Interner Bericht

Interactive Global Illumination

I. Wald, T. Kollig, C. Benthin, A. Keller, P. Slusallek

317/02

FACHBEREICH
INFORMATIK

UNIVERSITÄT
KAISERSLAUTERN

Postfach 3049 · D-67653 Kaiserslautern

Interactive Global Illumination

I. Wald, T. Kollig, C. Benthin, A. Keller, P. Slusallek

317/02

Universität Kaiserslautern
AG Numerische Algorithmen
Postfach 30 49
67653 Kaiserslautern
Germany

Januar 2002

Herausgeber: AG Numerische Algorithmen
Leiter: Professor Dr. S. Heinrich

Interactive Global Illumination

Ingo Wald[†] Thomas Kollig[‡] Carsten Benthin[†] Alexander Keller[‡] Philipp Slusallek[†]

[†]Saarland University [‡]Kaiserslautern University

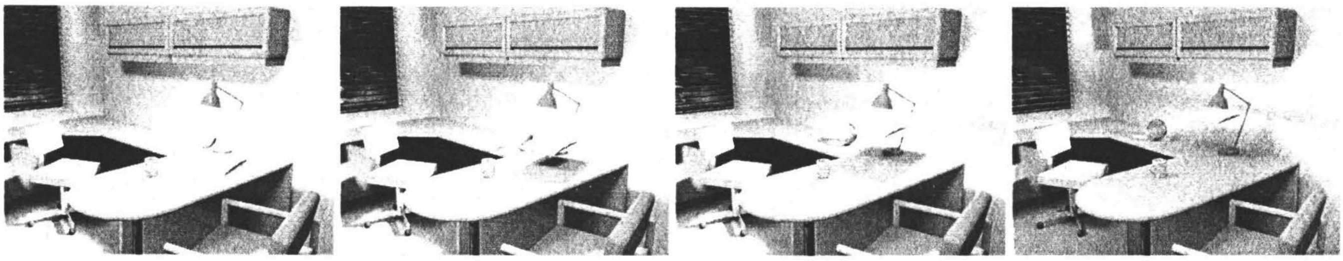


Figure 1: Scene with complete global illumination computed at 1 fps (640 × 480 on 16 dual-AthlonMP 1800+ PCs) while the book and glass ball are moved by a user. Note the changing soft shadows, the caustics from the glass ball, the reflections in the window, and especially the color bleeding effects on the walls due to indirect illumination.

Abstract

Interactive graphics has been limited to simple direct illumination that commonly results in an artificial appearance. A more realistic appearance by simulating global illumination effects has been too costly to compute at interactive rates.

In this paper we describe a new Monte Carlo-based global illumination algorithm. It achieves performance of up to 10 frames per second while arbitrary changes to the scene may be applied interactively. The performance is obtained through the effective use of a fast, distributed ray-tracing engine as well as a new interleaved sampling technique for parallel Monte Carlo simulation. A new filtering step in combination with correlated sampling avoids the disturbing noise artifacts common to Monte Carlo methods.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Raytracing, Animation, Color, shading, shadowing, and texture; I.6.8 [Simulation And Modeling]: Types of Simulation—Animation, Monte Carlo, Parallel, Visual; I.3.3 [Graphics Systems]: Picture/Image Generation—Display algorithms; I.3.2 [Graphics Systems]: Graphics Systems—Distributed/network graphics;

Keywords: Animation, Distributed Graphics, Illumination, Monte Carlo Techniques, Parallel Computing, Ray Tracing, Rendering Systems.

1 Introduction

With the availability of fast and inexpensive graphics hardware, interactive 3D graphics has become a mainstream feature of todays desktop and even notebook computers. All these systems are based on the rasterization pipeline. Recent hardware features such as multi texturing, vertex programming, and pixel shaders [Lindholm and Moreton 2001; NVIDIA 2001] have significantly increased the realism achievable with this environment.

However, these interactive systems are still limited to simple direct illumination from the light sources. With the pipeline rendering model it is impossible to compute the interaction between objects in the scene directly. Even simple shadows must be approximated in separate rendering passes for each light source.

Commonly, more complex lighting effects are precomputed offline with existing global illumination algorithms. They are expensive and slow, taking in the order of minutes to hours for a single update. Obviously, this works only for static illumination in equally static scenes and is insufficient for the highly dynamic environment of interactive applications.

The importance of realistic illumination and global illumination effects becomes apparent if we consider the significant efforts invested in lighting effects for the production of real and virtual movies. They employ a large staff of specially trained lighting artists to create the required atmosphere and mood of a scene through an appropriate illumination.

In real movie productions the artists have to work with the global effects of real light sources, while lighting artists in virtual productions have to simulate any global effects with local lighting. Only recently have global illumination algorithms been introduced in this environment, mainly in order to reduce time and money by automating some of the lighting effects. Previously global illumination was considered too inflexible and time consuming to be useful.

However, realistic lighting has a much wider range of application. It is instrumental in achieving realistic images of virtual objects, supporting the early design and prototyping phases in a production pipeline. Realistic lighting is particularly important for large, expensive projects as in the car and airplane industry but is equally applicable for projects from architecture, interior design, industrial design, and many others.

Algorithms for computing such realistic lighting often rely on ray tracing, which has long been well known for it high rendering times. Only but recently, research in faster and more efficient ray-tracing has drastically changed the environment in which global illumination operates. Even on commodity hardware, ray-tracing has been accelerated by more than an order of magnitude [Wald et al. 2001a; Wald et al. 2001b]. In addition, novel techniques allow for an efficient and scalable distribution of the computations over a number of client machines.

Since ray-tracing is at the core of most global illumination algorithms, one should suspect that global illumination algorithms should equally benefit from these developments. However, it turns out that fast ray-tracing implementations impose constraints that are incompatible with most existing global illumination algorithms (see Section 2.1).

1.1 Outline of the new Algorithm

In this paper we introduce a new, Monte Carlo global illumination algorithm that is specifically designed to work within the constraints of newly available distributed interactive ray tracers in order to achieve interactive performance on a small cluster of PCs.

Our algorithm efficiently simulates direct and mostly-diffuse interreflection as well as caustic effects, while allowing arbitrary and interactive changes to the scene. It avoids noise artifacts that are particularly visible and distracting in interactive applications.

Interactive performance is obtained through the effective use of a fast, distributed ray-tracing engine for computing the transport of light, as well as new interleaved sampling and filtering techniques for parallel Monte Carlo simulations. The idea of instant radiosity [Keller 1997] is used to compute a small number of point light sources by tracing particles from the light sources. Indirect illumination is then obtained by computing the direct illumination with shadows from this set of point light sources.

In addition illumination via specular paths is computed by shooting caustic photons [Jensen 2001] towards specular surfaces, extending their paths until diffuse surfaces are hit and storing the hits in a caustic photon map. All illumination from point light sources and caustic photons are recomputed for every frame.

We distribute the computation over a number of machines with a client/server approach by using interleaved sampling in the image plane. Based on a fixed pattern, samples of an image tile are computed by different machines with each machine using a different set of point light sources and caustic photons.

Finally, we apply filtering on the master machine to combine the results from neighboring pixels. This step is important for achieving sufficient image quality as it implicitly increases the number of point lights and caustic photons used at each pixel. We currently use a simple heuristic based on normals and distances for restricting the filter to a useful neighborhood.

1.2 Structure of the Paper

We start in Section 2 with an overview of the fast ray-tracing systems that have recently become available. In particular we discuss the constraints those systems impose on global illumination algorithms. In Section 3 we review previous work specifically with respect to these constraints. The details of the new global illumination algorithm are then discussed in Section 4 before presenting results in Section 5.

2 Fast Ray-Tracing

Ray-tracing is one of the oldest and most fundamental techniques used in computer graphics [Appel 1968; Whitted 1980; Cook et al. 1984]. In its most basic form it is used for computing the visibility along a ray or between two points. Most global illumination algorithms use ray-tracing in their core procedures to determine visibility or to compute the transport of light via particle propagation. Often most of their computation time is actually spent inside the ray-tracer.

Ray-tracing is also well-known for its long computation times. It requires traversing a ray through a precomputed index structure for locating geometry possibly intersecting the ray, computing the actual intersections, and finally evaluating some shader code at the intersection point. Each step involves significant computation and most applications require tracing up to several million rays. Consequently, those algorithms were limited to off-line computation, taking minutes to hours for computing a single image or global illumination solution.

Recently, ray-tracing has been optimized to deliver interactive performance on certain platforms. Muuss [Muuss and Lorenzo 1995; Muuss 1995] and Parker et al. [Parker et al. 1999b; Parker et al. 1999a; Parker et al. 1998] have shown that the inherent parallelism of ray-tracing allows one to efficiently scale to many processors on large and expensive supercomputers with shared memory. Exploiting the scalability of ray-tracing and combining it with low-level optimizations allowed them to achieve interactive frame rates for a full-featured ray-tracer, including shadows, reflections, different shader models, and non-polygonal primitives such as NURBS, isosurfaces, or CSG models.

Last year, Wald et al. [Wald et al. 2001a] have shown that interactive ray-tracing performance can also be obtained on inexpensive, off-the-shelf PCs. Their implementation is designed for good cache performance using optimized intersection and traversal algorithms and a careful layout and alignment of core data structures. A redesign of the core algorithms also allowed them to exploit commonly available processor features like prefetching, explicit cache management, and SIMD-instructions.

Together these techniques increased the performance of ray-tracing by more than an order of magnitude compared to other software ray-tracers [Wald et al. 2001a]. Because ray-tracing scales logarithmically with scene complexity, ray-tracing on a single CPU was able to even outperform the fastest graphics hardware for complex models and moderate resolutions.

In a related publication it was shown that ray-tracing scales well also in a distributed memory environment using commodity PCs and networks [Wald et al. 2001b]. Distributed computing is implemented in a client/server model and is mainly based on demand-loading and caching of scene geometry on client machines, as well as on load-balancing and hiding of network latencies through re-ordering of computations. It achieves interactive rendering performance even for scenes with tens of millions of triangles. Using more processors also allows one to use more expensive rendering techniques, such as reflections and shadows.

It is an obvious next step to use the fast ray-tracing engine to also speed up the previously slow global illumination algorithms that depend so heavily on ray-tracing. However, it turns out that this is not as simple as it seems at first: Most of these algorithms are incompatible with the requirements imposed by a fast ray-tracing system.

2.1 Constraints and Requirements

There is a large number of constraints imposed on any potential global illumination algorithm. In the following we briefly discuss each of these constraints.

2.1.1 Computational Constraints

Complexity. It has been shown that interactive ray-tracing can handle huge scenes with tens of millions of triangles efficiently [Wald et al. 2001b]. However, in most cases complex environments translate to complex illumination patterns that are significantly more costly to compute. While pure ray-tracing scales logarithmically with scene geometry, this is not the case for global illumination computations in general. This limits the complexity of scenes that we will be able to simulate interactively.

Performance. A target resolution of 640 × 480 pixels contains roughly 300,000 pixels. Furthermore, we assume that a single client processor can trace roughly 500,000 rays per second. Given a small network of PCs with 30 processor, e.g. 15 dual processor machines, we have a total theoretical budget of only 50 rays per pixel for estimating the global illumination for an image.

Parallelization. Due to price and availability considerations, we are targeting networks of inexpensive but fast PCs with standard network components. Due to this highly parallel environment, the global illumination algorithm must also run in parallel across a number of client machines. Furthermore, the ray-tracer schedules bundles of ray trees to be computed on each client. For best performance, the algorithms should offer enough independent tasks and these tasks should be organized similar to the ray scheduling pattern.

Other Costs. The underlying engine significantly speeds up ray-tracing compared to previous implementations. However, its speedup is limited to this algorithm. Other costs in a global illumination algorithms that have previously been dominated by ray-tracing can easily become the new bottleneck.

Communication. Commodity network technology, such as Fast-Ethernet or even Gigabit-Ethernet present significant hurdles for distributed computing. Compared to shared-memory systems, communication parameters differ by several orders of magnitude: bandwidth is low, measured in megabytes versus gigabytes per second, and latencies are high, measured in milliseconds versus fractions of microseconds.

Thus, a good algorithm must keep its bandwidth requirements within the limits of the network and must avoid introducing latencies. In particular, it must minimize synchronization across the network, which would result in costly round trip delays and in clients running idle.

Global Data. Many existing global illumination algorithms strongly depend on access to some global data structure. However, access to global data must be minimized for distributed tasks as it causes network delays and possible synchronization overhead to protect updates to the data.

Access to global data is less problematic before and after tasks are distributed to clients. In a client/server environment the global data can be maintained by the master and is then streamed to and from clients together with other task parameters.

Ideally, global data to be read by a client is transmitted to it with the initial task parameters. The task then performs its computations on the client without further communication. Global data to be written is then transmitted back to the master together with the other results. One must still be careful to avoid network latencies and bandwidth problems.

Amortization. Many existing algorithms perform lengthy precomputations before the first results are available. This processing is then amortized over remaining computations. Unfortunately, this strategy is inadequate for interactive applications, where the goal is to provide immediate feedback to the user. Preprocessing must be limited to a few milliseconds per frame. Furthermore, it must also be amortized over at most a few frames, as it might otherwise become obsolete due to interactive changes in the environment.

Accumulation. During interactive sessions many lighting parameters change constantly, making it difficult to accumulate and reuse previous results. However, this technique can be used to improve the quality of the global illumination solution in static situations.

Coherence. A fast ray-tracer depends significantly on coherent sets of rays to make good use of caches and for an efficient use of SIMD computations [Wald et al. 2001a]. An algorithm should send rays in coherent batches to achieve best performance.

2.1.2 Quality Constraints

Illumination Effects. Given current technology, it seems unrealistic to expect perfect results at interactive rates, yet. Therefore we focus on the major contributions of global illumination, such as direct and indirect illumination by point and area light sources, reflection and refraction, and direct caustics. Less attention is paid to less important effects like glossy reflection or caustics of higher order.

Display Quality. Operating at extremely low sampling rates, we have to deal with the resulting sampling artifacts. While some of these artifacts (e.g. high-frequency noise) are hardly perceivable in still images, they may become noticeable and highly disturbing in interactive environments. Therefore special care has to be taken of temporal artifacts.

Interactivity. We define interactive performance to be at least 1 global illumination solution per second. Since higher rates are very desirable, we are aiming for 4-5 frames per second.

3 Previous Work

The global illumination problem has been formularized by the radiance equation [Kajiya 1986].

Using finite element methods, increasingly complex algorithms have been developed to approximate the global solution of the radiance equation. In diffuse environments, radiosity methods [Cohen and Wallace 1993] were the first that allowed for interactive walkthroughs, but required extensive preprocessing and thus were available only for static scenes. Accounting for interactive changes by incremental updates [Drettakis and Sillion 1997; Granier and Drettakis 2001] forces expensive updates to global data structures and is difficult to parallelize.

The use of rasterization hardware allows for interactive display of finite element solutions. However, glossy and specular effects can only be approximated or must be added by a separate ray tracing pass [Stamminger et al. 2000]. Due to the underlying finite element solution, these approaches are not available at interactive rates.

Path tracing based algorithms [Cook et al. 1984; Kajiya 1986; Chen et al. 1991; Veach and Guibas 1994; Veach and Guibas 1997] correctly handle glossy and specular effects. The view dependency requires to recompute the solution for every frame. The typical discretization artifacts of finite element methods are replaced by less objectionable noise [Ramasubramanian et al. 1999], which however is difficult to handle over time. Reducing the noise to acceptable levels usually is obtained by increasing the sampling rate and results in frame rates that are far from interactive.

Walter et al. [Walter et al. 1999] achieved interactive frame rates by reducing the number of pixels computed in every frame. Results from previous frames are reused through image-based reprojection. It is most effective for costly, path tracing based algorithms. However, the approach results in significant rendering artifacts and is difficult to parallelize.

Path tracing based approaches can be supplemented by photon mapping [Jensen 2001]. This simple method of direct simulation of light results in biased solutions, but allows to efficiently render effects like caustics that may be difficult to generate with previous algorithms. Direct visualization of the photon map usually results in visible artifacts. A local smoothing or final gather pass can be used for removing these artifacts but is prohibitively expensive due to the large number of rays that must be traced.

Exploiting the local smoothness of the irradiance, an extrapolation scheme [Ward and Heckbert 1992] has been developed that considerably reduces the rendering time required by a local pass. For a parallel implementation global synchronization and communication are necessary to provide each processor with the extrapolation samples. Far too many of the expensive samples are concentrated around corners as illustrated in [Jensen 2001, p. 143] and the position of the samples is hardly predictable. This requires either dense initial samples and consequently is expensive or results in tremendous popping artifacts when changing geometry during interaction.

Instant radiosity [Keller 1997] allows for interactive radiosity without solution discretization: The lighting in a scene is approximated by point light sources generated by a quasi-random walk, and rasterization hardware is used for shadow computation. Arbitrary interactive changes to the environment were possible. However the large number of rendering passes required for a single frame limited interactivity to relatively simple environments.

4 Algorithm

This technical section introduces the new algorithm that meets the constraints (see Section 2.1) that were imposed by a low cost cluster of consumer PCs and overcomes the problems of previous work. For brevity of presentation space we assume familiarity with the radiance integral equation and refer to standard texts like e.g. [Cohen and Wallace 1993].

Similar to distribution ray-tracing [Cook et al. 1984] for each pixel an eye path is generated by a random walk. The scattered radiance $L(x,\omega)$ (for the selected symbols see Figure 2) at the endpoint x of the path in direction ω is approximated by

$$\begin{aligned} L(x,\omega) & \\ = \; & L_e(x,\omega) + \int_S V(y,x) f_r(\omega_{yx},x,\omega) L_{in}(y,x) \frac{\cos\theta_y \cos\theta_x}{|y-x|^2} dA(y) \\ \approx \; & L_e(x,\omega) + \sum_{j=1}^{M} V(y_j,x) f_r(\omega_{y_j x},x,\omega) L_j \frac{\cos\theta_{y_j} \cos\theta_x}{|y_j-x|^2} \\ & + \frac{1}{\pi r^2} \sum_{j=1}^{N} B_r(z_j,x) f_r(\omega_j,x,\omega) \Phi_j \quad , \end{aligned}$$

where $P := (y_j, L_j)_{j=1}^{M}$ is the set of point lights in y_j with radiance L_j [Keller 1997] and $C := (z_j, \omega_j, \Phi_j)_{j=1}^{N}$ is the set of caustic photons that are incident from direction ω_j in z_j with the flux Φ_j [Jensen 2001]. These sets have to be generated at least once per frame by random walks of fixed maximum path length. After this preprocessing step the scattered radiance is determined by only visibility tests $V(y_j,x)$ and photon queries, where $B_r(z_j,x)$ is 1 if $|z_j - x| \leq r$ and 0 else.

In comparison to bidirectional path tracing [Veach and Guibas 1994] the first sum uses only one technique to generate path space samples. For the majority of all path space samples this technique is best or at least sufficient to generate them. An exception are path space samples with a small distance $|y_j - x|$ or which are belonging to caustics. In order to avoid overmodulation the first group is handled in a biased way by just clipping the distance to a minimal value. Since samples of the second group cannot be generated by this technique their contribution is approximated by the second sum using photon mapping.

S	scene surface
A	area measure
L	scattered radiance
L_e	emitted radiance
L_{in}	incident radiance
f_r	bidirectional scattering distribution function
$V(y,x) \in \{0,1\}$	mutual visibility of y and x
θ	angle between incident direction and normal
ω_{yx}	direction from y to x

Figure 2: Selected Symbols.

For each pixel only one eye path is generated allowing for higher frame rates during interaction. The flickering of materials inherent with random walk simulation is reduced by splitting the eye path once at the first point of interaction with the scene. Then for each component of the bidirectional scattering distribution function a separate path is continued. Anti-aliasing is performed by accumulating the images over time, progressively improving image quality during times of no interaction.

In order to obtain interactive frame rates with the above algorithmic core on a cluster of PCs, the preprocessing must not block the clients and avoid multiple computation of identical results as far as possible, while further variance reduction is still needed to reduce noise artifacts and increase efficiency. The necessary improvements are discussed in the sequel.

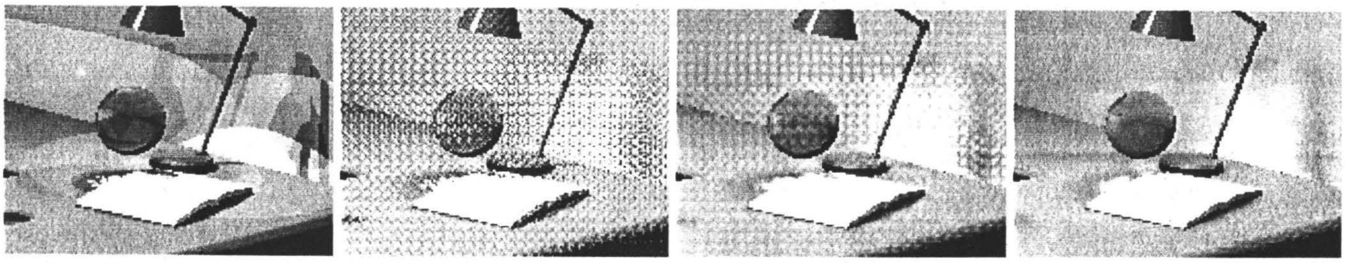
4.1 Fast Caustics

Shooting a sufficient number of photons is affordable in an interactive application, since the random walk simulations require only a small fraction of the total number of rays to be shot. However, the original photon map algorithms [Jensen 2001] for storing and querying photons are far too slow for interactive purposes: Rebuilding the 3d-tree for the photon map for every frame does not amortize, and the nearest neighbor queries are as costly as shooting several rays.

Therefore photon mapping is applied only to visualize caustics, where usually the photon density is rather high, and density estimation is applied with a fixed filter radius r. Assuming the photons to be stored in a 3-dimensional regular grid of resolution $2r$, only 8 voxels have to be looked up for a query. Since in practice only a few voxels will actually be occupied by caustic photons, a simple hashing scheme is used in order to avoid storing the complete grid. The photons that are potentially in the query ball are found almost instantaneously by 8 hash table lookups. Hashing and storing the photons in the hash table is almost negligible as compared to 3d-tree traversal and left-balancing of the 3d-tree as presented in the original work.

4.2 Interleaved Sampling

Generating a different set of point lights for each pixel is too costly, while using the same set causes aliasing artifacts (see Figure 3a). The same arguments hold for the direct visualization of the caustic photon map. In spite of the previous section's improvements, computing a sufficiently large photon map in parallel and merging the



a) no interleaved sampling no discontinuity buffer

b) 5 × 5 interleaved sampling no discontinuity buffer

c) 5 × 5 interleaved sampling 3 × 3 discontinuity buffer

d) 5 × 5 interleaved sampling 5 × 5 discontinuity buffer

Figure 3: Interleaved sampling and the discontinuity buffer: All close-ups have been rendered with the same number of rays apart from preprocessing. In a) only one set of point light sources and caustic photons is generated, while for b)-d) 25 independent such sets have been interleaved. Choosing the filter size appropriate to the interleaving factor completely removes the structured noise artifacts.

results would block the clients before actually rendering the frame and decrease the available network bandwidth.

However, generalizing interleaved sampling [Molnar 1991; Keller and Heidrich 2001] allows one to control the ratio of preprocessing cost and aliasing: Each pixel of a small $n \times m$ grid is assigned a different set P_k of light points and C_k of caustics photons ($1 \leq k \leq n \cdot m$). Padding this prototype over the whole image replaces aliasing artifacts by structured noise (see Figure 3b) while only a small number $n \cdot m$ of sets of light points and caustic photons has to be generated. Since interleaved sampling achieves a much better visual quality, the sets P_k and C_k can be chosen much smaller than P and C.

Each client computes the sets P_k of point lights and C_k of caustics photons by itself. Parallel tasks are assigned such that a client predominantly is processing tiles of equal identification k in order to allow for perfect caching P_k and C_k. Due to interleaved sampling synchronizing for global sets P and C (as opposed to [Christensen 2001]) is obsolete and in fact no network transfers are required.

4.3 The Discontinuity Buffer

The constraints of interactivity allow for only a small budget of rays to be shot, resulting in sets P_k and C_k of moderate size. Consequently the variance is rather high and has to be reduced in order to remove the noise artifacts. Taking into account that the irradiance is a piecewise smooth function the variance can be reduced efficiently by the discontinuity buffer.

For each pixel the server buffers the reflectance function accumulated up to the end point of the eye path, the distance to that point, the normal in that point, and the incident irradiance. The irradiance value consists of both the contribution by the point light sources P_k and the caustic photons C_k. Instead of just multiplying irradiance and reflectance function, the irradiance of the 8 neighboring pixels is considered, too: Local smoothness is detected by thresholding the difference of distances and the scalar product of the normals of the center pixel and each neighbor. If geometric continuity is detected, the irradiance of the neighboring pixel is added to the center pixel's irradiance. The final pixel color is determined by multiplying the accumulated irradiance with the reflectance function divided by the number of total irradiances included.

In the locally smooth case this procedure implicitly increases the irradiance sampling rate by a factor of 9, while at the same time reducing its variance by the same factor. Note that no additional rays have to be shot in order to obtain this huge reduction of noise, and that a generalization to larger than 3 × 3 filter kernels is straightforward. In the discontinuous case no smoothing is possible, however the remaining noise is superimposed on the discontinuities and such less perceivable [Ramasubramanian et al. 1999]. Since only the irradiance is blurred, texture details on the surface are perfectly reconstructed. Using the accumulation buffer method [Haeberli and Akeley 1990] in combination with the discontinuity buffer allows for oversampling in a straightforward way.

Including the direct illumination calculations into the discontinuity buffer averaging process, allows to drastically reduce the number of shadow rays to be shot, but slightly blurs the direct shadows. Similar to the irradiance caching methods, the detection of geometric discontinuities can fail. Then the same blurring artifacts become visible at e.g. slighty offset parallel planes.

Interleaved sampling and the discontinuity buffer perfectly complement each other (see Figure 3d) but require the filtering to be done on the server. In consequence an increased amount of data has to be sent to the server, which of course is quantized and compressed. Compared to irradiance caching the discontinuity buffer samples the space much more evenly and avoids the typical flickering artifacts encountered in dynamic scenes. In addition no communication is required to broadcast irradiance samples during rendering.

4.4 Minimal Randomization

The integrands in computer graphics are square-integrable, usually of high dimension and containing unknown discontinuities. Consequently the Monte Carlo method is appropriate for numerical integration. Since the pure Monte Carlo method is rather slow, we apply the much more efficient randomized quasi-Monte Carlo integration [Owen 1998]. This method of integration saves around 30% of computation time as compared to stratified sampling [Kollig and Keller 2001] and exposes much less variance, i.e. noise. The principle consists of using the estimator

$$\int_{(0,1)^s} f(x)dx \approx \frac{1}{r}\sum_{i=1}^{r}\frac{1}{n}\sum_{j=0}^{n-1} f(x_{i,j}),$$

where the samples $x_{i,j}$ for fixed i are of low discrepancy (for definitions see [Niederreiter 1992]) and for fixed j are independent sets of random realizations. Low discrepancy guarantees a much better uniform distribution of the samples than independent random samples can obtain. This implies good stratification properties that guarantee for faster convergence. On the other hand the independence makes the estimator a real Monte Carlo estimate that is valid for all square-integrable functions and in addition allows one to estimate the variance of the estimate.

The deterministic low discrepancy sequence of Sobol' can be generated in only a few lines of code [Press et al. 1992] in integer arithmetic. The points are randomized by just XOR-ing them with a random bit vector before floating point conversion [Friedel and

Keller 2001]. Taking the first n points a_j of the Sobol' sequence, r independent random realizations for the above scheme are obtained by $x_{i,j} := a_j \oplus b_i$, where b_i are r independent random bit vectors. Note that this randomization scheme preserves the good uniformity properties of the points.

In fact choosing only $r = 1$ randomized instances is sufficient to obtain a valid Monte Carlo estimator, while $r = 2$ instances already enable to estimate the error (see [Sobol' 1994]). In consequence variance and noise inherent with Monte Carlo methods can be almost avoided by this simple and minimal scheme of randomization. Tabulating the Sobol' sequence provides stratified sample generation at rates much faster than a pseudo-random number generator can achieve in combination with stratified sampling.

In the algorithm each identification k is assigned a subsequent subsequence of one instance of a randomized low discrepancy sequence. These samples are used for generating the sets P_k of point light sources and C_k of caustic photons. In case of geometric continuity the discontinuity buffer assembles the samples of neighboring pixels such joining different subsequences. Since these subsequences are part of the large sequence, the samples almost perfectly complement each other resulting in a superior convergence. In order to avoid the costly computation of high-dimensional low discrepancy sequences, padded replications sampling is used [Kollig and Keller 2001].

For the randomization each client needs an identical stream of only a few pseudo-random numbers per frame, which are created by the same pseudo-random number generator on each client. Thus any parallelization problems inherent with pseudo-random number generation are avoided and no communication is required during rendering.

5 Results and Discussion

For our experiments we have used a cluster of dual processor machines each equipped with two AMD AthlonMP 1800+ CPUs and 512 MB of RAM. All machines are connected to a fully switched 100 Mbit Ethernet. In order to handle the amount of pixel data sent to the server, a single Gigabit uplink from the switch was connected to the master machine that otherwise was identical to the clients.

In the following some example scenes are provided to show the performance, quality, and scalability of the proposed algorithm and its current implementation.

5.1 Example Scenes

Unfortunately it is difficult to visualize the temporal behavior of the approach in a printed publication. Therefore the accompanying video shows the examples captured in realtime from the screen of the master machine. The still images in this paper have been grabbed from the screen during interaction. The converged images are accumulated results of successive frames after interaction has stopped. This process usually takes about 1 or 2 seconds.

All scenes are rendered at video resolution of 640 × 480 pixels unless stated otherwise. A maximum path length of 4 is used for generating the point light sources. For all of the following examples 3 × 3 interleaved sampling has been used in combination with a 3 × 3 discontinuity buffer.

5.1.1 Simple Scenes with Caustics

The left image in Figure 4 shows a simple room lit by a single area light source located underneath the ceiling above the table, where a glass sphere casts a caustic. Global illumination is computed at 3.3 fps on 8 clients, while for each pixel 22 shadow rays are cast and 500 caustic photons are generated each frame. 4 of the 22 point light sources are located directly on the light source itself, while 18 are spread all over the scene.

The scene in the right image of Figure 4 contains two light sources with 5 samples each in addition to 20 indirect light samples. Roughly 1,500 photons per light source were used to represent the two caustics.

Temporal flickering is minimal in both scenes due to low complexity of the illumination, and shadows are smooth and clearly visible with all detail already in the dynamic scene. The dynamic and converged images only can be distinguished by the slightly better caustics and anti-aliasing due to accumulation.

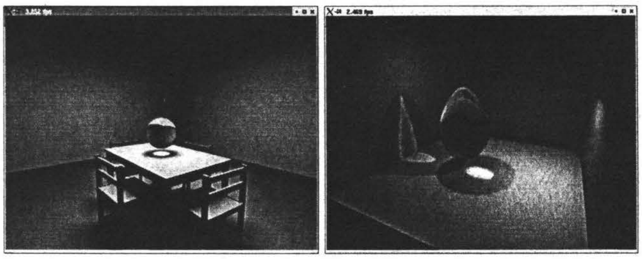


Figure 4: Two simple test scenes with a glass ball and a glass egg consisting of 800 and 4,000 triangles. These scenes render at 3.3 and 2.5 fps on 8 clients.

5.1.2 Invisible Date

The "Invisible Date" scene as shown in Figure 5 contains 9,000 triangles. It is lit mostly indirectly from the two lamps pointing towards the ceiling. No direct illumination reaches the furniture and the reflective floor. A glass sphere is floating below the ceiling casting caustics on the wall.

This scene nicely demonstrates the combination of specular illumination effects due to ray-tracing reflections and indirect illumination computed with the new algorithm. It uses 4 direct samples, 9 indirect samples, and 500 photons per light source. The scene renders at 2.6 fps on 8 clients.

Smooth penumbras are produced by the shelf and other objects. Some flickering is visible due to the low number of indirect samples. This could be avoided by taking more samples at the cost of lower frame rate.

Although the eye path is split once at the first point of interaction, subsequent path segments randomly select a component of the bidirectional scattering distribution function. This is the reason for some small flickering artifacts visible for the teapot and the cups on the table during interaction.

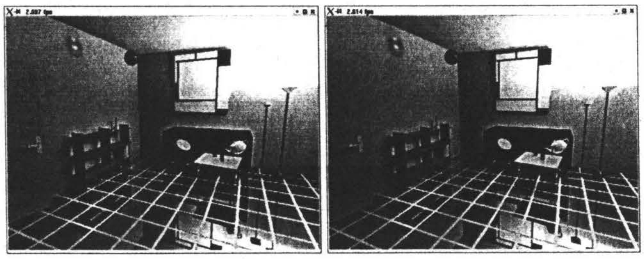


Figure 5: The Invisible Date scene containing mostly indirect illumination from two lights pointing towards the ceiling. It shows a combination of ray-traced reflections and smooth indirect illumination. Little differences can be observed between the dynamic and the converged image on the left and right, respectively.

5.1.3 Office

The office scene in Figure 6 contains 34,000 triangles. The lighting conditions are quite difficult due to significant occlusion. The complex indirect illumination is particularly visible in the bottom row of Figure 6, where the strongly illuminated book acts as a secondary area light source causing additional smooth shadows on the wall. In addition color bleeding is clearly visible on the wall in the upper row of images.

Global illumination is computed with 4 direct, 18 indirect, and 1,000 photons per light source, which results in a frame rate of 2.2 fps on 8 clients for both views. The sharp shadow boundaries from the point light sources visible in the non-converged images on the left already indicate noticeable flickering. While this is disturbing during movements it helps in locating important lighting effects. The illumination, however, quickly converges as soon as the movement stops and results in a highly detailed and smooth illumination pattern.

The effects of insufficient filtering of interleaved sampling are clearly visible at curved surfaces. They are resolved in the converged solution on the right.

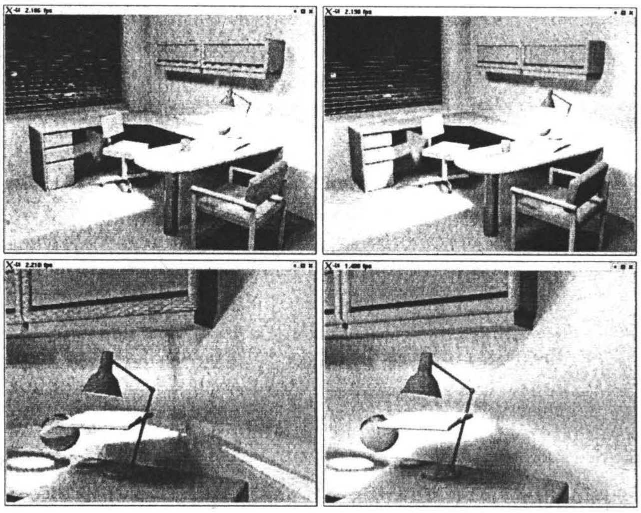


Figure 6: Two views of the office environment. The scene is illuminated by two area lights at the ceiling and a desk lamp. The images in the bottom row show a close-up of the detailed illumination patterns caused by the book under the desk lamp. While there is significant flickering during movements, the solution quickly converges to the smooth images on the right. Both views render at 2.2 frames per second on 8 clients.

5.1.4 Conference Room

The conference room test scene from Figure 7 contains illumination from 104 area light sources and consists of 290,000 triangles. Due to its material and geometrical complexity the scene renders only at 1.7 fps using 12 clients.

Subsampling is used to limit the number of shadow rays cast for each pixel. In this case, a total of 5 light sources are determined and sampled once. For the indirect illumination 20 indirect samples were used. Due to the well distributed location of the lights and the filtering by the discontinuity buffer, almost no flickering is visible. However, the limitations of the discontinuity buffer again become visible on curved surfaces. Here an improvement of the filtering is necessary. Finally the accumulated solution again is free from any artifacts.

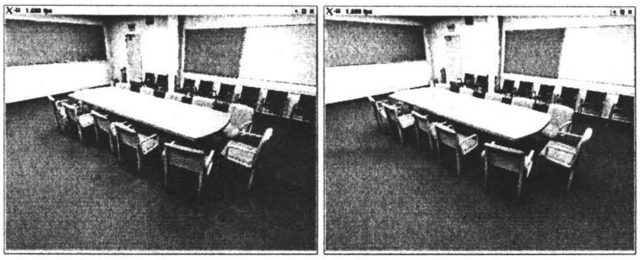


Figure 7: Conference room scene with 104 light sources and detailed shadows cast by the chairs. The scene renders at 1.7 fps on 12 clients and hardly shows any flickering.

5.1.5 Dynamic Environments

Interactive changes to the scenes (see Figure 1) are trivially handled by the global illumination algorithm, because the underlying ray-tracing engine handles changing geometry transparently to the application.

5.2 Simulation Quality

The combination of the idea of instant radiosity, interleaved sampling, the discontinuity buffer, and correlated sampling by randomized quasi-Monte Carlo integration allows our system to achieve relatively smooth image quality with very few samples.

The algorithm can be controlled by a small set of intuitive parameters: the number of point light sources, the number of caustic photons, and the filter size of the discontinuity buffer that automatically determines an interleaved sampling pattern of same size. The user can easily trade off rendering speed for image quality by adjusting these parameters and gets immediate feedback by the interactive system.

Flickering due to discontinuous changes in the illumination can become particularly apparent when successive frames are illuminated by different point light sources. As shown in Section 5.1, accumulating successive frames efficiently smooths out these discontinuities. As a consequence flickering can be reduced drastically by including moderate oversampling during interaction, however, at a reduced frame rate.

5.3 Scalability

The client/server concept of our global illumination algorithm relies on non-blocking clients. Thus the scalability of the system depends on the ratio of the overhead and work done by the clients, on the ability of the server to schedule tasks and to process their results, and on the bandwidth of the network.

The number of rays traced for a preprocess is mainly determined by the number of caustic photons and is usually small compared to the total number of shadow rays. If possible, tiles with the same identification k are scheduled to the same client. Due to different loads on the clients and in order to hide latencies, the ideal distribution cannot always be maintained. Our experiments show that on the average the clients have to preprocess data for roughly two different identifications. Consequently the ratio of the overhead and work is small and hardly influences scalability.

The main workload of the server consists of handling large amounts of pixel data and performing the filtering by the discontinuity buffer. Since these computations require information from adjacent pixels computed by different clients, they have to be performed on the server. Consequently frame rate and resolution are restricted by the performance of the server.

In a similar way the limited network bandwidth restricts the frame rate and resolution.

As expected, Table 1 shows that the algorithm scales almost perfectly up to 16 clients. The maximum performance of our server is restricted to process around 1.5 million pixels per second. This corresponds to roughly 5 frames per second at 640 × 480. Other than waiting for faster hardware this bottleneck can be avoided by distributing the server computations. This can be done easily, but introduces additional latency. Reducing the image resolution allows to easily scale this maximum performance, reaching more than 10 frames per second at 400 × 300 pixels.

Nevertheless there are no restrictions imposed on the scalability with respect to image quality: Increasing the number of rays obviously improves image quality. If the number of clients is increased by the same factor, still the same number of tasks is scheduled and the same amount of pixel data is transferred and processed. Only little overhead is introduced resulting in almost identical frame rates at unchanged resolution.

Number of clients	1	2	4	8	16
Room with table	0.4	0.8	1.6	3.2	5.3
Room with egg	0.3	0.7	1.4	2.7	5.4
Office	0.2	0.3	0.6	1.2	2.4

Table 1: The algorithm almost perfectly scales over the range of available clients. For frame rates above 5 fps the server workload limits performance.

6 Conclusion

We have shown that - contrary to general opinion - interactive global illumination is indeed feasible, and can even be realized on a low cost cluster of consumer PC hardware. Running on 8 to 16 cluster nodes, our system delivers interactive global illumination of up to 5 frames per second at a resolution of 640 × 480 for scenes ranging from simple environments to complex geometry containing dozens of light sources. All lighting is completely recomputed every frame, allowing one to interactively change rendering parameters, material properties, lighting and even geometry. High quality global illumination is obtained in only seconds.

All this has been achieved by analyzing the drawbacks of previous approaches and by designing our system specifically to match the underlying framework of a fast, scalable ray-tracing engine. There is no communication or synchronization between different tasks distributed among the clients and only little overhead calculations. These are important reasons for its good scalability.

Besides including arbitrary physical surface properties and volumetric effects, future work will concentrate on even more exploiting temporal coherence. Since most of the dynamic changes are continuous, information can be used for predictions for unbiased importance sampling. Furthermore the majority of rays shot are shadow rays towards point light sources, which offer a high degree of coherence. For this special problem, tracing the shadow rays can be made much more efficient by tracing bundles of rays benefitting from spatial coherence [Lukaszewski 2001]. Such speed improvements directly affect the rendering speed of our system.

Acknowledgments

Thomas Kollig has been supported by the Stiftung Rheinland-Pfalz für Innovation. This work was supported by Advanced Micro Devices (AMD).

References

APPEL, A. 1968. Some Techniques for Shading Machine Renderings of Solids. *SJCC*, 27–45.

CHEN, S., RUSHMEIER, H., MILLER, G., AND TURNER, D. 1991. A progressive Multi-Pass Method for Global Illumination. In *Computer Graphics (SIGGRAPH 91 Conference Proceedings)*, 165 – 174.

CHRISTENSEN, P. 2001. Photon Mapping Tricks. In *ACM SIGGRAPH Course Notes, Course 38: A Practical Guide to Global Illumintion using Photon Mapping*. ACM SIGGRAPH, 91–115.

COHEN, M., AND WALLACE, J. 1993. *Radiosity and Realistic Image Synthesis*. Academic Press Professional, Cambridge.

COOK, R., PORTER, T., AND CARPENTER, L. 1984. Distributed Ray Tracing. In *Computer Graphics (SIGGRAPH 84 Conference Proceedings)*, 137–145.

DRETTAKIS, G., AND SILLION, F. 1997. Interactive Update of Global Illumination Using A Line-Space Hierarchy. In *Proceedings of SIGGRAPH 97*, ACM SIGGRAPH / Addison Wesley.

FRIEDEL, I., AND KELLER, A. 2001. Fast generation of randomized low-discrepancy point sets. In *Monte Carlo and Quasi-Monte Carlo Methods 2000*, H. Niederreiter, K. Fang, and F. Hickernell, Eds. Springer, 257–273.

GRANIER, X., AND DRETTAKIS, G. 2001. Incremental Updates for Rapid Glossy Global Illumination. In *Computer Graphics Forum*, Blackwell Publishers, vol. 20, 268–277.

HAEBERLI, P., AND AKELEY, K. 1990. The Accumulation Buffer: Hardware Support for High-Quality Rendering. In *Computer Graphics (SIGGRAPH 90 Conference Proceedings)*, 309–318.

JENSEN, H. 2001. *Realistic Image Synthesis Using Photon Mapping*. AK Peters.

KAJIYA, J. 1986. The Rendering Equation. In *Computer Graphics (SIGGRAPH 86 Conference Proceedings)*, 143–150.

KELLER, A., AND HEIDRICH, W. 2001. Interleaved Sampling. In *Rendering Techniques 2001 (Proc. 12th Eurographics Workshop on Rendering)*, Springer, K. Myszkowski and S. Gortler, Eds., 269–276.

KELLER, A. 1997. Instant Radiosity. In *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, 49–56.

KOLLIG, T., AND KELLER, A. 2001. Efficient bidirectional path tracing by randomized quasi-Monte Carlo integration. In *Monte Carlo and Quasi-Monte Carlo Methods 2000*, H. Niederreiter, K. Fang, and F. Hickernell, Eds. Springer, 290–305.

LINDHOLM, E., AND MORETON, M. K. H. 2001. A user-programmable vertex engine. In *Computer Graphics (Proc. of Siggraph)*, 149–159.

LUKASZEWSKI, A. 2001. Exploiting coherence of shadow rays. In *Proceedings of AFRIGRAPH*, ACM SIGGRAPH, 147–150.

MOLNAR, S. 1991. Efficient Supersampling Antialiasing for High-Performance Architectures. Tech. Rep. TR91-023, The University of Noth Carolina at Chapel Hill, April.

MUUSS, M. J., AND LORENZO, M. 1995. High-resolution interactive multispectral missile sensor simulation for atr and dis. In *Proceedings of BRL-CAD Symposium '95*.

MUUSS, M. J. 1995. Towards real-time ray-tracing of combinatorial solid geometric models. In *Proceedings of BRL-CAD Symposium '95*.

NIEDERREITER, H. 1992. *Random Number Generation and Quasi-Monte Carlo Methods*. SIAM, Pennsylvania.

NVIDIA. 2001. *NVIDIA OpenGL Extension Specifications*. NVIDIA, March.

OWEN, A. 1998. Monte Carlo Extension of Quasi-Monte Carlo. In *Winter Simulation Conference*, IEEE Press, 571–577.

PARKER, S., SHIRLEY, P., LIVNAT, Y., HANSEN, C., AND SLOAN, P. P. 1998. Interactive ray tracing for isosurface rendering. In *IEEE Visualization '98*, 233–238.