FullMonte: a framework for high-performance Monte Carlo simulation of light through turbid media with complex geometry

Jeffrey Cassidy*[a], Lothar Lilge[b], and Vaughn Betz[a]

[a]Electrical and Computer Eng, University of Toronto, 10 King's College Rd, Toronto, ON, Canada

[b]Ontario Cancer Institute, 610 University Ave, Toronto, ON, Canada

ABSTRACT

Emerging clinical applications including bioluminescence imaging require fast and accurate modelling of light propagation through turbid media with complex geometries. Monte Carlo simulations are widely recognized as the standard for high-quality modelling of light propagation in turbid media, albeit with high computational requirements. We present FullMonte: a flexible, extensible software framework for Monte Carlo modelling of light transport from extended sources through general 3D turbid media including anisotropic scattering and refractive index changes. The problem geometry is expressed using a tetrahedral mesh, giving accurate surface normals and avoiding artifacts introduced by voxel approaches.

The software uses multithreading, Intel SSE vector instructions, and optimized data structures. It incorporates novel hardware-friendly performance optimizations that are also useful for software implementations. Results and performance are compared against existing implementations. We present a discussion of current state-of-the-art algorithms and accelerated implementations of the modelling problem. A new parameter permitting accuracy-performance tradeoffs is also shown which has significant implications including performance gains of over 25% for real applications.

The advantages and limitations of both CPU and GPU implementations are discussed, with observations important to future advances. We also point the way towards custom hardware implementations with potentially large gains in performance and energy efficiency.

Keywords: biophotonics, Monte Carlo, bioluminescence imaging, turbid media, acceleration, hardware, high-performance computing

1. INTRODUCTION

Many important applications such as bioluminescence imaging (BLI) rely on simulation of light propagation from emitters within turbid media. Further, many imaging and therapeutic biophotonic applications require iterative solutions to find or optimize the quantity of interest since no closed-form solutions exist. In BLI, the requirement is to determine which pattern of volume sources within the tissue gives rise to the observed pattern of light emission. Each iteration assumes a source distribution and performs a forward simulation consisting of hundreds of millions of packets, each having potentially hundreds of interactions, to achieve acceptable photon statistics throughout the region of interest. The time required to compute a solution is therefore a significant constraint in the analysis. A number of simulators have been demonstrated using a variety of assumptions, techniques, and technological platforms including CPU, GPU, and custom digital hardware. We present FullMonte, a framework to study the performance and accuracy of such simulations, and discuss the challenges relevant to hardware acceleration. Successful acceleration will ultimately lead to far greater use of techniques such as quantitative BLI.

This paper introduces a new software package, FullMonte, that performs Monte Carlo simulations of light propagation through an inhomogeneous volume of turbid media described by a tetrahedral mesh. It has performance comparable to the best available simulators with far increased flexibility in the information gathered.

* Corresponding author: jeffrey.cassidy@mail.utoronto.ca, Telephone: 1 416 206 5565

Incorporated in the software is a new optimization for calculating the photon packet direction after a scattering event which enhances performance on CPUs and has significant advantages for GPU or Field Programmable Gate Array (FPGA; custom digital logic) accelerator hardware. The choice of reduced precision for the mesh coordinates and propagation calculations is validated by comparison against an existing higher-precision simulator. There is also an exploration of the effect of a simulation parameter thus far ignored which offers significant control of the accuracy-performance trade-off. Using profiling information generated from the software, we also discuss features of the tetrahedral model that are challenging for hardware acceleration and propose solutions.

The discussion is organized as follows. First, a review of prior work is presented covering both previous modelling methods and techniques in both hardware and software used to accelerate them. Second, the important features of the FullMonte model will be described and compared with existing implementations. Third, a validation of FullMonte against an existing tetrahedral-based simulator (TIM-OS) is presented. Fourth, factors affecting speed and accuracy that apply both to FullMonte and prior work will be discussed. Finally, we outline future work to be undertaken using the framework presented, including design of an accelerator system based on FPGA custom digital logic.

2. PREVIOUS WORK

2.1 Software Models

One of the first and still most popular Monte Carlo (MC) light scattering codes for turbid media is MCML,[1] which solved the problem for a pencil beam incident on infinite planar layers with different material properties. Due to the radial symmetry of the problem, the absorption grid is reduced from 3D to 2D, and the geometry model to 1D. The memory requirements and result variance are correspondingly reduced at the cost of generality. More complicated source patterns are computed through convolution of pencil beams, however the underlying tissue geometry is fundamentally restricted to planar slabs with normally-incident light.

tMCimg[2] uses a 3D voxelized representation of the scattering volume and its material properties. The model is limited by the voxelized representation, which can create artifacts near refractive-index boundaries due to the quantization introduced. Sloped or smooth boundaries must be represented in a blocky "stair-step" fashion, leading to discretization error in computing surface normals and hence incorrect reflection and refraction calculations. Binzoni et al[3] provide a detailed discussion showing the negative impact of these approximations.

NIRFAST[4] takes a different approach, using the diffusion approximation to formulate the problem using the Finite Element Method (FEM). FEM uses a tetrahedral mesh and achieves very fast run times at the cost of restrictions imposed on the materials present. Significantly, it can not deal accurately with low-scattering materials, refractive index boundaries, or anisotropic materials. For some application domains these restrictions are minor; however, there exist relevant problems where the cost of MC is necessary to get an accurate solution.

MMCM[5] and TIM-OS[6] both use a tetrahedral mesh similar to that used for finite-element modelling. The two methods differ in the coordinate system used for intersection testing and the types of geometric primitives used. TIM-OS uses strictly tetrahedral elements, while MMCM can support multiple geometry types including tetrahedra. Both suites offer advantages over tMCimg because they offer a close approximation to smooth curved surfaces, and both can easily calculate accurate normals at material interfaces. Surprisingly, the authors of TIM-OS estimate (based on MCML's single-core performance) that TIM-OS is similar in speed to MCML when modelling geometries with radial symmetry, despite its more general geometry representation.

2.2 Acceleration Methods

Fang and Boas present MCX,[7] a GPU-based 3D simulation using a voxelized representation for both material properties and absorption locations. The model is conceptually similar to tMCimg but implemented on a different platform, achieving acceleration of 75-300x (depending on options) over tMCimg running on a single-core CPU.

Alerstam et al[8] demonstrate a very large (1000x) speedup for the restricted case of a semi-infinite homogeneous non-absorbing slab using the MCML algorithm on a GPU. Since the material slab is homogeneous and non-absorbing, no material properties or geometric information need to be fetched from memory which nearly eliminates the effect of memory performance on the simulation. Consequently, this acceleration result could

be regarded as a best possible case for acceleration using GPUs: to access a more general problem set or to score absorption within the tissue, the program must necessarily access significantly more memory which would limit performance. Their work on CUDAMCML[9] takes a similar approach to accelerating "classic" (multi-slab) MCML, and can achieve large acceleration ($\approx 100\times$) over CPU implementations.

Lo[1011] took a different approach, implementing MCML on custom digital logic circuits known as Field-Programmable Gate Arrays (FPGA), achieving speedup ranging from 65x to 80x and energy efficiency gains estimated at approximately 45x over a then-current Intel Xeon multicore processor. The implementation was subject to all of the restrictions of the MCML model, plus additionally a maximum absorption grid size of 256x256 and a maximum of five material layers due to hardware memory limitations. These results are impressive in terms of both run-time acceleration and power efficiency, which is an important consideration when scaling up to large volumes of computation. However, the model is subject to very tight restrictions that make it unsuitable for general problems including BLI reconstruction.

The planned FullMonte hardware implementation will build on the software model and its hardware-friendly features presented here, as well as the profiling information gathered. A more recent FPGA platform similar to the one used by Lo will be used. Custom logic design will allow many of the limitations of CPU- and GPU-based computation to be exceeded while implementing a fully 3D mesh-based solution with accuracy equal to the best tetrahedral MC simulators.

3. MODEL DESCRIPTION

FullMonte has some similarity to the existing TIM-OS model. The fundamental algorithm for both derives from MCML's "hop, drop, spin",[1] extending it to a tetrahedral mesh to describe the geometry instead of infinite planar layers. A tetrahedral representation offers the critical advantage that mesh-element surface normals are easily represented, and that the mesh is free to conform to interfaces with curves or complex shapes unlike tMCimg/MCX or MCML derivatives.

In addition to simply performing light propagation simulations, FullMonte was designed to study the factors influencing performance and the considerations required to port it to computational accelerators. One important difference is in the software design, which makes extensive use of C++ templates to facilitate various ways of logging the simulation results. One can, at compile time, select or de-select logging of memory accesses, volume absorption events, surface exit events, or other combinations. Absorption can be logged on a voxel basis, within a region of interest, on a per-tetrahedron basis, or in a user-defined way chosen at compile time. There is no run-time penalty for features not selected and the programmer has great latitude to define their own logging schemes. As will be discussed later, this permits tailoring the software to specific needs and gaining insight into the features of the model most important to its run-time performance.

There are also some programming changes made to enhance performance. For the propagation calculations themselves we also take a different approach, explicitly using Intel SIMD* Streaming Extensions intrinsics and the freely-available GCC compiler instead of relying on the Intel compiler to auto-vectorize the code. It also uses lower-precision coordinates for the geometry calculations and a different method of calculating the scattering, both of which have advantages for hardware acceleration.

3.1 Precision

FullMonte uses a mix of floating-point types to maximize performance without compromising accuracy. Fluence is accumulated using the 64-bit IEEE double type, to prevent rounding error in the case where a very large number of photon packets are summed. Spatial coordinates (positions, directions) use the 32-bit IEEE float type since the number of steps over which error can accumulate is limited by the number of interactions before the packet exits or dies (on the order of hundreds). This reduces the space requirements for storing the mesh, and reduces the memory bandwidth needed to load the geometry during the simulation. While apparently not a limiting factor when implemented on a CPU as shown by FullMonte's performance parity with TIM-OS (which

*Single Instruction Multiple Data, in which multiple data elements are computed using the same operation (instruction) in parallel

uses double precision), we expect it to prove useful for GPUs, which have higher throughput for single-precision calculation than double, or for custom hardware where additional precision costs silicon area.

A full validation against TIM-OS on general geometries is shown below, illustrating that the additional precision in spatial coordinates is not necessary. Alerstam shows a similar result for the more restricted case of semi-infinite non-absorbing media by validating a single-precision GPU against a double-precision CPU implementation.

3.2 Spin calculation

The "spin" step introduced in the MCML algorithm is also different in FullMonte. To compute a scattering event using the Henyey-Greenstein phase function, the algorithm needs both the current direction vector and a pair of unit vectors orthogonal to the direction of travel and each other. Other implementations (TIM-OS, MCML, and their derivatives) calculate those orthogonal vectors when needed by taking the cross product of the ray direction with an arbitrarily chosen basis vector (typically in the dz direction). After the spin step, those vectors are discarded and will be computed again for the next spin.

FullMonte, however, maintains at all times three vectors: d, giving the direction of travel, a the first orthogonal vector, and b the second orthogonal vector. Since they are maintained, it is not necessary to find the two orthogonal vectors; however, they must be rotated along with the direction vector to remain useful in the next step. The advantage is that this technique avoids division, square-root, and branching as discussed in greater detail in Appendix A.

Since it is necessary in both cases to generate the sine and cosine of the azimuthal scattering angle, a micro-benchmark test was performed focusing solely on the speed of calculation of the new direction given the sines and cosines. On an Intel Core i5 CPU, a speedup of nearly 2x was found for the new method. We believe that the performance difference is accounted for by the lack of costly operations (divisions, branches, square root) and by the efficient structure of 3x3 matrix multiplication for SSE-enabled processors. While the basic operation count increases (from 8 to 15 additions and 16 to 28 multiplications), those operations are hidden by efficient vectorization since up to four pairs of floating-point numbers may be operated on at once for the same cost as one.

In addition to its positive impact on the CPU runtime, these modifications are expected to be highly valuable for both GPU and FPGA implementations since those platforms have a higher penalty for branching and special operations than a CPU.

3.3 Software tools and libraries

In contrast to TIM-OS, FullMonte uses an entirely free and open-source set of tools and libraries. The SIMD-Oriented Fast Mersenne Twister by Saito and Matsumoto[12] is used for random number generation, and the GNU C Compiler (gcc) is used for compilation. Meshes and results may optionally be stored in a PostgreSQL database. High performance is achieved through use of multi-threading using Pthreads to exploit all of the CPU cores, as well as Intel SSE compiler intrinsics to maximize floating-point computation speed.

4. VALIDATION

Validation was performed against TIM-OS using the test cases presented in the original TIM-OS paper. In all cases, tight agreement was found to within statistical uncertainty for runs of one billion packets. An example graph is shown in Fig 1 for the "mouse" test case, comparing the simulation output element by element. The horizontal axis represents the total energy for each element found by the TIM-OS model, while the vertical axis is the percentage difference between FullMonte and TIM-OS. The left-hand plot shows the energy exiting through each surface mesh element, while the right-hand shows the total absorbed energy per volume element. As expected, variability increases for mesh elements with less fluence since they receive fewer packets to average.

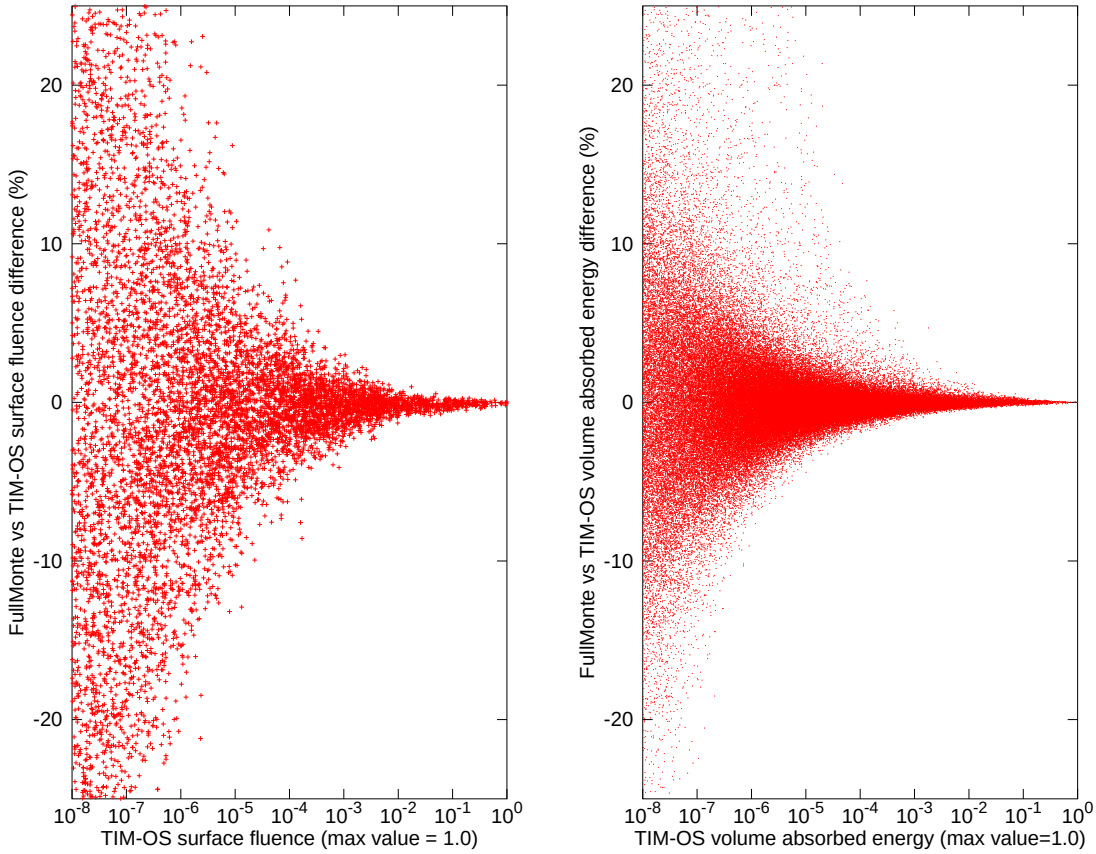

Figure 1. Comparison of TIM-OS results vs FullMonte: left, surface emission; right, absorbed energy per tetrahedral volume element

5. PERFORMANCE ANALYSIS

Run-time results comparable to TIM-OS were attained without relying on proprietary compilers or libraries. Results were produced on an Intel *Sandy Bridge* i7-2600K quad-core 3.4 GHz CPU with 8MB L3 cache and hyperthreading (allowing 8 virtual cores). TIM-OS was run using the recommended commands on the website, using 8 threads to fully utilize the four hyper-threaded cores. FullMonte was compiled with all profiling information turned off to maximize speed, and also used 8 cores.

The test data used is described in greater detail in the original TIM-OS paper,[6] and is freely available on the TIM-OS website[†]. Of the test data cases, the one most representative of BLI applications is named "mouse" and is based on the publicly-available Digimouse[13] mouse model. In the virtual mouse, there is a set of tetrahedra defined as a source representing a bioluminescent tumour to be imaged.

Other cases not representative of BLI imaging are included for validation purposes only. The lower performance on these cases ("onelayer" and "FourLayer") is due to a focus on tuning the software for complex cases applicable to BLI. There remains some inefficiency in the source-emission code which is more important to those cases having few photon interactions prior to exiting the geometry.

Table 1. Performance comparison vs TIM-OS

Case	TIM-OS (s)	FullMonte (s)	% difference
Digimouse	820.6	835.5	+1.8%
cube_5med	1575.8	1618.4	+2.7%
FourLayer	606.3	663.5	+9.4%
onelayer $\mu_s = 5$ $\mu_a = 0.05$	44.6	50.1	+12.5%
onelayer $\mu_s = 10$ $\mu_a = 0.05$	91.8	101.5	+10.6%
onelayer $\mu_s = 5$ $\mu_a = 0.20$	46.1	51.5	+11.8%
onelayer $\mu_s = 10$ $\mu_a = 0.20$	93.0	102.9	+10.7%

6. ACCURACY-PERFORMANCE TRADEOFFS

Both TIM-OS and FullMonte contain a parameter inherited from MCML referred to here as w_{min}, which is necessary for both computational efficiency and conservation of energy. When a photon packet becomes sufficiently weak (has weight $w < w_{min}$), it is given a 1-in-P chance of continuing with strength $w' = Pw$, otherwise it terminates. TIM-OS and MCML both use a value of $w_{min} = 10^{-5}$ and $P = 10$, both of which appear to be arbitrary choices receiving little discussion. Intuitively, one would expect the results to become increasingly accurate as w_{min} decreases since quantization error will decrease. On the other hand, the increased accuracy will require additional simulation time since a packet can endure more absorption events before it is retired. Each of those absorption events, however, is expected to contribute little since by definition it is more than five orders of magnitude smaller than a freshly-launched packet.

Expending computational resources to compute a result to an unmeasurable accuracy is a waste. Considering a typical resolution for a 16-bit low-light camera measuring the surface emittance (as is often the case in BLI), the maximum possible dynamic range is $2^{16} = 6.5 \times 10^5$ between the least significant bit and the maximum value. In order to gain performance, it was hypothesized that altering the value of w_{min} may lead to a performance

[†]https://sites.google.com/a/imaging.sbes.vt.edu/tim-os/

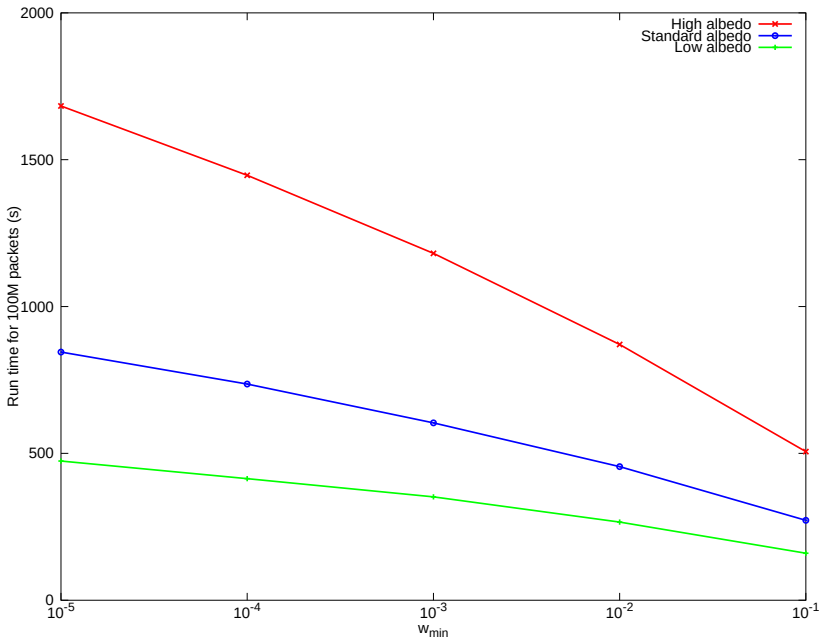

Figure 2. Impact of altering w_{min} on run time for three albedo scenarios

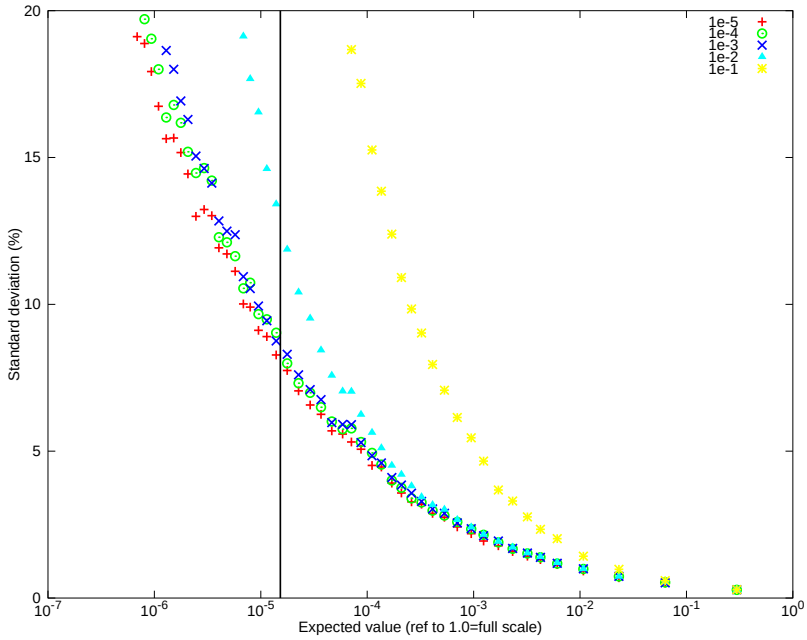

Figure 3. Result (surface fluence) variance as a function of w_{min} (10^{-5} to 10^{-1}) for "mouse" test case; vertical line indicates 16-bit detection limit

increase without measurable effect on accuracy. However, the amount of gain depends on the exact geometry, source location, and material properties used. If most packets exit the volume before losing roulette at a given w_{min}, only those remaining within the volume will require an increased number of steps if w_{min} is decreased.

To check the impact of material albedo ($\alpha = \frac{\mu_s}{\mu_s+\mu_a}$), three different sets of material properties were tested. Starting with the standard properties from the TIM-OS "mouse" test case, two other sets were generated with one-half and double the scattering coefficient to give lower and higher albedo cases respectively. For each value of w_{min} ranging in powers of ten from 10^{-5} to 10^{-1}, sixteen simulations of 10^8 packets were run with different random seeds. As expected from the earlier termination of photon packets, there was a significant reduction in required run time as illustrated in Fig 2. When increasing the parameter from 10^{-5} to 10^{-3}, a 28% decrease in required time was found across the three cases, with a larger 68% reduction at 10^{-1}.

To measure the accuracy cost of this tradeoff, the sixteen runs with the default $w_{min} = 10^{-5}$ were averaged to provide a "reference" value equivalent to a single low-variance run of 1.6×10^9 packets. Then, for each element in each simulation set with a different value of w_{min}, the standard deviation was calculated around the reference value. Measured in this way, the standard deviation includes both any shift in the mean versus the reference case (which should be zero), and the statistical variance between runs. Fig 3 shows the results for surface emittance in the standard albedo case, with the vertical axis showing the standard deviation as a percentage of the element's mean value. The horizontal axis shows the value of the element in the reference set normalized to the maximum value. A vertical line marks the detection limit (a single bit) for a 16-bit sensor if saturation is to be avoided. For ease of viewing, each point represents a bin of 100 elements. In this case, the change in variance is virtually zero within the detection limit of a 16-bit camera, up to $w_{min} = 10^{-3}$. As a result, a 28% decrease in run time can be had without making a measurable difference in the results.

The unique contribution of the w_{min} parameter is to control the distribution of error throughout the dynamic range of the image. Previously only the number of packets could be altered, allowing the entire curve to be shifted vertically through averaging. By altering w_{min}, the user can select to trade a faster simulation against one that is more accurate in the lower-fluence elements. Choosing carefully by considering the fluence range that is relevant to the user's application may permit significant speed gains.

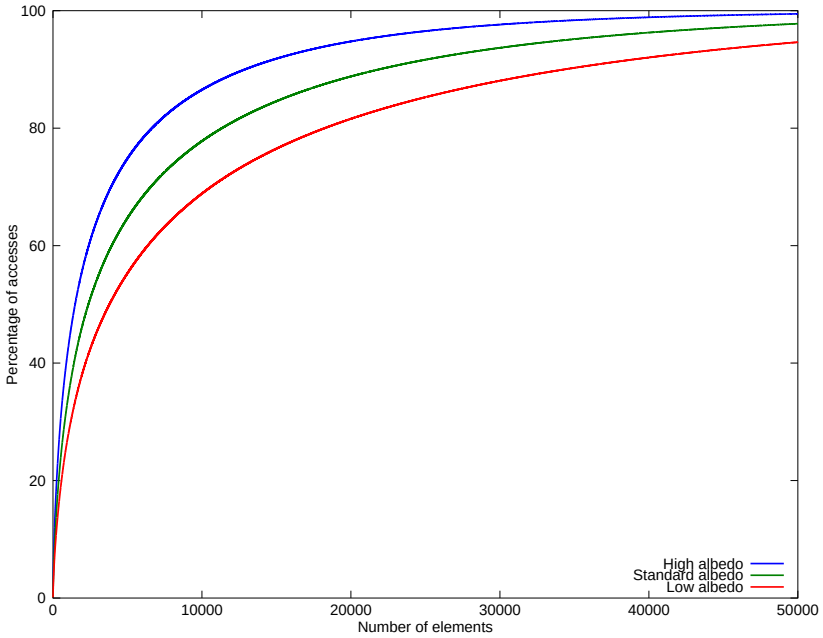

Figure 4. Memory locality: percentage of memory accesses occurring within the N most common addresses

7. MEMORY ACCESS PROFILING

One of the most significant challenges in moving from the MCML model with 1D tissue properties and 2D absorption to a general 3D model is the amount of data required to both store the geometry and record absorption/exit events. Indeed, MCML already performs most of the 3D ray propagation calculation before converting to cylindrical coordinates for storing absorption and reflection. The most significant simplification made in MCML is that the normal vector is always $+dz$, so there are several geometric operations where two-thirds of the components of one vector are known to be zero. When moving to fully-3D calculation that is no longer the case so there is some increase in operation count for determining ray-interface intersection and normals for reflection/refraction.

More significantly, though, the size of the geometric representation goes from a small (often single-digit) N layers, each described by only a thickness to hundreds of thousands of tetrahedra of four 3D Cartesian points each plus links to the four adjacent elements. Alternatively, as in the case of MCX, the representation may be hundreds of thousands of voxels or more. In either case storage and fast access to the geometric description is no longer trivial. Photon packets move frequently between mesh elements, making fast access to the geometry description performance-critical so the processor is not waiting for memory to proceed.

Further investigation revealed that memory access varies greatly between mesh elements as shown in Fig 4. The graph illustrates the percentage of all accesses occurring to the N most commonly accessed volume elements for the "mouse" test case. Despite a large mesh (over 300k elements), the bulk of memory access is confined to under 10% of those elements. Element reuse tends to happen immediately as multiple hits in a row, after a few hits to another element, or after "a long time" on the order of thousands of hits to other elements. Proper use of these properties is important to achieving good memory throughput, and hence addressing one of the crucial limitations to implementation performance.

7.1 Tetrahedral element ordering

In the original TIM-OS paper,[6] Shen and Wang correctly note that memory locality could be an important consideration for performance. They attempted to gain performance by using the reverse Cuthill-McKee algorithm

to reorder elements such that adjacent tetrahedra are more likely to be in adjacent memory addresses. While an interesting idea, it did not produce the hoped-for results which we believe was due to the details of the CPU cache in use. The Intel *Sandy Bridge* architecture uses 64-byte cache lines like most current processors. Since a geometry element is larger than that (100 bytes in FullMonte, 160 in TIM-OS) and the cache is loaded one cache line at a time, loading element i will not cause even element $i+1$ to be present in the cache. The sequence of elements in memory is therefore irrelevant. Contrary to the authors' speculation, we believe that regardless of the number of cores reordering will not result in a performance gain.

When testing FullMonte with the Cachegrind cache profiling tool, the last-level cache miss rate was found to be nearly zero (<0.5%) indicating that virtually all necessary data was resident in the L3[‡] processor cache which offers high-throughput low-latency access to memory. Further performance improvement through cache locality enhancements is therefore unlikely. Given that both programs also make extensive use of all available tools and processor instructions to speed up computation and use efficient algorithms, it is likely that TIM-OS and FullMonte jointly represent very nearly the best possible performance for this algorithm on current CPUs.

On the positive side, application performance should continue to increase so long as processor manufacturers continue to make processors with more cores and large, fast, shared L3 caches. Additional vector (SIMD) instructions and vector execution units may also make incremental gains in application performance over time.

7.2 Using GPU local memory

Fang and Boas[7] in their discussion of MCX (tMCimg on GPU) also try to exploit memory locality to gain performance by keeping voxels near the photon source in a faster local memory, arguing that voxels closer to the source are more likely to be accessed. While an appealing idea (and supported by our findings), it also led to an actual performance decrease due to GPU hardware architecture limitations. Specifically, GPUs have a fixed amount of memory (≈64kB) per cluster of multi-processors that can be allocated to shared storage or registers. The problem is that the greater amount of memory marked as shared, the less space is allowed for threads' registers and hence a smaller number of threads are able to run within the space allocated. Since GPUs rely on having a large number of threads in-flight at the same time to keep the processing units busy, the resulting limitation on the number of threads running decreased performance.

7.3 Challenges for accelerating mesh-based methods

The foregoing discussion highlights an interesting difference between GPU and CPU technology that is very relevant to the application at hand. CPUs rely on a small number of cores with a large, fast, shared cache accessing a large, slow RAM array in fine-grained increments (currently typically 64 bytes). The access granularity and cache size are ideal for a tetrahedral mesh application, but the overall speed is limited by the number of cores[§] and the floating-point execution speed. GPUs on the other hand have a large number of cores with small, non-shared caches accessing a smaller, fast RAM array in large blocks. While GPUs have excellent floating-point performance, their memory system is designed around moving large streams of data. The challenge with a GPU is to keep the floating-point cores fed, since the mesh element data is accessed far more randomly and in smaller-grained increments than the GPU is designed for. Since it is designed for throughput, the latency penalty for main memory access is much higher on a GPU so caching is yet more critical.

The CPU cache strategy seems to fit this application better because the CPU cache is large enough to hold all of the program's working set and permit access to it on a granularity scale similar to the mesh description. Although Alerstam has demonstrated huge acceleration of a semi-infinite non-absorbing slab geometry, that problem excludes any need to access either a mesh geometry or to read and store absorption data. Shen and Wang note in their paper that TIM-OS on a multi-core processor has a speed on the same order of magnitude as CUDAMCML on a GPU. One could reasonably assume that a simulator using a more complex model requiring more memory access and more calculation would be slower than the simpler CUDAMCML model. Unless CUDAMCML contains significant inefficiencies which can be remedied, this observation is not encouraging for

[‡]The level-3 cache is the largest, and the last searched before accessing main memory. On the computer used for testing it was 8MB shared between all cores.

[§]A limited number of cores is one of the costs of having a large shared cache; caching consumes a large portion of the die area.

work on GPU-based tetrahedral simulators. To get a useful acceleration ratio on GPU for a mesh-based system, it will be necessary to overcome the inherent barriers in GPU architecture to exploiting the particular locality of this problem.

We believe that custom hardware designed to exploit both short-term reuse using a small (a few elements) least-recently-used cache, and a cache designed to hold the most frequently-accessed (*not* most-recent) elements would be the best option. Such a strategy could be implemented dynamically, or one could run a few iterations to determine the access patterns and then fix the most-frequent cache set statically. The important point to note is that this strategy is not available on either CPU or GPU.

8. CONCLUSIONS

8.1 Numerical precision

It was demonstrated that the IEEE single representation offers sufficient precision for representing spatial coordinates such as packet position and direction in the program. This has the benefit of decreasing storage size by a factor of two, which more significantly reduces the demands on memory bandwidth and supports the use of single-precision calculation. The reduced precision leads to increased performance potential for both GPU and FPGA implementation, both due to faster computation and due to decreased memory requirements for the mesh.

8.2 New method for spin calculation

A new method was proposed for calculating the direction after a scattering event avoiding branches and square-root calculations that should offer significant benefits particularly for hardware acceleration. Microbenchmarks also demonstrate that it is significantly faster on a conventional CPU, though we have not isolated the impact within the context of the full algorithm. Being one of the most common operations (often hundreds per photon packet), it should have a significant impact on performance.

8.3 Use of appropriate min-w value

The intended application for the simulation should guide the choice of the roulette parameter w_{min} to optimize performance. For bioluminesence applications using a 16-bit detector (a dynamic range of 6.5×10^5), values of 10^{-3} to 10^{-4} differ from the reference results by an amount below the detection threshold, resulting in a performance boost averaging 25% compared to using the default 10^{-5}.

8.4 Importance of memory locality

We have demonstrated that for a large bioluminescence mesh, memory access effects are critical to the ability to accelerate the algorithm. CPU implementations are limited by the number of cores and their floating-point computing ability. GPU implementations of tetrahedral methods, of which none currently exist, would be significantly challenged by the fine-grained random memory access required. This is one area where a custom digital logic implementation using FPGA has room to excel, since it provides the designer the possibility of explicitly designing and managing cache hierarchy and strategies. While caching of fixed locations is an unusual paradigm that is difficult if not impossible to accomplish on most platforms (CPU/GPU), it would be quite feasible on an FPGA and allow exploitation of the locality shown in Fig 4. Custom logic would also be able to benefit from the distinction between read-only and read-write data.

8.5 Source code online

Readers are invited to download the source code from `www.eecg.utoronto.ca\~cassidy\fullmonte`

9. FUTURE WORK

The software model and embedded profiling tools described above will be used to design an optimized custom digital hardware implementation. This work will expand on the results shown previously by Lo et al,[10] removing the restrictions inherent in the MCML model: allowing a fully 3D tetrahedral geometry, inhomogeneous refractive index, and anisotropic scattering. Based on the findings and discussion above regarding the factors limiting performance, we will focus our efforts on the best possible memory subsystem using the flexibility offered by the FPGA not available on CPU or GPU platforms.

There also remain unexplored avenues for performance enhancement of the CPU implementation. Taking a cue from the original TIM-OS paper and the recent enhancements to MMCM,[14] one could use a SIMD logarithm function to increase speed when drawing step lengths. The current FullMonte implementation uses the scalar single-precision implementation in the GNU C library. There is also room for improvement in the code that emits photons from the source.

Lastly, a comparison of the Monte Carlo method shown here to FEM-based approaches for bioluminescence imaging such as NIRFAST is required. One would expect increased accuracy from FullMonte, particularly in cases where the diffusion approximation is not appropriate such as low-albedo, weakly-scattering, or highly inhomogeneous geometries. The precise difference should be quantified to assess whether the additional accuracy is worth the run time cost.

APPENDIX A. DETAILED DISCUSSION OF SPIN ALGORITHM CHANGE

Let the original direction of travel be unit column vector $d = d_x\hat{i} + d_y\hat{j} + d_z\hat{k}$. Let θ be the polar deflection angle from the incoming direction, and ϕ be the azimuthal angle.

A.1 CUDAMC / CUDAMCML / TIMOS

In CUDAMC and CUDAMCML, the scattered direction (when $d_z \neq 0$) is

$$d'_x = d_x \cos\theta + \frac{\sin\theta\,(d_x d_z \cos\phi - d_y \sin\phi)}{\sqrt{1-d_z^2}} \tag{1}$$

$$d'_y = d_y \cos\theta + \frac{\sin\theta\,(d_y d_z \cos\phi - d_x \sin\phi)}{\sqrt{1-d_z^2}} \tag{2}$$

$$d'_z = d_z \cos\theta - \sin\theta\cos\phi\sqrt{1-d_z^2} \tag{3}$$

which Alerstam et al note must be renormalized to keep it as a unit vector.

It is important that for directions $\hat{d}$ near $\hat{k}$ that the denominator of the second term in the above equations is very small, which may contribute to the numerical difficulties that necessitate renormalization in CUDAMC and CUDAMCML as noted in the source code comments. While the code avoids an exact divide-by-zero condition by handling that $d_z = 1$ as a special case, it permits values of $\sqrt{1-d_z^2}$ very near zero. TIM-OS performs a similar calculation, except that it uses double-precision arithmetic and checks for $|d_z| \leq 1-\epsilon$ to limit how small the denominator may become. TIM-OS does not renormalize the vector, which appears to be acceptable due to the higher precision of the double format.

Given $\hat{d}$, $\sin\theta$, $\cos\theta$, $\sin\phi$, and $\cos\phi$, the number of required operations is:

Table 2. Operation counts for CUDAMC / CUDAMCML spin technique

Stage	Add/Sub	Multiply	Divide	Sqrt	Branch
Calculating $\hat{d}$	6	13	1	1	1
Normalization	2	3	3	1	0
Total	8	16	4	2	1

A.2 FullMonte method

Instead of regenerating the normal vectors as needed, FullMonte maintains them throughout the loop. When a packet is launched, it is provided with a direction vector $\hat{d}$ and unit vectors $\hat{a}, \hat{b}$ that are orthogonal to each other and to the direction of travel. These three vectors create an orthonormal basis in $\mathbb{R}^3$, where the direction of travel is always $\begin{bmatrix}1 & 0 & 0\end{bmatrix}^T$ which can then be rotated using a spin matrix $S_{\theta,\phi}$. So described, a scattering operation is simply a computation of $\begin{bmatrix}\hat{d}' & \hat{a}' & \hat{b}'\end{bmatrix}^T = S_{\theta,\phi}\begin{bmatrix}\hat{d} & \hat{a} & \hat{b}\end{bmatrix}^T$. The scattered direction can be read from the first row of the result matrix. The second and third rows are retained for use in subsequent scattering operations.

To create the scattering matrix, first rotate the two orthogonal components (second and third element) by ϕ around the direction $\hat{d}$ (first element), then rotate $\hat{d}$ towards the new $\hat{a}$ by angle θ as follows:

$$S_{\theta,\phi} = \begin{bmatrix}\cos\theta & -\sin\theta & 0\\ \sin\theta & \cos\theta & 0\\ 0 & 0 & 1\end{bmatrix}\begin{bmatrix}1 & 0 & 0\\ 0 & \cos\phi & -\sin\phi\\ 0 & \sin\phi & \cos\phi\end{bmatrix} = \begin{bmatrix}\cos\theta & -\sin\theta\cos\phi & \sin\theta\sin\phi\\ \sin\theta & \cos\theta\cos\phi & -\cos\theta\sin\phi\\ 0 & \sin\theta & \cos\phi\end{bmatrix} \tag{4}$$

While there exist many possible choices of unit vectors orthogonal to the direction of travel, existing implementations (all of CUDAMC, CUDAMCML, TIM-OS) have opted to calculate them by normalizing the cross product of $\hat{d}$ with an arbitrarily-chosen fixed vector. To maximize the number of zeros in the multiplicative

terms, a vector with only one nonzero element is the simplest choice, with $\hat{k}$ (the dz direction) being chosen arbitrarily.

$$\hat{b} = -\frac{\hat{d} \times \hat{k}}{|\hat{d} \times \hat{k}|} = -\frac{1}{\sqrt{d_x^2 + d_y^2}} \left(d_y \hat{i} - d_x \hat{j} \right) \tag{5}$$

Since $\hat{d}$ is unit, we know that $d_x^2 + d_y^2 = 1 - d_z^2$. From $\hat{b} \perp \hat{d}$ and $|\hat{b}| = |\hat{d}| = 1$, the final vector can be found by

$$\hat{a} = \hat{d} \times \hat{b} = \frac{1}{\sqrt{d_x^2 + d_y^2}} \left(d_x d_z \hat{i} + d_y d_z \hat{j} + \left(d_x^2 + d_y^2 \right) \hat{k} \right) \tag{6}$$

Given these choices and the definition of $S_{\theta,\phi}$, the correspondence with 1-3 becomes clear.

The resulting scattering operation is numerically stable over thousands of repetitions even at 32-bit float precision (ie. after many iterations the three vectors remain orthonormal), and does not require expensive division or square-root operations. The cost to achieve this benefit is additional storage of a pair of 3-vectors for the photon packet. In our C++ reference model, these values appear to be assigned to registers by the compiler making the performance and storage overhead negligible. Other platforms such as GPUs may find the cost-benefit tradeoff different, however the lack of complex operations or branches is conceptually appealing.

Table 3. Operation counts for FullMonte spin technique

Stage	Add/Sub	Multiply	Divide	Sqrt	Branch
Calculate $\hat{d}'$	6	11	0	0	0
Calculate $\hat{a}', \hat{b}'$	9	17	0	0	0
Total	15	28	0	0	0

This formulation increases the amount of computing required in the case of reflection or refraction since $\hat{a}$ and $\hat{b}$ must be regenerated using Eq 5,6. However, the cost is acceptable because those events are one or two orders of magnitude rarer than scattering in the cases studied. In working with hardware acceleration platforms like GPU and particularly FPGA, avoidance of branch conditions and complex operations (sqrt, log, trig functions) are highly desirable. In the case of GPUs, there tend to be a smaller number of "special function units" capable of these operations which may run at slower speeds than the simple functions. In an FPGA, special function computation imposes heavy area and latency penalties since addition, subtraction, and multiplication are provided as fast and power-efficient built-in functions while others (square-root, divide, trig) must be solved iteratively.

ACKNOWLEDGMENTS

Lothar Lilge acknowledges support through the Canadian Institutes of Health Research and the Ontario Ministry of Health and Long-Term Care. Vaughn Betz acknowledges support received from the NSERC-Altera Industrial Research Chair in Programmable Silicon. Jeffrey Cassidy acknowledges support from the University of Toronto SGS Travel Grant Program to present at the conference. The authors also thank Robert Weersink of the Ontario Cancer Institute for valuable discussions on bioluminescence imaging.

REFERENCES

[1] Wang, L., Jacques, S. L., and Zheng, L., "MCML - Monte Carlo modeling of light transport in multi-layered tissues," *Computer Methods and Programs in Biomedicine* **47** (1995).

[2] Boas, D. A., Culver, J. P., Stott, J. J., and Dunn, A. K., "Three dimensional Monte Carlo code for photon migration through complex heterogeneous media including the adult human head," *Optics Express* **10** (2 2002).

[3] Binzoni, T., Leung, T. S., Giust, R., Rüfenacht, D., and Gandjbakhche, A. H., "Light transport in tissue by 3D Monte Carlo: influence of boundary voxelization," *Comput Methods Programs Biomed* **89**, 14–23 (Jan 2008).

[4] Dehghani, H., Eames, M. E., Yalavarthy, P. K., Davis, S. C., Srinivasan, S., Carpenter, C. M., Pogue, B. W., and Paulsen, K. D., "Near infrared optical tomography using nirfast: Algorithm for numerical model and image reconstruction," *Communications in Numerical Methods in Engineering* (2009).

[5] Fang, Q., "Mesh-based Monte Carlo method using fast ray-tracing in Plücker coordinates," *Biomedical Optics Express* **1**(1) (2010).

[6] Shen, H. and Wang, G., "A tetrahedron-based inhomogeneous Monte Carlo optical simulator," *Physics in Medicine and Biology* **55** (2010).

[7] Fang, Q. and Boas, D. A., "Monte carlo simulation of photon migration in 3d turbid media accelerated by graphics processing units," *Optics Express* **17**(22) (2009).

[8] Alerstam, E., Svensson, T., and Andersson-Engels, S., "Parallel computing with graphics processing units for high-speed Monte Carlo simulation of photon migration," *J Biomed Opt* **13**(6), 060504 (2008).

[9] Alerstam et al, "http://www.atomic.physics.lu.se/biophotonics/our%20research/monte%20carlo%20 simulations/gpu%20monte%20carlo/."

[10] Lo, W. C. Y., Redmond, K., Luu, J., Chow, P., Rose, J., and Lilge, L., "Hardware acceleration of a monte carlo simulation for photodynamic therapy treatment planning," *Journal of Biomedical Optics* **14** (Jan/Feb 2009).

[11] Lo, W. C. Y., *Hardware Acceleration of a Monte Carlo Simulation for Photodynamic Therapy Treatment Planning*, Master's thesis, University of Toronto (2009).

[12] Saito, M. and Matsumoto, M., "Simd-oriented fast mersenne twister: a 128-bit pseudorandom number generator," in [*Monte Carlo and Quasi-Monte Carlo Methods*], Keller, A., Heinrich, S., and Niederreiter, H., eds., 607–622, Springer Berlin Heidelberg (2008).

[13] Dogdas, B., Stout, D., Chatziioannou, A. F., and Leahy, R. M., "Digimouse: A 3D whole body mouse atlas from CT and cryosection data," *Physics in Medicine and Biology* **52** (2007).

[14] Fang, Q. and Kaeli, D. R., "Accelerating mesh-based Monte Carlo method on modern cpu architectures," *Biomedical Optics Express* **3**, 3223–3230 (12 2012).