Eurographics Symposium on Rendering (DL-only Track) (2020)
C. Dachsbacher and M. Pharr (Editors)

Temporal Sample Reuse for Next Event Estimation and Path Guiding for Real-Time Path Tracing

Addis Dittebrandt, Johannes Hanika and Carsten Dachsbacher

Karlsruhe Institute of Technology, Institute for Visualization and Data Analysis, Germany

Abstract

Good importance sampling is crucial for real-time path tracing where only low sample budgets are possible. We present two efficient sampling techniques tailored for massively-parallel GPU path tracing which improve next event estimation (NEE) for rendering with many light sources and sampling of indirect illumination. As sampling densities need to vary spatially, we use an octree structure in world space and introduce algorithms to continuously adapt the partitioning and distribution of the sampling budget. Both sampling techniques exploit temporal coherence by reusing samples from the previous frame: For NEE we collect sampled, unoccluded light sources and show how to deduplicate, but also diffuse this information to efficiently sample light sources in the subsequent frame. For sampling indirect illumination, we present a compressed directional quadtree structure which is iteratively adapted towards high-energy directions using samples from the previous frame. The updates and rebuilding of all data structures takes about 1ms in our test scenes, and adds about 6ms at 1080p to the path tracing time compared to using state-of-the-art light hierarchies and BRDF sampling. We show that this additional effort reduces noise in terms of mean squared error by at least one order of magnitude in many situations.

CCS Concepts
• ***Computing methodologies*** $\rightarrow$ ***Ray tracing; Visibility;***

1. Introduction

Rasterization has been the standard real-time rendering algorithm for decades since its early implementation in hardware and efficiency. Current graphics processing units (GPUs) are capable of processing hundreds of millions of primitives per frame at interactive rates. However, rasterization determines primary visibility and approximating complex or global illumination is left to secondary techniques resulting in complex and deep pipelines. For example, shadow computation although conceptually simple resulted in myriads of techniques [ESAW11], shading with many lights requires efficient culling [OBA12] and intricate shadow computation [OSK*14]. For indirect illumination, various approaches have been adapted to rasterization [KD10; CNS*11; RDGK12].

The recent advent of hardware ray tracing enables far more flexible real-time rendering algorithms. In particular, this includes the introduction of path tracing, which has been the standard in and exclusive to offline rendering for many years [FHH*19]. However, path tracing (or Monte Carlo techniques in general) comes at a cost: Noise. With real-time budgets, only few paths (or even just one) can be computed per pixel, leading to severely noisy images. Many recent approaches attempt to *denoise* these images by leveraging temporal coherence between successive frames as well as spatial coherence within a frame [MMBJ17; CKS*17; SKW*17]. Furthermore, denoising generally introduces artifacts like lag and blurriness which stem from this temporal and spatial reuse. Instead of only "repairing" images after rendering, it is thus essential to reduce the image noise by improving the sampling quality. This importance sampling, however, is challenging even in offline rendering (see e.g. recent work [VHH*19]) as the shape of the integrand is generally unknown.

In this paper we improve importance sampling for real-time path tracing in dynamic scenes, where the dynamic content requires the sampling to adapt over time. Obviously, these sampling techniques must not require precomputation exceeding the budget within one frame. To this end, we exploit, similar to many of the aforementioned approaches, temporal coherence by reusing the drawn samples (visibility queries) in one frame to construct improved sampling densities for the subsequent one. We focus on two pressing aspects in current real-time path tracing: next event estimation (NEE) with many light sources, and sampling of indirect illumination.

With many lights and complex geometry, the decision which light to sample for NEE is nontrivial. However, often the decisive factor is visibility, i.e. only light sources that are not blocked by scene geometry can contribute to the shading of a surface point. To

DOI: 10.2312/sr.20201135

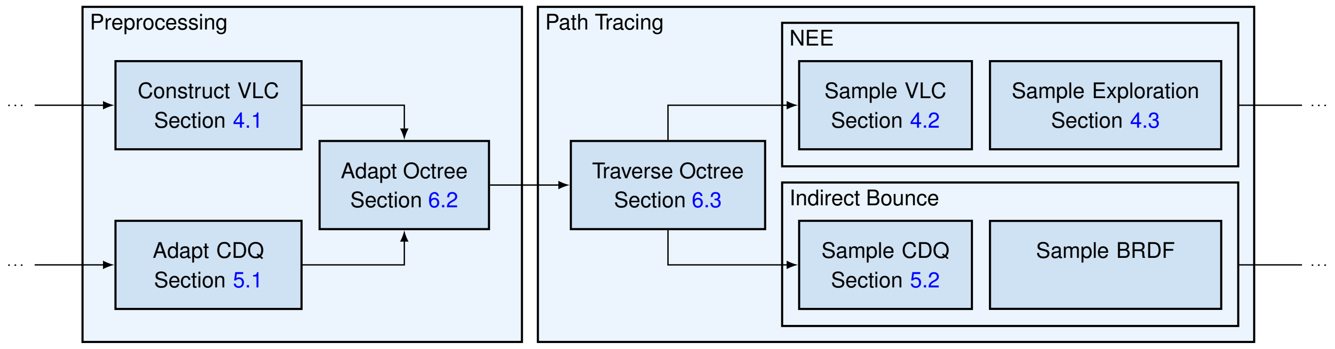

Figure 1: *Execution of components over a frame, split into a preprocessing and path tracing step. Arrows represent dependencies. Each leaf of the octree contains a unique Visibility Light Cache (VLC) and Compressed Directional Quadtree (CDQ) to account for spatial variation. During preprocessing, each VLC/CDQ and the octree itself is constructed/adapted based on samples of the previous frame. During path tracing, the adapted octree is traversed to access VLC and CDQ for importance sampling of visible lights (NEE) and high-energy directions (Indirect Bounce), respectively. Secondary techniques are used to explore new lights and to importance-sample directions according to BRDF.*

this end, we store lights that had visible samples in the previous frame in a Visibility Light Cache (VLC, section 4). Sampling then focuses on this set of lights (but also includes previously occluded lights). With a high-quality importance sampled selection based on Linearly Transformed Cosines (LTCs) for area lights [HDHN16], we achieve sampling closer to product importance sampling.

When computing indirect illumination, standard BSDF sampling is inadequate and causes fireflies when incident light covers small solid angles. To improve this sampling step, we introduce a Compressed Directional Quadtree (CDQ, section 5) inspired by offline path guiding [MGN17]. Our trees require only very few bits (32-128 bits). This compact representation trades memory usage for sampling quality, but is tractable for real-time and achieves significant variance reduction in many cases.

Obviously visibilities and sampling densities vary over the scene geometry. To adapt spatially, we partition the scene using an octree and provide individual estimators (VLCs and CDQs) per leaf node (section 6). The key is that we adapt this octree structure temporally based on the number of samples that fall within each leaf node. For this, we introduce a simple split/collapse algorithm with configurable thresholds. The advantage is that this structure automatically balances the available samples among all leaves, improving robustness of our techniques (see section 4.1 for caveats due to the fixed subdivision scheme). With a readily available octree structure *before* path tracing, we can also preallocate space for storing samples, instead of rearranging them in memory afterwards.

To summarize, the key contributions of this paper are:

- a Visibility Light Cache for product importance sampling NEE,
- a Compressed Directional Quadtree for guiding sampling of indirect illumination,
- a temporally- and sample-adaptive octree to organize estimators.

2. Overview

The VLC, CDQ and octree are mostly independent techniques and are discussed in the sections 4, 5 and 6, respectively. This section serves as a high-level overview on the combined interaction of their components as shown in figure 1. We divide execution of each frame into the two passes preprocessing and path tracing. During preprocessing, samples from the previous path tracing are used to construct the VLC and adapt the CDQ of each octree leaf node. For the VLC, the set of lights with unoccluded samples is computed (section 4.1) and the CDQ is adapted towards high-energy directions (section 5.1). The observed sample counts are used to adapt the octree with a simple threshold-based split-collapse scheme (section 6.2).

During path tracing, the octree is traversed at each surface interaction to access the contained VLC and CDQ (section 6.3). For Next Event Estimation (NEE), the VLC is used to select a visible light source (section 4.2). To discover new lights and ensure unbiasedness, a secondary exploration technique is used (section 4.3). The decision on which technique (VLC/exploration) to use is made stochastically based on a configurable fraction (typically around 10% for exploration). For the indirect bounce, the CDQ is used to sample a high-enery direction (section 5.2). Just as with NEE, a secondary technique is used. The BRDF is used in this instance with a configurable fraction (typically around 50%) to bound noise in the case of highly specular materials. All techniques are combined with MIS and the balance heuristic. To achieve this, they are specifially designed for efficient PDF computation (the VLC in particular).

3. Background and Related Work

Light transport Illumination on surfaces is computed by integrating the incident radiance field multiplied by the bidirectional scattering distribution function (BSDF), which describes the reflectance properties of the surface. This integration is usually performed using the Monte Carlo method, i.e. by constructing estimators which employ random sampling. The probability density function (PDF) which determines the distribution of the samples has a large impact on the error of the estimate, manifesting itself in variance. Thus, the search for good PDF is of great importance and central to our work. In case there are multiple candidate PDF which perform well in some parts of the integrand, it is possible to opti-

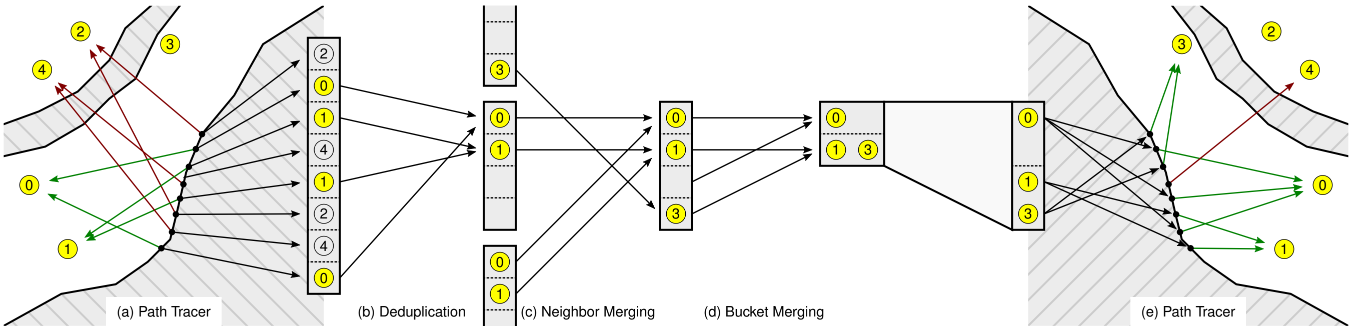

Figure 2: *Overview of Visibility Light Cache (VLC). (a) Samples from previous frame are placed into a region-specific buffer location. (b) Visible samples are deduplicated through a hash table. (c) Neighbor hash tables are merged to diffuse information (light 3 was not present in hash table). (d) Buckets of hash table are merged to minimize bucket count (capacity: 2). (e) Path tracer in next frame uses VLC. Buckets are selected uniformly. Lights in bucket are then importance sampled. For continued exploration, a fraction of samples is created with an exploration strategy (sample for light 2).*

mally combine these into a joint esimator by multiple importance sampling [VG95].

Direct illumination One particularly efficient sampling strategy is *next event estimation*, which connects transport path vertices to the anticipated next event on the light sources. Intuitively it computes the direct illumination contribution, and has been studied extensively [SWZ96]. Further work has refined this to build hierarchical acceleration structures for the emitters [WFA*05] and used these to produce unbiased estimators [DWB*06]. This approach has been perfected in production renderers [CK18], extended with BRDF awareness [LXY19] and refined for special cases like directional emission and volume scattering [FHH*19].

Another line of work to reduce the number of light sources to consider at every path vertex sorts light sources by contribution [War94] and more recent work considers shorter, learned lists of lights [VHH*19, Keller's presentation]. Similar considerations lead to hierarchical structures [VK16; VKK18]. To gain real-time performance, the contribution of light sources has been evaluated analytically [HDG99; Arv95; HDHN16], and fast ways to sample a list of lights on the GPU have been devised [KWR*17; Wal77]. We use a combination of the above ideas, and propose to use a learned structure for next event estimation in conjunction with a *path guiding* method to improve indirect lighting contributions.

Path guiding These methods have been around for some time [Jen95; HP02] but only relatively recently gained some traction because improved algorithms [VKŠ*14; MGN17; DK17] made path guiding useful in practical applications [VHH*19]. In these approaches, guide cache records are stored throughout the scene and are used to construct sampling densities that follow the incident radiance field. Ideally, these distributions also include the BSDF [HEV*16], but this comes at a performance penalty that is yet to be overcome for real time applications. We employ path guiding for indirect lighting, and the parts of the transport that are not handled by our next event estimation technique (non-geometric light sources such as an environment map).

Data structures Guiding caches store an approximation of a directional distribution, by means of Gaussian mixtures [VKŠ*14] or quad trees [MGN17]. We use the quad tree approach, and use a special compression technique suitable for the GPU [Jac89], though there are possible alternative encodings [DIP14]. To store the cache records themselves in an index structure, we use a sparse octree. It has been shown that hierarchical hashed grids can be very effective on the GPU as well [BFK19].

4. Visibility Light Cache for Next Event Estimation

Explicit sampling of light sources via NEE is a critical aspect for path tracers, since light sources can become very small and therefore only cover a very small solid angle (or none when handling point light sources) while still providing much contribution. This makes BSDF sampling ineffective, causing fireflies. Additionally, if light sources are many, we have to select one for sampling. This selection probability becomes quite important with increasing scene complexity. In particular, many light sources might not be visible, so selecting them would increase noise considerably. The limited path budget in real-time applications only aggravates this issue.

We therefore propose the Visibility Light Cache (VLC) which, for a given region, extracts set of visible lights from light samples in the previous frame. Figure 2 gives an overview of this process which is explained in more detail in section 4.1. From this set we select a light using high quality importance sampling with LTCs [HDHN16]. As we only consider visible light sources, we get closer to product importance sampling, but only under special circumstances; see section 4.2 for more details. As the VLC can only select lights that are already known, a secondary strategy is needed to explore new lights. This exploration strategy is presented in section 4.3.

4.1. VLC Construction

The construction algorithm takes light samples from the previous frame (figure 2 (a)) and extracts the set of visible lights to form the VLC used for importance sampling.

Deduplication (figure 2 (b)) is the primary task of extracting visible lights. A light sample consist of an identifier for the light, as well as a single bit that indicates whether the light was visible or not. A visible light can be represented by multiple visible samples, so we need to remove duplicates. We dispatch a compute shader where each work group handles one region. Samples are inserted into a hash table in shared memory with the light identifier as key. To handle collisions, we use a hash table with buckets (not shown in the figure). Invisible samples are discarded directly. We also tried to improve sampling by estimating fractional visibility from these samples, but this did not significantly improve the result in difficult situations. To bound computation time, only a fixed maximum number of the available samples is processed per region. To ensure representative selection, we gather samples all across the buffer.

We then merge the caches of neighbor regions (figure 2 (c)), to counteract that samples are not divided uniformly among the leaves. Due to our octree subdivision strategy (section 6) one leaf might receive almost all samples, while another leaf receives almost none. The former will just subdivide again to distribute the samples among more leaves, but the latter cannot compensate the low sample count. This is especially problematic with surfaces that are not exactly axis-aligned. As a result, the constructed VLC would be of poor quality (i.e. instability, as lights compete for few samples). Merging has the additional effect that newly discovered lights can propagate between neighboring caches. For occasions where the exploration strategy rarely samples an important light, this ensures that the light needs to be sampled by only one region, instead of by every region individually, leading to higher robustness.

Finally, the hash table is compacted (figure 2 (d)) to form the VLC which can then be used to sample a light source in the next frame (figure 2 (e)). This is described in more detail in section 4.2.

4.2. Importance Sampling of VLC

With the VLC, we want to perform high-quality selection of the contained area light sources. We leverage linearly transformed cosines (LTCs [HDHN16]) to compute sampling weights for each light. This technique was originally proposed for approximate shading with area light sources in closed-form. It can just as well be used for high-quality importance sampled selection of area lights with low variance, since the sampling weights are approximately proportional to the light contributions. Since only visible lights are considered, it gives almost perfect product importance sampled selection, assuming uniform emitters and full visibility of all lights to all contained surface points.

To sample a light source for a shading point, we construct a *cumulative distribution function* (CDF). As our importance measure incorporates information about the surface point itself, we cannot precompute the CDF for each region, but we instead have to do so on-demand for each surface point by linearly scanning through the lights. This, however, is very expensive even with modest light counts. Instead of iterating through all of them, the VLC is divided into a variable number of buckets with fixed capacity which the lights are distributed into. When selecting a light, a bucket can then first be chosen with uniform probability and only the lights contained in the single bucket are iterated over. For PDF computation, we need to efficiently retrieve the bucket that a light is possibly contained in. For that, we distribute lights deterministically among the buckets via hashing. In our instance, we use a plain modulo operation against the light identifier. Obviously, with many lights, the sampling quality degrades considerably with this technique, but it ensures a fixed computation budget, which is paramount for interactive rendering. To keep register pressure low, we do not store the CDF explicitly but traverse the bucket twice (once to accumulate all sample weights for normalization, and then to select a light source).

Algorithm 1: Branch-friendly VLC and Exploration sampling

```
Input: vlc, surface
Output: light, pdf

doExploration ← rng() < fraction;
if doExploration then
    light ← sampleExploration();                    // $\mathcal{O}(1)$
    bucket ← light.idx mod vlc.bucketCount;  // calc bucket
else // sample VLC
    light ← null;
    bucket ← $\lfloor$rng() · vlc.bucketCount$\rfloor$; // sample bucket
end

// first VLC traversal to accumulate weights
acc ← 0, weight ← 0;
for light′ ∈ vlc.buckets[bucket] do
    weight′ ← calcLightImportance(surface, light′);
    acc ← acc + weight′;
    if light = light′ then // find light for explor.
        weight ← weight′;
    end
end

if ¬doExploration then
    // second VLC traversal to select light
    light, weight ← sampleVLC(vlc, surface, bucket, acc);
end

pdf ← fraction · pdfExploration(light);             // $\mathcal{O}(1)$
pdf ← pdf + (1 − fraction) · weight/(acc · bucketCount);
```

The hashing method has the disadvantage that the buckets may not be filled evenly. As such, it is not clear how many buckets are needed given just the number of lights. We therefore construct the VLC by taking the hash table from section 4.1 and merge buckets bottom-up, as long as the capacity limit of each bucket is not exceeded (figure 2 (d)). Additionally, the light count may vary slightly from frame to frame, due to almost unimportant lights being rarely sampled. Normally, this is not problematic. But if this small variation causes a variation in the number of buckets, flickering artifacts appear. To combat this, we use a hysteresis approach: When merging buckets to a bucket count lower than in the previous frame, the capacity is artificially reduced to, e.g., 75%. After the final buckets are constructed, we sort the lights inside a bucket by their identifier to improve sampling stability.

4.3. Exploration Strategy

The VLC is only capable of selecting lights which were already selected in the previous frame. As such, new or yet unknown lights are never selected, resulting in a biased technique. A secondary "exploration" technique is needed which we combine with the VLC through MIS. The basic requirement for this technique is to be cheap to evaluate, ideally in constant time, since we already put a lot of computational effort into sampling the VLC with high quality. The sampling quality does not need to be especially good either, since we combine with the VLC right away. We thus opted for a constant-time sampling technique by precomputing a global PDF on the CPU over all light sources using the alias method [Wal77]. This is done every frame. The major problem of this approach is that the relationship between surface point and light source (e.g. distance, relative angles) is completely lost. We therefore use the following heuristic weights:

1. Energy of the light
2. Change of the light (e.g. energy and/or position)
3. Distance from light to camera

(1) gives higher weight to brighter lights. (2) gives higher weight to dynamic and/or appearing lights for improved reactivity. (3) is used as a cheap proxy for the distance between light and surface point. This is obviously only reasonable for points that are close to the camera. We compute separate PDFs for each heuristic. They are then blended with configurable weights and normalized to form the final PDF given as input to the alias method.

At each surface interaction, a stochastic decision is made to sample either the VLC or exploration technique through a configurable fraction. In most instances, no more than 10% of samples should be generated by the exploration technique, because the sampling quality is typically vastly inferior to the VLC, resulting in increased noise.

Since we combine both techniques through MIS, we need to efficiently compute the PDF of the other technique. For the exploration strategy, this requires a traversal through the bucket in which the sampled light might reside. Modern GPUs execute groups of threads in lockstep, but the technique is selected with a random number. Therefore virtually all groups have to execute both techniques. Thus, a naive implementation through separate code paths for VLC and exploration sampling would effectively cause three traversals of the VLC. But VLC traversal is the most expensive aspect, as it is computationally linear in bucket size. We resolve this issue by performing combined sampling of both techniques as shown in algorithm 1. The general idea is that we hijack the first traversal of the VLC, so that threads that have elected to use the exploration strategy use this traversal to also find the sampled light source in the VLC and write out the respective weight. Together with the accumulated weights, the PDF of the VLC can be computed. Just as with plain VLC sampling, two traversals are needed.

5. Compressed Directional Quadtree for Path Guiding

Guiding of indirect rays promises great variance reduction, especially if features with high contribution cover a small solid angle. But most techniques are exclusive to offline rendering due to needed precomputation and a large computational and memory overhead. We base our work on [MGN17] to achieve real-time path guiding. Directions are mapped to 2D through cylindrical coordinates. In this space, a directional quadtree structure is used for importance sampling which approximates the radiance field of a region. Regions are also embedded in the leaves of an octree structure over all surface points.

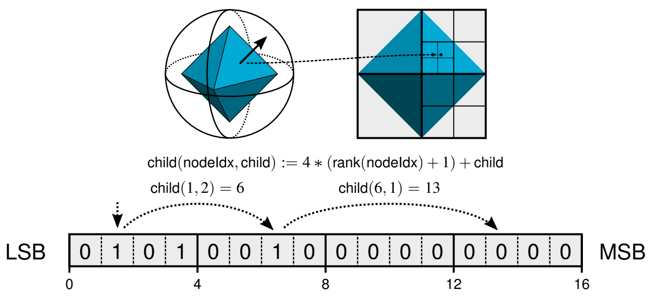

Figure 3: *Mapping of Sphere onto octahedron space, topology of quadtree as represented by a bit field and traversal of quadtree for a given direction. Quadtree is stored breadth first with a single bit per node. Rank operations are used to locate children of nodes.*

The problems for adopting this technique into a real-time context as-is are twofold: First, the technique does a lot of precomputation by drawing many samples just to build this quadtree structure. In fact, this is done iteratively with geometrically growing sample counts, such that the paths of these "learning" samples are guided as well. This approach is not feasible in real-time. Second, the octree occupies a large and varying amount of memory. Each node contains pointers to its immediate children as well as stochastic weights to guide traversal to important leaves. We already have to pay the cost of traversing the octree structure, so this can become prohibitively expensive.

We approach the memory problem by storing the quadtree in implicit form using a rank & select data structure [Jac89] with few bits (on the order of 32 - 128 bits). We use octahedron mapping [ED08] to map directions to 2D. Figure 3 shows an example instance. The tree is stored in breadth first order with each node being represented by a single bit. It denotes whether the node in question is an inner (1) or leaf node (0). We assume the root node to always be an inner node (a single leaf is not very useful for guiding). It is therefore always zero and does not need to be stored explicitly. This ensures that for bit field sizes which are a multiple of four (e.g. a 32-bit integer), all bits can actually be used. Rank & select operations are used to traverse the tree in both directions (see figures 3 and 5). As for the sampling weights, we simply omit them, and each leaf is given a uniform sampling weight instead. As such, a region is given higher sampling weight just by refining the quadtree towards this region. Of course, the resulting quadtree will be inferior to the original, but it will still improve sampling in difficult situations so that denoising can reconstruct the image without striking artifacts. Adaptation and sampling of this data structure are described in section 5.1 and 5.2, respectively.

5.1. CDQ Adaptation

We leverage temporal coherence to adapt a given quadtree with samples from the previous frame. We assume a fixed allocation of nodes (the maximum number of nodes that fit in the bit field). Adaptation happens by collapsing a selected inner node with only

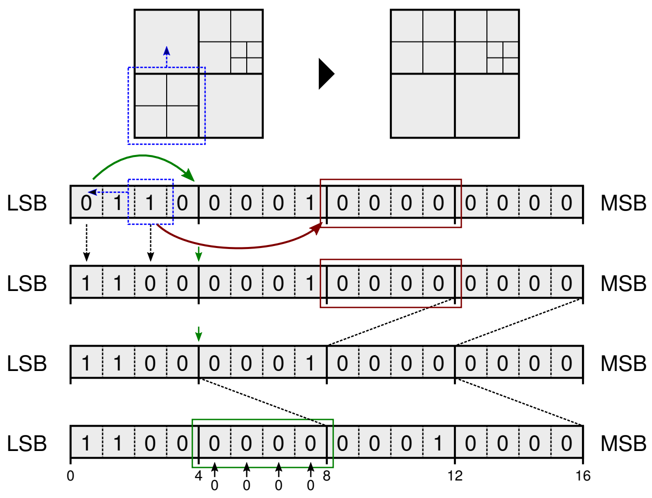

Figure 4: *CDQ adaption by leaf migration and implementation on bit field. First, children of selected nodes are computed. Then, bits of selected nodes are flipped. Deletion of old and insertion of new leaves is done by shifting all following bits. Finally, inserted leaves are zeroed.*

leaves as children and splitting a selected leaf node, i.e. the leaves are migrated between the selected nodes. As such the number of nodes always stays the same. We implement this adaptation directly on the bit field as depicted in figure 4, so that no uncompressed intermediate representation of the tree is needed. Essentially, we use two partial bit shifts to remove the old and to insert the new leaves in their respective location.

The nodes are selected based on the samples of the last frame. The samples are composed of their contribution and the direction (mapped into 2D by the octahedron map). To avoid adaptation towards light sources (they are already handled by NEE), we ignore light samples entirely. We run a compute shader with work groups consisting of a single subgroup (the collection of threads that execute in lockstep on the GPU). Each work group is responsible for one region. The samples are distributed among the leaves based on their respective direction and the squared contributions are accumulated in shared memory. We aggregate leaf contributions of the currently processed samples in the subgroup first. One thread per leaf is selected to accumulate the value in shared memory to avoid costly atomic float operations. We use squared contributions to have the CDQ adapt towards high-variance as well as high-energy regions (see [VKK18] eq. 5). We then run a min-max search: We select the leaf node with maximum contribution and the inner node with minimum combined contribution of its leaf children. Leaf migration is done only if the contribution of the leaf is higher than the combined contribution of the inner node. For stability, a configurable factor (i.e. 0.1) on the leaf contribution is used such that the leaf contribution must be significantly higher in order to warrant migration.

5.2. Importance Sampling of CDQ

We do not explicitly store any additional weights for stochastic CDQ traversal. But traversing all children with uniform probability

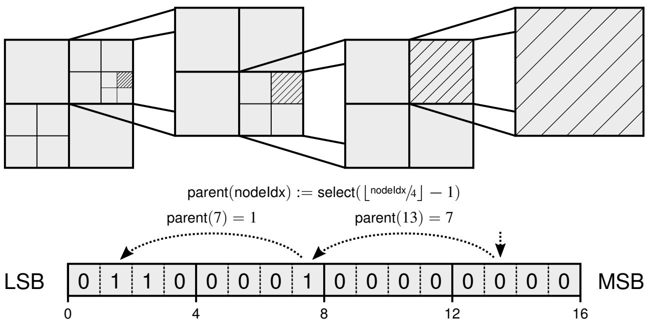

Figure 5: *Reconstruction of bounding box for a selected leaf. Successive select operations are used to rescale the bounding box into the parent frame until the root node is encountered.*

would result in overall uniform sampling. We instead approach this problem in a bottom-up fashion: We select a leaf node with uniform probability and then reconstruct its bounds by traversing its parents. Thus, a region in the quadtree that is more refined will be sampled more often, simply because more leaves are present in that region. The number of leaves is constant and the bit of leaf i can be computed through select(i) on the inverted bit field. Reconstruction is depicted in figure 5. With each traversed parent, the bounding box of the leaf is rescaled into the parent frame, until we arrive at the root node. The PDF is given as:

$$\text{PDF} = \frac{1}{A_{\text{leaf}}|\text{leaves}|} \frac{\|q\|_2^3}{4}$$

The first term is the probability of sampling a point on the octahedron map. A_{leaf} is the area of the containing leaf node and |leaves| is the total number of leaves. The second term is the correction factor which maps our PDF from the octahedron map to solid angle. q is a 3D vector obtained by mapping the sampled direction vector d onto the octahedron ($q = d/(|d_x| + |d_y| + |d_z|)$). To compute the PDF for a given direction (e.g. for MIS), we only need to access the bit field to compute A_{leaf} of the containing leaf node. This is achieved through a top-down traversal with which the bounding box is computed.

6. Temporal Sample-adaptive Octree

Visibility of lights as well as the direction of incoming contribution greatly depend on which surface point is considered. This makes the VLC and CDQ greatly depend on local information. We therefore cannot construct one global estimator for each technique, but instead have to do so for many regions in the scene. We need to organize these regions in a data structure with the following considerations: First, there must be enough samples available per region in order to construct stable estimators. Second, regions must be adapted under dynamic conditions in order to uphold the first consideration. Third, it must be accessible for both direct and indirect surface hits, so that the latter also benefits from the estimators.

We opted for an octree structure situated in world-space, where each leaf node defines a region for our estimators. The layout of this data structure in memory and the design decision leading to it are

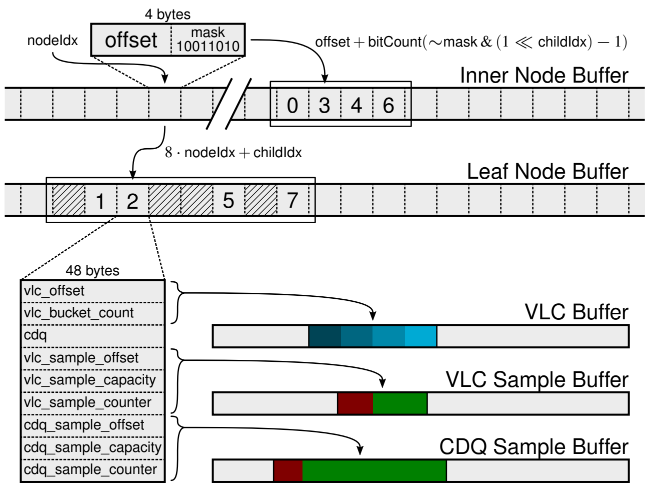

Figure 6: *Octree memory layout. We use two separate buffers for inner and leaf nodes. Inner nodes are composed of a 24-bit offset for the inner children and an 8-bit mask to encode leaves. Leaves are indexed implicitly instead of storing another offset per node. Inner children are stored compacted at the offset. Additional buffers (parent, neighbor and correspondence information) are not shown.*

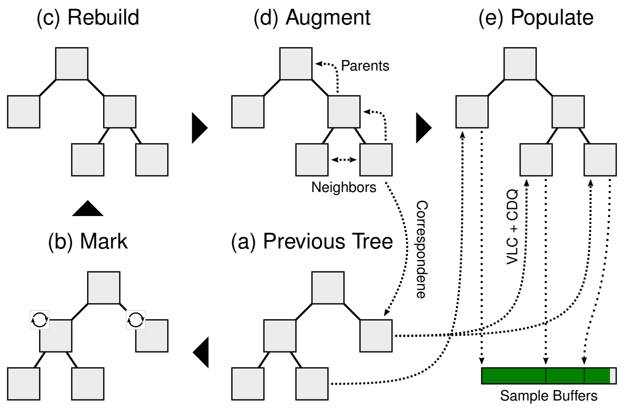

Figure 7: *Octree construction steps (visualized as binary tree) based on (a) previous tree. (b) Nodes are marked if they need to be split or collapsed based on the observed sample count. (c) The topology is rebuilt to account for marked nodes. (d) An augmentation pass adds parents to each node as well as neighbors and correspondence (equivalent node in previous tree) to each leaf. (e) This information is later used to populate the leaf structures (copy VLC pointers and CDQ, allocate space in sample buffers).*

given in section 6.1. Based on the collected per-leaf sample counts from the previous frame, the current octree is adapted through a simple split/collapse scheme as presented in section 6.2. Given a surface point, the octree is traversed top-down in order to access the corresponding estimators. Details are given in section 6.3.

6.1. Memory Layout

We optimized the memory layout such that most of the traversal only touches a small memory region (figure 6). We use two separate buffers for storing inner and leaf nodes. During traversal, only the first buffer needs to be considered until a leaf node is encountered. An inner node is composed of an 8-bit mask to encode which children are leaves (children are implicitly mapped to bits) and a 24-bit offset for the location of inner children. The inner children are stored at this location in a compacted way, i.e. there are no gaps if the current node also contains leaf children. To avoid storing another offset, leaves are indexed implicitly with the parent node index. While the node buffer remains small this way, this creates holes in the leaf buffer where the corresponding node is actually an inner node, ultimately leading to wasted space. But for a full tree with N children (each node has either zero or N children), given the number of inner nodes I, the total number of nodes is $N \cdot I + 1$. The wasted space, which is the ratio of inner nodes to the total number of nodes, is $I/(N \cdot I + 1)$. It converges to $1/N$ in the limit. As such, at most $1/8$ of storage is wasted in the leaf buffer. But overall, this waste is not the prime factor: Since the octree organizes surface samples, around half of all leaf nodes are empty anyways. We therefore focused on reducing the size of the inner node buffer, as it is accessed many times per traversal, while the leaf node buffer is accessed only once at the very end, thus giving the biggest potential in cache utilization.

Leaves store all necessary information for importance sampling and sample placement, namely offset and bucket count for the VLC buffer, the CDQ itself and offset, capacity and size triplets for the VLC and CDQ sample buffers. This amounts to a size of 48 bytes. We store leaves in an Array of Structures (AoS) format. We strongly recommend moving to a Structure of Arrays (SoA) format, as often only a single member (i.e. the CDQ) is needed for a particular purpose. Due to time constraints, we did not perform this rearrangement in our implementation.

6.2. Octree Adaptation

The octree needs to be continuously adapted in dynamic situations in order to ensure that each leaf node receives an adequate amount of samples (i.e. not too few and not too many). By observing the number of samples that each leaf received in the last frame, an adapted octree is constructed by splitting and/or collapsing nodes. The rough process is depicted in figure 7. It consists of four passes: Mark, rebuild, augment and populate. Mark decides which nodes need to be split and/or collapsed, rebuild constructs the new topology, augment adds additional information to nodes and leaves. Finally, the leaf structures are populated.

The mark step (figure 7 (b)) is responsible to decide for whether a node needs to be toggled, i.e. split or collapsed. Only leaf nodes can be split and only inner nodes with only leaves as children can be collapsed. A leaf node is marked for splitting if the observed sample count goes above a configurable upper threshold (i.e. 2048). An inner node is marked for collapsing if the combined sample count over all leaves goes below a configurable lower threshold.

The rebuild step (figure 7 (c)) creates the adapted topology based on the results of the mark step. A breadth first construction scheme is used, so we only construct one level in parallel at a time. This would require many dispatches, so we leverage shared memory in compute shaders to construct entire subtrees per work group. This is the reason why we defer augmentation of the tree and population of leaves to later passes: These would need to be stored in shared memory as well, decreasing the effective size of the tree that can be constructed in one work group. Also, once the topology is set, these steps can be done in parallel over all nodes, instead of just in parallel over each level.

The augmentation step (figure 7 (d)) adds additional information to the nodes and leaves of the octree. The first dispatch augments nodes by adding parent pointers to each child in a secondary buffer. The second dispatch augments leaves. This includes neighbor information as well as corresponding nodes in the previous tree. Parent pointers are used to reconstruct the position of a leaf in the tree. This position is used to traverse the previous tree for correspondence as well as to traverse an offset position in the new tree for the six neighbors with a common face. If one side has a finer subdivision, then a potentially unbounded number of neighbors exist on that side. We currently ignore this case for simplicity.

Leaves are populated (figure 7 (e)) by copying the CDQ and pointers to the VLC from the previous tree (using correspondence information) and by allocating space for samples based on the observed sample count. The copy operation is needed because otherwise two trees would need to be traversed: The previous tree for sampling the VLC and CDQ and the current tree for placing samples. But these two trees are almost identical (they only possibly differ in some split or collapsed nodes). The copy operation is trivial for unchanged and split nodes, because they uniquely map to a single node in the previous tree. A collapsed node has eight potential leaves from which the VLC and CDQ may be copied from. We simply copy from the node which had the biggest observed sample count. Another possibility is to have VLC and CDQ construction be aware of collapsed nodes, but this adds further complexity.

6.3. Reprojection & Combined Traversal

Given a surface point, the octree is traversed top-down in order to access the containing leaf node and the corresponding estimator as well as pointers to the sample storage. If the surface point is dynamic, however, the leaf in which it is currently contained might be empty, i.e. the VLC is empty and the CDQ is in its initial configuration. This would result in visible artifacts. We thus have to traverse the octree twice: For sampling, we use the previous position. For placing samples, the current position is used. The previous position is typically readily available, as correct motion vectors for, e.g. TAA, need to know the exact location of surfaces in the previous frame. However, we do not want to pay the cost of two traversals (at least not every time). If these two positions are identical, or even almost identical, most of the traversal is redundant. We therefore perform a combined traversal until the paths diverge.

7. Evaluation

We use a modified version of Quake 2 RTX as our evaluation framework. A standard path tracer with NEE is used. We trace two indirect rays, but only do NEE for the primary and secondary hit. From the tertiary hit we only collect direct emission (e.g. for environment maps). In total, five rays per path are traced. After the primary hit, we perform two simplifications: First, all lights are treated as uniform emitters. As we focus on just the selection of lights and not sampling of lights themselves, we do not want the additional noise that comes from sampling textured emitters. Lights are sampled with plain area sampling. Second, all surfaces are assumed to be fully diffuse after the primary hit. Otherwise, our reference renders were left with fireflies even after many thousands of samples. Unless otherwise noted, the octree is configured with a lower threshold of 2048 and an upper threshold of 4096, the VLC with a bucket size of 32 and an exploration fraction of 10% and the CDQ with a bit count of 128 and a brdf fraction of 50%. All images are rendered with one sample per pixel.

We compare the VLC against two other techniques. The first, we refer to it as Static Light Lists (SLL), is the NEE implementation that shipped with Quake 2 RTX. It uses potentially visible sets (PVS) [Tel92] derived from the BSP structure [FvDFH90] to collect potentially visible lights for each BSP leaf. Based on the leaf in which a light is contained, the light is distributed among the leaves that are in the PVS of the leaf. BSP and PVS construction are inherently static due to expensive computation. As such, only static scene elements can affect visibility. However, dynamic moving lights are possible, as finding the containing leaf and corresponding PVS is very fast. Just as with the VLC, this sampling procedure is linear in the number of lights. To bound the computation time, the lights are traversed with a variable stride and uniformly random initial offset such that only a maximum number of lights are considered. We configured this maximum to be 32, in line with the VLC bucket size. We augmented this technique with our LTC-based importance measure, so that the techniques mainly differ in how precisely they capture the set of visible lights.

The other technique is the Light Hierarchy (LH) as proposed by [CK18] and [MC19] but without splitting. A Bounding Volume Hierarchy (BVH) is constructed over all lights with each leaf containing a subset of the lights. This hierarchy is then traversed stochastically by evaluating an importance measure per child node which is used as a probabilistic weight to descend this child. We compressed and widened the structure in the spirit of [YKL17] to improve traversal speed and sampling quality (flattening the hierarchy can be seen as brute-force splitting, albeit not considering the surface point). Construction of the hierarchy is done each frame using standard binary binning [Wal07] and a final top-down traversal to collapse nodes. Thus, dynamic lights are handled transparently. We configured the width to be 8 children per node and the number of lights in each leaf node to be 2.

We compare the CDQ against plain BRDF sampling. To the best of our knowledge, there currently exists no other guiding technique capable to execute at interactive rates.

7.1. Quality Comparison

We compare the image quality (quantified with MSE) of our presented techniques against the SLL, LH and plain BRDF sampling. For that we instantiate all combinations of NEE and indirect sampling techniques (6 in total). We use scenes from Quake 2 RTX as well as scenes with more complex geometry as shown in figure 8.

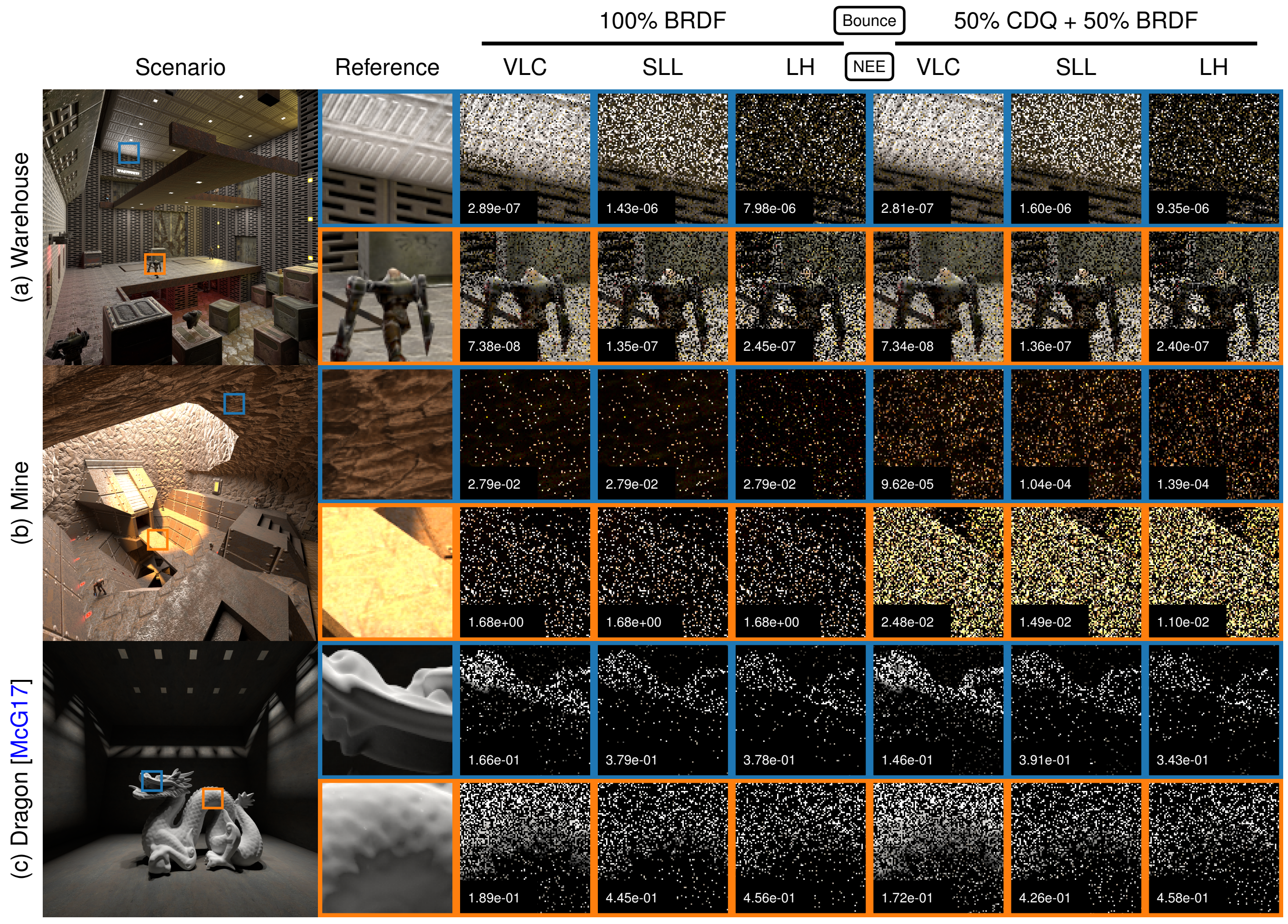

Figure 8: *Comparison of techniques with 1ssp frames in test scenes. Scenes (a) and (b) taken from Quake 2 RTX, © id Software, Inc. VLC consistently outperforms the other techniques in (a). Due to uniform illumination, CDQ provides no real benefit. CDQ is most useful with high contribution from a specific direction as with the sun in (b), improving both direct and indirect illumination. With very complex direct illumination as in (c) (area lights behind a cover), both techniques still improve sampling, but not by that much.*

Scene (a) shows a warehouse with relatively many light sources and mostly uniform illumination. The full scene is considerably larger, containing even more lights. By focusing only on relevant lights, the VLC excels here, achieving almost on order of magnitude lower error compared to the SLL and LH. The LH is worst in this instance. We assume that due to missing visibility information, lights outside of the room also receive a nonzero sampling weight. The CDQ does not provide any meaningful advantage here. But this is to be expected due to overall uniform illumination.

Scene (b) shows a mine which is open towards the sky. It is illuminated mostly by the environment map and the contained sun. Especially indirect illumination is affected by the spot which the sun directly illuminates. Our NEE implementation does not account for environment maps at all, rendering the VLC useless. With plain BRDF sampling, both directly and indirectly illuminated surfaces by the sun are severely undersampled. By refining towards the sun and the directly illuminated spot, the CDQ improves sampling by two to three orders of magnitude for these regions.

Scene (c) shows the Stanford dragon model illuminated by ten area lights behind a cover. This is a relatively complex situation as lighting is blocked by other geometry and the receiver is relatively complex. All techniques produce noisy results, but the VLC with the CDQ is still slightly better.

7.2. VLC & Octree Adaptation

In this section, we examine the temporal behavior of the VLC and octree. We are particularly interested in how long it takes for these techniques to fully adapt to a given situation from an uninitialized state (the worst case). We go over the frame sequence that is depicted in figure 9.

In the *initial state*, the octree is subdivided only once (because we cannot represent a single leaf node) and the VLC of each of the eight leaves is empty. As such, exploration sampling is used exclusively here. This leads to a very noisy image that lacks any detail of the scene. In the *second frame*, we directly see the VLC tak-

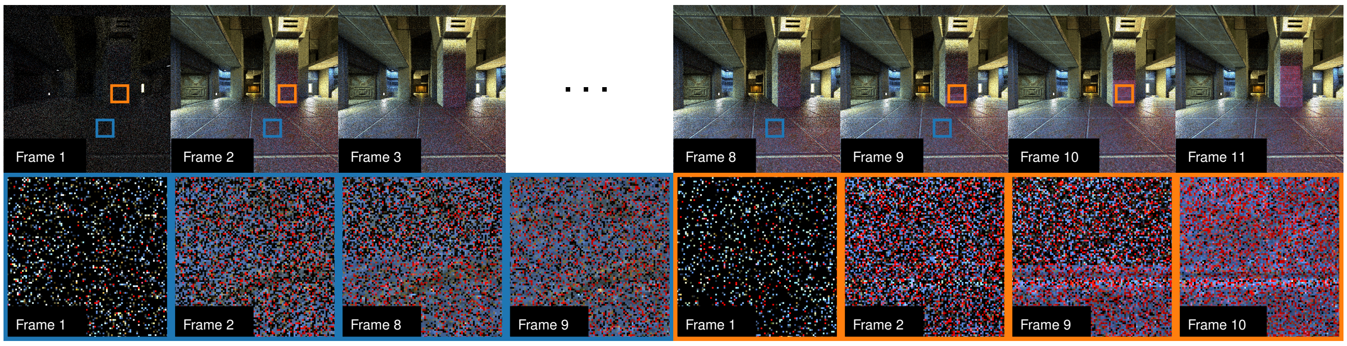

Figure 9: *VLC and octree adaptation over a frame sequence. Initially, only exploration sampling is used due to an empty VLC. Sampling is already improved after a single frame, but octree adaptation creates specialized VLCs with even better sampling in the subsequent frames.*

ing effect. If the octree were not to subdivide any further, the VLC would already be in a converged state. But this subdivision is still very rough with all the samples being deposited in a single leaf, leading to an imprecise VLC. This explains why there are still so many samples with zero contribution. There is no change in sampling until *frame eight*. The problem is that the octree has quite a large extent as it needs to cover the entire scene. The shown room is only a small part of it. As such, the octree needs multiple frames in order to subdivide towards this region. At frame eight, subdivision is refined enough so that we see first improvements at the floor, which continue with frame 9. The number of samples with zero contribution is now closer to the 10% exploration fraction. Up until frame 11, refinement continues with the pillar.

7.3. CDQ Adaptation

The main parameter of the CDQ is the number of bits with which the tree is encoded. We examine adaptation of the CDQ with different bit counts and rates of change. For this, we construct a scene which consists of a floor, a circular wall surrounding this floor and a rotatable light below the floor. The floor is illuminated indirectly by the light via the wall. The camera is pointed directly at the floor. We let the CDQ adapt to an initial position of the light. Then we perform a sudden, defined, rotational change of the light over one frame. We observe how long it takes for the CDQ to readapt to this new situation and how much worse the sampling quality initially is after the change compared to the readapted state.

In figure 10, we compare bit counts of 32, 64, 96 and 128 bits with rotational changes of 9 and 36 degrees. We also provide visualizations of the optimal sampling densities of the radiance fields (in the 2D octahedron map) and CDQ sampling densities adapted to the initial configuration. The plots show MSE of the CDQ over a sequence of frames. At frame zero, the change is performed. All variations exhibit fast convergence, but overall adaptation greatly depends on both parameters. At nine degrees, the change is not all that big. Neither the peak error nor time to readapt are very large for all bit counts. The 128 bit variant still takes longest at around 10 frames. Both aspects deteriorate at an increasing angle. At 36 degrees, the CDQ is initially worse than plain BRDF sampling. Since we still use the BRDF 50% of the time, it effectively bounds the error of the CDQ. This is crucial in this instance, as the previous adaptation no longer has any meaningful relation to the new situation. While the readapted CDQ still provides much improvement, the error is higher than in the initial configuration. We assume that the rigid subdivision scheme of the quadtree cannot adapt to the new configuration as well as to the initial configuration. Adaptation time has increased to 20 frames for the 128 bit variant.

Another observation is that adding more than 64 bits does not improve variance any significantly. We assume that this is mostly due to the radiance field being very smooth (along the wall). It is thus approximated relatively well already by the 64 bit instance. More bits can still be advantageous for, e.g. multi-modal irradiance, but we did not explicitly evaluate this aspect.

7.4. Performance

For our performance evaluation, we use an Nvidia RTX 2080 Ti at a resolution of 1920×1080. A walk around the Quake 2 RTX scene “base1” is used as the reference scenario. A video of this scenario is attached in the supplemental material (frames 100 to 300). We consider both the required preprocessing and impact on the path tracer for the presented techniques.

Figure 11 shows the averaged timings of all preprocessing over 200 runs. In total, around 1.2 ms are needed. The entirety of all octree operations (mark, adapt, augment and allocate) take around 0.2 ms. While this timing is relatively fast, the octree is not large to begin with. This probably means that the GPU is not saturated during this time. Deduplication of the VLC takes 0.2 ms, while neighbor and bucket merging take around 0.2 ms. The CDQ accumulation and adaption takes around 0.5 ms. Both CDQ accumulation and VLC deduplication iterate over their respective sample sets, but the latter takes longer due to more complex processing: For each sample, the given quadtree leaf needs to computed, while for the VLC only a few hash table accesses are needed.

The impact on the path tracer is shown in figure 12. We compare the impact against LH and SLL, taking around 11.2 ms and 15.1 ms, respectively. VLC is almost identical to SLL with 15.1 ms. Since both SLL and VLC are configured with the same bucket size of 32, this indicates that light selection is the limiting factor in this instance, and not octree traversal. Adding CDQ to VLC increases path tracing time to 17.1 ms.

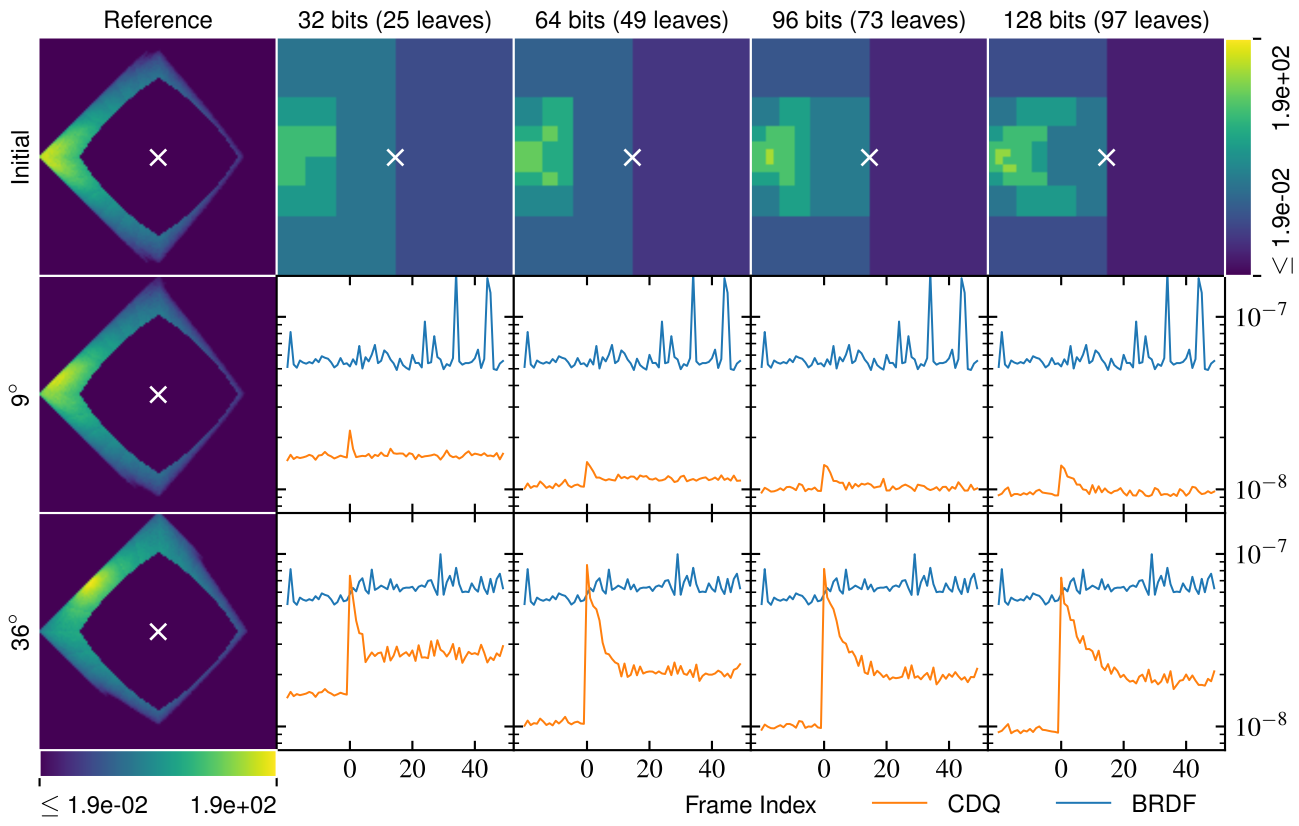

Figure 10: *CDQ static and dynamic adaptation through a rotatable light indirectly illuminating a surface via a circular wall. The left column visualizes the optimal sampling densities for the radiance field. The top row visualizes the CDQ sampling densities adapted to the initial configuration at different bit counts. The following rows show adaptation over frames quantified via MSE after a sudden rotation of the light source at different angles. A logarithmic viridis color mapping with the same range across all images is used.*

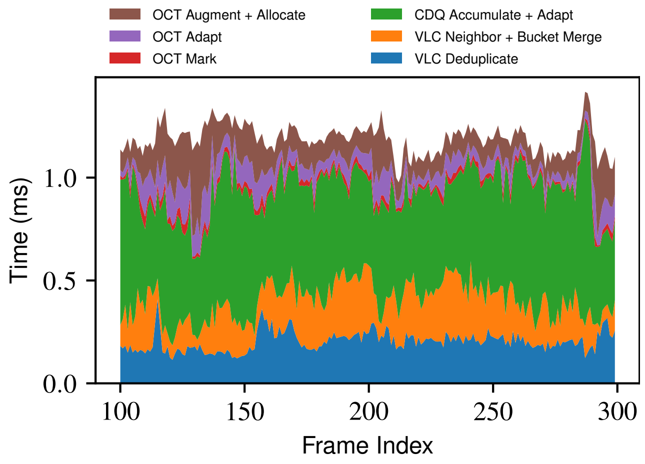

Figure 11: *Execution time of preprocessing operations over multiple frames (average of 200 runs). On average, around 1.2 ms are needed.*

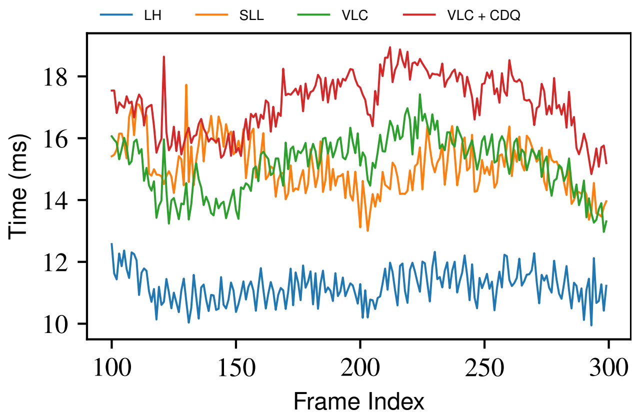

Figure 12: *Execution time of path tracer over multiple frames with different importance sampling strategies. On average, LH is fastest (11.2 ms) and VLC + CDQ is slowest (17.1 ms).*

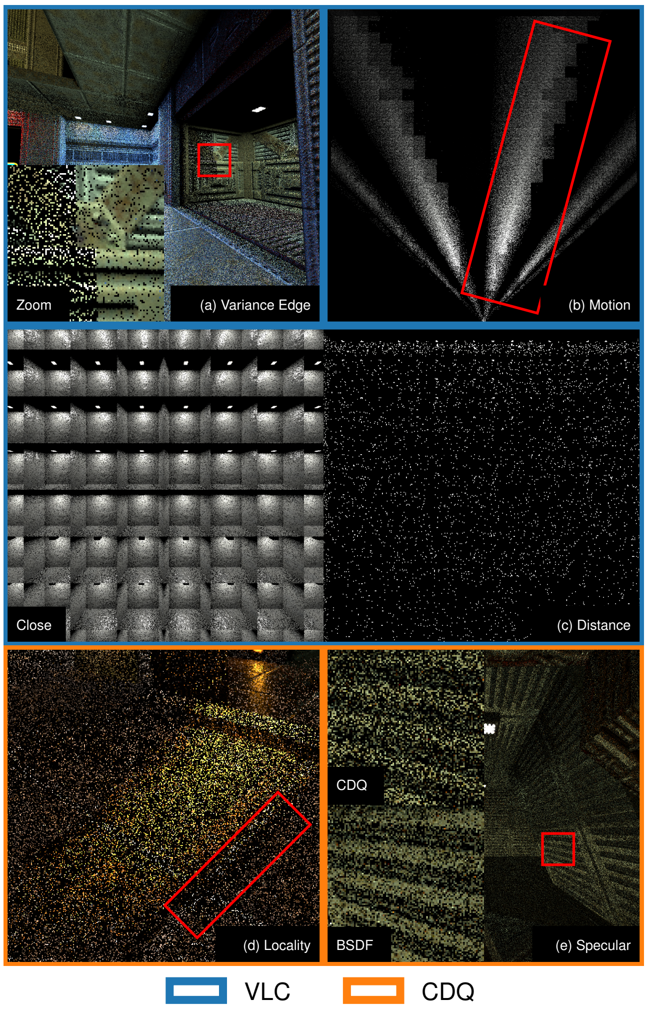

Figure 13: *Various artifacts encountered with the VLC and CDQ. (a) Variance edge due to varying number of lights in adjacent VLCs. (b) Fast moving shadows caused by lights traveling in front of a slit. (c) Heavily increased noise with larger distance from scene caused by rough octree subdivision (exaggerated by reducing VLC bucket size to 8 instead of 32). (d) Increased noise at the edge of a spot directly illuminated by the sun due to positionally dependent occlusion inside of octree leaf. (e) Increased noise when using CDQ on specular surfaces.*

8. Limitations & Future Work

While we have shown that both the VLC and CDQ are promising techniques for real-time path tracing, many technical and implementation problems remain that need to be addressed in the future.

The most fundamental problem is that there is an inherent delay of one frame when using samples from the previous frame. This can become problematic with very dynamic scenarios (e.g. strobing or fast-moving lights, see figure 13 (b)). A possible solution would be to use samples from the current frame as well, e.g., by running the path tracer in two phases (one for exploration and one for sampling).

Limitations associated to the temporal sample-adaptive octree are: (1) Oftentimes, the resolution is not enough to minimize variation within a leaf node (figure 13 (d)), e.g., due to positionally changing occlusion of lights and highlights. As resolution primarily depends on camera distance, this is especially problematic with distant geometry (figure 13 (c)). (2) We do not perform any interpolation between octree leaves. For the VLC, this is mostly not needed, as the point-specific knowledge introduced with the importance measure automatically smoothens the transition between regions. However, variations in contained lights and bucket count can be seen as variance edges (figure 13 (a)). This problem is even more pronounced with the CDQ, as it represents just a single CDF per leaf. (3) Large buffers are needed to store intermediate samples for VLC construction and CDQ adaptation. By directly deduplicating visible lights for the VLC and accumulating leaf contributions of the CDQ, memory consumption could be reduced significantly.

Limitations associated to the VLC are: (1) Scalability is limited, as it handles individual triangular area lights. This is an inherent disadvantage compared to, e.g., light hierarchies. While the bucket strategy always ensures a bounded computation time even with many lights, sampling quality deteriorates significantly. Selection on entire aggregates of lights (e.g. mesh lights) and/or a more informed distribution of lights among buckets might resolve this issue. (2) Lights with rare visible samples (e.g. due to partial occlusion) potentially need far more than a thousand samples per leaf to be stably part of the VLC. Persisting the VLC over multiple frames could be an option to increase the effective sample count. (3) The current global exploration strategy reverts to questionable heuristics to account for the missing relationship between surface point and light. Local exploration strategies per leaf, e.g. through well-known clustering techniques [OBA12], could resolve this issue. (4) Both the VLC and exploration strategy are sampled at a fixed fraction. Exploring approaches to dynamically determine them seems worthwhile. In particular, if no new lights can be discovered, solely the VLC could be sampled.

Limitations associated to the CDQ are: (1) It does not account for the BRDF, giving higher noise to specular surfaces at the 50% default BRDF/CDQ split (figure 13 (e)). (2) It is not stratified (one random number for sampling a leaf plus two for the area contained by the leaf), making blue noise sampling much less effective. A top-down traversal with sample-warping of two random numbers would be needed. (3) Adaptation speed is relatively slow because only a single leaf migration is performed per frame. Obviously, migrating multiple leafs at a time could drastically improve adaptation speed.

9. Conclusion

In this paper we have presented importance sampling techniques for next event estimation (VLC) and path guiding (CDQ) which construct improved sampling densities through sample reuse. With implicit consideration of visibility and high-quality importance sampling, the VLC is an important step towards full product importance sampling of direct illumination with many lights. By compressing a directional quadtree structure down to a few bits, the CDQ is a path guiding scheme that is tractable for real-time use. Both techniques are embedded in a temporally adapted octree structure situated in world-space which balances the available samples among leaves.

We have shown that both techniques provide significant variance reduction in various test scenes (section 7.1), while also adapting to entirely new situations in few frames (section 7.2 and 7.3). Performance for both techniques is within close reach for practical use (section 7.4). Still, we further wish to explore this field in order to free the techniques from the remaining artifacts (section 8).

Acknowledgements

We thank Christoph Peters for deriving the correction factor of the CDQ and insightful discussions on the presented techniques. Additionally, we thank the anonymous reviewers for their remarks which considerably helped to improve the paper.

References

[Arv95] ARVO, JAMES. "Applications of Irradiance Tensors to the Simulation of Non-Lambertian Phenomena". Proc. ACM SIGGRAPH. 1995, 335–342 3.

[BFK19] BINDER, NIKOLAUS et al. *Massively Parallel Path Space Filtering*. 2019. arXiv: 1902.05942 [cs.GR] 3.

[CK18] CONTY ESTEVEZ, ALEJANDRO et al. "Importance Sampling of Many Lights with Adaptive Tree Splitting". *ACM Comput. Graph. Interact. Tech.* 1.2 (2018) 3, 8.

[CKS*17] CHAITANYA, CHAKRAVARTY R. ALLA et al. "Interactive Reconstruction of Monte Carlo Image Sequences Using a Recurrent Denoising Autoencoder". *ACM Trans. Graph.* 36.4 (2017) 1.

[CNS*11] CRASSIN, CYRIL et al. "Interactive Indirect Illumination Using Voxel Cone Tracing: A Preview". Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. 2011, 207 1.

[DIP14] DUPUY, JONATHAN et al. "Quadtrees on the GPU". *GPU Pro 5*. 2014 3.

[DK17] DAHM, KEN et al. "Learning Light Transport the Reinforced Way". *Talks*. Proc. ACM SIGGRAPH. 2017 3.

[DWB*06] DONIKIAN, MICHAEL et al. "Accurate Direct Illumination Using Iterative Adaptive Sampling". *IEEE Trans. on Visualization and Comput. Graph.* 12.3 (2006), 353–364 3.

[ED08] ENGELHARDT, THOMAS et al. "Octahedron Environment Maps". *Vision Modeling and Visualization*. 2008 5.

[ESAW11] EISEMANN, ELMAR et al. *Real-Time Shadows*. 1st. 2011 1.

[FHH*19] FASCIONE, LUCA et al. "Path Tracing in Production". *Courses*. Proc. ACM SIGGRAPH. 2019 1, 3.

[FvDFH90] FOLEY, JAMES D. et al. *Computer Graphics: Principles and Practice*. 2nd. 1990 8.

[HDG99] HART, DAVID et al. "Direct Illumination with Lazy Visibility Evaluation". Proc. ACM SIGGRAPH. 1999, 147–154 3.

[HDHN16] HEITZ, ERIC et al. "Real-Time Polygonal-Light Shading with Linearly Transformed Cosines". *ACM Trans. Graph.* 35.4 (2016) 2–4.

[HEV*16] HERHOLZ, SEBASTIAN et al. "Product Importance Sampling for Light Transport Path Guiding". *Comput. Graph. Forum* 35.4 (2016), 67–77 3.

[HP02] HEY, HEINRICH et al. "Importance Sampling with Hemispherical Particle Footprints". Proc. Spring Conference on Comput. Graph. 2002, 107–114 3.

[Jac89] JACOBSON, G. "Space-Efficient Static Trees and Graphs". Proc. Symposium on Foundations of Comput. Science. 1989, 549–554 3, 5.

[Jen95] JENSEN, HENRIK WANN. "Importance Driven Path Tracing using the Photon Map". *Rendering Techniques*. 1995, 326–335 3.

[KD10] KAPLANYAN, ANTON et al. "Cascaded Light Propagation Volumes for Real-Time Indirect Illumination". Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. 2010, 99–107 1.

[KWR*17] KELLER, ALEXANDER et al. *The Iray Light Transport Simulation and Rendering System*. 2017. arXiv: 1705.01263 [cs.GR] 3.

[LXY19] LIU, YIFAN et al. "Adaptive BRDF-Oriented Multiple Importance Sampling of Many Lights". *Computer Graphics Forum* 38.4 (2019), 123–133 3.

[MC19] MOREAU, PIERRE et al. "Importance Sampling of Many Lights on the GPU". *Ray Tracing Gems*. 2019, 255–283 8.

[McG17] MCGUIRE, MORGAN. *Computer Graphics Archive*. https://casual-effects.com/data. 2017 9.

[MGN17] MÜLLER, THOMAS et al. "Practical Path Guiding for Efficient Light-Transport Simulation". *Comput. Graph. Forum* 36.4 (2017), 91–100 2, 3, 5.

[MMBJ17] MARA, MICHAEL et al. "An Efficient Denoising Algorithm for Global Illumination". Proc. of ACM SIGGRAPH / Eurographics conference on High Performance Graphics. 2017 1.

[OBA12] OLSSON, OLA et al. "Clustered Deferred and Forward Shading". Proc. of ACM SIGGRAPH / Eurographics conference on High Performance Graphics. 2012, 87–96 1, 12.

[OSK*14] OLSSON, OLA et al. "Efficient Virtual Shadow Maps for Many Lights". Proc. ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games. 2014, 87–96 1.

[RDGK12] RITSCHEL, TOBIAS et al. "The State of the Art in Interactive Global Illumination". *Comput. Graph. Forum* 31.1 (2012), 160–188 1.

[SKW*17] SCHIED, CHRISTOPH et al. "Spatiotemporal Variance-Guided Filtering: Real-Time Reconstruction for Path-Traced Global Illumination". Proc. of ACM SIGGRAPH / Eurographics conference on High Performance Graphics. 2017 1.

[SWZ96] SHIRLEY, PETER et al. "Monte Carlo Techniques for Direct Lighting Calculations". *ACM Trans. Graph.* 15.1 (1996), 1–36 3.

[Tel92] TELLER, SETH JARED. "Visibility Computations in Densely Occluded Polyhedral Environments". PhD thesis. University of California, Berkeley, 1992 8.

[VG95] VEACH, ERIC et al. "Optimally Combining Sampling Techniques for Monte Carlo Rendering". Proc. ACM SIGGRAPH. 1995, 419–428 3.

[VHH*19] VORBA, JIŘÍ et al. "Path Guiding in Production". *Courses*. Proc. ACM SIGGRAPH. 2019, 18:1–18:77 1, 3.

[VK16] VÉVODA, PETR et al. "Adaptive Direct Illumination Sampling". *Posters*. Proc. ACM SIGGRAPH ASIA. 2016, 43:1–43:2 3.

[VKK18] VÉVODA, PETR et al. "Bayesian Online Regression for Adaptive Direct Illumination Sampling". *ACM Trans. Graph.* 37.4 (2018), 125:1–125:12 3, 6.

[VKŠ*14] VORBA, JIŘÍ et al. "On-line Learning of Parametric Mixture Models for Light Transport Simulation". *ACM Trans. Graph.* 33.4 (2014) 3.

[Wal07] WALD, INGO. "On Fast Construction of SAH-Based Bounding Volume Hierarchies". Proc. IEEE Symposium on Interactive Ray Tracing. 2007, 33–40 8.

[Wal77] WALKER, ALASTAIR J. "An Efficient Method for Generating Discrete Random Variables with General Distributions". *ACM Trans. Math. Softw.* 3.3 (1977), 253–256 3, 5.

[War94] WARD, GREGORY J. "Adaptive Shadow Testing for Ray Tracing". *Photorealistic Rendering in Computer Graphics*. 1994, 11–20 3.

[WFA*05] WALTER, BRUCE et al. "Lightcuts: A Scalable Approach to Illumination". Proc. ACM SIGGRAPH. 2005, 1098–1107 3.

[YKL17] YLITIE, HENRI et al. "Efficient Incoherent Ray Traversal on GPUs through Compressed Wide BVHs". Proc. of ACM SIGGRAPH / Eurographics conference on High Performance Graphics. 2017 8.