FAST DAYLIGHT COEFFICIENT CALCULATION USING GRAPHICS HARDWARE

Nathaniel L Jones and Christoph F Reinhart
Massachusetts Institute of Technology, Cambridge, MA, USA

ABSTRACT

As people increasingly work in indoor environments, the need to provide natural lighting is becoming more widely recognized. Recent modelling standards such as LM-83 require the use of climate-based metrics based on daylight coefficients, rather than illuminance-based metrics that simulate single points in time. While calculations based on daylight coefficients are fast, computation of the daylight coefficients themselves is a slow process that must be repeated whenever the scene's geometry or materials change. Therefore, it remains impractical to obtain accurate annual daylight simulation results during early design stages when designs are fluid and quickly changing.

This paper describes the development of a new, faster tool to compute daylight coefficients using graphics hardware. The tool is an adaptation of *rtrace_dc*, the executable used by DAYSIM to calculate daylight coefficients, written using OptiX™, a free library for GPU-based ray tracing. The effectiveness of the new tool is demonstrated using models of a typical office space with speed and results compared to *rtrace_dc*. The speed of the new tool is measured both on a typical workstation graphics card and on a high-end graphics server. The results show that the new tool achieves similar accuracy to the serial version but does so in one-fifth the time.

INTRODUCTION

As humans spend 90 percent of the time inside of buildings, the need to provide natural lighting to indoor spaces is becoming widely recognized (EPA, 1989). However, defining, and therefore predicting, good daylighting is far from a straightforward task. While many lighting simulation tools lend themselves to point-in-time calculations, building performance metrics must take into account the annual performance of the building, which requires simulation under multiple solar positions and sky conditions. Climate-based daylighting metrics (CBDMs) represent the annual daylighting performance of a space, an abstract quantity, while agreeing closely with concrete occupant observations (Reinhart, et al., 2014). While CBDMs are useful from a building standards standpoint, their computation is slow and requires more memory than older illuminance-based metrics, which makes them more difficult to integrate into design tools. It remains impractical to obtain accurate annual daylight simulation results during early design stages when the designs change rapidly. In order to produce CBDM results at interactive rates to feed back into an iterative design process, we must use new methods and platforms to calculate them.

DAYSIM is a popular tool for computing CBDMs, frequently accessed through the graphic user interface of DIVA-for-Rhino (Jakubiec & Reinhart, 2011). DAYSIM computes *daylight coefficients*, measure of the illuminance contributions from direct and diffuse natural sources, which can in turn be used to compute the *daylight autonomy* of a space (Reinhart & Walkenhorst, 2001). Like the Radiance suite of programs on which it is based (Larson & Shakespeare, 1998), DAYSIM uses serial ray tracing to perform lighting calculations.

Recently, we have developed Accelerad, a suite of graphics processing unit (GPU)-enabled programs that implement the core ray tracing functionality of Radiance at speeds an order of magnitude faster (Jones & Reinhart, 2014a; Jones & Reinhart, 2014b). In this paper, we demonstrate how the same techniques that make Accelerad possible can be used to speed up DAYSIM calculations. First, we describe tools and metrics that are relevant to the problem. Then, we present factors that affect the accuracy and speedup achieved in GPU-based daylight autonomy calculations. Finally, we comment on limitations of GPU-based methods with respect to daylight coefficient calculation and on future work that may overcome these limitations.

BACKGROUND

Daylighting Metrics

Early metrics for measuring and predicting indoor daylight quantities, such as daylight availability and work plane illuminance under clear sky conditions, were quick to calculate but could not be extrapolated to annual values. Recent building standards and rating systems require the use of CBDMs, which consider annual daylight availability, rather than illuminance-based metrics that simulate only a point in time. The Illuminating Engineering Society of North America (IESNA) promotes the use of daylight autonomy as a CBDM through its standard LM-83 (IESNA, 2012).

LM-83 defines a point in space to be daylit if the point achieves a target illuminance of 300 lux or more for over 50% of occupied hours; this measure of daylight autonomy is abbreviated DA_{300lux}[50%]. Studies comparing simulated daylight autonomy results to collected observations further suggest that a point can be considered partially daylit if it achieves DA_{150lux}[50%] (Reinhart, et al., 2014).

In order to calculate daylight autonomy, DAYSIM first calculates daylight coefficients and then combines them with weather data to create a spatiotemporal illuminance mapping. Each daylight coefficient is a weighting factor that represents the contribution of a light source to a point, such that the total illuminance at that point is the sum of all direct and diffuse daylight coefficients multiplied by the respective luminances of their sources at a particular point in time. For diffuse calculations, DAYSIM uses 148 sources corresponding to the 145 Tregenza sky divisions (Tregenza, 1987) and three ring-shaped ground patches. For direct calculations, a location-dependent number of sources represent a uniform spatial distribution of sun positions at the appropriate latitude (Reinhart & Walkenhorst, 2001). At the authors' latitude, 63 direct daylight coefficients are required. Daylight coefficients are calculated using Whitted-style ray tracing (Whitted, 1980); starting from a grid of points at which daylight autonomies are to be calculated, rays are traced through a user-defined number of bounces until a source is encountered. The simulation can be made more accurate (and slower) by increasing the number of bounces through the *–ab* parameter or increasing the ambient accuracy by decreasing the *–aa* parameter.

Daylight Calculation on the GPU

The high degree of parallelism built into modern graphics processors has made their use very appealing for scientific applications. Within the building performance simulation community, they have been used mainly for computations involving manipulation of dense matrices, including applications in computational fluid dynamics (Zuo & Chen, 2010) (Wang, et al., 2011), acoustics (Guillaume & Fortin, 2014), and the three-phase method for lighting simulation (Zuo, et al., 2014). Jones, et al., (2011) reduced direct solar radiation calculations to a manipulation of dense matrices in OpenGL®, and Kramer, et al., (2015) extended this solution to general direct radiant heat exchange. These applications are generally well suited to GPU computation because of its single-instruction, multiple-data (SIMD) programming model in which the same operation is applied simultaneously to each matrix element. As a result, large speedups are possible through parallelism.

Ray tracing is highly parallel in concept because each primary ray acts independently of other rays. However, depending on the material of the surface hit by a ray, the computation it requires may involve not only different data, but different instructions as well. Rather than SIMD, this necessitates a single-instruction, multiple-thread (SIMT) programming model, and even then, divergence between threads computed in the same warp leads to program inefficiencies. GPU language extensions such as Compute Unified Device Architecture (CUDA®) from NVIDIA® and OpenCL™ from the Khronos™ Group make it possible to implement ray tracing on GPU shader processors (Aila & Laine, 2009; Wang, et al., 2009). The OptiX™ ray tracing engine, created by NVIDIA®, allows ray tracing to be performed on the GPU from any application without requiring the programmer to implement their own ray tracing algorithms (Parker, et al., 2010). Currently, there is no well-supported OpenCL™ alternative to OptiX™, although the Embree ray-tracing framework from Intel® may provide a CPU-based alternative in the future (Wald, et al., 2014).

The building performance simulation community uses OptiX™ for a variety of ray tracing tasks. Clark (2012) and Halverson (2012) demonstrate its use for modelling radiative heat transfer involved in the urban heat island effect. Andersen et al. (2013) use it for interactive visualization of cached Radiance results. Our own tool, Accelerad, replaces Radiance's own ray tracing engine with OptiX™ to achieve speeds twenty times faster than Radiance (Jones & Reinhart, 2014a; Jones & Reinhart, 2014b).

IMPLEMENTATION

Before describing the algorithms that compute daylight coefficients on the GPU, a brief primer in the DAYSIM code will be useful. DAYSIM is a collection of programs that call each other, and are in turn called by interfaces such as DIVA-for-Rhino. DAYSIM prepares daylight coefficients through three invocations to *gen_dc*, the first two to initiate calculation of diffuse and direct daylight coefficients, respectively, and the third to merge the results into a single file. A second program, *ds_illum*, combines the computed daylight coefficients with climate data, the results of which serve as input to calculate daylight autonomy. The first two runs of *gen_dc* do not directly create output; rather, they call another program, *rtrace_dc*, which performs ray-tracing calculations and generates daylight coefficients. In long DAYSIM calculations with high accuracy settings, *rtrace_dc* is responsible for most of the computation time.

DAYSIM's *rtrace_dc* is a relatively straightforward modification of Radiance's *rtrace*; it differs by the addition of a daylight coefficient array at each place where *rtrace* stores a colour in intermediate and result calculations. Each operation that modifies a colour is also performed on the elements of the daylight coefficient array. Similarly, *accelerad_rtrace* is also a modified version of *rtrace*, but the modification is more complex as it involves a language translation from C to CUDA® and a parallel irradiance caching strategy (Jones & Reinhart, 2014b). We created a GPU

implementation of *rtrace_dc* by combining both sets of modifications (Figure 1). The DAYSIM modifications were added to the Accelerad source code, as they were the smaller set of alterations.

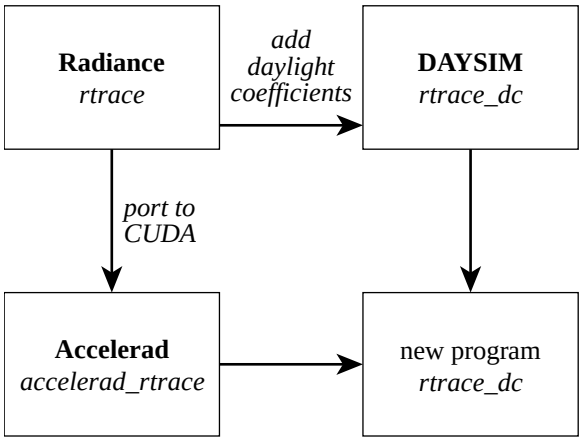

Figure 1 The new program combines the alterations made to Radiance rtrace *from DAYSIM with those from Accelerad*

Filling Warps

In CUDA®, groups of 32 threads called warps are processed together using a SIMT model. Filling a warp with rays that hit similar materials in the same order makes the computation more coherent and hence faster. We achieve better coherence by grouping primary rays based on the spatial positions of their origins. This technique works well when the sensor positions form a grid. Each warp might, for instance, perform calculations on a 4×8 section of the grid.

To communicate the dimensions of this grid to *rtrace*, we use the *–x* and *–y* arguments in an analogous fashion to their use in *rpict*. In the original *rtrace*, *–x* and *–y* are optional arguments used to flush output. When the dimensions are provided, *accelerad_rtrace* creates a launch context measuring *x* by *y* threads. If the parameters are not entered, the launch context will measure 1 by *n* threads, where *n* is the number of sensor points given in input. In the latter approach, the single column of active threads will incompletely fill the 4×8 warps. Hence, gridded input with *–x* and *–y* arguments is preferable for computational efficiency.

Memory Allocation

Memory limitations are an important consideration in porting code to the GPU. While today's GPUs have large global memory spaces (3 to 12 GB for the devices tested on), the on-chip L1 cache is much smaller and is shared by several parallel threads. In OptiX™, each thread is limited to 256 registers, which is not enough space to store even a single daylight coefficient array; the remainder must spill into global memory where it cannot be accessed as quickly.

We can reduce, though not eliminate, this inefficiency by allocating space for daylight coefficient storage in GPU global memory prior to starting the simulation. The strategy is to create a buffer in the GPU's global memory with dimensions $x \times y \times z$, where x and y are taken from the *–x* and *–y* arguments and z is based on the maximum number of reflections given by the *–lr* argument as follows:

$$z = 2DC \times (1 + lr) \quad (1)$$

where *DC* is the size of the array of daylight coefficients in bytes. This reserves one daylight coefficient array to store the cumulative daylight coefficients for each ray until tracing of that ray is complete and another daylight coefficient array for intermediate calculations at each hit, which is needed for ambient and Gaussian specular computations (Figure 2). Each GPU thread accesses only the daylight coefficient arrays belonging to one (*x*, *y*) pair. Under this scheme, a DAYSIM simulation of a 10×10 sensor grid with 148 single-precision daylight coefficients and a maximum of 8 ray reflections requires about a megabyte of GPU global memory, which is well within the limits of today's GPUs. Ambient calculations require additional memory depending on the number of irradiance cache entries (Jones & Reinhart, 2014b).

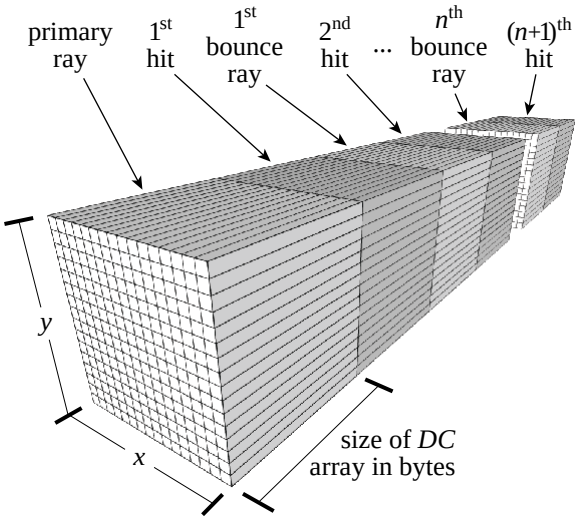

Figure 2 Arrays of daylight coefficients (DCs) are stored in global GPU memory so that each is indexed by thread ID (in the form (x, y)) and level of ray tracing recursion

Each ray payload and hit calculation stores an index to a daylight coefficient in global memory, rather than an entire set of daylight coefficients. The index, stored as an integer *x*-*y*-*z* triplet, requires 12 bytes and fits easily in the GPU thread's local memory. This also means that details of the implementation, such as the number and size of daylight coefficients, can be changed without affecting local memory requirements. For instance, while our current implementation copies DAYSIM's use of a single colour channel for daylight coefficients, future implementations could store separate daylight coefficients in red, green, and blue channels without increasing local memory requirements.

TEST RESULTS

We demonstrate the effectiveness of the GPU implementation of *rtrace_dc* by comparing it to a serial implementation. The serial implementation differs from the version of *rtrace_dc* distributed with DAYSIM in that it is compiled natively on Windows and incorporates the updates of Radiance 5.0. This allows a more accurate comparison of the serial and parallel versions as their algorithms are more nearly identical.

The model used for tests is the south-facing reference office at the authors' latitude (Reinhart, et al., 2013). The office interior measures 3.6 by 8.2 meters and is spanned by a grid of 1400 irradiance sensors (x = 56, y = 25) at 0.15-meter spacing (Figure 3). Except as noted, simulations used the recommended default parameters for the reference office, summarized in Table 1.

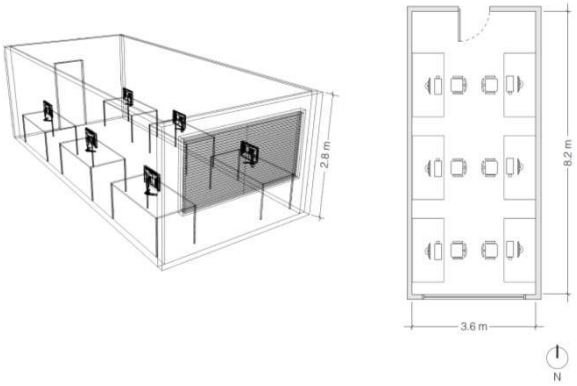

Figure 3 The reference office, shown in perspective and plan views, contains six workstations with a south-facing window (Reinhart, et al., 2013).

Table 1
Default simulation parameters

PARAMETER	CODE	VALUE
Ambient accuracy	*–aa*	0.05
Ambient bounces	*–ab*	7
Ambient divisions	*–ad*	1500
Ambient resolution	*–ar*	300
Ambient super-samples	*–as*	20
Direct jitter	*–dj*	0
Direct relays	*–dr*	2
Direct sampling	*–ds*	0.2
Maximum ray reflections	*–lr*	6
Minimum ray weight	*–lw*	0.004
Specular sampling	*–ss*	1
Specular threshold	*–st*	0.15

Accelerad introduces a parameter *–ac* to control the number of ambient values calculated at each bounce and stored in its irradiance cache (Jones & Reinhart, 2014b). For tests of the GPU implementation, this parameter was set to 4096 except where noted otherwise. The CPU implementation has no equivalent parameter as its irradiance cache is dynamically sized.

Simulations were run on two machines. The serial implementation of *rtrace_dc* was tested on a workstation with a 3.4 GHz Intel® Core™ i7-4770 processor and an NVIDIA® Quadro® K4000 graphics card with 768 CUDA® cores. Except as noted, the GPU version was tested on a workstation with a 2.27 GHz Intel® Xeon® E5520 processor and two NVIDIA® Tesla® K40 graphics accelerators with 2880 CUDA® cores each. These assignments were made so that the serial tests had access to a faster CPU and the parallel tests had access to more GPU cores.

The results that follow demonstrate the speed and accuracy of the GPU implementation of *rtrace* and *rtrace_dc* in comparison to the serial implementation and its dependence on ambient accuracy, irradiance cache size, sensor grid size, and number of GPU cores. Reported simulation times and daylight autonomies are based on an average of thirty trials.

Without Daylight Coefficients

A comparison of *rtrace* and *accelerad_rtrace* provides a baseline expectation for the speedup that can be achieved on a GPU. The model was run in both non-DAYSIM programs with the parameters from Table 1. Simulation times are represented in Figure 4. For the reference office model with the given simulation parameters, Accelerad produced a speedup factor of 6.1 over Radiance.

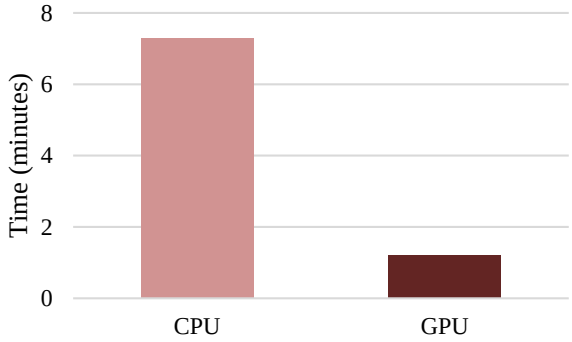

Figure 4 Ray tracing simulation time for the reference office in rtrace *and* accelerad_rtrace *without daylight coefficient calculations*

Ambient Accuracy

The CPU and GPU implementations differ with respect to varying ambient accuracy because of different strategies for placing ambient calculation points. In the CPU *rtrace_dc* version, the irradiance cache is sized dynamically, so reducing the value of *aa* results in reduced calculation point spacing, and thus both increased simulation time and accuracy. The parallel irradiance caching strategy used by the GPU requires that the irradiance cache's size be set in advance. Decreasing the value of *aa* no longer changes the number of ambient values calculated, but it still reduces the radius associated with each ambient value, so the time taken for intersection testing in ray tracing decreases. For the high-accuracy setting of *aa* = 0.05, the GPU implementation produces a speedup factor of

5.1 (Figure 5). Increasing the value of *aa* speeds up the CPU and slows down the GPU implementation. For the reference office model with the chosen simulation parameters, the break-even point occurs close to *aa* = 0.1.

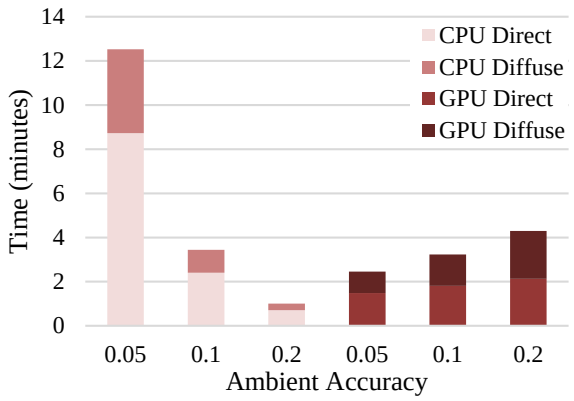

Figure 5 Simulation time as a function of changing ambient accuracy settings for CPU and GPU implementations of rtrace_dc

The accuracy of each simulation is presented in terms of simulated percent of floor area achieving DA_{300lux}[50%] and DA_{150lux}[50%]. In general, increasing the ambient accuracy parameter value increases the reported daylight autonomy because overlap from multiple cached irradiance values becomes more probably. This effect is more pronounced on the GPU, where the fixed-size cache results in poor spatial coverage, meaning some rays do not encounter any cached irradiance value at all.

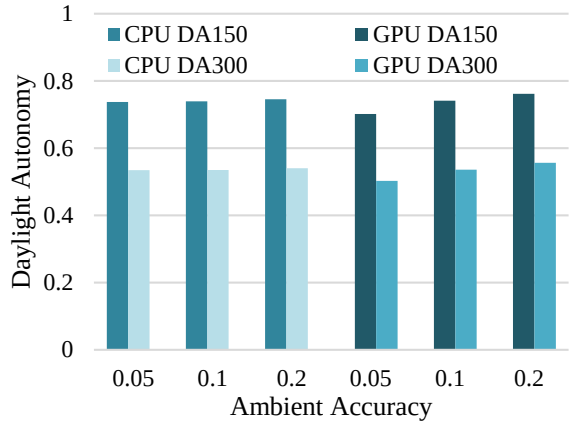

Figure 6 Calculated DA_{150lux}[50%] and DA_{300lux}[50%] for the reference office as a function of changing ambient accuracy settings

In CPU simulations, the floor area achieving DA_{300lux}[50%] ranged from 53.5% at *aa* = 0.05 to 54.0% at *aa* = 0.2. The increased accuracy achieved at *aa* = 0.05 is negligible because the metric represents both a spatial and time average. On the GPU, the floor area achieving DA_{300lux}[50%] ranged from 50.3% at *aa* = 0.05 to 55.6% at *aa* = 0.2. This larger range demonstrates the effect of incomplete ambient coverage at lower *aa* values. Similar results were found for DA_{150lux}[50%] (Figure 6). However, even the largest discrepancies between CPU and GPU calculations, 5.9% for DA_{300lux}[50%] and 4.9% for DA_{150lux}[50%], are reasonably small in comparison to other studies (Reinhart & Walkenhorst, 2001).

Irradiance Cache Size

The previous results highlight the importance of correctly sizing the irradiance cache for the GPU to provide good spatial coverage of the scene with ambient values. Figure 7 shows the effect of doubling the value of *ac* to increase the size of the irradiance cache. The larger irradiance cache increases GPU simulation time substantially, although the effect is not as pronounced as that of increasing *aa* on the CPU's speed, which effectively also creates a larger irradiance cache. The doubled irradiance cache size of 8192 still produces a speedup factor of 1.9 over *rtrace_dc* at the same ambient accuracy setting.

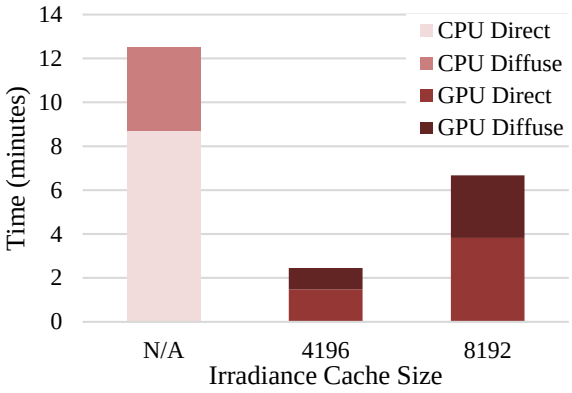

Figure 7 Simulation time as a function of changing irradiance cache size for CPU and GPU implementations of rtrace_dc

The larger irradiance cache produces more accurate results (Figure 8). The floor area achieving DA_{300lux}[50%] increases to 52.5% at *ac* = 8192. This represents a discrepancy between the CPU and GPU of only 1.8%. The corresponding discrepancy for DA_{150lux}[50%] is 2.0%.

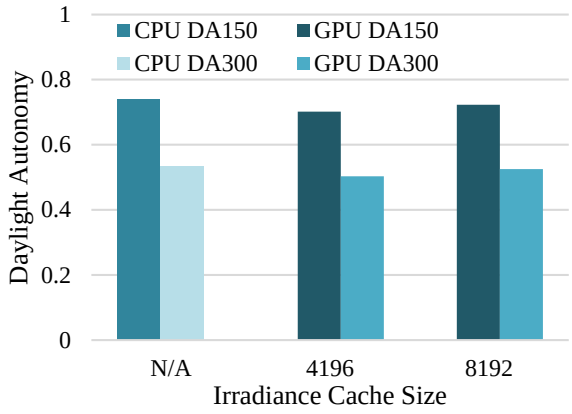

Figure 8 Calculated DA_{150lux}[50%] and DA_{300lux}[50%] for the reference office as a function of changing irradiance cache size

Grid Size

GPUs offer the promise of highly parallel computing, but their speed is inherently limited by the level of parallelism in the problem, which for computation of daylight coefficients is determined by the number of sensor points. Two additional models were used to assess the performance of the GPU implementation under larger computational loads. The first doubled the number of irradiance sensors to 2800 by including two side-by-side copies of the reference office in south-facing orientation. The second included twenty adjacent copies of the office, half facing south and half facing north, along with a denser grid of 32400 sensor points. For the GPU implementation, the irradiance cache size was also doubled to 8192 for the doubled model and increased to 16384 for the large model.

The speedup factor was 3.1 for the doubled model and 3.3 for the large model (Figure 9). These speedups are better than those achieved by increasing only *ac*, which allows us to separate the effect of irradiance cache size from coverage of area by the irradiance cache.

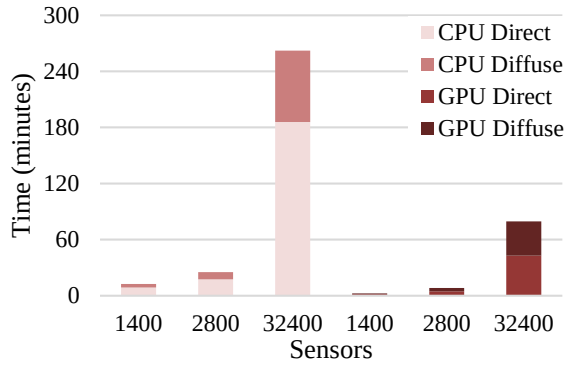

Figure 9 Simulation time as a function of changing sensor grid size for CPU and GPU implementations of rtrace_dc

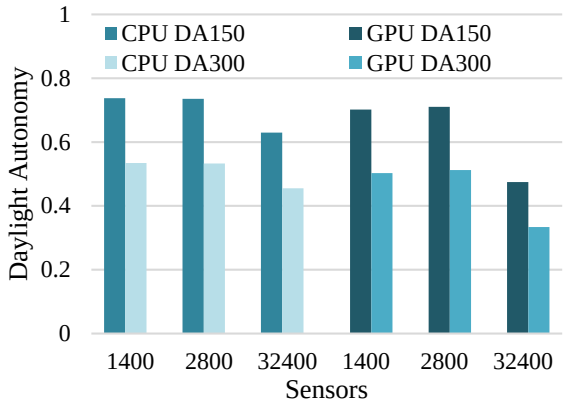

Figure 10 Calculated DA_{150lux}[50%] and DA_{300lux}[50%] for the reference office as a function of changing sensor grid size

Unfortunately, because ambient coverage is not improved in the larger models, the discrepancy in daylight autonomy between the CPU and GPU remains (Figure 10). The doubled model sees a slightly smaller discrepancy of 3.8% for DA_{300lux}[50%] and 3.4% for DA_{150lux}[50%]. This improvement suggests that increasing model size aids in the even distribution of ambient calculation points. The large model produces bigger discrepancies of 26.6% for DA_{300lux}[50%] and 24.6% for DA_{150lux}[50%]. At the scale of this model, the irradiance cache size is insufficient to generate more accurate results, but larger caches push the limits of the current hardware. Larger values of *aa* might also help to improve the accuracy of these GPU calculations.

GPU Core Count

The speed of daylight coefficient calculation for the reference office was tested on several different GPU configurations (Figure 11). The Quadro® K4000, which has roughly one-third the number of cores as the Tesla® K40, produces a smaller speedup factor of 1.2. Using only one Tesla® K40 instead of two results in a speedup factor of 3.8 instead of 5.1. The doubled number of cores available from two graphics accelerators does not double computational speed. This is the result of two factors; first, the setup required for each card is performed as a serial operation on the CPU, and second, additional time is required to synchronize memory between GPUs.

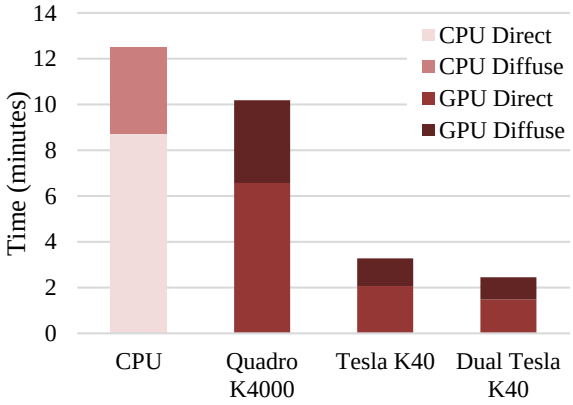

Figure 11 Simulation time as a function of hardware used to run rtrace_dc

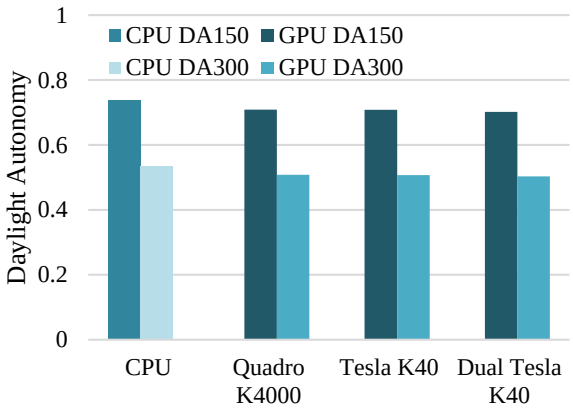

Figure 12 Calculated DA_{150lux}[50%] and DA_{300lux}[50%] for the reference office as a function of hardware used to run rtrace_dc

The daylight autonomies calculated by the different GPU configurations are nearly identical, as is expected given that the same algorithms are used on each GPU (Figure 12). The discrepancies in floor areas achieving DA_{300lux}[50%] range from 5.0% to 5.9% and from 3.9% to 4.9% for DA_{150lux}[50%]. While these are small differences, they are well outside the standard deviations recorded over thirty trials of each platform. This may indicate further work necessary in preserving stochasticity on varying numbers of threads or in synchronizing irradiance caches and daylight coefficients between multiple GPUs.

CONCLUSIONS

This study demonstrates the potential of GPU computation to speed up the calculation of daylight coefficients while producing acceptable accuracy. The maximum speedup factor of 5.1 achieved in these tests is a good first step, yet it seems unimpressive compared to the twenty-fold speedups seen in image-creation tests of *accelerad_rpict* (Jones & Reinhart, 2014b). There are a number of factors that must be considered to improve speeds in future work:

- Models need more potential for parallelism. Image generation assigns one primary ray to each pixel, such that a 512 × 512 image has 262,144 primary rays that may be traced in parallel. Irradiance sensor simulation assigns only one primary ray to each sensor, however, resulting in many fewer rays that could be traced in parallel. Even without daylight coefficient computation, *accelerad_rtrace* only produces a speedup factor of 6.1 on 1400 primary rays. In the future, we expect designers to simulate larger models, which will naturally result in an increased potential for parallelism.
- Better scene coverage is needed with smaller irradiance cache sizes. The most time-consuming simulation components of the GPU implementation are calculation of ambient values both within the irradiance cache and outside it when a ray does not intersect any cached ambient value. New algorithms that more evenly distribute ambient calculation points will produce results faster and with greater accuracy.
- Faster memory access and more efficient daylight coefficient storage are needed. Daylight coefficient calculation adds to Accelerad the need to store a large amount of frequently accessed data in the GPU's global memory. The memory requirement grows with the number of sensor points and also with the size of the irradiance cache. At the same time, daylight coefficient arrays are often sparse, and static allocation of these arrays necessarily leaves space for more bounces than are likely to be calculated. Condensing memory requirements and accelerating memory access will result in faster simulations.
- Graphics accelerator capabilities must increase. As shown in Figure 11, increasing the number of cores on a GPU is significantly more effective at improving performance than increasing the number of GPUs. Daylight coefficient calculation is thus well-positioned to take advantage of the current trend toward increased core counts (Sutter, 2005).

Many of these factors will also help to improve the accuracy of daylight coefficient calculation on the GPU. However, even in its current state, the relative error in daylight autonomy calculations, typically under 6%, is acceptable for early-stage design analysis. Future work should concentrate on speed improvements.

We hope in the future to translate the speedup potential of GPUs into tools to aid in better building design. Calculation of daylight coefficients in parallel using GPU computation promises to make CBDM calculations faster. This speedup will allow building designers to calculate CBDMs earlier and more frequently in the design process, and will in turn make designers more aware of and better informed about daylighting performance.

ACKNOWLEDGEMENT

This research was funded through the Kuwait-MIT Center for Natural Resources and the Environment by the Kuwait Foundation for the Advancement of Sciences. The Tesla K40 accelerators used for this research were donated by the NVIDIA Corporation.

REFERENCES

Aila, T. & Laine, S., 2009. Understanding the efficiency of ray traversal on GPUs. *Proceedings of High-Performance Graphics 2009,* pp. 145-149.

Andersen, M., Guillemin, A., Amundadottir, M. L. & Rockcastle, S., 2013. Beyond illumination: An interactive simulation framework for non-visual and perceptual aspects of daylighting performance. *Proceedings of BS2013: 13th Conference of International Building Performance Simulation Association, Chambéry, France, August 26-28,* pp. 2749-2756.

Clark, J. G., 2012. *A Fast and Efficient Simulation Framework for Modeling Heat Transport.* Master's thesis: University of Minnesota.

EPA, 1989. *Report to Congress on indoor air quality: Volume 2,* Washington, D.C.: United States Environmental Protection Agency.

Guillaume, G. & Fortin, N., 2014. Optimized transmission line matrix model implementation for graphics processing units computing in built-up environment. *Journal of Building Performance Simulation,* 7(6), pp. 445-456.

Halverson, S., 2012. *Energy Transfer Ray Tracing with OptiX*. Master's thesis: University of Minnesota.

IESNA Daylighting Metrics Committee, 2012. *Lighitng Measurement #83, Spatial Daylight Autonomy (sDA) and Annual Sunlight Exposure (ASE),* New York: IESNA Lighting Measurement.

Jakubiec, J. A. & Reinhart, C. F., 2011. DIVA 2.0: Integrating daylight and thermal simulations using Rhinoceros 3D and EnergyPlus. *Proceedings of Building Simulation 2011: 12th Conference of International Building Performance Simulation Association, Sydney, 14-16 November,* pp. 2202-2209.

Jones, N. L., Greenberg, D. P. & Pratt, K. B., 2011. Fast computer graphics techniques for calculating direct solar radiation on complex building surfaces. *Journal of Building Performance Simulation,* 5(5), pp. 300-312.

Jones, N. L. & Reinhart, C. F., 2014a. Physically based global illumination calculation using graphics hardware. *Proceedings of eSim 2014: The Canadian Conference on Building Simulation,* pp. 474-487.

Jones, N. L. & Reinhart, C. F., 2014b. Irradiance caching for global illumination calculation on graphics hardware. *2014 ASHRAE/IBPSA-USA Building Simulation Conference, Atlanta, GA, September 10-12,* pp. 111-120.

Kramer, S. C. et al., 2015. Fully parallel, OpenGL-based computation of obstructed area-to-area view factors. *Journal of Building Performance Simulation,* 8(4), pp. 266-281.

Larson, G. W. & Shakespeare, R., 1998. *Rendering with Radiance: The Art and Science of Lighting Visualization.* San Francisco: Morgan Kaufmann Publishers, Inc.

Parker, S. G. et al., 2010. OptiX: A general purpose ray tracing engine. *ACM Transactions on Graphics – Proceedings of ACM SIGGRAPH 2010,* 29(4).

Reinhart, C. F., Jakubiec, J. A. & Ibarra, D., 2013. Definition of a reference office for standardized evaluations of dynamic façade and lighting technologies. *Proceedings of BS2013: 13th Conference of International Building Performance Simulation Association, Chambéry, France, August 26-28,* pp. 3645-3652.

Reinhart, C. F. & Walkenhorst, O., 2001. Validation of dynamic RADIANCE-based daylight simulations for a test office with external blinds. *Energy and Buildings,* Volume 33, pp. 683-697.

Reinhart, C., Rakha, T. & Weissman, D., 2014. Predicting the daylit area—A comparison of students assessments and simulations at eleven schools of architecture. *LEUKOS: The Journal of the Illuminating Engineering Society of North America,* 10(4), pp. 193-206.

Sutter, H., 2005. A fundamental turn toward concurrency in software. *Dr. Dobb's Journal,* 30(3), pp. 16-22.

Tregenza, P., 1987. Subdivision of the sky hemisphere for luminance measurements. *Lighting Research and Technology,* Volume 19, pp. 13-14.

Wald, I. et al., 2014. Embree: A kernel framework for efficient CPU ray tracing. *ACM Transactions on Graphics – Proceedings of ACM SIGGRAPH 2014,* 33(4), pp. 143:1-8.

Wang, R., Zhou, K., Pan, M. & Bao, H., 2009. An efficient GPU-based approach for interactive global illumination. *ACM Transactions on Graphics – Proceedings of ACM SIGGRAPH 2009,* 28(3).

Wang, Y., Malkawi, A. & Yi, Y., 2011. Implementing CFD (computational fluid dynamics) in OpenCL for building simulation. *Proceedings of Building Simulation 2011: 12th Conference of International Building Performance Simulation Association, Sydney, 14-16 November,* pp. 1430-1437.

Whitted, T., 1980. An improved illumination model for shaded display. *Communications of the ACM,* 23(6), pp. 343-349.

Zuo, W. & Chen, Q., 2010. Fast and informative flow simulations in a building by using fast fluid dynamics model on graphics processing unit. *Building and Environment,* 45(3), pp. 747-757.

Zuo, W., McNeil, A., Wetter, M. & Lee, E. S., 2014. Acceleration of the matrix multiplication of Radiance three phase daylighting simulations with parallel computing on heterogeneous hardware of personal computer. *Journal of Building Performance Simulation,* 7(2), pp. 152-163.