NUMERICALLY RESOLVED RADIATION VIEW FACTORS VIA MULTI-GPU ACCELERATED RAY TRACING

Katie E. Richmond, Asher J. Hancock, Shervin Sammak and Matthew M. Barry*
*Author for correspondence
Department of Mechanical Engineering and Materials Science,
University of Pittsburgh,
Pittsburgh, Pennsylvania
United States of America,
E-mail: matthew.michael.barry@pitt.edu

ABSTRACT

A robust computational framework is presented to directly solve for the radiation view factors (F_{ij}) of participatory surfaces within complex three-dimensional geometries. This framework exploits the embarrassingly-parallel nature of the formulation and solution of F_{ij} through a multiple graphics processing units (GPU)-accelerated ray tracing scheme. The presented computational methodology was developed in Java and incorporated Aparapi for OpenCL compatibility. The surfaces of the geometries of interest are constructed via the creation of stereolithography (STL) files, which represent surfaces as tessellations. The shadow effect, where cast rays are obstructed by non-participatory surfaces, is handled via the Möller-Trumbore (MT) ray-triangle intersection algorithm. To ensure generality and robustness, a self-intersection algorithm is implemented for both planar and non-planar surfaces via the MT algorithm with back-face culling enabled. Validation of the algorithm was performed for a variety of three-dimensional geometries. The proposed multi-GPU framework was benchmarked to a conventional computer processing unit (CPU)-based version of the code and exhibited substantial decreases in computational time. Results indicate that near-linear speed-up is achievable with increasing numbers of GPUs. Additionally, a converging solution is obtained with increasing tessellation and GPU count, indicating no perceivable discrepancy in solutions in comparison to CPU and single-GPU based solutions.

INTRODUCTION

Ray tracing (RT) refers to the trajectory tracking of a path, as initiated by an emitter, and is an indispensable numerical tool in fields as disparate as animation and aerospace engineering. Recently, RT methodologies have attracted the interests of researchers and engineers as a means to calculate the radiation view factor (F_{ij}). The radiation view factor is a geometrical parameter that describes the proportion of radiation exchanged between two bodies. It is an important parameter when quantifying the radiative heat transfer rate, and many works exist that have attempted to calculate it in a tractable manner. For example, Walker et al. [1] presented a Monte Carlo governed ray-tracing method to estimate F_{ij} within an operational fiber drawing furnace. Yet, it was noted that large computational runtimes hampered its feasibility as an analysis tool. Likewise, Vilchez et al. [2] constructed an RT algorithm to calculate F_{ij} between a target and a spherical fireball with an obstructive surface in-between. While decent results were reported, this work was not robust enough to encompass other analytical cases.

Other uses of RT for calculating F_{ij} are seen in the environmental engineering sector. Wang et al. [3] developed a Monte Carlo RT procedure to determine F_{ij} and analyzed the transmittance of radiative thermal energy in urban environments, namely the interaction between trees and ambient buildings. Sönmez et al. [4] proposed a hybrid RT-Fibonacci lattice technique to estimate the sky view factor (SVF) via LiDAR data in a simulated

NOMENCLATURE

A	[m^2]	Surface area or Point
F_{ij}	[-]	Radiation view factor
H	[mm]	Height
K	[-]	Number of GPUs
$\overrightarrow{n}$	[-]	Unit normal vector
Q_i	[W]	Radiation heat transfer rate
$\overrightarrow{R}$	[-]	Radiative ray vector
t	[mm]	Thickness
T	[K]	Temperature
W	[mm]	Width

Special characters

ε	[-]	Emissivity
σ	$5.670\text{x}10^{-8}$ [$Wm^{-2}K^{-4}$]	Stefan-Boltzmann constant
θ	[degrees]	Polar angle
ϕ	Packing density	

Subscripts

i	Emitting surface
int	Interconnector
j	Receiving surface

Acronyms

CPU	Computer processing unit
GPU	Graphics processing unit
MT	Möller-Trumbore
RT	Ray-tracing
STL	Stereolithography
TEG	Thermoelectric generator

TU Delft campus. While fast results were achieved in this robust method, the authors note an average error of 3.82%, which may be too large in some circumstances. As demonstrated, there is an intrinsic tradeoff between computational runtime and numerical accuracy when utilizing RT to calculate the view factor. However, for RT methods to feasibly determine F_{ij}, both runtime and accuracy tradeoffs needs to be rectified.

All methods herein mentioned have been executed on the computer processing unit (CPU), but the current rise of the graphics processing unit (GPU) in scientific computing has shown promise in providing substantially better runtimes in certain problems, such as RT. Given the state of current RT and F_{ij} methodologies, it is clear that a fast, robust, and accurate numerical methodology is needed. In this paper, the work of Hancock et al. [5] was extended to capitalize on the embarrassingly parallel view factor problem and implement the method across multiple GPUs to achieve further runtime gains. The method proposed herein considers two analytical test cases – aligned parallel rectangles, and concentric spheres – to validate the method. Then, an analysis of the radiative heat transfer within a single-junction thermoelectric generator is presented to investigate the runtime improvements when sub-dividing the problem across two GPUs (in contrast to executing a single device).

BASIC PROBLEM SETUP

The radiation view factor is a geometrical parameter that represents the proportion of radiation exchanged between two surfaces, A_i and A_j, and is related to the radiation heat transfer rate, Q_i, between the two participating surfaces by

$$Q_i = \varepsilon\sigma A_i F_{ij}(T_i^4 - T_j^4) \tag{1}$$

where ε represents the emitting surface's emissivity, σ represents the Stefan-Boltzmann constant, and T represents the temperature of the emitting and receiving surfaces, respectively. F_{ij} is calculated as

$$F_{ij} = \frac{1}{A_i}\int_{A_i}\int_{A_j}\frac{\cos(\theta_i)\cos(\theta_j)}{\pi\|\overrightarrow{R}_{ij}\|^2}\,dA_i\,dA_j \tag{2}$$

where the numeric resolution of F_{ij} between two participating surfaces is accomplished through discretizing the total area of both surfaces into N_i and N_j triangular differential areas. The radiative ray vector, $\overrightarrow{R}_{ij}$, is cast from the centroid of an emitting tessellation, dA_i, to the centroid of the receiving tessellation, dA_j, and its magnitude is calculated by the Euclidean norm of the difference in centroidal coordinates. The polar angles, θ_i and θ_j, between the tessellations' unit normal vectors, $\overrightarrow{n}_i$ and $\overrightarrow{n}_j$, are calculated as

$$\theta_{i,j} = \cos^{-1}\left(\frac{\overrightarrow{n}_{i,j}\cdot\overrightarrow{R_{ij}}}{\|\overrightarrow{n}_{i,j}\|\|\overrightarrow{R_{ij}}\|}\right). \tag{3}$$

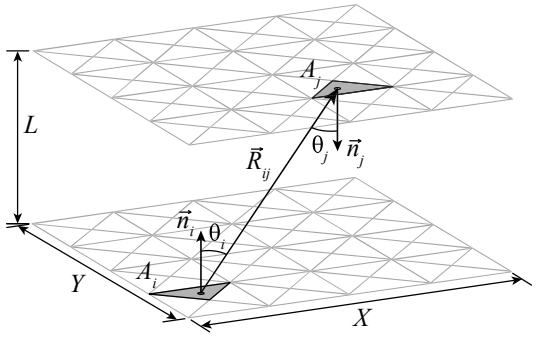

Figure 1. Depiction of the relevant variables within the calculation of F_{ij} between two parallel plates.

To depict the problem, Figure 1 shows the relevant variables used to calculate F_{ij} between two aligned parallel rectangles. In this work, the geometrical parameterization was accomplished via converting the models obtained from a modified mesh generator [6] into stereolithography (STL) files. In this manner, the necessary geometrical properties, such as edge lengths and centroidal locations, are easily determined through simple arithmetic operations. Additionally, STL files are readily exportable from most computer aided design software, making this methodology robust to a variety of use cases.

By discretizing the domain, the continuous formulation of Equation 2 is now computed as a summation across the discrete domain by accounting for each differential view factor, dF_{ij}, by

$$F_{ij} = \frac{1}{A_i}\sum_{i=1}^{N_i}\sum_{j=1}^{N_j}\frac{\cos(\theta_i)\cos(\theta_j)}{\pi\|\overrightarrow{R}_{ij}\|^2}\,dA_i\,dA_j = \frac{1}{A_i}\sum_{i=1}^{N_i}\sum_{j=1}^{N_j}dF_{ij}. \tag{4}$$

RAY-TRIANGLE INTERSECTION

In the event that there exists an obstructive geometry in-between the two surfaces undergoing radiative transfer, it becomes necessary to determine whether the cast ray is blocked before reaching its target; this phenomenon, known as the shadow effect, is an important facet to resolve when calculating F_{ij} between two bodies. Otherwise, said view factor will portray a value larger than its actual. An example of the shadow effect is demonstrated in the thermoelectric generator (TEG) model of Figure 4, as depicted by the dashed portion of the cast ray which intersects with the non-participatory inner geometry of the TEG. To rectify the shadow effect in a felicitous manner with the previously described problem setup, the renowned Möller–Trumbore (MT) ray-triangle intersection algorithm was utilized. The MT algorithm is an efficient intersection algorithm that alleviates the precomputation of the plane equation intrinsic of similar methods [8].

In this methodology, all the tessellations of obstructive surfaces in a given problem are labeled in separate STL files from the emitting and receiving surfaces. Then, during the RT procedure, every cast ray is computed against every triangle within the blocking geometry. If no intersection is detected, the emitted

ray reaches its receiver tessellation; otherwise, an intersection exists. In practical terms, if a cast ray is intercepted by a non-participatory surface, the ray will not contribute to the overall view factor calculation of Equation 4.

However, in some non-convex surfaces, it is possible for the emitted ray of one differential area to "self-intersect" with a different region of the emitter's body; an example of such a situation is apparent in Figure 3 where a ray could be intercepted by the opposing side of the inner sphere before reaching its target. Therefore, a more rigorous treatment of "blocking" surfaces within this RT procedure is required. The approach taken in this work was to consider all differential areas in the emitter not currently participating in the RT procedure and logically consider them as obstructive. This way, any "self-intersecting" geometries will be considered and the correct F_{ij} will be returned. Furthermore, to decrease computation time and increase fidelity when considering participation between curved surfaces, back-face culling was implemented such that erroneous intersections could be minimized [5].

GPU-ACCELERATED COMPUTING

GPUs are processing units that consist of hundreds of cores, which can operate thousands of threads, to parallelize computation and drastically improve computational performance. With their unique architecture, GPUs are best suited for high-throughput applications, making them attractive solutions for algebraically intensive procedures such as RT. In this procedure, both F_{ij} and the MT algorithm are independently performed on each element in the computational domain, i.e., each calculation does not depend on the state of its neighboring elements. This embarrassing parallel scheme is ideal for GPU-acceleration and is a solution to alleviate the cumbersome runtimes intrinsic of traditionally calculating the view factor.

In practice, a suite of Java classes were created to provide the numerical framework for analyzing different geometrical setups and harnessing GPU-acceleration for calculating F_{ij}. Aparapi, the open-source API used for converting the Java byte code to a GPU kernel during runtime, was heavily utilized [9]. However, it is noted that since not all programmatical operations were available for calling on the GPU, problem logistics, such as STL file reading and geometry precomputation, were completed on the CPU to alleviate any parallel execution errors. The flowchart represented in Figure 2 demonstrates the execution order of this view factor methodology and the sections of the program computed on different hardware are shaded light gray.

Previously, Hancock et al. [5] utilized a single GPU to calculate the view factor in a variety of setups, but the authors noted that even with GPU-accelerated programming, some large computational domains or complex surfaces remained infeasible to calculate. Therefore, this work aims to address some of the limitations in the previous study by extending the numerical methodology to work across multiple devices. In this manner, it was hypothesized that runtimes could be decreased by simultaneously utilizing multiple GPUs. The general framework involves creating different sets of kernels at runtime, which operate on specific

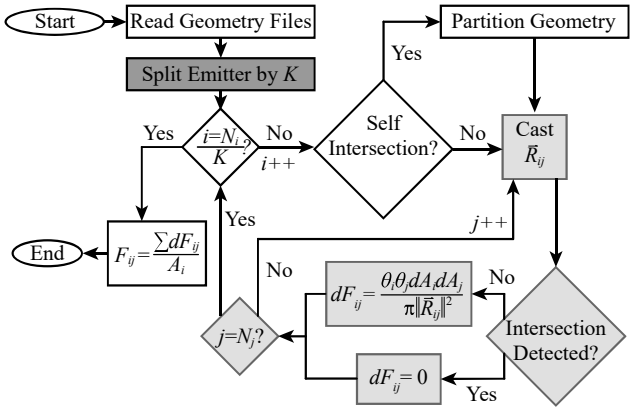

Figure 2. Flow chart for computational algorithm.

locations of the surface under consideration. In order to preserve Equation 4, the problem was subdivided such that the emitting surface was broken up into equal components corresponding to the number of GPUs available, denoted as K in Figure 2. Therefore, each GPU receives $1/K$ as many emitting tessellations, and each casts a proportionally fewer number of rays during the RT procedure.

RESULTS AND DISCUSSION

Within the following section, two separate validation cases are presented where the numerically calculated F_{ij} values obtained on a single, and on dual-GPUs, are compared to analytic solutions for varying tessellation count (represented as number of rays cast). Thereafter, the algorithm is demonstrated on a single-junction thermoelectric generator. The algorithm was implemented in IntelliJ IDEA, a Java IDE, and was executed on an Intel i7-8700K CPU with dual Nvidia GTX 1080Ti GPUs.

Validation: Aligned Parallel Rectangles

To validate the proposed F_{ij} formulation, the program was subjected to various geometrical problems with analytical solutions from [7]. The canonical example of aligned parallel rectangles was firstly considered. The geometrical setup for this test case is depicted in Figure 1 where X/L and Y/L equated to unity. To verify the subdivision of the RT procedure across multiple GPUs, an absolute difference of the view factor between single- and double-GPU executions are presented in Table 1. As seen, the differences are on the order of double-precision, and for an increasing number of cast rays, the calculated view factor began converging to the analytical solution.

Validation: Concentric Spheres

The final validation case presented in this work is the case of concentric spheres [10]. This geometrical setup, as seen in Figure 3, was chosen since the numerical resolution of F_{ij} requires use of the MT algorithm, back-face culling, and consideration of possible self-intersection.

The analytic solution yields a value of unity for F_{ij} via the summation rule, and the numerically determined values for varying tessellation count are reported in Table 2. As before,

Table 1. F_{ij} values for aligned parallel rectangles with X/L=1 and Y/L=1 with increasing mesh density. Absolute differences are between single- and multi-GPU implementations. Percent error is between multi-GPU and analytical solutions (F_{ij} = 0.199824895698387).

Rays Cast	F_{ij} (1 GPU)	F_{ij} (2 GPUs)	Abs. Diff	Error [%]
6.87e+10	0.199825202	0.199825202	2.00E-15	1.53e-4
2.75e+11	0.199825049	0.199825049	2.00e-15	7.67e-5
1.10e+12	0.199824972	0.199824972	6.00e-15	3.83e-5
4.40e+12	0.199824934	0.199824934	7.99e-15	1.92e-5
1.76e+13	0.199824915	0.199824915	3.40e-14	9.58e-6

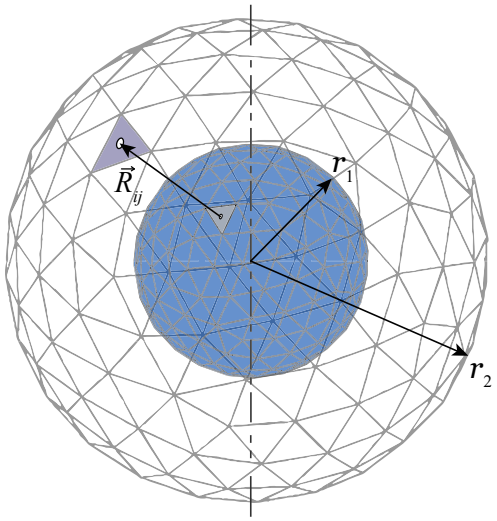

Figure 3. STLs representing concentric spheres with the inner sphere having a radius r_1 and the outer sphere has a radius r_2.

an absolute difference of the view factor between single- and dual-GPU executions are presented to showcase this solution algorithm's consistency. Likewise, monotonic convergence of the numerical view factor to the analytical was observed. This further indicates robustness of the MT ray-triangle intersection algorithm and back-face culling methods when applied to curved surfaces.

Table 2. F_{ij} values of concentric spheres for r_1/r_2=0.5 with increasing mesh density. Absolute differences are between single- and multi-GPU implementations. Percent error is between the multi-GPU's output and the analytical solution (F_{ij}=1).

Rays Cast	F_{ij} (1 GPU)	F_{ij} (2 GPUs)	Abs. Diff	Error [%]
2.00e+7	1.001156764	1.001156764	0	1.16e-1
1.37e+8	1.000421924	1.000421924	0	4.22e-2
4.19e+9	1.000069948	1.000069948	0	6.99e-3
7.40e+9	1.000063232	1.000063232	2.02e-14	6.32e-3

Test: Single-Junction Thermoelectric Generator

Once the proposed methodology was validated across multiple GPUs, an analysis into decreases in computation time for ascertaining the view factor across thermoelectric generators (TEGs) was conducted. TEGs are steady-state power-generation devices that develop a voltage potential as a result of an applied temperature gradient across a unicouple. The most basic unicouple is comprised of two semiconductor legs connected by metallic interconnectors, as shown in Figure 4. Analysis of the radiative transfer across the TEG's hot- and cold-sides is imperative in determining device performance [11; 12; 13]. Parasitic heat losses, such as radiative heat transfer between the hot- and cold-sides diminish the temperature difference across the device, reducing the device's power output and thermal conversion efficiency. Determining F_{ij} values is not a trivial matter due to the inclusion of blocking geometry (thermoelectric element legs and interconnectors) within the device, as shown in Figure 4 where a leg is blocking a ray cast from A_i to A_j, as denoted by the red "x".

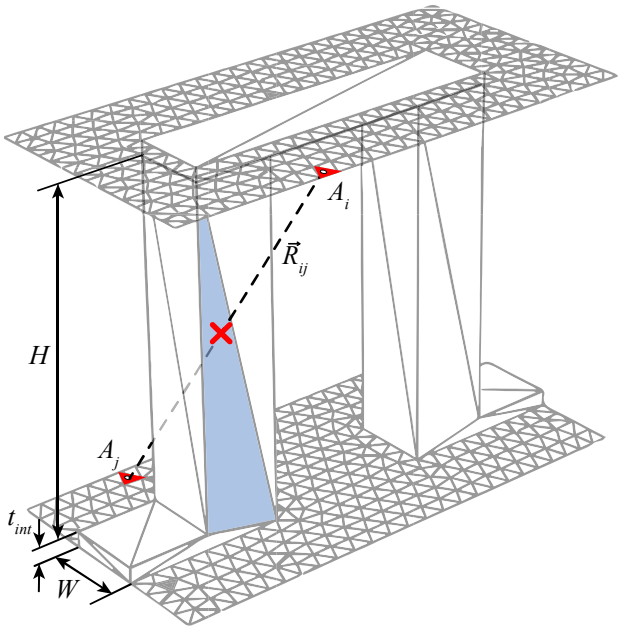

Figure 4. Schematic of a unit-cell TEG used within the RT algorithm. Geometrical parameters are as follows: H/W=4, t_{int}=0.25, ϕ=0.9

TRENDS AND RESULTS

Table 3 shows the average calculated F_{ij} values for CPU, single and dual-GPU execution on the unicouple TEG geometry. It is noted the reported computation times for the CPU, single and dual-GPU execution were repeatable, and the calculated F_{ij} values had on average an absolute difference of 1e-19, indicating no loss of fidelity when executed on single and dual-GPUs in comparison to values obtained via CPU-only execution. It is seen that the calculated F_{ij} values are monotonically converging toward the numeric solution of 5.754540651e-4 taken at 677,544

average tessellations per surface [5].

Table 3. F_{ij} values of unicopule TEG for various average tessellation counts.

		Time [s]		
Avg. Tess.	F_{ij}	CPU	1 GPU	2 GPU
2,527	5.759450587e-4	5.505	0.955	1.350
11,015	5.751577438e-4	16.13	2.764	1.881
71,586	5.754201918e-4	113.5	62.69	33.25
165,291	5.754852079e-4	472.8	274.7	151.8
491,186	5.7545765976e-4	4,160	2,387	1,372

The average runtime improvements of utilizing single and dual-GPU execution, in comparison to CPU-only execution, are a function of tessellation count. For single and dual-GPU execution, as the tessellation count increases, the CPU-computation time far exceeds the GPU-computation times, which is comprised of pre-computations, such as STL file reading, geometric computations, and data transfer. After a cut-in tessellation count, consistent decrements in runtimes are achievable when utilizing single and dual-GPU execution in comparison to CPU-only execution. Thus, considering the last three average tessellation counts within the series, the use of a single GPU allowed for an average attainable decrease in runtime of 1.76-times, while through the use of dual-GPUs, an average decrease in runtime of 3.19-times was achieved. These average speed-up values are less than those reported in [5], albeit different hardware was utilized in this study, as will be further elaborated upon in the following.

Figure 5 depicts the ratio of runtimes for single per dual-GPU cases for varying tessellation counts within the aforementioned unicouple TEG model. A runtime ratio greater than unity indicates that improved runtimes are achievable with the use the multiple GPUs, in comparison to executing the algorithm on a single GPU, as indicated by the green shaded region in Figure 5. When the average emitting and receiving tessellation count exceeds 2,527 per surface, the runtime ratio exceeds unity. With increasing tessellation count, the ratio in runtimes monotonically increases and approaches a maximal value of 1.89. When 71,586 tessellations were used to represent the emitting and receiving surfaces each, the runtime of the dual-GPU execution was nearing one-half the single-GPU execution, and near-linear runtime improvements were achieved. However, increasing surface fidelity by introducing further tessellations resulted in a monotonic decrement of the runtime ratio. This behavior can possibly be explained by the data transfers between the CPU and GPU during runtime.

As per the decrement in runtime improvements, as depicted in the flow chart of the computational algorithm in Figure 2, the differential view factor for every emitting tessellation is retrieved and stored for each receiving tessellation. Thus, as the program iterates through each receiver area in the domain, an array transfer of size proportional to the average number of tessellations is

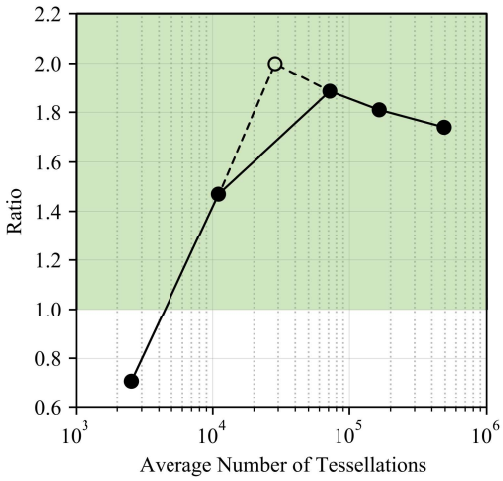

Figure 5. Single per dual-GPU execution runtimes vs. average tessellation count for select cases.

executed from the GPU back to the CPU. Data transfer across devices are expensive operations, and their effect is demonstrated when comparing runtimes between multiple GPU tests. In the single GPU execution, each GPU kernel executes sequentially, with the data transfer occurring sequentially as well. In the double GPU execution, two GPU kernels execute in parallel of the same size as the single GPU execution. However, in this scenario, the CPU must simultaneously compute and transfer data twice as large. In smaller tessellation scenarios, this transfer is tractable, and large runtime gains are seen once the computation time greatly exceeds pre-computation and data transfer times (see runtime ratio of approximately 0.7 for an average tessellation count of 2,527). However, as the problem size grows, i.e. the tessellation count increases, the CPU's processing capabilities are insufficient and a data bottleneck occurs. This results in slow-downs and a lessened GPU execution as the devices await new instructions from the CPU. Therefore, while runtime improvements were observed through the implementation of a multi-GPU method for sub-million tessellation scenarios, a consistent run-time improvement was not seen over a range of average tessellation counts. Rather, after reaching maximal run-time improvement, said improvements degrade as the problem size grew, which is plausibly related to the performance of the computer's CPU.

It is also noted, as shown in Figure 5 but not reported in Table 3, that a situation arose where near-linear speed-up was seen, i.e. dual-GPU execution was almost one-half of the single-GPU execution. This is shown as the maximal data point with dashed lines between preceding and proceeding points. Based on the summation conventions used within Equation 4, the computation time should be quadratic with respect to the average number of tessellations representing the emitting and receiving surfaces. This trend is evident, although to a lesser value in the exponent, with the computation times for all cases, except the situation where the average number of tessellations was equal to

28,361, which is the situation where near linear speed-up was seen. In this situation, the computation time repeatedly, but with consistency, exceeded the proceeding case, indicating this data point was an outlier. A possible explanation for this behavior is the ordering of edges of the STLs used within the blocking geometry for the MT algorithm. For every ray cast, said ray has to be compared against every STL representing blocking surfaces. If ray-triangle intersection occurs early in the algorithm, e.g. one of the first logical checks for intersection fails, minimal time is spent within the MT algorithm. If, however, the ray-triangle intersection occurs near the end of the STL blocking geometry list, a maximal amount of time is spent within the MT algorithm performance. Thus, ordering of the STLs representing the blocking and emitting geometries play a role in the algorithm's performance.

It is seen herein that a robust computational algorithm was developed and implemented to resolve radiation view factors quickly, and more importantly, accurately. Although near-linear speed-up was demonstrated only for limited test cases, it remains plausible in possible future work to achieve consistent, near linear speed-up. Specific run-time improvements for various geometries is not only contingent on the number of tessellations used within the emitting and receiving surfaces, but also on the hardware, namely the CPUs and GPUs used to execute the algorithm. Thus, performance benchmarking should be rigorously pursued on a variety of hardware configurations. It is hypothesized that in a situation where the CPU throughput is sufficient, linear speed-up could be achievable over a range of average tessellation values. Furthermore, runtime benchmarking should be pursued such that reported values are based on a set of averages, with the uncertainty reported to some confidence interval. Additionally, methods to optimize the determination of ray-triangle intersections, such as adaptive re-ordering of blocking STLs based on a real-time heuristic, projection methods, or data-driven modeling, could further reduce runtimes.

CONCLUSION

A multiple-graphics processing unit (GPU) ray-tracing algorithm was developed to determine radiation view factors F_{ij} for complex, three-dimensional geometries. The robust numerical framework handled emitting and participating surfaces as stereolithography (STL) files and considered the shadow effect via the incorporation of the Möller-Trumbore (MT) ray-triangle intersection algorithm. The multi-GPU ray tracing algorithm was validated on two canonical geometries, aligned parallel plates and concentric spheres, and was shown to yield results converging to analytic with increasing tessellation count. Additionally, the absolute difference in numeric solutions between single and multi-GPU executions were on the order of 1e-14 to 1e-15, indicating no loss of accuracy. Quantification of attainable runtime decrements were demonstrated on a unicouple thermoelectric generator (TEG). It was found that runtime improvement ratios for single per dual-GPU executions were often greater than unity for the range of average tessellation cases studied. Runtime improvements were attainable once a sufficient number of average tessellations were used, and said improvements increased to a maximum ratio of 1.89, indicating near-linear speed-up, with the possibility of linear speed-up being achievable. After maximal speed-up was achieved, run-time improvements decreased with increasing tessellation count. There is much future work to be conducted, such as reducing data bottle-necking and dynamic STL ordering to minimize the runtime while resolving the shadow effect.

ACKNOWLEDGMENT

The authors would like to thank the Center for Research Computing at the University of Pittsburgh for providing computational resources.

REFERENCES

[1] T. Walker, S.-C. Xue, and G. Barton. Numerical determination of radiative view factors using ray tracing. Journal of Heat Transfer 2010; 132(7).

[2] J. A. Vílchez, M. Muñoz, J. M. Bonilla, and E. Planas. Configuration factors for ground level fireballs with shadowing. Journal of Loss Prevention in the Process Industries 2018; 51: 169-177.

[3] C. Wang, Z.-H. Wang, Y.-H. Ryu, A single-layer urban canopy model with transmissive radiation exchange between trees and street canyons, *Building and Environment*, Vol. 191, 2001, pp. 107593.

[4] F. F. Sönmez, H. Ziar, O. Isabella, and M. Zeman. Fast and accurate ray-casting-based view factor estimation method for complex geometries. Solar Energy Materials and Solar Cells 2019; 200: 109934.

[5] A. J. Hancock, L. B. Fulton, J. Ying, C. E. Clifford, S. Sammak and M. M. Barry, A GPU-Accelerated Ray-Tracing Method for Determining Radiation View Factors in Multi-Junction Thermoelectric Generators, *Energy*, 2021.

[6] P.-O. Persson and G. Strang. A simple mesh generator in MATLAB. SIAM Review 2004; 46(2): 329-345.

[7] T. L. Bergman, F. P. Incropera, D. P. DeWitt and A. S. Lavine, Fundamentals of heat and mass transfer, 2011, John Wiley & Sons.

[8] T.Möller and B. Trumbore. Fast, minimum storage ray-triangle inter-section. Journal of Graphics Tools 1997; 2(1): 21-28.

[9] Aparapi Team. Official AMD Aparapi Repository. `https://github.com/aparapi/aparapi`, 2021.

[10] J. R. Howell, M. P. Menguc and R. Siegel, Thermal radiation heat transfer, 2010, CRC press.

[11] F. Meng, L. Chen and F. Sun, A numerical model and comparative investigation of a thermoelectric generator with multi-irreversibilities, *Energy* 2011, 36(5): 3513–3522.

[12] M. M. Barry, K. Agbim, P. Rao, C. E. Clifford, B.V.K. Reddy and M. K. Chyu, Geometric optimization of thermoelectric elements for maximum efficiency and power output, *Energy* 2016, 112: 388–407.

[13] M. M. Barry, J. Ying, M. J. Durka, C. E. Clifford, B.V.K. Reddy and M. K. Chyu, Numerical solution of radiation view factors within a thermoelectric device, *Energy* 2016, 102: 427–435