Published in final edited form as:

IEEE Trans Circuits Syst II Express Briefs. 2018 September ; 65(9): 1229–1233. doi:10.1109/TCSII.

2017.2775583

A Streaming Motion Magnification Core for Smart Image Sensors

Cong Shi and Gang Luo

Schepens Eye Research Institute, Massachusetts Eye and Ear, Department of Ophthalmology, Harvard Medical School, Boston, MA

Abstract

This paper proposes a modified Eulerian Video Magnification (EVM) algorithm and a hardware implementation of a motion magnification core for smart image sensors. Compared to the original EVM algorithm, we perform the pixel-wise temporal bandpass filtering only once rather than multiple times on all scale layers, to reduce the memory and multiplier requirement for hardware implementation. A pixel stream processing architecture with pipelined blocks is proposed for the magnification core, enabling it to readily fit common image sensing components with streaming pixel output, while achieving higher performance with lower system cost. We implemented an FPGA-based prototype that is able to process up to 90M pixels per second and magnify subtle motion. The motion magnification results are comparable to the original algorithm running on PC.

Index Terms

motion magnification; Laplacian pyramid; spatiotemporal filtering; smart image sensor; pixel stream processing

I. Introduction

Visual motion magnification technique was first invented by MIT Computer Science and Artificial Intelligence Laboratory in 2005 [1]. It can help to reveal subtle motions in the video that are very difficult or impossible for naked eyes to perceive, by visually exaggerating motion in image sequences [1]–[4]. The method has been applied to various areas including security inspection [5], emotion recognition [6], biomedical care [7] and structural modal analysis [8]. However, all those applications were exclusively implemented using conventional cameras and standard personal computers. They are not suitable for applications where low system cost, compact size, and real-time processing performance are required.

The emergence of smart image sensor technology has given rise to novel compact and fast vision systems [9]. A smart image sensor integrates both image sensing and processing components in a small system, providing *smart* functions as it is capable of processing the raw image pixels on demand before they are streamed out. The integration can be carried out on a chip-level [9]–[11] or a board-level [12]. Such tight coupling between sensing and the processing components facilitates faster data transmitting and lower power consumption than the traditional vision systems. Moreover, the processing cores in the smart image sensor are usually customized and optimized for vision applications, and often achieves higher processing speed than general purpose CPUs.

In this paper, we propose a motion magnification processing core for smart image sensors for making subtle motions more visible. Our core is based on the Eulerian Video Magnification (EVM) algorithm [2]. As the algorithm involves mainly spatial and temporal image filtering requiring dense yet simple pixel-level operations, it is very suitable for low-cost hardware implementation. For the purpose of hardware optimization, we implemented a modified EVM algorithm by swapping the order of the spatial filtering and temporal filtering steps, as suggested by Wadhwa et al [4]. Thus the temporal filtering needs to be performed only once rather than multiple times on different scale layers and the magnification effect remains the same.

The proposed hardware architecture for the magnification core supports the processing of pixel stream on the fly. Whenever one pixel is fed from the sensing component, the processing blocks in the core are invoked, and they operate on the pixel data in a pipeline scheme, and then output one pixel down the stream. Compared to the pixel- or column-parallel processing components commonly used in previous smart image sensors [10], [11], the proposed streaming architecture has three advantages. First, it is ready to be integrated with standard image sensors, since common image sensors output pixels in a serial stream, which directly fits our magnification core. Second, the pixel processing is synchronous with pixel I/O operations, and consumes no additional time compared to a standard image sensor. The system frame rate of the smart image sensor is only limited by the sensing component. Third, the streaming architecture is data-driven and requires local rather than global control. So the circuits design can be simplified and the system cost is reduced. We realized the core prototype on the ZC706 FPGA platform. The experimental results demonstrated that the proposed core can run at a speed up to 90M pixel/s, while consuming very few computational resources and moderate amount of memory resources. For example, our prototype core consumes only 5500 logic slices, 9 multipliers and 239 block RAMs for 960×544 image resolution. Most modern low end FPGAs have more resources than the core needs.

This paper proceeds as follows. In Section II, we briefly introduce the EVM algorithm and the proposed modification. The details of hardware design of the proposed motion magnification core are illustrated in Section III, followed by the FPGA prototype implementation and experimental results in Section IV. Finally, Section V concludes this paper.

II. Algorithm

A. State-of-the-art EVM Algorithm

The principle of EVM algorithm is that the image displacement due to motion causes local temporal brightness variations, which can be described using the first-order Taylor approximation, given the local brightness gradient is small [2]. However, the first-order Taylor approximation usually does not hold at locations where brightness gradient is large, for example, due to fine-scale textures in the image. Therefore the magnification factor has to be attenuated on those fine scales.

The overall EVM algorithm flow is as follows. Each image frame in the video sequence is first decomposed into a pyramid of Laplacian images of different scales by iterative spatial Gaussian filtering, image down-sampling and image subtraction. Then the Laplacian images at all scales undergo an identical pixel-wise temporal bandpass filtering, to generate brightness variations due to the motion at certain temporal frequencies of interests. Then the variations are magnified by different factors. Given a user-defined expected magnification level, the factors at those fine scales (the scale layers near the pyramid top) should be attenuated to smaller values according to the bound inequality proposed in [2]. Finally, the magnified motion data at all scales are collapsed and merged into the original image to construct a new frame by iterative image up-sampling and image summation. For more details about EVM algorithm, please refer to [2].

Empirically, the highest and lowest scale layers are always assigned a zero magnification factor, because the highest layer corresponding to the finest scale usually has the magnification factor attenuated nearly to zero. On the other end, the lowest layer consists of almost only DC spatial frequency component containing little motion information.

B. Modified EVM Algorithm

Since all the spatial and temporal filtering operations in the state-of-the-art EVM algorithm are linear, the order of the two types of filtering can be swapped, as suggested in [4]. This modification is particularly beneficial for FPGA implementations because in such a way, all layers need to undergo an identical temporal bandpass filtering only once before the spatial Laplacian pyramid construction. Thus, temporal filtering does not need to be repeated at multiple layers. Since even the simplest temporal bandpass filter requires at least two multiplying operations and two taps (i.e., two image frame buffers), the proposed modification can save expensive multiplies and many memory resources. With this improvement, our modified EVM algorithm flow is shown in Fig. 1. Because the highest scale does not take part in motion magnification, we can carry out the temporal bandpass filtering at the first down-sampled layer (Layer 1 in Fig. 1) to further reduce memory requirement. The lowest layer involves no computation at all, but is still counted as one layer (called dummy layer) in our algorithm. To our best knowledge, this is the first time the filter swapping idea to be implemented since it was proposed by Wadhwa et al [4].

In our algorithm, the 2D spatial Gaussian filter at each scale is realized by cascading one vertical and one horizontal 1D Gaussian filters, $\mathbf{G} = (1/16) \times [1 \ 4 \ 6 \ 4 \ 1]^T$ and $\mathbf{G}^T$. The filter coefficients are optimally chosen to satisfy the requirement that the sums of even and odd indexed coefficients must be equal for later up-sampling [13]. Also, the chosen coefficients allow multiplication through simple bit-shifting without multipliers (e.g., $\times 6$ can be realized by summing the results of 2-bit left shifting and 1-bit left shifting). In the up-sampling stage as shown in Fig. 1, each layer inserts zero-valued pixels between columns and rows of the magnified Laplacian images to double the image size. Then the same Gaussian filters used for down-sampling are used here for pixel interpolation. The resulting pixels are further multiplied by 4 to compensate for the inserted zero-valued pixels.

The pixel-wise temporal bandpass filter in our algorithm are realized by cascading first-order lowpass and highpass filters:

$$\text{Lowpass: } m[t] = c_l m[t-1] + (1 - c_l) p[t] \quad (1a)$$

$$\text{Highpass: } z[t] = c_h z[t-1] + (1 + c_h) \frac{m[t] - m[t-1]}{2}, \quad (1b)$$

where $p[t]$, $m[t]$ and $z[t]$ represent the input pixel, the intermediate lowpass filtered pixel and the final result pixel in frame t , respectively. The filter coefficients c_l and c_h can be obtained using an equation described in [14]:

$$c = \exp(-2\pi f_c / f_s), \quad (2)$$

where c is the desired lowpass or highpass filter coefficient, and f_c and f_s are the cutoff frequency and the sensor frame rate, respectively.

III. HARDWARE DESIGN

A. Streaming Architecture

Fig. 2 shows the proposed architecture of the motion magnification core. The Gaussian filter blocks are responsible for producing Laplacian images and down-sampled images. The *mag-merge-upsample* blocks magnify the Laplacian images in brightness, merge the magnified into the up-sampled images from lower layers, and up-sample the merged images to higher layers.

The core architecture is based on pixel stream processing scheme. The input pixel stream of one frame invokes the Gaussian filter block on the first layer, which in turn produces another pixel stream to invoke the temporal bandpass filter on the second layer, and so forth as in a Domino chain, until the output pixel stream appears. The processing operations in each block are pipelined so that the input and output streams can flow at the same rate, making the maximum throughput of this core one pixel per clock cycle. Note that the processing operations can also be done in parallel with the pixel integration and readout operations in the sensing component. Therefore the frame rate of the whole smart imager system is determined by the smaller one between sensing component frame rate and processing core frame rate. Each block is data-driven by its neighboring blocks without global control (except a global clock for synchronizing the operation pace in all blocks). Such streaming architecture has three advantages: readiness to fit the image sensing component, high performance with processing time hidden behind pixel I/O operations, and low system cost with simplified circuit design, as mentioned above in Section I.

However, there is a time lag between these streams, as shown in the bottom of Fig. 2. The blocks involving spatial filtering can produce the result pixel only after all of its neighboring pixels arrive. But such stream lag will not slow down the system frame rate because the operations in each block are pipelined at the same pace and thus exhibit same data

throughput, as shown in Fig. 2. To cope with the lag, FIFOs are employed to temporally store the input pixels and the Laplacian pixels. The *mag-merge-upsample* block later fetches the FIFO data upon the arrival of lagging up-sampled pixels from the lower layer. The FIFO depth of one layer is determined by the corresponding lag, which is in turn determined by the number of layers below the current layer in concern, and the image sizes on those layers. In this work, we choose the number of layers so that the size of discarded Laplacian pyramid residual on the dummy layer is no less than 8×8 to guarantee the frame lag to be less than half of the sensor frame time. Therefore it is sufficient to have the FIFO depth in each layer equal to half of the input image resolution at that layer.

B. Block Design

Fig. 3 shows the circuit diagrams of the computational blocks in the architecture. The 4-stage pipelined temporal bandpass filter in Fig. 3(a) consists of a lowpass filter and a highpass filter designed according to Eq. (1). The filter coefficients are coded in an unsigned 7-bit fractional format. The input pixel $p(t)$ is an unsigned 8-bit integer and the lowpass result $m(t)$ is unsigned 15-bit including 7-bit fractional. The final result $z(t)$ is signed 16-bit including 7-bit fractional. Two buffers are used to store the filtered results of the previous frame. For the *baby* sequence that has the highest resolution in Table I, this requires $960 \times 544 \times 2 \times 16 \text{bit} = 2 \text{MB}$, which is an acceptable amount of memory in nanoscale devices.

The Gaussian filter block in Fig. 3(b) has a 5-stage pipeline through a vertical Gaussian filter, a horizontal Gaussian filter, a 2:1 down-sampler and a Laplacian generator. When a pixel at image location (x, y) arrives, the vertical Gaussian filter calculates the result at $(x, y - 2)$, from the original pixels stored in the pixel row FIFOs. The result along with previously vertical filtered results stored in the vertical results shifter are further processed by the horizontal Gaussian filter to derive the 2D Gaussian filtering result at $(x - 2, y - 2)$. This result is then conditionally outputted via the down-sampler, and in combination with the delayed corresponding input pixel produces the Laplacian pixel at $(x - 2, y - 2)$. The coordinate monitor counts the current image column and row coordinates of the Gaussian filtered pixel, and controls the down-sampling action. It also assists the 1D Gaussian filters to automatically handle border situations by padding zeros (not drawn in Fig. 3(b)). The Gaussian filter block is configurable in compile-time to support different pixel data formats. In the topmost layer, the input and down-sampled pixels are unsigned 8-bit integers. In other layers, the input and down-sampled pixels are signed 16-bit including 7-bit fractional. The Laplacian pixel is signed 17-bit including 7-bit fractional, the LSB of which is rounded before it is stored in the Laplacian FIFO with 16-bit width.

The *mag-merge-upsample* block is also 5-stage pipelined, with the first stage responsible for Laplacian pixel magnification and its merge with the up-sampled pixel from its next lower layer. The magnification factor is an unsigned 7-bit integer. All the fractional bits of the magnification multiplier output are rounded before merging. The next four pipeline stages are the vertical and horizontal Gaussian filters, the same as used in the Gaussian filter block. But the input and output pixel formats of the two filters are configured as signed 20-bit integer in this block. To support the quadruple rate of up-sampled pixel stream, the block borrows a signal as the synchronous signal from its next higher layer. Such signal is

synchronous with the Laplacian pixel stream from the Gaussian filter block. It controls the Gaussian filters in this block to either select the merged pixels or just insert zeros as their inputs, according to current pixel coordinate. It also synchronizes the access to Laplacian FIFO and the up-sample FIFO during pixel magnification and merging. The up-sample FIFO is used to handle the situation when the lower layer produces one up-sampled pixel while the upper layer is inserting zeros and cannot consume the pixel immediately. On the topmost layer, the *mag-merge-upsample* block is simplified as only a merge block that sums the up-sampled pixel from the second layer and the original pixel. The summation result is truncated to range $[0, 255]$ as the final unsigned 8-bit result.

All the data bit-precisions for the blocks mentioned above are carefully selected to achieve the best tradeoff between resource consumption and magnification effect. Those rounded bits do not result in a human perceivable degradation on the magnification effect, as will be shown in Fig. 5.

IV. FPGA prototype and experiments

We have implemented a prototype of the proposed motion magnification core on the ZC706 FPGA platform, and built an evaluation system to test the core, as shown in Fig. 4. To facilitate the testing, we employed an on-chip virtual sensor containing a FIFO to hold a row of pixel data. For each frame in the video sequence, the hard IP core of ARM processor on the FPGA chip transfers a row of pixel data from the PC to the virtual sensor FIFO. Then the FIFO outputs these pixels continuously to the magnification core. This procedure repeats until all rows of pixels in this frame are transferred and outputted, just like a real image sensor outputting pixels in a row-by-row manner. The magnified pixels were temporally stored in an output FIFO (with a depth to hold pixels of one frame), and then transferred via the ARM processor back to PC for displaying. The core prototype can run at a speed up to 90M clock frequency, equivalent to a 90M pixel/s processing speed. The maximum throughput of our core architecture is one pixel per clock cycle, as mentioned in Section III-A. Therefore, our core can be used in smart image sensors of 1920×1280 at 36 fps, or of 640×480 at 290 fps, or of 320×240 at 1170 fps, assuming the image sensing component can run at the same or higher frame rate.

We selected the video sequences with different image resolutions used in [2] as benchmarks. We configured our prototype core to support those different resolutions and corresponding number of scale layers accordingly. The FPGA resource consumption for each configuration is given in Table I (excluding the consumption on the virtual sensor and the output FIFO). The logic resource consumption is very little and it increases very slowly with the image resolution. The memory resource consumption increases proportionally with the image resolution, because the intermediate spatial pyramid and previous temporally filtered images need to be stored. However, the manufacturing cost of memory in FPGA and ASIC is usually much less than that of logic resources. The more memory consumption due to higher image resolution would not increase overall system cost drastically.

For each video sequence, our used filter coefficients and the magnification factors are listed in Table II, including the zero valued magnification factors for the topmost and the dummy

layers. These parameters were tuned for our hardware implementation to produce results visually comparable to those provided online by H.-Y. Hao *et al* [2]. Fig. 5 demonstrates the effect of motion magnification for 3 sequences. The results of PC software in [2] were converted from color space to grayscale for a fair visual comparison. For each sequence, we selected an image column (marked by a red line), and concatenated the $y-t$ slices at the vertical line in all frames. Larger image variation can be seen in the $y-t$ slices of magnified sequences. For the *baby* sequence, note the movement at the zip region due to the baby breath is magnified. For the *shadow* sequence, note the vertically back-and-forth movement of the shadow on the house become perceptible after motion magnification. In the *wrist* sequence, the ulnar pulse is not visible in the original video but is clearly revealed in magnified sequence. The magnified videos for all the 9 sequences produced by our core are provided in the supplement.

Finally, to demonstrate the acceleration capability of our core, we compared the processing speed of our hardware prototype on a ZC706 platform running at 90 MHz to a PC with Intel i7-4790 CPU running at 3.60 GHz under 64-bit Windows operating system. The desktop version used the same algorithm parameters and was implemented in C# language with fixed-point data types. As Table III shows, the speedup factor was above 27. The frame rates were averaged over all the frames in each benchmark sequence.

V. CONCLUSION

This paper proposes a real-time motion magnification core for smart image sensors. The core is based on the EVM algorithm, and employs pixel stream processing architecture that features readiness to fit sensing components, higher processing performance, and lower system cost. The processing blocks in this core are data-driven without complicated global controls. These blocks are all running in pipelines to maximize their throughput. The proposed core was prototyped on an FPGA platform. It processes 90M pixels/second, and consumes very few logic resources and moderate amount of memory resources. The experimental results of the prototype core are visually comparable to the PC-software based results. This demonstrates that our core has good potential in smart image sensors for high-speed low-cost embedded applications.

In our future work, we will employ the phase-based [15] instead of EVM algorithm for larger magnification factor, use address-event representation (AER) processors [16], [17] for higher power efficiency, and integrate our magnification core into a custom VLSI system with a real physical image sensor.

Acknowledgments

The authors would like to thank Shrinivas Pundlik for his contribution in polishing this manuscript. This work is supported in part by NIH grants AG049174 and EY025913.

References

1. Liu C, Torralba A, Freeman W, Durand F, Adelson E. Motion magnification. *ACM Transactions on Graphics*. 2005; 24(3):519.

2. Wu H, Rubinstein M, Shih E, Gutttag J, Durand F, Freeman W. Eulerian video magnification for revealing subtle changes in the world. *ACM Transactions on Graphics*. 2012; 31(4):1–8.
3. Wadhwa N, Rubinstein M, Durand F, Freeman W. Phase-based video motion processing. *ACM Transactions on Graphics*. 2013; 32(4):1.
4. Wadhwa N, Wu H-Y, Davis A, et al. Eulerian video magnification and analysis. *Communications of the ACM*. 2016; 60(1):87–95.
5. Bharadwaj S, Dhamecha T, Vatsa M, Singh R. Computationally Efficient Face Spoofing Detection with Motion Magnification. 2013 IEEE Conference on Computer Vision and Pattern Recognition Workshops. 2013
6. Le Ngo A, Oh Y, Phan R, See J. Eulerian emotion magnification for subtle expression recognition. 2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP). 2016
7. McLeod A, Baxter J, de Ribaupierre S, Peters T. Motion magnification for endoscopic surgery. *Medical Imaging 2014: Image-Guided Procedures, Robotic Interventions, and Modeling*. 2014
8. Chen J, Wadhwa N, Cha Y, Durand F, Freeman W, Buyukozturk O. Modal identification of simple structures with high-speed video using motion magnification. *Journal of Sound and Vibration*. 2015; 345:58–71.
9. Amaravati A, Xu K, Romberg J, Raychowdhury A. A 130nm 165nJ/frame Compressed-Domain Smashed-Filter Based Mixed-Signal Classifier for “In-sensor” Analytics in Smart Cameras. *IEEE Transactions on Circuits and Systems II: Express Briefs*. 2017
10. Schmitz J, Gharzai M, Balkir S, Hoffman M, White D, Schemm N. A 1000 frames/s Vision Chip Using Scalable Pixel-Neighborhood-Level Parallel Processing. *IEEE Journal of Solid-State Circuits*. 2017; 52(2):556–568.
11. Leñero-Bardallo JA, Häfliger P. A dual-operation-mode bio-inspired pixel. *IEEE Transactions on Circuits and Systems II: Express Briefs*. 2014; 61(11):855–859.
12. Tomasi M, Pundlik S, Luo G. FPGA–DSP co-processing for feature tracking in smart video sensors. *Journal of Real-Time Image Processing*. 2014; 11(4):751–767.
13. Burt P, Adelson E. The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions on Communications*. 1983; 31(4):532–540.
14. Smith S. *The scientist and engineer’s guide to digital signal processing*. 1st. San Diego, Calif: California Technical Pub; 1997.
15. Wadhwa N, Rubinstein M, Durand F, Freeman WT. Riesz pyramids for fast phase-based video magnification. *IEEE International Conference on Computational Photography (ICCP)*. 2014:1–10.
16. Camunas-Mesa L, Zamarreno-Ramos C, Linares-Barranco A, et al. An event-driven multi-kernel convolution processor module for event-driven vision sensors. *IEEE Journal of Solid-State Circuits*. 2012; 47(2):504–517.
17. Yousefzadeh A, Plana LA, Temple Steve, et al. Fast Predictive Handshaking in Synchronous FPGAs for Fully Asynchronous Multisymbol Chip Links: Application to SpiNNaker 2-of-7 Links. *IEEE Transactions on Circuits and Systems II: Express Briefs*. 2016; 63(8):763–767.

Fig. 1.
The proposed modified EVM algorithm flow.

Fig. 2.
The proposed motion magnification core architecture.

Fig. 3. The computational block diagrams. (a) The temporal bandpass filter block. (b) The Gaussian filter block. (c) The *mag-merge-upsample* block.

Fig. 4.
The FPGA prototype of the proposed core and the evaluation system.

Fig. 5.

Some motion magnification results by our prototype core in comparison with that produced by PC software in [2]. (a) *baby* sequence. (b) *shadow* sequence. (c) *wrist* sequence. The $y-t$ slices are created by concatenating image column at the red vertical line of all frames, to illustrate the image change over the time.

TABLE I
FPGA PROTOTYPE RESOURCE COMSUMPTION ON ZC706 PLATFORM

Resolution × # of layers		Logic resources			Memory resources		
		Slice reg. (437,200)	Slice LUT as logic (218,600)	DSP/Multiplier (900)	Slice LUT as memory (70,400)	Block RAM (545)	
	<i>guitar</i>	3233 (0.7%)	3466 (1.6%)	7 (0.8%)	2559 (3.6%)	58.5 (11%)	
	<i>camera</i>	3935 (0.9%)	3349 (1.5%)	8 (0.9%)	1287 (1.8%)	121 (22%)	
	<i>baby2, subway, wrist</i>	4006 (0.9%)	4008 (1.8%)	8 (0.9%)	1827 (2.6%)	124 (23%)	
	<i>face</i>	5016 (1.1%)	5011 (2.3%)	9 (1.0%)	2183 (3.1%)	239.5 (44%)	
	<i>face2</i>	5083 (1.2%)	5166 (2.4%)	9 (1.0%)	2327 (3.3%)	239.5 (44%)	
	<i>shadow</i>	5080 (1.2%)	5244 (2.4%)	9 (1.0%)	2975 (4.2%)	239.5 (44%)	
	<i>baby</i>	5051 (1.2%)	5505 (2.5%)	9 (1.0%)	3435 (4.9%)	239.5 (44%)	

TABLE II

ALGORITHM PARAMETERS FOR EACH VIDEO SEQUENCE

	# of layers	Mag. factors	Bandpass filter				
			f_s	f_{ch}	f_{cl}	c_l	c_h
<i>baby</i>	7	0, 6, 10, 10, 10, 0	30	0.4	3	68×2^{-7}	118×2^{-7}
<i>baby2</i>	6	0, 6, 10, 10, 10, 0	30	0.4	3	68×2^{-7}	118×2^{-7}
<i>camera</i>	6	0, 38, 79, 120, 120, 0	300	20	50	45×2^{-7}	84×2^{-7}
<i>face</i>	7	0, 4, 9, 21, 40, 40, 0	30	0.5	5	45×2^{-7}	115×2^{-7}
<i>face2</i>	7	0, 1, 3, 8, 18, 20, 0	30	0.5	5	45×2^{-7}	115×2^{-7}
<i>guitar</i>	5	0, 48, 50, 50, 0	600	72	92	49×2^{-7}	60×2^{-7}
<i>shadow</i>	7	0, 3, 9, 10, 10, 10, 0	30	0.5	5	45×2^{-7}	115×2^{-7}
<i>subway</i>	6	0, 3, 8, 19, 39, 0	30	1.8	3.1	67×2^{-7}	88×2^{-7}
<i>writs</i>	6	0, 8, 18, 20, 20, 0	30	0.4	3	68×2^{-7}	118×2^{-7}

Note: The f_s is the frame rate that the benchmark sequence was captured at in [2], not the running frame rate of our core. The f_{cl} , f_{ch} are the lowpass and highpass cutoff frequencies of the bandpass filter. They are converted to the filter coefficients c_l , c_h using Eq. (2).

Author Manuscript

Author Manuscript

Author Manuscript

Author Manuscript

TABLE III

PERFORMANCE COMPARISON

	<i>guitar</i> (432×192)	<i>camera</i> (512×384)	<i>baby2, subway, wrist</i> (640×352)	<i>face</i> (528×592)	<i>face2</i> (570×718)	<i>Shadow</i> (800×520)	<i>Baby</i> (960×544)
On PC	40 fps	16.4 fps	14.5 fps	10.4 fps	7.9 fps	7.7 fps	6.2 fps
On our core	1085 fps	457.8 fps	399.5 fps	287.9 fps	219.9 fps	216.3	172.3
Speedup	27.1×	27.9×	27.6×	27.7×	27.8×	28.1×	27.8×