Acceleration of High Dynamic Range Imaging Pipeline Based on Multi-threading and SIMD Technologies

Radosław Mantiuk and Dawid Pająk

Szczecin University of Technology
Zolnierska 49, 71-210 Szczecin, Poland
{rmantiuk,dpajak}@wi.ps.pl
http://zgk.wi.ps.pl

Abstract. In this paper we present a holistic approach to CPU based acceleration of the high dynamic range imaging (HDRI) pipeline. The high dynamic range representation can encode images regardless of the technology used to create and display them, with the accuracy that is only constrained by the limitations of the human eye and not a particular output medium. Unfortunately, the increase in accuracy causes significant computational overhead and effective hardware acceleration is needed to ensure a utility value of HDRI applications. In this work we propose a novel architecture of the HDRI pipeline based on CPU SIMD and multi-threading technologies. We discuss the impact on processing speed caused by vectorization and parallelization of individual image processing operations. A commercial application of the new HDRI pipeline is described together with evaluation of achieved image processing speed-up.

Keywords: high dynamic range imaging, SIMD architecture, SSE, multi-threading architecture, image processing, computer visualization.

1 Introduction

The advances in high dynamic range imaging (HDRI), especially in display and camera technology, have a significant impact on existing imaging systems. The assumptions of traditional low-dynamic range imaging, designed for paper print as a major output medium, are ill suited for the range of visual material that is shown on modern displays. The high dynamic range representation can encode images regardless of the technology used to create and display them, with the accuracy that is only constrained by the limitations of the human eye and not a particular output medium.

The disadvantage of HDRI technology is computational complexity. A single pixel in an high dynamic range image consumes 4 times more memory storage in comparison to a low dynamic range image (3 floating point numbers (4-bytes long each) against 3-bytes in the low dynamic range representation). This complexity means that HDRI pipeline is not used in many applications which would

M. Bubak et al. (Eds.): ICCS 2008, Part I, LNCS 5101, pp. 780–789, 2008.

significantly benefit from HDR accuracy. For example, processing of huge datasets from medical computer tomography or RAW photographs is problematic for typical personal computers or laptops. Moreover, the constant increase of image resolution and complexity of image processing algorithms will make this problem even worse in the future.

In this paper we argue that by usage of modern CPU technologies, it is possible to accelerate the image processing of the HDRI pipeline significantly. The acceleration can be achieved based on existing CPU capabilities: the SIMD instruction set, multi-processor and multi-core architectures. SIMD (Single Instruction Multiple Data) instructions allow to speed-up processing of floating point vector data, which can represent HDR pixels. Because of the independent transformation of individual pixels, most HDRI algorithms are well suited for parallel processing. Multi-threading architecture accelerates such processing almost by a factor of the available threads.

A goal of efficient HDRI computing is to accelerate the whole HDRI pipeline rather than to speed-up individual operations. In this paper we propose the architecture of the accelerated pipeline which benefits from careful optimization of HDRI algorithms and from effective RAM memory management. We also introduce queueing techniques that enable grouping many simple operations into one complex command path. This way automatic optimization of CPU hardware usage is implemented and effective acceleration of complex algorithms is possible.

We review existing SIMD and multi-threading based technologies in section 2. The concept of high dynamic range imaging pipeline and its possible acceleration techniques are discussed in Section 3. In section 4 we present an architecture of our novel HDRI pipeline which uses SIMD and multi-threading technologies to speed-up data processing. We then describe a software package that operates on the new HDRI pipeline (section 5) and discuss achieved results.

2 Previous Work

A general approach to image processing using SIMD and parallel hardware can be found in a few software packages. The VIPS (VASARI Image Processing System) library [7] seems to be the most known LGPL system for processing huge images. It divides images into small arrays and uses multi-threading to effectively process them on SMP (Symmetric Multiprocessor) computers. Intel Integrated Performance Primitives (Intel IPP) [8] is a library of multi-core-ready, optimized software functions for multimedia data processing. Careful programming in plain C/C++ code and compilation based on IPP compilers can speed-up applications. The Image Processing Toolbox (IPT) [10] provides a set of functions for image manipulation and analysis. The IPT capabilities include SIMD and multi-threading optimized color space transformations, linear filtering, mathematical morphology, geometric transformations, image filtering and analysis. Acceleration is achieved based on the O-Matrix engine [10] that supports fast matrix processing. Multi-threading and SIMD architecture is also exploited by the GENIAL (GENeric Image Array Library) [9] library to speed-up computation of signal processing

algorithms. The architecture of GENIAL is based on the same conventions as the Standard Template Library (STL), consisting of containers, iterators, adaptors, function objects and algorithms. The intensive use of templates makes it possible for the library to automatically adapt calculations on containers to the specified problem in order to achieve faster execution. There are a few acceleration toolkits in the medical imaging community. ITK (Insight Segmentation and Registration ToolKit) [11] is a library for image segmentation and registration. Another available choice, MITK (Medical Imaging ToolKit) [12] uses CPU SIMD instructions to accelerate matrix and vector computations, and linear and tri-linear interpolation computations. Both toolkits provide a general framework for medical imaging rather than a set of highly optimized image processing functions.

All these approaches seem to be general and not optimized for HDRI processing. In particular they are not intended for throughout rendering of the HDRI pipeline. In this paper we propose a more efficient solution at the cost of rejecting generality of computations. We present a holistic approach to CPU based acceleration of the HDRI pipeline. We don't deal with GPU (Graphics Processor Image) acceleration techniques [1]. The usage of GPU seems to be promising for fast HDRI processing but this technology is not common on many platforms (e.g. mobile phones or PDA devices). Moreover, advanced GPU capabilities (e.g. shader support) are not well standardized yet so we leave the GPU acceleration as future work.

3 Acceleration of the HDRI Pipeline

High dynamic range imaging [2] is a new paradigm that involves a highly accurate representation of images. As it originates from light simulation (computer graphics rendering) and measurements, the pixels of HDR images are assigned a physical meaning. This highly accurate representation of images gives an unique opportunity to create a common imaging framework, that could meet the requirements of different imaging disciplines.

Figure 1 illustrates an example of the HDRI pipeline [3] that starts with acquisition of a real world scene or rendering of an abstract model using computer graphics techniques and ends at a display. This pipeline overcomes shortcomings of a typical graphics pipeline that doesn't support devices of a higher dynamic range or a wider color gamut [4].

The drawback of the HDRI pipeline is that much more data has to be processed in comparison to a typical 8-bit pipeline. Pixels of an HDR image are represented by a vector of floating point values: one 4-byte floating point number for each of 3 or more color channels (e.g. in the case of multi-spectral imaging more than 10 channels should be supported). Number of bbp (bits per pixel) is then four or more times higher than for 8-bit images. Additionally, it should be considered that most HDRI devices generate huge sets of data because of growing sampling resolutions of input and output devices. Like in typical images, dimensionality of HDRI images is important and change of context in both horizontal and vertical directions cannot be neglected. Summing up, to achieve

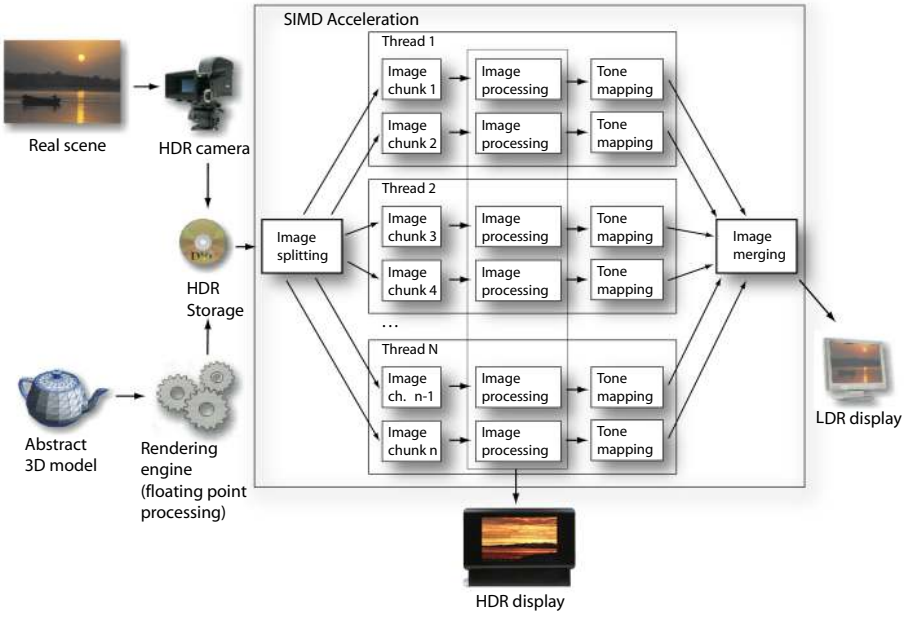

Fig. 1. High dynamic range imaging pipeline accelerated by SIMD and multi-threading operations

the same performance as for the low dynamic range pipeline, the data in the HDRI pipeline need to be processed much faster.

A recent development of CPU hardware allows for the speed-up HDRI processing significantly. In particular the SIMD instruction can be exploited effectively by processing many data in one CPU cycle. HDRI processing is susceptible to parallelization so usage of multi-threading accelerates computations. To exploit SIMD and multi-threading efficiently, we selected a set of algorithm crucial to HDRI processing. Then, we propose a new architecture of the HDRI pipeline.

A set of selected algorithms is depicted in Figure 2. From the acceleration viewpoint, the most important group of operations is matrix arithmetic. This group covered both matrix-by-matrix operations (e.g. matrix multiplication) and scalar-by-matrix operation (e.g. multiplication of all matrix elements by a scalar value). Also, matrix manipulation algorithms, like transposition or vertical and horizontal shift, should be considered. Channel masking and pixel masking can eliminate selected color channels or pixels from pipeline processing based on conditional expressions. The accumulation algorithms, such as computation of a sum of pixel values in an image area, is time consuming and should be accelerated. In many cases HDR images must be transformed by a non-linear functions and Look-Up-Table (LUT) is the fastest and simplest way to proceed. Finally, a selected group of advanced image processing algorithms are accelerated. This group includes image scaling, color space conversions and color profile conversions.

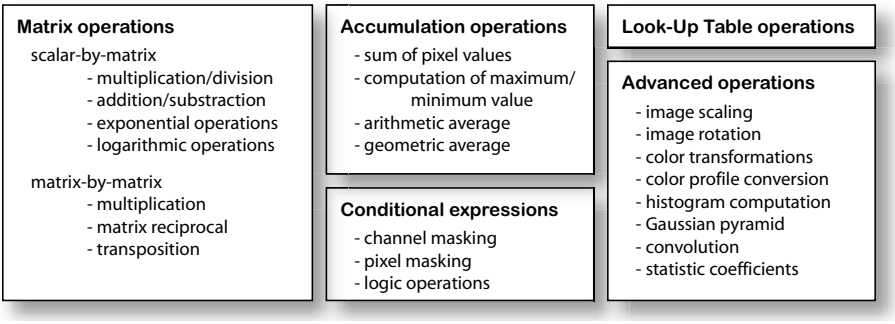

Fig. 2. HDRI operations crucial for fast image processing

4 Using SIMD Operations and Multi-threading in HDRI Processing

In Figure 1 a novel HDRI pipeline accelerated by CPU SIMD operations and multi-threading is presented. The main goal of the pipeline processing is to exploit the SIMD and multi-threading architectures efficiently. Moreover, data interchange between CPU and computer RAM memory should be limited as much as possible. All intermediate and temporary results should be stored in CPU registers or CPU cache memory. If a larger temporary storage is required (e.g. in local tone mapping operators [2]), algorithm could exploit L1 cache, as it offers lower access latency and greater performance than system memory. However in this case, an HDR image must be divided into suitable chunks due to limited cache size.

Parallel processing is exploited in a new pipeline. We divide an HDR image into arrays/chunks and the chunks are processed independently. The size of an array is limited by the size of CPU L1 cache rather than the number of available threads or processors. We noticed that even in a one-thread system, it is faster to process small chunks of data rather than the whole image at once. In the pipeline implementation a fixed number of threads (equal to the number of execution units reported by an operating system) is created at run-time. Threads become active only when new tasks are assigned to them and go to sleep right after they finish processing scheduled tasks. Thread management is handled by operating system. It is the operating system's responsibility to identify Hyper-threading, multi-threading or multi-processor hardware and manage threads efficiently.

The goal of SIMD computation is to perform a single operation on many data elements. Modern CPU hardware is equipped with a set of SIMD arithmetic, logical, comparison and conversion instructions. They process 128-bit words in one CPU cycle so a 4-times speed-up of basic operations is potentially possible. Almost all current CPUs offer the SIMD instruction set. Examples are SSE [5] in Intel and AMD processors or AltiVec in IBM's Power PC.

HDRI processing is especially suited to acceleration based on the SIMD architecture. A common 4-channel RGBA representation (red, green, blue, and alpha channels) of an HDR pixel can be considered as one 128-bit word and all channels can be processed simultaneously. The CPU SIMD instruction set delivers most of operations required in HDRI computing [6]. In the case of advanced operations (like logarithm computation or exponentiation), which are not available, we use existing instructions to approximate results.

For example to compute $log_2(x)$, we use specific features of a single precision floating-point number representation. As defined in the IEEE 754-1985 specification, a single precision number is described by the equation: $s * 2^e * m$, where s is a sign bit, e is an 8-bit exponent and m is a 24-bit normalized mantissa. We calculate the $log_2(x)$ value by extracting the exponent from the number representation and adding it to the approximation of $log_2(m)$ of extracted mantissa: $log_2(x) = log_2(2^e * m) = e + log_2(m)$, where $x > 0$. In our implementation we use the Chebyshev mini-max fifth degree polynomial to approximate the function $log_2(m)$. This technique results in a small relative error (10^{-6}) and can be implemented very efficiently on SIMD architecture.

Most of the operations performed inside the HDRI pipeline are executed on luminance (1 channel) value of HDR pixels. This makes the calculation even more efficient as we process 4 pixels in each instruction. An example of this set of operations is a Look-up table transformation which lets the user apply a custom non-linear transformation on input luminance values (e.g. custom global TMO curve). Given a set of non-overlapping input ranges ($[a_0, b_0]..[a_n, b_n]$) and their mapped counterpart values ($[c_0, d_0]..[c_n, d_n]$) we transform luminance value l using the formula: $l_e = c_x + ((l - a_x)/(b_x - a_x)) * (d_x - c_x)$, where $a_x \leq l < b_x$. The most time consuming task is finding the right input range for every pixel. By using SIMD selective write operations and bit masking we speed-up the procedure by finding the ranges for 4 pixels at the same time. Because of the characteristics of image data (neighbourhood pixels have relatively small differences in luminance values) the performance drawback coming from redundant loop passes (some pixels in the vector might have their range locked but we continue the search until ranges for all 4 pixels are found) is, in most cases, negligible.

5 Example Application: The HDRI Library

We implemented the HDRI pipeline as a Windows dynamic library. All public methods in the library are accessed through a simple C API similar to OpenGL with internal state machine on the library side. The library uses multi-threading and vector processing capabilities. The functions are manually optimized with intensive usage of MMX/SSE/SSE2 intrinsic code. The initialization code is able to detect the features of the CPU (by invoking the *cpuid* instruction) and choose the best possible code path for the library methods. Also a special 64-bit version is available for Win64 systems (additional performance gain comes from the increased number of CPU SIMD registers). The library itself was implemented in C++.

The activity diagram in Figure 3 shows simplified processing stages of HDRI pipeline. Actual pipeline stages implementation is much more complicated and includes additional features/modules. We skip them because they do not influence the design of the pipeline acceleration methods.

A synchronization of all working threads is required to gather the results, calculate required image specific factors and then move to the next stage. The synchronization of threads is implemented via the Win32 event system which is known for low latency and quick kernel dispatching time. Even if we use the most efficient synchronization method available, this task tends to be the slowest element of the pipeline processing (the delays are not even correlated with image data size). By carefully designing the pipeline stages and operation types at each stage we minimized the synchronization count to the absolute minimum.

Besides the fixed functionality pipeline we have implemented a mechanism which lets the programmer combine many simple SIMD accelerated functions (addition, multiplication, division, etc.) with their corresponding arguments into one complex command, which is then executed by multi-threading, chunk processing engine of the pipeline. This queueing method offers additional flexibility to computational abilities of the library, however it comes with a price of degraded locality of calculations and increased memory bandwidth usage. Thus it is not able to compete in terms of performance with the fixed functionality pipeline.

A set of 5 HDR images (6 M-pixels each) was used to conduct performance tests of the library. The tests covers both low level (basic data processing operations) and advanced HDRI processing operations. The timings from the upper part of table 1 are arithmetic means of results we got for individual test images. The lower part of the table covers tests for execution of the whole HDRI pipeline starting from HDR image blending (for 5 input images) and ending at tone mapping (TMO) and LDR image generation. The test platform was a Windows XP based PC equipped with Athlon X2 3800+ dual core CPU and 2GB DDR RAM.

Noticeable speed-ups are visible in both internal functions as well as in overall pipeline performance. The computational power of SIMD architecture is especially exposed in calculation of log-average (arithmetic mean of logarithms) or sRGB gamma correction where approximated functions are used instead of their built-in counterparts.

Decreased acceleration ratio of functions with conditional execution (compare timings for log-average and conditional log-average) is caused by the streaming nature of SIMD computation: we calculate values of all elements in the vector and write the result depending on the condition bit mask. In the case of SISD implementation we only calculate the results for elements which pass the condition statement.

Operations performing well on a super-scalar FPU (e.g. image blending) or resistant to vectorization due to algorithm nature (tone mapping) still take advantage of parallel processing and speed-up close to the theoretical limit of 2 (number of cores in our test system). False color image generation[1] performance

[1] A process of encoding the HDR image luminance into an LDR image where certain luminance values are mapped to a predefined RGB triplets from LUT.

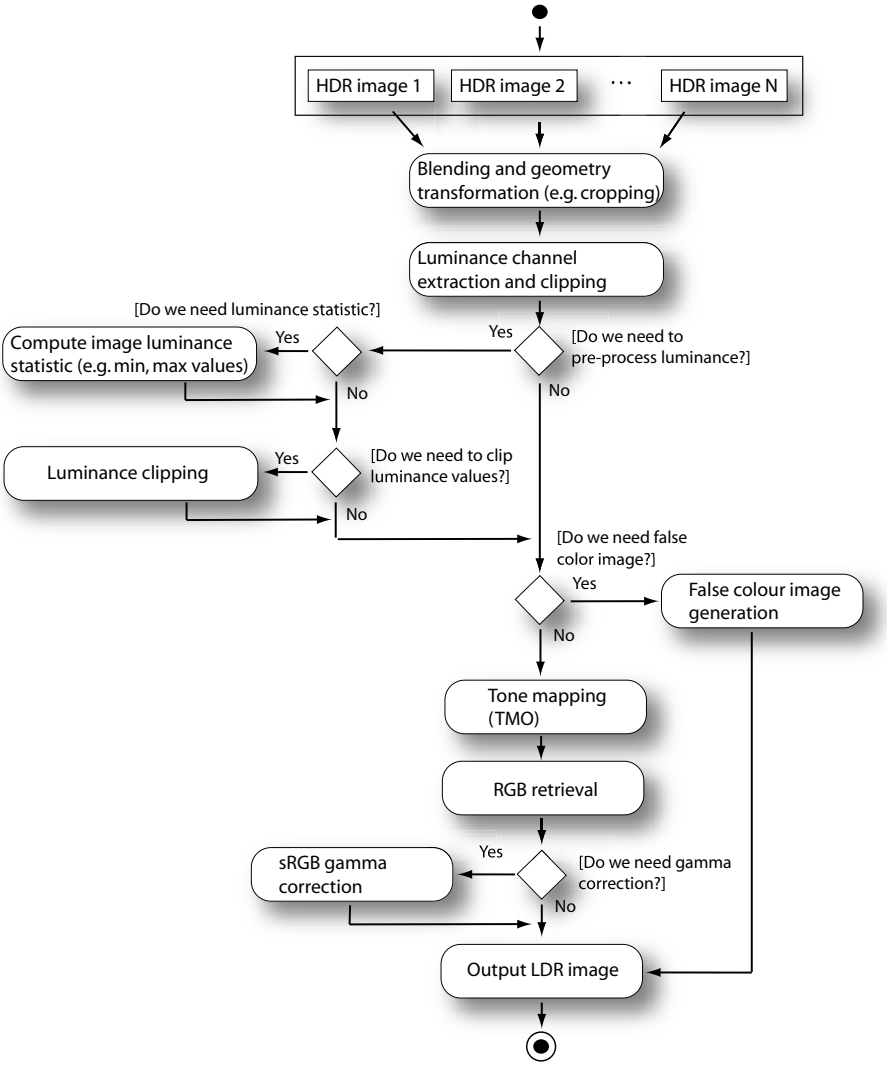

Fig. 3. Selected HDRI library pipeline stages (simplified). The default configuration of the HDRI pipeline combines a specified number of the input HDR images into one intermediate image, which is then processed and written as an output LDR image. Most of the functions work on the luminance channel which is extracted in the preprocessing stage. At the end of pipeline the results of luminance processing are applied on all channels of input HDR image.

does not scale linearly with the number of available execution units and seem to be bound by memory bandwidth.

We have compared our implementation with the VIPS library in low level function performance. All tested functions of HDRI library performed better than

Table 1. Computation time of internal pipeline functions. For SIMD implementation both single-threaded (st) and multi-threaded (mt) tests were performed. The speed-up is a ratio between measured time of a single-threaded FPU code and SIMD multi-threaded implementation.

Operation	Time [ms]			Speed-up
	FPU(st)	SIMD(st)	SIMD(mt)	
HDR image blending	257	238	139	1.85
Log-average	395	33	19	20.79
Conditional log-average	238	43	25	9.52
XYZ to RGB conversion	64	38	20	3.2
Conditional XYZ to RGB conversion	90	49	22	4.09
sRGB gamma correction	1971	302	157	12.55
Global photographic TMO [13]	3146	590	360	8.74
Gamma TMO	3356	621	375	8.95
False color image generation (LUT example)	690	321	220	3.14

their VIPS equivalents. For example, the computation speed of log-average is about 11 times faster in our solution (both test programs utilize multi-threading architecture). This was expected as VIPS library is architecture independent and does not use SIMD instruction set or math functions approximations.

6 Conclusions and Future Work

The limitations of the existing low-dynamic range imaging technology can be addressed and eliminated in the HDR imaging pipeline that offers higher precision of visual data. In this work we outline acceleration techniques for processing HDR images. We use the SIMD and multi-threading technologies available in present CPU hardware to overcome computation bottlenecks. Those technologies, together with the proposed novel data processing architecture, make the HDRI pipeline as fast as the traditional pipeline. In the final section of the paper we describe the software library implemented based on the proposed design. The utility values of this library was proved in commercial applications (they will be available on the market in the near future).

Another increase of computation speed of the HDRI library could be achieved using specialized instructions from the SSE3/SSE4 extensions (e.g. usage of hardware dot product and horizontal data processing instructions could speed up accumulation operations by a factor of 4). At the time of writing of the implementation the CPUs with SSE3/SSE4 support were not propagated enough in the market to count this code path in the library design but we plan to use those operations in the future. We have also been extending functionality of the library to accelerate more complex image processing operations (e.g. histogram computation or Gaussian pyramid usage). In the future we plan to port the library to GPU hardware.

Acknowledgments. The research work, which results are presented in this paper, was sponsored by Polish Ministry of Science and Higher Education (years 2006-2008).

References

1. Owens, J.D., Luebke, D., Govindaraju, N., Harris, M., Kruger, J., Lefohn, A.E., Purcell, T.: A survey of general-purpose computation on graphics hardware. In: Proc. of Eurographics 2005, State of the Art Reports, September 2005, pp. 21–51 (2005)
2. Reinhard, E., Ward, G., Pattanaik, S., Debevec, P.: High Dynamic Range Imaging. In: Data Acquisition, Manipulation, and Display. Morgan Kaufmann, San Francisco (2005)
3. Mantiuk, R., Krawczyk, G., Mantiuk, R.: High Dynamic Range Imaging Pipeline: Merging Computer Graphics, Physics, Photography and Visual Perception. In: Spring Conference on Computer Graphics 2006 Posters and Conference Materials, April 20-22, pp. 37–40 (2006)
4. Mantiuk, R., Krawczyk, G., Mantiuk, R., Seidel, H.P.: High Dynamic Range Imaging Pipeline: Perception-motivated Representation of Visual Content. In: Proc. of SPIE. Human Vision and Electronic Imaging XII. 649212, vol. 6492 (2007)
5. Intel64 and IA-32 Architectures Optimization Reference Manual (May 2007)
6. Intel Architecture Software Developer Manual, Instruction Set Reference, vol.2 (1999)
7. Martinez, K., Cupitt, J.: VIPS - a highly tuned image processing software architecture. In: Proceedings of IEEE International Conference on Image Processing 2, Genova, vol. 2, pp. 574–577 (2005)
8. Taylor, S.: Intel Integrated Performance Primitives Book. ISBN 0971786135, ISBN13 9780971786134 (2004)
9. Laurent, P.: GENIAL - GENeric Image Array Library, `http://www.ient.rwth-aachen.de/team/laurent/genial/genial.html`
10. Harmonic Software Inc.: IPT - The Image Processing Toolbox for O-Matrix, `http://www.omatrix.com/ipt.html`
11. Ibanez, L., Schroeder, W., Ng, L., Cates, J.: The ITK Software Guide, 2nd edn., Kitware, Inc. Publisher (November 2005)
12. Zhao, M., Tian, J., Zhu, X., Xue, J., Cheng, Z., Zhao, H.: The Design and Implementation of a C++ Toolkit for Integrated Medical Image Processing and Analysis. In: Proc. of SPIE Conference, vol. 6, pp. 5367–5374 (2004)
13. Reinhard, E., Stark, M., Shirley, P., Ferwerda, J.: Photographic Tone Reproduction for Digital Images. ACM Trans. on Graphics 21(3), 267–276 (2002)