compare profiling data, the implementation dependence makes it difficult to draw definitive conclusions.

With that caveat, we measure postfiltering complexity using machine level profiling in Table 3.4. The numbers were generated using the profiling tool IPROF 0.3 [37].

Figure 3.1: Comparison of total decoder complexity.



Each entry in the table represents the number of machine instructions used to decode the standard MPEG-4 test sequences on a 170 MHz Sparc Ultra-1. The MoMuSys MPEG-4 VM implementation is shown with and without the postprocessing filters. In this particular realization of the VM decoder, postfiltering accounts for between 40 and 50 percent of the real machine instructions, as shown in the last column. This result is smaller than the earlier prediction of 60 to 90 percent. One possible explanation is that the profiling data counts additional instructions for implementation-dependent operations such as loop overhead and pointer arithmetic. Another is that the postfiltering routines may be at a higher degree of code optimization than the motion and IDCT portions of the decoder. Still, this result shows that postfiltering can be expensive in a real DCT-based decoder implementation.

3.5 Coding Efficiency

We now consider the effect of DCT post-processing on the coding efficiency comparison. We thus provide both PSNR and visual results. Table 3.5 shows luma PSNR

Table 3.4: IPROF analysis (millions of instructions)

Seq	Label	VM	VM+PP	%PP
1	Cont-10	1795	3428	47.64
2	Hall-10	1687	3383	50.13
3	Mom-10	1852	3508	47.21
4	Cont-24	2493	4639	46.26
5	Silent-24	2500	4729	47.13
6	Mom-24	2576	4765	45.94
7	Coast-48	3005	4949	39.28
8	Fore-48	2979	4971	40.07
9	News-48	6859	13568	49.45
10	Coast-112	16306	28251	42.28
11	Fore-112	16060	28130	42.91
12	News-112	13700	27080	49.41
13	Mob-1024	28950	49547	41.57
14	Stef-1024	30190	53932	44.02

results by sequence for the matching-pursuit based system and also for the MPEG-4 VM with and without post-filtering. For the VM runs, rate control is based on constant quantizer step size using the values given earlier in Table 2.4. The matching pursuit system was then run to match the bit rate of the VM system for every frame, as described earlier in Section 2.3.3.

Two basic trends can be seen in Table 3.5. First, the post-filtering has only a small effect on the PSNR of the VM, and it decreases the PSNR more often than it increases it. Second, the matching pursuit system produces significantly higher PSNR numbers than the DCT-based system, regardless of post-processing. As seen earlier in Chapter 2, the improvement is more than 1 dB for half the tested sequences.

We also show an alternate result based on a different matching pursuit rate control mechanism. The same MPEG-4 VM runs are used, but now the matching pursuit encoder is run to match the Y-PSNR of the postfiltered VM run at every frame. The required

Table 3.5: Comparison of Luminance PSNR (dB). Matching pursuit matches the bit rate of the postfiltered VM run at every frame.

Seq	Label	MP	VM	VM+PP	Diff: MP-(VM+PP)
1	Cont-10	31.45	29.82	29.88	1.57
2	Hall-10	31.79	30.14	30.30	1.50
3	Mom-10	33.25	32.68	32.64	0.61
4	Cont-24	34.37	33.23	33.38	0.99
5	Silent-24	32.28	30.94	31.04	1.24
6	Mom-24	35.88	35.37	35.27	0.61
7	Coast-48	30.23	29.55	29.42	0.81
8	Fore-48	31.51	31.23	31.14	0.37
9	News-48	32.50	31.22	31.14	1.36
10	Coast-112	27.68	26.40	26.07	1.60
11	Fore-112	30.30	28.79	28.66	1.64
12	News-112	35.55	34.18	34.23	1.32
13	Mob-1024	27.01	26.37	26.45	0.56
14	Stef-1024	29.86	29.61	29.49	0.37

matching pursuit rate control mechanism was introduced in Chapter 2. The results are shown in Table 3.6 in terms of the average bit rate used for both matching pursuit and MPEG-4 VM encoders. We see that the matching pursuit system typically uses between ten and thirty percent fewer bits to obtain the same objective quality as the postfiltered VM system. Note that application of the postfilters does not affect the encoded bit rate of the VM, since the postfilters are applied only at decoding time.

We have shown that the MPEG-4 post-processing filters have little effect on the PSNR. They do, however, show a much greater visual effect. Figure 3.2a shows a sample frame of the Container Ship sequence coded at 10 kbit/s by the MPEG-4 VM without post-processing. Note the block artifacts which appear in the smoothly varying water. Additional distortion and ringing are seen in the small boat at the upper left of the image. Figure 3.2b shows the effect of the postfiltering. The block artifacts have mostly disappeared from the

Table 3.6: Comparison of average bit rate. Matching pursuit matches PSNR of the MPEG-4 VM at every frame.

Seq	Label	MP-kbit/s	VM-kbit/s	% Diff
1	Cont-10	8.26	10.35	20.2
2	Hall-10	7.87	10.13	22.3
3	Mom-10	9.38	10.57	11.2
4	Cont-24	19.83	26.13	24.1
5	Silent-24	20.25	25.34	20.1
6	Mom-24	20.30	23.54	13.8
7	Coast-48	42.33	51.57	17.9
8	Fore-48	47.43	49.59	4.4
9	News-48	38.62	47.92	19.4
10	Coast-112	88.93	131.05	32.1
11	Fore-112	110.81	137.67	19.5
12	News-112	87.42	114.67	23.8
13	Mob-1024	905.01	1019.88	11.3
14	Stef-1024	1022.55	1064.09	3.9

water, and the distortion around the small boat has been smoothed out. Now compare these images to Figure 3.2c, which is the equivalent frame coded using matching pursuits. Both the water and the small boat are clean and free from systematic artifacts, despite the fact that no post-processing is used. Also note that the matching-pursuit image is sharper and preserves more detail than the post-processed DCT. This is seen mostly in the small boat and in the outline and features of the container ship.

Similar results are shown for frame 100 of the Coast Guard sequence encoded at 112 kbit/s. Figure 3.3a shows the MPEG-4 VM result without postprocessing. Note the blocking artifacts which are most easily seen in the water. These are reduced significantly in the postprocessed version, as shown in Figure 3.3b. However, the smoothing effect of the filters causes detail to be lost. Compare the image in Figure 3.3b to the matching pursuit result, as shown in Figure 3.3c. The matching pursuit result shows much more detail in the water. The rocks behind the boat are also much sharper in the matching pursuit

Figure 3.2: Frame 50 of Container Ship coded at 10 kbit/s. (a) DCT-based MPEG-4 VM. (b) MPEG-4 VM with postfiltering. (c) Matching Pursuit.

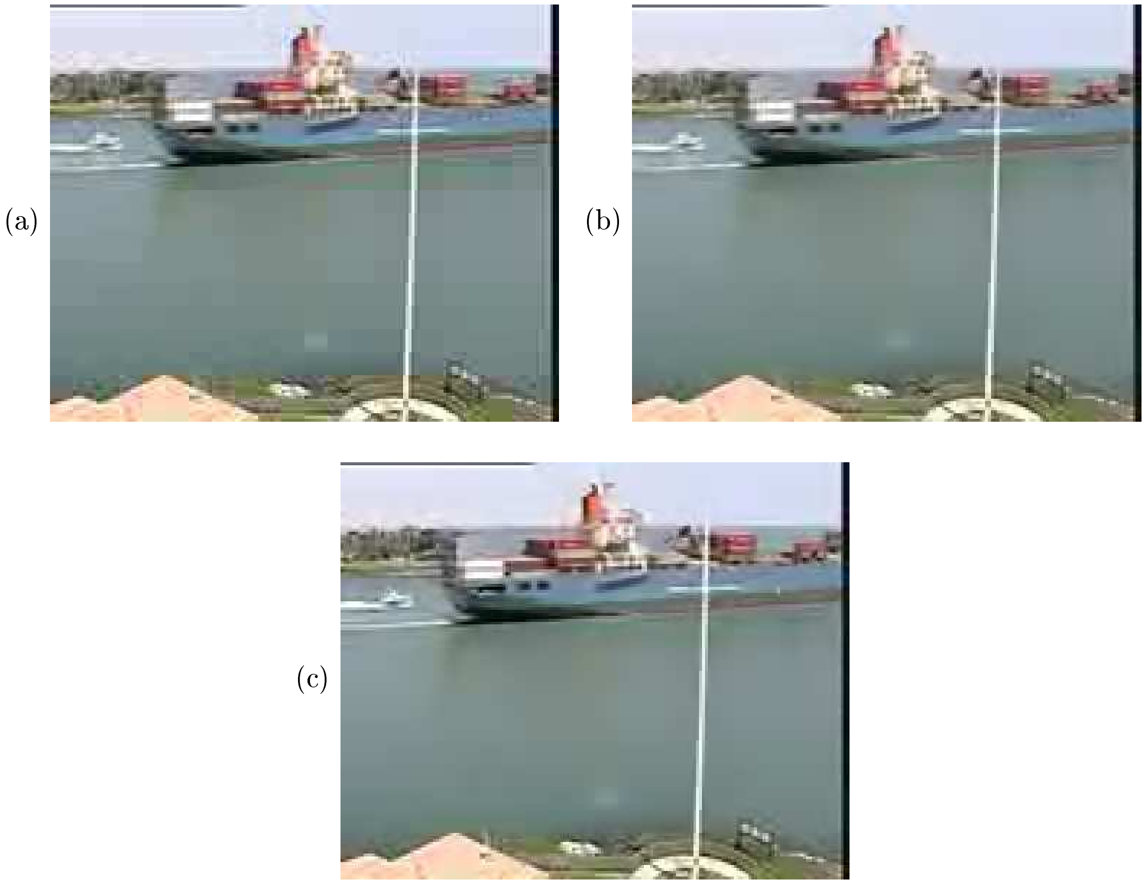


Figure 3.3: Frame 100 of Coast Guard coded at 112 kbit/s. (a) DCT-based MPEG-4 VM. (b) MPEG-4 VM with postfiltering. (c) Matching Pursuit.

(a) (b) (c)

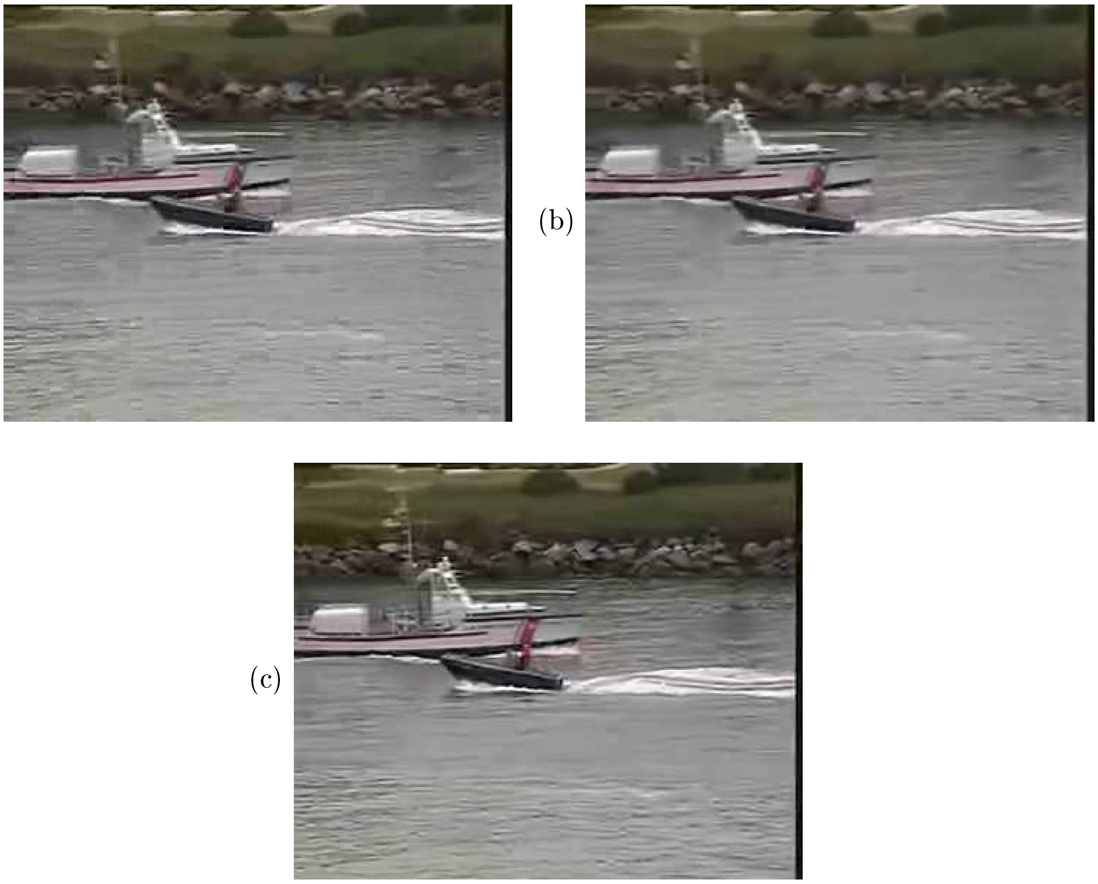


image. Finally, the lines and features on the large boat are sharper and more detailed when matching pursuit is used.

We conclude our decoder comparison by noting that the matching pursuit and DCT-based decoders have about the same complexity in the absence of post-processing. However, the DCT based system requires postprocessing to eliminate systematic blocking and ringing artifacts at low bit rates. This postprocessing is a significant computational burden, and can make a DCT based decoder much more complex than a matching pursuit decoder. Matching pursuit does not require postfiltering. As seen in our visual comparison, the matching pursuit result is a visual improvement over even the postfiltered DCT. Matching pursuit provides sharper image detail, and does so without introducing systematic artifacts.

3.6 Integer Matching Pursuit

The results presented above are based on a floating point matching pursuit implementation. This means that the encoder stores the basis functions as floating point values, and the inner product computation and atom subtraction procedures at the encoder are done using floating point arithmetic. The "atom pasting" used to reconstruct the image at the decoder also requires floating point math. In this section, we will investigate the possibility of replacing these floating point operations with scaled integer arithmetic. The experiment is designed to show that there is little difference in terms of compression efficiency, and that integer matching pursuit has some interesting implications for decoder design.

The integer version of matching pursuit is desirable for several reasons. First, different computing platforms may have slightly different floating point implementations. If a sequence is encoded on one platform and decoded on another, decoder mismatch may result. A second factor is the popularity of integer SIMD instructions which may significantly speed integer operations on modern microprocessors. Intel's MMX is perhaps the best known of these. Finally, the integer matching pursuit decoder no longer requires a floating point frame buffer for atom pasting, and so memory requirements are reduced.

The floating point matching pursuit encoder uses 20 1-D elementary bases of varying lengths. The 400 separable combinations of these elementary bases form the 2-D basis library for coding residual images [45]. These 1-D bases have unit-norm in the floating point version. To represent these as integer filters, we introduce a scaling factor of $2^6 = 64$. Specifically,

$$g_{INT}(x) = RoundToInt(f_{FLOAT}(x) \times 2^6)$$

where f_{FLOAT} is a 1-D basis function in the floating point matching pursuit implementation, and g_{INT} is the corresponding 1-D function in the integer implementation. The effective 2-D dictionary functions for integer matching pursuit are then formed by taking the separable products of all combinations of these 1-D scaled integer functions. Each 2-D basis is thus effectively scaled by 2^{12}, as are any computed modulus values.

At the encoder, the inner product computation is performed using the fast separable technique introduced in the previous chapter, but with the scaled integer versions of the 1-D basis components. The scaling factor does not otherwise affect the encoder, since it essentially disappears during quantization. This is because the scaled-integer modulus values are mapped to a set of integer bin indices, thus removing the scaling factor. Both integer and floating point matching pursuit encoders transmit the quantized coefficients at the same scale using the same set of integer-valued bin indices.

At the decoder, each modulus-weighted dictionary function forms an atom, and the summation of coded atoms forms a representation for the residual signal. Decoding thus consists of adding all coded atoms to the motion prediction image. To reduce both complexity and memory requirements at the decoder, the atoms are added directly to the 8-bit motion prediction image. Atom summation is thus done at the original image scale.

To do this using scaled integer arithmetic, we must compute each atom-pixel value $r(x, y)$ in a manner which maintains sufficient precision and simultaneously removes the scaling factors intrinsic to the basis vectors. That is,

$$r(x, y) = (g_h(x) \times ((g_v(y) \times p)/2^6))/2^6$$

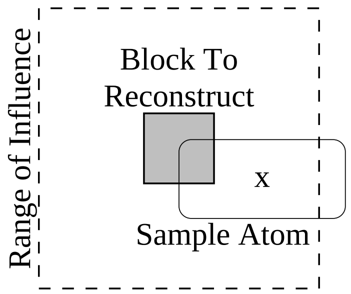
where p is the quantized coefficient value and $g_h(x)$ and $g_v(y)$ are the scaled integer values of the horizontal and vertical 1-D basis components. Note that the downscale divisions by 2^6 are done using nearest-integer rounding. Each such division is efficiently implemented as an addition followed by a 6-bit downshift, as follows:

$$result = (value + 32) >> 6$$

The atoms are added to the prediction image in a summation buffer. We seek to minimize the memory required for this buffer, while maintaining an order independence property and insuring proper clipping. We now clarify these requirements and show that a buffer with a depth of 9-bits is sufficient for a practical decoder.

Order independence means that the final reconstruction result is independent of the order in which the atoms are added to the summation buffer. This allows some useful freedom in the design of the decoder. For example, to minimize latency, the decoder should add atoms to the summation buffer in the order in which the atoms are received and decoded. Alternately, one could rearrange the order of atom addition to reduce the spatial extent of the summation buffer. Figure 3.4 shows a blockwise design. The summation buffer is reduced to a block of arbitrary size, and all overlapping atoms are added to the block's motion prediction signal. A reduced summation buffer may be desirable because the buffer must have a higher precision than the 8-bits to which the final result will be clipped.

Figure 3.4: Blockwise atom summation allows reduction in the size of the atom pasting buffer.



Higher precision and clipping are needed because of overflow and underflow. The original source image is of 8-bit resolution, as is the prediction image to which the atoms are added. The final summation result is the decoded representation of the original 8-bit source image. Since the coding is lossy, the final result as well as the intermediate summation values may exceed the [0,255] intensity range of the original motion residual source. This is most common when source pixels are near the limits of the range. Clipping to [0,255] is thus necessary. However, we wish to avoid designs which clip the intermediate summation values, since such clipping destroys the order independence property. For example, suppose a particular pixel has a motion prediction value of 250, and two overlapping atom pixels with values of +10 and -12. If intermediate values are clipped, then the end result clearly depends on the order of summation, as

$$Clip(250 - 12) + 10 = 248$$

but

$$Clip(250 + 10) - 12 = 243$$

If only the final summation result is clipped, order independence follows simply from commutativity. This requires that the summation buffer be capable of containing the final result before clipping, as well as all intermediate summation results. However by defining the summation with modulo arithmetic we reduce the requirement such that only the final result must fit into the summation buffer. The proof of this is straightforward and follows from the fact that if a, b and c are integers with $(a+b+c) \in [0, M)$, then

$$(((a+b)_{modM}) + c)_{modM} = (a+b+c)_{modM} = a+b+c$$

The summation buffer must then have sufficient bit-depth to contain only the final result including any overflow or underflow. Suppose the encoder is restricted such that the final summation result does not overflow the 8-bit range by more than 128 in either direction. In this case, a 9-bit summation buffer with the clipping strategy shown in Figure 3.5 is sufficient. Using signed integer modulo-512 arithmetic, final values in the range [-128,-1] indicate underflow and are clipped to 0, while values in the range [-256,-129] indicate wrapped overflow and are clipped to 255. Table 3.7 shows that our proposed encoder restriction is already met in practice. The table shows the maximum amounts by which the final summation result overflows or underflows the 8-bit range during integer matching pursuit decoding of twelve standard sequences. The numbers show that the final summation result is always within 60 of the 8-bit range, which is well below the limit of 128 proposed above. We thus conclude that a 9-bit summation buffer allows proper clipping and preserves the order independence property. For a software implementation on a real machine, however, a 16-bit short integer implementation is more convenient.

Table 3.8 shows the coding efficiency of the integer matching pursuit codec compared to that of a comparable floating point matching pursuit codec. The only difference between these two codecs is the introduction of scaled integer arithmetic as described above.

Figure 3.5: Clipping strategy for a 9-bit summation buffer.

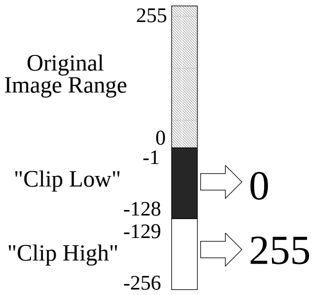


Table 3.7: Maximum overflow and underflow in final summation result.

Seq	Label	Over	Under	Seq	Label	Over	Under
1	Cont-10	41	-28	7	Coast-48	36	-23
2	Hall-10	38	-28	8	Fore-48	32	-15
3	Mom-10	0	0	9	News-48	51	-26
4	Cont-24	34	-20	10	Coast-112	60	-34
5	Silent-24	0	-6	11	Fore-112	43	-53
6	Mom-24	0	0	12	News-112	39	-27

We see that the reduced precision of the integer arithmetic causes little or no drop in coding efficiency performance. The integer codec is within 0.06dB of the floating point codec for all twelve tested sequences.

Note that the baseline floating point encoder used for this experiment differs slightly from the one described in Chapter 2 in the way that the atom position information

Table 3.8: Y-PSNR (dB) - Float vs. Integer M.P.

Seq	Label	Float	Int	Diff	Seq	Label	Float	Int	Diff
1	Cont-10	31.32	31.26	-0.06	7	Coast-48	30.15	30.15	0.00
2	Hall-10	31.61	31.66	0.05	8	Fore-48	31.37	31.35	-0.02
3	Mom-10	33.12	33.13	0.01	9	News-48	32.20	32.19	-0.01
4	Cont-24	34.32	34.31	-0.01	10	Coast-112	27.65	27.66	0.01
5	Silent-24	32.08	32.07	-0.01	11	Fore-112	30.26	30.25	-0.01
6	Mom-24	35.77	35.72	-0.05	12	News-112	35.27	35.26	-0.01

is transmitted. The Chapter 2 encoder uses a "combined mode" encoding where atoms are grouped by macroblock and transmitted in position order along a modified spiral pattern introduced in [3]. For the current experiment, an older "separate mode" encoding is used. Atom are still sent using differential position codes, but all atoms in the current frame are arranged in raster scan order and encoded as a single group. This method was fully described in [45]. Separate mode encoding is slightly less efficient than combined mode [3], and so the floating point results shown in Table 3.8 are an average of 0.14 dB lower than the earlier results.

This difference does not impact our current experiment, which requires a comparison between matching pursuit codecs which differ only in whether floating point or integer arithmetic is used. The two matching pursuit codecs which are compared in Table 3.8 meet this criterion, and both use separate mode atom encoding. We may thus safely conclude that the introduction of scaled integer arithmetic does not adversely affect coding efficiency.

3.7 Conclusions

In this chapter, we demonstrated that the matching pursuit video encoder outperforms the emerging MPEG-4 standard in terms of both PSNR and visual quality. The matching pursuit system also has the advantage that it requires no postfiltering. This allows the decoder to remain simple, while avoiding the reduction in detail which accompanies the use of smoothing filters. The postfilters used for the MPEG-4 VM were shown to remove DCT artifacts and thus provide a visual improvement, though the effect on PSNR was minimal. Matching pursuit was shown to provide significant visual improvement, delivering more detailed images than the DCT system regardless of whether postfiltering was used. Furthermore, our complexity analysis reveals that DCT postfiltering may have a prohibitive cost. The examined filters were shown to increase decoder complexity by a factor of three to eight in our algorithmic operation counts, and by a factor of two in machine instructions for the tested implementation. Note that our complexity comparisons should be considered preliminary in that further algorithm improvements and code optimization might produce different results. This is especially true of the IPROF comparison, which is both machine and code implementation dependent.

Finally, a modified matching pursuit codec using scaled integer arithmetic was shown to perform within 0.06dB of a comparable floating point codec. Such a version allows reduced storage and potentially faster implementations on modern machines.