A GPU-OPTIMIZED BINARY SPACE PARTITION STRUCTURE TO ACCELERATE THE MONTE CARLO SIMULATION OF CT PROJECTIONS OF VOXELIZED PATIENT MODELS WITH METAL IMPLANTS

Andreu Badal and Aldo Badano

Division of Imaging and Applied Mathematics, CDRH, OSEL, U.S. Food and Drug Administration
10903 New Hampshire Avenue, Silver Spring, Maryland 20993, USA

ABSTRACT

Monte Carlo x-ray transport simulation codes can generate radiographic images that are equivalent to images produced by clinical systems. Most codes optimized for medical imaging use voxels to represent the patient anatomy and employ delta scattering (Woodcock algorithm) as an essential acceleration technique. With delta scattering all voxels have the same attenuation and x rays cross multiple voxels in each step reducing the time spent accessing memory and computing voxel interfaces. A drawback of this approach is that it is inefficient in phantoms with highly attenuating voxels. We present a binary space partition structure, *bitree*, that improves the performance of delta scattering by selecting an optimum attenuation within different regions while minimizing the interfaces to be crossed. The described *bitree* and its traversal algorithm are optimized for GPU-computing and have been implemented in the PENELOPE-based MC-GPU code. The bitree approach reduced the execution time by 88.6% for the simulation of a head CT scan with a metallic implant.

Index Terms— CT, Monte Carlo, GPU, metal artifact

1. INTRODUCTION

X-ray transport simulation codes based on the Monte Carlo (MC) method are useful tools for the study of medical imaging systems. Using accurate computational models for the x-ray source, patient anatomy, and x-ray detector, a MC code can generate virtual radiographic projection images that are equivalent, within the known limits of the implemented models, to the images produced by clinical systems. Computed tomography (CT) or tomosynthesis volumetric images can also be generated reconstructing multiple projections obtained at different angles. Images simulated with different imaging system parameters (source-to-detector distance, energy spectrum, exposure time, etc.) can be used to optimize system performance for a particular detection task and to investigate the trade off between image quality and patient dose. Some advantages of generating virtual images with MC algorithms are: availability of estimates of the patient organ doses; possibility to obtain images without electronic noise or detector blurring; exactly known geometry (truth state available for the evaluation of segmentation algorithms and target detection tasks); use of mono- or poli-energetic spectra; and ability to simulate novel imaging modalities or detector technologies that are not currently available.

State-of-the-art, general-purpose MC codes can be used to simulate medical imaging applications but the simulation of the billions of x rays required for a typical projection image may involve hundreds of hours of computation, making the simulation of the hundreds of projection images in a CT scan not feasible [1]. A promising approach to speed up MC codes is parallel execution and, in particular, the use of massively parallel architectures such as Graphics Processing Units (GPU) as the main computing devices.

In 2009 Badal and Badano introduced MC-GPU [3,4], the first GPU-based MC code for the simulation of medical x-ray images. This code can generate virtual radiographs between 27 [3] and 34 [4] times faster running in a GPU than running in a CPU core. Other GPU-based MC codes have been recently introduced [2,5]. MC-GPU can estimate radiation doses received by virtual patients undergoing diagnostic imaging and can simulate radiographic projection images and entire CT scans. While doses can be estimated in near-real time [6], CT simulations still require many hours of computation in the GPU [7]. New methods to speed up the GPU code are desirable in order to extend the use of MC-simulated CT data to applications that require a large amount of simulated images, such as iterative reconstruction algorithms and large optimization studies.

2. METHODS

2.1. MC-GPU code and delta scattering

MC-GPU [3,4,6,7] is an open-source MC x-ray transport simulation code based on the established PENELOPE subroutines [8]. MC-GPU implements the photon transport models from PENELOPE 2006 and does not include electron transport. The range of the secondary electrons generated by diagnostic x-ray sources is usually smaller

than the voxel size and a detailed electron transport would have a minimal or null effect on the tallied images and organ doses. The source code is written in C and uses the CUDA library from NVIDIA[1] [9] to access the hundreds of computing cores of the GPU. The Message Passing Interface is used to create CPU threads that automatically run the simulation in multiple GPUs in parallel.[2]

MC-GPU uses a voxelized geometry model in which the patient anatomy is described by a uniform 3D Cartesian grid. A voxelized patient model can be generated segmenting a CT scan and requires knowledge of the atomic composition (material) and density of each voxel. Some high-quality voxelized phantoms are freely available [10]. To speed up the photon tracking inside the voxelized geometry the delta scattering, or Woodcock algorithm, is used [4]. This technique is based on forcing that all voxels have the same attenuation as the most attenuating voxel. In this way, x rays can cross multiple voxels in a single step. In order to keep the results of the simulation unbiased, despite the artificially high attenuation, virtual interactions that do not modify the particle energy or direction (i.e., delta scattering) have to be introduced. The probability of having a virtual interaction depends on the ratio between the attenuation of the voxel where the interaction takes place and the selected maximum attenuation. This technique speeds up the code because it is not anymore necessary to compute the intersection of the ray trajectory with the voxel interfaces and because the number of times the voxelized geometry arrays are accessed to determine the composition of a particular voxel is reduced. Accessing the main memory is especially time-consuming in the GPU architecture and must be minimized to have good computational performance.

Delta scattering provides a remarkable speed up in simulations where all voxels have similar attenuations and when the shortest photon mean free path between interactions is longer than the voxel size. However, delta scattering is not efficient in studies in which a patient model includes high-density metal voxels such as dental fillings, prosthetic implants, shrapnel, or opaque contrast agents such as iodine or gold nanoparticles. In addition, delta scattering is not optimal in the low density regions of the geometry, especially in the air region where the probability of having an x-ray interaction is very low and there may be thousands of virtual interactions for each real one.

2.2. Binary tree structure

We propose to optimize the delta scattering algorithm using a space partition structure that separates the high, low and "medium" density regions. The objective is to select the maximum attenuation value depending on the local material composition and, as a result, minimize the number of virtual interactions. The most efficient space partitioning structure is the *kd-tree* [11], a technique commonly used in computer graphics to sort triangles describing the objects in computer games. In this work, we describe a binary space partition structure, called *bitree*, which is a particular case of kd-tree in which the given volume is recursively divided in two volumes (sub-nodes) of equal size, successively for each Cartesian axis. If the bitree is always divided in the three coordinates, giving 8 sub-nodes, the final structure is an *octree* (an octree-based MC code was developed in the past to handle a triangle-mesh geometry model [1]).

2.2.1. Bitree creation, storage and traversal

The root node of the bitree is the bounding box of the voxelized volume. Following a predefined splitting condition, the root volume is divided in two equal parts along the x axis, and then the two subnodes and their descendents are recursively subdivided successively along the y, z and x axis. The heuristic splitting condition that we have developed stops the subdivision of a node whenever one of the following conditions apply: (1) all the voxels inside the node have low density, (2) the fraction of low density voxels is smaller than a given threshold, (3) the fraction of high density voxels is larger than a threshold, (4) a maximum number of subdivisions has been reached. In case (1) the node is identified as low density. In case (2) the node has medium density. In case (3) the node has high density. In case (4) the node is identified as high density if there is at least one dens voxel or medium otherwise.

The bitree is stored in the cached GPU constant memory as a 16-bit integer array, with one element per node. The final (not split) nodes have a negative value identifying the density region at that location, while the intermediate nodes store the position in the array of their second half sub-node. The first half sub-node is located in the following position.

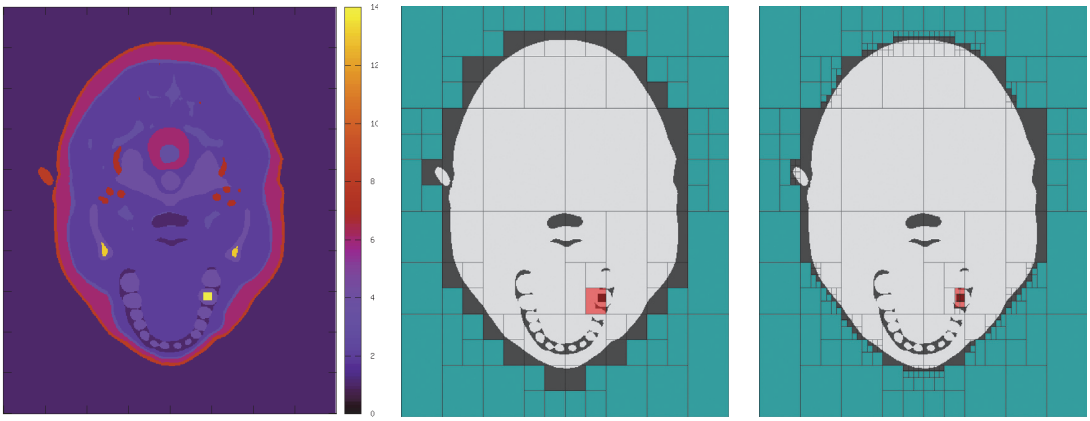
To find the density region at a given point in space we have to traverse down the bitree structure to identify the final node at that position. To reduce the memory use we do not keep a stack with the previously crossed nodes and we do not pre-calculate which nodes are located behind each node walls, as it is commonly done in CPU kd-tree traversal algorithms [11]. Our bitree traversal algorithm starts at the root node. The following sub-node (the first or second half) is determined checking the location of the point with respect to the splitting plane at the center of the node. After some iterations we find the final node and determine the density region. The coordinates of the node boundaries are obtained by keeping track of the location of the splitting planes.

2.2.2. Bitree implementation in MC-GPU

We have implemented the bitree structure in the MC-GPU code. Two relevant modifications in the algorithm were

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

[2] MC-GPU code is available at: http://code.google.com/p/mcgpu/.



(a) (b) (c)

Figure 1: (a) Visualization of the material composition in an axial plane of the head phantom. Each color represents a different material, with air being material 1 and metal material 14. (b) Bitree structure separating the air region (light blue) and the high density metal (red), generated with up to 12 subdivisions. (c) Bitree allowing 18 subdivisions.

required. First, at the beginning of each track the bitree is searched to find the density region at the initial position and the location of the node boundaries. Second, after sampling the distance to the next interaction the code computes the distance to the closest bitree boundary. If the boundary is located closer than the interaction, the particle is stopped right after the boundary and the density region of the new node is searched. Then the distance to the next interaction is re-sampled with the appropriate mean free path and the particle tracking is resumed.

2.3. Example application: metallic artifacts in CT

We used MC-GPU to simulate a head CT scan with a metallic dental implant. The input patient anatomy was the head of the high-resolution Duke phantom from the Virtual Family[3] [10] with 400x500x480, 0.5 mm voxels. A 5x5x3 mm^3 amalgam filling was inserted inside a tooth of the phantom. The composition of the amalgam was 50% mercury, 35% silver, 13% tin and 2% copper, with a density of 8 g/cm^3. The axial CT scan was acquired with 360 projections equally spaced around the head, a 120 kVp collimated cone-beam source, a source-to-detector distance of 90 cm, and a source-to-rotation axis distance of 60 cm. The detector had 400x32, 1 mm pixels. The cone-beam CT was reconstructed with a filtered back-projection-based Feldkamp algorithm. For each projection, $4 \cdot 10^9$ x rays were simulated (over 10^{12} x rays total). To study the effect of the metallic object on the execution time and on the efficiency of the bitree, the simulation was performed four times, with and without the dental filling, and with and without the bitree. The simulations were run in two workstations, using in parallel 9 NVIDIA GeForce GTX 580 and 3 GTX 480 GPUs (12 GPUs with 6048 computing cores in total).

[3] Virtual Population models are available at: www.itis.ethz.ch/services/anatomical-models/virtual-population/

3. RESULTS

Figure 1 shows an axial slice of the initial voxelized geometry and two different bitree structures separating the air around the phantom and the high density filling inside the tooth (the lines represent the node boundaries). The thresholds used to generate Fig. 1 (b) were a minimum of 25% of air and a maximum of 75% of metal with a maximum of 12 subdivisions. With these parameters the bitree structure contained 1107 nodes. Figure 1 (c) shows the equivalent bitree with 18 subdivisions. For this case the number of nodes increased 11 times to 12189 nodes.

Figure 2 shows two simulated projection image pairs with and without a metal implant. These are two of the 360 projections simulated for each head CT scan. Figure 3 shows the central slice of the reconstructed CT and the ratio between the image without and with metal. Without the bitree the execution time was 3.7 hours without metal (107.94 million x-ray/s, 12 GPUs in parallel) and 34.6 hours with metal (11.55 million x-ray/s). Using a bitree with thresholds at 75% and 18 subdivisions (237 nodes) the simulation time with metal was reduced 88.6% to 4.0 hours (100.91 million x-ray/s). We confirmed that the computed pixel values and organ doses with and without the bitree were equivalent, within the MC statistical uncertainty.

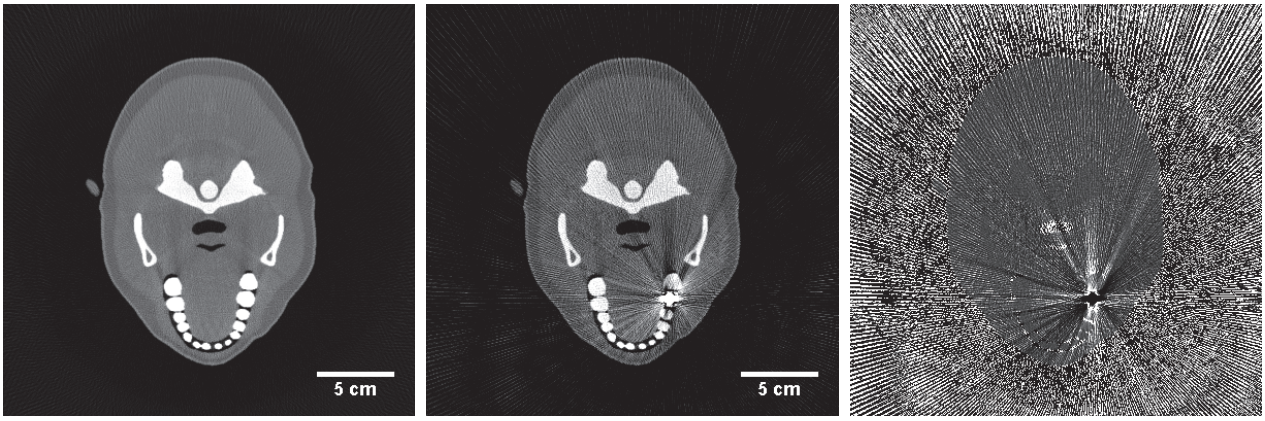
4. DISCUSSION AND CONCLUSIONS

We introduced an algorithm that can prevent the loss of computational performance of MC codes that use delta scattering in the presence of highly attenuating materials, and applied it to the simulation of diagnostic images with MC-GPU. The described algorithm is general enough to be used in other MC codes, including accelerated MC codes for radiotherapy treatment planning [2,12]. The presented simulation results show that, when the bitree structure is not used, the simulation with a small metal volume required 9.3



(a) (b)

Figure 2: Sample simulated radiographic images: (a) lateral projections without (above) and with the metal implant (below); (b) frontal projections without (above) and with the metal implant (below). The detector size is 40.0x3.2 cm^2.



(a) (b) (c)

Figure 3: Reconstructed CT scans using the 360 simulated projections: (a) no metal present; (b) metal tooth filling present; (c) ratio of the images (a) and (b) showing the characteristic metal artifacts in the reconstruction.

times more runtime than the simulation without metal. Using the optimized bitree structure to isolate the metal insert it was possible to restore the computational performance and reach a similar simulation time with and without the metal, with just a 7.0% increase in execution time due to the overhead of the bitree traversal. The bitree does not change the particle transport physics nor the accuracy of the results.

The characteristic metal artifacts seen in clinical CT images can be clearly observed in the simulated data [Fig. 3 (c)]. Accurate MC simulations allow the study of artifacts caused by increased scatter from the metal object and beam hardening effects, which are physical processes that would not be included in a simple CT simulation code based on ray-tracing.

REFERENCES

[1] A. Badal, I. Kyprianou, Diem Phuc Banh, A. Badano, and J. Sempau, "penMesh—Monte Carlo Radiation Transport Simulation in a Triangle Mesh Geometry," IEEE Trans. Med. Imaging 28, pp. 1894-1901, 2009

[2] S. Hissoiny, B. Ozell, H. Bouchard, and P. Després, "GPUMCD: A new GPU-oriented Monte Carlo dose calculation platform," Med. Phys. 38, pp. 754-765, 2011

[3] A. Badal and A. Badano, "Accelerating Monte Carlo simulations of photon transport in a voxelized geometry using a massively parallel GPU," Med. Phys. 36, pp. 4878-4880, 2009

[4] Wen-mei W. Hwu, *GPU Computing Gems (Emerald Edition)*, Chapter 50, Morgan Kaufmann/Elsevier, 2011

[5] G. Pratx and L. Xing, "GPU computing in medical physics: A review," Med. Phys. 38, pp. 2685-2698, 2011

[6] A Badal and A Badano, "Fast and Accurate Estimation of Organ Doses in Medical Imaging Using a GPU-Accelerated Monte Carlo Simulation Code," Med. Phys. 38, p. 3411, 2011

[7] A. Badal, I. Kyprianou, D. Sharma, and A. Badano, Fast cardiac CT simulation using a graphics processing unit-accelerated Monte Carlo code, Proc. SPIE 7622, pp. 762231, 2010

[8] 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, 2006

[9] NVIDIA Corporation, "CUDA programming guide," Technical Report available at www.nvidia.com/cuda, 2011

[10] A. Christ, W. Kainz *et al*., "The Virtual Family–Development of surface-based anatomical models of two adults and two children for dosimetric simulations", Phys. Med. Biol. 55, N23, 2010 (www.itis.ethz.ch/services/anatomical-models/virtual-population/)

[11] T. Foley and J. Sugerman, "KD-tree acceleration structures for a GPU raytracer," Proc. HWWS'05, pp. 15–22, 2005

[12] 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, pp. 2263–2291, 2000