Eurographics Symposium on Parallel Graphics and Visualization (2006)
Alan Heirich, Bruno Raffin, and Luis Paulo dos Santos (Editors)

Accelerating the Irradiance Cache through Parallel Component-Based Rendering

Kurt Debattista[†1] & Luís Paulo Santos[‡2] & Alan Chalmers[1]

[1]Department of Computer Science, University of Bristol, United Kingdom
[2]Departamento de Informática, Universidade do Minho, Portugal

Abstract

The irradiance cache is an acceleration data structure which caches indirect diffuse samples within the framework of a distributed ray-tracing algorithm. Previously calculated values can be stored and reused in future calculations, resulting in an order of magnitude improvement in computational performance. However, the irradiance cache is a shared data structure and so it is notoriously difficult to parallelise over a distributed parallel system. The hurdle to overcome is when and how to share cached samples. This sharing incurs communication overheads and yet must happen frequently to minimise cache misses and thus maximise the performance of the cache. We present a novel component-based parallel algorithm implemented on a cluster of computers, whereby the indirect diffuse calculations are calculated on a subset of nodes in the cluster. This method exploits the inherent spatial coherent nature of the irradiance cache; by reducing the set of nodes amongst which cached values must be shared, the sharing frequency can be kept high, thus decreasing both communication overheads and cache misses. We demonstrate how our new parallel rendering algorithm significantly outperforms traditional methods of distributing the irradiance cache.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

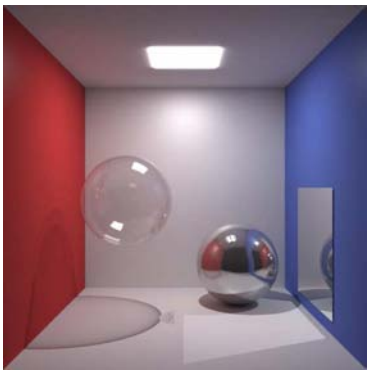
1. Introduction

Traditionally, distributed ray tracing calculated the indirect diffuse component through sub-sampling the hemisphere by shooting a large number of rays at each intersection point. The problem was further compounded due to the recursive nature of this algorithm. Ward *et al.* noticed that the indirect diffuse component was generally a continuous function over space not affected by the high frequency changes common with the specular component. The irradiance cache [WRC88] was designed to exploit this insight. Ward *et al.*'s irradiance cache has become a fundamental algorithm for rendering high fidelity images using global illumination [Her04] (see Figure 1), whether as a stand-alone algorithm for computing indirect diffuse values or when used in conjunction with photon mapping [Jen01]. It is an acceleration data structure which caches indirect diffuse samples within the framework of a distributed ray-tracing algorithm [CPC84]. Initial indirect diffuse samples are calculated the traditional way and the result is cached in the irradiance cache's spatial data structure. Whenever a new indirect value is required the irradiance cache is first consulted. If one or more samples fall within the user-defined search radius of the indirect diffuse value to be computed, the result is extrapolated from the samples using a weighted averaging strategy. Ward *et al.* demonstrated that the irradiance cache offered an order of magnitude improvement in overall computational time over the traditional method. Performance is improved even further when rendering animations of static

† Thanks to Veronica Sundstedt for the scenes and Greg Ward for technical discussions. This work was partially funded by the 3C Research programme.
‡ Parallel cluster supplied by program SEARCH (SErvices and Advanced Research Computing with HTC/HPC cluster), supported by the Portuguese "Fundação para a Ciência e Tecnologia"

(a) The Kalabsha temple

(b) The corridor model



(c) The modified Cornell box

Figure 1: *Models used for experimentation.*

scenes, since the indirect diffuse computation remains constant.

Since the irradiance cache is a shared data structure it is notoriously hard to parallelise efficiently on distributed systems, particularly because it is at its most efficient when each cached sample can be used immediately, thus avoiding replicated computations of diffuse samples among processes. However, sharing implies communication overheads induced by having to transmit values; thus, there is a trade-off between cache misses and sharing frequency. This has been expressed in previous approaches whereby groups of cached values are either stored at a central node and then retrieved by other nodes or broadcast to every node whenever some threshold is reached. We propose a different approach to solving the problem. By conforming to the philosophy of the irradiance cache, we subdivide computation at the component level and transfer any indirect diffuse computations to a set of dedicated indirect diffuse rendering nodes, where indirect values are computed and stored. Sharing amongst the reduced set of indirect diffuse nodes occurs with higher frequency than among other nodes, reducing the sharing overhead while still maintaining a high irradiance cache hit ratio. To the best of our knowledge, this is the first parallel rendering approach to decompose the rendering problem into components as a means of making best use of available resources.

In order to demonstrate our approach we implemented a number of parallel renderers, all extensions of the lighting simulation system *Radiance* [LS98]. Two of these represent the traditional methods [KMG99, RCLL99], while a third renderer uses the novel component-based approach to compute the irradiance cache in parallel.

This paper is divided as follows. In the next section we present related work. In Section 3, we demonstrate the motivation behind our approach. In Section 4 we present our own implementations of traditional approaches to parallelising the irradiance cache. In Section 5 we describe our novel parallel component-based approach to the irradiance cache. In Section 6 we compare results from the traditional renderer and the new parallel renderer. Finally in Section 7 we conclude and describe possible future work.

2. Related work

Our work draws on related work in parallel and component-based rendering.

2.1. Parallel rendering

Parallel rendering algorithms have been used to alleviate the cost of rendering for a number of years. Reinhard *et al.* [RCJ98] and Chalmers *et al.* [CDR02] offer a comprehensive analysis of the standard approaches for static and dynamic load balancing, data and task management, and more advanced approaches. Parker *et al.*'s parallel ray tracer [PMS*99], through optimised code, ray traced simple scenes interactively on a shared memory parallel computer. Wald *et al.* [WBWS01] also obtained interactive rates for ray tracing, this time over a distributed cluster and by using cache-coherent techniques and SIMD instructions commonly found in modern architectures. Subsequently, in [WKB*02], they extended their distributed ray tracer to interactively render images using global illumination by adapting Keller's instant radiosity algorithm [Kel97]. Günther *et al.* [GWS04] extended this distributed framework further to support caustics through photon mapping.

2.2. Parallel irradiance cache

There have been a number of implementations of a parallel irradiance cache within *Radiance*. The standard *Radiance* distribution [LS98] supports a parallel renderer over a distributed system using the Network File System (NFS) for concurrent access of the irradiance cache. This has been known to lead to contention and may result in poor performance when using inefficient file lock managers. Koholka *et al.* [KMG99] used the Message-Passing Interface (MPI) instead of NFS for their distributed *Radiance* implementation. The irradiance cache values are broadcast amongst processors after every 50 samples calculated at each slave. Robertson *et al.* [RCLL99] presented a centralised parallel version of *Radiance* whereby the calculated irradiance cache values are sent to a master process whenever a threshold is met. Each slave then collected the values deposited at the master by the other slaves.

2.3. Component-based rendering

Rendering has been divided into components on a number of occasions in order to solve the problem more efficiently. Wallace *et al.*'s [WCG87] multipass algorithm computed the diffuse component with a rendering pass and used a z-buffer algorithm for view dependent planar reflections. Ward *et al.*'s irradiance cache [WRC88] as described previously can be viewed as a component-based approach. Sillion and Puech [SP89] adapted a technique proposed by Wallace *et al.* [WCG87], using ray tracing for computing the specular component and the form factors of the non-planar objects, enabling multiple specular reflections. Shirley's [Shi90] used a three pass method: path tracing from the light source was used for caustics, soft indirect illumination was obtained through radiosity and stochastic ray tracing completed the rest of the components. Slusallek *et al.* [SSH*98] introduced lighting networks as a technique to render scenes based on combining the implementations of different rendering algorithms into a network, each algorithm computing different components of the rendering equation [Kaj86]. Other multi-pass algorithms that calculated components separately include Heckbert's [Hec90] and Chen *et al.*'s progressive multipass method [CRMT91]. Stokes *et al.* [SFWG04] presented a perceptual metric which predicted the importance of the components for a given scene, and used it to drive a path tracing renderer. The primary rays collected information about the scene and then used the perceptual metric to allocate the individual component calculations to resources based on their importance. Debattista *et al.* [DSSC05] have shown that component-based rendering can be used on a selective rendering framework to reduce computation times without compromising perceived visual quality and also on a predictive framework for time-constrained rendering.

3. Irradiance cache analysis

The irradiance cache improves rendering times by allowing the object space to be subsampled and the indirect diffuse values to be extrapolated if there are previously evaluated values within a user defined search radius. The number of extensively calculated indirect values is thus drastically reduced. However, the rendering times of images of acceptable quality from scenes with significant ratios of diffuse reflectors is still dominated by the diffuse interreflection component. The rendering time is thus strongly correlated with the number of diffuse indirect computations as shown in Figure 2, for the Kalabsha temple rendered on a single machine with different diffuse interreflections quality parameters. The ambient accuracy parameter of *Radiance*, which controls the acceptable search radius for extrapolation, varies from 0.3 to 0.1 (larger values corresponding to a larger search radius), thus resulting in more indirect diffuse calculations, or irradiance cache misses (#ICmisses). It can be seen that T_{amb}, time spent on indirect diffuse (ambient) calculations, increases linearly with the number of cache misses, while the time spent on other components (T_{notamb}) remains constant (around 7.3 seconds).

A parallel implementation of the irradiance cache on a distributed memory system might compromise efficiency due to replicated computations across the processors. Since each process has its own addressing space they will not be able to use each others' cached indirect values, thus increasing the aggregated time spent computing these (sum of T_{amb} across all PEs). A mechanism has to be provided to share cached values, traditional approaches being a centralised server or a broadcast mechanism as described in section 2.2. Higher sharing frequencies result in lower irradiance cache miss rates. However, even with very high sharing frequencies, there will always be a latency associated with sharing, meaning that a given value might not be avail-

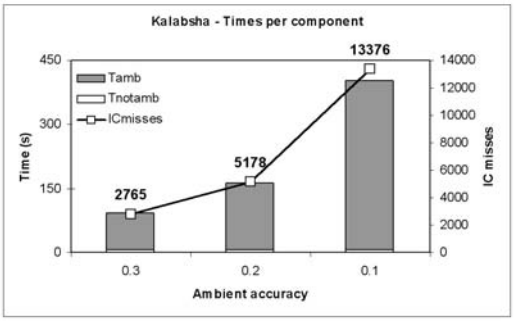


Figure 2: *Irradiance cache misses, T_{amb} and T_{notamb} (small constant time of around 7.3 seconds).*

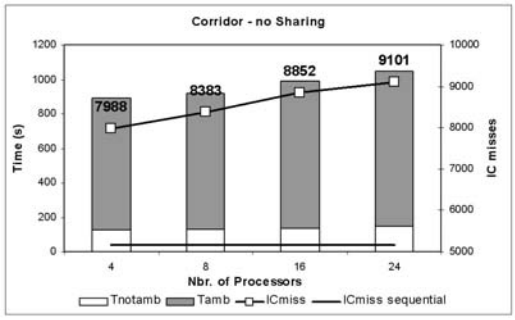
able when required even if it was already computed elsewhere. Additionally, sharing implies overheads. Communication bandwidth and processor computation time are spent preparing, transferring and reading the messages. Sharing overheads and latency increase with both the sharing frequency and the number of processors. Figure 3(a) shows, for the corridor scene, that irradiance cache misses are larger for the parallel no-sharing implementation than for the sequential single-processor version and that these increase with the number of processors, resulting in larger aggregated indirect diffuse calculation times; Figure 3(b) shows that sharing the irradiance cache values alleviates this problem, but it is still present and still depends on the number of processors.

Increasing the number of processors and the sharing frequency is desirable to reduce execution time, but these result in realization penalties that impact on efficiency. We propose to subdivide the processing elements onto two sets: one dedicated to compute only the indirect diffuse component and the other to compute the remaining components of the illumination model. By reducing the number of processors that contribute to the irradiance cache, the sharing frequency among these can be kept high, thus reducing both irradiance cache misses and sharing overheads. Cached values are shared with the remaining PEs with lower frequency, in order to reduce the number of indirect diffuse calculations requests forwarded to the specialised processors.

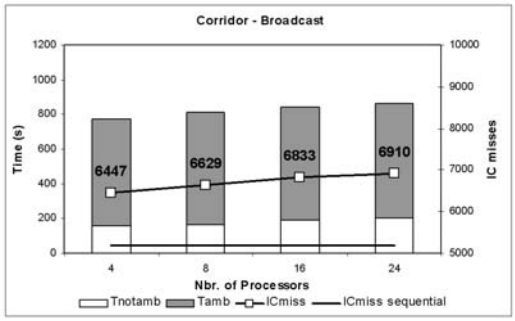
4. Traditional parallel irradiance cache approaches

In this section we outline our own implementation of the two traditional approaches of parallelising the irradiance cache. All our implementations use the Message Passing Interface (MPI). These implementations will prove useful to evaluate the performance of the new component-based approach.

Both approaches are based on an image plane decomposition of the workload, which is subdivided amongst processors by a Master Controller, MC, in a demand-driven fashion. Initial image tiles are sent to the processing elements, PEs, and when these are computed the resulting image tile is



(a) No Sharing



(b) Broadcast Sharing

Figure 3: *Aggregated rendering times and IC misses for the corridor scene.*

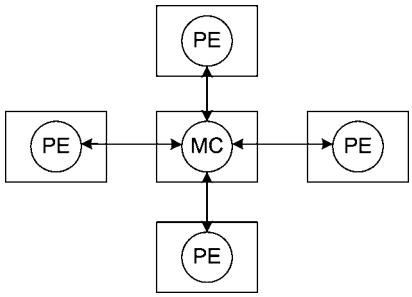


Figure 4: *Centralised Parallel Irradiance Cache.*

sent back to the MC, which forwards new work to the communicating PE. These implementations differ on how they share newly computed indirect diffuse values.

Robertson *et al.* [RCLL99] presented a centralised parallel version of *Radiance* where the Master Controller is used to exchange irradiance cache samples after these are computed by the PEs. In our implementation of this approach, each of the PEs computes indirect values and stores them in an outgoing buffer. Whenever a user-defined threshold, we term the *synchronisation threshold*, is reached, the buffer is transmitted to the MC. The MC maintains a list of all the samples sent to it and also a running status of which samples

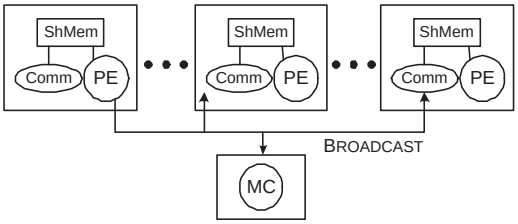


Figure 5: *Broadcast Parallel Irradiance Cache.*

each slave has been given so far. When a buffer is received by the MC, the set of new samples since the last communication with the sending PE is sent back. The software architecture of this approach is outlined in Figure 4. Although simple to implement, this approach might not scale well, due to the potential bottleneck on the central Master Controller.

Koholka *et al.* [KMG99] propose that whenever a group of irradiance cached values is computed, they are broadcast to other nodes, instead of being sent to the MC. Figure 5 illustrates the software architecture for this system. This approach is more complex than the previous method, since each PE is allowed to broadcast to every other slave. For our implementation, in order to maximise computation, each PE has a separate communicator process which listens for incoming irradiance cache samples. The communicator process communicates with the PE over shared memory. Whenever a set of samples is received, the communicator process stores the data in a shared memory area, where the computation process can collect it and insert it onto the local irradiance cache. This approach removes the contention on the centralised node but can still run into scalability issues due to the global broadcast operation.

5. Component-based parallel irradiance cache

In this section we present our novel parallel irradiance cache algorithm. We begin with the theory behind our work and subsequently describe the component-based parallel algorithm and implementation.

5.1. Rendering by components

In this section we describe rendering by components as used only in *Radiance*. This description can easily be extended to the general case [DSSC05].

The radiance at a pixel (x,y) in direction $-\Theta$ which intersects an object in the scene at point p is given by the rendering equation [Kaj86]:

$$L(x,y) =$$

$$L_e(p \rightarrow \Theta) + \int_{\Omega_\Psi} f_r(p, \Theta \leftrightarrow \Psi) \cos(N_p, \Psi) L(p \leftarrow \Psi) \delta_{\omega_\Psi}$$

The total set of Ψ directions distributed over the hemisphere Ω_Ψ can be conceptually subdivided into subsets of directions, commonly thought of as lighting components. In *Radiance* lighting calculations are subdivided into direct (L_d), indirect specular (including glossy, L_s) and indirect diffuse components (L_a) [War94]. The direct and specular components are removed from the integral by spawning rays into the appropriate directions, but for diffuse interreflections the integral must be approximated using Monte Carlo integration techniques. The previous equation becomes

$$L(x,y) = L_e(p \rightarrow \Theta) + L_d(p \rightarrow \Theta) + L_s(p \rightarrow \Theta) + L_a(p \rightarrow \Theta)$$

For the later component only diffuse interactions are computed, thus $f_r(p, \Theta \leftrightarrow \Psi) = k_a(p)$ and $L_a(p \rightarrow \Theta)$ is given by

$$L_a(p \rightarrow \Theta) = \int_{\Omega_{\Psi,a}} k_a(p) \cos(N_p, \Psi) L(p \leftarrow \Psi) \delta_{\omega_{\Psi,a}}$$

where directions included on direct and specular components are excluded from $\Omega_{\Psi,a}$. Using Monte Carlo integration:

$$L_a(p \rightarrow \Theta) \approx \frac{\pi k_a(p)}{N} \sum_{i=0}^{N-1} \cos(N_p, \Psi_i) L(p \leftarrow \Psi_i)$$

Indirect diffuse lighting computations are required not only for primary rays, but at every level of recursion b along the specular paths. Subscripted ordinal prefixes will be used to refer to points and coefficients at different levels of recursion along the specular path, ${}_1p$ and ${}_1\Theta$ referring to the primary ray intersection. To correctly weight the ambient contribution for pixel (x,y) at recursion level b, the specular coefficients ${}_jT = f_r({}_jp, {}_j\Theta \leftrightarrow {}_j\Psi) \cos(N_{jp}, {}_j\Psi)$ must be rippled down the path, resulting in

$$< L_a({}_bp \rightarrow {}_b\Theta) >=$$

$$\frac{\pi k_a({}_bp) \prod_j^{b-1} {}_jT}{N} \sum_{i=0}^{N-1} \cos(N_{bp}, {}_b\Psi_i) L({}_bp \leftarrow {}_b\Psi_i) \qquad (1)$$

5.2. Component-based approach

For our novel component-based approach the distributed processors are divided into two sub groups. The first group, the PAs, are dedicated to the indirect diffuse calculations (ambient calculations). The second group, the PRs, are responsible for the traditional rendering, except for indirect diffuse calculations. Figure 6 illustrates our system architecture. As in the case with the broadcast method, the PAs overlap communication and computation using distinct processes. The processes communicate using shared memory.

The PRs obtain work from the master controller, MC, using an image space demand-driven approach as described in

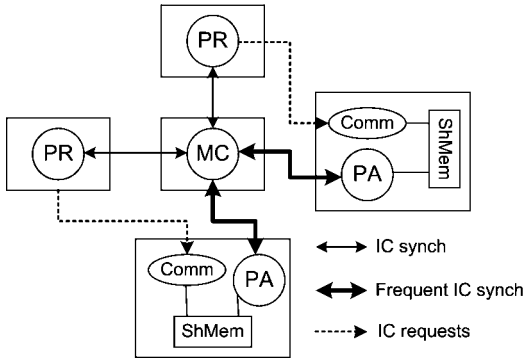


Figure 6: *Component-Based Parallel Irradiance Cache.*

Section 4. The PRs take on the role of intersecting and shading all forms of primary and secondary rays until an indirect diffuse calculation is required. At this point, the PR's local irradiance cache is consulted. If a cache miss occurs the essential attributes from the current ray being calculated, including pixel coordinates, intersection point, reference to intersected object, rippled coefficient ($_jT$, equation 1), *etc.*, are stored on an *outgoing indirect diffuse request buffer*. Whenever the outgoing buffer meets the *sending threshold*, it is sent to a particular PA, which is selected on a round robin basis for each PR. When the radiance of a pixel is finally calculated (this might be only partially computed since the indirect diffuse computation is migrated to one of the PAs), the result is stored in a buffer representing the image plane. Whenever the PR runs out of work, it requests a new task from the MC. For reasons which will become apparent below, unlike the approaches described in Section 4, the image plane buffer is only sent back at the end of the frame calculation. When all image tiles have been processed by the PRs, the MC enters a load balancing phase as described in section 5.3. The PRs also maintain a local irradiance cache. This is synchronised with the MC using a centralised approach. To reduce sharing overheads the synchronisation is less frequent than that of the PAs; it occurs only when asking for a new task from the MC. With time, the outgoing indirect diffuse calculations will decrease substantially due to this local irradiance cache.

Work is sent directly from the PRs to the PAs in the form of the *indirect diffuse request buffer*, which is reconstructed into rays on the receiving PA. The PA's function is then to calculate the indirect diffuse value from the intersection point stored in the ray. This involves shading and may also involve further recursive indirect calculations, including shadows, specular and indirect diffuse calculations. All these calculations are performed locally and the final calculated radiance is multiplied by the original $_jT$ coefficient and stored in the appropriate pixel coordinates on the image plane buffer. The irradiance cache values stored in the PAs are shared with the MC, following a centralised approach, much more frequently than PRs do. This ensures that the irradiance cache samples are readily available on the PAs when required. Synchronisation occurs whenever a PA meets the *synchronisation threshold*.

Similarly with the PRs, the image plane buffer is only sent back to the MC at the end of the frame computation, avoiding sending computed radiance values back as messages to the PRs' originally responsible for the pixel. This is possible due to the linear nature of geometric optics, which allows image plane composition by summing the different components' contributions for each pixel. This is a significant advantage, since it allows PRs to avoid the complex synchronisation issues required to await for and then store the final result for each pixel before sending the result back, improving computational performance. The MC composites all image plane buffers at the end of the frame to generate the final rendered image.

5.3. Load balancing

A major issue with partitioning the processors onto two subsets is how to balance the load between PAs and PRs. While a correctly chosen PA to PR ratio helps to minimise load imbalance, in reality optimal static load balance is impossible to achieve due both to the unpredictable nature of ray tracing and because PRs keep on forwarding requests up to the end of their assigned tasks and these requests take much longer to compute than the remaining components. In order to balance the system we allow PRs to change state and download work from PAs whenever the MC has run out of image tiles.

Every time a PA synchronises its irradiance cache with the MC, it also sends the number of pending requests on that PA's queue. Since this happens often, the MC has a precise image of the load on every PA. When a PR requests a new task and all image tiles have already been assigned, the MC selects the most loaded PA and signals it that some of its load should be sent to that PR. Upon reception of this signal, the PA's communicator process forwards to the PR a fraction of its load. It is the PA's communicator that decides how many pending requests to send to the PR, since it has access to the queue's current length. This process is repeated until all load has been processed.

At this load balancing stage all processors synchronise their local irradiance cache with the MC at the same frequency as the PAs, but in reality this rarely occurs because few new irradiance cache samples are generated.

6. Results

Results were obtained with a cluster of 12 machines each with two Intel Xeon processors at 3.2 GHz with 2 GB of memory under Linux. All the nodes are connected by a 1 Gbit switch. 2, 4, 8 and 12 nodes were used to run these experiments (4, 8, 16 and 24 processors) with an additional

node acting as the master controller. In order to demonstrate the potential of this approach still images were rendered from three selected scenes. The Kalabsha temple [SCM04], Figure 1(a) (resolution 672×512), and the corridor model, Figure 1(b) (resolution 512×512), were rendered with one ambient bounce and an ambient accuracy of 0.2. The modified Cornell box, Figure 1(c) (resolution 512×512), was rendered with an ambient accuracy of 0.1. Default *Radiance* parameters were used for all other settings.

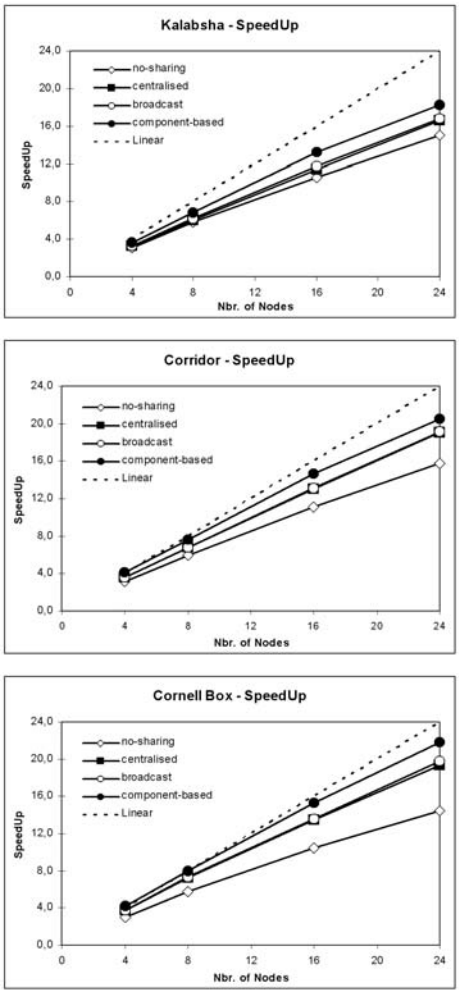


Figure 7: *Speed-up for the different scenes and algorithms.*

Plots of the achieved speedups can be seen in Figure 7. Results are shown for a parallel version that never shares any irradiance cache values (no sharing), the centralised, broadcast and the new component-based approach. Results reflect speedup when compared to the dedicated uniprocessor version of *Radiance* running under the same settings. The centralised and broadcast implementations synchronise the irradiance cache for every new 50 indirect samples, as suggested by Koholka *et al.* [KMG99]. For the component-based approach half the processors were allocated as PAs. These synchronise for every new 8 samples.

It is clear that the new algorithm outperforms the other algorithms in all cases. Speedup gain for each scene relatively to the second best algorithm is shown in Table 1. This is due to a huge reduction on irradiance cache misses, compared with the other parallel approaches, as illustrated in Figure 8 for the Kalabsha temple and 24 processors. This reduction is achieved by increasing the frequency of the irradiance cache sharing operation, while at the same time increasing its locality by restricting it to the PAs.

Scenes	Processors			
	4	8	16	24
Kalabsha	10.3%	10.7%	11.7%	8.8%
Corridor	15.5%	12.1%	11.9%	6.9%
Cornell	12.6%	8.4 %	13.2%	10.0%

Table 1: *Speedup gains relatively to second best algorithm*

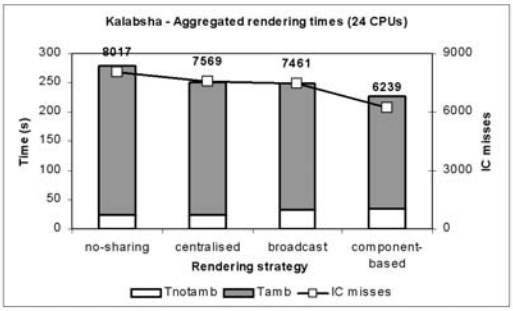


Figure 8: *Aggregated execution times and cache misses for different algorithms.*

With 4 processors the component-based also achieves super-linear speedup for the corridor (speedup=4.1, efficiency=103.1%) and Cornell (speedup=4.2, efficiency=105.7%) scenes . For a larger number of processors the speedup is noticeably close to linear; for the Cornell scene efficiency never falls below 90%, even with 24 processors. It is notable that super-linear speedup is never achieved by any of the other two renderers. This is a highly encouraging result indicating the new component-based renderer is more efficient even than the traditional uniprocessor approach in these cases. This is due to the fact that for all scenes the number of irradiance cache misses registered with the new approach is less than those registered with the sequential uniprocessor version, as shown in Figure 9 for the Cornell box. The reordering of indirect diffuse samples computation results on a higher hit ratio, compared to the sequential version where these are computed in raster order. The same effect is verified with the centralised and broadcast approaches, but to a lesser extent and insufficient

to achieve super-linear speedup. These results can also be attributed to making better use of the processors' physical caches, particularly since the PAs contribute to only the indirect calculations and the cache footprint does not become corrupted by the other components. These results indicate that the component-based approach may be suitable for shared memory implementations also.

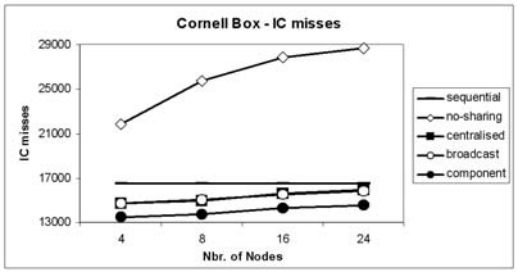


Figure 9: *Cache misses for the Cornell Box.*

With respect to assessing load balancing, Figure 10 shows the percentage of aggregated idle time, computed as the ratio of the sum of idle times across all processors and the aggregated rendering time, i.e., the rendering time multiplied by the number of processing elements. It can be seen that, up to 24 processors, idle times never go above 5.5%; we can conclude that the system is reasonably balanced, although the imbalance increases with the number of processors.

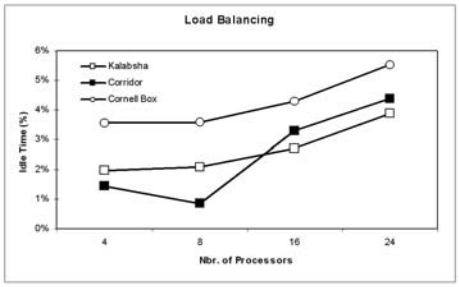


Figure 10: *Idle times due to load imbalance.*

7. Conclusion and future work

Despite the order of magnitude improvement the irradiance cache can provide over traditional distributed ray-tracing, the computational times for high quality graphics are still substantial. Parallel processing is one approach which can significantly reduce the overall computational time. However, the inherently sequential nature of how the irradiance cache is created and used has previously prevented the maximum benefit being gained from this structure in any parallel implementation. In this paper we have presented a novel method of dividing the work load amongst the processors of our cluster. This ensures that the computationally expensive part of establishing the irradiance cache is dealt with by a subset of dedicated processors, decreasing sharing overheads and latency. This allows a high sharing frequency, reducing the number of redundant indirect diffuse samples that need to be calculated. Such an approach has resulted in significant performance increases in our parallel implementation.

Future work will investigate how the algorithm performs within rendering of high-quality walkthroughs, in particular how the rendering cost could be dissipated from the first frames to ensure a smoother frame rate. Part of this work will also consider the use of the irradiance cache for dynamic scenes [TMD*04, SKDM05]. This approach could also be combined with selective rendering approaches, primarily the work of Yee *et al.* [YPG01] and Debattista *et al.* [DSSC05], who modulate their selective rendering efforts on the irradiance cache according to models of visual perception. The component-based model will be extended to handle radiance caching, thus allowing interpolation for glossy reflections, as proposed by [KGBP05].

References

[CDR02] CHALMERS A., DAVIS T., REINHARD E.: *Practical Parallel Rendering*. AK Peters Ltd, July 2002.

[CPC84] COOK R. L., PORTER T., CARPENTER L.: Distributed ray tracing. In *ACM SIGGRAPH '84* (1984), pp. 137–145.

[CRMT91] CHEN S. E., RUSHMEIER H. E., MILLER G., TURNER D.: A progressive multi-pass method for global illumination. In *ACM SIGGRAPH '91* (1991), pp. 165–174.

[DSSC05] DEBATTISTA K., SUNDSTEDT V., SANTOS L. P., CHALMERS A.: Selective component based rendering. In *3rd Int. Conf. on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia* (2005), ACM Press.

[GWS04] GUENTHER J., WALD I., SLUSALLEK P.: Realtime Caustics using Distributed Photon Mapping. In *Eurographics Symposium on Rendering* (2004).

[Hec90] HECKBERT P. S.: Adaptive radiosity textures for bidirectional ray tracing. In *ACM SIGGRAPH '90* (1990), pp. 145–154.

[Her04] HERY C.: Rendering evolution at industrial light & magic. In *Rendering Techniques* (2004), pp. 19–22.

[Jen01] JENSEN H. W.: *Realistic image synthesis using photon mapping*. A. K. Peters, Ltd., Natick, MA, USA, 2001.

[Kaj86] KAJIYA J. T.: The rendering equation. In *ACM SIGGRAPH '86* (1986), pp. 143–150.

[Kel97] KELLER A.: Instant radiosity. In *ACM SIGGRAPH '97* (1997), pp. 49–56.

[KGBP05] KRIVANEK J., GAUTRON P., BOUATOUCH K., PATTANAIK S.: Improved radiance gradient computation. In *SCCG '05: 21st Spring Conf. on Computer graphics* (New York, NY, USA, 2005), ACM Press, pp. 155–159.

[KMG99] KOHOLKA R., MAYER H., GOLLER A.: MPI-parallelized Radiance on SGI CoW and SMP. In *ParNum '99: 4th Int. ACPC Conf.* (1999), Springer-Verlag, pp. 549–558.

[LS98] LARSON G. W., SHAKESPEARE R.: *Rendering with radiance: the art and science of lighting visualization*. Morgan Kaufmann Publishers Inc., 1998.

[PMS*99] PARKER S., MARTIN W., SLOAN P.-P. J., SHIRLEY P., SMITS B., HANSEN C.: Interactive ray tracing. In *Symp. on Interactive 3D graphics* (1999), ACM Press, pp. 119–126.

[RCJ98] REINHARD E., CHALMERS A., JANSEN F. W.: Overview of parallel photo-realistic graphics. In *Eurographics '98 State of the Art Reports* (August 1998), Eurographics Association, pp. 1–25.

[RCLL99] ROBERTSON D., CAMPBELL K., LAU S., LIGOCKI T.: Parallelization of radiance for real time interactive lighting visualization walkthroughs. In *ACM/IEEE Supercomputing '99* (1999), ACM Press, p. 61.

[SCM04] SUNDSTEDT V., CHALMERS A., MARTINEZ P.: High fidelity reconstruction of the ancient egyptian temple of kalabsha. In *AFRIGRAPH 2004* (November 2004), ACM SIGGRAPH.

[SFWG04] STOKES W. A., FERWERDA J. A., WALTER B., GREENBERG D. P.: Perceptual illumination components: a new approach to efficient, high quality global illumination rendering. *ACM Trans. on Graphics 23*, 3 (2004), 742–749.

[Shi90] SHIRLEY P.: A ray tracing method for illumination calculation in diffuse-specular scenes. In *Graphics Interface '90* (Toronto, Ontario, 1990), Canadian Information Processing Society, pp. 205–12.

[SKDM05] SMKY M., KINUWAKI S.-I., DURIKOVIC R., MYSZKOWSKI K.: Temporally Coherent Irradiance Caching for High Quality Animation Rendering. In *EUROGRAPHICS 2005* (Dublin, Ireland, 2005), vol. 24 of *Computer Graphics Forum*, Blackwell.

[SP89] SILLION F., PUECH C.: A general two-pass method integrating specular and diffuse reflection. In *ACM SIGGRAPH '89* (1989), pp. 335–344.

[SSH*98] SLUSALLEK P., STAMMINGER M., HEIDRICH W., POPP J.-C., SEIDEL H.-P.: Composite lighting simulations with lighting networks. *IEEE Computer Graphics and Applications 18*, 2 (1998), 22–31.

[TMD*04] TAWARA T., MYSZKOWSKI K., DMITRIEV K., HAVRAN V., DAMEZ C., SEIDEL H.-P.: Exploiting temporal coherence in global illumination. In *SCCG '04: 20th Spring Conf. on Computer graphics* (New York, NY, USA, 2004), ACM Press, pp. 23–33.

[War94] WARD G. J.: The radiance lighting simulation and rendering system. In *ACM SIGGRAPH '94* (1994), pp. 459–472.

[WBWS01] WALD I., BENTHIN C., WAGNER M., SLUSALLEK P.: Interactive rendering with coherent ray tracing. In *Computer Graphics Forum* (2001), Chalmers A., Rhyne T.-M., (Eds.), vol. 20, Blackwell Publishers, Oxford, pp. 153–164.

[WCG87] WALLACE J. R., COHEN M. F., GREENBERG D. P.: A two-pass solution to the rendering equation: a synthesis of ray tracing and radiosity methods. In *ACM SIGGRAPH '87* (1987), pp. 311–320.

[WKB*02] WALD I., KOLLIG T., BENTHIN C., KELLER A., SLUSALLEK P.: Interactive global illumination using fast ray tracing. In *13th Eurographics Workshop on Rendering* (2002), pp. 15–24.

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. In *ACM SIGGRAPH '88* (1988), pp. 85–92.

[YPG01] YEE H., PATTANAIK S., GREENBERG D.: Spatiotemporal sensitivity and Visual Attention for efficient rendering of dynamic Environments. In *ACM Trans. on Computer Graphics* (2001), vol. 20, pp. 39–65.