Speeding Up Motion Estimation in Modern Video Encoders Using Approximate Metrics & SIMD Processors

Steven Pigeon & Stéphane Coulombe
steven.pigeon@etsmtl.ca
stephane.coulombe@etsmtl.ca

École de Technologie Supérieure
Dept. of Software and IT Engineering
1100 Notre-Dame Ouest, Montréal
Québec, Canada

September 28, 2009

Motion Compensated Video Coding

Almost all modern video codecs rely on motion compensated video coding as their primary mean of spatial and temporal redundancy reduction

But Motion Compensation requires **Motion Estimation** which is still computationally intensive

In the past, the focus was on the development of efficient predictive search methods

New focus: Implementation-specific Speed-ups!

Implementation Aspects

Implementation-specific Speed-Ups

- Astute exploitation of the machine, CPU, Compiler, and algorithms.
- Exploiting the machine:
  - Using the ISA (instruction set architecture) to its fullest
  - ...especially SIMD “multimedia” extensions

Motion Estimation Algorithms

Gradient-descent heuristics

M. E. algorithms are—after the predictive step—very often only gradient-descent type algorithms, searching for a local minimum

To do so, they suppose that the error surface generated by the metric is approximately concave around the position of the best match

Therefore, it is reasonable to suppose that most M. E. are quite resilient to approximate metrics for image matching

Motion Estimation Algorithms

Resilience to Approximate Metrics

Solutions were proposed before:
[Chan96, Kwan97, Liu93, Tom06]

- Early Termination
- Progressive/Hierarchical Sampling

but...

- Do not take into account the underlying machine
- branch-intensive, more code, more math, etc.

Motion Estimation Algorithms

Resilience to Approximate Metrics

The resilience suggests make full use of approximate metrics

...which in turn allows the exploitation of SIMD multimedia extensions for better run-time

SIMD *must* be considered: sequential integer operations are (comparatively) very slow for the high computational demands of video codecs

Approximate Metrics

Classical SAD

There are very few metrics actually used in codecs. One is the MSE, the other is the SAD:

$$\mathrm{SAD}(I,J) = \sum_{x=1}^{16}\sum_{y=1}^{16} |I_{x,y} - J_{x,y}|$$

where I and J are two 16×16 (1-component) pixels image patches.

$\rightarrow$ It considers all the points.

Approximate Metrics

Modified SAD

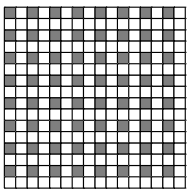
We propose to generalize the SAD:

$$\mathrm{SAD}_M(I,J) = \sum_{x=1}^{16}\sum_{y=1}^{16} M_{x,y}\ |I_{x,y} - J_{x,y}| \qquad (1)$$

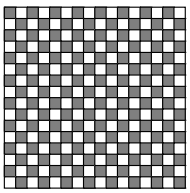
with M, a 16×16 binary matrix, conditionally enabling or disabling pixels in the metrics.

Approximate Metrics

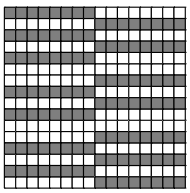
Proposed Approximate Metrics



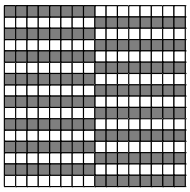
(a) Sparse



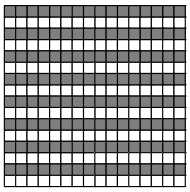
(b) Quincunx



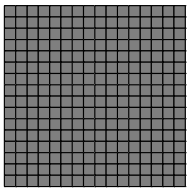
(c) S-Deint



(d) Deint



(e) Inter.



(f) Full

Figure: Proposed Metrics

Approximate Metrics

The Matrix M

The Matrix M...

- Allows to adjust Sampling Density
- Allows to build-in machine-specific constraints—such as SIMD friendly patterns

→ Must balance the two!

Testing Motion Estimation Resilience

Building Proper Tests

Selecting 'benchmark' sequences in CIF/QCIF:

- Akiyo
- Bus
- Foreman
- ...

Selecting relevant Motion Estimation Algorithms:

- Full Search
- UMHexS
- EPZS
- PMVFAST
- ...

Testing Motion Estimation Resilience

The Foreman Sequence, Full Search

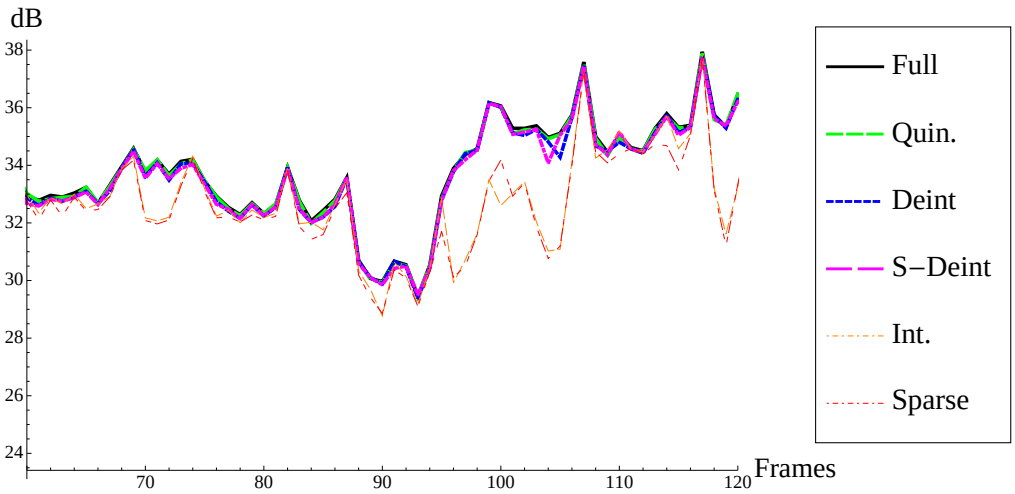


Figure: Foreman CIF Sequence, using Full Search

Testing Motion Estimation Resilience

The Foreman Sequence, EPZS

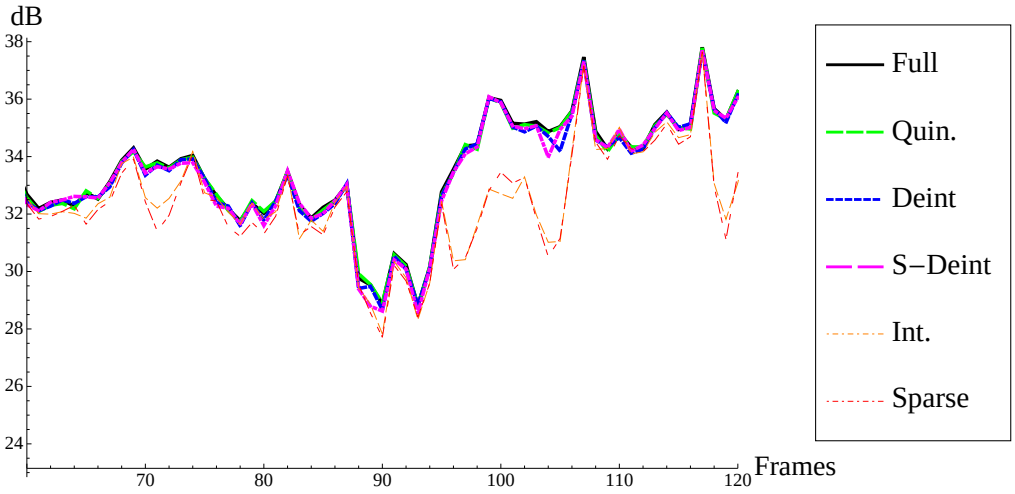


Figure: Foreman CIF Sequence, using EPZS

Testing Motion Estimation Resilience

The Foreman Sequence, PMVFAST

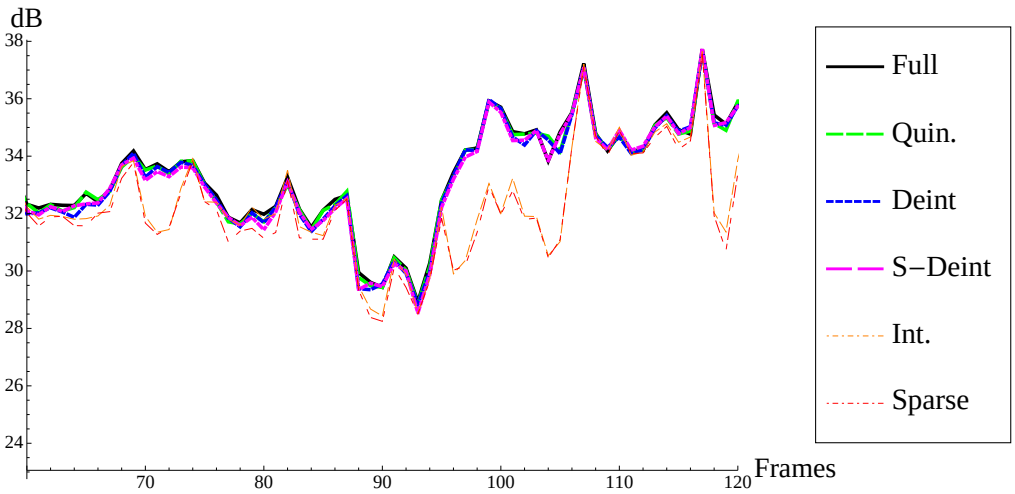


Figure: Foreman CIF Sequence, using PMVFAST

Testing Motion Estimation Resilience

Loss of Quality, in dB, for QCIF

CIF	SAD	Quin.	Deint	S-Deint	Int.	Sparse
Akiyo	43.3	-0.01	-0.03	-0.03	-0.05	-0.07
Bus	24.1	-0.03	-0.07	-0.10	-0.11	-0.28
Foreman	31.5	-0.05	-0.06	-0.09	-0.13	-0.18
News	35.9	-0.03	-0.06	-0.07	-0.06	-0.19
Mobile	25.4	-0.04	-0.03	-0.04	-0.04	-0.16
Stefan	25.1	-0.04	-0.07	-0.11	-0.09	-0.22
Tempete	27.2	-0.02	-0.03	-0.03	-0.03	-0.07

Table: PSNR for QCIF sequences using full search

CIF	SAD	Quin.	Deint	S-Deint	Int.	Sparse
Akiyo	43.3	-0.01	-0.03	-0.02	-0.05	-0.07
Bus	23.5	-0.01	-0.02	-0.06	-0.09	-0.20
Foreman	31.3	-0.08	-0.08	-0.11	-0.17	-0.28
News	35.9	-0.04	-0.05	-0.07	-0.05	-0.22
Mobile	25.4	-0.03	-0.02	-0.04	-0.04	-0.15
Stefan	24.8	-0.02	-0.02	-0.03	-0.04	-0.11
Tempete	27.0	-0.03	-0.04	-0.05	-0.03	-0.07

Table: PSNR for selected QCIF sequences using EPZS

Testing Motion Estimation Resilience

Loss of Quality, in dB, for CIF

CIF	SAD	Quin.	Deint	S-Deint	Int.	Sparse
Akiyo	42.8	-0.02	-0.05	-0.07	-0.05	-0.11
Bus	25.1	-0.01	-0.06	-0.10	-0.13	-0.34
Foreman	32.2	-0.03	-0.10	-0.11	-0.76	-0.86
News	36.5	-0.03	-0.06	-0.09	-0.10	-0.23
Mobile	25.2	-0.04	-0.07	-0.08	-0.09	-0.26
Stefan	26.0	-0.01	-0.08	-0.10	-0.12	-0.25
Tempete	27.0	-0.01	-0.04	-0.06	-0.05	-0.12

Table: PSNR for CIF sequences using full search

CIF	SAD	Quin.	Deint	S-Deint	Int.	Sparse
Akiyo	42.7	-0.02	-0.05	-0.07	-0.06	-0.11
Bus	24.3	-0.00	-0.05	-0.05	-0.17	-0.34
Foreman	31.9	-0.04	-0.11	-0.11	-0.73	-0.83
News	36.2	-0.03	-0.08	-0.12	-0.08	-0.24
Mobile	25.1	-0.03	-0.05	-0.06	-0.07	-0.23
Stefan	25.7	-0.01	-0.09	-0.09	-0.11	-0.22
Tempete	26.5	-0.02	-0.05	-0.07	-0.06	-0.13

Table: PSNR for selected CIF sequences using EPZS

Testing Motion Estimation Resilience

Resilience is Verified!

Resilience is Verified...

- For most Approximated Metrics
- For SIMD-friendly Metrics
  - But have to be dense enough
  - But have to be spread enough
- $\lessapprox$ 0.1 dB worst case for Deint and S-Deint!

Machine-Specific Speed Ups

Choosing The Right Tools

Choosing a platform (and its tools) to study :

- The ubiquitous x86/86_64 Family [1]
- SIMD instruction set: SSE and SSE2 levels widely available
- Generic processors of all kinds: netbooks to high-end servers
- Sophisticated development tools:
  - GNU/Linux Ubuntu 8.04 LTS as operating system
  - Intel C Compiler (ICC) 11.x
  - Intel Performance Primitives (IPP) 6.0.x (used as a benchmark)

[1] In the server/workstation/home worlds; while all the major game consoles have PowerPC-derived Chips

Machine-Specific Speed Up

Experimental Code

The Experimental Code

- "Vanilla" C versions of metrics
- "Vanilla" C versions with auto-vectorization
- SSE/SSE2-level assembly language versions of metrics using:
  - smart call conventions
  - full constant-propagation
  - simple addressing modes

Machine-Specific Speed Up

A case study: Core T2500 @ 2.00GHz

	QCIF			
Implementation	pixels	Calls/μs	Pixels/μs	Speed-up
SAD, C	100%	1.40	358.4	1:1
SAD, IPP	100%	7.14	1827.8	5.1:1
SAD, C, Vect.	100%	7.53	1927.7	**5.4:1**
MSE, C	100%	1.45	371.2	1:1
MSE, C, Vect.	100%	4.45	1139.2	3.2:1
Sparse, C, Vect.	25%	5.53	353.9	4:1
S-Deint, C, Vect.	44%	3.67	411.0	2.6:1
Quin., C, Vect.	50%	2.67	341.8	1.9:1
Int., C, Vect.	50%	3.46	442.9	2.5:1
Deint, C, Vect.	50%	3.20	409.6	2.3:1
SAD, SSE2	100%	8.27	2117.1	5.9:1
Sparse, SSE2	25%	13.13	850.3	9.4:1
S-Deint, SSE2	44%	15.98	1789.8	11.4:1
Quin., SSE2	50%	7.39	945.9	5.3:1
Int., SSE2	50%	14.50	1856.0	10.4:1
Deint, SSE2	50%	13.88	1776.6	9.9:1

Table: QCIF timing results for the Intel Core T2500 (accuracy within ±1%).

Machine-Specific Speed Up

A case study: Core T2500 @ 2.00GHz

	CIF			
Implementation	pixels	Calls/μs	Pixels/μs	Speed-up
SAD, C	100%	1.30	332.8	1:1
SAD, IPP	100%	5.36	1372.2	4.1:1
SAD, C, Vect.	100%	5.71	1461.8	4.4:1
MSE, C	100%	1.41	361.0	1.1:1
MSE, C, Vect.	100%	3.93	1006.1	3.0:1
Sparse, C, Vect.	25%	4.87	311.7	3.7:1
S-Deint, C, Vect.	44%	3.40	380.8	2.6:1
Quin., C, Vect.	50%	2.42	309.8	1.9:1
Int., C, Vect.	50%	3.33	426.2	2.6:1
Deint, C, Vect.	50%	2.97	380.2	2.3:1
SAD, SSE2	100%	5.94	1520.6	4.6:1
Sparse, SSE2	25%	9.95	636.8	7.7:1
S-Deint, SSE2	44%	9.60	1075.2	7.4:1
Quin., SSE2	50%	5.63	720.6	4.3:1
Int., SSE2	50%	10.48	1341.4	8.1:1
Deint, SSE2	50%	8.14	1041.9	6.3:1

Table: CIF timing results for the Intel Core T2500 (accuracy within ±1%).

Effects on Quality

- Effects of Approximated Metrics are *negligible*
- $\lessapprox 0.1$ dB before quantization, for most sequences, even with high motion

  On average, the fast, SIMD-friendly, approximate metrics perform about as well as the exact metric
- Some metrics are *bad*: $\lessapprox 1$ dB before quantization

  Even though the Interlaced metric is very SIMD-friendly and the Sparse metric very fast, and the average loss is much smaller than 1 dB, these metrics may incur an unacceptable loss upto $\lessapprox 1$ dB, especially on high-motion videos such as Bus and Foreman. They should therefore be avoided.

Speed Up

Vectorizing Compilers

- Optimizing/Vectorizing Compilers are still finicky

  Compiler do not always recognize vectorizable code even if it was written with care.
- Vectorized code not always very impressive

  Even when the compiler detects the vectorization potential, it does not necessarily produce very efficient code. For example, the auto-vectorized Quincunx approximate metric has a speed up of 1.9:1 relative to the non-vectorized C++ code, but the hand-crafted SSE2 version offers 5.3:1 !
- Vectorizing compilers still have a *long* way to go!

  So even if optimizing compilers are better at generating code than other compilers, we cannot rely on them very heavily for speed optimization.

Speed Up

- CIF and QCIF exhibit different performance characteristics
- QCIF: up to 11.4:1 using S-Deint (with a loss of $\lessapprox$ 0.1 dB).
  IPP delivers 5.1:1
- CIF: up to 8.1:1 with S-Deint
  IPP delivers 4.1:1

Conclusion

Results

- $\lessapprox$ 0.1 dB loss with good, SIMD-friendly, fast, approximate metrics
- Speed Ups up to
  - 11.4:1 from non-vectorized C code for QCIF
  - 8.1 for CIF

  $\gtrapprox 2:1$ against IPP (and auto-vectorized code)
- A viable alternative to costly exact computation of the SAD

Conclusion

Future Directions

- Characterize resulting Quality of Approximate Metrics with Quantization
- Characterize speed ups in codecs like MPEG-4 AVC / H.264

References

- Yui-Lam Chan, Wan-Chi Siu
  New Adaptive Pixel Decimation for Block Motion Vector Estimation
  IEEE Trans. Circuits and Systems for Video Technology, V6(1) (Jan 1996) p. 113–118
- Chok-Kwan Cheung, Lai Man Po
  A Hierarchical Block Motion Estimation Algorithm Using Partial Distortion Measures
  Int. Conference On Image Processing (ICIP), V3 (1997) p. 606–609
- Bede Liu, André Zaccarin
  New Fast Algorithms for the Estimation of Block Motion Vectors
  IEEE Trans. Circuits and Systems for Video Technology, V3(2) (Apr 1993) p. 148–157
- Federico Tombari, Stefano Mattoccia
  Template Matching Based on the L_p Norm using Sufficient Conditions with Incremental Approximation
  Procs. IEEE Int. Conf. on Advanced Video and Signal-Based Surveillance (Nov 2006) p. 20–26

This work was sponsored by

The National Engineering and Science Research Council of
Canada
and by

Vantrix Corporation

http://www.nserc-crsng.gc.ca/ http://www.vantrix.com/