High-performance, robustly verified Monte Carlo simulation with FullMonte

Jeffrey Cassidy
Ali Nouri
Vaughn Betz
Lothar Lilge

Jeffrey Cassidy, Ali Nouri, Vaughn Betz, Lothar Lilge, "High-performance, robustly verified Monte Carlo simulation with FullMonte," *J. Biomed. Opt.* **23**(8), 085001 (2018), doi: 10.1117/1.JBO.23.8.085001.

High-performance, robustly verified Monte Carlo simulation with FullMonte

Jeffrey Cassidy,[a,*] Ali Nouri,[a] Vaughn Betz,[a] and Lothar Lilge[b,c]
[a]University of Toronto, Department of Electrical and Computer Engineering, Toronto, Ontario, Canada
[b]University of Toronto, Department of Medical Biophysics, Toronto, Ontario, Canada
[c]Princess Margarent Cancer Centre, Toronto, Ontario, Canada

Abstract. We introduce the FullMonte tetrahedral 3-D Monte Carlo (MC) software package for simulation, visualization, and analysis of light propagation in heterogeneous turbid media including tissue. It provides the highest computational performance and richest set of input, output, and analysis facilities of any open-source tetrahedral-mesh MC light simulator. It also provides a robust framework for statistical verification. A scripting interface makes set-up of simulation runs simple, including parameter sweeps, while simultaneously providing customization options. Data formats shared with class-leading visualization tools, VTK and Paraview, facilitate interactive generation of publication-quality fluence and irradiance maps. The simulator can read and write file formats supported by other similar simulators, such as TIM-OS, MMC, COMSOL (finite-element simulations), and MCML to support comparison. Where simulator features permit, FullMonte can take a single test case, run it in multiple software packages, and load the results together for comparison. Example meshes, optical properties, set-up scripts, and output files are provided for user convenience. We demonstrate its use in several test cases, including photodynamic therapy of the brain, bioluminescence imaging (BLI) in a mouse phantom, and a comparison against MCML for layered geometries. Application domains that can benefit from use of FullMonte include photodynamic, photothermal, and photobiomodulation therapies, BLI, diffuse optical tomography, MC software development, and biophotonics education. Since MC results may be used for preclinical or even clinical experiments, a robust and rigorous verification process is essential. Being a stochastic numerical method, MC simulation has unique challenges associated with verification of output results since observed differences may be due simply to output variance or actual differences in expected output. We describe and have implemented a rigorous and statistically justified framework for comparing between simulators of the same class and for performing regression testing. [DOI: 10.1117/1.JBO.23.8.085001]

Keywords: Monte Carlo; photodynamic therapy; bioluminescence imaging; simulation; MCML; diffuse optical tomography.

Paper 180211R received Apr. 9, 2018; accepted for publication Jul. 10, 2018; published online Aug. 10, 2018.

1 Introduction

Monte Carlo (MC) modeling of the Boltzmann (radiative transfer) equation for light propagation has a long history,[1] including previous 2-D open-source implementations for layered media (the venerable MCML[2]) and 3-D open-source efforts based on cubic voxel models[3–5] or tetrahedral mesh representations, such as MMC[6] and TIM-OS.[7] Unlike the commonly used diffusion approximation method, which makes physical approximations and is therefore inaccurate under certain conditions,[8] MC methods will converge toward the exact solution as the number of photon packets is increased, making them an attractive "gold standard" option for nontrivial geometries. The references cited above provide thorough coverage of the universally adopted "hop, drop, spin" kernel and its associated variance-reduction techniques, so we will not re-explain them here.

We prefer a tetrahedral mesh geometry over a cubic grid voxel approach for both computational efficiency in low-scattering voids, as well as its ability to approximate smooth surfaces, which is important where abrupt changes in optical properties occur.[9] In cases with refractive index changes (notably areas with air- or liquid-filled voids), the lack of smooth normals causes inaccurate refraction calculations. When disparate materials meet at a curved surface, voxels containing a mix of materials also hinder estimation of fluence from the absorbed photon weight. By contrast, a tetrahedral-mesh approach permits arbitrarily good approximation to curved surfaces, as well as the ability (supported in FullMonte but not TIM-OS or MMC) to directly score fluence transmission through interior triangular mesh faces where desired. We use this feature to simulate fluence in the very thin surface layers such as the urothelium in the bladder.

Despite on-going advances in processor and compute-accelerator technologies, MC simulation run-time remains a constraining factor. Compared with cases constrained by symmetry (e.g., MCML's cylindrically symmetric geometry reduces the dimensionality), working with 3-D heterogeneous tissue requires more packets and hence more run time for accurate results. The time and expertise required to develop, validate, and tune high-performance MC software remain substantial as well.

Our open-source package "FullMonte" addresses all of these concerns by providing the highest-performance open-source simulation kernel, augmented by scripting visualization and analysis capabilities that are easy to use and customize. Performance is achieved through the use of low-level optimized C++ code described in Sec. 6. Since the code is interoperable with other MC simulators and comes with both unit and

*Address all correspondence to: Jeffrey Cassidy, E-mail: jeffrey.cassidy@mail.utoronto.ca

regression tests, users may have confidence in results obtained by FullMonte and may easily cross-verify them with other simulators using our statistical framework. Not only is such comparison possible, it is also integrated within an automatic test suite for multiple test cases and can be run at any time with a single command to give a summary of test pass/fail results. For ease of use, integration of the Tcl ("tickle," Tcl developer xchange www.tcl.tk) scripting language relieves the burden of reading and writing C++ code for most users while permitting efficient batch processing. By connecting to the widely used open-source, cross-platform Visualization Toolkit (VTK, Kitware, Inc., New York, New York www.vtk.org), and the Paraview GUI built on top of it (Kitware, Inc., New York, New York www.paraview.org), we also enable interactive, publication-quality 3-D graphics with a powerful and intuitive interface used by many leading organizations including US national labs. The C++ code provides many customization points for software developers to build on, avoiding duplication of effort in recreating and reverifying the basic kernel before testing new types of MC data collection, simulation methods, or light sources. By building on top of the provided platform, developers can save considerable time in designing, tuning, and verifying their code and storing and analyzing data.

Users are invited to register free of charge at our Gitlab site GitLab, San Francisco, California (www.gitlab.com/FullMonte/FullMonteSW) and view the most up-to-date code, testing results, Docker images, wiki tutorials (including inputs, outputs, and visualizations), and issue tracker. The remainder of the paper shows the highlights of that content: how to install and run the simulator, use cases and means of extending the simulator's functionality, verification methodology, verification results against other simulators, and performance comparisons.

2 Using the Simulator

To illustrate the use of FullMonte, we provide in the package the data and scripts necessary to simulate photodynamic therapy (PDT[10]) of the brain using an interstitial point source. Since PDT relies on light-activated drugs, the treatment efficacy depends on the fluence, throughout the target volume.

The easiest way to install and run FullMonte is through its Docker image available from our Gitlab registry (www.gitlab.com/FullMonte/FullMonteSW). Docker, San Francisco, California (www.docker.com) is a lightweight virtual machine that runs on Linux, Mac, and Windows, and provides a way for differing host operating systems to run programs in a uniform environment. It runs images (which contain their own files, libraries, programs, and configuration settings) in isolated environments called containers. There are also convenient facilities for "pulling" images from remote servers when new versions are available. Since all necessary data are stored in the image, FullMonte can run from its Docker image without the need to install prerequisite libraries or compile code; on starting the container, the simulator is ready to run.

The Tcl scripting interface exposes all the functionality of the underlying C++ classes. On a successful install, there will be a program `tclmonte.sh`, which can be called with a single argument specifying the Tcl script file to run. Most of FullMonte's examples are provided as Tcl scripts within the Docker image (see container path `/usr/local/FullMonteSW/share/Examples`), so that users may set up experiments by modifying existing scripts rather than starting from scratch. Several of the scripts come with comprehensive tutorials on the Gitlab wiki (www.gitlab.com/FullMonte/FullMonteSW/wikis), which include step-by-step discussion plus all input data, scripts, output data, and visualization files required to replicate the figures. Paraview can load and save its program state (data files, filters, color map, viewpoint, etc), and we provide such state files with the examples to make figure generation simple.

New users should start by locating a suitable example in our wiki, running it, confirming that the output is as expected, and then modifying the associated script appropriately to change the mesh, optical properties, source characteristics, packet counts, etc. Tcl's syntax is simple, admitting a single-page description yet still permitting access to all of FullMonte's predefined classes, whose capabilities include batch runs, material property perturbation, source placement, loading and saving any of the supported file formats, simple GUI support with Tk.

Within the FullMonte distribution (Docker image or fully compiled suite), there are also command-line tools designed to present FullMonte in a way that mimics the calling conventions and input file types (meshes, sources, optical properties) of other simulators (MMC, TIM-OS). These are primarily useful when cross-verifying with the other simulators. To understand how those programs work, please consult the documentation for MMC or TIM-OS, the FullMonte wiki, and the built-in test cases within the source tree.

Advanced users and developers may also opt to install directly from source code using the Git repository, which requires installation of several prerequisite packages and is considerably more time-consuming. The only benefit of that approach is the ability to customize the code. Since FullMonte is open-source with a modular design, advanced users may also customize and/or link FullMonte into their own C++ code, as we have done to apply FullMonte to convex optimization of PDT source power allocation and placement.[11]

Below, we provide details of each of the phases of an *in-silico* experiment with FullMonte.

2.1 *Running Your First Simulation on Docker*

After creating a Gitlab user account (replace `user.name` below with your account name), you can login to the registry and run the image by typing: `docker login -u user.name registry.gitlab.com/fullmonte/fullmontesw/fullmonte-run docker run -pull -rm -ti -v /host/path:/output registry.gitlab.com/fullmonte/fullmontesw/fullmonte-run`

Note that only a single login command is required per session. The options provided cause Docker to pull an updated image from the registry (`-pull`) when available, delete the container when it is finished running (`-rm`), make a host path accessible to the container (`/host/path`, which should be replaced by the location you want the output data to appear; data written to `/output` in the container will appear on the host at that path), provide a terminal (`-t`) and run interactively (`-i`). Within that container, you can run a simulation of a point source located inside the bladder by invoking the simulator inside the container and giving it a script name to run: `cd /output tclmonte.sh /usr/local/FullMonteSW/share/Examples/BladderDirectional/sim.tcl`

Simulation time will depend on machine performance but may be a few tens of seconds. The wiki page "directed surface scoring" gives a detailed walkthrough of the simulation results and visualization.

2.2 *Problem Definition*

Inputs which define the problem—geometry, optical properties, and source configuration—are the key arguments to the program that change the expected result. The FullMonte kernel itself outputs its results in terms of packet weight. If N packets are launched, then the total energy absorbed plus exited from the system will equal N arbitrary units. To model total energy emission from the source E_0 in physical units (e.g., Joules), the results must be normalized by a factor of E_0/N, which is a task left to the user.

Technically, packet count then alters the expected output, but only by a proportionality constant that should be removed in postprocessing. Its most important effects are on result variance and run time, so we class it as a runtime option instead of part of the problem definition.

The units of length must be consistent between the mesh and the optical properties, which are given in inverse-length units, and results will be reported in the same units. For instance, if a mesh is given in mm, then the optical properties will be interpreted as mm^{-1} and after applying the `EnergyToFluence` filter on the data, results will be in $\frac{\text{packet weight}}{\text{mm}^2}$ (which can be rescaled to $\mathrm{J\,mm}^{-2}$ or $\mathrm{W\,mm}^{-2}$ as desired). A mismatch between the mesh unit and optical-property unit will cause the simulator to use incorrect optical properties.

2.2.1 *Geometry*

Delineation of medical images and conversion of the results into a volumetric tetrahedral mesh (www.cgal.org) are complex tasks but both commercial and open-source (segmentation: ITK-Snap www.itksnap.org, meshing: CGAL www.cgal.org) tools exist, so we provide only file import-export facility in FullMonte. While there are many file formats, tetrahedral meshes are fundamentally composed of points and tetrahedra, with each tetrahedron being assigned to a region. Points are specified in a list with (x, y, z) coordinates for each. Tetrahedra are given by the index in the point list for each of its four vertices and an associated material code. FullMonte can read and write the mesh formats used by other simulators (TIM-OS, MMC), the COMSOL multiphysics package, and the open-source Visualization Toolkit (VTK). The Colin27 brain mesh produced by Fang[6] and distributed with MMC is used below as an example (Fig. 1) with over 400,000 tetrahedra.

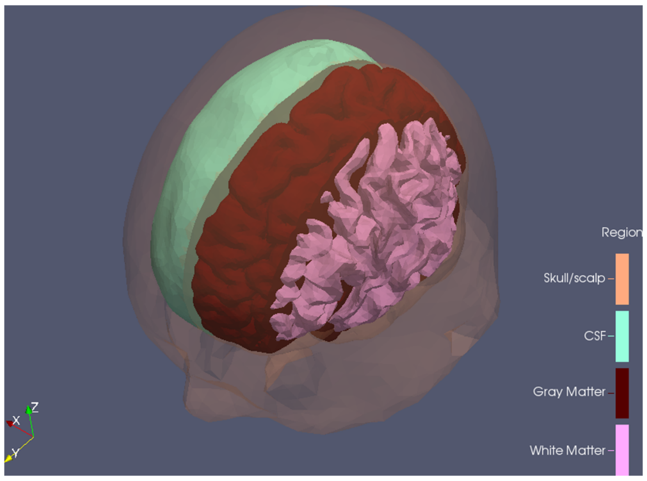

Fig. 1 Cutway view of Fang's Colin27[6] mesh used for testing, a good example of adaptive mesh sizing and representation of curved surfaces.

When asked to handle MCML layered geometries, each layer is represented as a degenerate tetrahedron with two parallel faces in the xy-plane (the top and bottom layer bounds) and two faces at infinity. Layered cases may therefore be run through the unmodified tetrahedral kernel.

2.2.2 *Optical properties*

Each of the regions (subsets of the tetrahedra with identical material code) is associated with a material for simulation. FullMonte currently supports a maximum of 32 distinct materials, though that could be increased without major effort if needed. Optically, each material is described by its scattering coefficient μ_s, absorption coefficient μ_a, anisotropy factor g, and refractive index n, which can be provided by TIM-OS or MMC formatted text files, or directly in the body of a Tcl script. The scattering and absorption coefficients are specified in inverse length units, which must match the units of the geometry description.

2.2.3 *Light sources*

FullMonte currently supports many source types, a superset of its peer simulators with the exception of MMC's wide-field illumination:[12] isotropic point source, isotropic line source, isotropic emission from within a tetrahedral element, pencil beam, ball [sphere discretized into a set of tetrahedral mesh elements, useful for bioluminescence imaging (BLI)[13] experiments], and composite sources consisting of any weighted linear combination of the above.

We place sources by using FullMonte to convert the input mesh from its native format to VTK and loading it in Paraview for visualization. Paraview provides widgets for placing geometric primitives like points, lines, and spheres as well as facilities for picking geometric elements like individual tetrahedra from the mesh. The resulting parameters can be provided through the scripting interface (or indeed C++ code if the user is linking FullMonte with their own program).

Settings can also be saved to or read from text file formats using a few lines of script. When running MMC-format cases, the source parameters are extracted from a JSON settings file. FullMonte can also read TIM-OS' text source definition, which is limited to combinations of point, triangle, tetrahedral, and pencil beam sources. Due to its radial symmetry, MCML allows only a pencil beam normally incident on its layers, so the use of FullMonte's layered kernel implies that choice of source.

2.3 *Runtime Options*

By runtime options, we mean settings which impact the behavior of the simulator in ways other than the first moment of the output distribution. The expected results are invariant to the run parameters, but the result variance, the distribution of variance across different output quantities, and the computational run time may change significantly.

The number of packets to simulate (N_0) is the most important such quantity, since run time is proportional to N_0 while output quantity uncertainty (standard deviation) decreases in proportion to $\sqrt{N_0}$. Users will need to assess qualitatively or quantitatively what level of accuracy is acceptable given the runtime cost. We opt for 10^6 as a default value that often yields a fluence map of subjectively reasonable quality within seconds to tens of seconds for the BLI and PDT cases tested so far. The packet count required to achieve a given level of quality depends on a complex interaction of mesh fineness, optical properties, the output quantity of interest, tolerable level of variance, and region of interest. Unfortunately, a general method for determining an acceptable packet count for a given application does not exist so the user must rely on judgment and experiment. When examining linear combinations of elements, or when the output quantity otherwise contains some variance-reducing tendency (e.g., dose-volume histograms for PDT),[14] then far fewer (10× or more) packets may be required with correspondingly faster run time. For high spatial resolution, low desired output variance, and low-fluence/low-interaction regions of interest, orders of magnitude more packets may be called for to get acceptable output statistics. The user can assess result variance by running multiple times with different random seeds. FullMonte provides both command-line and scripting methods to generate per-seed results or a mean–variance summary.

For finer control of time-quality tradeoffs, the roulette threshold[15] (weight threshold below which packets are subjected to roulette elimination) modifies the run time-variance tradeoff in elements with very low fluence. Since roulette bundles a number of low-weight packets into a single higher-weight packet, increasing the threshold will speed up the simulation at the cost of fewer but higher-weighted absorption events (i.e., higher result variance) in low-fluence regions. MMC provides a command-line option with a default value of 10^{-6}, whereas TIM-OS has an unnamed constant fixed at compile time (TIMOS.cpp line 1782), FullMonte has multiple ways to set the value which defaults to 10^{-5}, and MCML is fixed at compile-time to 10^{-4} (MCML.h line 74). To compare simulator performance and quality fairly, this value should be set equally for all parts of the comparison. We use FullMonte's value of 10^{-5} when comparing with MMC and TIM-OS, and 10^{-4} to match MCML.

2.4 *Simulation Kernel*

FullMonte's kernel is designed so that the same photon-propagation logic can be used to capture a number of different lifecycle events in various and configurable ways detailed in Sec. 3. The most common case—scoring surface exit and volume absorption events—is the one provided with the standard command-line interface. It outputs the total, non-normalized (non-unitized) packet weight exiting through each surface face and deposited in each volume element. These choices reflect a design preference for maximal simplicity so that the output from the kernel is exactly what was recorded in the simulation, with all normalization, unit conversion, and changes of physical quantity (e.g., energy to fluence) being deferred to a postprocessing stage. Conservation of energy can be checked by summing the total surface and volume scores, and comparing against the total launched.

When cross-verifying FullMonte, we found and reported a bug in MMC that appears under certain combinations of options (Havel raytracing, energy output, piecewise-constant scoring basis `-O E -C 0 -M H`) that is a direct artifact of a more-complex control path in which the control flow through photon tracing depends on multiple command-line options: ray-tracing method, output type, and output coordinate basis. Our election to keep photon propagation, event scoring, and output postprocessing code fully separate with well-defined interfaces reduces the probability of such errors.

2.5 *Output Data*

When the result is normalized by the number of packets (a factor of $1/N_0$), it gives a Green's function solution in units of dimensionless weight. Packet weight is analogous to photon arrival probability (directly proportional to energy via $E_\gamma = h\nu$), so multiplication of the Green's function by the appropriate number of physical photons emitted N_γ, or the total energy E_0 emitted gives the result as a physical quantity. The results may also be interpreted per unit time as photon or radiant flux.

FullMonte also has the ability to score the energy crossing internal boundaries, which can be useful for modeling thin layers, such as the urothelium for PDT of the bladder. Meshing such structures directly is computationally expensive, may produce meshes with many small tetrahedra that degrade quality (in terms of output variance) and numerical stability. Separate logging of photon transit through the layer is preferable because it avoids these issues. A detailed example is provided with the source distribution and Docker image (`Examples/BladderDirectional`).

2.6 *Postprocessing*

From the raw exit and absorption scores (s) and a specification of total energy emitted (E_0), FullMonte can produce a number of different outputs. Per volume element, it can provide physical energy absorbed by rescaling weight ($E = s/E_0$), average energy density (E/V), or average fluence ($\Phi = E/V\mu_a$). Over the surface, typically, it is the emittance (exit energy per unit area) that is of interest. Other options include dose–volume and dose–surface histograms for regions partitioned by tissue type.

2.7 *Visualization*

Visualization is made possible using VTK. Basic Tcl script users can export `.vtk` files for viewing in Paraview, as shown in the examples. More advanced users can automate generation of figures using VTK's own Tcl scripting interface from the files generated. VTK and Paraview provide their own extensive documentation and tutorials showing how to use them.

2.8 *Result Export*

We provide simple text-format export facilities so that the mesh and associated per-element data can be saved and read into other postprocessing tools, such as MATLAB. When running multiple simulations of the same problem, we define a mean–variance file format for future comparison. Data from TIM-OS, MMC, and MCML can also be imported for comparison or conversion purposes.

3 Extensibility and Use Cases

Those interested in simply using FullMonte may safely skip this section; anyone interested in understanding how it was constructed or in customization should find this of interest.

Since it was designed with modularity and customization in mind, it is easy for developers to add features to FullMonte while preserving the correctness and performance of the existing code. We discuss in Sec. 4 a set of verification strategies that reduce the probability of modifications introducing undetected bugs. Functionalities, such as reading input, simulating, post-processing, and writing output, are grouped into related concepts (families of related C++ classes) in FullMonte, including:

- Reader—reads geometry definitions from various formats
- Scorer—processes event notifications from the photon-tracing logic to accumulate output scores
- Kernel—binds one or more scorers to the photon-tracing loop
- Query—takes output data and converts or compares it
- Source—a theoretical description of a light source
- Emitter—a concrete class responsible for launching photons described by a source into the tracing loop given a random-number generator
- OutputData—holds output scores from the scorer
- Writer—writes geometry, material, source, and output-data information to various file formats.

The various concepts are largely independent and relate only through well-defined interfaces, which helps to localize the scope of changes to be made to add new features. None of the extensions listed below required any change to the core photon-propagation logic, which benefits simplicity, verification, and maintenance.

3.1 *MCML Support*

For verification, we modified FullMonte to support MCML-like geometry definitions and scoring since it is widely used and trusted. Handling the problem description required creation of a new reader class to handle the.`mci` file format, extracting the material definitions, layer information, and scoring grid information. We created a facility to convert the layered geometry into a degenerate tetrahedral mesh in which the top and bottom faces of each tetrahedron are parallel in the xy-plane, and the other two are at infinity. Since FullMonte typically scores absorption per tetrahedral element, we created a new `Scorer` class that accepts absorption events and accumulates them to cylindrical bins defined by (r, z) as MCML does. Last, the conversion from output weight score to fluence used the existing EnergyToFluence class and required only that the newly created layered geometry description provide the volume and material identifier for each bin. The core photon-tracing logic is completely unmodified compared to the normal 3-D tetrahedral mode, meaning the change set is confined to a small amount of new code and no modifications to the propagation core.

3.2 *Path Tracing*

To visualize the operation of the simulator for debugging and educational purposes, we produced a version of the simulator to capture the full position trace of each photon, producing Fig. 2. We added a `PathScorer` that concatenates the positions from event notifications of launch, reflection, refraction, and scattering. To invoke that scorer, we created a `TetraTraceKernel` that binds the standard core loop to the `PathScorer`, without altering the core photon-propagation logic. After postprocessing to reformat the data, the result is saved as a `.vtk` file for visualization.

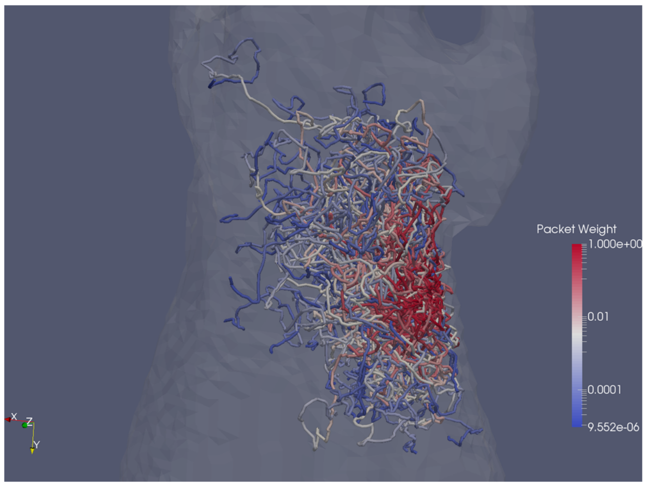

Fig. 2 A BLI testcase using an embedded source with 1k packet traces simulated by TetraTraceKernel, rendered by Paraview (script file mouse_paths.tcl, state file MouseTrace.pvsm)

3.3 *New Source Types*

Physical light sources are represented by two classes in FullMonte: a class which specifies the logical description of the optode and an `Emitter`-derived class which has the machinery to use a random-number generator to emit photons matching that description into a specific mesh. We provide an EmitterFactory class that maps logical source descriptions to concrete Emitter derived classes. When launching photons, we use C++ virtual function mechanism to dispatch a launch request from the kernel, which knows only that it has a pointer to some kind of `Emitter`, to a specific derived class which handles the details. Since photon launch is infrequent compared to ray-tetra intersection, scattering, and other operations, the overhead imposed is negligible.

4 Verification Methodology

Unit tests are provided to test that key pieces (e.g., random number generation, scattering, photon launch) behave in the expected ways. Since other MC simulators exist which employ the same roulette termination scheme, we validate against those—TIM-OS and MMC for tetrahedral meshes, MCML for layered geometries—to check that overall results are correct. As development proceeds, we rely on regression tests to check that future versions of the software produce output that conforms to the previously accepted version.

Since MC simulation is by nature a stochastic approximation to the real solution, the output will always have some level of variance, which must be accounted for when comparing results within or between simulators to determine if the difference is due to random fluctuation or a systematic difference. Random fluctuation is expected but systematic difference suggests a disagreement in result that needs to be explained. Previous efforts have tended to focus on qualitative comparison of results[6] or on aggregate reflectance/absorption with standard deviation,[7] or on imposition of arbitrary error bounds for very long (high packet count/low-variance) simulation runs.[7] In contrast, we introduce here a statistically justified comparison method based on the well-known χ^2 test.

Software testing tends to be effective only when it is repeatable, automatic, simple, and has clear results. We have

Table 1 Events impacting conservation of energy in the simulation kernel, where w_0 is the packet weight before the event, P is probability of roulette win, and μ_a, μ_s are the optical absorption/scattering coefficients.

	Event	δw
Source	Launch	$+1$
	Roulette win	$+(P-1)w_0$
Sink	Exit	$-w_0$
	Roulette termination	$-w_0$
	Absorption	$-\frac{\mu_a}{\mu_a+\mu_s}w_0$

incorporated a number of such formalized tests into FullMonte, so that both users and developers are more likely to catch faults in their code. The statistical method adopted for comparing datasets has the advantage that the developer may choose the trade-off of run time cost versus sensitivity (both increase with N_0).

4.1 *Internal Consistency Checks*

In addition to the primary outputs of exit and absorption photon scores, the simulator also produces other quantities of interest to assess performance and check conservation of energy. All provided FullMonte kernels return a result specifying the disposition of weight between total absorption, roulette increase/decrease, and exit. The values can be compared against the totals launched to verify that all energy is accounted for (Table 1 summarizes lifecycle events), where a conservation failure would indicate either a problem in the scoring logic, an unaccounted loss of weight within the kernel, or a numerical problem in weight accumulation.

MMC invests some extra computational effort in performing double-precision Kahan summation for each mesh element score to guard against underflow. We opted instead for higher-performance simple summation at double precision and check ex post if there are problems arising from that choice in the form of lost weight, an event which has not yet arisen. The total over all elements for absorption and exit can be checked against the sum of the per-element scores as well to identify whether numerical underflow is a significant issue. Such checks would also catch catastrophic failures of simulation logic in which packets are lost, which is primarily useful for developers to quickly identify logical errors or numerical precision issues when making changes in the core photon propagation loop (which ought to be infrequent, per Sec. 3).

Checks take the form of both rigid conservation laws which must be true to within rounding error and statistical checks which are zero-mean with an associated variance. Physically, photons entering the tissue will either exit or be absorbed, so the physical energy launched should equal the energy absorbed plus exiting. In simulation, roulette is a random process introduced as a computational shortcut that preserves expected conservation of weight (energy), so weight added via roulette wins less that dropped by roulette termination should equal zero within statistical variance. The results should account for all energy so the total weight sourced (by launch and roulette) and disposed should be equal to within rounding error; any imbalance in excess of the roulette variance can be attributed to numerical precision effects.

Regardless of the scoring system (spatial binning) employed, the total of each absorption or exit score should also match that shown in the conservation scorer to within rounding error. Intuitively, the total weight of all photon absorption or exit events ought to equal the sum over all volume and surface elements.

4.2 *Unit Testing*

The source code provided for FullMonte includes a number of unit tests verifying key functions, such as the distribution of photons emitted by the various `Emitter` classes, the distributions of the random-number generators, and geometric primitives, such as ray-tetrahedron intersection tests crucial to the success of the kernel. After compiling the code, a complete set of unit tests can be run by the command `ctest -R unit_` to provide a summary with the number of tests run and identification of any failing tests. Provision of unit tests also provides a quick check that any new feature or performance enhancement has not caused the existing code to become inaccurate, so that developers may work to improve performance or features but remain confident that key specifications are still met.

4.3 *Cross-Verification with Other Simulators*

In the physical system being simulated, the radiance $L(\mathbf{r}, \hat{\mathbf{n}}, t)$ as a function of position $\mathbf{r}$, direction $\hat{\mathbf{n}}$, and time t must be a solution to the radiative transfer equation. The MC scoring system traces photon packets through the geometry, reporting photon lifecycle events (e.g., launch, arrival at an element boundary, refraction, reflection, absorption, scattering, domain exit) to `Scorer` classes, which accumulate scores to approximate physical quantities of interest. Typical scoring arrangements include the energy absorbed within a volume or passing through a surface, though they can be further refined by subdividing into slices of time, angular resolution, etc. We consider the statistical distribution of a single packet's contribution $\mathbf{S}$ to the output score vector in terms of its mean μ and variance σ^2. In any correct MC formulation, the mean μ must be proportional to the physical solution dictated by the radiative transfer equation. On the other hand, the per-packet variance and its distribution between the output elements depend on the exact MC formulation and variance-reduction schemes (most notably here, roulette form and parameters).

To conduct a simulation, we generate N independent photon paths whose contributions to the score vector $\mathbf{s}$ are drawn from $\mathbf{S}$ and sum their contributions to produce an output score $\mathbf{x}$ with elements $x_j = \sum_{i=1}^{N} s_{ij}$. Its expected value and variance are given by

$$\mathbb{E}[\mathbf{X}] = N\mu, \tag{1}$$

$$\mathrm{Var}[\mathbf{X}] = N\sigma^2. \tag{2}$$

When using an MC simulator, variance is typically a secondary consideration as users are concerned only to set a packet count N to make the variance negligibly small for their purposes. To robustly compare two datasets, though, the tolerance permitted needs to account for the expected variance to assess whether a difference is statistically significant, or can plausibly be dismissed as random variation.

We pose equality comparison of two independent output samples $\mathbf{x}, \mathbf{y}$ as a test of the null hypothesis that they are drawn from independent MC simulations of the same problem, which necessarily means independent identically distributed packet score contributions described by μ, σ. Considering a measure of the difference Δ with different numbers of packets simulated N_x, N_y yields

$$\mathbb{E}[\mathbf{X}] = N_x \mu, \tag{3}$$

$$\mathbb{E}[\mathbf{Y}] = N_y \mu, \tag{4}$$

$$\mathbf{\Delta} = N_y \mathbf{x} - N_x \mathbf{y}, \tag{5}$$

$$\mathbb{E}[\mathbf{\Delta}] = 0. \tag{6}$$

While the difference simulating identical systems is expected to be zero, it will, in practice, have some variance, which must be distinguished from systematic difference. Previous work has relied on subjective comparison, imposition of arbitrary tolerances, excluding low-value (which generally means high-variance) elements, or running very long large packet counts to drive the variance to nearly zero. By considering the MC result variance, we arrive at a more rigorous standard in which the simulated data itself is used to derive an appropriate comparison tolerance. Under the null hypothesis, the per-packet score contribution parameters are the same, so by substitution of Eq. (2), we know the variance of the difference vector:

$$\mathrm{Var}[\mathbf{\Delta}] = \sigma^2 (N_y^2 N_x + N_x^2 N_y). \tag{7}$$

Since the actual score contribution variance σ^2 is not known, it must be estimated. Let us consider the comparison of a reference arm $\mathbf{x}$ generated by simulating N_r independent runs of N_p packets to provide empirical estimates of variance via Eq. (2). Though the score contribution vector is not normally distributed, a sufficiently large summation of such event contributions will converge toward normal. With that estimate in hand, we may proceed to test whether an observed difference Δ falls within the distribution proposed by the null hypothesis.

To assess whether the simulator in the test arm produces significantly different results, we use it to generate a single simulation output $\mathbf{y}$, possibly using a different number of packets. Using the variance estimated from the reference arm, we assume that the variance of the test arm is the same, scaled to the number of photons run by using Eq. (2). Since the convergence relied upon above for estimating σ depends on the frequency of accumulation events in the score vector, different vector elements will require different numbers of packets to converge to a given level. We can speed up the test by considering only a subset of elements $\mathcal{C}$ that converge at a relatively small number of packets for comparison. Taking elements with a coefficient of variation (σ/μ) that is under 10% in both the reference and test arms has worked well as a guideline.

Under those assumptions, we produce a vector of z-scores for the difference $\mathbf{\Delta}$ between the reference and test results:

$$z_i = \frac{1}{\sqrt{N_y^2 N_x + N_x^2 N_y}} \frac{N_y x_i - N_x y_i}{\sigma_i}. \tag{8}$$

The sum of squares of $k = |\mathcal{C}|$ such variables is χ^2 distributed with k degrees of freedom, leading to a standard statistical test of the null hypothesis with test statistic:

$$X^2 = \frac{1}{N_y^2 N_x + N_x^2 N_y} \sum_{i \in C} \frac{(N_y x_i - N_x y_i)^2}{\sigma_i^2}. \tag{9}$$

If the observed vector is sufficiently improbable given the variance established in the reference arm, then we conclude that the test and reference arms model different phenomena.

4.3.1 *Test coverage*

Because the test requires that the elements in the comparison set $\mathcal{C}$ are approximately normally distributed, we must ensure that we run sufficient packets to achieve good statistics for all elements in $\mathcal{C}$. If there are not, then the sample variance is not a reliable estimate and the observed difference may not be normally distributed so the test statistic will not be χ^2 distributed. Each component of the score vector will have a different distribution and rates of interaction, implying a different number of packets to satisfy the assumption. All else equal, the higher the fluence in an element the higher the probability of an interaction event, so regions closer to light sources will have better statistics. Larger elements (volume or area) also receive more events due to a larger region of integration. For volume elements, the transport mean-free path $(1/\mu_\alpha + \mu_\mathrm{s})$ also impacts the frequency of photon-packet interactions since more interactions occur per unit length traveled. We can choose a trade-off between test coverage and run time by comparing a subset $\mathcal{C}$ of the domain.

We use the coefficient of variation of the score as a reasonable and readily available criterion for choosing that subset. Since the actual distribution is non-negative, we must at least ensure that zero values of the normal distribution are many standard deviations away. In practice, excluding elements with a coefficient of variation exceeding $v_0 = \sigma/\mu > 0.1$ has worked well. In the software implementation of the χ^2 test, we order the elements in terms of ascending coefficient of variation. The user may request tests of two forms: either include elements with a coefficient of variation below a cutoff value $v \leq v_0$ or include elements in ascending order until they cumulatively account for some specified fraction (e.g., 95%) of the total weight emitted. As more packets are simulated, the coefficient of variation of each element will decline so more elements will meet the inclusion criteria, providing a way to trade-off speed (smaller N_0, faster simulation, higher variance) for sensitivity (larger N_0).

4.3.2 *Use in regression testing*

In software engineering, regression testing is the process of ensuring that modifications (feature additions, bug fixes, performance enhancements) do not invalidate the existing key behaviors of a piece of software. We can use the developed χ^2 test method above to compare two versions of the same piece of software for regression testing, by using stored mean and variance data from a previous version of the software to provide the reference arm with the new software under test as the test arm.

One of the benefits of this test policy is that the runtime is tilted strongly toward the reference arm, since it requires N_r runs of N_p packets (we have generally opted for $N_r = 64$ and $N_p = 10^6$, respectively). Since the idea is to ensure that the software's key behaviors have not been altered in the course of modifications, a regression test should be changed rarely if at all so the expensive variance-estimation runs are only required when establishing a new regression test. Once the test is accepted and added to the suite, only the mean and variance values need to be stored. Subsequent checks of the test case require only a single run of N_p (or even possibly fewer) packets, which is N_r times faster than test case creation.

4.3.3 *Use in cross-verification testing*

In creating the regression test cases described above, the test engineer must be concerned with the validity of the reference data. In isolation, the test will only show if the behavior has changed, not whether it was correct in the first place. Before committing a new regression test to the version-control repository, we therefore check it against one or more other simulators. For tetrahedral geometries, we run two tests against the provided reference arm, with either MMC or TIM-OS providing the test arm. In the layered case, we compare against MCML using a modified tetrahedral geometry kernel and the same cylindrical scoring system. The null hypothesis is that the result provided by the other simulator is similar within statistical variance to the stored regression test mean and variance. If agreement is sufficiently improbable, it indicates a problem with the test case that must be diagnosed and resolved before committing it.

5 Cross-Verification Results

As part of the FullMonte distribution (www.fullmonte.org), we provide four full regression test cases with mean–variance reference data and a build target that runs the checks (details are provided on the software's wiki page). To facilitate three-way comparison, the tetrahedral cases were restricted to point sources so they could be run exactly on both TIM-OS and MMC. For each case, mean–variance data were generated with FullMonte and compared to the applicable other simulators prior to acceptance. Acceptance criteria require that the χ^2 tests cover both 95% of total scored weight and all elements with a coefficient of variation $v \leq 10\%$, with a rejection significance of $p = 0.05$.

The Digimouse mesh and optical properties were obtained from TIM-OS' example suite and contain different values for the various anatomical structures. The Colin27 mesh was borrowed from MMC's example suite, including a set of optical parameters for 630 nm for scalp/skull, cerebral-spinal fluid, gray matter, and white matter. For both Colin27 and Digimouse, the refractive indices are all matched, with the exception of the skin/air interface. Meshes were converted to VTK format using FullMonte's import–export facilities, and then loaded into Paraview for manual source placement at an arbitrary location deep within the mesh. The two MCML cases are taken from the MCML source code distribution with multiple layers of inhomogenous optical properties (including refractive index); the light source is implicitly a normally incident pencil beam. The test cases are summarized below with their relevant statistics (Table 2).

Both MCML test cases were very narrowly rejected as originally posed. If we permitted a very small increase to the coefficient of variation, adding an extra 0.1% of the mean to the standard deviation, then the test passes. MCML uses a less sophisticated, shorter-period random number generator so it may be that some residual correlation is causing a bias. The results matched MMC exactly as run, while TIM-OS required a 5% increase in variance to be accepted. These results are within reason and have adopted the slightly looser standard deviation bounds (TIM-OS, MMC 1.05σ, and MCML $\sigma + 0.001\mu$) into our regression tests.

Table 2 Summary of regression test cases. In each case, the reference arm was generated from $N_r = 64$ runs of $N_p = 10^6$ packets.

		Volume	Elements
Case name	Type	Total	$\lvert\mathcal{C}\rvert$
MCML-Sample0	Layered	2000	1972
MCML-Sample2	Layered	2000	1999
Colin27-Point0	Tetrahedral	423,375	17,655
Digimouse-Point0	Tetrahedral	306,773	28,060

5.1 *Subjective Comparison*

In all test cases, the difference appears subjectively zero-mean, with the magnitude increasing away from the source as elements receive fewer scoring events, and hence, have higher variance due to a lower frequency of packet arrival and hence smaller sample. Spatially correlated structure would indicate an underlying difference in the phenomena simulated, but we note no such structure in any of the comparisons. The figures below illustrate a 3-D case (Colin27-Point0 MMC vs. FullMonte) in Fig. 3 and a 2-D case in Fig. 4 with MCML.

5.2 *Delta Statistics*

Since our comparison test relies on assumptions about the statistical distribution of the difference between simulation runs, we must first validate those assumptions. If the assumptions and the null hypothesis are true, then the scaled difference vector z of Eq. (8) will have a unit normal distribution. We checked this assumption using a Q-Q (quantile–quantile) plot, which shows each measured data point's measured value on the y axis against the value of the test distribution at the same quantile on the x. A unit-slope straight line through the origin indicates a perfect match. Figure 5 shows the result from the MCML-Sample0 case was the difference vector, which is indeed normally distributed with zero mean. The variance very slightly exceeds expected, which suggests that it may be slightly underestimated, likely due to low-fluence regions that have not quite converged to a normal distribution. Results for the other test cases were similar, indicating that the model is correct for the simulators, test cases, and parameters (run counts and packets per run) posed, so the results are well described by a normal distribution.

5.3 χ^2 *Test Validation*

Since the test is statistical, we need to check that our test is suitably sensitive and specific.

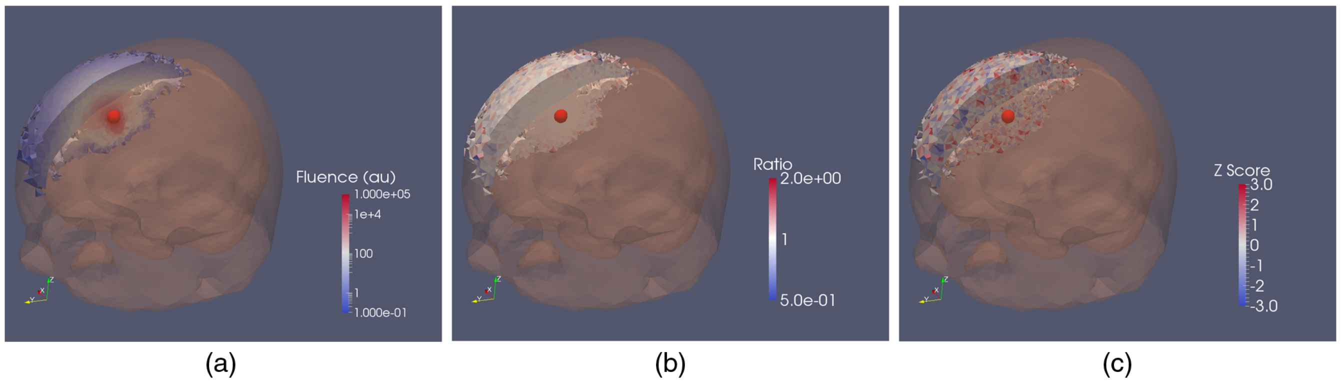

Fig. 3 Colin27-Point0 case showing a cut view through a 3-D model. Samples with a coefficient of variation exceeding 20% have been excluded. The red sphere shows the point source location. From left to right: (a) fluence, (b) MMC/FM result ratio, and (c) difference *z*-score.

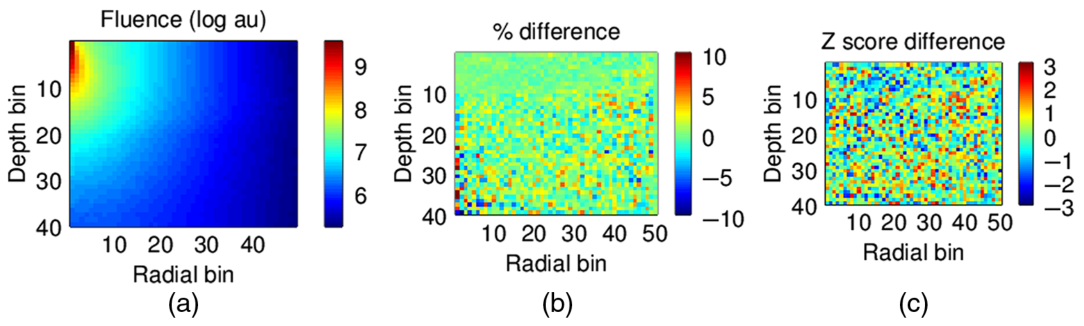

Fig. 4 Comparison of FullMonte versus MCML showing from left to right: (a) fluence, (b) percentage difference, and (c) *z*-score difference.

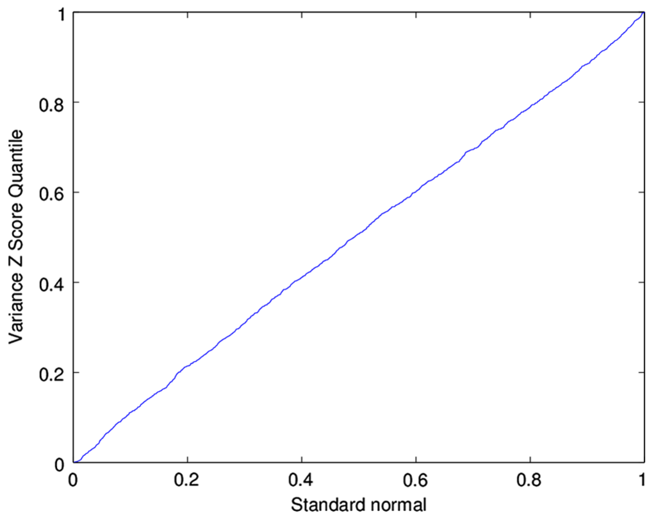

Fig. 5 A Q–Q plot of the scaled MCML-FullMonte difference vector (*z*-score) against the standard normal distribution, showing strong agreement for the MCML-Sample0 regression test.

5.3.1 *Specificity*

We ran our Colin27-Point0 test case with 164 different random seeds, each time with 10^6 packets. The first 64 runs were used to produce the reference mean–variance output.

First, we ran an in-set test in which each of the 64 elements used to estimate the reference parameters was tested sequentially against the reference mean–variance values. Though this is not quite proper since the test and reference are slightly correlated (each test contributed to the reference statistics), it was a reasonable first check. Consistent with the findings on examining the delta statistics above, the variance was just slightly underestimated, which led to an incorrect rejection of the null hypothesis. We remedied that by increasing the observed standard deviation by 5%, i.e., multiplying the standard deviation by a factor of 1.05, for all comparisons. Following that change, 1/64 was flagged as having a significant difference, which is well within the specified p value for false positives.

For a more proper test of specificity, we did an out-of-set test with the additional 100 fully independent runs. Of those, 0/100 was flagged as significantly different, though the χ^2 values were on average slightly higher than the in-set test, as expected. We conclude that the test is reasonably specific as configured.

5.3.2 *Sensitivity*

To check that we can indeed detect correctly when the null hypothesis is false, we altered the optical parameters subtly and ran another 64 simulations at 10^6 packets to compare against the reference arm. With Colin27-Point0 and an increase of 1% in scattering in the gray matter, the test flagged a difference in absorbed energy 27/64 times. The distribution of χ^2 statistics from the test runs is presented in Fig. 6, with a vertical line showing the $p = 0.05$ critical value. Values to the left are accepted as identical to the reference within variance at $p = 0.05$ while those to the right are significantly different.

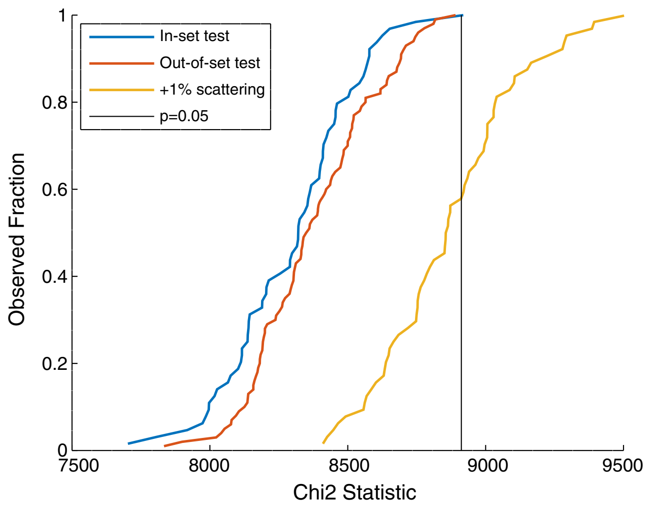

Fig. 6 The empirical cumulative distribution (CDF) of observed χ^2 values for the test covering 95% of absorbed energy in Colin27-Point0. The critical value $\chi_c^2 = 8911.91$ is shown as a vertical line and was computed for $p = 0.05$ using the normal approximation for 8694 degrees of freedom (the number of tetras included).

We conclude that the test as configured is acceptably sensitive to changes in optical properties, despite its modest run time (seconds to tens of seconds). Choosing longer simulations (larger packet counts) will leave the mean unchanged while reducing variance, which would increase sensitivity even further if needed.

6 Performance

The FullMonte kernel is written in low-level optimized C++ to make full use of x86 vector instructions (AVX/AVX2) and multithreading. Compiler intrinsic directives and the C++11 threading facilities were used directly rather than relying on compiler autovectorization (TIM-OS) or OpenMP (MMC).

Some of FullMonte's performance-enhancing features include vectorized generation of random numbers (Henyey–Greenstein, 2-D/3-D isotropic unit vectors, exponential random variables, etc.) and use of SIMD vector instructions for key geometry steps. While TIM-OS used compiler autovectorization and the Intel Math Kernel Library (MKL) to achieve some of the gains, vectorization of key distributions such as Henyey–Greenstein is both crucial to MC simulation performance and beyond the ability of compilers to do automatically. This extends the approach similar taken by Fang and Kaeli[16] in enhancing MMC, most notably using AVX2 instructions where available, hand-writing the key ray-tetrahedron intersection code, and vectorizing the Henyey–Greenstein distribution. Further advantage is gained by using Cartesian coordinates over barycentric coordinates, which is unlikely to lead to precision issues in a well-conditioned mesh. FullMonte also use single-precision floating-point arithmetic for positions and distances, and double-precision for energy accumulation to avoid rounding and underflow errors. We take the tight result agreement between MMC, TIMOS, and FullMonte as practical confirmation of these points.

All performance data presented here were measured on an Intel Xeon E5-1620 quad-core CPU running at 3.5 GHz, with AVX2 instructions. Since the processor uses Hyperthreading, each physical core can act as two logical cores, so both TIM-OS and FullMonte were run using eight threads for best performance. Performance scales linearly with the number of cores (up to 8 on an 8-core machine tested so far), so it should be possible to achieve performance that is integer factors faster with higher core counts and clock rates.

TIM-OS was compiled using the Intel C/C++ compiler with the settings as specified on its website. The website states that a lower optimization level (GCC-O2) must be used due to a crash; we remedied the cause of that bug to enable full optimization level.

MMC was installed from the binary provided online (mcx.sourceforge.net on May 29, 2017) and ran in its default tracing mode. FullMonte was compiled in Release mode with AVX2 extensions enabled (Table 3).

In the original TIM-OS study,[7] the authors demonstrated that TIM-OS outperforms MMC, parallel MCML,[2] and a GPU-accelerated version of MCML.[17] We have repeated the comparison with TIM-OS, MCML, and MMC here. The GPU-accelerated MCML code is no longer maintained, and is in any case such a restricted geometry model that we do not consider it a relevant basis for comparison. We conclude that FullMonte is the fastest open-source tetrahedral MC code available by a significant margin.

7 Conclusion

Custom MC code is time-consuming to write and validate, duplicates efforts others have already made, and without significant effort and expertise may risk being incorrect or run orders of magnitude slower than a tuned implementation. The community is better served by sharing a common, customizable, high-performance, validated platform so that researchers can answer their questions faster and with less effort. Beyond class-leading performance, it is distinctive in its extensible modular software

Table 3 Performance data for the test cases specified in Table 2 (described in Sec. 5). Speedup refers to FullMonte speed over the named simulator.

		TIM-OS		MMC		MCML	
Case	FullMonte (s)	Time (s)	Speedup	Time (s)	Speedup	Time (s)	Speedup
MCML-Sample0	2.9					8	2.7
MCML-Sample2	3.8					12	3.2
Colin27-Point0	37	64	1.73				
Digimouse-Point0	7.3	8.9	1.22	17.7	2.42		

architecture, statistically robust regression test suite, unit testing, and extensive input–output facilities. Tools for modeling, visualizing, and analyzing results are provided that integrate with powerful and widely used open-source visualization tools. The software should run on any recent x86 processor with AVX instructions (AVX2 provides a significant performance boost); we have tested it on Linux (Ubuntu 14.04, GCC 4.9) and Mac OS (GCC 4.9 and LLVM 6.0). It is also deployed via Docker, a lightweight container system for Windows, Linux, and Mac, which enables fast, easy, consistent deployment without compiling from source. FullMonte will be useful across many domains, including optical detection, imaging, and treatment of disease as well as in biophotonics education through its visualization capabilities. Prospective users are invited to our website at www.fullmonte.org to view documentation, examples, and to download the package.

Disclosures

The authors declare that they have no conflicts of interest regarding this work.

Acknowledgments

Funding was provided by Theralase Inc, IBM, Altera, the Canadian Natural Sciences and Engineering Research Council (NSERC/Altera Industrial Research Chair in Programmable Silicon), the Canadian Institutes of Health Research (CGS-D), the Ontario Centres of Excellence, and the University of Toronto. The authors also wish to thank William Kingsford, who helped package the software for distribution, and Siyun Li who found an important bug in the geometry model.

References

1. B. Wilson and G. Adam, "A Monte Carlo model for the absorption and flux distributions of light in tissue," *Med. Phys.* **10**(6), 824–830 (1983).
2. L. Wang, S. L. Jacques, and L. Zheng, "MCML - Monte Carlo modeling of light transport in multi-layered tissues," *Comput. Meth. Programs Biomed.* **47**(2), 131–146 (1995).
3. D. Boas et al., "Three dimensional Monte Carlo code for photon migration through complex heterogeneous media including the adult human head," *Opt. Express* **10**, 159–170 (2002).
4. Q. Fang and D. A. Boas, "Monte Carlo simulation of photon migration in 3D turbid media accelerated by graphics processing units," *Biomed. Opt. Express* **17**, 20178–20190 (2009).
5. S. Jacques, T. Li, and S. Prahl, "mcxyz.c, a 3D Monte Carlo simulation of heterogeneous tissues Version June 1," https://omlc.org/software/mc/mcxyz/index.html (2017).
6. Q. Fang, "Mesh-based Monte Carlo method using fast ray-tracing in Plücker coordinates," *Biomed. Opt. Express* **1**, 165–175 (2010).
7. H. Shen and G. Wang, "A study on tetrahedron-based inhomogeneous Monte Carlo optical simulation," *Biomed. Opt. Express* **2**, 44–57 (2010).
8. S. L. Jacques and B. W. Pogue, "Tutorial on diffuse light transport," *J. Biomed. Opt.* **13**(4), 041302 (2008).
9. T. Binzoni et al., "Light transport in tissue by 3D Monte Carlo: influence of boundary voxelization," *Comput. Meth. Programs Biomed.* **89**, 14–23 (2008).
10. B. C. Wilson and M. S. Patterson, "The physics, biophysics and technology of photodynamic therapy," *Phys. Med. Biol.* **53**, R61–R109 (2008).
11. A.-A. Yassine et al., "Automatic interstitial photodynamic therapy planning via convex optimization," *Biomed. Opt. Express* **9**, 898–920 (2018).
12. R. Yao, X. Intes, and Q. Fang, "Generalized mesh-based Monte Carlo for wide-field illumination and detection via mesh retessellation," *Biomed. Opt. Express* **7**, 171–184 (2016).
13. V. Ntziachristos et al., "Looking and listening to light: the evolution of whole-body photonic imaging," *Nat. Biotechnol.* **23**, 313–320 (2005).
14. J. Cassidy, V. Betz, and L. Lilge, "Treatment plan evaluation for interstitial photodynamic therapy in a mouse model by Monte Carlo simulation with FullMonte," *Front. Phys.* **3**(6) (2015).
15. J. Cassidy, L. Lilge, and V. Betz, "FullMonte: a framework for high-performance Monte Carlo simulation of light through turbid media with complex geometry," *Proc. SPIE* **8592**, 85920H (2013).
16. Q. Fang and D. R. Kaeli, "Accelerating mesh-based Monte Carlo method on modern CPU architectures," *Biomed. Opt. Express* **3**(12), 3223–3230 (2012).
17. E. Alerstam, T. Svensson, and S. Andersson-Engels, "Parallel computing with graphics processing units for high-speed Monte Carlo simulation of photon migration," *J. Biomed. Opt.* **13**(6), 060504 (2008).

Jeffrey Cassidy is a computer engineering PhD candidate at the University of Toronto. He received his BEng degree in electrical engineering from McGill University in 2006, and his MASc degree in computer engineering from University of Toronto in 2014. His research focuses on accelerating Monte Carlo simulation of light propagation using both conventional computers and FPGA custom logic, optimization of PDT treatment, and related applications.

Ali Nouri is an undergraduate student in electrical and computer engineering at the University of Toronto, expecting to graduate in 2020.

Vaughn Betz is a professor at the University of Toronto. He received his BSc degree from the University of Manitoba in 1991, his MS degree from the University of Illinois at Urbana-Champaign in 1993, and his PhD degree from the University of Toronto in 1998, all in electrical engineering. His research interests include computer simulation and optimization of photodynamic therapy, and the development of more efficient computer hardware.

Lothar Lilge is a senior scientist at the Princess Margaret Cancer Centre and professor of medical biophysics at the University of Toronto. He was trained in Germany (University in Muenster), the Wellman Laboratory of Photomedicine (Boston), and at McMaster University (Hamilton, Canada). His current research encompasses photodynamic therapy dosimetry, treatment planning, tissue response modification, and evaluation of novel photosensitizers. Other research interests include optical diagnostic techniques for mammographic prescreening and risk assessment, low level laser therapy, and optical tools in microfluidics.