Eurographics Symposium on Parallel Graphics and Visualization (2014)
M. Amor and M. Hadwiger (Editors)

Clustered Pre-convolved Radiance Caching

Hauke Rehfeld, Tobias Zirr and Carsten Dachsbacher

Karlsruhe Institute of Technology

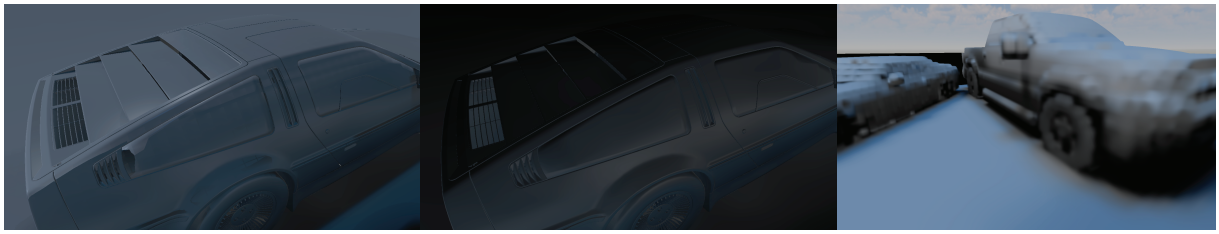

Figure 1: *A scene rendered with our method (left). We render indirect illumination (center) using pre-convolved radiance caching (RC); Our contribution is an efficient and robust computation of the RC based on a voxelization (right) and view-adaptive cache placement.*

Abstract

We present a scalable method for rendering indirect illumination in diffuse and glossy scenes. Our method builds on pre-convolved radiance caching (RC), which enables reusing the incident radiance computed at a surface point for its neighborhood. Our contributions include efficient and robust generation of these RCs based on a pre-filtered voxel representation that stores scene-geometry and surface illumination. In addition, we describe a distribution strategy that places the RCs according to screen-space clusters to ensure all pixels have valid radiance data when evaluating indirect illumination. The results demonstrate the scalability of our method and analyze the relation between render quality, surface glossiness and computation time, which depends on the number of caches and their resolution.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

1. Introduction

Photorealistic appearance of virtual objects in computer generated imagery has always been one of the challenging goals of computer graphics. This is not pure self-purpose, as can be seen from the many applications, ranging from video games and movie productions, to simulators, marketing and rapid-prototyping scenarios where photorealism plays an important role. However, in a number of cases, interactive performance is more important than absolute physical correctness or completeness, so visually plausible results have to be, and are, sufficient [YCK*09]. In this paper, we focus on plausibility, while providing efficient means to compute the reflection on the surfaces directly visible to the camera. The reflected light is gathered from a voxel representation of the scene using a variant of voxel cone tracing [CNS*11] and stored in radiance caches akin to pre-convolved radiance caching [SNRS12]. If the voxelization stores only direct illumination, the resulting image contains "one-bounce indirect illumination"; if the voxelization already stores a (coarse) global illumination signal then the result is akin to final gathering (a common strategy in interactive global illumination renderers [REG*09, RDGK12]).

Our contributions address the two challenges that remain open in the original pre-convolved radiance caching work [SNRS12]: efficient and robust computation as well as view- and scene-adaptive placement of the radiance caches.

Radiance caches (RCs) are typically low-resolution, depending on the glossiness of the surface. Using a hierar-

chical voxelization to collect incident radiance enables us to adaptively pre-filter the radiance signal which is crucial to avoid aliasing or costly supersampling. Throughout this paper we store direct illumination and assume diffuse surfaces in the voxelization (and thus simply speak of *direct* and *indirect* illumination); we refer to orthogonal work concerned with computing and storing voxel-based global illumination [MGP10, Rau13].

2. Previous Work

Computer graphics literature exhibits a huge body of previous work on (interactive) global illumination which would be impossible to cover in this section. Instead we refer the reader to a recent survey [RDGK12] and focus on the most related work here.

Many-Light Methods One class of methods that recently gained a lot of attention is the many-lights method [DKH*13] originating from Instant Radiosity [Kel97]. The core idea is to represent all lighting in a scene with an appropriate set of virtual point lights (VPLs). The challenge for interactive rendering is to quickly compute lighting from a sufficiently large set of VPLs. Reflective Shadow Maps (RSMs) [DS05] render the scene from the light to capture directly lit surfaces. From these surfaces, VPLs for one-bounce indirect illumination can be sampled when rendering the scene from the camera. The same authors introduce splatting to distribute the illumination from the RSM into the frame buffer [DS06]. Ritschel et al. [RGS09] use the G-buffer to approximate GI in screen-space (compared to light space in RSMs) by extending screen-space ambient occlusion. Kaplanyan and Dachsbacher [KD10] use a method inspired by the discrete ordinate method to approximate light transport, and use a RSM to initialize the simulation with direct illumination. Multi-resolution Splatting [NW09, NW10, NSW09] splats illumination from point lights accelerated using a min-max depth-map into a multi-resolution framebuffer to reduce fill-rate requirements. Tiled Deferred Shading [OA11] is even more efficient. This method creates rectangular tiles over a G-buffer and culls lights per tile to save on lighting calculations. The authors extended their work to Clustered Deferred Shading [OBA12], where they cluster the pixels of each tile separately by world-space position and normal.

Shading can also be sped up by interleaved sampling [KH01]. In the context of VPLs, interleaved sampling can be used to evaluate only a subset of VPLs per pixel [SIMP06, RGK*08]. Imperfect Shadow Maps [RGK*08, REH*11] can be used to solve the visibility problem for VPL lighting by computing shadow maps from a point-based representation of the scene. Also using point primitives, micro-rendering [REG*09] rasterizes a lit point-hierarchy for each pixel of the G-buffer for one-bounce global illumination, while Holländer et al. [HREB11] substantially optimize the traversal of such a hierarchy for GPUs.

Voxel-based Methods Thiedemann et al. [THGM11] use an atlas to voxelize the scene and then use hierarchical ray/voxel tests to compute global illumination. GigaVoxels [Cra11, CNS*11] introduce an elaborate scheme to voxelize the scene into a pre-filtered, sparse voxel octree in real-time. Radiance from the leaf nodes of the tree is propagated upwards and accumulated in the inner nodes. They then use this octree to trace cones that collect incident radiance. While cone tracing sacrifices some accuracy for an approximate result, it is very fast, as with growing cone diameters it can access higher levels of the hierarchy and collect the pre-filtered radiance without missing high-frequency details of the scene. Rauwendaal [Rau13] uses Voxel Cone Tracing combined with a hierarchical scene voxelization to compute global illumination, and examine how isotropic and anisotropic functions can be stored in the hierarchy using low order spherical harmonics.

(Ir)radiance Caching Instead of computing the irradiance per pixel, Irradiance Caching [WRC88] interpolates between the results of nearby pixels. As this does not allow to compute glossy reflections, Radiance Caching [KGPB05] uses spherical harmonics to define a directional function that stores incident radiance. Pre-convolved radiance caching (PCRC) [SNRS12] proposes to store incident radiance in a texture with one texel per direction and pre-convolve it—in essence creating a mip-map-pyramid of the radiance texture with accumulated radiance in the higher levels. Additionally, a low-resolution mip-map level is cosine folded to store irradiance. Using these two textures, evaluating the RCs requires only two texture lookups instead of re-evaluating the reflection functions. Irradiance can also be gathered into screen-space caches from a photon map [WWZ*09] to integrate caustics. To enable low to moderately glossy BRDFs, radiance can be represented with spherical harmonics instead. The caches are placed in screen-space using k-means, where the initial caches are distributed according to a modified version of Ward's illumination change term over the pixels of each node of a quad-tree over the depth-buffer. While this leads to a good distribution of caches, it is quite expensive to calculate even on the GPU.

3. Clustered Pre-convolved Radiance Caching

When using PCRC for indirect illumination, algorithms to *create, distribute* and *evaluate* the RCs are required. As we focus on interactive applications, we provide efficient, fully dynamic and fully parallel variants.

Creation To allow for efficient parallel creation of the pre-convolved RCs, we employ a hierarchical scene voxelization. It is relit and its levels are rebuilt every frame, using an approach similar to mip-mapping to create individual

levels of the hierarchy. The hierarchy enables us to use cone-marching [CNS*11] to fill the pixels of the RCs.

Distribution We employ a clustering scheme based on [OBA12] to distribute RCs in screen-space using a deferred shading G-buffer. Our approach adapts to scene geometry to ensure that valid data is available when evaluating the RCs, while keeping the number of clusters (and therefore caches) low and predictable.

Evaluation As we found the splatting approach advocated in PCRC to be relatively slow, we tailored our cache distribution to allow for a *gathering* approach when collecting the incident radiance from RCs.

3.1. Cone-marching a Pre-filtered Hierarchical Voxelization for efficient Cache Creation

To fill the RCs, we need to collect the incident radiance around the cache hemispherically into the base layer of the RC texture. While RCs are relatively low-resolution (16^2–32^2) and can use approximated results [YCK*09], they must not omit bright parts of the scene. Using traditional rendering algorithms like rasterization or raytracing would require taking several samples per pixel of the RC texture to avoid this.

Cone-marching in a pre-filtered voxel hierarchy starts traversal in a high-resolution voxel level and uses lower resolution levels when the cone diameter gets larger, see Figure 2. It allows for fast, approximate traversal, and the pre-filtered low-resolution levels of the hierarchy ensure important parts of the scene are represented in the result even when using larger cones.

Because this meets our requirements so well, we simply cone-march one cone per pixel of the RC to collect the incident radiance. While the result is approximate, this gives us a sampling of the whole hemisphere of the RC without missing important details of the scene.

The voxel-hierarchy can store direct or indirect illumination, each layer holding pre-filtered illumination from the layer below. Thus our pre-convolved RCs generated with this voxel-hierarchy allow for one-bounce indirect illumination in the case of direct light, or multi-bounce indirect illumination if the hierarchy stores global illumination.

Voxelization As the voxelization algorithm is strictly orthogonal to our algorithm, we rely on a pre-process that statically voxelizes input meshes. It samples all surfaces uniformly, writes each sample to its cell in the voxelization and averages the surface normals of all samples in the cell.

The resulting voxelization saves only one normal and one albedo value per cell, and a list of all non-empty voxels. It could be extended to save information dependent on the direction or BRDF of the surface, see [Rau13]. Note that while we currently support only rigid transformations, we could easily voxelize every frame [CG12] [CLY*14]—all other steps are already fully dynamic and not scene-dependent.

Voxel Illumination After normals and albedo are collected, we compute direct illumination for each non-empty voxel using its average surface normal. Shadow cones are cast to each light source and the opacity of voxels hit by the cone is accumulated to attenuate the light's contribution.

Pre-filtered Hierarchy To allow for fast accumulation of indirect illumination using cone-marching, we need a pre-filtered hierarchy of down-sampled versions of the generated voxel shading information. We base our approach upon mip-mapping and average the voxel level contents in cubes of eight voxels each, where a voxel accumulates the illumination of all voxels that occupy the same space in the parent level. However surfaces may intersect only half of the voxels in a 2^3 cube; when naively averaging (*volume coverage*), the resulting voxel will have an alpha of 1/2 despite containing a full blocker. To alleviate this problem we compute the alpha of lower-resolution voxels as $1/4$ of the sum of their source voxels' alpha. This *surface coverage* keeps surfaces opaque in lower-resolution versions. We then pre-multiply the albedo with the clamped alpha.

Multiple Voxel Hierarchies While we currently have not implemented real-time voxelization, we support rigid transformation of meshes with arbitrary transformation matrices. Several mesh-voxelizations can be instanced multiple times with different transformations that may be updated at run time. For each instance, we allocate a separate voxel hierarchy, as illumination will differ depending on instance orientation and surrounding shadow casters.

While the order of rendering steps stays the same with multiple meshes, we need to account for intersections in multiple hierarchies when cone-marching to retain the smooth results obtained with pre-filtered voxel cone-marching. We repeat casting of the same ray for all hierarchies in the scene, and when we find a second (or *n*-th) hit, we order the current and the new hit according to their depth and blend between their results in the correct back-to-front

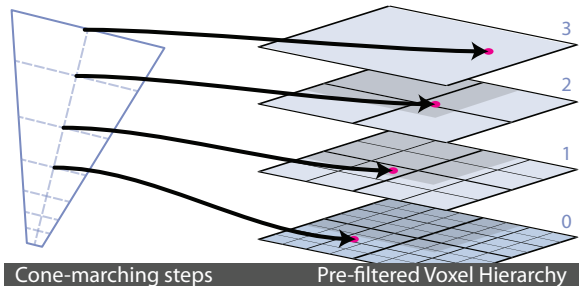

Figure 2: *Cone-marching starts in a high-resolution level (0) of the voxel hierarchy and accesses coarser levels (1-3) with growing cone diameters.*

Figure 3: *RCs rendered with cone-marching at* 64^2.

order. This yields correct results for shadowing, as shadows from different hierarchies can be multiplied commutatively, and semi-correct results for radiance accumulation, depending on the order in which the hierarchies are traversed and what hits are found. Naturally this approach cannot blend all cone-marching hits that accumulate along individual rays, but in practice this is not a problem.

As an example, consider two hits in different hierarchies: We can correctly blend the respective radiances by reordering the hits according to the last stored hit depth and the new computed hit depth. If there is a third hit in yet another hierarchy, blending might introduce a small error: we save hit information only for the closest hit, so we cannot blend correctly if new hits lie between the previous hits. Instead these hits are simply blended behind. It would be possible to store all hits and then use correct blending between them at low additional cost, but as we collect only indirect illumination, the error is typically neglectable.

3.2. Clustered Cache Distribution

The RCs need to be distributed so that they collect incident radiance that can be shared by as many G-buffer pixels in the vicinity of the cache as possible, while taking special care to ensure all pixels find valid data in at least one of the caches.

RCs have a world space influence radius, so their influence over the screen diminishes with depth while their number increases. However we do not want to spend too much computation time on parts of the screen that are far away [OBA12]. Rather, we want to keep the number of RCs relatively constant over the screen. Also, their normal is quite important for optimal results, as they only save incident radiance for the upper hemisphere.

Clustering We aim to create clusters of G-buffer pixels that can share the same RC, so we can place exactly one RC into each cluster. The clusters created by Clustered Deferred Shading are a good starting point, as they subdivide space adaptively inside the boundaries of screen-space tiles, so clusters are small/local and they are roughly evenly distributed in screen-space.

Clustered Deferred Shading also clusters by normal, subdividing each main direction (cube face) 3×3 times. This potentially creates many clusters per tile. Because our clusters decide where RCs are placed, we need to ensure that all relevant normal directions are represented with as few clusters as possible.

Adapting to Normals Our adaptive clustering approach adapts to the actual normals and reduces the number of clusters, especially where surfaces smoothly bend over cluster borders. This is important, as filling RCs is expensive, and smooth surfaces are most efficiently represented by a single RC.

Before pixels are assigned to clusters, we build a small 2D histogram of surface normals for each tile. Then we apply a Gaussian kernel to the histogram to average the normal directions and get a good estimate for the directions that are most important to represent with RCs. Using the convoluted histogram, we extract a three-axis frame per tile, starting with the highest-rated direction.

After we select an axis, we decrease the rating of similar (co-linear) directions:

$$w_n = (w_n + 1)(1 - |\vec{n} \cdot \vec{n}_{\text{best}}|^8)$$

where $\vec{n}$ is a normal direction in the normal histogram, w_n is the current weight assigned to this normal and $\vec{n}_{\text{best}}$ is the normal that was last selected as the best normal.

We then continue until we have selected three axes. These represent the most important normal directions in the tile and are roughly orthogonal if the normals in the tile require all axes to be represented. We then create clusters based on each pixel's alignment with these axes by encoding it into the global cluster sort key.

After this we sort pixels by cluster key, and extract resulting clusters using neighbor comparisons, parallel prefix sum scans and reductions, just as in Clustered Deferred Shading.

Placing Caches After we have defined the clusters, we need to choose a representative pixel in each cluster to place a cache on. It is important that the cache:

- aligns well with the average normal in the cluster,
- minimizes the distance to the cluster's surface points,
- lies as far out from the actual surfaces as possible, so raycasting through the voxel hierarchy can use small biases and surface offsets.

Therefore we place the cache on a pixel that maximizes:

$$\frac{\vec{n}_p \cdot \vec{n}_c + (\vec{p} - \vec{c}) \cdot \vec{n}_c}{1 + d_{\vec{c}}}$$

where $\vec{p}$ is the pixel's position, $\vec{n}_p$ the pixel's normal, $\vec{c}$ the center of the cluster's bounding box, $\vec{n}_c$ the average normal in the cluster's pixels, $d_{\vec{c}}$ the distance of $\vec{p}$ to the line extending from $\vec{c}$ in direction of $\vec{n}_c$. Here, $\vec{n}_p \cdot \vec{n}_c$ ensures aligment of the surface normal, $(\vec{p} - \vec{c}) \cdot \vec{n}_c$ tries to choose the pixel that extends most from the surface and dividing by $d_{\vec{c}}$ gives more weight to RCs that represent the center of the cluster.

Figure 4: *Performance breakdown of our method at 2M pixels with 10 primary lights. Dark stages are orthogonal to our work.*

3.3. Evaluating Radiance Caches with Gathering

Evaluating a RC for a pixel of the G-buffer requires the following steps: For diffuse illumination we use the surface normal of the pixel to retrieve the value of the corresponding cosine-weighted integral in the pre-generated diffuse lookup table [SNRS12]. For glossy illumination, we use the reflection vector and choose a mip-level in the RC's pre-convolved mip-map that corresponds to the surface's glossiness. For this, we calculate the solid angle of the Phong-lobe (at 1%) and interpolate between corresponding mip-levels.

Typically, there are multiple RCs influencing each pixel. To prefer more accurate RCs over those that are farther away their contribution has to be weighted. PCRC uses a weight based on spatial distance and a maximum angle.

PCRC also suggests to use splatting to evaluate the RCs. To splat a RC, its conservative sphere of influence is projected onto the screen. Then, for every G-buffer pixel inside the sphere's radius, the RC is evaluated, the result accumulated and its weight calculated. After all RCs have been splatted into the radiance buffer, the contribution of a pixel's splats has to be normalized by the sum of weights of the corresponding RCs. While the splatting approach works, it induces large amounts of overdraw, is cumbersome to implement in CUDA if maximum performance is required (no idle threads, etc.) and requires an extra normalization pass.

Since our clustered cache distribution is tile-based, we can rely on a maximum screen-space influence of the RCs, which makes it feasible to switch to a *gathering* approach. We collect the contribution for a single pixel by looping over all RCs in the surrounding tiles that lie inside a specified screen-space cache radius—typically two tiles.

Radiance Cache Weights If splatting is used to evaluate the RCs, a simple world-space weight can be used to attenuate their contribution [SNRS12]:

$$w_w = \left(1 - \left(\frac{\vec{p}_w - \vec{c}_w}{r_w}\right)^2\right)^+$$

where $\vec{p}_w$ is the pixel's position, $\vec{c}_w$ is the cache's position and r_w is the radius of the cache, all of them in world-space. However, care has to be taken to ensure that each splat is sufficiently large to cover all pixels inside the cache's world-space radius.

As large splats severely impact performance and gathering is most efficient when using a fixed maximum screen-space radius, we introduce a second, screen-space weight:

$$w_s = \left(1 - \left(\frac{\vec{p}_s - \vec{c}_s}{r_s}\right)^2\right)^+$$

where $\vec{p}_s$ is the pixel's position, $\vec{c}_s$ is the cache's position and r_s is the gathering radius, all of them in screen-space. We then calculate the weight of a cache as the minimum of the two, which ensures that there are never hard edges where the screen-space radius is not large enough.

4. Implementation Details

We work from a deferred shading G-buffer; all subsequent steps of our algorithm are implemented in CUDA and run fully in parallel on the GPU. Furthermore, all steps are easy to parallelize over multiple GPUs.

Voxel Hierarchy We use relatively low voxel resolutions (~128^3). Increasing the resolution does not improve the quality of our results noticeably, depending upon the glossiness of the scene, but requires more voxels to be illuminated, resulting in decreased performance.

Cone-marching When cone-marching, we march along the direction of the cone, choosing step size and mip-level based on the distance to the apex of the cone and a given cone opening angle.

Sampling a prefiltered volume texture with quadrilinear filtering yields smooth "anti-aliased" edges (quadrilinear = trilinear filtering in two volume texture mip levels + linear blending between mip levels). For each step the mip level is computed using the current cone opening diameter as target voxel edge length. The step length is chosen proportional to the current cone opening diameter. We achieved best results using half the current cone opening diameter as next step length. This ensures that we do not fully miss extrema that cancel out in-between voxels due to linear interpolation. Using larger steps, e.g. one edge length respective diameter, sampling in-between alternating opaque and transparent voxels with linear filtering could always yield the average opacity. In contrast, using half the step size, the oscillation is clearly visible in the samples.

We always start cone-marching with an opening diameter of 3/4 of the cell edge length. This is a compromise between starting with pure ray marching—until a cone diameter of one cell edge length is reached—and *immediately* starting to use higher mip-levels. For small opening angles, starting with ray marching would destroy the speed-up achieved with cone-marching without inreasing precision substantially. For large opening angles, the resulting cone apex offset is small anyway. To avoid self-occlusion of rays spawned near surfaces, we also take half a cone step along the ray direction away from the ray origin before taking the first sample.

As the opening radius and thus the cone steps grow linearly with distance to the cone's apex, the next cone step

Figure 5: *RC resolution and number compared, with a Phong-exponent of 150. If not variable, 4096 RCs at a resolution of* 16^2 *are used. With low numbers of RCs, details in the reflection are lost, while lowering their resolution makes reflections blocky.*

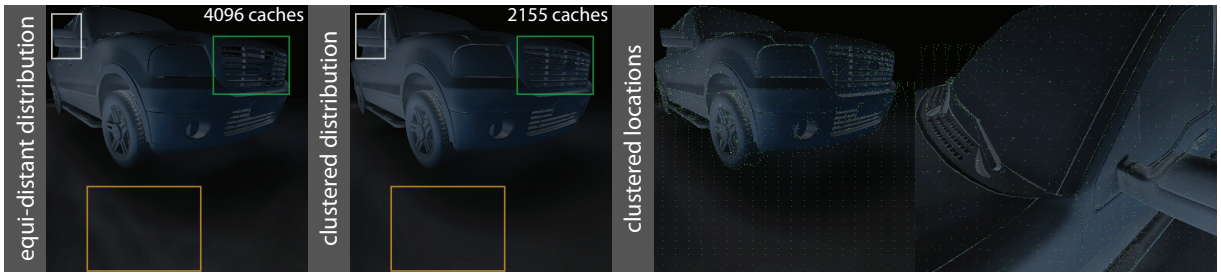

Figure 6: ***Left:*** *Comparison between clustered and equi-distant RC distribution.* ***Right:*** *RC locations when using our clustered distribution. Each green dot corresponds to the location of one RC.*

can be computed from the previous cone step just by adding $\tan(\alpha)\cdot\Delta$, where α is the cone opening angle and Δ is the last cone step. This corresponds to exponential growth, which nicely eliminates the logarithms required to compute the mip level in each step: Applying logarithm rules, we can incrementally update the mip-level using one addition per step.

5. Results

Unless otherwise mentioned, all results show only indirect illumination. The voxel hierarchy is illuminated by 10 directional lights that are sampled from a hemispherical environment map. If direct light is enabled, shadows are cast with cone-marching to all lights. RC resolution is 16×16, and 4096 RCs are distributed over the image plane. The cars scene has about 1.9M triangles. All surfaces use a Phong-BRDF with a constant Phong exponent of 50. Timings were measured on a NVIDIA GTX 670, with a resolution of 2M pixels (1742×1145).

Figure 1 shows our method with full direct and indirect lighting. Our method implements indirect illumination on complex, high-frequency geometry with only minor artifacts at interactive framerates. See Figure 4 for a performance breakdown. Note that direct illumination is orthogonal to our contribution; artifacts are visible in the direct illumination shadows because we simply use the same voxel hierarchy to trace shadow rays.

Figure 3 shows a cropped RC-texture. Note how the saved radiance is smooth, as our pre-filtered voxel-hierarchy accumulates the radiance in higher levels.

RC resolution is an important factor, but not as important as the number of RCs. Figure 5 shows how detail in the reflection in front of the cars is lost when using a low number of RCs with a Phong-exponent of 150. Lowering the resolution of the RCs does not influence results severely, but makes for blockier reflections.

Figure 6 compares our clustered RC distribution to a screen-space uniform, equi-distant distribution of RCs. The equi-distant distribution leaves some fragments without valid RCs, especially on high-frequency detail. As our clustered distribution adapts to the surface geometry, it does not show this problem and handles even the grill (green boxes) or fine details on the body (white boxes) of the car without perceivable artifacts. While our clustering preserves a roughly uniform distribution—but adapts to depth changes—if surfaces are flat, it strongly improves upon the results of the equi-distant distribution, which exhibits a dis-

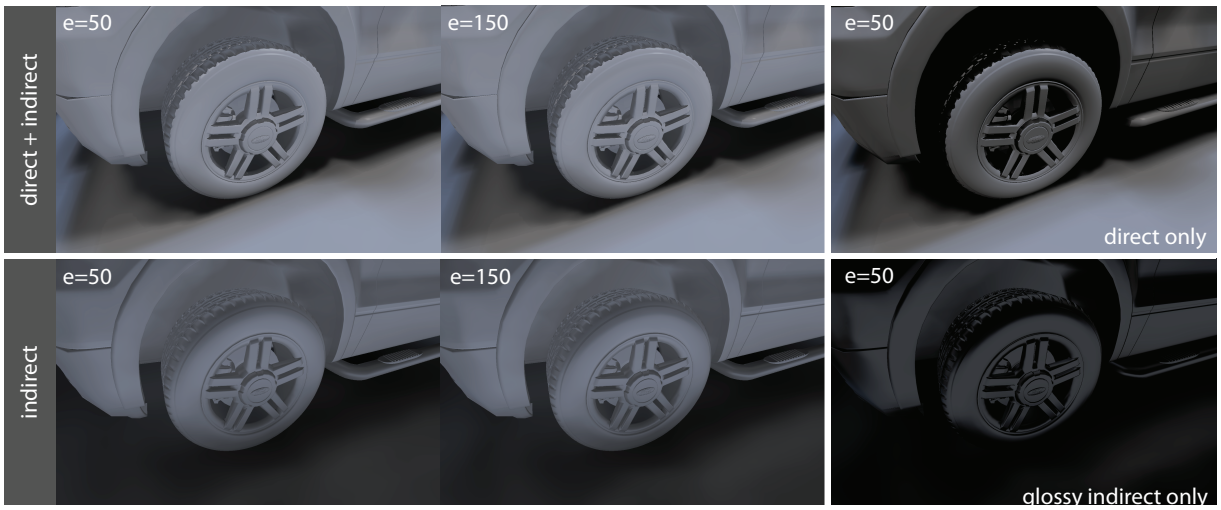

Figure 7: *Indirect illumination with different Phong-exponents e. Even with high Phong-exponents our method provides visually plausible results.*

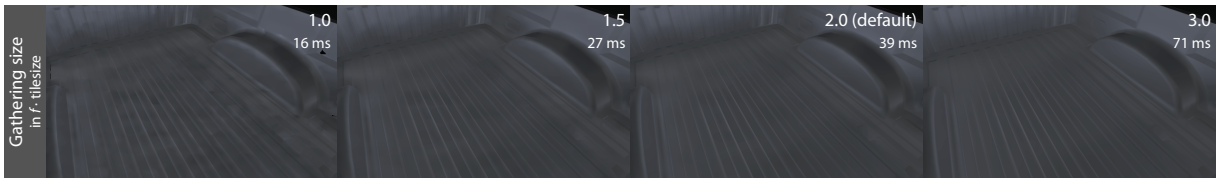

Figure 8: *Gathering size. A value of f will gather in a radius of $f \cdot tilesize$ around the current pixel. Larger values gather farther, but require to evaluate more RCs. Also given is the time required to gather and evaluate all RCs.*

tinct muddiness (orange boxes). On the right side we show another example how our clustered distribution adapts well to high-frequency and bend geometry.

Figure 7 shows how our method behaves with different Phong-exponents. Our algorithm copes quite well rendering glossy materials with a Phong-exponent of 150.

The gathering-radius is a parameter to the evaluation of the RCs, visualized in Figure 8. Our default value is a trade-off between muddiness and performance. If even higher-quality results are desired, the time spent evaluating RCs increases.

Temporal stability of RC locations is not fully solved in our implementation. When moving the camera, RCs follow surfaces, but jump if a surface leaves the cluster (tile) completely. See Section 6 for ideas how to alleviate this problem.

6. Conclusions and Future Work

We present an interactive, fully dynamic and fully parallel approach to global illumination which is suitable for applications where plausible approximations of global illumination are sufficient. It performs quite well in glossy scenes, even when dealing with small details and high-frequency geometry. Flat or smoothly bend surfaces are also handled with high quality.

We plan to improve on the temporal coherence by reusing RC positions in the next frame akin to the ideas of incremental instant radiosity [LLK07] where VPLs are reused, and Wang et al. [WWZ*09], where caches from the last frame are classified into the current clusters. For this purpose, RC positions could be stored in a BVH, and then checked if a RC from the last frame is still valid when distributing them. The BVH could also be used to find relevant RCs during evaluation.

The quality of the direct light is not perfect, as voxelization artifacts are easily spotted in direct shadows. Using shadow maps or fast GPU-raytracing instead of cone-marching could improve the quality of the direct illumination tremendously, but is not the focus of this work. Olsson et al. [OSK*14] discuss the fast generation of shadow maps.

As our voxelization is not sparse, currently scene-size and voxelization detail is limited. While switching to sparse voxel octrees would sacrifice cone-marching performance, it would easily allow for much larger scenes [CNS*11].

References

[CG12] CRASSIN C., GREEN S.: Octree-based sparse voxelization using the gpu hardware rasterizer. In *OpenGL Insights*, Cozzi P., Riccio C., (Eds.). CRC Press, July 2012, pp. 303–319. 3

[CLY*14] CHANG H.-H., LAI Y.-C., YAO C.-Y., HUA K.-L., NIU Y., LIU F.: Geometry-shader-based real-time voxelization and applications. *The Visual Computer 30*, 3 (2014), 327–340. 3

[CNS*11] CRASSIN C., NEYRET F., SAINZ M., GREEN S., EISEMANN E.: Interactive indirect illumination using voxel cone tracing. *Computer Graphics Forum (Proc. of Pacific Graphics) 30*, 7 (2011). 1, 2, 3, 7

[Cra11] CRASSIN C.: *GigaVoxels: A Voxel-Based Rendering Pipeline For Efficient Exploration Of Large And Detailed Scenes*. PhD thesis, Université de Grenoble, 2011. 2

[DKH*13] DACHSBACHER C., KRIVÁNEK J., HASAN M., ARBREE A., WALTER B., NOVÁK J.: Scalable realistic rendering with many-light methods. *Computer Graphics Forum* (2013). 2

[DS05] DACHSBACHER C., STAMMINGER M.: Reflective shadow maps. In *Proc. of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2005), pp. 203–208. 2

[DS06] DACHSBACHER C., STAMMINGER M.: Splatting indirect illumination. In *Proc. of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2006), pp. 93–100. 2

[HREB11] HOLLÄNDER M., RITSCHEL T., EISEMANN E., BOUBEKEUR T.: Manylods: Parallel many-view level-of-detail selection for real-time global illumination. *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering) 30*, 4 (2011). 2

[KD10] KAPLANYAN A., DACHSBACHER C.: Cascaded light propagation volumes for real-time indirect illumination. In *Proc. of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2010), pp. 99–109. 2

[Kel97] KELLER A.: Instant radiosity. In *SIGGRAPH '97* (1997), pp. 49–56. 2

[KGPB05] KRIVÁNEK J., GAUTRON P., PATTANAIK S., BOUATOUCH K.: Radiance caching for efficient global illumination computation. *IEEE transactions on visualization and computer graphics 11*, 5 (2005), 550–61. 2

[KH01] KELLER A., HEIDRICH W.: Interleaved sampling. In *Proc. of Eurographics Workshop on Rendering* (2001), pp. 269–276. 2

[LLK07] LAINE S., LEHTINEN J., KONTKANEN J.: Incremental instant radiosity for real-time indirect illumination. In *Proc. of Eurographics Symposium on Rendering* (2007), pp. 4–8. 7

[MGP10] MAVRIDIS P., GAITATZES A., PAPAIOANNOU G.: Volume-based Diffuse Glonal Illumination. *Proc. of Computer Graphics, Visualization, Computer Vision and Image Processing* (2010). 2

[NSW09] NICHOLS G., SHOPF J., WYMAN C.: Hierarchical image-space radiosity for interactive global illumination. *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering) 28*, 4 (2009), 1141–1149. 2

[NW09] NICHOLS G., WYMAN C.: Multiresolution splatting for indirect illumination. In *Proc. of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2009), pp. 83–90. 2

[NW10] NICHOLS G., WYMAN C.: Interactive indirect illumination using adaptive multiresolution splatting. *IEEE Transactions on Visualization and Computer Graphics 16*, 5 (2010), 729–741. 2

[OA11] OLSSON O., ASSARSSON U.: Tiled shading. *Journal of Graphics Tools 15*, 4 (2011), 235–251. 2

[OBA12] OLSSON O., BILLETER M., ASSARSSON U.: Clustered deferred and forward shading. In *Proc. of High Performance Graphics* (2012), pp. 87–96. 2, 3, 4

[OSK*14] OLSSON O., SINTORN E., KÃĎMPE V., BILLETER M., ASSARSSON U.: Efficient virtual shadow maps for many lights. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2014), ACM. 7

[Rau13] RAUWENDAAL R.: *Voxel based indirect illumination using Spherical Harmonics*. PhD thesis, Oregon State University, 2013. 2, 3

[RDGK12] RITSCHEL T., DACHSBACHER C., GROSCH T., KAUTZ J.: The state of the art in interactive global illumination. *Computer Graphics Forum 31*, 1 (2012), 160–188. 1, 2

[REG*09] RITSCHEL T., ENGELHARDT T., GROSCH T., SEIDEL H.-P., KAUTZ J., DACHSBACHER C.: Micro-rendering for scalable, parallel final gathering. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia) 28*, 5 (2009), 132:1–132:8. 1, 2

[REH*11] RITSCHEL T., EISEMANN E., HA I., KIM J., SEIDEL H.-P.: Making imperfect shadow maps view-adaptive: High-quality global illumination in large dynamic scenes. *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering) 30*, 3 (2011). 2

[RGK*08] RITSCHEL T., GROSCH T., KIM M. H., SEIDEL H. P., DACHSBACHER C., KAUTZ J.: Imperfect shadow maps for efficient computation of indirect illumination. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia) 27*, 5 (2008), 1. 2

[RGS09] RITSCHEL T., GROSCH T., SEIDEL H.-P.: Approximating dynamic global illumination in image space. In *Proc. of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2009), pp. 75–82. 2

[SIMP06] SEGOVIA B., IEHL J.-C., MITANCHEY R., PÉROCHE B.: Non-interleaved deferred shading of interleaved sample patterns. In *Graphics Hardware* (2006). 2

[SNRS12] SCHERZER D., NGUYEN C. H., RITSCHEL T., SEIDEL H.-P.: Pre-convolved Radiance Caching. *Computer Graphics Forum (Proc. of Eurographics Symposium on Rendering) 4*, 31 (2012). 1, 2, 5

[THGM11] THIEDEMANN S., HENRICH N., GROSCH T., MÜLLER S.: Voxel-based global illumination. In *Proc. of ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (2011), pp. 103–110. 2

[UNRD13] ULBRICH J., NOVAK J., REHFELD H., DACHSBACHER C.: Progressive visibility caching for fast indirect illumination. In *Proceedings of International Workshop on Vision, Modeling, and Visualization* (2013).

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. In *SIGGRAPH* (1988), Beach R. J., (Ed.), ACM, pp. 85–92. 2

[WWZ*09] WANG R., WANG R., ZHOU K., PAN M., BAO H.: An efficient gpu-based approach for interactive global illumination. *ACM Transactions on Graphics 28*, 3 (July 2009), 91:1–91:8. 2, 7

[YCK*09] YU I., COX A., KIM M. H., RITSCHEL T., GROSCH T., DACHSBACHER C., KAUTZ J.: Perceptual influence of approximate visibility in indirect illumination. *ACM Transactions on Applied Perception 6*, 4 (2009), 1–14. 1, 3