Eurographics Conference on Visualization (EuroVis) 2014
H. Carr, P. Rheingans, and H. Schumann
(Guest Editors)

Volume 33 (2014), Number 3

Parallel Irradiance Caching for Interactive Monte-Carlo Direct Volume Rendering

R. Khlebnikov[1], P. Voglreiter[1], M. Steinberger[1], B. Kainz[2] and D. Schmalstieg[1]

[1]Institute for Computer Graphics and Vision, Graz University of Technology, Austria
[2]Department of Computing, Imperial College London, UK

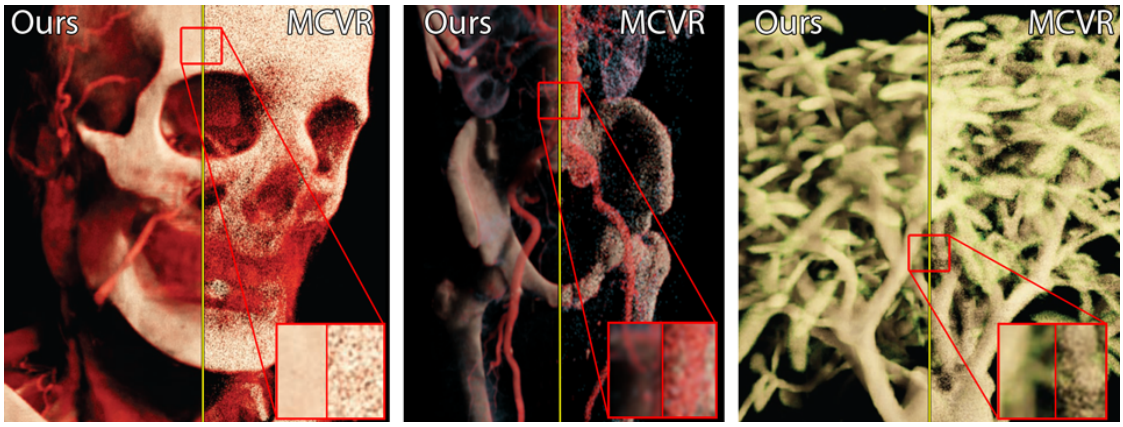

Figure 1: *Three volumes with enlarged areas rendered with Irradiance Caching (left half) and normal Monte Carlo Volume Rendering (right half) after the same time. The extracted areas show a significant reduction in noise with our approach.*

Abstract

We use irradiance caching for Monte Carlo direct volume rendering to achieve fast convergence for complex scenes with arbitrary sources of light, and show that our approach achieves up to four times increased convergence rate. We propose a technique to save and reuse the illumination within the volume without preprocessing by utilizing three procedures that run concurrently on a single GPU. The first procedure is the main rendering procedure. The second procedure computes new cache entries, and the third procedure corrects the errors that may arise after creation of new cache entries. In this paper, two distinct approaches allow massive parallelism of cache entry creation, while keeping the memory footprint low. First, a novel extrapolation approach outputs high quality irradiance approximations. Second, we derive a suitable priorization scheme to increase the convergence rate by dedicating more computational power to those parts of the screen showing complex scene features.

1. Introduction

Perception of volumetric data displayed with Direct Volume Rendering (DVR) approaches can be greatly improved if global illumination effects are taken into the account during rendering. This includes both strong effects, such as global shadows, and more subtle ones, such as scattering. One of the approaches to compute such costly global illumination effects is Monte Carlo Volume Rendering (MCVR).

MCVR uses the stochastic integration techniques to solve the Radiative Transfer Equation (RTE) [Cha60], which describes the complex interaction of light with the participating media. One of the advantages of MCVR is that the rendering can be done in a progressive manner. This allows show-

ing intermediate results, thus enabling interactive data exploration. However, the first few iterations may be very noisy, which may force the user to suspend interaction until higher levels of convergence are achieved.

Irradiance [WRC88] and radiance [KGPB05] caching techniques significantly improve the convergence rate by storing and reusing the illumination computation results for nearby pixels. This does not reduce the quality, because the illumination field varies smoothly in large parts of natural scenes. An additional advantage of such caching techniques is that the cache can be built during rendering – once we encounter a position within the scene not covered by the cache, we compute a new cache entry and then proceed with rendering. However, for interactive DVR, multiple problems prevent using traditional techniques.

First, the majority of modern DVR systems employ the GPU to achieve interactive frame rates. The massive parallelism required to effectively utilize the power of the GPU is achieved by dedicating one thread per pixel and executing the threads in SIMD manner in thread blocks. Thus, if all the threads within a thread block compute a similar cache entry at the same time, a lot of computational effort is wasted since many redundant entries are created. Second, even if the cache density is controlled, displaying the results for the thread block will stall until all the necessary cache entries are created, which affects interactivity significantly.

In this paper, we propose an integrated approach that allows to build the irradiance cache in parallel to rendering on a single GPU, without leading to excessively dense caches or rendering stalls. To achieve compatibility with most DVR applications, we formulated the following goals:

- Irradiance caching should not interfere with the interaction and the visualization, allowing for efficient data exploration. This includes both virtual camera motion as well as interactive transfer function design.
- Caching should improve the convergence rate of the MCVR without sacrificing its quality.
- Memory footprint of the cache should be minimal.

These goals are contradictory to some extent. For instance, a very low memory footprint may be achieved at the expense of quality or interactivity. Achieving a good balance in a GPU-friendly way is not trivial, as multiple interlocking tasks compete for the GPU time and must be scheduled. In this regard, we present the following contributions:

- We demonstrate object- and screen-space approaches which allow the new cache requests to be handled in a massively parallel MCVR system without rendering stalls or creating excessively dense cache on a single GPU.
- We show a prioritization scheme that allows to distribute the available computational power between building the irradiance cache and the actual rendering.
- We propose a modification to the exponential irradiance extrapolation approach, which is more accurate for scenes with high irradiance field gradients.

Overall, we show that our method improves the convergence rate and, consequently, the visualization quality of MCVR without reducing its interactivity.

2. Background and related work

In this section, we give a brief description of the core technologies building the foundation of our approach. In particular, we discuss MCVR and irradiance caching.

2.1. Global illumination in volume rendering

There are numerous approaches to computing global illumination for DVR. They differ concerning the range of effects they can achieve as well as concerning performance and memory requirements. An excellent overview of the existing approaches towards computing advanced volumetric illumination in interactive volume rendering is given by Jönsson *et al.* [JSYR13] in their state of the art report.

The radiative transfer equation [Cha60] serves as a base for most methods which compute global illumination in participating media. In their early work, Kajiya and Von Herzen [KVH84] propose to use a two-pass approach to include global illumination effects for rendering of volumetric datasets. During the first pass, the radiance is estimated for each voxel, which is consecutively integrated along view rays during the second pass. However, the first pass is very time consuming and thus not applicable to interactive visualization of dynamic scenes. Many approximations for improving performance exist. Ambient occlusion is used for efficient shadow computation in a local neighborhood [SA07, RMSD*08, DVND10] and summed area tables provide an approximation including global shadowing [SMP11].

Rezk-Salama [Sal07] proposed to use a Monte-Carlo approach for physically-based volume rendering. However, displaying only a set of isosurfaces limits the quality of this method. More recently, Kroes *et al.* [KPB12] demonstrated that progressive Monte-Carlo rendering is feasible for interactive DVR. We use their method as basis for our method (Section 3).

Photon mapping approaches [JC98] have also been applied to DVR. Jönsson *et al.* [JKRY12] propose Historygrams to increase the speed of photon map re-computation for transfer function changes. However, the photon gathering step is still quite time consuming, leading to low frame rates when the camera moves. Precomputed radiance transfer approaches [SKS02] allow to overcome the high runtime cost. Zhang *et al.* [ZD13] propose to use precomputed photon maps for high quality interactive volume data visualization. While the rendering performance is very high, full recomputation of the photon map is still required upon a transfer function change.

Global illumination can also be approximated by a diffusion process [Sta95]. Zhang *et al.* [ZM13] propose to use a convection-diffusion partial differential equation. While

the performance and range of effects achieved by this approach is high, it can only handle certain light types. Weber *et al.* [WKSD13] apply an approach using virtual point lights (VPL) in interactive volume rendering. Even though the resulting image is visually appealing, interactive visualization severely restricts the VPL number.

2.2. Irradiance caching

The irradiance caching technique [WRC88] takes advantage of the fact that the indirect irradiance field is mostly smooth. The irradiance and some additional quantities, such as the irradiance gradient [WH92], are computed for a sparse set of cache points, which are then used during rendering to interpolate the irradiance, avoiding costly integration over all directions. Křivánek *et al.* [KGPB05] proposed radiance caching which stores and interpolates direction-dependent radiance using spherical harmonics. Later, Jarosz *et al.* [JDZJ08] extended the radiance caching approach for use with participating media.

The shape of the influence zone of cache entries affects the memory footprint necessary to achieve high quality results. We need to consider the total number of cache entries and the amount of data stored for each of them. The most widespread shape is spherical [JDZJ08]. The goal of storing very little information for each cache entry is affected, if the cache density increases significantly in the areas of high frequency illumination changes (Fig. 3). Furthermore, spherical influence zones cause excessive cache density in all directions, even if irradiance changes rapidly in only one direction. Therefore, an adaptive shape of the influence zone is more suitable for highly inhomogeneous media, which, for instance, can be observed in direct volume rendering of scientific data. Ribardière *et al.* proposed adaptive records for irradiance caching methods, applied to surface data [RCB11a] and volume data [RCB11b] rendering, where the influence zones are adapted according to geometrical features and irradiance changes.

There have been attempts at bringing irradiance caching to parallel systems, mostly for multiple nodes using MPI. Robertson *et al.* [RCLL99] propose distributing cache generation over several nodes and use frame to frame coherence for geometric scenes to reduce the computational demand. However, single nodes still compute entries for screen regions serially which limits the overall parallelism of the approach. On top, cache synchronization among nodes is problematic. More recently, Debattista *et al.* [DSC06] propose separating rendering from irradiance calculations. They share parts of the locally computed cache of single nodes at particular time intervals, which leads to issues with latency and cache misses for entries computed on a different node. This has also been observed by Kohalka *et al.* [KMG99]. Gautron *et al.* [GKBP05] proposed the radiance cache splatting approach to enable efficient use of the GPU for rendering with radiance cache. Debattista *et al.* [DDPdSC11] propose a wait-free data structure for populating irradiance cache on parallel systems with shared memory. However, both methods create the cache entries sequentially for the whole screen or a large tile and do not handle participating media.

3. Background: Exposure Render

We use the *Exposure Render* method presented by Kroes *et al.* [KPB12]. The Exposure Render system applies the Monte Carlo approach to solve the radiance transfer equation for the purpose of interactive progressive volume rendering. In this section, we describe the relevant techniques applied in the Exposure Render system that we further use to build our caching approach.

In its essence, the goal of volume rendering is to solve a radiative transfer equation for a heterogeneous participating medium. Using the notation employed by Jarosz *et al.* [JDZJ08], the radiative transfer equation for non-emissive participating media can be written as:

$$L(\mathbf{x},\vec{\omega}) = \int_0^s T_r(\mathbf{x}\leftrightarrow\mathbf{x}_t)\sigma_s(\mathbf{x}_t)L_i(\mathbf{x}_t,\vec{\omega})dt + T_r(\mathbf{x}\leftrightarrow\mathbf{x}_s)L(\mathbf{x}_s,\vec{\omega}), \tag{1}$$

where $L(\mathbf{x},\vec{\omega})$ is the radiance reaching $\mathbf{x}$ along a ray with direction $\vec{\omega}$, which is parametrized as $\mathbf{x}_t = \mathbf{x} - t\vec{\omega}, t \in (0,s)$, and $\mathbf{x}_s$ is the exit point from the volume in direction $-\vec{\omega}$. $L_i(\mathbf{x},\vec{\omega})$ is the in-scattered radiance at position $\mathbf{x}$ in direction $\vec{\omega}$. Finally, the transmittance, T_r, is computed as

$$T_r(\mathbf{x}'\leftrightarrow\mathbf{x}) = e^{-\tau(\mathbf{x}'\leftrightarrow\mathbf{x})}, \tag{2}$$

where τ is the optical thickness:

$$\tau(\mathbf{x}'\leftrightarrow\mathbf{x}) = \int_{\mathbf{x}'}^{\mathbf{x}} \sigma_t(\mathbf{x})dx. \tag{3}$$

Here, σ_t is the extinction coefficient, which is the sum of the scattering coefficient σ_s and the absorption coefficient σ_a. The solution of the radiative transfer equation may be obtained using a Monte-Carlo approach. In the Exposure Render framework, it is implemented as follows:

- For each pixel on the screen, a single ray is cast into the scene through a random position within this pixel.
- Each ray propagates through the volume to find a single tentative collision point (scatter event).
- For this event, the illumination is estimated by casting a ray towards a randomly selected sample located at one of the lights within the scene. Similarly to the previous step, if a collision point with the medium is found, the point is considered unlit. Otherwise, this light sample in conjunction with sampling a bidirectional reflectance distribution function (BRDF) and/or a phase function is used to compute the illumination and considered in the current estimate for this pixel.
- The current estimate is incorporated into the final pixel color using a running average approach.

In this paper, we accelerate the illumination computation

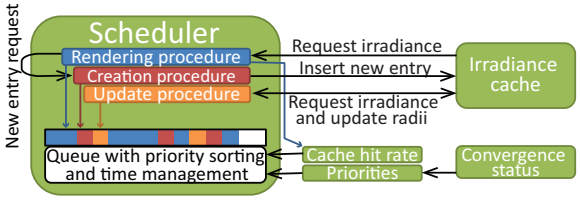

Figure 2: *The overview over our system. Three procedures run in parallel on a single GPU. The scheduler distributes time slots for each of them based on the cache hit rate. The rendering procedure is also prioritized based on screen-space convergence information.*

step for isotropic phase functions using the irradiance cache. In this case the irradiance can be computed as the integral of the incident radiance over the sphere of directions:

$$E(\mathbf{x}) = \int_{\Omega} L(\mathbf{x}, \vec{\omega}) d\vec{\omega} \tag{4}$$

4. Cache management

The main objective of our approach is to generate an irradiance cache in a massively parallel way while concurrently performing parallel DVR. To achieve this goal, we run three procedures simultaneously on a single GPU (Fig. 2):

4.1. GPU Scheduling

We use an open source implementation of *Softshell* [SKK*12] as a basis for scheduling our tasks on the GPU. In a so-called persistent megakernel approach, Softshell provides an implementation of dynamic GPU queuing. Input data of an algorithm, called procedure in Softshell, is represented either as 'Work Item', a single date for one thread, or 'Work Package', a pre-configured collection of data for multiple threads. Queues residing on the GPU collect the data. The scheduler draws items from the queues and assigns worker threads from a persistent pool. Softshell also features dynamic generation of work on the GPU: During execution of a work package or item, additional data can be added to the queues and again drawn out of them by the scheduler. It is also possible to execute several procedures concurrently by using multiple queues. Additionally, Softshell provides prioritization of certain parts of the work to be done. Firstly, if several procedures are present, Softshell can distribute the total available computing power proportionally among these. Secondly, work packages within a queue can be drawn according to their priority. For example, some image regions which are still rather noisy might need more attention than quickly converging background.

To implement our approach, we use the following three largely independent procedures:

The rendering procedure is responsible for generating the output image.

The cache entry creation procedure processes cache entry requests and computes the necessary quantities.

The cache update procedure improves the quality of cache entries by eliminating discontinuities in the estimated irradiance field.

The *rendering procedure* implements the basic Exposure Render approach with the modified illumination computation step. The work packages express a block of pixels of the output image and accordingly, the input data are screen coordinates per thread. For each scattering event, we check whether cache information is available. If so, we extrapolate the irradiance and use it for shading. If not, we add a new work package for creating a cache entry and perform MCVR shading. Upon finishing, the work package adds itself back to the queue for the upcoming iterations.

The *cache entry creation procedure* draws work packages from the queues for processing the requests and inserts new entries into the cache (Section 4.2). In this case, a whole work package corresponds to one single cache entry with input data encompassing the spatial position at which we want to create a new entry as well as the primitive we have locked according to the chosen method (Section 5) for later unlocking. This is a single-shot event, hence no additional work is generated.

However, because the cache entry creation procedure accounts only for irradiance changes in a local neighbourhood, some discontinuities in the cached irradiance arise in the areas where global illumination changes with high frequency. To address this, we initiate several work packages for the *cache update procedure* to detect discontinuities and adjust the cache accordingly (Section 4.3). One work package again resembles a region of the screen and one item a single pixel. We pick a random cache entry along each ray protruding towards the scene and update the picked entries. We create a pool of work packages which execute this procedure with an unlimited number of iterations. Each work package picks a cache entry, updates it and re-emits itself for iterative execution.

The persistent thread base and built-in means for interrupting GPU execution allow us to perform frameless rendering. We aim at a frame rate of 20 Hz to balance computation time versus overhead caused by the interruption. Additionally, we dynamically adjust the per-task time frame of the procedures. higher the cache hit rate, the less time we spend on creating new cache entries and instead focus on using the already present information. Alternatively, this could also be achieved by fixing the maximum number of iterations per procedure in a frame-based approach, or even with traditional kernels. However, using Softshell provides us more flexibility.

4.2. Cache entry creation

We take the following steps to create a single cache entry:

- Compute the local coordinate frame.
- Estimate the incoming irradiance at the centre and at six points offset along the axes of the local coordinate frame.
- Compute validity radii for each of the six axis directions.

Below, we first show how we estimate the irradiance at the necessary positions. Next, we show how to extrapolate the irradiance to arbitrary positions. Third, we explain how we choose validity radii based on our extrapolation approach. Finally, we justify the selection of the local coordinate frame.

Estimation of irradiance values. The incoming irradiance is estimated with Monte Carlo integration using multiple importance sampling with power heuristics [Vea98]. We control the number of samples N for the Monte Carlo integration using the standard deviation based error estimate with 95% confidence interval [HGI08]:

$$1.96\sqrt{\frac{S^2(N)}{N}} < \frac{\gamma}{\gamma+1}E(N), \tag{5}$$

where $E(N)$ and $S^2(N)$ are the average and the variance of irradiance estimates. Note that we use the relative error estimate due to the fact that the human eye is most sensitive to relative and not absolute values of illumination change according to Weber's law of just noticeable differences [TFCRS11, p. 37]. Therefore, larger absolute errors are admissible for brighter regions. The parameter γ controls the acceptable relative error for the irradiance estimation.

Irradiance extrapolation and validity radius. Consider a cache entry located at position x_0 in a homogeneous medium. In our algorithm, the irradiance is estimated at position x_0 (E_0) and a position with an offset Δ (E_Δ). Similar to Jarosz *et al.* [JDZJ08], we assume that the irradiance is an exponential function of position. Therefore, it is necessary to compute the extinction coefficient σ_v to perform the extrapolation to arbitrary locations. As in case of inhomogeneous media, this value does not correspond to the actual extinction coefficient. We call σ_v the *virtual* extinction coefficient. Given E_0, Δ and E_Δ, it is computed as:

$$\sigma_v = -\ln\left(\frac{E_\Delta}{E_0}\right)\cdot\frac{1}{\Delta} \tag{6}$$

Consequently, the extrapolated irradiance at an arbitrary position x can be computed as:

$$E(x) = E_0\cdot\exp(-\sigma_v x) \tag{7}$$

To use the Eq. (7) in three dimensions, we compute the virtual extinction coefficient in a particular direction using barycentric coordinates:

$$\sigma_v(x,y,z) = \frac{\sigma_v^{(x)}\cdot x+\sigma_v^{(y)}\cdot y+\sigma_v^{(z)}\cdot z}{x+y+z}, \tag{8}$$

where (x,y,z) are the coordinates of the point in the local coordinate frame of the cache entry, and $\sigma_v^{(*)}$ are the virtual extinction coefficients along the corresponding axes within the point's octant.

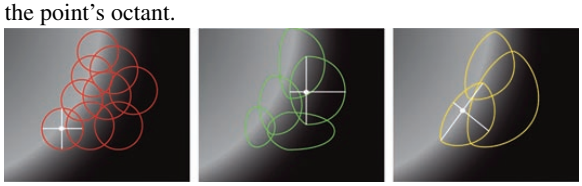

Figure 3: *Influence of the entry shape on the cache density. The white lines denote the radii corresponding to the local coordinate frame of an entry. If the entry is represented as a union of elliptical sections, (middle) less cache entries are required compared to the spherical ones (left). Orienting these shapes along the opacity gradient (right) reduces the cache density even more because in many cases, the largest irradiance gradient coincides with the opacity gradient. Orienting the entries allows for larger validity zones, since the radius has to be small only along one axis.*

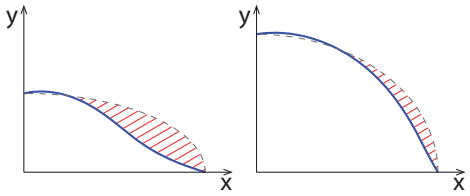

Figure 4: *(left) When interpolating virtual extinction coefficients, the ellipsoidal shape may become inconsistent with the actual validity radius if the ratio of irradiances at different axes is very high . Therefore, to avoid visual artefacts, we discard the influence of the cache entry in areas beyond the interpolated validity radius (red hatching). (right) However, for lower ratios, the ellipsoidal shape is a good estimate for the actual validity radius.*

Given this extrapolation approach, we compute the validity radius for each axis such that the extrapolated irradiance at the edge of the influence zone differs by not more than one ε relative to the estimated irradiance at position Δ:

$$R = \Delta\cdot\frac{\ln(E_R/E_0)}{\ln(E_\Delta/E_0)}, \tag{9}$$

where

$$E_R = \begin{cases} (1+\varepsilon)E_\Delta & \text{if } E_0 \le E_\Delta \\ (1-\varepsilon)E_\Delta & \text{if } E_0 > E_\Delta. \end{cases} \tag{10}$$

We define the shape of the cache entries as a union of elliptical sections with the corresponding half-axes in each octant of the local coordinate frame (Fig. 3). While an ellipsoid is a good estimate, the actual validity radius for the interpolated virtual extinction coefficient may be smaller than the corresponding ellipsoid radius. With our exponential extrapolation approach, this may lead to significant errors in case the ratio of the estimated irradiance at the ends of half-axes is large (Fig. 4). Therefore, we discard the influence of a cache

entry if the point lies outside the validity radius computed with the interpolated virtual extinction coefficient.

If a point lies within the validity zone of multiple cache entries, we compute the log-space weighted average of the per-entry extrapolation results [JDZJ08]:

$$E(\mathbf{x}) \approx \exp\left(\frac{\sum_{k\in C} \ln(E_k) w(d_k)}{\sum_{k\in C} w(d_k)}\right) \tag{11}$$

where the weight is computed as $w(d) = 3d^2 - 2d^3$ with $d_k = 1 - \|\mathbf{x}_k\|/r_k(\mathbf{x}_k)$, $\mathbf{x}_k$ are the coordinates of the point in relation to the local coordinate frame of the cache entry k, and E_k is the irradiance estimated using the cache entry k. Since we use anisotropic ellipsoids as cache entries, each of the radii is computed as: $r(\mathbf{x}) = \|\mathbf{x}\| \cdot \left(\frac{\mathbf{x}_x^2}{r_x^2} + \frac{\mathbf{x}_y^2}{r_y^2} + \frac{\mathbf{x}_z^2}{r_z^2}\right)^{-0.5}$

Local coordinate frame. To minimize the effect of errors introduced by the 3D interpolation of the virtual extinction coefficient, it would be preferable to orient one of the axes of a cache entry's local coordinate frame along the irradiance gradient. The exact orientation would require knowing the gradient in advance. This is a very expensive computation, requiring to store the local coordinate frame with each cache entry. However, we have observed that in many cases the largest change in irradiance is related to the variation of the extinction coefficient. Therefore, orienting the local coordinate frame along the gradient of the extinction coefficient is a good approximation of the optimal orientation. Furthermore, as the computation of extinction coefficients is deterministic and can be done using a simple central differences approach, we can easily recompute it on demand rather than expend additional memory for its storage.

We have compared the cache size for spherical, as well as non-oriented and oriented pseudo-ellipsoidal influence zones. The results have shown that spherical zones indeed lead to very high cache densities (more than four times larger). Oriented and non-oriented zones in many cases produce very similar cache sizes. However, we observed that the cache size for oriented zones was 5-10% smaller than that of the non-oriented ones for optically dense materials. Because using oriented zones does not require additional storage and little computational effort, we use oriented cache entries throughout the remainder of this paper.

4.3. Cache update

While computing the validity radius gives a good approximation for the cases where the irradiance varies smoothly, it can still be erroneous to a certain degree (Fig. 5, left). Deviations result from considering global illumination information only partially during the estimation of incoming irradiance in the local neighbourhood of the cache entry centre. To avoid the resulting artefacts, we incorporate a cache update procedure into our system.

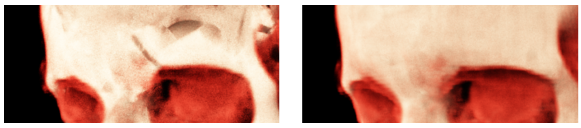

Figure 5: *(left) Visible artefacts can be produced if extrapolated irradiance does not capture the actual irradiance field. (right) The update procedure removes such artefacts.*

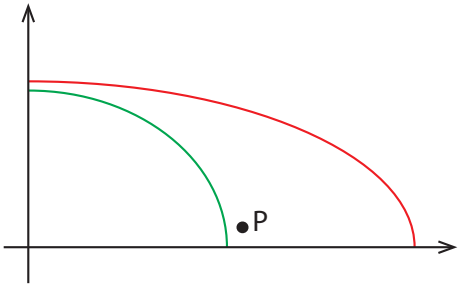

Figure 6: *Reduction of radii in the local coordinate frame of the cache entry to exclude point P from the validity area in the corresponding octant of the cache entry. The reduction is done proportionally to the coordinates of point P in the local coordinate frame.*

The cache update is done in a similar way as proposed by Křivánek *et al.* [KBPv08]. The update procedure searches for scatter events in the same way as the main rendering procedure. Whenever the predictions of overlapping cache entries conflict at the scatter event position, the contribution of the cache entry with the lowest weight is removed from the computation by reducing its radii in the corresponding octant. The radii are reduced proportionally to the event's coordinates in the local coordinate frame (see Fig. 6). We repeat this process, until all conflicts are resolved, with the special case of only one remaining influence.

4.4. Cache entry storage

We store the cache entries in a multi-reference octree, similarly to Křivánek and Gautron [KG09]. Even though the multi-reference octree requires additional storage space, its

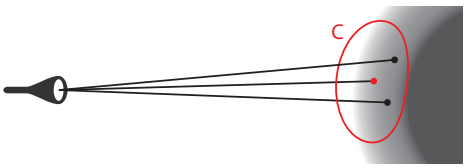

Figure 7: *Neighboring rays may issue cache entry creation requests at very similar positions. We avoid redundant cache entries by allowing only one cache entry to be created per octree node. Once a cache entry C is created, creation requests within the influence of C will be ignored.*

performance is significantly higher than that of its single-reference counterpart [Kar11]. Since we only need to store additional references rather than full entries, the impact on the memory footprint is small.

We use oriented bounding boxes enclosing the cache entries to find the nodes for which to store a reference. Additionally, as the update procedure may change the extent of a cache entry influence zone and reduce the number of influenced octree nodes, we rebuild the octree from scratch at particular time intervals (Table 1). This overall reduces the total number of references (Section 7).

5. Parallelization of cache entry computations

Neighboring rays are likely to issue a scattering event at similar positions, especially if they encounter optically dense material (Fig. 7). This may lead to too dense and redundant cache regions despite low frequency irradiance variation.

For solving this issue, we evaluated two different approaches. The first method forbids simultaneous creation of more than one entry per octree node. Since the octree resides in the object-space of the volume, we call this approach *object-space locking*. The second technique is based on the fact that the creation of new cache entries is driven by image-based ray generation. Hence, locking parts of the screen in form of superpixels is an alternative. We call this scheme *screen-space locking*.

For both versions, upon requesting a new entry, we first try to lock the corresponding primitive (node or superpixel) using atomic operations, similar to Debattista *et al.* [DDPdSC11]. Upon success, we issue the request for creating a new cache entry, and, as soon as its estimation has finished, we insert it into the octree and release the lock. Requests for already locked primitives are discarded and the sample of the scattering event is shaded with MCVR.

5.1. Object-space locking

In this technique, each octree node can be separately locked. If we cannot extrapolate the irradiance from the cache, we check whether we can lock the node furthest down in the branch enclosing the event position. Improving over the method of Debattista *et al.* [DDPdSC11], however, our approach also allows parallel, asynchronous determination of the positions where new cache entries need to be created.

The effects of this approach are twofold. First, it allows parallel creation of multiple cache entries in distant parts along a single ray. Second, the octree adapts its local resolution to cache density. (Figure 8).

5.2. Screen-space locking

In contrast to the aforementioned method, *screen-space locking* operates on rectangular regions of the screen (superpixels). For an optimal distribution of requests, we use two restrictions. First, each superpixel may only issue one request at a time. We limit the total number of cache entry requests throughout the whole screen to balance workload.

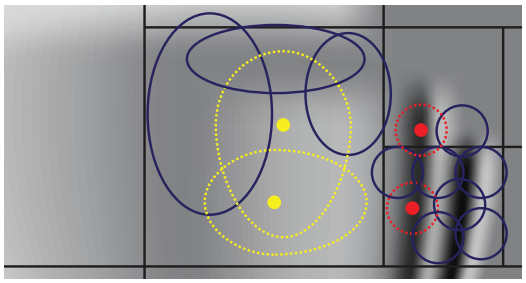

Figure 8: *As the octree nodes are small in the areas of high cache density, more cache entries can be created in these areas in parallel. This does not lead to the creation of redundant cache entries, because the new entries are likely to have small radii as a high cache density suggests high frequency of irradiance change in this area. In this illustration, the simultaneous creation of two entries, shown in red, will be allowed, while it will be forbidden for the two yellow ones, even though the distance between them is the same.*

Only a fraction of all rays in a superpixel are actually in need of a new entry, since some may have encountered available cache information in a particular iteration. The ratio of requests to total rays bears information on the projected cache distribution in the current region. We only issue creation of a new cache entry, if a randomized rejection test based on the ratio passes. This method allows for accessing the same node of the octree from different superpixels on the screen. Consecutively, for preventing conflicts, we use atomic operations on the reference list per node.

6. Priorization

To control the assignment of available processing power to procedures, we use a two-level priorization. First, we dynamically adjust the ratio of processing time spent on each of the three procedures based on the cache hit rate. Second, we use different priorities for the individual image regions during the rendering procedure to direct processing power towards parts further from convergence.

Cache status directed scheduling. If the number of cache entries ensures that the majority of lighting computation requests can be extrapolated from the cache, it is not necessary to allocate time for creating additional cache entries. To achieve this goal, we dynamically adjust the time frame assigned to the individual procedures based on the cache hit rate (h). We estimate h over a small time interval to consider multiple depths for each pixel. The time frames are then computed as: $t_c = t_{c,min} + (1 - h^{e_c}) \cdot (t_{c,max} - t_{c,min})$, where t_c corresponds to the relative time frame assigned to the cache creation procedure, $t_{c,min}$ and $t_{c,max}$ are the minimum and maximum time that should be used for cache creation, respectively. e_c allows to control a non-linear translation from

hit rate to assigned time. Based on the intuition that the number of required cache updates should be proportional to the number of newly created cache entries, we use the same formula with different minimum and maximum times for the cache update procedure. The relative time spent on rendering simply corresponds to the remaining time (parameters given in Table 1).

Rendering image priorities. Based on the geometric relations of the scene and lighting, the convergence rate of individual parts of the image may differ strongly. For instance, a region completely in shadow may converge to black within a single iteration, while complex light interactions from multiple light sources will result in very slow convergence rates. Thus, we want to provide a more uniform convergence rate across the entire image. We do that by estimating the expected gain from running another iteration per region. If we assume that the sample variance does not change after an additional iteration, then the error estimate will be reduced by

$$\Delta E = 1.96 \left(\sqrt{\frac{S^2(N)}{N}} - \sqrt{\frac{S^2(N)}{N+1}} \right) \tag{12}$$

Using ΔE, we compute the average expected error reduction for each block and use it directly rendering priority.

7. Implementation

An essential part of our approach is the concurrent execution of cache creation, cache update and rendering. Using the traditional execution model based on GPU shaders or SIMD languages like CUDA, a dynamic concurrent execution cannot be set up easily. Even with the most recent *dynamic parallelism*, which allows kernel launches from the GPU, we were unable to implement concurrent caching and rendering efficiently. The overhead of dynamic parallelism itself and finding a mapping between launched threads and actual work was too high in our experiments. Thus, we use *Softshell* [SKK*12], an open-source framework, which occupies the GPU to provide custom scheduling in software. Defining the execution steps in Softshell is done via *procedures*. Each procedure represents an executable function to be launched dynamically from the GPU.

To influence scheduling on the GPU, priorities can be assigned to individual procedure calls. In this way, the available processing power can be dynamically assigned to the most important procedures. A synchronized time basis provided in Softshell enables recording of the time spent on individual tasks. Using these measurements to update procedure priorities, we can schedule procedures in such a way that they receive predefined portions of the overall available processing time. For instance, we could use 20% for cache creation and 80% for rendering.

Each procedure is run in blocks of 128 threads. The main rendering procedure divides the screen into blocks of 16×8 pixels and performs ray casting with one pixel per thread. Each thread in the cache entry creation procedure casts seven rays towards randomly selected lights – one ray for the central position and one for each of the six positions offset in both directions of the axes of local coordinate frame. Then, the results are combined using shared memory. If the stopping criterion is met (Eq. (5)), the newly created cache entry is inserted into the octree, which uses atomic operations to avoid potential race conditions. Otherwise, the process is repeated. Finally, each cache entry update thread randomly selects one of the pixels within the 16×8 pixel block, casts a ray to find a scattering point and performs cache entry radius reduction if necessary. Random selection of a pixel avoids updating the cache in neighboring object-space positions.

Table 1: *The parameter values used in our experiments.*

Cache entry creation		
Estimation accuracy	γ	0.1
Validity radius threshold	ε	0.1
Position offset	Δ	2 voxels
Minimum influence zone radius	-	0.01 voxels
Maximum influence zone radius	-	16 voxels
Cache entry update		
Cache entry update threshold	-	0.1
Procedure time distribution		
Max. creation procedure fraction	$t_{c,max}$	15%
Min. creation procedure fraction	$t_{c,min}$	5%
Max. update procedure fraction	$t_{u,max}$	2%
Min. update procedure fraction	$t_{u,min}$	1%
Exponent for fractions update	e_c	3
Screen-space locking		
Max. number of simultaneously created cache entries	-	1000
Max. number of simultaneously created entries per superpixel	-	1
Miscellaneous		
Octree rebuild interval	-	1000 ms
Cache hit rate update interval	-	100 ms

8. Results

We evaluate several conditions of our proposed method in comparison to plain MCVR. In the following, we discuss cache creation in static scenes and during interaction, and also evaluate convergence rates for a fully built cache. Furthermore, we provide measurements for comparing the behavior of object- vs. screen-space locking. For our experiments, we used the following datasets: Bonsai ($512 \times 512 \times 182$), Manix ($512 \times 512 \times 460$), Macoessix ($512 \times 512 \times 460$) in full, half and quarter resolution each.

In Table 3, we list both the total number of cache entries for our three tested volumes, as well as the total number

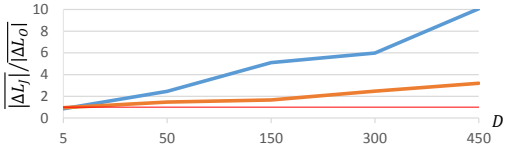

Figure 9: *The ratio of average relative error of extrapolated radiance values computed using the approach presented in [JDZJ08] ($\overline{|\Delta L_J|}$) to the ones computed with our method ($\overline{|\Delta L_O|}$) for scene with a single light source (blue) and with additional background illumination (orange). The red line shows the value of* 1 *which means no difference in average errors of two methods. The horizontal axis shows the global density factor D which is used to convert transfer function values in range* $[0,1]$ *to the density of participating medium. Our method is significantly more accurate for higher gradient magnitudes, which are caused by larger global volume density factor D or by the light setup. Lower estimation errors for our method lead to the lower number of cache entries required to represent the irradiance field.*

of references stored in the multi-reference octree for object-space locking. During all our experiments, we measured an average of six to seven references for each cache entry. In fact, these additional five to six references per entry, stored as indices, are a good trade-off memory-wise. Single referencing induces the necessity of searching through neighbor nodes on each level along an octree branch leading to significantly higher number of costly global memory accesses, decreasing the performance.

Extrapolation accuracy for single cache entries. We compared the radiance extrapolation accuracy computed using the method by Jarosz *et al.* [JDZJ08] with our method (Section 4.2). The radiance for our method is obtained by multiplying the irradiance value by the isotropic phase function value and scattering coefficient. Figure 9 shows that with increasing global density, i. e., growing gradient magnitudes, the Jarosz method, which is relying on only a single gradient value, becomes insufficient to capture the high frequency changes in the irradiance field. By comparison, our method produces drastically lower error, quickly amortizing the memory and computation overhead (a factor of 2-4×) required for the more complex cache entry creation. Higher gradients induced by the lack of background illumination cause similar effects.

Behavior during and after interaction. During interaction with the scene, new regions of the dataset may appear which are not covered by the cache yet. However, we implicitly exploit frame-to-frame coherence, since previous regions rarely vanish instantly, and we generate new entries on-the-fly. As expected, stopping interaction leads to quickly and drastic decrease in error, whereas plain MCVR starts from scratch. Figure 10 shows average error rates for the

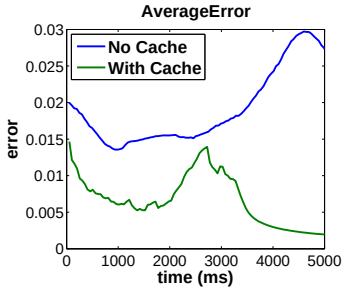

Figure 10: *Average error observed during rotating Bonsai by 360 degrees. Even during interaction, our method (green) has less error compared to plain MCVR (blue). Stopping interaction (3300 ms) leads to rapid convergence due to present cache information.*

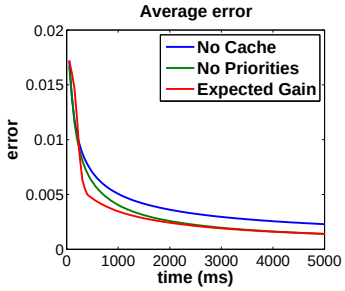

Figure 11: *Average error progression observed for the static Bonsai with initially empty cache. Both rendering with priorities (red) and without priorities (green) achieve faster convergence than plain MCVR (blue).*

Bonsai during rotation by 360 degrees and then stopping interaction.

Convergence rates in static scenes. Static scenes starting with an empty cache and without user interaction are a hard case for a cache-based algorithm, since a lot of noise is accumulated initially. We consider three different lighting setups. First, we use a single small, distant area light. Second, we use a setup with three very intense, large area lights close to the volume. Third, we use four rather dim, large area lights surrounding the volume.

For each of these light setups, we have measured the convergence rate of our approach compared to plain MCVR using the error estimate shown on the left side of Eq. (5). This definition of the error per pixel in screen-space allows us to locally estimate the difference to an optimal solution. Since we observed very similar outcomes for all scene experiments, we provide the average ratio of plain MCVR versus our irradiance cache error over 200 frames in Table 2 and Figure 11 (plots in supplemental material). Note that despite the difficult setting, our method consistently outperforms plain MCVR event without priorities.

Table 2: *Comparison of plain MCVR error estimates with our irradiance caching method. The numbers express the ratio of plain MCVR error over irradiance caching error, averaged over 200 frames. The higher the number, the faster our approach converges. The values in bold show the ratio with "expected gain" priorities enabled. Despite static scenes, we achieve drastically lower average error for all cases.*

	512x512x* ; 1280x720			512x512x* ; 1920x1080			256x256x* ; 1920x1080		
Lights	Bonsai	Manix	Macoessix	Bonsai	Manix	Macoessix	Bonsai	Manix	Macoessix
1	1.58 **1.62**	1.49 **1.55**	1.65 **1.70**	1.42 **1.65**	1.44 **1.57**	1.64 **1.74**	1.94 **1.99**	1.89 **2.29**	1.42 **1.58**
3	1.38 **1.34**	1.23 **1.28**	2.67 **2.56**	1.28 **1.38**	1.24 **1.31**	2.72 **2.78**	3.36 **3.51**	1.71 **2.40**	2.55 **2.76**
4	1.20 **1.29**	1.23 **1.25**	2.56 **2.46**	1.23 **1.34**	1.12 **1.22**	2.42 **2.57**	4.03 **3.89**	2.03 **2.05**	2.86 **2.95**

Object-space vs screen-space locking. As discussed in Section 5, we evaluate two different methods for preventing creation of excessive amount of redundant cache entries at once. In our first measurement, we rotate the volume by 360 degrees and record the number of cache entries and tree references. We show the results for the detailed structures of the Bonsai dataset. In a second, more extensive test, we focus on static scenes showing Manix and Bonsai with varying parameters. These include volume resolution (512^3, 256^3, 128^3), screen resolution (1920×1080, 1280×720, 640×480) as well as background illumination (on, off), while the viewpoint is fixed. Our tests show very marginal differences in convergence rates. For low screen resolutions, screen-space locking seems to create entries in more beneficial locations and achieves the desired cache hit rates and convergence values slightly faster. For high screen resolutions, object-space locking creates less redundant entries and achieves high cache hit rates faster with comparable cache sizes. For detailed outcomes of all parameter combinations, please refer to the supplemental material.

Table 3: *Cache sizes observed for object-space locking. The reference counters refer to the total number in the multi-reference octree.*

Volume	Cache Entries	References
Bonsai	32421	193939
Manix	30755	206807
Macoessix	24774	150660

9. Conclusion and future work

We have presented a method to create an irradiance cache for volumetric data in parallel, concurrently to MCVR. Furthermore, we have proposed two techniques for preventing the creation of redundant cache entries, thus keeping our memory footprint small. Additionally, we have discussed a new irradiance extrapolation method, which outperforms previous approaches. Our experiments illustrate a drastic increase in convergence rate when compared to standard MCVR, even in the most difficult situations for our approach. Especially during and after interaction, our method achieves a considerable improvement over MCVR, outperforming it by up to four times.

In future work, we will incorporate multiple scattering, as well as adopt spherical harmonics to allow for anisotropic phase functions and camera-dependent effects. Additionally, we will investigate different data structures for caching.

References

[Cha60] CHANDRASEKHAR S.: *Radiative Transfer*. Dover Books on Intermediate and Advanced Mathematics. Dover Publications, 1960. 1, 2

[DDPdSC11] DEBATTISTA K., DUBLA P., PEIXOTO DOS SANTOS L., CHALMERS A.: Wait-free shared-memory irradiance caching. *Computer Graphics and Applications, IEEE 31*, 5 (2011), 66–78. 3, 7

[DSC06] DEBATTISTA K., SANTOS L. P., CHALMERS A.: Accelerating the Irradiance Cache through Parallel Component-Based Rendering . Heirich A., Raffin B., dos Santos L. P., (Eds.), Eurographics Association, pp. 27–34. 3

[DVND10] DIAZ J., VAZQUEZ P.-P., NAVAZO I., DUGUET F.: Real-time ambient occlusion and halos with summed area tables. *Computers and Graphics 34*, 4 (2010), 337 – 350. 2

[GKBP05] GAUTRON P., KŘIVÁNEK J., BOUATOUCH K., PATTANAIK S.: Radiance cache splatting: A gpu-friendly global illumination algorithm. In *Proceedings of the Sixteenth Eurographics Conference on Rendering Techniques* (Aire-la-Ville, Switzerland, Switzerland, 2005), EGSR'05, Eurographics Association, pp. 55–64. 3

[HGI08] HOLMES M., GRAY A., ISBELL C.: Ultrafast monte carlo for statistical summations. In *Advances in Neural Information Processing Systems 20*, Platt J., Koller D., Singer Y., Roweis S., (Eds.). MIT Press, Cambridge, MA, 2008, pp. 673–680. 5

[JC98] JENSEN H. W., CHRISTENSEN P. H.: Efficient simulation of light transport in scences with participating media using photon maps. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 1998), SIGGRAPH '98, ACM, pp. 311–320. 2

[JDZJ08] JAROSZ W., DONNER C., ZWICKER M., JENSEN H. W.: Radiance caching for participating media. *ACM Transactions on Graphics 27*, 1 (Mar. 2008), 7:1–7:11. 3, 5, 6, 9

[JKRY12] JÖNSSON D., KRONANDER J., ROPINSKI T., YNNERMAN A.: Historygrams: Enabling interactive global illumination in direct volume rendering using photon mapping. *Visualization and Computer Graphics, IEEE Transactions on 18*, 12 (2012), 2364–2371. 2

[JSYR13] JÖNSSON D., SUNDÉN E., YNNERMAN A., ROPINSKI T.: A Survey of Volumetric Illumination Techniques for Interactive Volume Rendering. *Computer Graphics Forum (conditionally accepted)* (2013). 2

[Kar11] KARLIK O.: *Data Structures for Interpolation of Illumination with Radiance and Irradiance Caching*. Master's thesis, Czech Technical University in Prague, 2011. 7

[KBPv08] KŘIVÁNEK J., BOUATOUCH K., PATTANAIK S., ŽÁRA J.: Making radiance and irradiance caching practical: adaptive caching and neighbor clamping. In *ACM SIGGRAPH 2008 classes* (New York, NY, USA, 2008), SIGGRAPH '08, ACM, pp. 77:1–77:12. 6

[KG09] KŘIVÁNEK J., GAUTRON P.: Practical global illumination with irradiance caching. *Synthesis Lectures on Computer Graphics and Animation 4*, 1 (2009), 1–148. 6

[KGPB05] KŘIVÁNEK J., GAUTRON P., PATTANAIK S., BOUATOUCH K.: Radiance caching for efficient global illumination computation. *Visualization and Computer Graphics, IEEE Transactions on 11*, 5 (2005), 550–561. 2, 3

[KMG99] KOHOLKA R., MAYER H., GOLLER A.: Mpi-parallelized radiance on sgi cow and smp. In *Parallel Computation*, Zinterhof P., VajterÄĚÂąic M., Uhl A., (Eds.), vol. 1557 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 1999, pp. 549–558. 3

[KPB12] KROES T., POST F. H., BOTHA C. P.: Exposure render: An interactive photo-realistic volume rendering framework. *PLoS ONE 7*, 7 (07 2012), e38586. 2, 3

[KVH84] KAJIYA J. T., VON HERZEN B. P.: Ray tracing volume densities. *SIGGRAPH Comput. Graph. 18*, 3 (Jan. 1984), 165–174. 2

[RCB11a] RIBARDIÈRE M., CARRÉ S., BOUATOUCH K.: Adaptive records for irradiance caching. *Comput. Graph. Forum 30*, 6 (2011), 1603–1616. 3

[RCB11b] RIBARDIÈRE M., CARRÉ S., BOUATOUCH K.: Adaptive records for volume irradiance caching. *The Visual Computer 27*, 6-8 (2011), 655–664. 3

[RCLL99] ROBERTSON D., CAMPBELL K., LAU S., LIGOCKI T.: Parallelization of radiance for real time interactive lighting visualization walkthroughs. In *Supercomputing, ACM/IEEE 1999 Conference* (1999), pp. 61–61. 3

[RMSD*08] ROPINSKI T., MEYER-SPRADOW J., DIEPENBROCK S., MENSMANN J., HINRICHS K. H.: Interactive volume rendering with dynamic ambient occlusion and color bleeding. *Computer Graphics Forum (Eurographics 2008) 27*, 2 (2008), 567–576. 2

[SA07] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on gpus. In *Proceedings of the I3D, ACM* (2007). 2

[Sal07] SALAMA C. R.: Gpu-based monte-carlo volume raycasting. In *Proceedings of the 15th Pacific Conference on Computer Graphics and Applications* (Washington, DC, USA, 2007), IEEE Computer Society, pp. 411–414. 2

[SKK*12] STEINBERGER M., KAINZ B., KERBL B., HAUSWIESNER S., KENZEL M., SCHMALSTIEG D.: Softshell: dynamic scheduling on gpus. *ACM Trans. Graph. 31*, 6 (Nov. 2012), 161:1–161:11. 4, 8

[SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Trans. Graph. 21*, 3 (July 2002), 527–536. 2

[SMP11] SCHLEGEL P., MAKHINYA M., PAJAROLA R.: Extinction-based shading and illumination in gpu volume raycasting. *IEEE TVCG 17*, 12 (2011), 1795–1802. 2

[Sta95] STAM J.: Multiple scattering as a diffusion process. In *Rendering Techniques 1995*, Hanrahan P. M., Purgathofer W., (Eds.), Eurographics. Springer Vienna, 1995, pp. 41–50. 2

[TFCRS11] THOMPSON W., FLEMING R., CREEM-REGEHR S., STEFANUCCI J. K.: *Visual Perception from a Computer Graphics Perspective*. A K Peters/CRC Press, 2011. 5

[Vea98] VEACH E.: *Robust monte carlo methods for light transport simulation*. PhD thesis, Stanford University, Stanford, CA, USA, 1998. AAI9837162. 5

[WH92] WARD G. J., HECKBERT P. S.: Irradiance Gradients. *1992 Eurographics Workshop on Rendering* (1992), 85–98. 3

[WKSD13] WEBER C., KAPLANYAN A. S., STAMMINGER M., DACHSBACHER C.: Interactive direct volume rendering with many-light methods and transmittance caching. *Proceedings of the Vision, Modeling, and Visualization Workshop* (2013). 3

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. *SIGGRAPH Comput. Graph. 22*, 4 (June 1988), 85–92. 2, 3

[ZD13] ZHANG Y., DONG ZHAO MA K.-L.: Real-time volume rendering in dynamic lighting environments using precomputed photon mapping. *IEEE TVCG 19*, 8 (aug 2013), 1317–1330. 2

[ZM13] ZHANG Y., MA K.-L.: Fast global illumination for interactive volume visualization. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2013), I3D '13, ACM, pp. 55–62. 2