A fast three-dimensional gamma evaluation using a GPU utilizing texture memory for on-the-fly interpolations

Lucas C. G. G. Persoon, Mark Podesta, Wouter J. C. van Elmpt, Sebastiaan M. J. J. G. Nijsten, and Frank Verhaegen[a)]

Department of Radiation Oncology (MAASTRO), GROW—School for Oncology and Developmental Biology, Maastricht University Medical Centre, Maastricht, The Netherlands

(Received 18 November 2010; revised May 2011; accepted for publication May 2011; published 20 June 2011)

Purpose: A widely accepted method to quantify differences in dose distributions is the gamma (γ) evaluation. Currently, almost all γ implementations utilize the central processing unit (CPU). Recently, the graphics processing unit (GPU) has become a powerful platform for specific computing tasks. In this study, we describe the implementation of a 3D γ evaluation using a GPU to improve calculation time.
Methods: The γ evaluation algorithm was implemented on an NVIDIA Tesla C2050 GPU using the compute unified device architecture (CUDA). First, several cubic virtual phantoms were simulated. These phantoms were tested with varying dose cube sizes and set-ups, introducing artificial dose differences. Second, to show applicability in clinical practice, five patient cases have been evaluated using the 3D dose distribution from a treatment planning system as the reference and the delivered dose determined during treatment as the comparison. A calculation time comparison between the CPU and GPU was made with varying thread-block sizes including the option of using texture or global memory.
Results: A GPU over CPU speed-up of 66 ± 12 was achieved for the virtual phantoms. For the patient cases, a speed-up of 57 ± 15 using the GPU was obtained. A thread-block size of 16×16 performed best in all cases. The use of texture memory improved the total calculation time, especially when interpolation was applied. Differences between the CPU and GPU γs were negligible.
Conclusions: The GPU and its features, such as texture memory, decreased the calculation time for γ evaluations considerably without loss of accuracy. © *2011 American Association of Physicists in Medicine*. [DOI: 10.1118/1.3595114]

Key words: radiotherapy, dose verification, gamma evaluation, 3D dosimetry, graphics processing unit (GPU), calculation speed, texture memory, interpolation

I. INTRODUCTION

In radiotherapy, dose verification prior to and during treatment has become important due to complex dose delivery techniques. To compare measured to planned dose distributions, the widely accepted gamma (γ) evaluation method is frequently used.[1,2] The γ evaluation is a method to quantify differences in dose distributions. A major disadvantage is that the method is slow.

Different γ implementations have been proposed in the literature[3–6] with the fastest calculation times reported of 0.6 s for cube sizes typically used in clinic. Predominantly, the computer's central processing unit (CPU) is used for the implementation. Recently, the graphics processing unit (GPU) has developed into a specialized device for specific computing tasks. The GPU is a powerful, massively parallel programmable architecture,[7] which can be used to decrease the calculation time for many applications. In radiotherapy, the GPU has already been used in a wide range of applications (e.g., ray-tracing, dose calculation, and CT reconstruction techniques)[8–14] reporting speed-up ratios from 6 to 908.

In this study, we implement a 3D γ evaluation on a GPU to reduce calculation time.

II. METHODS AND MATERIALS

II.A. Algorithm

The γ evaluation[1,2] combines a dose-difference and distance-to-agreement criterion for the comparison of dose distributions. A comparison dose ($D_c(r_c)$) is compared with a reference dose ($D_r(r_r)$). The dose difference is defined by $\delta(r_c,r_r) = D_c(r_c) - Dr(r_r)$ and the spatial distance by $r(r_c,r_r) = |r_c - r_r|$. For all spatial grid point combinations, a function $\Gamma(r_c,r_r)$ is calculated by

$$\Gamma(r_c, r_r) = \sqrt{((r^2(r_c, r_r)/\Delta d^2) + (\delta^2(r_c, r_r)/\Delta D^2))} \quad (1)$$

ΔD and Δd are the acceptance criteria. ΔD being a percentage of the maximum reference dose and Δd being the distance to agreement. Then, for all points r within a spatial search volume $\dot{\mathrm{U}}$, the γ is calculated by

$$\gamma(r) = \min\{\Gamma(r_c, r_r)\} \forall \{r_c \epsilon \dot{\mathrm{U}}\} \quad (2)$$

In our method, we apply a spherical volume $\dot{\mathrm{U}}$ of $1 \times 1 \times 1$ cm^3 centered around r_r according to Wendling *et al.*,[4] and their early stopping criterion is applied, which calculates $\Gamma(r_c,r_r)$ starting from the smallest $r^2(r_c,r_r)$ to the largest

TABLE I. Patient characteristics.

Case	Reference dose cube size (voxels)	Zero padded dose cube size (voxels)	Treatment site	Treatment plan type	No. beams
A	$166 \times 124 \times 88$	$192 \times 128 \times 88$	Head and neck	3D CRT	4
B	$139 \times 131 \times 88$	$160 \times 160 \times 88$	Lung	SBRT	10
C	$148 \times 141 \times 79$	$160 \times 160 \times 79$	Lung	3D CRT	4
D	$164 \times 134 \times 88$	$192 \times 160 \times 88$	Head and neck	IMRT	7
E	$141 \times 133 \times 78$	$160 \times 160 \times 78$	Head and neck	IMRT	7

possible $r^2(r_c,r_r)$ and stops when $\sqrt{(r^2(r_c,r_r)/\Delta d^2)}$ exceeds the smallest found $\Gamma(r_c,r_r)$.

For this study, global dose differences with respect to the maximum D_r with criteria of $\Delta D = 3\%$ and $\Delta d = 3$ mm were used.

II.B. GPU computing

An NVIDIA Tesla C2050 GPU (NVIDIA, Santa Clara, CA, USA) was used with double precision compute capability 2.0, 3 GB GDDR5 integrated memory, and NVIDIA's Compute Unified Device Architecture (CUDA) version 3.2 of the SDK. CUDA uses a parallel programming model and extends the C programming language by allowing the definition of kernels that are executed N times in parallel by N CUDA threads on the GPU.[7] To execute kernels, there are two important parameters to define: the thread-block size, a vector of three elements (x,y,z), and the grid-block size, which is a vector of two elements (x,y). The thread-block size cannot exceed a limit of 1024 threads for the TESLA C2050 and is hardware-specific. The thread-block size is the number of simultaneous launched threads. The total number of threads to be executed is the same as the product of the thread-block size and grid-block size. In our implementation, we executed for each voxel a thread keeping the product of the thread and grid block equal to the number of voxels. Memory access was conducted either using texture memory with the hardware built-in 3D texture functions to fetch voxels and immediate interpolate or by using global memory and an equivalent on-the-fly interpolation.[7] Texture memory is special hardware built on-top of the global memory to access data. CUDA supports a subset of the texturing hardware capabilities normally used for graphics and includes caching and interpolation (e.g., trilinear).

II.C. CPU computing

The CPU simulations were executed on an Intel Xeon X5550 2.66 GHz quad-core CPU with hyperthreading. CPU and GPU calculations were implemented in C/C++ using Microsoft Visual Studio 2008 (Microsoft, Redmond, WA, USA). For benchmarking, single core performance was investigated, while multicore was also implemented using the parallel programming API OpenMP. Furthermore, the on-the-fly interpolation was implemented with an equivalent algorithm as implemented by NVIDIA.[7]

II.D. Benchmarking and examples

Several cubic virtual phantoms were simulated.[4] The virtual phantom test is a setup with two dose cubes: a centered one and a shifted one with magnitude changes. These phantoms were tested with varying cube sizes with a voxel size of $0.1 \times 0.1 \times 0.1$ cm^3. No interpolation of the comparison dose cubes (D_c) was applied. The dose of the centered volume was set to a value of 100 cGy while outside the central volume it was set to 0 cGy. For benchmarking, three artificial errors were introduced in D_c.

Next, to show applicability in clinical practice, five patient cases (Table I) were evaluated using the 3D dose distribution from a treatment planning system as a reference D_r and the delivered dose measured during treatment. The measured dose cube D_c was on-the-fly interpolated to a grid size three times finer than the original. The D_r and D_c are, when necessary, zero padded to a multiple of 32 pixels in a slice.

To investigate the sensitivity of different thread-block sizes and the performance of texture memory, varying thread-block sizes were tested (i.e., $n \times n$ with $n = 8$, 16, 32) and all calculations were executed using global memory and texture memory. The accuracy between the CPU and GPU implementations were evaluated by comparing several metrics.

III. RESULTS

Table II shows the GPU over CPU speed-up for the virtual phantoms. The speed-up ranged from 49 to 82 times. Furthermore, it can be seen that the thread-block size of 16×16 is optimal for all phantoms and shows the highest speed-up of 66 ± 12. The calculation time for the virtual phantoms using texture memory management decreased on average by 5%. No differences in accuracy between CPU and GPU results exceeding $2{\cdot}10^{-7}$% were observed. For the patient cases, the speed-up ranged from 28 to 82 times. It was observed that the thread-block size of 16×16 was also optimal with a speed-up of 57 ± 15. Using textures for the interpolation added an average performance gain of 30% in comparison with the GPU implementation using global memory and the on-the-fly interpolation. No differences exceeding $9.1{\cdot}10^{-4}$% were observed.

IV. DISCUSSION

In this work, we showed that the use of GPU architecture can speed-up the calculation of 3D γ evaluations considerably compared to CPU calculations. In our implementation, the interpolation step is executed on the GPU using texture memory. This is one of the major advantages of using GPUs to accelerate the calculation of 3D γs. The benefit of using GPUs for the γ evaluation will increase further when new

TABLE II. Calculation time and speed-ups for the different virtual phantoms and patient cases. The introduced error is a shift and dose change in comparison with the reference dose. The different errors are given in parentheses for the 128^3 phantoms. The shift is introduced in each direction (x,y,z). The calculation time is based on an average of five runs of the application. The standard deviation for the GPU kernel execution time was on average 2.5%, while for the CPU the average standard deviation was 1.5%. For the patient cases and 384^3 phantom the speed-up without using the texture memory is given in parentheses.

	CPU	GPU			Speed-up		
		Thread-block size			Thread-block size		
	Average time (s)	8×8 Average time (s)	16×16 Average time (s)	32×32 Average time (s)	8×8	16×16	32×32
Phantom 128^3, 5 cm^3 centered dose volume							
I (0.1 cm, 1%)	0.24	0.005	0.005	0.005	**49**	**49**	**49**
II (0.2 cm, 3%)	0.80	0.140	0.010	0.011	**50**	**70**	**63**
III (0.3 cm, 5%)	1.52	0.023	0.017	0.018	**65**	**89**	**84**
Phantom 192^3, 7.5 cm^3 centered dose volume							
I	0.81	0.015	0.014	0.015	**54**	**58**	**54**
II	1.95	0.042	0.037	0.039	**46**	**53**	**50**
III	7.99	0.133	0.116	0.120	**60**	**69**	**67**
Phantom 256^3, 10 cm^3 centered dose volume							
I	1.92	0.034	0.032	0.033	**56**	**59**	**58**
II	5.74	0.085	0.084	0.090	**67**	**68**	**63**
III	16.55	0.210	0.209	0.220	**79**	**79**	**75**
Phantom 384^3, 15 cm^3 centered dose volume							
I	6.59	0.122	0.120	0.135	**54 (52)**	**55 (53)**	**48 (48)**
II	17.07	0.257	0.241	0.277	**66 (61)**	**71 (67)**	**61 (58)**
III	59.94	0.800	0.730	0.761	**74 (62)**	**82 (72)**	**78 (63)**
Case							
A	0.36	0.009	0.009	0.011	**41 (28)**	**41 (30)**	**33 (28)**
B	0.82	0.011	0.010	0.012	**74 (51)**	**82 (63)**	**68 (41)**
C	0.59	0.012	0.010	0.015	**49 (46)**	**59 (50)**	**40 (35)**
D	0.67	0.018	0.015	0.024	**37 (29)**	**44 (34)**	**28 (24)**
E	0.70	0.015	0.013	0.021	**47 (33)**	**54 (39)**	**33 (25)**

graphics hardware is introduced because GPUs evolve much faster than CPUs.[7]

There were a few considerations when choosing from the methods of speed increase available on the GPU architecture. The data volume sizes and search box sizes used in γ calculations exclude the option of effectively using the GPU shared memory to its potential. Coalesced global memory access is desirable and would offer the next best speed increase but much care must be taken to ensure a simple organized access pattern of the stored array. While an effective manipulate/copy/execute structure may be possible, it would be difficult to maintain as both the input volume resolution as well as the γ criterion vary from run to run. We decided to use texture fetching as it offers a good performance on memory reads that do not follow the access patterns required by global memory and has a higher bandwidth if there is locality to concurrent access patterns. This is suited to the style of a γ search box.[7] The memory access pattern is less linear and more local due to the specific ordered geographical search box. On the FERMI architecture, there is also the option of using the Level 1 (L1) or Level 2 (L2) cache which both use error checking correction (ECC) in contrast to texture memory, which is non-ECC. Using texture memory has the benefit of hardware accelerated trilinear interpolation, which is processed efficiently on a GPU, reducing the need to develop a fast interpolation or divided simplex function. This means a comparatively fast implementation of interpolation using texture memory is possible while keeping decent performance. We observed that using the texture memory is beneficial for the overall calculation time, on average 5% faster for the virtual phantoms and 30% faster for the patient cases. For the patient cases, the advantage of using texture memory versus global memory was higher because interpolation was executed on-the-fly.

A thread-block size of 16×16 resulted in an optimal calculation time for all cases. NVIDIA provides several optimization tools; the *visual profiler* and the *occupancy calculator*. Occupancy can be seen as the number of concurrent threads in use and is a good measure of GPU efficiency. It was found that the thread-block size of 8×8 has the lowest occupancy of 33%. This is because we use 30 registers per thread. The other two tested thread-block sizes have an occupancy of 66%. The best performance of 16×16 can be explained because the number of thread-blocks per multiprocessor is six in comparison with only one thread-block for 32×32 thread-block size. The 16×16 thread-block size and the six thread-blocks per multiprocessor optimizes the number of warps (concurrently launched threads) processed on a multiprocessor and is, in this case, better able to fill the latency around expensive operations resulting in better overall performance.

The software and source-code using a fixed thread-block size of 16×16 is freely available for non-commercial research

use. On the website[15], both a separate C/C++ program as well as a MATLAB mex-file are downloadable. All compiled packages support only the windows platform.

V. CONCLUSIONS

In this work, fast *y* evaluation was implemented on GPU architecture. Comparison of large 3D dose matrices can be done in subsecond calculation times, with no loss of accuracy compared to CPU implementations. Further speed-ups may be realized by harnessing clusters of GPUs.

[a]Author to whom correspondence should be addressed. Electronic mail: frank.verhaegen@maastro.nl; Telephone: +31-88-4455666; Fax: +31-88-4455667.

[1]D. A. Low, W. B. Harms, S. Mutic, and J. A. Purdy, "A technique for the quantitative evaluation of dose distributions," Med. Phys. **25**, 656–661 (1998).
[2]D. A. Low and J. F. Dempsey, "Evaluation of the gamma dose distribution comparison method," Med. Phys. **30**, 2455–2464 (2003).
[3]T. Ju, T. Simpson, J. O. Deasy, and D. A. Low, "Geometric interpretation of the gamma dose distribution comparison technique: interpolation-free calculation," Med. Phys. **35**, 879–887 (2008).
[4]M. Wendling, L. J. Zijp, L. N. McDermott, E. J. Smit, J. J. Sonke, B. J. Mijnheer, and M. van Herk, "A fast algorithm for gamma evaluation in 3D," Med. Phys. **34**, 1647–1654 (2007).
[5]J. Yuan and W. Chen, "A gamma dose distribution evaluation technique using the k-d tree for nearest neighbor searching," Med. Phys. **37**, 4868–4873 (2010).
[6]X. Gu, X. Jia, and S. B. Jiang, "GPU-based fast gamma index calculation," Phys. Med. Biol. **56**, 1431–1441 (2011).
[7]NVIDIA, "NVIDIA CUDA Programming Guide v3.2," NVIDIA, 2010.
[8]M. de Greef, J. Crezee, J. C. van Eijk, R. Pool, and A. Bel, "Accelerated ray tracing for radiotherapy dose calculations on a GPU," Med. Phys. **36**, 4095–4102 (2009).
[9]F. Xu, W. Xu, M. Jones, B. Keszthelyi, J. Sedat, D. Agard, and K. Mueller, "On the efficiency of iterative ordered subset reconstruction algorithms for acceleration on GPUs," Comput. Meth. Progr. Biomed. **98**, 261–270 (2009).
[10]R. Jacques, R. Taylor, J. Wong, and T. McNutt, "Towards real-time radiation therapy: GPU accelerated superposition/convolution," Comput. Meth. Progr. Biomed. **98**, 285–292 (2009).
[11]C. Men, X. Jia, and S. B. Jiang, "GPU-based ultra-fast direct aperture optimization for online adaptive radiation therapy," Phys. Med. Biol. **55**, 4309–4319 (2010).
[12]S. Hissoiny, B. Ozell, and P. Després, "A convolution-superposition dose calculation engine for GPUs," Med. Phys. **37**, 1029–1037 (2010).
[13]S. S. Samant, J. Xia, P. Muyan-Ozcelik, and J. D. Owens, "High performance computing for deformable image registration: towards a new paradigm in adaptive radiotherapy," Med. Phys. **35**, 3546–3553 (2008).
[14]P. Després, J. Rinkel, B. Hasegawa, and S. Prevrhal, "Stream processors: A new platform for Monte Carlo calculations," in *Third McGill Workshop on Monte Carlo Techniques in Radiotherapy Delivery and Verification*, edited by F. Verhaegen, J. Phys.: Conf. Ser. **102**, 012007 (2008).
[15] http://www.MISTIR.info