Computers & Graphics 103 (2022) 75–89

Contents lists available at ScienceDirect

Computers & Graphics

journal homepage: www.elsevier.com/locate/cag

Technical Section

Clustered voxel real-time global illumination[☆]

Alejandro Cosin Ayerbe, Gustavo Patow[*]

ViRVIG, Universitat de Girona, Spain

A R T I C L E   I N F O

Article history:
Received 30 September 2021
Received in revised form 16 January 2022
Accepted 20 January 2022
Available online 29 January 2022

Dataset link: https://github.com/AlejandroC1983/cvrtgi

Keywords:
Real-time
Global illumination
Voxelization
Clustering
Diffuse surfaces

A B S T R A C T

Real-time global illumination is an extremely challenging problem because of its intrinsic complexity due to the interplay between complex geometry, multiple light bounces, and stringent real-time frame-rate requirements. In this paper, we present a new technique that enables the real-time computation of global illumination in generic scenes for diffuse surfaces and static geometry. Our technique combines a voxel-based representation of the scene, a voxel-clustering algorithm, and an iterative light-propagation algorithm based on the resulting clusters. Although our implementation is currently designed to handle the first (and main) bounce, the technique allows for multiple light bounces. Summing up all these components results in a flexible algorithm capable of providing global illumination effects and on-the-fly illumination computations.

1. Introduction

Real-time global illumination is still an open problem in Computer Graphics. In the last decades, we have witnessed many attempts to solve light transport in real-time, some of them approaching acceptable similarities with ground truth renders for simple cases, but without fully solving the problem, e.g., without visibility computations [1–3]. Lambertian, glossy and specular materials have been reproduced with different degrees of success through different families of techniques [4,5], each having its own limitations. The resulting techniques already allow impressive, yet limited, real-time visualizations of global illumination in complex scenes.

The ability to reproduce real-time indirect illumination with no strong restrictions in the number of light bounces for all variety of materials with quality near to a production renderer remains unsolved. The main problems are the visibility determination at the point where irradiance is being evaluated, and the recursive nature of that evaluation, as described by Kajiya's rendering equation [6]. Without this information or an adequate substitute, no irradiance gathering can be appropriately performed. Traditional renderers solve this problem through brute force sampling and working at geometry level for the whole scene, which cannot be achieved in real-time for complex scenes [6,7] although recently some advances have been achieved relying in foveated rendering like in the work by Koskela et al. [8]. Bottlenecks like scene geometry, probe density and placement, or voxelization data structure updates populate current real-time techniques [9–11]. Finding means to solve this visibility problem occupies most of the efforts in the field.

In our work, we gather irradiance following a scene-voxelization approach, using a discrete representation of the diffuse indirect illumination at the voxel level. Our design allows multiple emitters and light bounces for dynamic cameras, although we implement only a single light bounce for performance reasons. Following the steps of Thiedemann et al. [12] and Sugihara et al. [13], we extend this family of techniques by clustering voxels to increase the gathering effectiveness and control the number of elements to deal with in the scene regardless of the voxelization resolution, which in the case of a voxel approach can quickly escalate for complex scenes such as the one shown in Fig. 1, with more than half a million voxels. In particular, our contributions are:

- The adaptation of GPU clustering techniques into the voxel domain, using normal directions assigned to voxels to generate the clusters.
- An efficient set of data structures to store voxelization and clustering information, allowing quick retrieval of all fragments, visibility and cluster information for each voxel.
- A real-time global illumination algorithm accounting for visibility based on a 3D version of Bresenham's [14] line algorithm in voxel space, a lazy irradiance evaluation for voxels only visible from the camera and the use of clusters to gather irradiance from large visible portions of the scene

☆ This article was recommended for publication by P. Poulin.
* Corresponding author.
E-mail address: gustavo.patow@udg.edu (G. Patow).

https://doi.org/10.1016/j.cag.2022.01.005

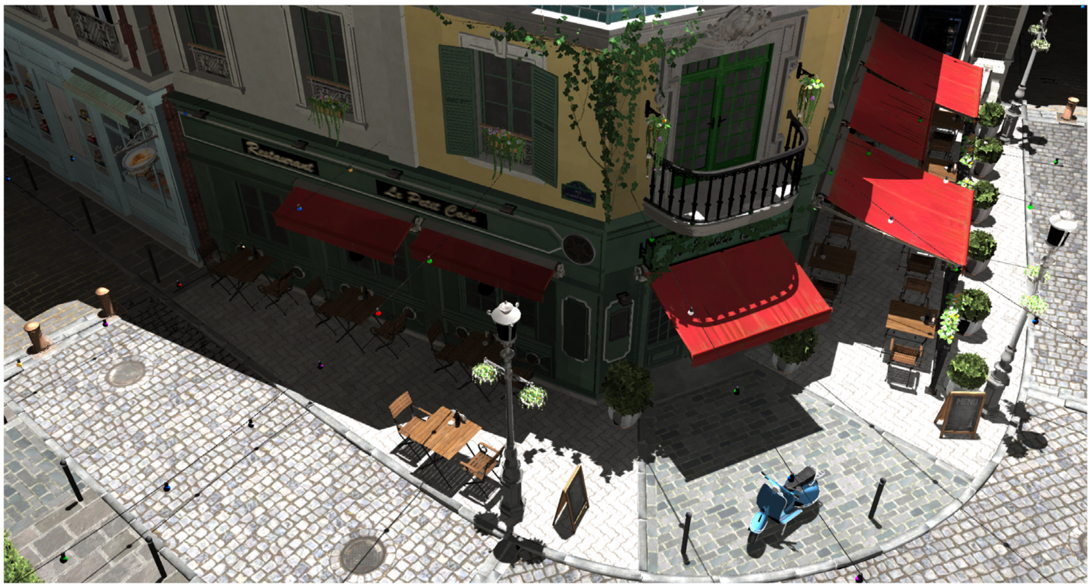


Fig. 1. Real-time single bounce diffuse indirect illumination for the Amazon Bistro Scene with voxelization size 64 at 63 FPS on a GeForce RTX 2060 with 6 GB of VRAM.

from each voxel, avoiding exhaustive visibility tests for the voxels each cluster comprises.

The complete source code of our approach is available for download at https://github.com/AlejandroC1983/cvrtgi.

2. Related work

Real-Time Global Illumination is a challenging research topic being tackled in the last decades by many approaches with different levels of success. The main research directions were established with Instant Radiosity by Keller [15], giving birth to a plethora of techniques based on the idea of building Virtual Point Lights (VPLs) from a shadow map generated at the emitter position, and having as main drawback requiring visibility shadow maps for each VPL. Irradiance Volumes by Greger et al. [16] offered a precomputed cosine-weighted approximation of irradiance in a two-level uniform grid, opening the road to many other techniques relying on the integration of irradiance on a set of points in the scene. Sloan et al. [17] established with their Precomputed Radiance Transfer another reference in real-time GI, modeling also shadows, interreflections and diffuse-to-glossy materials for dynamic environments although requiring a preprocess of the model. Reflective Shadow Maps by Dachsbacher et al. [1] constitutes also a pillar in the field achieving indirect illumination in a cheap and scalable fashion, although occlusions are not accounted for. Other approaches like from Dachsbacher et al. [18] achieve occlusion considering light that needs to be removed or *antiradiance*, taking as basis clustering methods used for hierarchical radiosity. Irradiance Volumes became later the basis for Light Propagation Volumes from Kaplanyan et al. [19] where lattices and spherical harmonics represent the spatial and angular distribution of light in the scene, not requiring any precomputation and accounting for occlusion and participating media.

Nalbach et al. [20] pushed the boundaries of screen space techniques by building a surfel version of the scene and splatting onto a multi-resolution framebuffer, which achieves one-bounce indirect illumination for dynamic scenes. In their work, Mara et al. [21] show how deep G-buffers can be applied to global illumination, using previous frames to perform depth peeling with minimum separation and multi-bounce GI by computing one bounce per frame and accumulating results from several frames. Kol et al. [22] show another way to overcome the Virtual Point Light limitation by rendering the scene multiple times with a combined scene and view hierarchical representation, which allows one to achieve sublinear performance for scene complexity and scene view number, sharing renderings among views. Another major contribution to Irradiance Volumes is the work from Majercik et al. [23], which computes global illumination with fully dynamic scenes and lighting for diffuse, glossy and specular materials and several light bounces with a technique that can be used in production engines. New approaches allow techniques like Precomputed Radiance Transfer to keep contributing to the area. In their work, Currius et al. [24] train convolutional neural networks to estimate light field values through spherical Gaussians for static scenes.

Scene geometry voxelization is an active research field with implications in many areas in Computer Graphics. Starting from the first CPU approaches like the one by Wang et al. [25] offering an efficient 3D voxelization algorithm that, through an analytical 3D antialiasing technique, generates gap-free voxel models. Later versions like the work by Beckhaus et al. [26] gave place to efficient and elaborate GPU methods. Zhang et al. [27] extended with their work GPU conservative voxelization algorithms improving performance and memory footprint compared with previous approaches. Further improvements were introduced by Schwarz et al. [28] achieving an efficient algorithm for a 6-separating surface GPU voxelization, thinner while also gap-free, together with a solid voxelization algorithm. Crassin et al. [29] showed how to build a sparse voxel octree from a voxelized volume, constructing acceleration structures on the GPU thanks to buffer load and store operations, which were added with OpenGL 4.2. In their work, Heitz et al. [30] achieve smooth transitions between LoD levels, local illumination, occlusion and anti-aliasing using an enriched voxel representation of detailed surfaces. Vicini et al. [31] use voxels as well to model both opaque and non-structured geometric aggregates for volumetric scene representations, which are able to approximate level of detail with a quality close to ground truth.

Precomputation times have been one of the main bottlenecks in many Global Illumination techniques for various cases like computing light transport paths, probe location and radiance transport or wavelet representation of light transport [11,32,33]. In the past decade, some voxel-based techniques achieved indirect illumination, avoiding the main bottlenecks of older techniques and considering the whole voxelization volume as the place to simulate light transport, allowing view-independent approaches in scenes with increasing amount of geometry. Crassin et al. [10] relied on elaborate and complex GPU data structures, in combination with mip-mapping, to reduce the cost for evaluating the irradiance at a point by launching several cones around the point's hemisphere. Thiedemann et al. [12] developed a fast voxel path tracer that allowed a quick traversal of the voxelization volume, backprojecting the hit points into a Reflective Shadow Map (RSM) [1] to compute irradiance from that point. These techniques rely on building additional data structures from the voxelized scene, scene postprocessing being a common procedure in the area. Many papers later on have based their work on these two last techniques, e.g., Sugihara et al. [13] extended Voxel Cone Tracing, using voxel information just for determining visibility, backprojecting onto layered RSMs but requiring therefore one RSM per emitter. Chen et al. [34] presented an improvement for both techniques which only considers the set of lit voxels in a scene to perform irradiance computations, avoiding the need to keep the RSM of each light, which can scale poorly in terms of performance and memory, addressing the many-light problem for the voxel-based GI techniques. Irradiance Volume and voxel approaches were also combined by Papaioannou [35], where RSMs approximate irradiance values through spherical harmonics in a uniform grid. This allows a big performance increase since no shadow maps need to be built for visibility testing. Our work is closer to Thiedemann et al. [12] and Chen et al. [13] but we completely avoid RSMs, relying on some scene postprocessing but, unlike methods that require more than 40 min of precomputations for scenes like Sponza Atrium as Silvennoinen et al. [11], we are able to offer GI in less than three seconds at a cost of a lower quality.

There are approaches that avoid to deal with a part, or the totality, of scene elements. Lehtinen et al. [36] completely avoided scene geometry processing by building a hierarchical function basis to simulate light transport for complex scenes using point samples and a scattered data approximation approach. In the geometry domain we can find Delta Radiance Transfer by Loos et al. [37], where they use a set of precomputed shapes that have a predominant role in the scene to compute a coarse version of light transport for dynamic scenes, extending previous work to add indirect shadows and interreflections from finer-scale geometry. Partially avoiding scene elements through clustering geometry is another example. One of the first papers about this topic by Smits et al. [38] achieved improvements of two orders of magnitude for hierarchical radiosity in complex scenes by grouping scene elements and bounding energy transfer error. In their work, Willmott et al. [39] presented another hierarchical radiosity offline algorithm with geometry clustering based on the normal vector direction, using level of detail in some steps, having great performance in larger and more complex scenes. In the real-time GI area, we can find examples of clustering applied to RSM, such as the work by Prutkin et al. [40] where a k-means clustering is applied to the RSM results, considering the resulting clusters as area light sources, greatly reducing the number of VPLs with a low approximation error. Our technique is related to these as it uses clustered voxels as a proxy to simulate light transport, to finally transfer the GI results to the actual scene geometry.

3. Overview

Our technique works by clustering voxels using as criteria the approximate normal direction similarity, as done by Willmott et al. [39]. Working at cluster level allows in the first place, when computing the irradiance reaching each voxel, to gather in a single step all the irradiance of all lit voxels contained at all clusters visible from the one being processed. Of course, this assumes some error for the whole cluster visibility and irradiance arriving from the voxel computations, but greatly reduces the number of evaluations needed. Secondly, the cluster approach keeps under control the number of scene elements to process for the light transport simulation, with only a few thousands of clusters for scenes with more than half a million voxels. Also, we avoid irradiance computation in the final frame, focusing on caching irradiance per voxel face for later interpolation through a lazy computation for voxels visible from the camera. Those two measures greatly decouple the dependency from the scene voxelization resolution, allowing good quality results while scaling up with a much better performance. This supposes an advantage over similar approaches for static scenes.

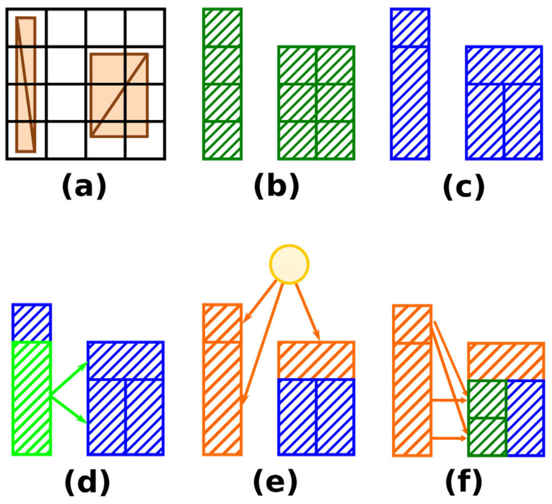


Fig. 2. Pipeline of our system. From left to right: (a) rendered geometry, (b) voxelization step, (c) voxel clustering, (d) cluster visibility, (e) light propagation from an emitter, (f) two voxels (green outline) gather irradiance from lit clusters.

Our goal is to compute, in real-time, the diffuse indirect illumination in a voxel-based 3D scene representation by an aggressive reduction of the elements involved in the simulation thanks to clustering the voxels and using the clusters to compute light transport. Fig. 2 summarizes the most important steps of our approach.

Our technique starts by voxelizing the scene (only once, since our approach is for static scenes) using standard techniques. During this process we also keep the list of geometry fragments generated, called *voxel fragments*, for later reuse. They are stored in streamlined data structures in linear buffers. See Section 4.1 and the supplemental material for a detailed description.

Independently of the efficiency of our data structures, using a voxel-based approach for light exchange might be prohibitive for real-time calculations, so we further reduce the number of interactions by clustering voxels using a criterion based on approximate normal directions. See Section 4.2.

We then compute and cache, for each voxel face of each generated voxel, the visible clusters from that face through ray marching in voxel space. See Section 4.3.

After these steps, real-time light transport can occur. Our algorithm uses, for each face of every voxel in the camera view, the list of cached visible clusters and identifies the lit ones to compute irradiance reaching that voxel face, resembling the irradiance gathering techniques used for off-line rendering. See Section 4.4.

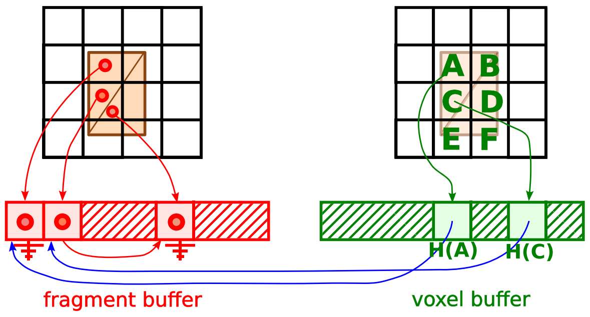


Fig. 3. Voxelization example. Left: Generated fragments (red dots) are stored in a buffer, using a linked-list data structure for each voxel. Right: fragment voxel coordinates (green capitalized letters) are hashed (function *H()*) and tagged in a buffer, using hashed coordinate values as indices. The content of the buffer at that index is the index of the first fragment in the fragment buffer, which allows fragment retrieval for any voxel coordinates.

Our implementation supports dynamic emitters and cameras. Finally, we avoid costly rendering computations in the final frame by computing irradiance with a simple interpolation between the irradiances of each voxel neighbor of the fragments. See Section 4.5.

4. Voxelization, clustering and light transport

4.1. Scene voxelization

We use a standard approach for the voxelization process, where the scene is rendered with an orthographic camera from the three axis–aligned directions with no depth test nor face culling [41], even though it could be done in a faster way as in the method by Schwarz and Seidel [28]. For each voxel, we need to know whether it is empty or occupied, and which are the corresponding generated fragments (the *voxel fragments*), information necessary later on for the clustering stage, see Section 4.2. For this reason, we build a set of auxiliary data structures, such as a linear buffer (the *fragment buffer*) to store all generated fragment information, which avoids extra memory requirements since, in most scenes, only a small percentage of voxels are actually occupied. Please view the supplemental material for an in-depth explanation of this buffer. For each fragment, we store its world position, normal, reflectance, and an index to the next fragment in the same voxel. See Fig. 3(left).

We opted for a trivial hashing with no collision, although voxel data access can be done with more sophisticated parallel-friendly methods [42,43]. At run-time, to speed up access to voxel fragments, we hash the voxel coordinates and use them as indices in a temporal buffer, the *voxel buffer*, with as many indices as the voxelization. At each index position we store the index in the *fragment buffer* of the first fragment generated at that voxel. See Fig. 3(right). Since each voxel can generate an number of fragments that varies greatly, trying to store them in the voxel buffer would represent a challenge: We would either need to keep all fragments generated at each voxel sequentially in a buffer, together with an indirection table storing how many fragments each voxel has, and with the *voxel buffer* pointing to the first fragment in the buffer (equivalent to the current approach); or we would need an extended *voxel buffer* where each element has enough room to store the maximum number of fragments generated at any voxel, which can consume the available GPU memory. For a detailed explanation and example of those data structures, please refer to the supplemental material.

The *voxel buffer* offers quick access to all voxel fragments generated, but its size can quickly top available GPU memory

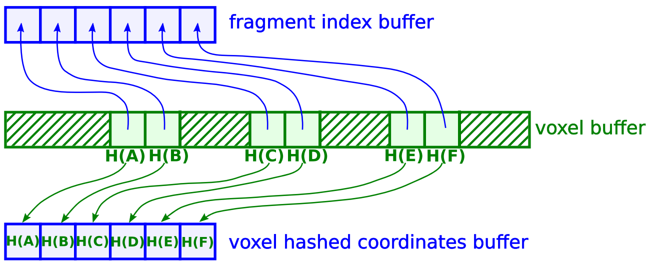

fragment index buffer
voxel buffer
H(A) H(B)
H(C) H(D)
H(E) H(F)
H(A) H(B) H(C) H(D) H(E) H(F)
voxel hashed coordinates buffer

Fig. 4. Compacted buffers: These buffers avoid the use of the original voxel buffer (green), saving memory. Indices to access fragment data for each voxel are stored (upper blue buffer), together with voxel hashed coordinates (bottom blue buffer).

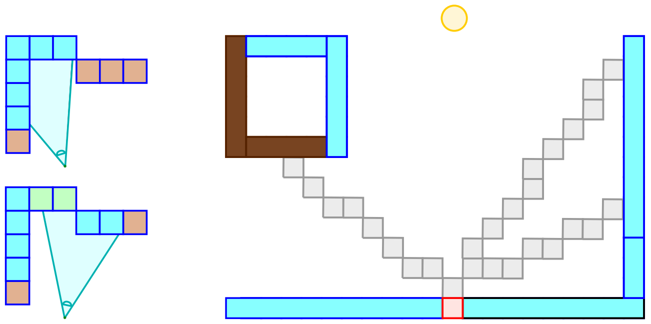


Fig. 5. Left: Irradiance is computed only for visible voxels from the camera, caching the values for reuse. If the camera moves, already computed irradiance is reused (green voxels). Right: Computations for the top face of the red voxel. During the per voxel face cluster visibility step in Section 4.3, several rays in texture space are cast using Bresenham line algorithm (gray voxel lines) in search for visible clusters (brown and cyan clusters intersected by the gray voxel lines). An *atomic OR* operation avoids two or more threads which intersect the same cluster from adding the same cluster as visible more than once (gray voxel lines on the right part reaching the same cluster). During the irradiance gathering step, cached visible clusters are reviewed. Lit clusters (cyan) add irradiance to the voxel face, non-lit clusters (brown) do not add any irradiance. For the irradiance computation we use a differential area form factor formula where the normal directions considered are the voxel face's main axis ($\pm x$, $\pm y$, $\pm z$) and the cluster normal direction, explained in Section 4.2.

if voxelization resolution increases, wasting a lot of memory due to a vast majority of empty voxels. We implement a GPU prefix sum algorithm [44,45] to compact all non-empty indices in the *voxel buffer*. Fig. 4 shows the result of this process, storing only occupied *voxel* indices into the *fragment index buffer*, and their hashed values into another buffer. For a description of the remaining buffers used and an in-depth analysis and examples of how to use them, please refer to the supplemental document for more details on these data structures.

4.2. Voxel clustering

As a second stage, we cluster the non-empty voxels. This functionality allows us to remove negligible voxel–voxel interactions to compute the irradiance for scene elements. See Fig. 5(right) for an example. Once a group of voxels with similar traits have been clustered, the owner cluster is assigned a common mean normal and mean reflectance value, which can be used to gather irradiance in a much quicker way than in a per voxel basis, making also data structures much simpler and flexible.

Mesh clustering and segmentation is still an active research field. Some methods use point clouds as input [46–48], whereas others directly use the geometry [49]. To properly process voxelized volumes, which essentially are 3D images, we found that

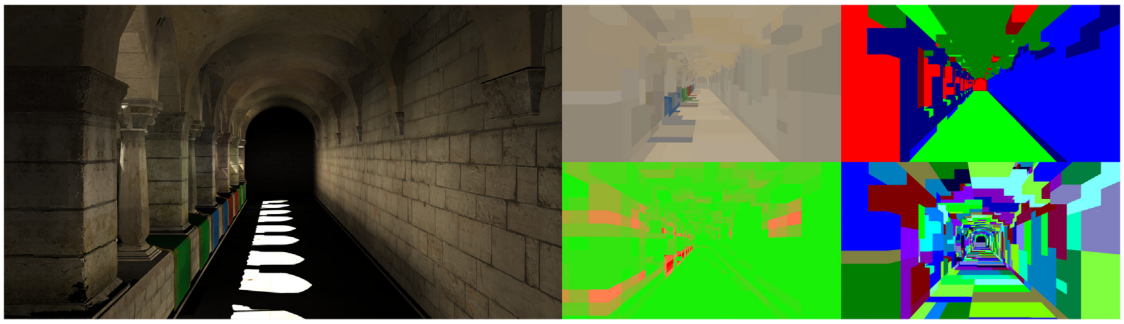


Fig. 6. Sponza Atrium processing at 221 FPS on a GeForce RTX 2060 with 6 GB of VRAM. Left half: screen capture. Right half: from left to right, and top to bottom, cluster mean reflectance, cluster mean normal, fragment density and clustering results. The fragment density goes from 1 (green) to 20 or more (red) fragments per voxel.

methods based on 2D image segmentation could be good candidates. Since we wanted a fully GPU-based implementation, we discarded techniques such as the one by Comaniciu et al. [50], for its dependence on recursiveness; or the one by Vedaldi et al. [51], which does not allow control over cluster size or number. We determined to use Achanta et al.'s [52] SLIC Superpixels, which overcomes all those limitations. This algorithm considers a five-dimensional space *Labxy*, where the first three dimensions (*Lab*) are in a perceptually uniform color space, and where numerical changes map to equivalent changes in perceived color; and the other two *xy* correspond to pixel position. Pixel clustering uses a distance measure in this space, making the clusters to have similar size and area.

In our case, the distance function used for voxel clustering is not based on color, but on an approximate normal direction, using a distance function that favors only voxels with a normal very similar to the cluster's normal. Eq. (1) shows the distance function, where we compute the angle between the normal direction of the cluster and the normal direction of the candidate voxel to be added to the cluster. If the angle is smaller than 35°, the voxel is added to the cluster. Since several clusters can be trying concurrently to add the same voxel, we store the minimum distance of each voxel to the latest cluster it was added to, allowing each cluster to check atomically if it is possible to minimize the distance to a specific voxel and add it. As a consequence, a cluster will always add a voxel tested for the first time, even if the angle between the two normal directions is larger than 35°. This minimizes the number of voxels that are not added to any cluster. See Algorithm 1 for a breakdown of the steps followed during voxel clustering.

$$\text{distance}(n_0, n_1) = \begin{cases} 1 & \text{if } angle(n_0, n_1) < 35^\circ \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

To help the clustering step to classify voxels, we have to determine the normal direction for each voxel, which can involve some computations since some voxels in the scene can have a large number of fragments. The size of a voxel in world space depends on both the voxelization resolution and the size of the scene being voxelized. However, independently of the resolution of the voxelization, there will always be some voxels with more than one fragment (e.g., at edges or corners of the geometry). Since geometric density cannot be controlled, many fragments per voxel can be generated in dense regions of the scene, giving place to a high disparity in the normal directions of the fragments generated. See Fig. 6(right), fragment density. Due to this disparity, we follow a two-step approach to determine an approximate normal, called *heuristic normal*. For each voxel we consider a volume of 3 × 3 voxels in each of the major axes (*X*, *Y* and *Z*) in

Algorithm 1 Voxel Clustering

```
1:  Compute mean normal for each voxel
2:  for $numIterations = 10$ do
3:      Init cluster-voxel distance buffer with max value
4:      for each cluster do
5:          for each voxel in cluster neighbourhood do
6:              if voxel is occupied then
7:                  if distance(cluster normal, voxel normal)
                            is minimized then
8:                      assign voxel to cluster
9:                  end if
10:             end if
11:         end for
12:     end for
13:     Compute new cluster centers and mean normal directions
14: end for
15: Compute cluster info (AABB, main direction, reflectance)
16: Prefix sum pass with generated clusters to compact results
17: Compute generated cluster neighbours
```

both the positive and negative directions. For each 3 × 3 grid, we count the number of voxels that are not occupied, which leads to a lower occlusion of the voxel being analyzed. We then pick as normal direction the axis with the smaller number of occupied voxels. In case of a tie, we use the per-fragment normal data as criteria to determine which direction to choose. We accumulate the angle formed by each normal and the major axes analyzed and we pick the direction with the lowest accumulated value. Fig. 7 shows an example of this heuristic computation. Voxels in need of this heuristic approach do not always generate a satisfactory outcome, possibly resulting in light or shadow leaks, since the resulting clusters adopt as mean normal the one given by the voxels they include. However, in all our experiments this has not been a problem, and its effect is even further reduced by the use of filtering techniques, as explained below.

Our implementation starts with a user-defined number of empty clusters, ranging between 60 K and 250 K. We base this number on two points. First, we make sure the total number of voxels each empty cluster has to test is not too large while guaranteeing empty clusters to be densely and uniformly distributed in the voxelization volume before the clustering process starts. The second and most important point is to keep the final number of clusters generated below 10 K, which in our experience is enough to offer good results. In the sixth row of Fig. 12 we can see how an excessively small number of clusters can result in a lower frequency irradiance and can also generate light leaks. We achieve the 10 K cluster limit for all scenes tested after 10 iterations of our clustering algorithm. See Fig. 8 for a detailed

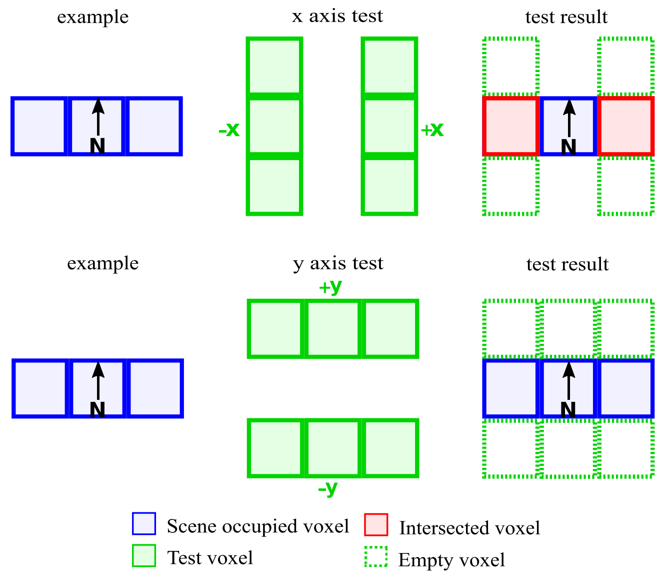


Fig. 7. Heuristic normal computation. In the top row we show how we apply the x axis test. In the leftmost part we show a set of voxels, considering the central one for testing. In the center, we show the sets of voxels to be tested for the negative and positive x directions. In the rightmost part we can see how both tests lead to one occupied voxel, highlighted in red. In the bottom row we show the y axis test, with the same set of voxels and central voxel to be tested. The central part shows the voxels to test for the positive and negative y directions, the rightmost part shows the results. As we can see, the y axis test is able to find more free voxels (3) than the x axis test (2). Since both positive and negative y have the same number of non occupied voxels, to decide between both directions we consider the accumulated angle between the $+y$ and $-y$ directions and the normal directions of the fragments present at the voxel being tested, which in this case is a single fragment with a normal N pointing in the $+y$ axis. The direction $+y$ is the one minimizing the accumulated angle value, and is therefore selected as the heuristic normal.

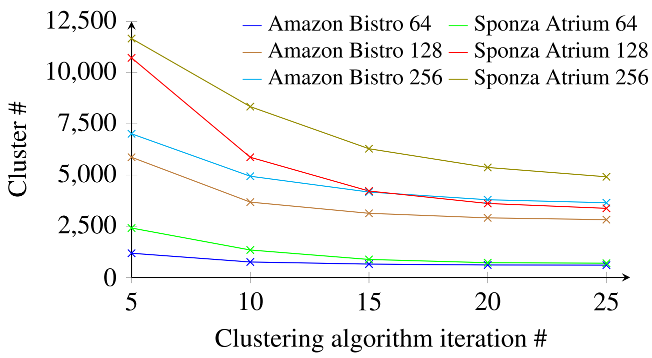


Fig. 8. Number of generated clusters depending on the number of iterations of the clustering algorithm for Sponza Atrium and Amazon Bistro scenes for voxelization resolutions 64, 128 and 256. As can be seen, we achieve our goal of keeping the number of generated clusters below 10 K from iteration 10 onwards.

plot of the number of voxels generated for the Sponza Atrium and Amazon Bistro scenes at voxelization sizes of 64, 128 and 256, depending on the number of iterations of the clustering algorithm. The empty clusters are disposed in a uniform grid scheme according to Achanta et al. [52]. See Table 4 for a detailed description of the number of empty clusters and final clusters generated for different voxelization sizes and scenes. Each initial empty cluster is driven by a separate thread in a compute shader, each collecting voxels in a greedy way. Non-empty clusters are stored in an array with additional information generated in later steps (i.e., index to the voxels owned by the cluster, mean reflectance, mean normal, AABB, neighbor clusters, etc.).

Once the clustering finishes, the vast majority of the voxels have been assigned a cluster. The remaining ones will not be taken into account as sources of irradiance when performing light gathering. Voxels in this situation ranged from 0.15% and 0.4% of the total voxels of the scene for the Sponza Atrium and Amazon Bistro scenes respectively, to around 10% and 4.4% in the case of a fine-grained voxelization. One possibility would be to do a pass checking all voxel assignments, and creating new clusters for each unassigned voxel. However, in our current implementation we determined to skip this step as, in our experiments, this approximation had a small impact on the simulation. See Fig. 9 for a comparison of irradiance gathering between our current approach and when adding an extra pass that assigns all unassigned voxels to the nearest cluster, taking them into account for irradiance gathering.

4.3. Per voxel face cluster visibility caching

Since our technique works for static scenes, we perform a single fast visibility step where we cache the clusters visible from each voxel face. Later, when light exchange computations are performed, this allows us to reduce the computation of the irradiance reaching a voxel face. This is done by testing the list of visible clusters from that voxel face, looking for clusters tagged as *lit*, meaning they have at least one voxel visible from the emitter, and therefore receiving irradiance. For each voxel face, we dispatch 128 compute threads. Each thread shoots a ray from its corresponding voxel face into the upper hemisphere with respect to that voxel face's normal direction ($\pm x$, $\pm y$, $\pm z$ axis) using a 3D implementation of Bresenham's [14] line algorithm. If a non-empty voxel is found, we take the index of the cluster of that non-empty voxel and add it to the list of clusters visible from that voxel face. The index of the cluster found is atomically verified to avoid two or more threads from the same voxel face being tested from adding the same cluster index more than once. The process to build the list of visible clusters per voxel face follows a similar approach to other buffers in our work aiming to save memory: We build an initial buffer with size *(number of generated voxels)* $\times$ *(voxelization resolution, i.e., 64, 128 or 256)* $\times$ 6 and later perform a prefix sum operation to compact all indices, generating also two more buffers that indicate, for each voxel face, how many visible clusters that voxel face has and where in the compacted buffer those cluster indices begin. This allows us to use only between 14% and 22% of memory when compared with the brute force approach. Table 3 shows a comparison of the performance in the clustering visibility and final frame stages when using 32, 64 and 128 rays per voxel face for different voxelization sizes for the Sponza Atrium and Amazon Bistro scenes. We can see a ratio of 6–10 more time needed to compute the cluster visibility as the number of rays per face grows. See Fig. 12 for a comparison of the quality in the Amazon Bistro scene when using 32, 64 and 128 rays per voxel face at different voxelization resolutions.

4.4. Real-time light transport

Light transport computation consists of four steps. The first one builds one shadow map from each emitter, a *scene geometry distance shadow map* used in the second step to determine if a given voxel is visible from an emitter, allowing us to inject irradiance.

In the second step, we identify lit voxels and their respective clusters. We accumulate irradiance from the emitter into the voxels, and then we *pull* it to their owner clusters, so we can later test at each voxel face whether any visible cluster has irradiance and compute a light bounce. A compute shader analyses per voxel visibility from the emitter, atomically accumulating

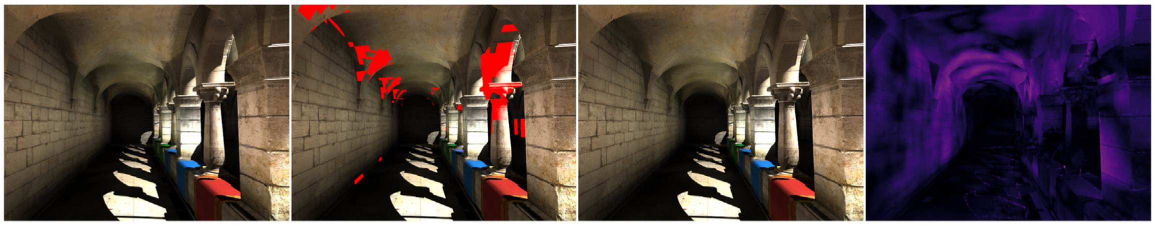


Fig. 9. From left to right: Sponza Atrium scene at a voxelization of size 256 (leftmost), showing in red the voxels that have not been assigned to any cluster (center-left). We added an extra pass to assign all unassigned voxels to its nearest cluster (center-right). A difference between the scene where not all voxels are assigned to a cluster (leftmost) and the scene where all voxels have been assigned to a cluster (center-right), generated using NVIDIA's FLIP [53] tool. As we can see in the leftmost image, taking into consideration all scene voxels as sources of irradiance for light gathering represents a small contribution in the final results for this scene and this voxelization size, which has the highest number of unassigned voxels to clusters (around 10%).

(through the use of GLSL *atomicAdd* to prevent other threads from overwriting irradiance to the same cluster), for each lit voxel, the irradiance into its owner cluster. We avoid a filtering pass and any 2D postprocess in the scene geometry distance shadow map regarding voxel visibility by testing just a set of points generated around each voxel, one from the center of each face and one from the center of each edge, displaced from the voxel center 1.5 times the size of a voxel. See Fig. 10(left). This simplification can lead to light leaks, but they are vastly avoided by using a simple neighbor occupied test for each lit voxel, as described in Fig. 10. Also, two buffers, one for each occupied voxel and one for each occupied cluster, are used to identify lit voxels and clusters with at least one lit voxel. One possible approach for the lit voxel test could be to project the shadow map texels to world space and verify if they fall inside or are close enough to an occupied voxel. This would imply projecting all the texels of the geometry distance shadow map which, for lower resolutions like 2048 $\times$ 2048, already suppose processing several millions of elements. This option requires further research since on one side, it could improve the performance in scenes where the number of generated voxels is too large, but on the other it might impact the performance of scenes with a small number of voxels, since each occupied voxel found requires a search of its index in the compacted data structure, which has a cost due to memory bandwidth consumption. To support more than one emitter, we only require to compute the distance shadow map for each one and repeat the test for the set of points generated around each voxel. Our approach can therefore support an arbitrary number of (primary) emitters with a straightforward extension of the current implementation. We currently test all voxels in the scene, not applying any culling technique. Adding simple approaches like frustum culling could increase performance, specially helping to avoid texture data access when testing the set of points for a culled voxel. As Sugihara et al. [13] did, we use a proxy data structure (the clusters in our case) to store irradiance from the emitters, with the advantage that we only need to deal with a few thousands of clusters per scene on the worst case, independently of the number of voxels used.

The third step computes the light bounces. First, a compute shader determines the voxels visible from the main camera using a small geometry distance shadow map in the same way the lit voxels are identified from the emitter. In this case, having a lower accuracy does not have big consequences like light or shadow leaks. Therefore we just generate 12 points around each voxel at a distance twice the size of a voxel for the test. This avoids computing irradiance reaching non-visible voxels, and to reuse already computed results whenever possible. See Fig. 5(left). Then, the core of the light transport simulation is computed: For each visible voxel, we dispatch 64 threads per *voxel face*, caching the gathered irradiance at each voxel. This per voxel irradiance caching can be done at the center of the voxel, at each voxel face, or using a finer subdivision. We opted for doing it for each voxel face since it eases per voxel face Gaussian filtering and indirect lighting interpolation with neighboring voxels in the final lighting pass. Each thread takes up to two cached visible clusters from its corresponding voxel face, and verifies whether each cluster has any lit voxels. If it has, the accumulated irradiance at the cluster is gathered and assigned to the corresponding voxel face being tested. Fig. 5(right) shows an example. In our experiments, taking from one to up to four visible clusters per thread generated a performance penalty between 16% and 35% respectively when compared with the two-cluster case, two clusters per thread being the value that offers a better balance between the number of threads dispatched and the time and memory needed to process each cluster. When the voxelization is too coarse, irregularities may appear because of considering the cluster center as the geometric origin of irradiance. To prevent this, for those clusters closer than a given threshold from the voxel face tested, instead of adding the computed cluster irradiance, we loop through each of the voxels the cluster owns and we add the irradiance of each lit voxel found. We follow this approach regardless of the voxelization resolution and the scene size.

The final step is to apply twice a Gaussian filtering to remove noise in the gathered irradiance. For each voxel visible from the camera, we apply a Gaussian filter to the faces in the same direction of its neighboring non-empty voxels. To avoid shadow leaks due to non-lit or empty neighbor voxels, the kernel discards these cases by assigning a weight of 0, before renormalization. In Fig. 11 we compare two Cornell Box scenes, manually configured to be as similar as possible, one with our technique and one with a ground truth render. Thanks to the Gaussian filtering the ceiling is able to blur and distribute strong irradiance values reaching a few voxels, achieving similar results to the ground truth. See Fig. 12 for an in-depth comparison of results with and without filtering.

When the emitter changes its position or orientation, indirect lighting is rebuilt only for the voxels visible from the camera, saving unnecessary computations, which in our experiments are usually more than 70% of the total candidate exchanges for the rendering-oriented scenes we tested. In case only the camera moves, irradiance is computed only for those visible voxels whose irradiances have not yet been computed, caching any previous computation. See Fig. 5(left). This eventually removes any unnecessary re-computations as long as the emitter does not change position or orientation.

As the first bounce carries most of the total energy exchanged [54], our current implementation only accounts for this bounce. In Fig. 13 we compare our work with a ground truth render (obtained with the ray-tracer LuxCoreRender). In Fig. 14 we show a comparison of our work with other real-time techniques and a ground truth render in a more complex scene. For successive light bounces, we would only need, for each bounce and camera visible voxel, to extend the computation of irradiance to all voxels visible from each of those voxels that contribute to the camera

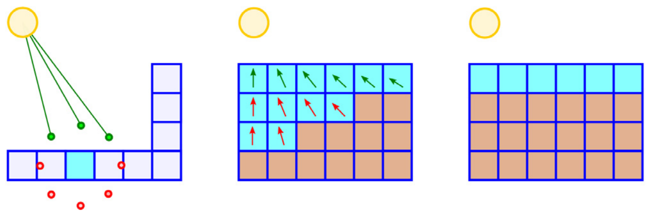


Fig. 10. Lit voxel computations. Left: Points generated around each voxel center at distance 1.5 times the size of a voxel to test for the visibility from the emitter. Center: Scene geometry distance shadow map precision can sometimes lead to voxels which are not visible from the emitter to be identified as visible (red arrows) even if using high resolution shadow maps for scenes like Sponza Atrium. Right: A simple neighboring test verifies if the voxel identified as lit has an occupied neighbor in the light direction, to reduce these issues.

Table 1
Timings in ms for each one of the most important steps in our approach, for 64^3, 128^3 and 256^3 resolutions for the Sponza Atrium and a 1.8M triangles version of the Amazon Bistro scene.

	Sponza Atrium			Amazon Bistro		
Voxelization size	64	128	256	64	128	256
Voxelization	33.5	46.9	82.9	231.6	246.9	256.8
Prefix Sum	3.1	9.7	195.8	1.49	8.9	58.7
Clustering x10 loop	235.1	1512.8	1512.9	205.1	1426.6	1244.8
Cluster Visibility	108.3	580.4	3676.6	104.6	605.2	5431.9
Light bounce	1.7	6.3	32.4	0.4	1.9	7.4
Frame (Full HD)	3.3	3.7	4.2	9.8	10.2	11.6

Table 2
Breakdown of techniques performed for each light bounce update for 64^3, 128^3 and 256^3 resolutions and their times (ms) for the Sponza Atrium and a 1.8M triangles version of the Amazon Bistro scene. The *Distance Shadow Map* (i.e., scene geometry distance shadow map) and *Lit Cluster* rows correspond to the single emitter present in the scene.

	Sponza Atrium			Amazon Bistro		
Voxelization size	64	128	256	64	128	256
Dist. Shadow Map	2.9	2.8	2.7	8.7	9.8	9.7
Lit Cluster	0.2	0.8	2.6	0.3	0.8	3.2
Camera visible voxel	0.1	0.4	1.9	0.1	0.1	0.5
Light bounce	1.7	6.3	32.4	0.4	1.9	7.4
Gaussian filtering	0.9	2.9	22.3	0.6	1.0	8.1
Total	5.8	13.2	61.9	10.1	13.6	28.9

Table 3
Comparison of times in ms for the cluster visibility and final frame (1080p) stages in Sponza Atrium (top table) and Amazon Bistro (bottom table) at voxelization resolution of 64, 128 and 256 when using 32, 64 and 128 rays per voxel face.

Scene	Sponza Atrium								
Rays/Face	32			64			128		
Voxel Size	64	128	256	64	128	256	64	128	256
Cluster Vis.	5.4	34.2	438.4	6.7	41.1	760.2	108.3	580.4	3676.6
Frame Time	3.4	4.1	4.1	3.1	3.8	4.3	3.3	3.7	4.2

Scene	Amazon Bistro								
Rays/Face	32			64			128		
Voxel Size	64	128	256	64	128	256	64	128	256
Cluster Vis.	3.2	27.3	314.3	2.9	29.2	276.9	205.1	1426	1244
Frame Time	10.3	11.0	11.1	10.3	10.1	11.3	9.8	10.2	11.6

view, information we already have from the cluster visibility step. The implementation would not require any extra data structures, just an extended implementation of the light bounce algorithm that adds irradiance from the precomputed visible clusters. One option we consider to increase performance of successive light bounces is to limit by distance which clusters add irradiance to each voxel face, this idea being left for further research.

4.5. Final frame

Finally, each rendered fragment, knowing its own world position, computes its own voxel coordinates. Through the data structures presented in Section 4.1, we recover the index in the buffer where we store per voxel filtered irradiance information, considering the voxel face most similar to the fragment normal direction. Direct lighting is computed through a form factor, while indirect lighting is computed through bilinear interpolation with its neighboring voxels. Both contributions are added and multiplied by the fragment reflectance. Also, we apply a post-process FXAA [55] pass to reduce aliasing artifacts.

5. Results & discussion

5.1. Results

We developed our technique in C++ using Vulkan as real-time and compute graphics API. We used for all the measurements a computer with a Ryzen 7 3800XT AMD CPU, 16 GB of RAM memory and a GeForce RTX 2060 graphics card with 6 GB of VRAM memory. All captures were done at 1920 $\times$ 1080 resolution.

Table 1 shows times measured for the most important steps in our approach for the Sponza Atrium (262 K triangles) and a 1.8M triangles version of the Amazon Bistro scene, which originally has 2.8M triangles (we removed some of the peripheral parts of the scene since they were not playing any role in the simulation), for three different voxelization resolutions. The light transport computation performed when the emitter moves offers real-time frame-rates (5.8 ms to 61.9 ms). We can observe that, as voxelization resolution increases, there are two main factors that affect the *cost* of the light transport computation. First, the number of voxels to test for light gathering increases proportionally to the resolution of the voxelization. Second, the increasing number of evaluations for querying data structures and Bresenham's line algorithm traversal. GPU prefix sum and clustering steps also grow as voxelization resolution increases. On the other side, scene voxelization and final frame times offer a quite stable performance, the final frame times being specially interesting, always below 12 ms for a full HD viewport: thanks to the simplicity of our data structures, the process to compute irradiance at each fragment is the same for every voxelization resolution, consisting of an interpolation of the neighboring voxels, while the heavy light transport computations are all performed only when the emitter is updated or new voxels, whose irradiance has not yet been computed, become visible. We observed the number of scene elements rendered affected the performance of the final frame times, specially for the most complex scene analyzed, Amazon Bistro. After adding a simple compute frustum culling implementation based on the work by Willems [56] we observed an increased frame-rate of about 120%. More sophisticated occlusion culling and LOD algorithms like the one by Hudson et al. [57], which in similar urban environments are able to cull an average of 55% of the geometry, could further improve the final frame times.

As we can see, there is a clear trade-off between the voxelization resolution, the number of rays shot for the cluster visibility computations and the size of the Gaussian filter used to smooth the results out. Fig. 12 shows a comparison of indirect illumination results for different voxelization resolutions maintaining the same number of rays per voxel face and the Gaussian kernel size. As voxelization resolution increases, the need for a wider Gaussian filtering becomes evident.

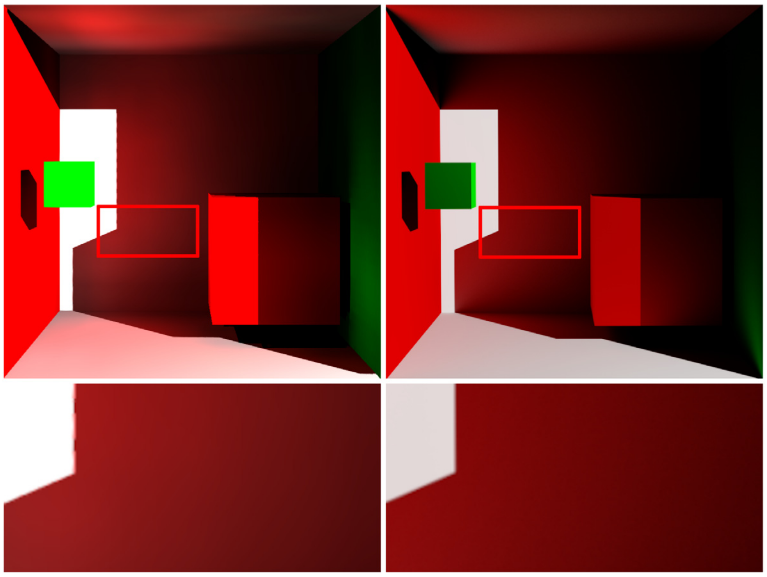


Fig. 11. Two Cornell Box scenes. Top row: Our technique with 128 rays per voxel face at 32^3 resolution at 786 FPS on a GeForce RTX 2060 with 6 GB of VRAM (left) and ground truth at 1024 samples per pixel rendered with Cycles (right), peak signal-to-noise ratio error between the two images is 20.122. In the bottom row, details of our technique and the ground truth render (PSNR error between both images 23.233). In both cases the scenes have been configured manually to be as similar as possible.

Table 4
Voxelization vs. clustering comparison and memory usage for 64^3, 128^3 and 256^3 voxelization resolutions for the Sponza Atrium and an 1.8M triangles version of the Amazon Bistro scene. The Memory Total row represents the amount of memory in MB used by our technique including temporal buffers not used in the light bounce and final frame steps. The Memory Final row shows the amount of memory in MB finally used by our technique.

	Sponza Atrium			Amazon Bistro		
Voxel size	64	128	256	64	128	256
Voxel #	23K	107K	514K	15K	68K	300K
SupePixel #	30K	250K	120K	30K	250K	120K
Cluster #	1357	7873	8733	745	3685	4983
Cluster %	5.9%	7.3%	1.7%	5.0%	5.4%	1.6%
Mem. Total	90.7	428.5	1945.8	60.9	284.9	1190.3
Mem. Final	13.4	77.9	313.6	7.9	42.7	176.3

5.2. Light bounce performance

Table 2 shows a breakdown of the techniques performed when updating the emitter for light transport and the associated times in ms. The camera view for this time measurements is similar to the one in Fig. 6 for the Sponza Atrium scene, and similar to the one in Fig. 12 for the Amazon Bistro scene. We used the same size of tridimensional Gaussian filtering kernel (5^3) for all different voxelizations. In Fig. 15 we show the time per frame in ms for a scene camera traversal and dynamic emitter update for both the Sponza Atrium and Amazon Bistro scenes at a voxelization with resolution 128. The values for the dynamic emitter update show stable and bounded values around 16 ms for Sponza Atrium and 20 ms for Amazon Bistro, since using the same camera view yields the same computations for each emitter update. The times in the scene traversal show some spikes for both scenes, which correspond to irradiance being computed for those voxels that become visible as the camera traverses the scene, also expected due to the geometrical complexity of the scene. In both cases the frame times are well bounded, offering a stable behavior. Since we store all the irradiance in a common and view-independent structure (the generated clusters), each primary emitter in the scene just requires a scene geometry distance shadow map pass, which is already present in many game engines, and to test the visible voxels from that emitter. The first two rows in Table 2 show the associated times for an emitter in Sponza Atrium and Amazon Bistro for different voxelization sizes, ranging from 3.1 ms to 12.9 ms. We detected a performance bottleneck in the scene geometry distance shadow map. Disabling all compute thread dispatches during a light bounce (lit cluster, camera visible voxel and light bounce) reduced the time needed by the scene geometry distance shadow map by about 40%. A performance analysis with NVIDIA's Nsight showed a higher amount of time of idle unused warp slots, affecting the time the shadow map step needs to complete. Further research is needed to proper diagnose this bottleneck and to find the best order for the techniques in our pipeline to maximize GPU performance. Our technique could support analytical area lights, the only requirement for an emitter being to be able to test whether each voxel is visible. Point lights could also be supported, requiring to test voxel visibility from each one of the six scene geometry distance shadow maps generated. The cost to generate those shadow maps could be compensated with techniques like improved dual-paraboloid shadow maps (Vanek et al. [58]).

The three parameters affecting the *quality* of our technique are the number of rays per voxel face during the cluster visibility cache computation, the filtering kernel size, and the number of clusters. We show in Fig. 12 their influence on the results. In the first row, we show captures for the same scene at 64, 128 and 256 voxelization resolutions with the default parameters (128 rays per voxel face for cluster visibility, Gaussian filter kernel of size 5^3). The second and third rows cover the cases when 64 and 32 rays per voxel face for cluster visibility are considered, the latter one showing some small sampling problems like a lower gathered irradiance resulting in a darker indirect illumination, and also a slightly poorer quality, as can be seen in the reddish indirect illumination in the windows on top of the awnings. See Table 3 for a comparison of the performance when using 32, 64 and 128 rays per voxel face in the clustering visibility and frame time stages for different voxelization sizes for both the Sponza Atrium and Amazon Bistro scenes. The fourth row in Fig. 12 shows the effect of using a Gaussian filter with a 3^3 kernel instead of the 5^3 usual one. In this situation, as the density of the voxelization increases, the error to properly filter irradiance

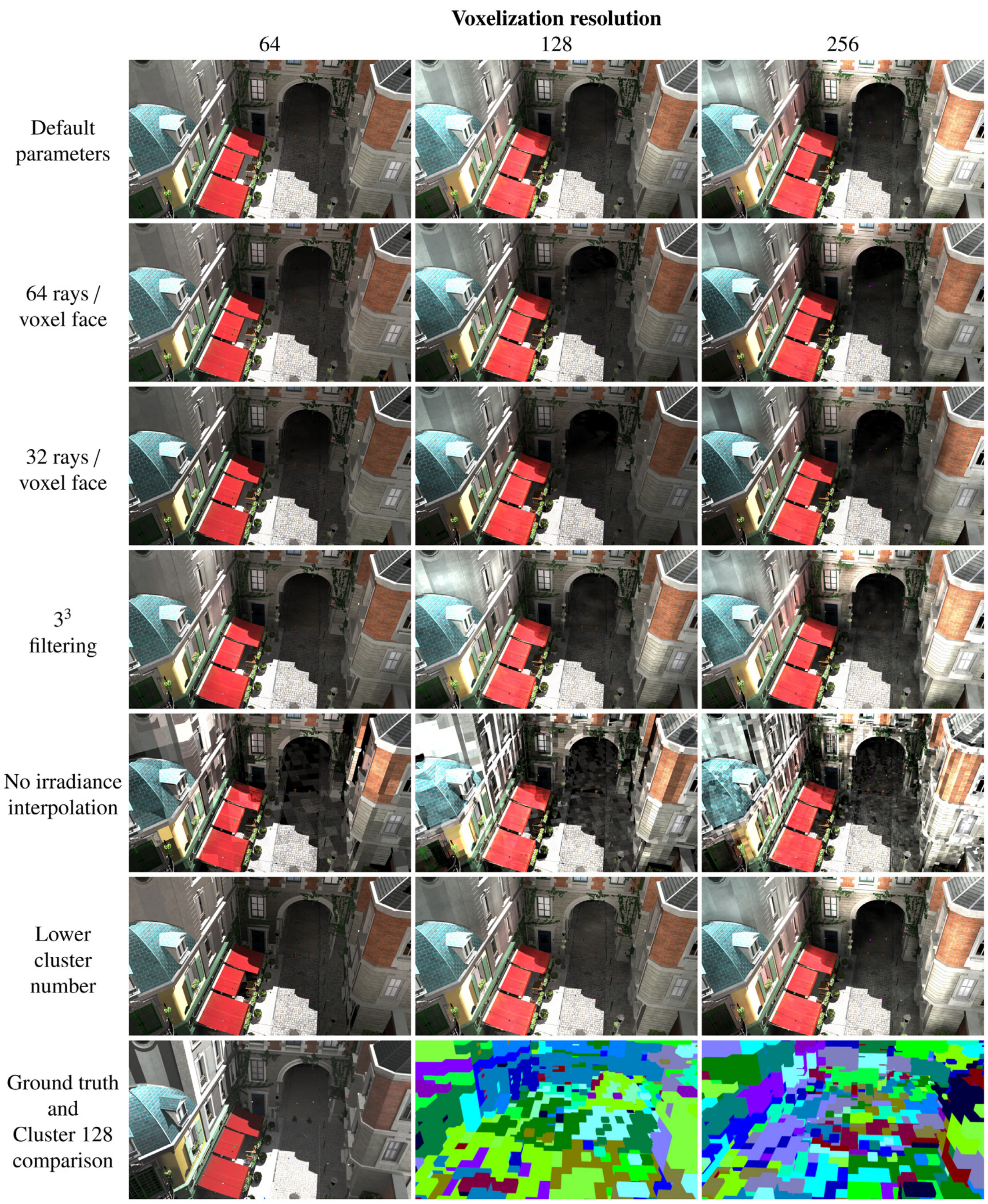


Fig. 12. First row: Example scene at three different voxelization resolutions: 64^3 (left), 128^3 (center) and 256^3 (right), with 128 rays per voxel face and Gaussian kernel of size 5^3. As can be seen, the 64^3 case is unable to reproduce the light bounce on the blue roof, while the 256^3 case starts suffering from irradiance aliasing due to the need of a larger Gaussian kernel filtering. Second and third rows: Influence of the numbers of rays per voxel face. Fourth row: Smaller Gaussian filtering kernel size. Fifth row: No irradiance interpolation. Sixth row: Small cluster number for three different voxelization resolutions displayed in the first row. Insufficient number of rays per voxel face leads to inhability to properly sample the environment, visible in the 32 ray per voxel face case for the 64^3 and 128^3 voxelizations. A Gaussian filter of size below 3^3 makes irradiance distribution not uniform, noticeable in the 256^3 voxelization case. A smaller number of clusters reduces the frequency of irradiance changes, leading to a poorer quality, the 128^3 voxelization being a good example. In the last row (seventh), a ground truth render at 1024 samples per pixel with Cycles (Blender v2.83) is shown (left), together with a clustering using only 10% of the usual clusters (center), showing the default results on the right.

from neighboring voxels increases, as can be seen in the 256 voxelization resolution case. The need to interpolate irradiance from neighboring voxels is shown in the fifth row. The sixth row shows how the number of clusters influences the results: A small number limits the frequency of the irradiance distribution, which is specially visible in the 128 voxelization case. The last row shows how the number of generated clusters vary when using a too small number of clusters together with a ground truth render. Comparing the ground truth render with our work in the first row of Fig. 12, as voxelization resolution increases we can appreciate a more detailed indirect illumination and occlusion, although the 256 voxelization case shows some light leaks in areas with a weak illumination due to the size of the Gaussian kernel 5^3 not being big enough to cover a large enough area around each filtered

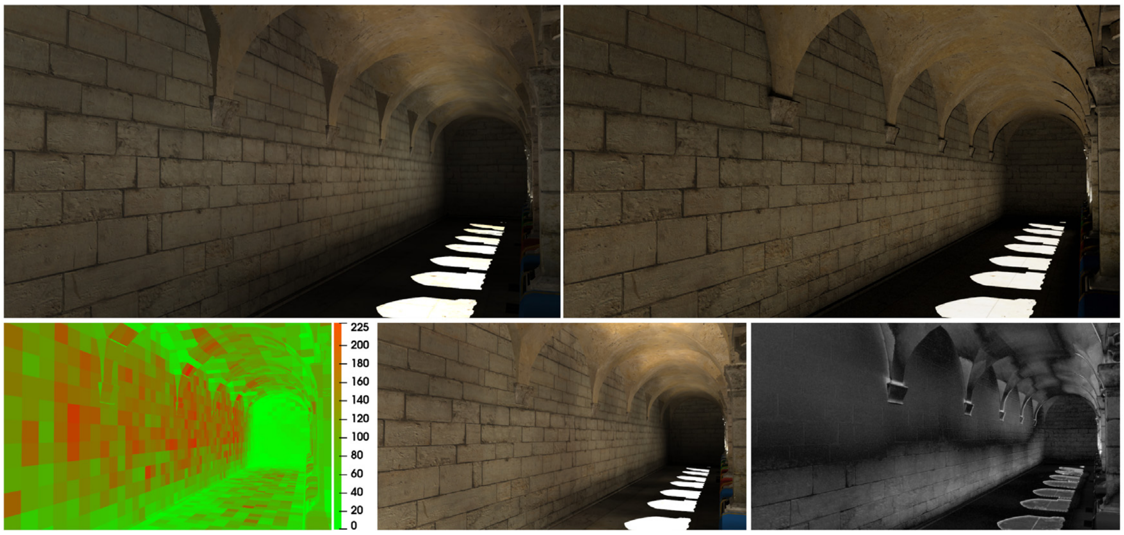
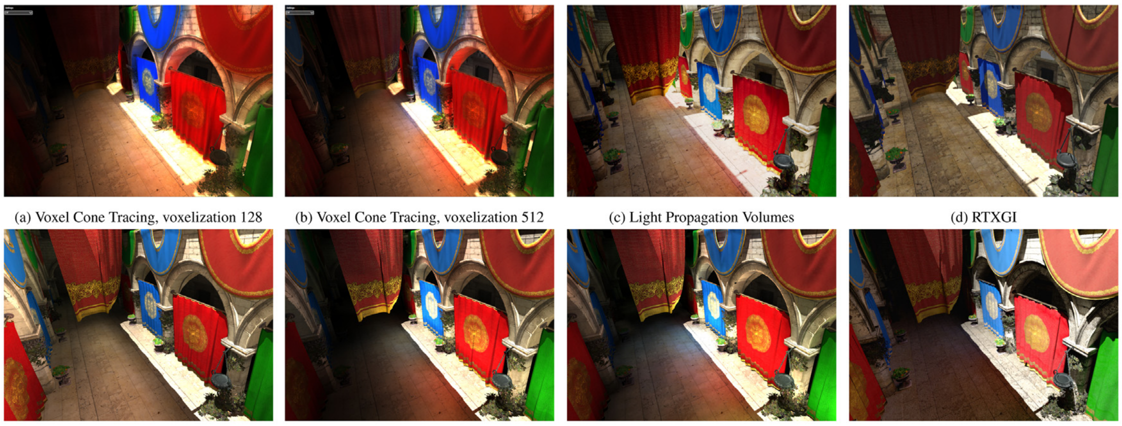


Fig. 13. Our technique at 258 FPS (top left) and a ground truth render at 1000 samples per pixel (top right) of the same scene. As can be seen, the irradiance distribution in our work resembles the render results, although the irradiance reaching some voxels can vary in some cases (bottom left) due to the rays traced in voxel space coming across more clusters containing lit voxels. The color code varies from green (no lit voxel found) to red (225 or more lit voxels found).



(a) Voxel Cone Tracing, voxelization 128 (b) Voxel Cone Tracing, voxelization 512 (c) Light Propagation Volumes (d) RTXGI

(e) Our work with voxelization size 64 (f) Our work with voxelization size 128 (g) Our work with voxelization size 256 (h) LuxCoreRender render with 1024 samples per pixel

Fig. 14. Light propagation in Sponza Atrium for different techniques: Voxel Cone Tracing (VCT) at 128^3 voxelization (a), Voxel Cone Tracing at 512^3 voxelization at 613 FPS on an RTX 3090, which roughly is 250 FPS on an RTX 2060 (b), Light Propagation Volumes (LPV) in Unreal Engine 4 at 197 FPS (c), RTXGI at 108 FPS with Unreal Engine 4.25.3 RTXGI branch and RTXGI v1.1.12 using a $16 \times 16 \times 16$ probe grid (d), our technique at 64^3 voxelization at 199 FPS (e), our technique at 128^3 voxelization at 183 FPS (f), our technique at 256^3 voxelization at 163 FPS (g) and a render of the same scene with LuxCoreRender at 1024 samples per pixel (h). Both VCT images courtesy of Cyril Crassin. At 128^3 voxelization our technique is able to reproduce color bleeding where VCT is not. At 256^3 voxelization we offer higher frequencies and more detailed irradiance gathering than VCT at 512^3 while only using less than 30% of memory. See Fig. 17 for a detailed memory comparison. Note that Unreal Engine's implementation of LPV does not have occlusion. Note also that both LPV and RTXGI captures do not have alpha maps. All images in this figure have been configured manually.

voxel. The 3^3 filtering case illustrates, in the same figure, how those light leaks aggravate for the 256 voxelization case. Reducing the number of clusters and using a smaller number of rays per voxel face in the cluster visibility stage can help minimize this issue, as can be seen in the *64 rays/voxel face* and *Lower cluster number* cases, in Fig. 12.

In Fig. 13(top-left) we can see another limitation of our technique, where the back wall is unable to gather enough irradiance when compared with the ground truth render in the top-right part of the figure. The lack of visible lit clusters from the voxels on that wall, depicted in the bottom-left part of the figure, avoids those voxels from gathering enough irradiance. This could be solved by redistributing the sampling rays, increasing ray density (bottom-middle in the figure).

5.3. Ground truth comparison

Fig. 13 shows a comparison between our technique and a ground truth render with LuxCoreRender [61]. As can be seen, there are some differences in the irradiance distribution in certain areas, although our technique displays a great similarity with the ground truth render. The differences are due to, when performing the per voxel face irradiance gathering, some voxel faces coming across more clusters that contain lit voxels than other voxel faces. The bottom left part of Fig. 13 shows a false color heatmap of the number of lit voxels each voxel face computes, ranging from green (no lit voxels found) to red (225 or more lit voxels found). As can be seen, the areas matching the perceptible differences show a higher accumulation of lit voxels. The bottom right part of Fig. 13 shows, in false color, the error in the irradiance distribution between our technique and the ground truth render using NVIDIA's FLIP tool, which is an algorithm that emphasizes

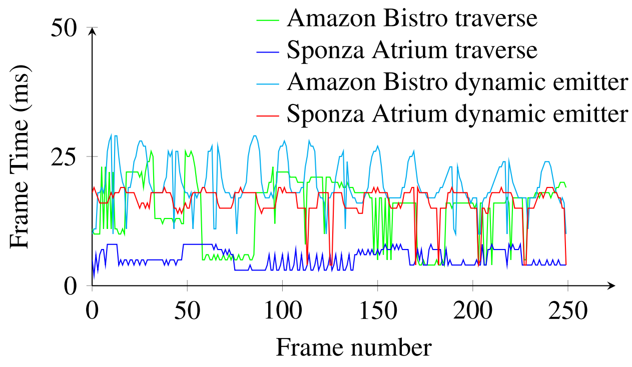


Fig. 15. Plot of the frame rate (ms) for Amazon Bistro with a scene camera traversal (green, mean time is 14.8 ms), Amazon Bistro with a dynamic emitter (cyan, mean time 19.8 ms), Sponza Atrium with a scene camera traversal (blue, mean time 5.3 ms) and Sponza Atrium with a dynamic emitter (red, mean time 16.3 ms). For the camera traversal plots, the emitter was modified before starting the traversal, triggering the computation of irradiance for each new visible voxel, which explains the spikes. In the dynamic emitter plots, the spikes correspond to emitter updates. We took the emitter update measurements with a voxelization size 128^3 and a camera similar to Fig. 6 for Sponza Atrium, and similar to Fig. 12 for Amazon Bistro. The frame times for dynamic emitter include the final frame pass.

perceptual differences. Fig. 16 shows how, as the voxelization resolution increases, the difference with ground truth decreases, as the increasing number of voxels allows us reproduce higher frequencies and a more detailed irradiance gathering in complex scenarios. For instance, in the figure we can see that the tables and chairs below the awnings on both sides of the scene show occlusion effects similar to the ground truth and not present in lower voxelization resolutions.

5.4. Comparison of real-time techniques

In Table 4 we show how voxel number and memory usage grow proportionally with the voxelization resolution, showing a dependency on the number of voxels and clusters but not on the scene geometry, since the Amazon Bistro scene contains almost 7 times more geometry than Sponza Atrium. Although any voxelization grows with n^3, with n the number of voxels on the side of the volume, the main source of non-empty voxels are the surfaces, which are the ones that actually exchange light, and grow as n^2. The number of initial clusters needed for the clustering step varies between 30 K and 250 K. After the clustering step, empty clusters are discarded. The number of final non-empty clusters ends up being of only a few thousands, i.e., for $n = 256$ the clustering is able to deal with more than half a million voxels and boil it down to less than 8.75 K clusters. Thus, the number of generated clusters is below 7.5% of the total number of voxels, for all resolutions and scenes tested.

Fig. 17 shows a comparison of memory usage for different voxelization sizes between our technique and the Sparse Voxel Octree used by Crassin et al. [10] for Voxel Cone Tracing, using the data provided by Crassin [59]. As can be seen, as the voxelization resolution increases, our technique shows a memory growth rate similar to the Sparse Voxel Octree. We show in Table 4 the final amount of memory used, with only the resources needed for the light bounce and final frame steps. It includes the per voxel face cluster visibility data, which represents between 37% and 53% of the final memory usage. Fig. 14 shows a set of Sponza Atrium scenes configured to be as similar as possible. As can be seen, our technique can generate color bleeding at voxelization size 128^3, where Voxel Cone Tracing (VCT) at the same voxelization size cannot, requiring a higher voxelization level (256^3), which requires more memory than our technique (77.9MB vs 167.6MB). In the same figure, we show how our technique at voxelization size 256^3 is able to offer better quality, with higher frequencies and more detailed irradiance gathering than VCT at voxelization resolution 512^3 using less than 40% of memory (313.6MB vs 827MB). Thanks to our compact data structures and to our strong irradiance gathering method, we can offer similar results to VCT with less memory and with a smaller level of voxelization. Also, when comparing with VCT we can see the effect of thin geometry in our work: VCT shows clear light leaks between the drapes and the wall, whereas in our case we can correctly approximate irradiance for most of the fragments thanks to storing irradiance per voxel face. We also offer captures of Light Propagation Volumes and NVIDIA's RTXGI, both done with Unreal Engine. In comparison, our technique is able to offer higher frequencies and a better sampling of the close and distant scene elements that contribute to the irradiance at a point, although those techniques support fully dynamic scenes and glossy materials (and specular materials as well in the case of RTXGI). The techniques compared in Fig. 14 order RTXGI as the least performing technique (108 FPS), followed by our work at voxelizations 256^3 (163 FPS) and 128^3 (183 FPS), Light Propagation Volumes (197 FPS), our work at voxelization 64^3 (199 FPS) and Voxel Cone Tracing at 512^3 (250 FPS). We do not have information about the performance of VCT at voxelization 128^3. In Fig. 18 we compare our work with the same techniques from Fig. 14, in a much simpler scene. We can see how in this case our work (496 FPS) has a much higher performance than Light Propagation Volumes (LPV) (276 FPS) and RTXGI (139 FPS), while showing a greater similarity with the ground truth. When compared with VCT, and since there is no access to the original implementation by Crassin et al. [10], we extrapolated the FPS values using the implementation by Kröker [60] with the same (manual) scene configurations as in Figs. 14 and 18. We estimate that the VCT implementation from Crassin et al. [10] would run at 350–400 FPS on this figure, below our work's performance (496 FPS). Another possible comparison with VCT to see the impact of our cluster approach could consist of using a hybrid approach. We could use the same information as in VCT (a Sparse Voxel Octree) but, instead of tracing cones in texture space selecting different mip map levels based on distance, tracing rays in texture until they get across an occupied voxel and sampling from the corresponding mip map level based on distance as well. In our estimations, this could allow a higher quality than VCT when using a considerable number of rays (probably 64 or more) thanks to the more precise sampling, but the associated cost of tracing those rays would be prohibitive, since in VCT the indirect lighting is computed per fragment, instead of using a view independent structure to store irradiance such as our voxels and clusters, which allows us to decouple it from the final frame.

Conclusions

We presented clustered voxel global illumination, a technique to compute diffuse indirect illumination with dynamic cameras and emitters based on clustering of a voxelized scene. We show an efficient irradiance gathering technique relying only on the generated clusters and requiring a very small amount of per voxel information. We introduced the concept of voxel clustering, a way to deal with the uncontrolled growth of voxels as resolution increases, keeping under control the number of final clusters. We also decoupled irradiance from the final frame computation, being computed only once for visible voxels if the emitter remains the same. As avenues for future work, we would like to find ways to accelerate irradiance gathering, e.g., discarding voxel faces that we can determine as not needed for irradiance approximation, handle dynamic scenes, and introduce glossy materials and participating media.

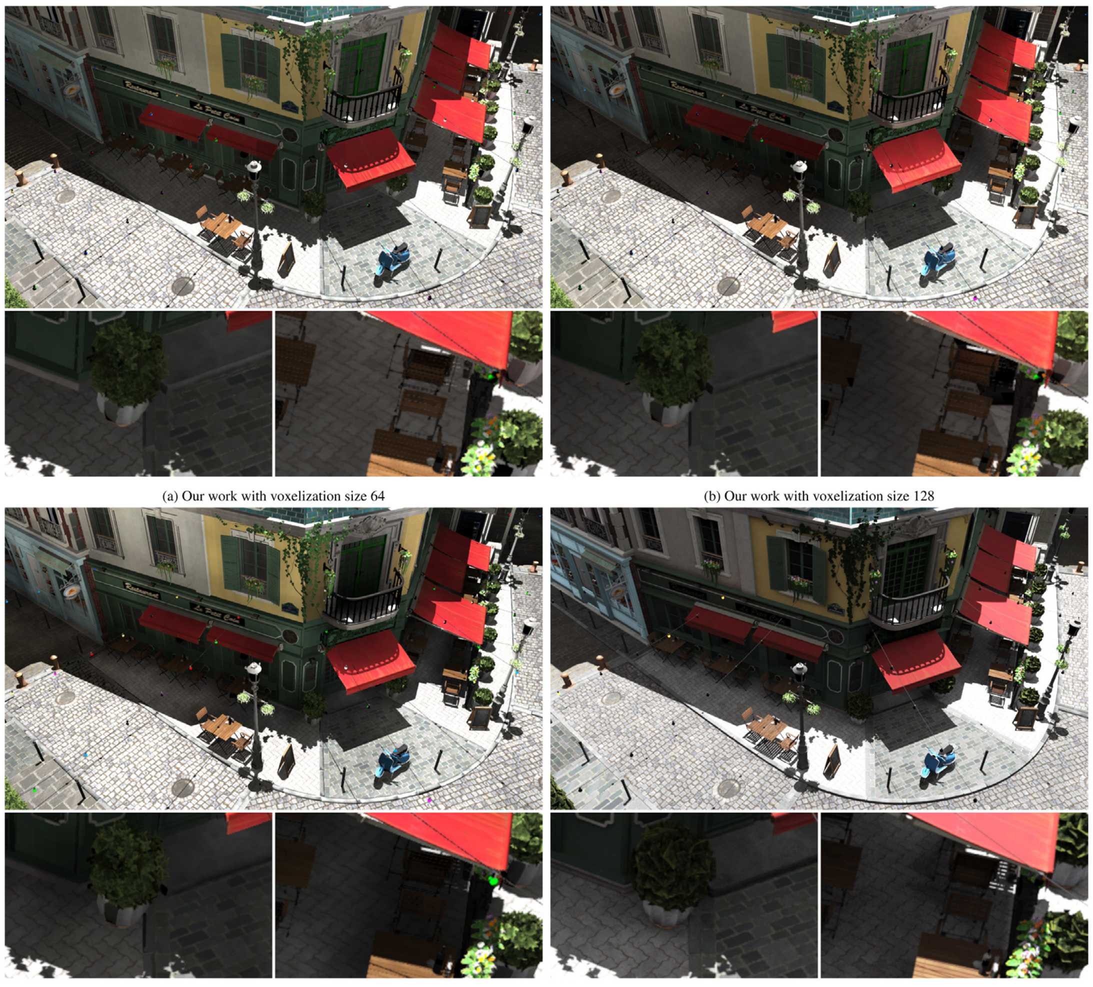

(a) Our work with voxelization size 64
(b) Our work with voxelization size 128
(c) Our work with voxelization size 256
(d) Ground truth with 1024 samples per pixel

Fig. 16. In (a), (b) and (c) we show how, as voxelization size increases, the quality offered by our work gets closer to the ground truth, rendered in (d) with Cycles renderer (Blender v2.83) with 1024 samples per pixel. The 256 voxelization size case shows great similarities with the ground truth render, reproducing occlusion effects that are not present in renders with lower voxelizations. This is especially visible around the chairs and tables (see detail images). The peak signal-to-noise ratio values for our work at voxelization sizes 64, 128 and 256 and the ground truth render are 18.172, 18.203 and 17.42 respectively.

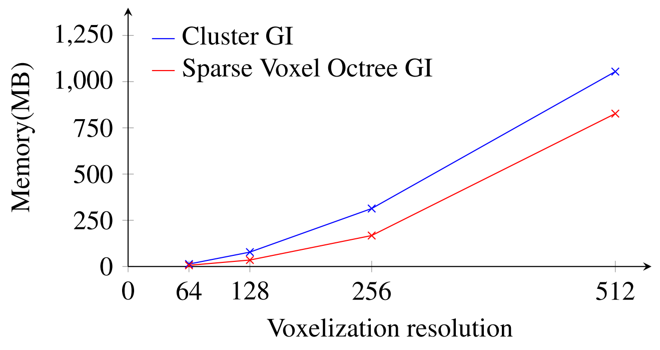


Fig. 17. Comparison of memory usage between our technique (blue) and VCT Sparse Voxel Octree implementation [59] (red). As the voxelization size increases, our memory usage grows at a similar rate while achieving results with quality similar to higher VCT Sparse Voxel Octree voxelization sizes. See Fig. 14.

CRediT authorship contribution statement

Alejandro Cosin Ayerbe: Methodology, Software, Investigation, Conceptualization, Writing – original draft, Writing – review & editing. **Gustavo Patow:** Formal analysis, Investigation, Supervision, Conceptualization, Writing – original draft, Writing – review & editing.

Code availabiliy

The code to reproduce the images in this paper is available on GitHub (https://github.com/AlejandroC1983/cvrtgi).

Acknowledgments

We would like to thank Cyril Crassin for all the help and information he provided during the revisions of this manuscript. This work was partially funded by the TIN2017-88515-C2-2-R from Ministerio de Ciencia, Innovación *y* Universidades, Spain.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cag.2022.01.005.

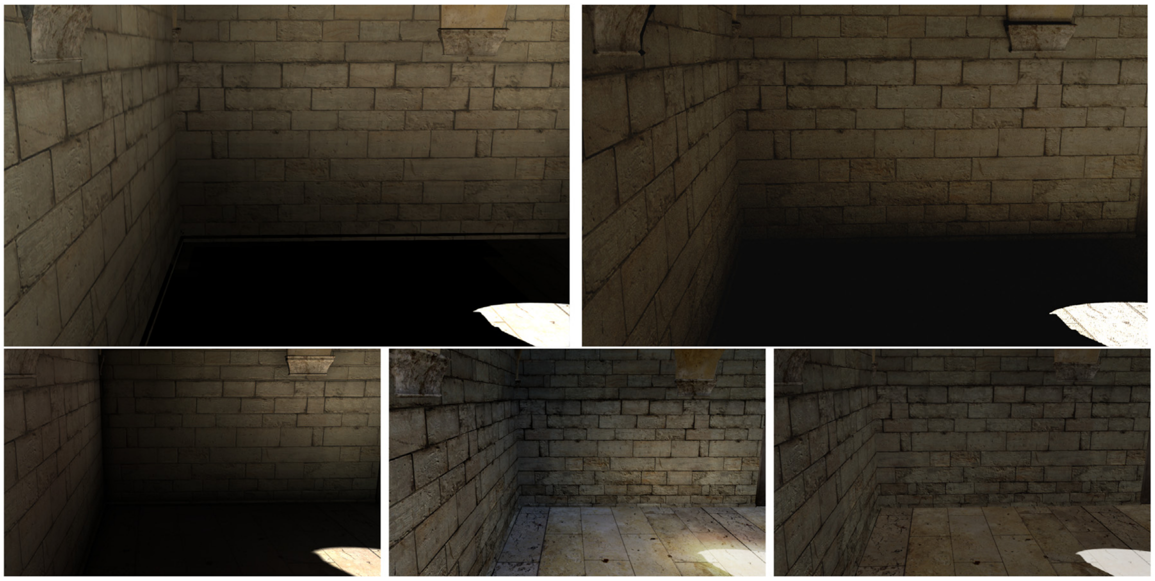


Fig. 18. Comparing methods. Top row: Our work at 496 FPS (left) and ground truth with LuxCoreRender with 1024 samples per pixel (right). Bottom row: Voxel Cone Tracing implementation from Kröker [60], Light Propagation Volumes at 276 FPS (center) and RTXGI at 139 FPS (right). Since we do not have access to the original Sparse Voxel Octree Voxel Cone Tracing implementation from Crassin et al. [10], the FPS value for Voxel Cone Tracing for this figure has been extrapolated, measuring the FPS from Kröker [60] with the same scene configurations as in Fig. 14 and in this figure. We estimate the original implementation from Crassin et al. [10] would run at about 350–400 FPS on this scene. Scenes have been manually configured to be as similar as possible.

References

[1] Dachsbacher C, Stamminger M. Reflective shadow maps. In: Proceedings of the 2005 symposium on interactive 3D graphics and games. I3D '05, New York, NY, USA: ACM; 2005, p. 203–31. http://dx.doi.org/10.1145/1053427.1053460, URL http://doi.acm.org/10.1145/1053427.1053460.

[2] Nichols G, Shopf J, Wyman C. Hierarchical image-space radiosity for interactive global illumination. Comput Graph Forum 2009;28:1141–9.

[3] Dachsbacher C, Stamminger M. Splatting indirect illumination. In: Proceedings of the 2006 symposium on interactive 3D graphics and games. New York, NY, United States: Association for Computing Machinery; 2006, p. 93–100.

[4] Wang R, Wang R, Zhou K, Pan M, Bao H. An efficient GPU-based approach for interactive global illumination. In: ACM SIGGRAPH 2009 papers. Oxford, UK: Wiley-Blackwell Publishing Ltd; 2009, p. 1–8.

[5] Kaplanyan A, Dachsbacher C. Cascaded light propagation volumes for real-time indirect illumination. In: Proceedings of the 2010 ACM SIGGRAPH symposium on interactive 3D graphics and games. I3D '10, New York, NY, USA: ACM; 2010, p. 99–107. http://dx.doi.org/10.1145/1730804.1730821, URL http://doi.acm.org/10.1145/1730804.1730821.

[6] Kajiya JT. The rendering equation. ACM SIGGRAPH Comput Grap 1986;20(4):143–50.

[7] Whitted T. An improved illumination model for shaded display. In: Proceedings Of the 6th annual conference on computer graphics and interactive techniques. New York, NY, United States: Association for Computing Machinery; 1979, p. 14.

[8] Koskela M, Lotvonen A, Mäkitalo M, Kivi P, Viitanen T, Jääskeläinen P. Foveated real-time path tracing in visual-polar space. In: Boubekeur T, Sen P, editors. Eurographics symposium on rendering - dl-only and industry track. Avenue de Frontenex 32, 1207 Geneva, Switzerland: The Eurographics Association; 2019, p. 367–74. http://dx.doi.org/10.2312/sr.20191219.

[9] Ritschel T, Grosch T, Kim MH, Seidel HP, Dachsbacher C, Kautz J. Imperfect shadow maps for efficient computation of indirect illumination. ACM Trans Graph 2008;27(5):129:1–8. http://dx.doi.org/10.1145/1409060.1409082, URL http://doi.acm.org/10.1145/1409060.1409082.

[10] Crassin C, Neyret F, Sainz M, Green S, Eisemann E. Interactive indirect illumination using voxel cone tracing: A preview. In: Symposium on interactive 3D graphics and games. I3D '11, New York, NY, USA: ACM; 2011, p. 207. http://dx.doi.org/10.1145/1944745.1944787, URL http://doi.acm.org/10.1145/1944745.1944787.

[11] Silvennoinen A, Lehtinen J. Real-time global illumination by precomputed local reconstruction from sparse radiance probes. ACM Trans Graph 2017;36(6):230:1–230:13. http://dx.doi.org/10.1145/3130800.3130852, URL http://doi.acm.org/10.1145/3130800.3130852.

[12] Thiedemann S, Henrich N, Grosch T, Müller S. Voxel-based global illumination. In: Symposium on interactive 3D graphics and games. I3D '11, New York, NY, USA: ACM; 2011, p. 103–10. http://dx.doi.org/10.1145/1944745.1944763, URL http://doi.acm.org/10.1145/1944745.1944763.

[13] Sugihara M, Rauwendaal R, Salvi M. Layered reflective shadow maps for voxel-based indirect illumination. In: Proceedings of high performance graphics. HPG '14, Goslar Germany, Germany: Eurographics Association; 2014, p. 117–25, URL http://dl.acm.org/citation.cfm?id=2980009.2980022.

[14] Bresenham JE. Algorithm for computer control of a digital plotter. IBM Syst J 1965;4(1):25–30.

[15] Keller A. Instant radiosity. In: Proceedings Of the 24th annual conference on computer graphics and interactive techniques. SIGGRAPH '97, New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.; 1997, p. 49–56. http://dx.doi.org/10.1145/258734.258769.

[16] Greger G, Shirley P, Hubbard PM, Greenberg DP. The irradiance volume. IEEE Comput Graph Appl 1998;18(2):32–43. http://dx.doi.org/10.1109/38.656788.

[17] Sloan P-P, Kautz J, Snyder J. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In: Proceedings of the 29th annual conference on computer graphics and interactive techniques. New York NY United States: Association for Computing Machinery; 2002, p. 527–36.

[18] Dachsbacher C, Stamminger M, Drettakis G, Durand F. Implicit visibility and antiradiance for interactive global illumination. ACM Trans Graph (TOG) 2007;26(3):61–es.

[19] Kaplanyan A, Dachsbacher C. Cascaded light propagation volumes for real-time indirect illumination. In: Proceedings of the 2010 ACM SIGGRAPH symposium on interactive 3D graphics and games. New York, NY, United States: Association for Computing Machinery; 2010, p. 99–107.

[20] Nalbach O, Ritschel T, Seidel H-P. Deep screen space. In: Proceedings of the 18th Meeting of the ACM SIGGRAPH symposium on interactive 3D graphics and games. 2014, p. 79–86.

[21] Mara M, McGuire M, Nowrouzezahrai D, Luebke DP. Deep g-buffers for stable global illumination approximation. In: High performance graphics. 2016, p. 87–98.

[22] Kol TR, Bauszat P, Lee S, Eisemann E. MegaViews: Scalable many-view rendering with concurrent scene-view hierarchy traversal. Comput Graph Forum 2018;38(1):235–47. http://dx.doi.org/10.1111/cgf.13527.

[23] Majercik Z, Guertin J-P, Nowrouzezahrai D, McGuire M. Dynamic diffuse global illumination with ray-traced irradiance fields. J Comput Graph Tech 2019;8(2).

[24] Currius RR, Dolonius D, Assarsson U, Sintorn E. Spherical Gaussian light-field textures for fast precomputed global illumination. Comput Graph Forum 2020;39:133–46.

[25] Wang SW, Kaufman AE. Volume sampled voxelization of geometric primitives. In: Proceedings visualization'93. Berlin, Heidelberg: Springer-Verlag,IEEE; 1993, p. 78–84.

[26] Beckhaus S, Wind J, Strothotte T. Hardware-based voxelization for 3d spatial analysis. In: Proceedings of the 5th international conference on

computer graphics and imaging. 20, New York, NY, United States: Association for Computing Machinery, Canmore, Alberta, Canada; 2002, p. 47–54.
[27] Zhang L, Chen W, Ebert DS, Peng Q. Conservative voxelization. Vis Comput 2007;23(9–11):783–92.
[28] Schwarz M, Seidel H-P. Fast parallel surface and solid voxelization on GPUs. ACM Trans Graph (TOG) 2010;29(6):1–10.
[29] Crassin C, Green S. Octree-based sparse voxelization using the GPU hardware rasterizer. In: Opengl insights. Patrick Cozzi and Christophe Riccio,Boca Ratón, Florida, USA: CRC Press; 2012, p. 303–19, http://www.seas.upenn.edu/~pcozzi/OpenGLInsights/OpenGLInsights-SparseVoxelization.pdf,Chapter.
[30] Heitz E, Neyret F. Representing appearance and pre-filtering subpixel data in sparse voxel octrees. In: High performance graphics 2012. Eurographics; 2012, p. 125–34.
[31] Vicini D, Jakob W, Kaplanyan A. A non-exponential transmittance model for volumetric scene representations. ACM Trans Graph (TOG) 2021;40(4):1–16.
[32] Jendersie J, Kuri D, Grosch T. Real-time global illumination using precomputed illuminance composition with chrominance compression. J Comput Graph Tech Vol 2016;5(4):8–35.
[33] Kontkanen J, Turquin E, Holzschuch N, Sillion FX. Wavelet radiance transport for interactive indirect lighting. In: Symposium on rendering. Avenue de Frontenex 32, 1207 Geneva, Switzerland: The Eurographics Association; 2006, p. 161–71.
[34] Chen Y-Y, Chien S-Y. Lighting-driven voxels for memory-efficient computation of indirect illumination. Vis Comput 2016;32(6–8):781–9. http://dx.doi.org/10.1007/s00371-016-1235-y.
[35] Papaioannou G. Real-time diffuse global illumination using radiance hints. In: Proceedings of the ACM SIGGRAPH symposium on high performance graphics. HPG '11, New York, NY, USA: ACM; 2011, p. 15–24. http://dx.doi.org/10.1145/2018323.2018326, URL http://doi.acm.org/10.1145/2018323.2018326.
[36] Lehtinen J, Zwicker M, Turquin E, Kontkanen J, Durand F, Sillion FX, et al. A meshless hierarchical representation for light transport. In: ACM SIGGRAPH 2008 papers. New York NY United States: ACM Transactions on Graphics; 2008, p. 1–9.
[37] Loos BJ, Nowrouzezahrai D, Jarosz W, Sloan P-P. Delta radiance transfer. In: Proceedings of the ACM SIGGRAPH symposium on interactive 3D graphics and games. New York NY United States: Association for Computing Machinery; 2012, p. 191–6.
[38] Smits B, Arvo J, Greenberg D. A clustering algorithm for radiosity in complex environments. In: Proceedings of the 21st annual conference on computer graphics and interactive techniques. New York NY United States: ACM Transactions on Graphics; 1994, p. 435–42.
[39] Willmott AJ, Heckbert PS, Garland M. Face cluster radiosity. In: Rendering techniques' 99. Vienna: Springer; 1999, p. 293–304.
[40] Prutkin R, Kaplanyan A, Dachsbacher C. Reflective shadow map clustering for real-time global illumination. In: Eurographics (short papers). Avenue de Frontenex 32, Geneva, Switzerland: Eurographics Association; 2012, p. 9–12.
[41] Takeshige M. In: Nvidia, editor. The basics of gpu voxelization. 2015, https://developer.nvidia.com/content/basics-gpu-voxelization.
[42] García I, Lefebvre S, Hornus S, Lasram A. Coherent parallel hashing. ACM Trans Graph (TOG) 2011;30(6):1–8.
[43] Alcantara DA, Sharf A, Abbasinejad F, Sengupta S, Mitzenmacher M, Owens JD, et al. Real-time parallel hashing on the GPU. In: ACM SIGGRAPH asia 2009 papers. New York, NY, United States: Association for Computing Machinery; 2009, p. 1–9.
[44] Bruce M. Prefix sums on GPUs. GPGPU2 workshop, University of Cape Town; 2014, URL http://gpu.cs.uct.ac.za/Slides/prefix-sum.pdf.
[45] Merrill Duane GM. Single-pass parallel prefix scan with decoupled look-back. Tech. rep, NVIDIA Corporation; 2016, URL https://research.nvidia.com/sites/default/files/pubs/2016-03_Single-pass-Parallel-Prefix/nvr-2016-002.pdf.
[46] Gelfand N, Guibas LJ. Shape segmentation using local slippage analysis. In: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on geometry processing. Computer Graphics Laboratory, Stanford University: Eurographics Symposium on Geometry Processing, ACM; 2004, p. 214–23.
[47] Wang J, Yu Z. Surface feature based mesh segmentation. Comput Graph 2011;35(3):661–7.
[48] Yan D-M, Wang W, Liu Y, Yang Z. Variational mesh segmentation via quadric surface fitting. Comput Aided Des 2012;44(11):1072–82.
[49] Lee J, Kim S, Kim S-J. Mesh segmentation based on curvatures using the GPU. Multimedia Tools Appl 2015;74(10):3401–12.
[50] Comaniciu D, Meer P. Mean shift: A robust approach toward feature space analysis. IEEE Trans Pattern Anal Mach Intell 2002;24(5):603–19.
[51] Vedaldi A, Soatto S. Quick shift and kernel methods for mode seeking. In: European conference on computer vision. European Conference on Computer Vision, Springer, University of California, Los Angeles, Computer Science Department; 2008, p. 705–18.
[52] Achanta R, Shaji A, Smith K, Lucchi A, Fua P, Süsstrunk S. Slic superpixels. Tech. rep, School of Computer and Communication Sciences, École Polytechnique Fédrale de Lausanne; 2010.
[53] Andersson P, Nilsson J, Akenine-Möller T, Oskarsson M, Åström K, Fairchild MD. FLIP: A difference evaluator for alternating images. Proc ACM Comput Graph Interact Tech 2020;3(2).
[54] Tabellion E, Lamorlette A. An approximate global illumination system for computer generated films. ACM Trans Graph 2004;23(3):469.
[55] Lottes T. FXAA: Fast approximate anti-aliasing. Tech. rep, NVIDIA Corporation; 2011, URL http://developer.download.nvidia.com/assets/gamedev/files/sdk/11/FXAA_WhitePaper.pdf.
[56] Willems S. In: Khronos, editor. Compute culling. 2015, https://github.com/SaschaWillems/Vulkan/blob/master/examples/computecullandlod/computecullandlod.cpp.
[57] Hudson T, Manocha D, Cohen J, Lin M, Hoff K, Zhang H. Accelerated occlusion culling using shadow frusta. In: Proceedings of the thirteenth annual symposium on computational geometry. 1997, p. 1–10.
[58] Vanek J, Navrátil J, Herout A, Zemčík P. High-quality shadows with improved paraboloid mapping. In: International symposium on visual computing. Springer; 2011, p. 421–30.
[59] Crassin C. Personal communication. 2021.
[60] Kröker W. Voxel cone tracing GI. 2021, GitHub Repository, GitHub, https://github.com/compix/VoxelConeTracingGI.
[61] LuxCoreRender team. LuxCoreRender. 2020, URL https://luxcorerender.org/.