Biomedical Optics EXPRESS

Accelerating mesh-based Monte Carlo method on modern CPU architectures

Qianqian Fang [1,*] **and David R. Kaeli** [2]

[1]*Martinos Center for Biomedical Imaging, Massachusetts General Hospital, Charlestown, MA 02129 USA*

[2]*Department of Electrical and Computer Engineering, Northeastern University, Boston MA 02115 USA*

**fangq@nmr.mgh.harvard.edu*

Abstract: In this report, we discuss the use of contemporary ray-tracing techniques to accelerate 3D mesh-based Monte Carlo photon transport simulations. Single Instruction Multiple Data (SIMD) based computation and branch-less design are exploited to accelerate ray-tetrahedron intersection tests and yield a 2-fold speed-up for ray-tracing calculations on a multi-core CPU. As part of this work, we have also studied SIMD-accelerated random number generators and math functions. The combination of these techniques achieved an overall improvement of 22% in simulation speed as compared to using a non-SIMD implementation. We applied this new method to analyze a complex numerical phantom and both the phantom data and the improved code are available as open-source software at *http://mcx.sourceforge.net/mmc/*.

OCIS codes: (170.3660) Light propagation in tissues; (170.5280) Photon migration; (170.7050) Turbid media

References and links

1. D. A. Boas, D. H. Brooks, E. L. Miller, C. A. DiMarzio, M. Kilmer, R. J. Gaudette, and Q. Zhang, "Imaging the body with diffuse optical tomography," IEEE Signal Proc. Mag. **18**, 57–75 (2001).
2. A. P. Gibson, J. C. Hebden, and S. R. Arridge, "Recent advances in diffuse optical imaging," Phys. Med. Biol. **50**, R1–R43 (2005).
3. L. H. Wang, S. L. Jacques, and L. Q. Zheng, "MCML - Monte Carlo modeling of light transport in multilayered tissues," Comput. Meth. Prog. Bio. **47**, 131–146 (1995).
4. D. A. Boas, J. Culver, J. Stott, and A. Dunn, "Three dimensional Monte Carlo code for photon migration through complex heterogeneous media including the adult human head," Opt. Express **10**, 159–170 (2002).
5. Q. Fang and D. A. Boas, "Monte Carlo simulation of photon migration in 3D turbid media accelerated by graphics processing," Opt. Express **17**, 20178–20190 (2009).
6. H. Li, J. Tian, F. Zhu, W. Cong, L. V. Wang, E. A. Hoffman, and G. Wang, "A mouse optical simulation environment (MOSE) to investigate bioluminescent phenomena in the living mouse with the monte carlo method," Academ. Radiol. **11**, 1029–1038 (2004).
7. E. Margallo-Balbás and P. J. French, "Shape based Monte Carlo code for light transport in complex heterogeneous Tissues," Opt. Express **15**, 14086–14098 (2007).
8. N. Ren, J. Liang, X. Qu, J. Li, B. Lu, and J. Tian, "GPU-based Monte Carlo simulation for light propagation in complex heterogeneous tissues," Opt. Express **18**, 6811–6823 (2010).
9. C. Wächter, "Quasi-Monte Carlo light transport simulation by efficient ray tracing," Ph.D. dissertation (Ulm University, Ulm, Germany, 2007).
10. I. Wald, "Realtime ray tracing and interactive global illumination," Ph.D. dissertation (Saarland Univ., Saarbrücken, Germany, 2004).
11. H. Shen and G. Wang, "A tetrahedron-based inhomogeneous Monte Carlo optical simulator," Phys. Med. Biol. **55**, 947–962 (2010).

#173095 - $15.00 USD Received 23 Jul 2012; revised 25 Sep 2012; accepted 27 Sep 2012; pub. 12 Nov 2012

12. Q. Fang, "Mesh-based Monte Carlo method using fast ray-tracing in Plücker coordinates," Biomed. Opt. Express **1**, 165–175 (2010).
13. Q. Fang, "Comment on 'A study on tetrahedron-based inhomogeneous Monte-Carlo optical simulation'," Biomed. Opt. Express **2**, 1258–1264 (2011).
14. CGAL Editorial Board, *CGAL User and Reference Manual*, 3rd ed. (2009).
15. Q. Fang and D. A. Boas, "Tetrahedral mesh generation from volumetric binary and grayscale images," in *IEEE International Symposium on Biomedical Imaging: from Nano to Macro, 2009. ISBI '09* (IEEE, 2009), pp. 1142–1145.
16. M. Shevtsov, A. Soupikov, and A. Kapustin, "Ray-triangle intersection algorithm for modern CPU architectures," in P*roceedings of GraphiCon 2007* (2007), Vol. 11, pp. 33–39.
17. J. Havel and A. Herout, "Yet faster ray-triangle intersection (using SSE4)," IEEE Trans. Visualiz. Comput. Graphics **16**, 434–438 (2010).
18. D. Badouel, *Graphics Gems* (Academic, 1990).
19. M. Saito and M. Matsumoto, *Monte Carlo and Quasi-Monte Carlo Methods 2006* (Springer, 2008).
20. J. Pommier, "Simple SSE and SSE2 optimized sin, cos, log and exp" (2007), http://gruntthepeon.free.fr/ssemath/.
21. B. Dogdas, D. Stout, A. Chatziioannou, and R. M. Leahy, "Digimouse: a 3D whole body mouse atlas from CT and cryosection data," Phys Med Biol. **52**, 577–87 (2007).
22. W. F. Cheong, S. A. Prahl, and A. J. Welch, "A review of the optical properties of biological tissues," IEEE J. Quantum Electronics **26**, 2166–2185 (1990).
23. S. Powell and T. S. Leung, "Highly parallel Monte-Carlo simulations of the acousto-optic effect in heterogeneous turbid media," J. Biomed. Opt. **17**, 045002 (2012).

1. Introduction

Near-infrared (NIR) light with wavelength between 600 and 1000 nm can penerate deep into the tissue [1], owning primarily to the relatively low optical absorption of human tissue chromorphores, namely oxy/deoxy-hemoglobin, lipids and water. This remarkable characteristic makes NIR light a suitable candidate to probe deep tissue physiology such as angiogenesis and oxygen metabolism in a safe and non-invasive manner. Substantial efforts have been made in the past decade to utilize NIR imaging to facilitate the investigation of breast cancer and human brain functions, monitoring therapy responses, and a range of other applications [2]. A notable challenge for NIR imaging is the high scattering within the tissue. The transport of low-energy NIR photons is highly nonlinear and exhibits a complex and diffusive pattern. Accurate and efficient numerical methods play an essential role in NIR-based techniques such as diffuse optical tomography (DOT) and fluorescence molecular tomography (FMT).

The Monte Carlo (MC) method is a numerical technique offering both generality and accuracy in modeling photon transport inside human tissue, and often serves as the gold standard in biophotonic applications. After 20 years of evolution, biophotonic MC algorithms have become increasingly more efficient and capable in handling complex tissue structures. One of the first MC codes, MCML [3], can only simulate problems within layered infinite media. A more general approach was proposed by Boas *et al.* for 3D heterogeneous domains using voxelated representations [4, 5]. However, to achieve higher accuracy near curved boundaries in a voxelated domain, one has to increase the global density of the grid resulting in a significant increase in memory costs and execution time. Over the past few years, several mesh-based approaches have been pursued to obtain better accuracy in modelling complex tissue boundaries. In [6], Li *et al.* reported a Mouse Optical Simulation Environment (MOSE) that allows one to define 3D heterogeneous structures using geometric primitives. Margallo-Balbás *et al.* [7] and Ren *et al.* [8] have reported methods using triangular surface meshes and ray-surface intersection calculations to propagate photons in a piece-wise-constant domain. This approach achieves better accuracy near tissue boundaries, however, the speed of the ray-surface intersection tests depends on the partitioning scheme of the surfaces [9] and the traversal of the data structure often introduces significant computational overhead [10].

In the past year, two new MC algorithms were proposed to perform photon transport simulations using a 3D volumetric mesh. These algorithms include the Tetrahedron-based Inho-

#173095 - $15.00 USD Received 23 Jul 2012; revised 25 Sep 2012; accepted 27 Sep 2012; pub. 12 Nov 2012

mogeneous MC Optical Simulator (TIM-OS) by Shen & Wang [11] and the Mesh-based MC Method (MMCM) by Fang [12]. It has been shown that both methods pursue a similar approach, but differ slightly in the ray-tracing formulation used [13]. Compared to surface-based approaches, the use of a tetrahedral mesh leads to significant savings in the overhead of testing ray-triangle intersections. On average, only 2.5 ray-triangle intersection tests are required per step in MMCM and four for TIM-OS. Combined with the recent advances in image-based mesh generation methods [14],[15], the development of these new mesh-based techniques provides a robust path to fast and accurate modeling of photon propagation inside complex anatomical structures.

Ray-tracing techniques are at the heart of both mesh-based MC techniques. It is natural to hypothesize that by incorporating the latest ray-tracing methods developed by the computer graphics community, one can further advance mesh-based MC simulations. In particular, several Single Instruction Multiple Data (SIMD)-based methods have been proposed recently, including Ward's method [10] and Shevtsov's method [16]. Recently, Havel and Herout proposed a hybrid approach (referred to as H&H hereafter) that effectively combines both methods with an efficient Streaming SIMD Extensions (SSE) implementation [17]. Moreover, in [16], Shevtsov *et al.* described a branch-less ray-tracing strategy to take advantage of the pipelining mechanism in modern CPU architectures. Incorporating these innovations into biophotonic MC simulation is expected to improve overall speed. Our main interest in this report is to discuss the implementation and comparison of four ray-tracing techniques: a Plücker-based ray-tracer, a Badouel-based ray-tracer and their SSE counterparts, in a mesh-based MC simulation.

The contribution of this work is to demonstrate the use of compteporary ray-tracing techniques for efficient MC simulations on a modern CPU processor. We highlight the ray-tracing nature of biophotonic MC simulations, and suggest that a thorough investigation to the ray-tracing technique state-of-the-art can not only guide the performance improvement of many existing MC algorithms, but also provide inspirations for the future generations of MC algorithms. In the remaining sections, we first introduce the general structure of a mesh-based Monte Carlo photon simulation. We subsequently examine the computational bottlenecks present in non-SSE implementations with a profiling analysis. Using the knowledge learned from the profiling study, we improve the simulation performance by incorporating SSE in the ray-tracing, random number generation and the calculations of math functions. We then examine the improvement in simulation speed by re-profiling the SSE-enabled codes and report the improved runtimes. We conclude with a brief discussion on limitations and future prospects.

2. Methods

2.1. Mesh-based Monte Carlo Method

In a mesh-based MC simulation, the problem domain is first discretized using a tetrahedral mesh (or other shapes [12]). This is followed by the launching of a large number of photon packets each initialized with a unitary weight. Each simulated photon packet enters the domain at a specified source location $\mathbf{p}$, enclosed by a tetrahedron e_0, along an initial directional vector, $\mathbf{v}$. Position $\mathbf{p}$ and vector $\mathbf{v}$ define a 3D "ray". For each movement, we calculate a random scattering length d along $\mathbf{v}$ based on the local scattering coefficient (μ_s) inside e_0 [3]. This ray is then tested against the four triangular faces of e_0 using a ray-triangle intersection test [9] to determine the exiting point $\mathbf{p}_x$. If $||\mathbf{pp}_x|| \leq d$, we move the photon to $\mathbf{p}_x$ by setting $\mathbf{p} \leftarrow \mathbf{p}_x$ and update e_0 by the index of the neighboring element at the exiting triangle, and repeat the above ray-tetrahedron test. If $||\mathbf{pp}_x|| > d$, we then move the photon to the end of the scattering path by $\mathbf{p} \leftarrow \mathbf{p} + \mathbf{v}d$, and start a new scattering event. To do so, we generate 1) an exponentially-distributed random scattering length d, 2) a $[0, 2\pi]$ uniformly-distributed random azimuth angle, and 3) a random elevation angle based on the Henyey-Greenstein phase function [3, 4]. With

#173095 - $15.00 USD Received 23 Jul 2012; revised 25 Sep 2012; accepted 27 Sep 2012; pub. 12 Nov 2012

the updated propagation vector **v** and d, we continue to propagate the photon until it exits the domain. As the photon moves along its trajectory, the photon packet weight is attenuated exponentially based on the Beer-Lambert law [1] using the absorption coefficient (μ_a) of each enclosing element. The weight-loss is deposited into the mesh using the Barycentric coordinates calculated from the above ray-tracing step [12].

When a photon hits a boundary between two media with different refractive indices, a reflection coefficient, R, is calculated based on Fresnel's law. A uniformly distributed random number, $s \in [0,1]$, is selected. If $s > R$, we transmit the photon across the boundary and update **v** by the refraction direction. Otherwise, the photon stays in the current medium and propagates along a reflected direction. When a photon passes through an external surface of the mesh, we terminate the photon and launch a new one. At the end of the simulation, we output the volumetric flux (or fluence) accumulated over the mesh by all simulated photon packets and save the distances traveled by the exiting photons inside each tissue type (referred to as the partial path-lengths). With the path-length information, one can quickly re-calculate the optical intensity at the detectors without repeating the entire simulation [4].

2.2. Computational bottlenecks and profiling analysis

We have published an open-source C implementation of the above algorithm, namely *mmc*. Independently, Shen and Wang have published a C++ software, *timos* [11]. By comparison [13], *mmc* uses a Plücker-coordinates-based ray-tracing scheme for generality while *timos* uses a simplified Badouel algorithm [18] for better speed. In both cases, the simulations are accelerated using multi-threaded computing [12, 11] but neither of the published codes were optimized with SIMD.

Here, we perform a profiling analysis to identify the computational bottlenecks in the non-SSE *mmc* implementation. An open-source profiler, Callgrind, is picked for its availability and performance. The benchmark details are described in Section 3. Running the profiler, we extract the run-time for each function of the simulation code and generate a run-time-sensitive call-graph using a visualization tool, Kcachgrind. The resulting call-graph is shown in Fig. 1(a). Each block in Fig. 1(a) represents a C-function corresponding to each step of the simulation (`sincos` is a math function to produce a pair of sine and cosine simultaneously). The size of the block is proportional to the execution time spent within the function. We report the percentage of the run-time for each function by normalizing with the total run-time. Note that the result here is for a single CPU core as the profiler is restricted to run with a single thread.

From Fig. 1(a), we observe that the Plücker-coordinate-based ray-tracing subroutine (`plucker_raytet`) and the scattering calculations (`next_scatter`) take the majority of the simulation time, accounting for 66.74% and 26.92%, respectively. Within the ray-tracing function, the absorption and energy deposit calculations (`plucker_absorb`) account for 24.98% of the total run-time, while the rest of the calculations are related to the ray-tetrahedron intersection tests. Within the photon-scattering calculations, we notice that the math and random number generator (RNG) functions are the main consumers of the run-time, accounting for 11.01% and 8.91%, respectively. From Fig. 1(a), it is clear that the ray-tetrahedron intersection test is one of the biggest hotspots in the simulation. In addition, optimizing math and RNG functions is also helpful in enhancing the overall speed.

2.3. SSE-accelerated ray-tetrahedron intersection tests

We implemented two SSE-accelerated ray-triangle intersection methods in our software: an SSE ray-tracer based on the H&H method [17] and an SSE partial Baudoul's method with branch-less structure [18, 16]. Both the H&H ray-tracer and the Plücker-based ray-tracer in [12] require the calculation of the Barycentric coordinates. The partial Baudoul's methods, either

#173095 - $15.00 USD Received 23 Jul 2012; revised 25 Sep 2012; accepted 27 Sep 2012; pub. 12 Nov 2012

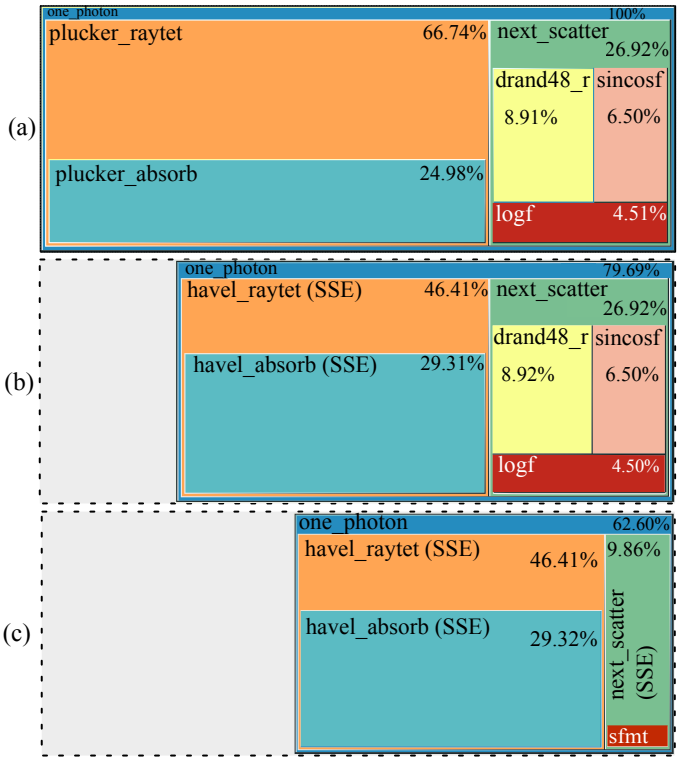

Fig. 1. Subroutine-level profiling for mesh-based MC simulation (a) without SSE, (b) with Havel & Herout's SSE-based algorithm, and (c) with additional SSE-enabled math library and RNG. The area and percentage in each block represent the run-time for each subroutine normalized by the total run-time of case (a). If a block is enclosed by another, its percentage contributes to that of the enclosing block. Profiling is performed within a single thread.

SSE or non-SSE, skip the Barycentric coordinates with presumably reduced accuracy [13].

For the branch-less Baudoul's method, we use a Structure-of-Arrays (SoA) format [10] to organize the face norm data in the memory. For example, instead of defining $\mathbf{N} = \{N_x, N_y, N_z, d\}$ as in [17], we define $\mathbf{N_x} = \{N_x^1, N_x^2, N_x^3, N_x^4\}$ where N_x^i $(i = 1, \cdots, 4)$ is the N_x component of the i-th face in a tetrahedron, and so on. With this approach, a single SSE command can advance one operation for four triangular surfaces simultaneously. An SSE implementation without the SoA memory layout was found to provide almost no acceleration.

2.4. Accelerating RNG and math functions with SSE

From Fig. 1(a), the RNG and math functions account for 72% of the scattering-related calculations. Using SSE to accelerate these functions is expected to improve the overall speed as well. In the previously released *mmc* code, the default RNG was a 48-bit multi-threaded RNG function, `drand48_r`, defined in the GNU C library (glibc). This RNG uses a linear-congruent algorithm to produce each random number. In this work, we replace `drand48_r` by a more efficient SIMD-oriented Fast Mersenne Twister (SFMT-19937) algorithm [19]. The SFMT-19937 algorithm has proven to have a vary long period ($2^{19937} - 1$), and is specifically optimized with SIMD parallelism.

To accelerate the calculation of the math functions (`log`, `sincos`), we leveraged a free math library, SSEMath by Julien Pommier [20]. In this library, each call to the math function

#173095 - $15.00 USD Received 23 Jul 2012; revised 25 Sep 2012; accepted 27 Sep 2012; pub. 12 Nov 2012

produces four values simultaneously with SSE commands. To utilize this fast math library in the photon scattering module, we created a static buffer with a length multiple of four and filled the buffer with random numbers generated by the SFMT RNG. We subsequently call the SSE version of the `log` and `sincos` functions to compute four function values simultaneously. Each time the MC simulator requests a random function, we pop a precomputed value from the buffer. When a buffer is used up, we refill the buffer by repeating the above process.

3. Results

In this section, we profile the mesh-based Monte Carlo code with different ray-tracing algorithms and RNG/math configurations. From the profiling output, we quantitatively compare the proposed algorithms to demonstrate the benefits of using SSE acceleration.

For this purpose, we build a benchmark problem using the widely-used Digimouse atlas [21]. From the segmented digital mouse CT image, we create a 3D tetrahedral mesh using an image-based mesh generator, iso2mesh [15], and utilities based on the CGAL library [14]. As seen in Fig. 2, the resulting mesh contains 42,301 nodes and 209,907 tetrahedral elements, including 21 distinct tissue regions according to the segmentation. The mesh data, script, and optical properties of each tissue type [22] are provided online on our website [12]. A pencil-beam point source is positioned on the skin above the right-hemisphere of the brain. All simulations are performed on a Dell workstation running 64bit Ubuntu Linux 10.04. The computer has a quad-core Intel E5520 CPU. The software was compiled with gcc 4.4.3 with `-O3` and `-msse4` options. We launched a total of 10^6 photons to produce the profiling results using Callgrind.

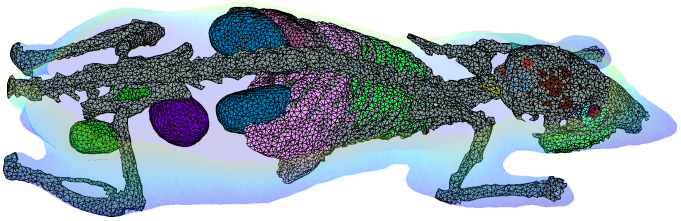

Fig. 2. A tetrahedral mesh generated from the Digimouse atlas. Different colors represent different tissue types.

In Fig. 1(b), we report the performance benefits of replacing the non-SSE Plücker-based ray-tracer with the H&H's SSE-based ray-tracer. The area of each box is proportional to its run-time. The width of the largest box in Fig. 1(b) is scaled by matching the area of the `next_scatter` block with that in Fig. 1(a), given that RNG and math functions are not changed in the two cases.

Next, we evaluate the improvement obtained from the SSE-accelerated math functions and the added SFMT RNG in the code. The updated performance results are shown in Fig. 1(c). Again, the size of each box is proportional to its run-time and the area for `havel_raytet` is set to match that in Fig. 1(b).

Table 1. Simulation run-times (in seconds) for various ray-tracers.

Methods	Plücker	H&H†	Badouel†	Branch-less Badouel†
Ray-tracing (s)‡	65.00	33.18	24.21	18.88
Overall (s)	155.65	121.48	110.18	81.92

† with SSE-enabled math and RNG functions;
‡ estimated from the profiling outputs.

#173095 - $15.00 USD Received 23 Jul 2012; revised 25 Sep 2012; accepted 27 Sep 2012; pub. 12 Nov 2012

In addition to the subroutine-level profiling analysis, in Table 1, we summarize total run-times for the ray-tracing calculation and the total simulation time across various combinations of ray-tracers and SSE-enabled techniques. In each case, we simulate 10^7 photons with four CPU cores running in parallel. Note that the two Badouel-based algorithms perform less computation than the other two algorithms at a cost of reduced accuracy.

4. Discussion

From Table 1, it is clear that the SSE-accelerated ray-tracing techniques make measurable improvements in the overall simulation speed. The total run-time is reduced by 22% for SSE-based H&H's method compared to the Plücker's method. For the simplified ray-tracing using the Badouel's method, the branch-less algorithm reduces the simulation time by 26% by improving pipelining on a modern CPU. The acceleration of the ray-tracing calculation alone is 2X when compared to H&H with Plücker-based methods; comparing the branch-less and branched Badouel's methods, the speedup is 1.3X. From the profiling results shown in Fig. 1(b) and (c), we see that the SSE-enabled RNG and math functions have accelerated the photon scattering calculations by 2.7X. Better computational efficiency was clearly achieved by simultaneously exploiting the multiple processor cores, the SIMD capability, and pipelining available in modern CPU processors. It is interesting to note that the speed-up ratio calculated by the profiling data in Fig. 1 is over-estimated compared to that derived from Table 1. We believe this is due to the fact that the profiler can only utilize a single CPU thread, thus, is a result of variable overhead when running single-threaded versus multi-threaded. Nonetheless, the relative run-times of the subroutines can provide clear insight and help guide us during code optimization.

The ability to fully utilize the parallelism provided in a modern CPU makes this code one of the efficient MC models available. In the recent years, given the considerable progress on effectively exploiting graphics processing units (GPU) for general purpose computing, new opportunities are available to further accelerate MC simulations. Although the SSE instructions used in this work can not be executed on a GPU directly, modern GPU hardware typically contains many wide SIMD units, allowing parallel execution of code fragments on a grander scale. This can potentially lead to dramatic reduction in simulation speed and ease of parallel programming. Recently, Powell and Leung reported a GPU implementation of the mesh-based MC for acousto-optic applications [23]. The authors compared their GPU code to the first release of *mmc* and demonstrated a 2X speed-up with a standard benchmark [12, 23]. Interestingly, with the techniques reported here and other improvements, the latest version of *mmc* has achieved 103 photons/ms for the same benchmark, a speed twice as fast as the reported GPU code. Based on our previous study [5], we strongly believe that a GPU has great potential to provide massive acceleration compared to a modern CPU. The rather low speed-up achieved in the forementioned GPU implementation may be related to suboptimal memory utilization. This GPU code, as well as other voxel- or surface-based MC codes, can potentially benefit from the efficient ray-tracing and memory layout schemes reported here.

In summary, we have investigated the potential of accelerating biophotonic Monte Carlo simulations on modern CPUs. We have used SSE constructs to accelerate ray-tracing calculations, as well as key math functions and random number generators. With these techniques, we were able to speed up the ray-tracing calculation by a factor of two, and improve the overall speed by 22 to 26%. The updated source code and benchmark dataset is provided on our website at *http://mcx.sourceforge.net/mmc/*. The software can not only take advantage of multi-core processors on a symmetric multiprocessor (SMP) system, but also has the potential to gain massive acceleration when running on a software distributed shared memory (SDSM) cluster, either locally or via cloud computing. The proposed techniques should also scale gracefully when many-core CPU chipsets become available on the market.

#173095 - $15.00 USD Received 23 Jul 2012; revised 25 Sep 2012; accepted 27 Sep 2012; pub. 12 Nov 2012

Acknowledgment

QF wishes to thank the funding support from the Bill & Melinda Gates Foundation (#OPP1035992).

#173095 - $15.00 USD Received 23 Jul 2012; revised 25 Sep 2012; accepted 27 Sep 2012; pub. 12 Nov 2012