EUROGRAPHICS 2003 / P. Brunet and D. Fellner
(Guest Editors)

Volume 22 (*2003*), *Number 3*

A Scalable Approach to Interactive Global Illumination

Carsten Benthin, Ingo Wald, Philipp Slusallek

{benthin,wald,slusallek}@graphics.cs.uni-sb.de
Computer Graphics
Saarland University

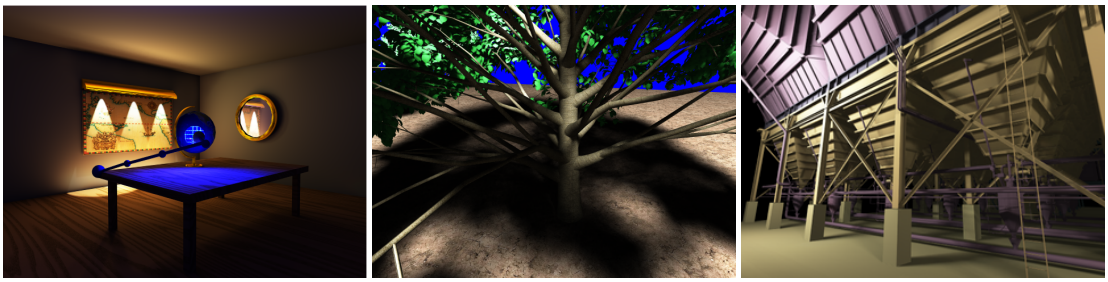

Figure 1: *Examples with detailed global illumination effects fully recomputed at interactive rates on a cluster of 24 dual PCs. Left: Room with a globe and an animated light source causing quick changes in indirect illumination rendered at 4.5 fps. The lamp currently illuminates the ocean giving the front of the room a bluish color. Center: Complex shadow of an entire maple tree with 1.5 million triangles at 4–6 fps. Right: Global illumination in a 50 million polygon power plant model running at 2 fps.*

Abstract

The addition of global illumination can dramatically increase the realism achievable when rendering virtual environments. In particular with interactive applications we expect the environment to reflect changes in the scene due to global lighting effects instead of it being just a static backdrop. However, a sufficiently fast and accurate computation of global illumination at interactive rates has been difficult even with recent approaches based on realtime ray tracing.

In this paper we present a highly scalable approach to interactive global illumination. It fully recomputes a high-quality solution for each frame and thus offers immediate feedback even for dynamic scenes, achieving more than 20 fps for simple scenes. Compared to previous systems we increased the raw performance by a factor of up to eight and removed the bottlenecks that were limiting scalability. The system now scales linearly in quality and available computing resources, tested with up to 48 CPUs in a commodity PC-cluster. Due to its logarithmic scaling property with respect to scene complexity it even supports lighting simulation in complex scenes with more than 50 million triangles. This scalability allows applications to perform flexible performance trade-offs. We also argue that the realism achievable through interactive global illumination will make it a standard feature of future 3D graphics systems once the required computing resources are readily available.

1. Introduction

Global illumination effects contribute significantly to perception of "realism" in virtual environments. They provide the often subtle but important cues that have been missing in interactive computer graphics resulting in that "artificial" look often attributed to computer generated images. In traditional computer graphics the environment is completely unaffected from any changes to objects in a scene. Global illumination not only provides realistic illumination

in the scene, it also causes the environment to "reflect" such changes to geometry, materials, or lighting. The environment is no longer a static backdrop but participates in the user's activities — just as in our every day experience.

In the past such realism was only achievable with heavy manual tuning of scenes, e.g. using precomputed lighting simulations and dynamic light maps. However, such tuning is only economically feasible for high-volume applications with predefined scenes such as computer games. Other interactive application scenarios including virtual reality, architectural walkthroughs, product design, and visualization cannot currently achieve the same realism.

If we want to provide the same realism for all 3D graphics applications we need a global illumination system that provides immediate feedback to the user and is sufficiently accurate for capturing the often subtle lighting effects. Global illumination requires complex computations and has been slow and far from interactive in the past, often taking minutes to hours even for a single image of a diffuse environment. Several attempts to achieve interactive performance have been made [5, 2, 3] but were usually held back by the lack of ray tracing performance.

An alternative is to incrementally compute the global illumination results and display approximate information during the computation [22, 7, 4, 17]. While these approaches achieve interactive rendering performance, full global illumination updates are still slow and intermediate results often show strong artifacts.

Realtime ray tracing has been a promising development for interactive global illumination as it offers the fundamental functionality needed by any such algorithm. Interactive ray tracing has first been explored by Muuss et al. [13] and Parker et al. [14] using massive parallelization on shared-memory supercomputers. More recently Wald et al. have shown that interactive ray tracing performance can also be achieved on small clusters of commodity PCs [19, 21].

In order to achieve interactive ray tracing on the desktop some form of hardware support will be inevitable and first research results are promising. Purcell et al. [15] showed how the core ray tracing computations can be mapped to pixel shaders in programmable graphics hardware using a stream computing abstraction. In addition, Schmittler et al. [16] demonstrated that a scalable custom hardware solution for ray tracing would be able to deliver realtime performance at comparable hardware costs.

A first implementation of interactive global illumination was presented by Wald et al. [20] and demonstrated interactive performance at up to 5 fps at reasonable image quality. However, the approach was still limited regarding frame rate and scalability.

In this paper we present a new approach that is based on the same algorithms but avoids these limitations by removing all the scalability issues and significantly improving basic performance. We use stream computing to further exploit the coherence in the algorithm. As a consequence we are able to provide significantly higher frame rates while simultaneously achieving superior illumination accuracy and image quality.

We start the presentation with a brief review and analysis of the previous interactive global illumination approach [20] in Section 2. Section 3 presents our new approach to scalable interactive global illumination with results and a comparison following in Section 4.

2. Previous Approach

Building on their previous work of building a fast ray tracing engine [19, 21] Wald et al. used this technology to also accelerate lighting simulation and presented the first system for interactive global illumination (IGI) [20]. The approach uses a modified *instant radiosity* algorithm [10], which is accelerated by *interleaved sampling* [11] and uses the *discontinuity buffer* [11] for achieving almost perfectly smooth illumination results. This choice of algorithms allowed to take maximum advantage of the increased ray tracing speed. In contrast, many traditional global illumination algorithms contain bottlenecks that prohibit their parallel and distributed execution.

2.1. Algorithm and Implementation

With the *instant radiosity* algorithm, the illumination in a scene is approximated by a small number of N (e.g. 20-30) *virtual point lights* (VPLs). These VPLs are generated in a preprocessing step by performing random walks of light particles from the light sources into the scene, thus approximating the light transport. VPLs are placed at the hit points of the random walk with their power determined by the reflection along the path and the local reflection coefficient.

The scene is then illuminated from these VPLs, which are described by their position x_i, the surface normal $n_i(x_i)$, and their emitted radiance L_i. The irradiance at any given surface point x with normal n is then approximated by summing over the contributions from all VPLs:

$$E(x) = \sum_{i=0}^{N-1} V(x,x_i)\frac{\cos\theta_x \cos\theta_i}{||x-x_i||^2} L_i, \qquad (1)$$

where $V(x,y)$ specifies the visibility between two points x and y, and θ_x, θ_i are the angles between the normal vectors n_x, n_i and the vector $x - x_i$, respectively.

The approach is ideally suited for interactive use: For each frame we can obtain a completely new solution by precomputing just a few dozens of VPLs, which requires shooting only a few hundred rays. The final illumination computation requires tracing many coherent shadow rays but this maps well to a distributed and parallel implementation due to its inherent parallelism and the good scaling properties of ray tracing with respect to scene complexity and computing resources.

The method works best in diffuse environments with little occlusion, but quality drops only slowly as non-diffuse effects are added and occlusion increases moderately (see below). Because all computations are performed on a per pixel basis, the algorithm provides smooth results and does not suffer from the well-known meshing problems of finite element approaches. Because it discretizes illumination sources instead of receivers, discretization artifacts are limited to lighting singularities that can easily be dealt with by capping the quadratic distance attenuation factor. This however introduces some bias.

The previous approach increased image quality by roughly an order of magnitude by combining interleaved sampling [11] with discontinuity buffering [11]. Interleaved sampling is used to increase the set of irradiance samples used for each pixel by assigning different sets of VPLs for each pixel of a 3×3 tiling of the image plane. This however would result in highly visible structured noise artifacts that can be removed by the discontinuity buffer.

The discontinuity buffer [11] trades off spatial resolution of illumination for higher quality lighting simulation by filtering irradiance across a 3×3 neighborhood of pixels. Because the filter support exactly matches the interleaved sampling pattern the typical noise of stochastic sampling (in this case Randomized Quasi-Monte-Carlo sampling [12] of light sources) is completely removed. However, filtering must be limited to a meaningful neighborhood by weighting filter coefficients based on differences between normals and positions of the irradiance samples. The combination of these two methods allows us to increase image quality by using nine times as many VPLs at almost the same cost.

The indirect illumination computed this way was easily augmented by specular effects like reflections and refractions due to the underlying ray tracing system. Finally, caustics have been added by using a simplified version of photon mapping [9, 20].

Like the original system, our system is built on top of a realtime ray tracing system running on a cluster of commodity PCs [18]. It client/server architecture and its distributed computing imposed some interesting challenges on efficient global illumination implementations. In particular the small bandwidth and high communication latency have to be taken into account.

The distribution architecture follows the common approach with a demand-driven job distribution based on image tiles, where the server is responsible for scheduling these image tiles across the clients. After a client has finished rendering its tile the pixels color values are sent back to the server for display. Clients and server are interconnected via an off-the-shelf Ethernet network. For more details on the parallel implementation see [18, 21].

2.2. Analysis

The above system for interactive global illumination already achieved impressive results but was still limited by a number of performance and scalability issues that we analyze in greater detail below. This analysis motivates the improvements for the new system described in the next section.

Probably the main drawback of the previous system was due to performing many computations on the server, including the discontinuity filtering. The design of filtering on the server requires sending additional data to the server, such as irradiance, normal, and depth. Even though the data had been quantized and compressed, the increased bandwidth limited performance even with a gigabit network. Even worse, filtering on the server required additional computation for (de-)quantizing and (de-)compression, did not scale with the number of clients, and had relatively poor performance compared to ray tracing due to its high memory bandwidth requirements on the server.

The main lesson taken from recent interactive ray tracing system [19] is that coherence is the key for good ray tracing performance. The design of the previous system already took this into account by shooting shadow rays in a coherent order. However, these rays were still computed individually and where neither traced in packets nor by using the SIMD instructions offered on modern processors [19, 8, 1]. This severely limited the raw performance of the approach. SIMD processing alone offers a performance increase by at least a factor of two as shown later in Section 4.

The above issues placed strict limits on the performance of the system and the quality of its images, in particular during interaction. The number of VPLs had to be low for achieving reasonable frame rates. This resulted in rather low image quality but still allowed a rough estimate of the overall illumination in the scene during interaction. However, good image quality that provided the necessary illumination details without artifacts was only obtained in static situations. In this case the display was gradually improved by adding the illumination of more and more VPLs.

While the underlying ray tracing system did support arbitrary programmable shaders, the previous IGI system was limited to a simple shading model, supporting only purely diffuse and perfectly specular effects. Because of a missing framework for efficiently handling programmable shaders in previous IGI system, no advanced texturing or procedural effects were supported.

3. A Fast and Scalable Approach

The new high-performance and scalable system for interactive global illumination keeps the core algorithms of the previous system including the use of Randomized Quasi Monte Carlo techniques [11, 12]. Our modifications mainly address improvements in performance, scalability, and consequently image quality.

Scene \ RT-Technique Shading	SSE none	SSE simple	non-SSE simple
Shirley-6 (static)	3.7	1.55	0.9
Shirley-6 (dynamic)	2.54	1.29	0.7
Conference (static)	2.5	1.25	0.77
Conference (dynamic)	1.7	1.0	0.58

Table 1: *Ray casting performance in millions rays per second on a single CPU at a resolution of* 1024×1024 *pixels using a AthlonMP 1800+ (1.5 GHz) processor and various scenes. Given our current ray tracing performance, shading is becoming a bottleneck. Even with simple Phong-like shading, computing rays with SSE can increase performance by roughly a factor of two. We also provide the numbers for pure ray casting without shading, which doubles performance again. The performance numbers for dynamic scenes are somewhat lower than for static scenes.*

Increased performance: We completely reimplemented the core algorithms, which now allows for using SIMD instructions and for exploiting the fast packet-traversal code of the underlying ray tracer. This increases performance by a factor of up to 8 depending on resolution.

Better scalability: We removed the scalability bottlenecks discussed above allowing the new system to scale linearly in number of rendering clients, frame rate, and resolution while maintaining the logarithmic scalability with respect to scene size thus supporting massive scenes with tens of millions of polygons.

Improved image quality: The improved performance and scalability allows for increasing the image quality even during interaction with the scene. In addition we added support for textures, programmable light source, surface shaders, and provide efficient anti-aliasing and tone mapping.

3.1. High Performance using Coherent Rays

Wald et al. [19] showed that the key to efficient ray tracing is exploiting coherence. Instead of tracing rays individually they handled coherent packets of rays in a breadth-first manner and optimized their code using the streaming SIMD extensions (SSE) of modern processors. The latter optimization alone increased performance by more than a factor of two.

Since the previous system was published, additional performance improvements have been developed for realtime ray tracing. This has widened the gap between single ray computations and packet-traversal (see Table 1). While the ray tracing core is now capable of casting close to four million rays per second on a single CPU, this performance drops by roughly a factor of four when tracing single rays, even if only simple shading is used. In fact, shading even starts to become the bottleneck, suggesting that further performance improvements are best achieved by better optimizing shading computations. In our case, "shading" means computing VPL contributions, evaluating BRDFs, sampling textures, and computing procedural material properties.

Modern compilers now offer better support for SIMD code. Instead of having to write the code in pure assembly language the compiler now offer *intrinsics* for SSE instructions. While intrinsics are still essentially assembly instructions hidden in function call form, they can be placed between normal C++ instructions and allow to symbolically reference normal C++ variables directly. SSE-code written in this style is much better maintainable and the compiler can optimize it through better scheduling and register assignment.

3.1.1. Generation of Coherent Ray Packets

As outlined in [19], the most natural way of combining rays to coherent packets is to combine primary rays from neighboring pixels. This maximizes coherence inside a packet, and allows to also generate packets of coherent shadow rays by connecting the (usually coherent) hit points to the point light sources. However, this no longer works for instant global illumination due to interleaved sampling where neighboring pixels no longer share the same set of VPLs.

In order to restore coherence for packets of shadow rays, we could reorder the rays according to the VPL sets. Even though this reordering seems simple and straightforward it requires scatter-gather operations that are very costly due to memory bandwidth and the number of instructions required. This overhead offsets any benefits gained through coherent ray tracing.

An even simpler solution is to already group primary rays according to the interleaving set used (see Figure 2). This reduces the coherence of primary rays by artificially increasing their spacing by a factor of three and results in a higher overhead for primary rays (about 10% to 20%). However, it allows to easily and efficiently generate coherent packets of shadow rays without any overhead for reordering operations. The slightly smaller coherence for primary rays is easily offset by the ability to compute all rays as packets and by using SSE code.

3.1.2. Streaming Computations

SIMD ray casting can only handle packets of four rays due to SSE's limitation to four float values per register or operation. However, additional performance can still be achieved by reformulating the *entire* algorithm in a more streaming-like and breadth-first way. Instead of directly proceeding to shading and shadow ray generation after having cast the first packet of four rays, we cast *all* primary rays for the current VPL set in a tile and store their results. Finally, *all* computed hit points are connected in turn to each VPL in the current set before computing additional rays (e.g. for reflections).

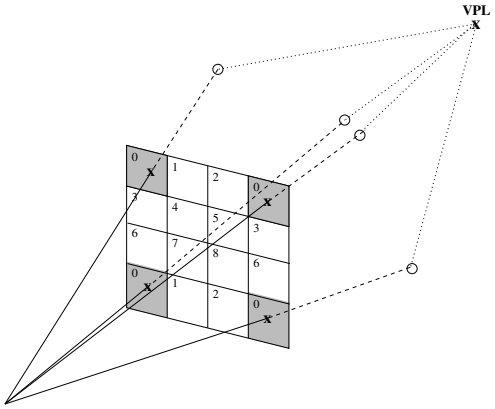

Figure 2: *Generating coherent ray packets with respect to a 3×3 interleaved sampling pattern. Only combining rays with the same interleaving pattern (here: set 0) allows to directly connect the four primary hit points to the same VPL for generating a coherent packet of shadow rays.*

This approach is a more general and cleaner design than the previous one and better matches the trade-off in todays software and hardware. It maximizes coherence by using coherent ray tracing with larger packets. In addition, the efficiency of SSE code increases with larger data blocks, as any setup-overhead can be amortized over more data values [8, 1].

The streaming approach also allows for supporting ray tracing architectures that directly support larger packet sizes. For example, the SaarCOR hardware architecture for ray tracing [16] uses a packet size of 8×8 rays. This approach also fits better to the streaming processing mode of modern programmable GPUs [15], thus the new algorithm could also be directly mapped to such architectures.

3.2. Streaming Shading Computations

The streaming paradigm is not limited to shooting rays. In the past the computation of global illumination has been clearly dominated by the cost of tracing rays. However, with the realtime ray tracing systems of today this is no longer true. Table 1 shows that even with very simple Phong-like shaders the cost of shading already starts to dominate ray tracing.

Thus, it becomes necessary to use the streaming approach also for shading computations. However, this is much more difficult due to the variability of shading code written by users. As a consequence, we chose to change the programming model for shaders by splitting it into two separate entities: a *BRDF shader* and a *surface shader*.

The BRDF shader is responsible for evaluating the lighting contribution and evaluating the BRDF, based on a set of parameters provided by the surface shader. The BRDF shader is also responsible for tracing reflection and refraction rays. Because most traditional surface shaders rely on a small number of standard BRDFs for evaluating the illumination, we expect this separation to impose few limitations in practice. The separation of the shaders, however, allows us to optimize the few BRDF shaders by applying the same streaming approach as described above.

The surface shaders only compute the parameters for the BRDF-shaders. They are fully programmable by the user and allow for all the standard procedural shading techniques such as texture access and noise functions to modify the local surface properties [6].

This new programming model for shaders allows for highly optimized shading while still providing most of the options of fully programmable shaders. As a result we can

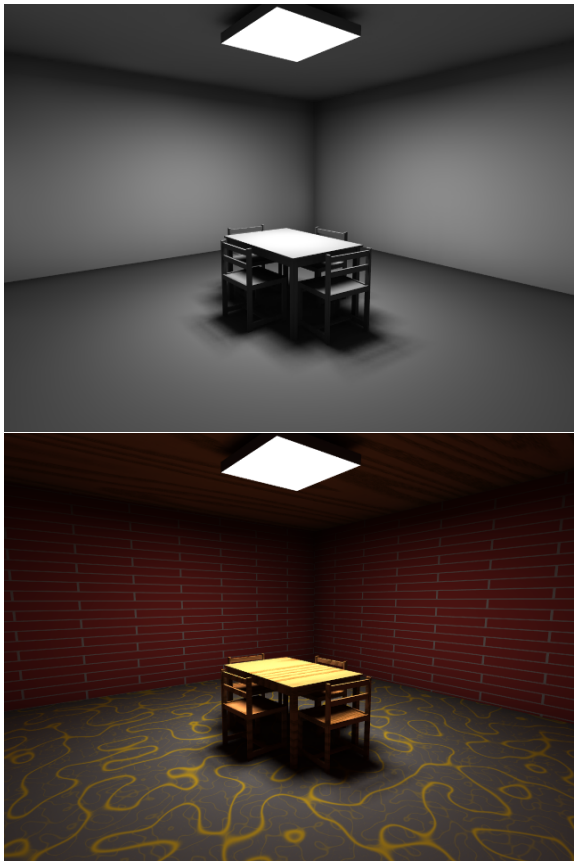

Figure 3: *Freely programmable procedural shading in a globally illuminated scene. The standard "Shirley 6" test scene (left) and after applying several procedural shaders (marble, wood, and brickbump). Even with shaders that make extensive use of noise function the performance only drops to 3.7 fps compared to 4.5 fps with constant diffuse reflection.*

combine global illumination with efficient procedural shading for better image quality as shown for example in Figure 3.

However, the bulk of all shading operations, such as computing the contribution from the VPLs, generating and shooting the shadow rays, and evaluating the BRDF for each non-occluded shadow ray, are now performed by highly optimized SSE code using the streaming approach.

3.3. Efficient Anti-Aliasing by Interleaved Super-Sampling

The original system provided high-quality anti-aliasing using progressive over-sampling in static situations but suffered from artifacts during interaction. This was caused by the low image resolution and the fact that only a single primary ray was used per pixel.

Efficient anti-aliasing is still an unsolved problem in ray tracing as the rendering time increases linearly with the number of rays traced. Anti-aliasing by brute-force super-sampling in each pixel is thus quite costly, in particular in an interactive context. On the other hand, methods like adaptive super-sampling are problematic due to possible artifacts and the increased latency of refinement queries in a distributed setup.

However, in our case anti-aliasing can be implemented with little performance impact using a similar interleaving approach as for sampling VPLs. Instead of connecting each primary ray to all M VPLs in the current set, the VPLs are grouped into N different subsets with roughly M/N VPLs each. We then use N primary rays per pixel for anti-aliasing, each ray computing illumination only with its own subset of VPLs. For the images in this paper we use $N = 4$.

With the faster global illumination computation and the typically large number of VPLs, the overhead of the $N-1$ additional rays is usually in the order of 20% to 30%, which is well justified by the resulting increase in image quality (see Figure 4).

3.4. Improving Scalability

Because of the high preprocessing costs of photon mapping we do not support it in the new version. This allows us to move the discontinuity buffer from the server to the clients resulting in a tremendously reduced network bandwidth as we only need to transfer final quantized pixel colors to the server. The load on the server is also minimized and is limited to managing the load balancing by handling out tiles and copying the finished pixel values to the frame buffer.

A drawback of filtering on the client side is that a 3×3 interleaved sampling pattern requires that each client computes an overhead of one additional pixel on each side of the tile. For a tile size of 40×40 pixels the overhead of 164 pixels is now easily tolerable when using ray packet traversal

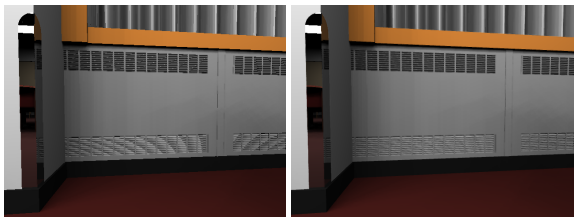

Figure 4: *Efficient anti-aliasing. Left: A single primary ray per pixel, exhibiting strong aliasing artifacts. Right: 4 primary rays per pixel, resulting in reduced aliasing. Using our method, anti-aliasing is very cost efficient: While the left image renders at 4 fps, the anti-aliased image is only slightly slower, running at 3.2 fps. As both images use the same total number of shadow rays per pixel, the quality of the lighting simulation is virtually indistinguishable.*

(roughly 10%). During discontinuity filtering all pixel data can easily be kept in the processor caches. Fortunately, the cache effects of filtering after a tile has finished rendering have little effects on the ray tracing computations. The sharing of cache entries between tiles is rather small anyway. In our implementation the filtering code has also been implemented with SIMD code.

Due to moving the filtering step on the client side each client now has to generate all sets of VPLs per pattern. Without the need to shoot thousands of caustic photons for photon mapping the costs of generating a few hundred VPLs are easily tolerable.

In order to further enhance image quality we added support for client-side tone mapping for handling the high dynamic range of radiance values. The client-side implementation was preferred for the same reasons as for filtering but introduced another complication. Global information about the frame, such as the average luminance, is not available to clients, which processes only a single tile at a time. We therefore chose to compute the required values separately for each tile and combine them on the server side. The server then updates the tone mapping parameters and broadcasts them to all clients to be used for the next frame. This introduces an additional latency of one frame for changes in tone mapping, which is negligible in practice.

4. Results

After discussing the individual improvements of the new system, we next evaluate its overall performance. All of the following experiments have been performed on a cluster of 24 dual AthlonMP 1800+ PCs with 512 MB memory each. All clients are connected to a commodity 100 Mbit switch, while the server machine uses a gigabit-ethernet connection to the gigabit uplink of the switch. The gigabit connection is required for handling the large bandwidth of pixels data at

Figure 5: *Increased image quality. Left: Comparison of old (far left) and new (left center) system both running on 8 clients. We adjusted the parameters such that the new system achieves the same frame rate. The increased performance thus allows for significantly more VPLs yielding an image quality that corresponds to a converged solution of the old system after accumulating results for several seconds. Note that the improved image quality is much more visible during interaction with the scene. Using the same parameters as the old system the new system achieves a frame rate of 12 fps. Two images on the right: Details of the two solutions on the left, respectively. Note, that these image are fully recomputed for every frame and thus the same quality is maintained even during interactions.*

high resolution and/or high frame rates. This setup is identical to the one used by Wald et al. [19, 20].

More information on the example scenes can be found in Figures 7 to 9.

4.1. Direct Performance Comparison

We start with a direct comparison of the new system with the previous implementation by Wald et al. For this test we used no programmable shading and turned off anti-aliasing as these features were not available in the old system. Note, however, that all other computations are included in this comparison, including shooting of rays, computing the contributions of the VPLs, cost for filtering with the discontinuity buffer, and handling of network communication. Also note that the measurements for the new system already includes the 10% overhead due to the loss of coherence as mentioned earlier.

The results of the experiments are shown in Table 2. At the previously published resolution of 640×480 the new system outperforms the old one by a factor of 2.5 to 3.2. This improvement is mainly due to the inefficiencies of the old system with respect to tracing coherent rays and better cache utilization by the new streaming approach. For higher resolutions the advantages of the new system increase even more, up to a factor of 8. At higher resolution we even see a *super-linear* behavior due to the increased coherence between rays. Table 3 illustrates the sub-linear costs of ray packet traversal as image resolution is increased.

	Office	Conference	Power plant
640×480	2.7	3.2	2.5
800×600	2.9	3.41	2.7
1000×1000	4.2	7.2	4.0
1600×1200	5.7	8.0	5.1

Table 2: *Ratio of performance comparing the new versus the previous system using 64 VPLs/pixel with full shading and filtering at various resolutions. Even with the 10% overhead due to overlap at tile boundaries the new system outperforms the original by up to a factor of 8 at higher resolutions.*

	Office	Conference	Power Plant
640x480	1.72 (1.41)	1.12 (0.85)	0.33 (0.28)
800x600	1.77 (1.45)	1.22 (0.94)	0.42 (0.29)
1000x1000	1.84 (1.49)	1.33 (1.03)	0.44 (0.31)
1600x1200	2.00 (1.63)	1.46 (1.09)	0.48 (0.34)

Table 3: *Million rays per second on AthlonMP 1800+ CPUs at different resolutions with 16 VPLs, full shading and filtering. Using ray packet traversal the new system offers sub-linear costs in the number of pixels. The number in parenthesis are for dynamic environments, which impose a ray tracing overhead due to additional processing of scene changes.*

After removing the server bottleneck and the network bandwidth issues of the previous system (see Section 3.4) we are no longer limited to a maximum of 4 to 5 frames per second at video resolution of 640×480. With the new system we achieve up to 20 fps at video resolution, and still 8 fps at full screen resolution (1024×1024) with 12 VPLs per pixel. This is another step towards practical use of the system, where a minimum frame rate of 10 fps is typically needed.

Rendering high-quality images at high frame rates still requires a substantial number of PCs. As all scalability bottlenecks have been removed, we can efficiently use all these rendering clients, and achieve almost-linear scalability in the number of clients, as can be seen in Figure 6: For all tested scenes, scalability for up to 24 client PCs (48 CPUs) is almost perfectly linear. Note that this is not only true for trivial scenes but also for highly complex scenes such as the maple

trees (see Figure 1, center image) or even when computing global illumination in the 50-million triangle power plant scene (see Figure 1, right image). This power plant scene consists of four instances of a 12,5 million triangle plant. Though we did not have access to more machines for testing, it seems that performance should scale well beyond our 48 CPUs. A strict upper limit is still given by the available network bandwidth to the server machine.

5. Conclusions and Future Work

In this paper we presented a scalable approach to interactive global illumination based on the previous work by Wald et al. [20]. We improved both the raw performance of the basic algorithm by a factor of 2.5 to 8 using stream computing with larger packets of rays and by optimizing the implementation with SIMD code. Using the same number of client PCs, the increased performance translates to many more VPLs and consequently better image quality at the same frame rate (see Figure 5).

In addition we removed the previous scalability bottlenecks, which allows us to scale almost perfectly with available computing resources, image resolution, and frame rate.

These improvements allow us to provide significantly better image quality in particular during interactions. The system provides immediate feedback to a user interacting with a scene by fully recomputing the global illumination solution for every frame. Detailed global illumination results provide an unprecedented realism even at interactive rates.

In addition we provide programmable shading, tone-mapping, and an efficient anti-aliasing technique that performs super-sampling at a fairly moderate cost. These techniques again improve image-quality with only a slight impact in performance.

Due to the logarithmic scaling of the underlying ray tracing engine we are able to efficiently compute global illumination even in scenes with up to 50 million triangles. However, this scene is at the edge of what is currently possible.

Even with the results show in this paper a number of significant problems remain. The algorithm still does not scale well in large scenes with significant occlusion and many light sources where the stochastic distribution of VPLs introduces lighting artifacts that are difficult to eliminate. In both cases the algorithms are unaware of the relative view importance of specific light sources.

It is also important to stress that the presented approach is biased because of the way we deal with the singularities at VPL locations. While we already consider glossy effects during the generation of VPLs the reflection characteristics of the surface at the VPL is not yet taken into account while illuminating the environment.

The current implementation is missing support for photon mapping. Photon mapping could be easily integrated into the current approach by computing all sets of caustic photons on each client. However, this preprocessing costs would pose a significant limit on performance and scalability. Therefore, we will concentrate on optimizing the photon mapping step itself.

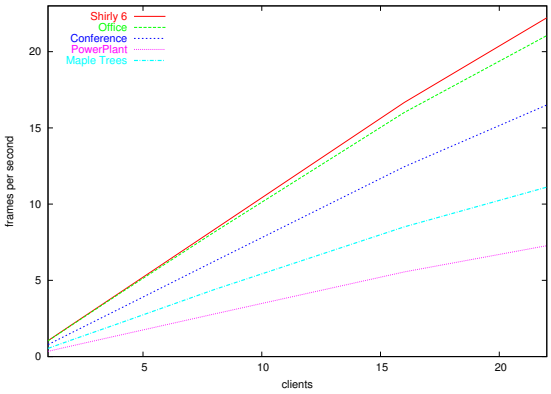

Figure 6: *Scalability with client PCs: Performance is essentially linear up to 24 PCs/48 CPUs. This applies to scenes ranging from several hundred triangles (Shirley-6) up to the the power plant with 50 million triangle (four instances). Also note how the new system scales in frame rate well beyond the original system, which was limited to at most 5 frames per second.*

Finally, it will be interesting to evaluate how our algorithm can be mapped to other hardware architectures, e.g. to the SaarCOR architecture [16] or to programmable graphics hardware [15]. The scalability of the new approach and its support for stream computing should help supporting these hardware architectures.

The significant improvement in realism in particular during interactive sessions with the system strongly indicates that interactive global illumination will play a major role in future 3D graphics systems. Similar to texturing a few years ago it will become a mandatory feature as soon as the required computing resources become readily available on the desktop. Interactive global illumination might be the next “killer feature” for 3D graphics applications, including computer games.

Acknowledgements

We would like to thank all the people that have contributed to this paper, in particular Andreas Dietrich, Thomas Kollig, and Alexander Keller for many helpful discussions and Oliver Deussen for providing the maple tree model. This project has been financially and practically supported by Intel Corporation.

References

1. AMD Corporation. *AMD Technical Resources*. http://www.amd.com/us-en/Processors/TechnicalResources/.
2. Kavita Bala, Julie Dorsey, and Seth Teller. Radiance Interpolants for Accelerated Bounded-Error Ray Tracing. *ACM Transactions on Graphics*, 18(3):213–256, August 1999.
3. Alan Chalmers, Tim Davis, Toshi Kato, and Erik Reinhard. Practical Parallel Processing for Today's Rendering Challenges. In *SIGGRAPH 2001 Course Notes, Course 40*, July 2001.
4. Kirill Dmitriev, Stefan Brabec, Karol Myszkowski, and Hans-Peter Seidel. Interactive Global Illumination Using Selective Photon Tracing. In *Proceedings of the 13th Eurographics Workshop on Rendering*, pages 21–34, 2002.
5. George Drettakis and François Sillion. Interactive Update of Global Illumination Using A Line-Space Hierarchy. *Computer Graphics*, pages 57–64, August 1997. (Proceedings of ACM SIGGRAPH 1997).
6. David Ebert, Kenton Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing and Modeling – A Procedural Approach*. Academic Press Professional, 2nd edition edition, 1998.
7. Jörg Haber, Karol Myszkowsky, Hitoshi Yamauchi, and Hans-Peter Seidel. Perceptually Guided Corrective Splatting. *Computer Graphics Forum*, 20(3):142–152, 2001. (Proceedings of Eurographics 2001).
8. Intel Corporation. *Intel Pentium 4 Processors Manuals*. http://developer.intel.com/design/pentium4/manuals/.
9. Henrik Wann Jensen. Global Illumination using Photon Maps. *Rendering Techniques 1996*, pages 21–30, 1996. (Proceedings of the 7th Eurographics Workshop on Rendering).
10. Alexander Keller. Instant Radiosity. *Computer Graphics*, pages 49–56, 1997. (Proceedings of ACM SIGGRAPH 1997).
11. Alexander Keller and Wolfgang Heidrich. Interleaved Sampling. *Rendering Techniques 2001*, pages 269–276, 2001. (Proceedings of the 12th Eurographics Workshop on Rendering).
12. Thomas Kollig and Alexander Keller. Efficient Multidimensional Sampling. *Computer Graphics Forum*, 21(3):557–563, 2002. (Proceedings of Eurographics 2002).
13. Michael J. Muuss. Towards Real-Time Ray-Tracing of Combinatorial Solid Geometric Models. In *Proceedings of BRL-CAD Symposium '95*, June 1995.
14. Steven Parker, Peter Shirley, Yarden Livnat, Charles Hansen, and Peter Pike Sloan. Interactive Ray Tracing. In *Proceedings of Interactive 3D Graphics (I3D)*, pages 119–126, April 1999.
15. Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray Tracing on Programmable Graphics Hardware. *ACM Transactions on Graphics*, 21(3):703–712, 2002. (Proceedings of ACM SIGGRAPH 2002).
16. Jörg Schmittler, Ingo Wald, and Philipp Slusallek. SaarCOR – A Hardware Architecture for Ray Tracing. In *Proceedings of Eurographics Workshop on Graphics Hardware*, pages 27–36, 2002.
17. Parag Tole, Fabio Pellacini, Bruce Walter, and Donald P. Greenberg. Interactive Global Illumination in Dynamic Scenes. *ACM Transactions on Graphics*, 21(3):537–546, 2002. (Proceedings of ACM SIGGRAPH 2002).
18. Ingo Wald, Carsten Benthin, and Philipp Slusallek. OpenRT - A Flexible and Scalable Rendering Engine for Interactive 3D Graphics. Technical report, Saarland University, 2002. Available at http://graphics.cs.uni-sb.de/Publications.
19. Ingo Wald, Carsten Benthin, Markus Wagner, and Philipp Slusallek. Interactive Rendering with Coherent Ray Tracing. *Computer Graphics Forum*, 20(3):153–164, 2001. (Proceedings of Eurographics 2001).
20. Ingo Wald, Thomas Kollig, Carsten Benthin, Alexander Keller, and Philipp Slusallek. Interactive Global Illumination using Fast Ray Tracing. *Rendering Techniques 2002*, pages 15–24, 2002. (Proceedings of the 13th Eurographics Workshop on Rendering).
21. Ingo Wald, Philipp Slusallek, and Carsten Benthin. Interactive Distributed Ray Tracing of Highly Complex Models. *Rendering Techniques 2001*, pages 274–285, 2001. (Proceedings of the 12th Eurographics Workshop on Rendering).
22. Bruce Walter, George Drettakis, and Steven Parker. Interactive Rendering Using the Render Cache. *Rendering Techniques 1999*, pages 19–30, 1999. (Proceedings of the 10th Eurographics Workshop on Rendering).

Figure 7: *Left: Another view of the room with globe containing roughly 20,000 triangles. This time the animated lamp illuminates North America which reflects a yellowish color into the room (compare to Figure 1a). Right: The conference room with 280,000 triangles rendering at about 20 fps on 22 dual PCs with 12 VPLs.*

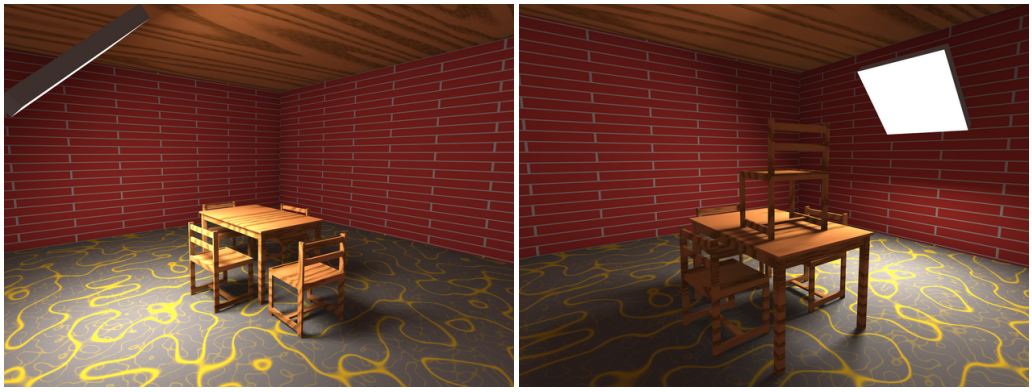

Figure 8: *The well-known Shirley-6 test scene consisting of 600 triangles augmented with procedural shading. These frames render at roughly 22 fps with 12 VPLs allowing arbitrary changes in the scene with immediate global illumination feedback that includes even subtle lighting and shadow details.*

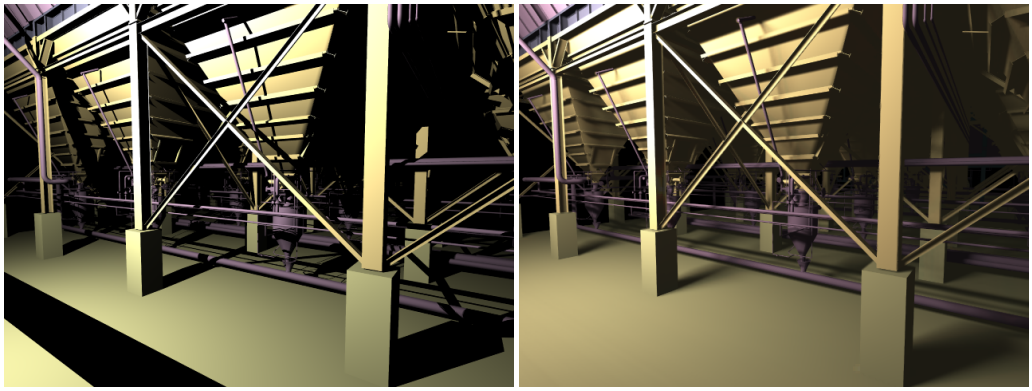

Figure 9: *The same view of the power plant scene containing four copies of the model with a total of 50 million triangles rendered at about 2 fps with global illumination. These two views clearly show the difference between direct illumination with hard ray traced shadows and the smooth lighting due to the global illumination simulation. Note the detailed smooth shadow and the indirect illumination.*