Monte Carlo Simulation of X-Ray Imaging Using a Graphics Processing Unit

Andreu Badal and Aldo Badano

***Abstract*—A code for Monte Carlo simulations of radiation transport using a Graphics Processing Unit (GPU) is introduced. The code has been developed using the CUDA™ programming model, an extension to the C language that allows the execution of general purpose computations on the new generation of GPUs from NVIDIA. The accurate Compton and Rayleigh interaction models and interaction mean free paths from the PENELOPE package, and a generic voxelized geometry model, have been implemented in the new code. The secondary particles generated by Compton, photoelectric and pair-production events are not transported. An ideal x-ray detector and a cone beam source can be defined to reproduce an imaging system and facilitate the simulations of medical imaging applications. A 24-fold speed up factor with the GPU compared to the CPU is reported for a radiographic projection of a detailed anthropomorphic female phantom. A description of the simulation algorithm and the technical implementation in the GPU are provided. This work shows that GPUs are already a good alternative to CPUs for Monte Carlo simulation of x-ray transport.**

but 2009!! Likely EVEN Faster

I. Purpose

MONTE Carlo (MC) codes implementing accurate atomic interaction models are commonly used for the simulation of medical applications of ionizing radiation. However, detailed MC simulations usually require long computing times, and this limits the applicability of MC-based software in certain cases.

The purpose of this work is to investigate if MC simulations of radiation transport in x-ray imaging applications can be significantly accelerated executing the simulations in a GPU (Graphics Processing Unit) instead of in a standard CPU (Central Processing Unit), taking advantage of the vast computing power available in state-of-the-art graphic cards [1], [2].

II. Methods

A MC simulation code that runs entirely in a GPU has been developed [3] using the Compute Unified Device Architecture (CUDA™) programming model developed by the NVIDIA Corporation [4]. The code implements the accurate photon interaction models and the interaction mean free paths from the PENELOPE 2006 [5] package, which describes the radiation transport between 50 eV and 1 GeV and has been extensively validated in the past [6]. The secondary particles generated by Compton, photoelectric and pair-production events (*i.e.*, electrons, positrons and fluorescence photons) are not simulated in the GPU code. The simulation geometry is described with voxels using a file format compatible with the penEasy package [7]. To facilitate the simulation of diagnostic imaging systems, the code also includes an ideal (100% detection efficiency) energy-integrating x-ray detector and a configurable, mono-energetic x-ray source. The detector scores the energy of all particles that enter each pixel using CUDA's thread-safe *atomicAdd* function to assure that the multiple GPU threads do not modify the pixel value at the same time; the values are stored in global memory with units of meV, using *unsigned long long int* (64 bit) data type [4]. A flow diagram showing the structure of the implemented GPU code (kernel) is presented in Fig. 1.

Likely updated

The ray-tracing of the voxelized geometry is efficiently handled using a Woodcock tracking algorithm [8], [9], [10] (also called δ-scattering or fictitious interaction tracking). This technique introduces a new kind of interaction, *virtual interaction*, that makes the total mean free path constant for all the voxels, for each energy value. With a constant mean free path (equal to the minimum mean free path found in the whole voxelized object), the intersection between the photon trajectory and the voxel interfaces do not have to be calculated. If the minimum mean free path is longer than the voxel size, multiple voxels may be crossed in each step and the ray tracing may be significantly speeded up. This algorithm also optimizes the performance by minimizing the number of times that voxel compositions (material and density) have to be read from the slow GPU global memory. To implement the Woodcock tracking, the minimum total mean free path at each energy, taking into account the ratio between the material nominal density ρ^{Nominal} and the maximum density found in the voxels ρ^{Max}, is calculated as

Speed up

What? this?

Total Path??

$$\tilde{\lambda}_{\text{Woodcock}}(E) = \left[\lambda_{\text{Total}}(E,M)\frac{\rho^{\text{Nominal}}(M)}{\rho^{\text{Max}}(M)}\right]_{\text{Min}} . \quad (1)$$

Mean free path

At any point during the simulation of a photon with energy E, the distance d to the following interaction can be sampled by

$$d = -\tilde{\lambda}_{\text{Woodcock}}(E)\ \ln(\xi) , \quad (2)$$

$(-\infty, 0)$

where ξ is a uniform random number between 0 and 1. Then, considering that the photon suffers an interaction inside a voxel made of material M and density ρ^{Voxel}, the probability of having a real interaction is given by

$$P_{\text{Real}} = \frac{\tilde{\lambda}_{\text{Woodcock}}(E)}{\lambda_{\text{Total}}(E,M)\frac{\rho^{\text{Nominal}}(M)}{\rho^{\text{Voxel}}}} , \quad (3)$$

A. Badal (Andreu.Badal-Soler@fda.hhs.gov) and A. Badano are with the U.S. Food and Drug Administration, Division of Imaging and Applied Mathematics, Office of Science and Engineering Laboratories, Center for Devices and Radiological Health, Silver Spring, MD 20993-0002 USA.

The mention of commercial products herein is not to be construed as either an actual or implied endorsement of such products by the U.S. Department of Health and Human Services.

U.S. Government work not protected by U.S. copyright

What is meant by mean free path?

The mean free path is the average distance traveled by a moving particle (such as an atom, a molecule, a photon) between successive impacts (collisions), which modify its direction or energy or other particle properties.

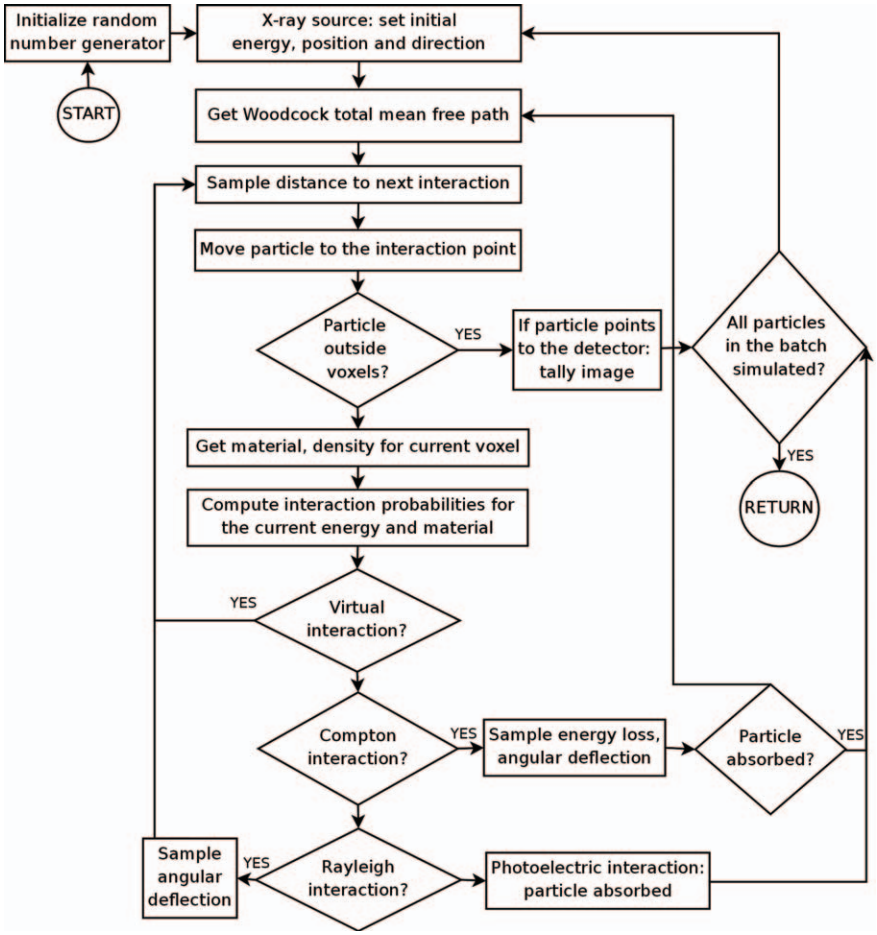

Fig. 1: Diagram of the implemented Monte Carlo photon transport algorithm for the simulation of x-ray imaging systems, using the Woodcock (or fictitious interaction) tracking scheme.

where λ_{Total} is the total photon interaction mean free path for the current energy and material (the probability of a virtual interaction is $P_{\mathrm{Virtual}} = 1 - P_{\mathrm{Real}}$). Whenever a real interaction takes place, the kind of interaction can be sampled using the corresponding inverse mean free paths

$$\frac{1}{\lambda_{\mathrm{Total}}} = \frac{1}{\lambda_{\mathrm{Compton}}} + \frac{1}{\lambda_{\mathrm{Rayleigh}}} + \frac{1}{\lambda_{\mathrm{Photoel.}}} + \frac{1}{\lambda_{\mathrm{PairProd.}}}. \quad (4)$$

The virtual interactions introduced by the Woodcock algorithm do not change the dynamic state of the photon; contrarily, the Rayleigh (elastic) interactions modify the photon direction of movement, and the Compton (inelastic) interactions modify both the energy and the direction. If the initial energy of the photons is higher than 1.022 MeV, electron–positron pairs may also be created in the track. Even though the current code does not transport charged particles, the probability of pair production events is included in the total mean free path. Both photoelectric and pair production events are simulated by finalizing the photon track and assuming that the photon energy is locally deposited.

The Woodcock, total, Rayleigh and Compton mean free paths are pre-calculated and tabulated in the CPU using PENELOPE's material database, and then stored in the GPU global memory. During the simulation, the required values are loaded using 96 bit (float3) read instructions, and the mean free paths for the particular photon energy are calculated using a linear interpolation. The hardware texture interpolation routines built in the GPU are not currently used, but could potentially be used to speed up the linear interpolation.

An essential component of a MC code is the pseudo-random number generator used to sample the probability distribution functions. For the presented code, the combined congruential generator used in PENELOPE (*ranecu*) was adapted to the GPU using CUDA. To ensure that the pseudo-random sequences used by the different GPU threads are uncorrelated, the sequence splitting technique is used. This technique is implemented by initializing the generators in each thread with randomness seeds sufficiently separated in the generator cycle to avoid the overlapping of the pseudo-random number subsequences, as implemented in the seedsMLCG code [11]. To minimize the total time spent initializing the random number generator, each thread simulates a batch of several thousands particle tracks sequentially.

In order to reduce the number of accesses to the slow GPU global memory, the detector, source and interpolation parameters are stored in the GPU constant memory (all data is expected to fit inside the fast cache). The data for the Compton cross section model is stored in the shared memory,

significantly reducing the sampling time. The Rayleigh cross section is stored in the global memory due to the large amount of data needed for this model. However, the time spent sampling Rayleigh interactions is much less than Compton because the sampling algorithm is simpler and the interaction probability is more than one order of magnitude lower at the x-ray energy range.

It is worth noting that the stochastic nature of the particle transport may cause *branching* in the code, that is, the particles simulated in parallel in different threads may suffer different kinds of interactions. Typically, branching reduces the code performance; for example, threads that suffer Rayleigh will have to wait while Compton is being sampled by other threads, and vice versa. However, the main MC loop (including the radiation source, geometry tracking, mean free path interpolation, kind of interaction sampling, etc) is always executed concurrently by all the threads, independently of the particle energy or the previous interactions. As a result, the simulation can still be accelerated by the massively parallel GPU architecture. In the particular case of simulating low energy photons (x rays) the effect of branching is even less relevant because the most probable interaction is the photoelectric effect and the majority of particles that suffer a real interaction in the geometry are readily absorbed. The effect of branching could be minimized by storing particles that suffer each kind of interactions in different *stacks* of particles, and then simulating all the interactions in parallel. However, the profiling of the current code in the GPU suggests that branching is not a limiting factor, and therefore creating the different stacks and storing the particles in memory may no reduce the total simulation time.

To show the capabilities of the new code, a sample simulation was performed using a voxelized anthropomorphic female phantom from the Virtual Family [12]. The phantom had $265\times150\times840$, $2\times2\times2$ mm^3 voxels and 9 different biological materials, with different densities in each organ. The simulation was executed in a single GPU of a NVIDIA GeForce GTX 295 dual-GPU card, which had 240 processing cores at 1.24 GHz and 896 MByte of global memory. The same code was compiled for the CPU using the Intel C compiler and the simulation was repeated in one core of a Quad Core™ 2 CPU at 2.66 GHz. For maximum performance, the code used only single precision operations, and it was compiled in the GPU using NVCC's *fast math* option [4].

III. Results

Figure 2 (a) presents the GPU-simulated projection image of the female phantom tallied using the ideal x-ray detector with 750×1350, 1×1 mm^2 pixels and located 10 cm downstream the phantom. In total 10^{10}, 48 keV x rays were simulated, with a source-to-detector distance of 100 cm (the simulated energy corresponds to the mean energy of a 90 kVp x-ray spectrum). Figure 2 (b) shows the image that would be produced by primary x rays only, that is, the previous image after removing the signal produced by the scattered radiation. The images produced by x rays that suffered one Compton, one Rayleigh or multiple interactions in the track are provided in Fig. 2 (c), (d) and (e) respectively. The speed of the presented simulation in the GPU was 3427251.5 x-rays per second, with a total computing time of 48.9 minutes. The speed of the same code executed in the CPU was 140825.2 x-rays per second. The relative difference between the GPU and CPU images was estimated to be below 1%, within the inherent statistical uncertainty of the MC simulation.

IV. Discussion, Conclusions and Future Work

We have shown that GPU computing can be used to significantly speed up MC simulation of photon transport in a complex geometry. A 24-fold speed up factor using a GPU compared to a single CPU has been reported in a realistic diagnostic imaging setting. Since the code has been designed to take maximum advantage of a massively parallel execution, the performance is expected to grow almost linearly with the number of GPU or CPU computing cores employed. For this reason, we are currently developing a version of the code that can create multiple CPU threads and that will be able to use the four cores of the CPU, or the two GPUs in the GeForce 295 card. With this new version, the difference between the CPU and the GPU is expected to be reduced. However, more GPUs can be attached to the system, and the overall performance per computer (and per dollar invested in hardware) is still much in favor of the GPUs.

Work is underway to analyze the effect of using double-precision calculations and to benchmark the results with the original PENELOPE code. An extension of the presented code for the simulation of charged particles is being considered. Variance reduction techniques could be also implemented to further accelerate the GPU code.

References

[1] P. Després, J. Rinkel, B. H. Hasegawa, and S. Prevrhal, Stream processors: a new platform for Monte Carlo calculations, J. Phys.: Conf. Ser. 102, 1–6 (2008).

[2] 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, 1–3 (2008).

[3] A. Badal and A. Badano, Accelerating Monte Carlo simulations of photon transport in a voxelized geometry using a massively parallel Graphics Processing Unit, Med. Phys. 36 (2009).

[4] CUDA™ Programming Guide (Version 2.2), Tech. rep., NVIDIA Corporation, available at www.nvidia.com/cuda (2009).

[5] F. Salvat, J. M. Fernández-Varea and J. Sempau, PENELOPE – A code system for Monte Carlo simulation of electron and photon transport, NEA-OECD, Issy-les-Moulineaux, available at www.nea.fr/html/dbprog/peneloperef.html (2006).

[6] S.-J. Ye, I. A. Brezovich, P. Pareek and S. A. Naqvi, Benchmark of PENELOPE code for low-energy photon transport: dose comparisons with MCNP4 and EGS4, Phys. Med. Biol. 49, 387–397 (2004).

[7] J. Sempau and A. Badal, penEasy—a generic, modular main program and voxelised geometry package for PENELOPE, code available at www.upc.es/inte/downloads/penEasy.htm.

[8] E. Woodcock, T. Murphy, P. Hemmings and S. Longworth, Techniques used in the GEM code for Monte Carlo neutronics calculations in reactors and other systems of complex geometry, Proc. Conf. on Applications of Computing Methods to Reactor Problems, Argonne National Laboratories Report ANL-7050 (1965).

[9] J. Sempau, S. J. Wilderman and A. F. Bielajew, DPM, a fast, accurate Monte Carlo code optimized for photon and electron radiotherapy treatment planning dose calculations, Phys. Med. Biol. 45, 2263–2291 (2000).

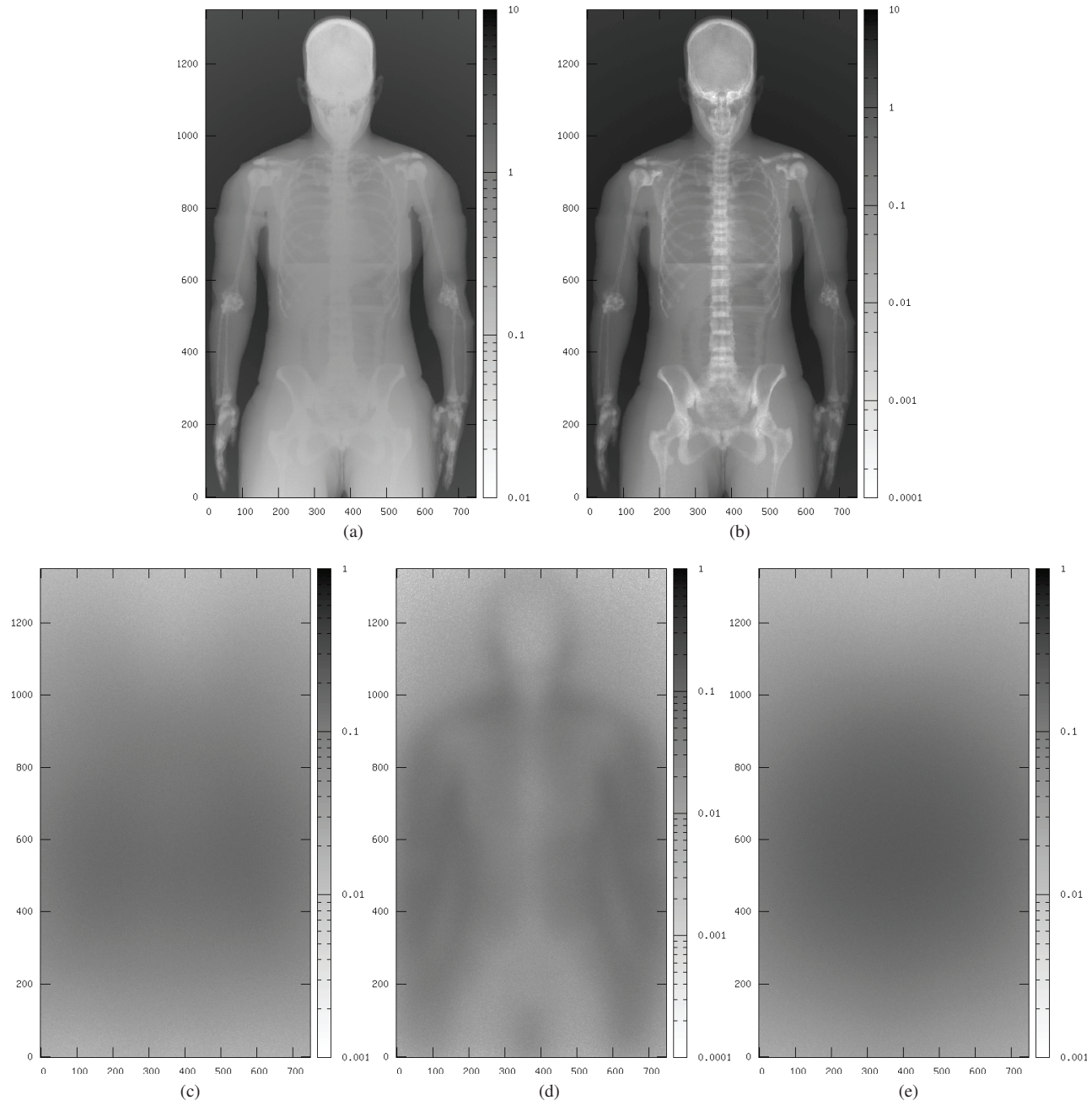

Fig. 2: Sample GPU-simulated radiographic images: (a) projection image produced by all x rays reaching the detector, (b) image produced only by non-scattered x rays, (c) image produced by x rays that suffered a single Compton scatter, (d) image by single Rayleigh scatter, (e) image by multiple scattered x rays. The units of the logarithmic gray scale are eV/cm^2.

[10] N. S. Rehfeld, S. Stute, J. Apostolakis, M. Soret and I. Buvat, Introducing improved voxel navigation and ctitious interaction tracking in GATE for enhanced efciency, Phys. Med. Biol. 54, 2163–2178 (2009)

[11] A. Badal and J. Sempau, A package of Linux scripts for the parallelization of Monte Carlo simulations, Comput. Phys. Commun. 175, 440–450 (2006).

[12] A. Christ, W. Kainz, E. G. Hahn, K. Honegger, M. Zefferer, E. Neufeld, W. Rascher, R. Janka, W. Bautz, J. Chen, B. Kiefer, P. Schmitt, H. P. Hollenbach, J. X. Shen, M. Oberle and N. Kuster, The Virtual Family—Development of anatomical CAD models of two adults and two children for dosimetric simulations, submitted to Phys. Med. Biol. (2009).