2014 ASHRAE/IBPSA-USA
Building Simulation Conference
Atlanta, GA
September 10-12, 2014

IRRADIANCE CACHING FOR GLOBAL ILLUMINATION CALCULATION ON GRAPHICS HARDWARE

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

ABSTRACT

Recent developments in integrated circuit technology tend toward increased numbers of cores rather than faster clock speeds, so software must use parallelism to achieve faster run times. The ray tracing performed by Radiance is highly parallelizable in concept, with the exception of irradiance caching that serially stores and retrieves results of expensive indirect irradiation computations. This paper describes a novel method of parallel irradiance caching for global illumination on a graphics processing unit (GPU).

The irradiance caching method in this paper closely follows the placement and distribution of rays produced by Radiance using the free OptiX™ ray tracing engine. Ambient values are stored in GPU memory which can as necessary be used to create a bounding volume hierarchy (BVH) of known irradiance records. Queries into the irradiance cache are performed by casting a short ray into the BVH. The current implementation is able to generate images similar to those created by Radiance's RPICT program up to twenty times faster.

INTRODUCTION

Moore's law gave Radiance a free ride for many years. Until the mid-2000's, central processing unit (CPU) clock speeds dependably doubled every 1.5 to 2 years in accordance with the law, which states that the number of transistors on integrated circuits increases at that rate. This allowed Radiance's developers and users to pursue ever more complicated simulations with the assurance that the software would speed up accordingly on new generations of hardware. However, today's integrated circuit designers favor allocating the extra transistors of new chip generations to additional cores rather than higher clock speeds. As a result, the speed of serial programs like Radiance has not increased in the last ten years (Sutter, 2005). In order to integrate computationally expensive uses of Radiance, such as glare analysis and daylight autonomy studies, into design tools that run at interactive speeds, software developers must pursue multicore solutions.

The Radiance suite of programs (Larson & Shakespeare, 1998) has become the gold standard for global illumination calculation used by architects and lighting designers. This can be attributed to Radiance's flexibility, open source availability, and extensive validation through comparison to physical architectural spaces (Grynberg, 1989; Ng, et al., 2001; Galasiu & Atif, 2002), controlled environments (Mardaljevic, 1995; Reinhart & Herkel, 2000; Mardaljevic, 2001; Reinhart & Walkenhorst, 2001), and specific material properties (Reinhart & Andersen, 2006). Radiance is used as a simulation engine by widely-used building performance simulation tools such as IES<VE>, Ecotect®, OpenStudio, DAYSIM, and DIVA for Rhino. However, Radiance simulations tend to be slow, especially in large scenes. As a result, global illumination simulation with Radiance tends to take place late in the design process, after most design decisions are made, or use simplified simulation settings that may not accurately predict physical conditions. Faster simulations are necessary in order to better predict and design interior lighting.

In this paper, we propose a solution to speed up Radiance calculations by tracing multiple primary rays in parallel on a graphics processing unit (GPU). First, we introduce irradiance caching as a method commonly used in Radiance to speed up serial calculations and describe our method for reading an irradiance cache (IC) on the GPU by mapping it to a bounding volume hierarchy (BVH). Then, we describe how to create and adaptively vary the size of an IC using a multi-stage method on the GPU. Our strategy can be adapted to fit various scenes and view types. Finally, we demonstrate the effectiveness of our method on two scenes of vastly different scales: a fictitious small office and Harvard University's Gund Hall. Our implementation using the OptiX™ 3.5.1 ray tracing engine from NVIDIA® produces results up to twenty times faster than Radiance with accuracy within Radiance's ambient accuracy parameter.

BACKGROUND

The Radiance package includes a number of executable programs built around a specialized backward ray tracing engine. In backward ray tracing, primary rays are emitted from an origin point (a virtual camera or illuminance sensor) to sample the environment. Wherever a ray intersects a surface, it recursively spawns one or more new rays, depending on the surface material, and gathers their results into a single value that is returned as the parent ray's result (Whitted, 1980). Typically, a small number of spawned rays are required for direct and specular reflections, and a much larger number of rays are spawned to sample the indirect irradiance due to ambient lighting at the intersection point. Consequently, ambient calculations tend to dominate the total ray tracing computation time. In Radiance, each ray returns red, green, and blue values in units of radiance ($W \cdot sr^{-1} \cdot m^{-2}$). The array of values returned from the primary rays produces an image.

Ray Tracing on the GPU

While GPUs have been primarily designed for raster rendering, the development of GPU-based ray tracers has closely paralleled the development of programmable GPU raster pipelines. Early GPU ray tracers relied significantly on coopting elements of the raster pipeline and imitated its state machine programming interface (Purcell, et al., 2002; Deitrich, et al., 2003). General purpose GPU (GPGPU) language extensions such as Compute Unified Device Architecture (CUDA™) from NVIDIA® made it possible to implement all components of a ray tracing engine on GPU shader processors (Aila & Laine, 2009; Wang, et al., 2009). In 2010, NVIDIA® released the OptiX™ ray tracing engine, which uses CUDA™ to perform both ray traversal and shading on the GPU (Parker, et al., 2010).

The OptiX™ library is designed to replace serial CPU-based ray tracing engines in existing source code. OptiX™ provides built-in BVH creation and ray traversal algorithms to detect potential ray-surface intersections. The programmer is only required to re-implement ray generation, intersection testing, closest hit, any hit, and miss algorithms as CUDA™ programs. OptiX™ compiles these programs into assembly code and uses a just-in-time compiler to create device-specific instructions at runtime.

OptiX™ has been used to accelerate other building performance simulation tasks. Clark (2012) and Halverson (2012) demonstrate its use for modeling radiative heat transfer involved in the urban heat island effect. Andersen et al. (2013) use it for interactive visualization of cached Radiance results. We have previously demonstrated that by editing the source code of Radiance's RPICT and RTRACE programs, they can perform ray tracing using OptiX™ at speeds twenty times faster than Radiance's default ray tracing engine, provided no irradiance caching is performed (Jones & Reinhart, 2014). However, in order to make the OptiX™ engine's speed competitive with Radiance, irradiance caching must be implemented on the GPU.

Irradiance Caching

While direct and specular reflections change abruptly over spatial dimensions, ambient lighting due to indirect irradiance is less variable. A single ambient value may be applied to all ray intersections within a calculated radius of the point where it was measured. An irradiance cache (IC) is a collection of indirect irradiance values and associated validity radii stored in a hierarchical acceleration structure (an octree in Radiance) that allows them to be quickly retrieved based on geometric position. Given two cached irradiance values at points *E1* and *E2* in Figure 1, the irradiance at point *A* may be found by interpolation, and the irradiance at point *B* may be found by extrapolation. Only when a ray intersection is not contained within the validity radius of any IC record (such as at point *C*) must a new record be calculated and added to the IC. This strategy reduces overall ray tracing time by an order of magnitude (Larson & Shakespeare, 1998), but it also eliminates the potential for straightforward parallelization because the final value of each ray depends on the IC records created by previous rays.

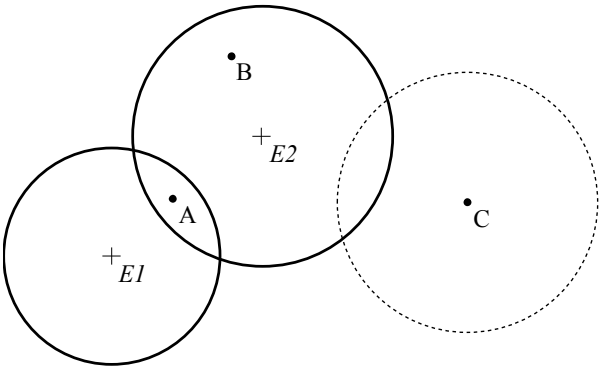

Figure 1 IC records may be applied to all points within their valid radii (Larson & Shakespeare, 1998).

Radiance spawns many ambient rays to create each new IC record. Each ray may in turn spawn new ambient rays if it fails to hit within a preexisting IC record. Every IC record is assigned a level corresponding to the number of ambient bounces taken by the ray that created it. The number of levels is limited by the "-ab" argument in Radiance. At each lower level, IC records accumulate more radiance as a result of a greater number of bounce paths that reach their positions. An IC record cannot influence the ambient radiance of a ray spawned more

than one level below, as this would reduce the number of ambient bounces contributing to the calculated radiance at that point. Only level zero IC records contribute to the ambient radiance of primary rays. Thus, increasing the number of ambient bounces also increases the indirect illumination from sources that reaches the camera (Figure 2). This creates a paradox for parallel IC creation: the number and position of IC records at each level depends on the radiance magnitude received from the level above and the visibility to IC records at the level below. If creation of records within each level is to be fully parallelized, there is no starting place.

Various methods have been proposed for creating an IC using concurrent threads. Strategies for CPU clusters typically involve occasional synchronization of separate local ICs assigned to each thread. This can occasionally result in duplicate IC records created simultaneously by more than one CPU. On UNIX systems, multiple instances of Radiance may share a single irradiance cache using network file locks (Larson & Shakespeare, 1998). Synchronization can also be performed using the Message-Passing Interface (MPI) (Koholka, et al., 1999; Debattista, et al., 2006). Dubla et al. (2009) propose a multi-threaded approach that allows wait-free synchronization of local ICs. All of these methods allow different threads to create overlapping IC records, but this happens infrequently because the number of concurrent CPU threads is small.

Unfortunately, this assumption does not hold for the GPU. Modern GPUs implement single-instruction, multiple-thread (SIMT) architectures in which groups of 32 threads called warps simultaneously execute the same command on different data. SIMT architecture allows threads within a warp to take divergent execution paths as a result of the data they receive, but this reduces parallel efficiency, as some threads must idle while others perform the divergent task (NVIDIA, 2012). Faster GPU ray tracing is achieved when the rays computed by each warp are coherent, hitting the same triangles and calling the same intersection programs. Were the Radiance IC strategy to be implemented directly on the GPU, it is highly likely that many threads in each warp would attempt to create overlapping IC records, severely reducing computational efficiency. Furthermore, adding records to the IC's hierarchical acceleration structure can require redistribution of nodes within the structure, leaving the IC temporarily unreadable to threads from other warps. Hence, efficient IC creation is, by nature, a serial operation.

Using GPUs, others have implemented IC creation as a pre-process carried out prior to ray tracing-based image creation. The key insight of these approaches is that appropriate locations for IC records can be predicted based on the camera's location within the scene. Křivánek and Gautron (2009) use splatting to store irradiance values in a two-dimensional cache that may be projected onto the scene from the camera's vantage point. This avoids the need to store IC records in a hierarchical acceleration structure, but it only considers one ambient bounce. Wang et al. (2009) use adaptive seeding and k-means clustering to select locations for IC records, followed by photon mapping to evaluate irradiance values at these points. Frolov et al. (2013) create an irradiance cache in 20 to 30 passes, where each pass involves both addition of IC entries visible to the camera and elsewhere in the scene for secondary rays. Locations for IC records within the field of view are selected using image processing techniques, while those elsewhere in the scene are chosen by z-curve clustering.

All of these existing methods have some limitations. Because they depend on the camera's field of view to determine IC record locations, they do not scale well to situations in which the camera moves or rotates, such as in adaptive zone glare analysis (Jakubiec & Reinhart, 2012). They also assume that the rendered spaces are at least mostly enclosed, and they provide no explicit mechanism for dealing with views to the exterior, which will be common in analysis of daylit scenes. We seek to address these shortcomings.

ALGORITHMS

On the GPU, we must read from and write to the IC at separate times. First, we discuss our method for reading from the IC, which may be performed in conjunction with various methods of IC creation. Then, we describe two methods for creating IC records on the GPU, one optimized for small enclosed spaces and the other adapted to large open spaces.

Figure 2 RPICT renderings with number of ambient bounces ranging from 0 (left) to 5 (right). Adding ambient bounces increases the overall radiance of the scene originating from the sky, though the effect is imperceptible beyond five bounces. Mean image luminance (μ) is shown in cd/m^2.

Reading from an Irradiance Cache in Parallel

Whether or not we use the GPU for IC creation, we can save an IC to a binary file to enable multiple simulations of a scene. Here, we describe how to use an existing IC in OptiX™. Our first step is to enter all available level zero IC records into a BVH acceleration structure. Each IC record represents a disc over which a given indirect irradiance value is valid, along with directional vectors indicating the disc's orientation in space and gradients in the plane of the disc. While OptiX™ generates the BVH automatically, we must specify a bounding volume for each disc. Our OptiX™ bounding box program defines an axis-aligned bounding box (*AABB*) for each entry as

$$AABB_i = P_i \pm ar\sqrt{1 - D_i^2} \qquad (1)$$

where P_i and D_i are the ith coordinates of the disc's center point and normal direction, respectively, r is its radius, and a is Radiance's ambient accuracy parameter (Figure 3). The *AABB*s of all IC records are independent and can be computed in parallel on the GPU, although their insertion into the BVH tree is a serial operation.

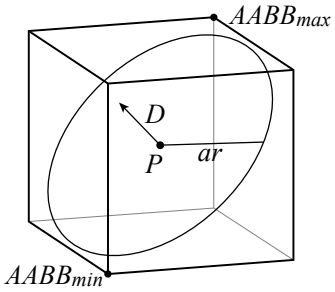

Figure 3 An axis-aligned bounding box for a disc.

Once the IC records are mapped to the BVH, we proceed with the final gather. We use the OptiX™ implementation described in Jones & Reinhart (2014), which is itself based on the source code of RPICT. This implementation follows the behavior of Radiance as closely as possible at material intersections (although currently only plastic, metal, translucent, glass, and light materials are implemented). However, we make the following alteration: at each intersection with a normal material, instead of spawning thousands of ambient rays into the scene, we spawn a single very short ray into the IC BVH acceleration structure. Our OptiX™ intersection program checks each intersected IC record's level, validity radius, and normal direction using the tests from Radiance's `sumambient()` method, which is responsible for summing the contributions of relevant IC records, and adjusts the radiance value in the ray's payload accordingly. At the conclusion of this short ray's traversal, its payload contains the weighted average of the ambient contributions from all IC records it intersected that passed the tests.

If the existing IC does not provide good coverage of the scene, it is possible that a short ray into the IC BVH will not hit any IC records. In this case, it will return an ambient radiance value of zero (Figure 4). To handle this, we calculate the ambient value at the intersection point by spawning new rays into the scene as in Radiance's `doambient()` method, which calculates indirect irradiance at a point when no IC is available. However, this causes poor warp coherence since each ray's samples are likely to hit different objects. To improve performance, we use this method to fill gaps only during the final gather and allow only one ambient bounce in an attempt to reach other IC records.

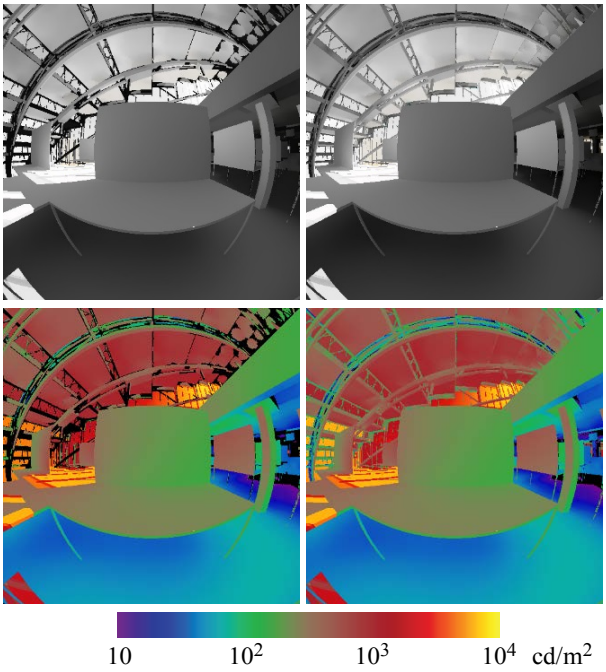

Figure 4 The scene with poor ambient coverage (left) can be filled in during final gather (right).

Creating an Irradiance Cache for Enclosed Spaces

In enclosed spaces, there is limited surface area that needs to be covered by the IC, and there is a good chance that each surface patch will be covered at multiple levels of the IC. In this case, we sample the scene geometry once to choose IC record locations, and we reuse the same locations for new IC records at each level. This eliminates the need to resample the scene geometry for each IC level using additional OptiX™ kernel calls that can double the overall computation time. We first sample the scene to generate a list of candidate IC record locations, then reduce the number of candidates while maintaining even scene coverage using k-means

clustering, and finally create IC records at each cluster in an iterative manner, proceeding from highest to lowest levels (Figure 5). The IC records for level zero are fed into the final gather algorithm described in the previous section.

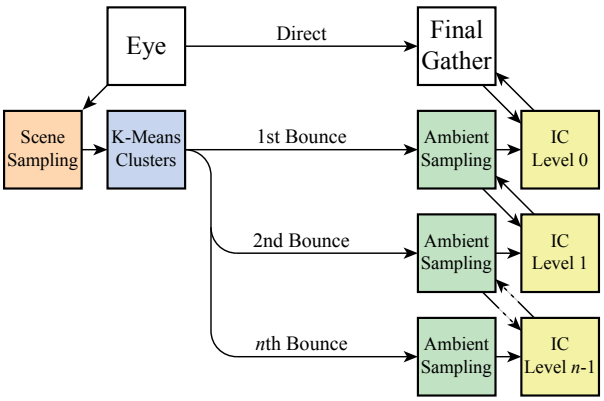

Figure 5 In enclosed spaces, IC record locations from a single call to the sampling kernel are used to create the IC at each level.

First, we sample the scene to create a list of candidate IC record locations. Using an OptiX™ kernel, we cast rays from the eye position into the scene and record the position and surface normal direction of the first hit point. If the eye position and field of view are to remain static, we choose the initial ray directions to form a grid over the image using Radiance's "-vt" argument and a user-specified sampling density. If the eye rotates between images, as it does in glare analysis (Jakubiec & Reinhart, 2012), we distribute the initial ray directions over equal solid angle sections of a sphere. In order to include geometry that is not visible from the eye position, we allow a user-defined number of bounces and record an additional position and normal pair at each new intersection. For each bounce, a random cosine-weighted reflection direction is chosen within the hemisphere defined by the surface normal. The output of this OptiX™ kernel is a list of points and corresponding normals which will be candidate IC record locations.

This list likely contains far more candidates than needed to cover the surfaces in the space, which could cause excessive ray traversal times. Fortunately, we can reduce the list's size by any of a number of clustering methods. For simplicity, we perform iterative k-means clustering to find a user-defined number of cluster centers using CUDA™, starting from a randomly chosen set of candidates. After clustering, the candidate IC record location nearest each cluster center will be used in the next step to generate an IC record.

K-means requires a distance metric in order to cluster nearby objects. In this case, the metric must consider not only Euclidian distance, but also the normal discrepancy between candidates. We use the modification by Wang et al. (2009) of the error in the split sphere model (Larson & Shakespeare, 1998)

$$\varepsilon = \alpha\|x_i - x_k\| + \sqrt{2 - 2(n_i \cdot n_k)} \qquad (2)$$

to relate error ε to the change in position x and normal direction n from candidate i to cluster center k, given a user-defined weighting factor α that accounts for scene size. This modified error metric is preferable because it can be used without calculating the indirect irradiance at every candidate location.

The IC is built through iterative calls to an ambient sampling OptiX™ kernel. In each call to this kernel, one IC record is created in parallel for each chosen candidate location. The process is similar to Radiance's `doambient()` method, which computes the indirect irradiance at a point by sampling the scene with rays, except that no supersampling takes place because OptiX™ does not provide an efficient sorting mechanism or memory to store a large number of ambient samples per thread. This is acceptable because the cost of using a large number of ambient divisions is much lower on the GPU than on the CPU. The first call to the kernel creates the highest IC level by sampling the environment with no ambient bounces. After each kernel call, the new IC records are entered into a BVH as described in the previous section. Subsequent calls to the kernel repeat the indirect irradiance calculation at each location by sampling the IC from the previous round. The IC generated at level zero is used by the final gather as previously described.

Creating an Irradiance Cache for Open Spaces

In open spaces, higher-level IC records are likely to be spread out geometrically, while lower-level records will tend to cluster near the eye position. This differs from the previous case in that we must now choose different record locations for each IC level in order to achieve optimal coverage for each ambient bounce (Figure 6). We make three changes to the method described in the previous section. First, the kernel used initially to sample the scene generates only one point-normal pair per GPU thread as no bounces are needed. The candidate locations, again chosen by k-means clustering, serve as the locations for only the IC records at level zero. Second, we introduce a new scene sampling OptiX™ kernel that spawns rays from the previous cluster centers to identify new candidate IC record locations using Radiance's ambient sampling distribution. The point-normal pairs from this kernel also undergo k-means clustering, and the results are used both as locations for level one IC records and as new input to the same kernel. This process recurses through the number of iterations specified by Radiance's "-ab" argument. Third, while the

IC creation kernel is still called once per level as in the previous section, it now receives a different set of input locations on each call, consuming both the cluster centers from the corresponding level and the IC from its previous invocation. As before, the level zero IC is used for the final gather.

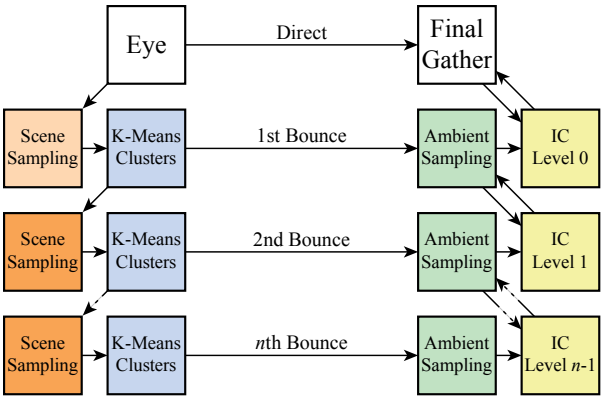

Figure 6 In open spaces, IC record locations are separately calculated for each IC level based on locations reached at the previous level.

VALIDATION

We demonstrate the speed and accuracy of our OptiX™ implementation by comparing it to Radiance's RPICT program for two scenes. The first, a fictitious small furnished office composed of 278,695 triangles, fits the criteria for an enclosed space. The second, a model of Harvard University's Gund Hall with 187,208 triangles, is characteristic of open spaces. We calculate the speedup factor as the ratio of RPICT computation time to the computation time of our OptiX™ implementation with the same number of ambient bounces. In order to quantify the error introduced by our method, we report the mean radiance of the OptiX™-generated image as a percentage of the mean radiance in the RPICT-generated image with the most ambient bounces. This is an imperfect metric because RPICT does produce rendering artifacts, but it serves to demonstrate the extent of agreement between RPICT and the OptiX™ implementation.

Simulations were run on two machines. The first was an active workstation with a 3.4 GHz Intel® Core™ i7-4770 processor and an NVIDIA® Quadro® K4000 graphics card with 768 CUDA™ cores. The second was a dedicated graphics workstation with a 2.27 GHz Intel® Xeon® E5520 processor and two NVIDIA® Tesla® K40 graphics accelerators with 2880 CUDA™ cores each. The OptiX™ implementation was configured to use either one or both accelerators. The standard version of RPICT was run only on the first machine with the faster processor.

Enclosed Space

The small office scene was rendered with varying numbers of ambient bounces in both RPICT and our OptiX™ implementation (Figure 7). Using an ambient accuracy of 5%, minimum ray weight of 0.2%, and 3000 ambient divisions, the number of rays cast by RPICT leveled off at 1.28×10^8 after five ambient bounces, which took 46.5 minutes. We take five ambient bounces to be optimal for this scene with these settings.

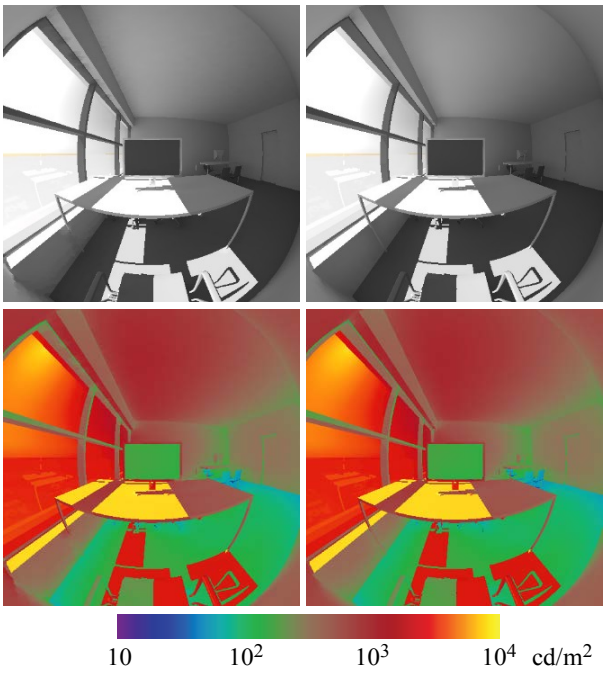

Figure 7 The small office scene rendered with five ambient bounces in RPICT (left) and 17 times faster in our OptiX™ implementation (right).

The small office scene was rendered using the OptiX™ implementations for both enclosed and open spaces. The enclosed method returned results in half the time of the open method for tests with 4096 or more clusters. Because ICs of this size provide good coverage of the small scene, the two methods have comparable accuracy. As a result, we report only the performance of the faster enclosed method.

As with RPICT, the OptiX™ implementation's rendering time increases and its error decreases until five ambient bounces, after which they become essentially constant (Figure 8). Because IC records at higher levels can be built using exponentially fewer rays, the speedup factor is greater for higher numbers of bounces, reaching a maximum of 17 times RPICT's speed when using 4096 clusters.

Increasing the number of clusters also reduces error, though the effect on speed is more complicated. Low

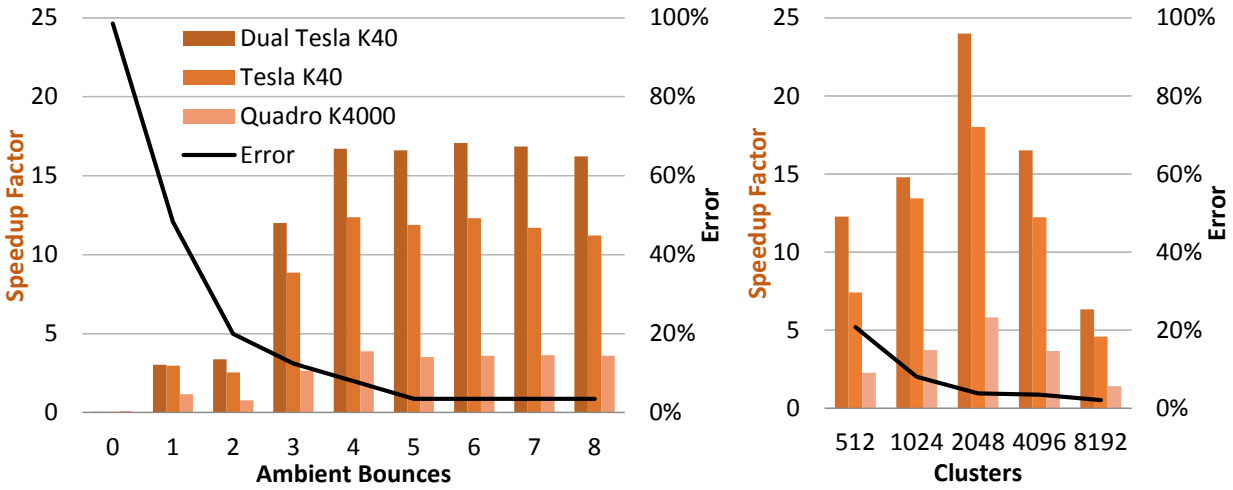

Figure 8 For the small office scene, the speedup factor increases and error decreases with the number of ambient bounces using 4096 clusters (left). Error decreases with the number of clusters, but large numbers of clusters require greater traversal time using five ambient bounces (right).

cluster counts result in reduced ambient coverage, which produces more incoherent work during the final gather, increasing computation time. High cluster counts increase the time for ray traversal of the IC BVH. Using five ambient bounces, a 24-fold speedup can be achieved with 2048 clusters per IC level, but increased accuracy can be achieved with more clusters.

In all cases, the OptiX™-generated images display less radiance than their RPICT counterparts, though the difference is minimal beyond five ambient bounces. The discrepancy is due to less than optimal ambient coverage. The 5% ambient accuracy value used for RPICT produced visually-apparent poor coverage in the OptiX™ implementation. The reported OptiX™ implementation trials used a setting of 10% ambient accuracy, 5% less accurate than RPICT, in order to increase the validity radii of IC records according to equation (1). While this would introduce rendering artifacts into RPICT by spacing IC records farther apart, the ambient accuracy setting does not have this effect in our OptiX™ implementation because the spacing of IC records is determined by the clustering algorithm. In fact, certain rendering artifacts introduced by RPICT are notably absent in the OptiX™ rendering (*e.g.* the lower left-hand wall in Figure 7) because the latter builds the entire IC before calculating any pixel value. We also note that the measured error in our images is less than the difference in ambient accuracy settings.

Open Space

The Gund Hall scene was also rendered with varying numbers of ambient bounces in RPICT and our OptiX™ implementation, although the enclosed method was not used due to the scene's size (Figure 9). Using the same settings as before, the number of rays cast by RPICT leveled off at 1.15×10^9 after five ambient bounces, which took 198 minutes. We again take five ambient bounces to be optimal for this scene with these settings.

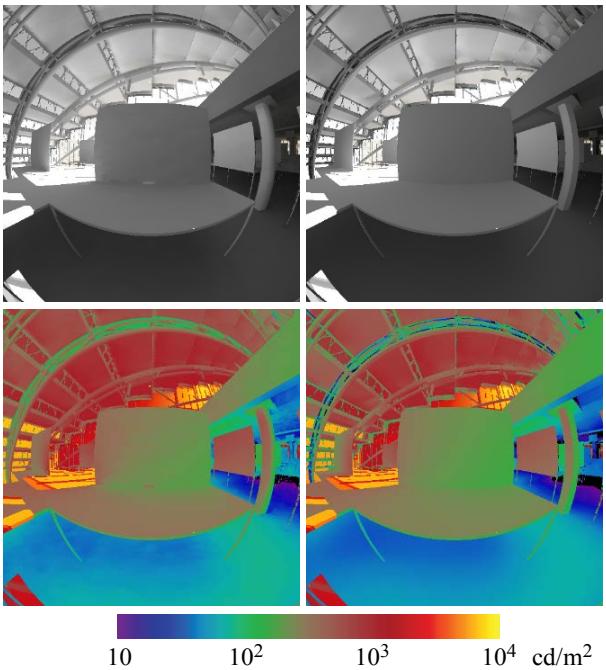

Figure 9 The Gund Hall scene rendered with five ambient bounces in RPICT (left) and 20 times faster in our OptiX™ implementation (right).

Again, the OptiX™ implementation's rendering time increases and its error decreases until five ambient bounces, after which they become more or less constant

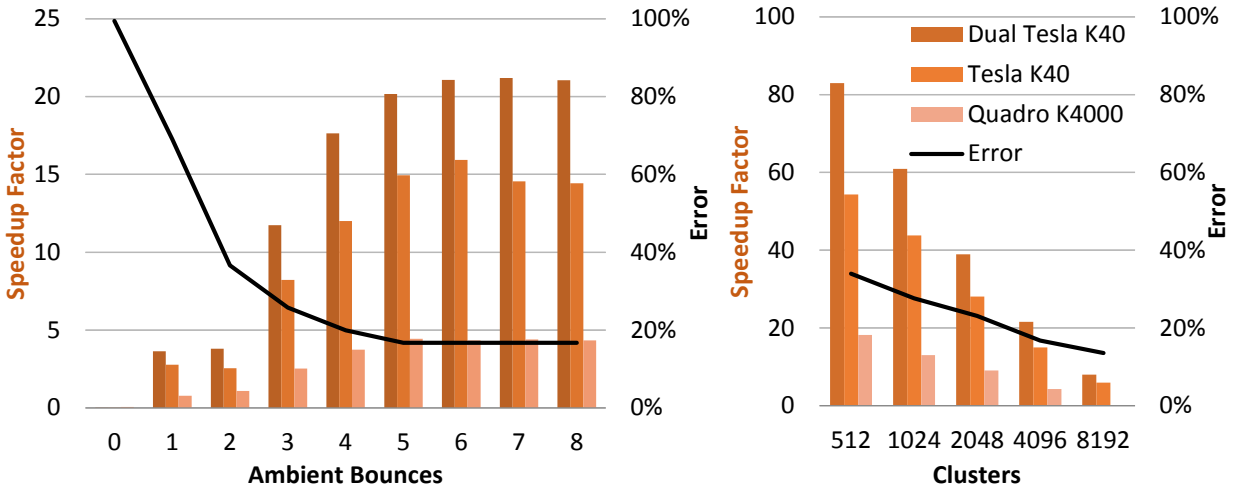

Figure 10 For Gund Hall, the speedup factor increases and error decreases with the number of ambient bounces using 4096 clusters (left). Error decreases with the number of clusters, but large numbers of clusters require greater traversal time using five ambient bounces (right).

(Figure 10). In this larger scene with 4096 clusters, the maximum speedup is 21 times RPICT's speed. Faster speeds can be achieved using fewer clusters because the coherence of final gather rays does not degrade as quickly when IC level zero has its own set of record locations. However, error still increases due to poor coverage at higher levels when the number of cluster centers is low.

The increased error in the Gund Hall scene indicates that coverage is generally poorer here than in the small office scene. This is to be expected, given that Gund Hall is a larger space. To offset this effect, the OptiX™ renderings use an ambient accuracy setting of 25% to increase IC record validity radii. This ultimately produces a 17% difference in mean radiance between RPICT and our OptiX™ implementation with 4096 clusters. While this error appears large, some of it must be attributed to rendering artifacts produced by RPICT (*e.g.* under the table in Figure 10). We again note that the error observed is less than the 20% difference between the OptiX™ implementation and RPICT ambient accuracy settings.

CONCLUSION

Ray tracing for global illumination simulation has many potential uses for architectural design, both actively used and as yet unexplored. In many cases, such as in early design or when global illumination simulation serves as a preprocessing step to another type of analysis, it is important that the simulation execute quickly. However, software developers can no longer depend on faster clock speeds to produce shorter execution times. Instead, they must rely on parallelism.

In this paper, we have demonstrated that irradiance caching, along with other core algorithms from Radiance, can be implemented in OptiX™ to achieve a twenty-fold speed increase in global illumination simulation. By precomputing a separate IC for each ambient bounce level, we can duplicate RPICT results with reasonable accuracy. In enclosed spaces, we can further reduce computation time by reusing the same locations for IC records at each level.

The primary source of error at this preliminary stage of development is poor ambient coverage of the scene. We have shown that in enclosed spaces, ICs that provide good scene coverage can be generated in parallel on the GPU. However, ICs generated in parallel can produce poor coverage for open spaces. Additional ambient rays in the final gather stage can improve image appearance, but it is still necessary to make up for the missing radiance in other ways. Continued work is necessary to determine appropriate number and placement of IC records so as to maximize scene coverage. Ultimately, the accuracy achieved by OptiX™ must be judged against measurements of physical spaces in order to avoid bias from RPICT rendering artifacts.

There are many potential benefits to the architecture profession if Radiance algorithms can be parallelized on the GPU. While single-threaded programs cannot be expected to run faster on new generations of hardware, newer generations of GPUs, like the Tesla® in our trials, continue to outperform their predecessors. Future hardware generations are likely to surpass the twenty-fold speed increase we have demonstrated. Faster simulation results can be produced more frequently as an aid to design, and their sooner availability makes it less likely that the design will change during the simulation,

which renders the results useless. Accurate simulation results make it easier for architects to correctly size windows and provide adequate artificial lighting without consuming unneeded electricity. They also reduce the likelihood of glare, which can decrease productivity in a work environment. Faster ray tracing will also make annual simulations such as daylight autonomy studies more practical, as these take much longer than point-in-time simulations. Thus, we believe that the ability to create and use ICs on the GPU will be of great benefit to building designers.

ACKNOWLEDGMENT

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. Frédo Durand and Jaroslav Křivánek offered useful insight into previous work on irradiance caching. Thanks also to J. Alstan Jakubiec for providing the models and Radiance input files used for timing and illustration.

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.

Debattista, K., Santos, L. P. & Chalmers, A., 2006. Accelerating the irradiance cache through parallel component-based rendering. *Proceedings of the 6th Eurographics conference on Parallel Graphics and Visualization,* pp. 27-35.

Deitrich, A., Wald, I., Benthin, C. & Slusallek, P., 2003. The OpenRT application programming interface - towards a common API for interactive ray tracing. *Proceedings of the 2003 OpenSG Symposium,* pp. 23-31.

Dubla, P., Debattista, K., Santos, L. P. & Chalmers, A., 2009. Wait-free shared-memory irradiance cache. *Proceedings of the 9th Eurographics Symposium on Parallel Graphics and Visualization,* pp. 57-64.

Frolov, V., Vostryakov, K., Kharlamov, A. & Galaktionov, V., 2013. Implementing irradiance cache in a GPU photorealistic renderer. In: M. L. Gavrilova, C. K. Tan & A. Konushin, eds. *Transactions on Computational Science XIX.* Berlin: Springer, pp. 17-32.

Galasiu, A. D. & Atif, M. R., 2002. Applicability of daylighting computer modeling in real case studies: comparison between measured and simulated daylight availability and lighting consumption. *Building and Environment,* 37(4), pp. 363-377.

Grynberg, A., 1989. *Validation of Radiance,* Berkeley, CA: Lawrence Berkeley Laboratories. Document ID 1575.

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

Jakubiec, J. A. & Reinhart, C. F., 2012. The 'adaptive zone' - A concept for assessing discomfort glare throughout daylit spaces. *Lighting Research and Technology,* 44(2), pp. 149-170.

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

Koholka, R., Mayer, H. & Goller, A., 1999. MPI-parallelized Radiance on SGI CoW and SMP. *Proceedings of the 4th International ACPC Conference Including Special Tracks on Parallel Numerics and Parallel Computing in Image Processing, Video Processing, and Multimedia: Parallel Computation,* pp. 549-558.

Křivánek, J. & Gautron, P., 2009. Practical global illumination with irradiance caching. *Synthesis Lectures on Computer Graphics and Animation,* 4(1), pp. 1-148.

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

Mardaljevic, J., 1995. Validation of a lighting simulation program under real sky conditions. *Lighting Research and Technology,* 27(4), pp. 181-188.

Mardaljevic, J., 2001. The BRE-IDMP dataset: a new benchmark for the validation of illuminance prediction techniques. *Lighting Research and Technology,* 33(2), pp. 117-136.

Ng, E. Y.-Y., Poh, L. K., Wei, W. & Nagakura, T., 2001. Advanced lighting simulation in architectural

design in the tropics. *Automation in Construction,* 10(3), pp. 365-379.

NVIDIA, 2012. *CUDA C Programming Guide,* PG-02829-001_v5.0, October 2012.

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

Purcell, T. J., Buck, I., Mark, W. R. & Hanrahan, P., 2002. Ray tracing on programmable graphics hardware. *ACM Transactions on Graphics - Proceedings of ACM SIGGRAPH 2002,* 21(3), pp. 703-712.

Reinhart, C. F. & Andersen, M., 2006. Development and validation of a radiance model for a translucent panel. *Energy and Buildings,* 38(7), pp. 890-904.

Reinhart, C. F. & Herkel, S., 2000. The simulation of annual daylight illuminance distributions - a state-of-the-art comparison of six RADIANCE-based methods. *Energy and Buildings,* 32(2), pp. 167-187.

Reinhart, C. F. & Walkenhorst, O., 2001. Validation of dynamic RADIANCE-based daylight simulations for a test office with external blinds. *Energy and Buildings,* 33(7), pp. 683-697.

Sutter, H., 2005. A Fundamental Turn Toward Cuncurrency in Software. *Dr. Dobb's Journal,* 30(3), pp. 16-22.

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).

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