Computers & Graphics 108 (2022) 74–85

Contents lists available at ScienceDirect

Computers & Graphics

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

Special Section on RAGI

Interactive VPL-based global illumination on the GPU using fuzzy clustering

Arnau Colom[a,*,1], Ricardo Marques[b,2], Luís Paulo Santos[c,d,3]

[a] *Departament de Tecnologies de la Informació i les Comunicacions, Universitat Pompeu Fabra, Barcelona, Spain*
[b] *Departament de Matemàtiques i Informàtica, Universitat de Barcelona, Barcelona, Spain*
[c] *Departamento de Informática, Universidade do Minho, Braga, Portugal*
[d] *HASLab, INESC TEC, Portugal*

ARTICLE INFO

Article history:
Received 18 July 2022
Received in revised form 7 September 2022
Accepted 18 September 2022
Available online 24 September 2022

Keywords:
Ray tracing
Global illumination
Interactive rendering
GPU

ABSTRACT

Physically-based synthesis of high quality imagery, including global illumination light transport phenomena, results in a significant workload, which makes interactive rendering a very challenging task. We propose a VPL-based ray tracing approach that runs entirely in the GPU and achieves interactive frame rates while handling global illumination light transport phenomena. This approach is based on clustering both shading points and VPLs and computing visibility only among clusters' representatives. A new massively parallel K-means clustering algorithm, enables efficient execution in the GPU. Rendering artifacts, that could result from the piecewise constant approximation of the VPLs/shading points visibility function introduced by the clustering, are smoothed away by resorting to an innovative approach based on fuzzy clustering and weighted interpolation of the visibility function. The effectiveness of the proposed approach is experimentally verified for a collection of scenes, with frame rates larger than 3 fps and up to 25 fps being demonstrated.

1. Introduction

Physically-based rendering of high quality imagery, including global illumination light transport phenomena, results in a significant workload, which more often than not implies long rendering times. Over the last decade, however, many developments have been proposed both at the hardware and algorithmic levels, which in many cases allow for interactive frame rates. Achieving interactivity requires clever algorithmic exploitation of some characteristics of radiance distribution within a typical scene, such as, for example, the local smoothness of indirect lighting. These allow for faster estimations of the light transport integral, hopefully without a perceptually relevant impact on the rendered images.

This work's main goal is to enable global illumination at interactive frame rates, therefore contributing to a more thorough understanding of how some simplifications can be introduced in the estimation of the light transport integral and how these impact on both performance and perceived image quality. Specifically, we address scenes with static geometry/materials and lighting conditions; only the viewpoint is allowed to change, thus supporting interactive walkthroughs (we will show when analyzing experimental results that fully dynamic scenes can be easily supported). The presented solution only supports diffuse materials. However, as discussed in the last section, extending the method to handle glossy materials is feasible.

Enabling global illumination at interactive frame rates has driven the main decisions made when designing the proposed approach. The global illumination requirement led us to select ray tracing as the visibility evaluation algorithm, since adjusting sampling rates and integrating secondary light transport phenomena is straightforward. The interactivity requirement led us to establish as a secondary goal that the entire algorithm runs on a GPU, completely avoiding costly memory transfers between the devices. We use NVidia's OptiX to trace all primary and secondary rays [1].

An efficient use of the GPU's execution model requires the algorithm to follow a quite uniform control flow among a massive number of threads and to exhibit coherent memory accesses. This regularity can only be attained, up to a certain degree, if the trees of rays are traversed breadth first, rather than depth-first as on a typical canonical ray traced based rendering algorithm. The proposed algorithm is thus organized on multiple passes, each corresponding to a very specific task. One such pass entails finding which scene points are relevant to the current view point,

* Corresponding author.
E-mail addresses: arnau.colom@upf.edu (A. Colom), ricardo.marques@ub.edu (R. Marques), psantos@di.uminho.pt (L.P. Santos).
1 Research Assistant.
2 Serra Húnter Fellow.
3 Assistant Professor.

https://doi.org/10.1016/j.cag.2022.09.008

as determined by primary or specular rays shot from the camera; these scene points are referred to as shading points (SP).

Additionally, global illumination requires simulation of indirect lighting. Evaluating this by resorting to extensive hemisphere sampling implies shooting an unsustainable number of incoherent rays, which would compromise the interactivity requirement for all but the simplest scenes and lighting conditions. Using an algorithm based on VPLs was thus an early design decision. With respect to hemispherical sampling, using VPLs has two advantages: (i) separating the tracing of the VPLs from the visibility evaluation stage of the rendering algorithm allows for a much more regular workload, and (ii) the re-utilization of the light paths captured by the VPLs across all shading points dramatically reduces the number of visibility rays to be shot.

Notwithstanding, tracing shadow rays between all SPs and all VPLs still implies a huge workload. As demonstrated by some of the results described in Section 2.2, indirect lighting' smoothness allows for approximated visibility estimation. We leverage on these results and interactively cluster both SPs and VPLs, using a massively parallel K-means clustering algorithm running in the GPU. Shadow rays are traced only between SP clusters' representatives and VPL clusters' representatives. Indirect lighting for each SP is evaluated using the visibility results for the corresponding SP cluster representative; geometric and BRDF data, however, will correspond to the actual shading point and the appropriate VPL cluster representative.

Clustering as described above corresponds to a piecewise constant approximation of the VPLs/shading points visibility function. This approximation introduces highly perceivable artifacts, as shown in the following sections. These artifacts are perceptually more relevant in regions of the image with intricate indirect shadows, due to the fact that global and local lighting are not explicitly handled in different manners by the proposed algorithm. These artifacts are smoothed away by allowing each SP to belong to F clusters, rather than a single one. On the rendering pass, and for each SP, a VPL cluster representative visibility is evaluated by resorting to weighted interpolation among all the clusters the SP belongs to. Fuzzy clustering and weighted interpolation are efficiently implemented in the GPU allowing for interactive frame rates.

This paper's main contributions are as follows:

- rendering approach that allows interactive VPL-based global illumination with ray tracing in the GPU;
- massively parallel interactive fuzzy K-Means algorithm, which exploits 3D neighborhood information in the form of a regular grid to speedup searching for clusters' centroids;
- efficient handling of soft shadowing artifacts by resorting to fuzzy clustering and weighted visibility interpolation, rather than explicitly handling separately global and local lighting on a GPU non-efficient manner.

The remaining of this article is structured as follows: in Section 2 we provide the necessary background, present the state of the art and position our contribution with respect to the existing works. In Section 3 we present the details of our approach. Follows a thorough set of results (Section 4), and finally we present our conclusions and future work in Section 5.

2. Background

2.1. Rendering equation and virtual point lights

In the following, we provide the basic theoretical background on VPLs-based rendering necessary for the understanding of our proposed approach. For a more detailed explanation please refer to [2]. The *rendering equation* [3] formulates the light transport across a scene. It states that the outgoing radiance L_o at a given surface point p towards the direction ω_o is given by:

$$L_o(p, \omega_o) = L_e(p, \omega_o) + L_r(p, \omega_o)\,, \tag{1}$$

where $L_e(p, \omega_o)$ is the self-emitted radiance from point p towards the direction ω_o, and $L_r(p, \omega_o)$ is the light reflected at point p towards ω_o. L_r can be further developed as follows:

$$L_r(p, \omega_o) = \int_{\Omega_+} L_i(p, \omega_i)\, \rho(p, \omega_i, \omega_o) \cos(n_p, \omega_i)\, \mathrm{d}\omega_i\,, \tag{2}$$

with Ω_+ representing the unit hemisphere around the normal n_p at the surface point p, $L_i(p, \omega_i)$ being the incident radiance at p from the direction ω_i, $\rho(x, \omega_i, \omega_o)$ being the *bi-directional reflectance distribution function* (BRDF) and $\cos(n_p, \omega_i)$ is the cosine of the angle between n_p and ω_i.

Eq. (2) is defined as an integral over the hemisphere. Using the change of integration variable given by:

$$\mathrm{d}\omega_i = \frac{\cos(n_{p'}, -\omega_i)}{\|p' - p\|^2}\mathrm{d}p'\,,$$

where p' is a surface point with normal $n_{p'}$, we can re-define Eq. (2) as an integral over the set A of all the scene surface points p' yielding:

$$L_r(p, \omega_o) = \int_A L_i(p, \omega_i)\, \rho(p, \omega_i, \omega_o) G(p, p') V(p, p')\, \mathrm{d}p'\,, \tag{3}$$

where V is the visibility function, and G is the geometry term defined as:

$$G(p, p') = \frac{\cos(n_p, \omega_i) \cos(n_{p'}, -\omega_i)}{\|p' - p\|^2}\,. \tag{4}$$

In general, both the hemispherical and the area formulations of the rendering equation (Eqs. (2) and (3), respectively) have no analytic solution. Therefore, its value is typically computed using numerical approximations. The class of *many light rendering* methods, based on the seminal paper of Keller [4], tackles the problem of providing a solution to the rendering equation by resorting to *virtual point lights* (VPLs). The VPLs are generated by stochastically tracing particles from the light sources towards the virtual scene. At each intersection point, a new VPL is created, capturing the reflected light at that intersection point towards the scene. Using the resulting VPLs, the reflected radiance at a given point p of the scene is then approximated as:

$$\hat{L}_r(p, \omega_o) = \sum_{j=1}^{N} I_j(p)\, \rho(p, \omega_i, \omega_o) G(p, p_j) V(p, p_j)\,, \tag{5}$$

where N is the number of used VPLs, $I_j(p)$ is the contribution of the jth VPL to the illumination at point p and p_j is the position of the jth VPL.

2.2. Related work

Indirect lighting is typically locally smooth, in particular over diffuse surfaces. VPL-based rendering solutions have been accelerated by leveraging local smoothness. In general, these approaches cluster VPLs and/or shading points using some similarity metric. Visibility evaluation between shading points and VPLs is then performed using each cluster representative point, rather than the original data. This results in a drastic reduction on the number of computations and, consequently, on rendering time. In the following, we provide an overview of the most representative approaches. For more details, please refer to [2].

Lightcuts [5] were the first proposal to exploit VPLs hierarchical clustering according to some similarity metric, therefore allowing for rendering times that scale sub-linearly with the number of VPLs. The tree root node consists in a single cluster grouping all VPLs and the leaves are the individual VPLs themselves. Inner nodes represent different levels of clustering. Lighting local adaptation is achieved by generating for each SP a cut of the tree graph, based upon an estimate of the rendering error. Visibility is then evaluated between the SP and the representative VPLs for the clusters lying along the cut. With multidimensional lightcuts [6] this proposal is extended to a bidirectional approach, where a hierarchy of pairs of VPLs and SPs is built. Cuts are then generated at rendering time, resulting in an implicit clustering of VPLs and SPs. By clustering bidirectionally greater efficiency and scalability is achieved than with lightcuts. These methods did not target GPUs and are far from interactive due, to a great extent, to the generation of different cuts per pixel and to the sophisticated perceptual criteria used to both build the hierarchy and generate the cuts.

Dong et al. [7] proposed a GPU approach based on clustering the VPLs and aiming towards interactive frame rates. VPLs are clustered using K-means and each cluster is used to create a virtual area light (VAL). Visibility between shading points and VALs is evaluated using GPU-efficient Convolution Soft Shadow Maps (CSSM). Although interactive frame rates are achieved, the method's efficiency is based on the utilization of low resolution CSSMs, which means that thin indirect shadows suffer from aliasing and contact shadows are not well resolved.

Hašan et al. [8] formulate the VPL lighting problem as a matrix, with rows corresponding to individual shading points and columns corresponding to different VPLs. Computing indirect light for a given shading point amounts to evaluate all elements of the corresponding row and then summing across the row. The Matrix Row–Column Sampling (MRCS) algorithm clusters both shading points and VPLs. Rows are stochastically selected using stratified uniform sampling. VPLs are then clustered, on the CPU, using information from the reduced set of shading points. Visibility among the clustered shading points and VPLs is evaluated in the GPU using shadow maps. This is a hybrid (CPU+GPU) non-interactive algorithm, requiring several costly data transfers between the devices.

LightSlice [9] is a non-interactive CPU-oriented algorithm, which builds upon MRCS [8], but significantly improves on it by considering both global and locally relevant VPLs. SPs and VPLs are independently clustered, the later capturing globally relevant lighting information. For each SPs' cluster, the global VPLs clustering is further refined, capturing locally relevant lighting information. Visibility is evaluated by ray tracing only between the shading point clusters' representatives and the associated local VPL clusters.

Davidovič et al. [10] also build on [8] and handle global and local lighting in two distinct, additive, steps. Global lighting is computed similarly to MRCS, but resorting to a more complex clustering algorithm and similarity metric. Local lighting is based upon the generation of a new, view dependent, set of local lights per tile of shading points. For each tile a number of rays are traced into the scene and the tile's local lights are created at the respective hit points, which are connected to the global lights using importance sampling. The tile's shading points are shaded using this set of local lights, assuming full visibility. This locally lit image is then composited with the global lighting using a weighted sum.

Jarabo et al. [11] propose a fully bidirectional hierarchical clustering of both the M SPs and the N VPLs. Two initial hierarchies, for the SPs and the VPLs, are initially built and combined at rendering time. Inspired on hierarchical radiosity methods, a cut is generated from the bidirectional hierarchy and the final image is rendered from this cut. The integration of the clustering along the two dimensions allows for $\mathcal{O}(\log(M+N))$ rendering times.

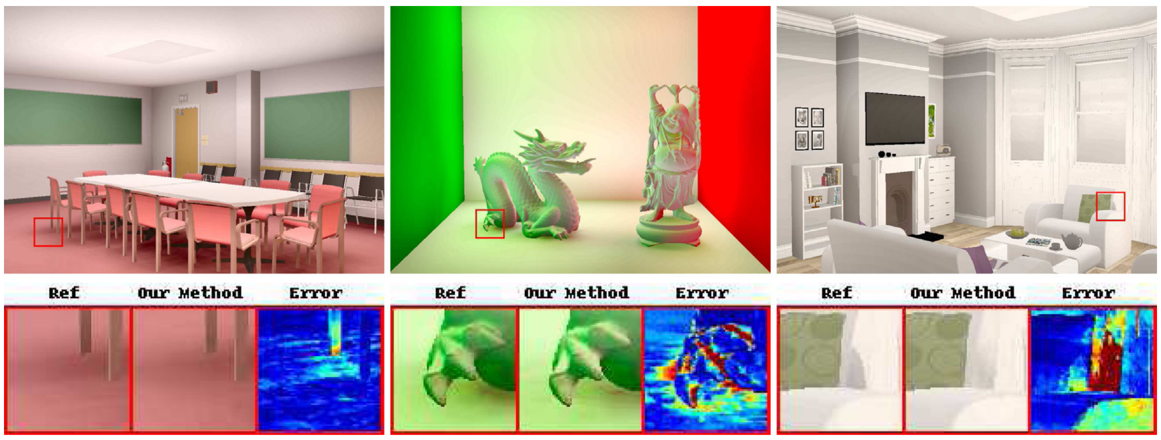
Daqi et al. [12] use a lighting grid hierarchy (LGH), built in real time in the GPU, to approximate lighting from many VPLs. The contribution of each VPL is distributed to the eight vertices of the voxel containing it. A *grid light* is then generated for each vertex with non-zero power and placed at the geometrical center of the VPLs contributing to it. The *grid lights* are rasterized as coarse spheres to approximate their non-shadowed contribution to each SP. The approximated contribution used to build a probability distribution over the *grid lights* for each SP. k *grid lights* are then stochastically selected, per SP, and their visibility is evaluated by shooting shadow rays. The average the visibility of these k samples is used as a shadow ratio estimator per SP. These estimators are filtered in screen-space to reduce high-frequency noise and used for shading against the previously evaluated non-shadowed contributions. Interactive frame rates are demonstrated with this hybrid rasterization/ray tracing approach.

Tatzgern et al. [13] propose using Stochastic Substitution Trees (SST), an hierarchical approach inspired by lightcuts, but with the lighting associated with inner nodes approximated by a normal distribution. Clusters' representatives within a cut are then stochastically sampled, trading structured artifacts by high frequency noise. The proposed method runs entirely in the GPU, using an efficient parallel algorithm to build the SST and leveraging the dedicated ray tracing modules for VPL tracing and SST sampling. A rasterization pass generates first hit from the camera. Additionally, recurrent autoencoders are used for denoising and temporal filtering; these run in the GPU's tensor cores, further leveraging modern graphics devices computing capabilities.

Our proposal is related to the previously described methods, in that it clusters both SPs and VPLs. Visibility is evaluated between SP clusters' representatives and VPL clusters' representatives, rather than between all SPs and all VPLs, resulting in a huge workload reduction. Shading, on the other hand, is evaluated per SP with respect to each VPL cluster representative (Eq. (6)). It runs entirely in the GPU, including all ray tracing and clustering steps, avoiding costly memory transfers between devices and enabling interactive frame rates. Artifacts due to local lighting are handled in a GPU efficient manner, by resorting to fuzzy clustering. Our method is entirely ray-tracing based, contrarily to some of the above hybrid methods [7,12,13].

Wang et al. [14] originally inspired our approach with their proposal to use interactive K-means clustering in the GPU. Shading points are clustered in the context of a caching/interpolation method similar to the irradiance caching [15], but applied to the final gathering stage of photon mapping. Final gathering is evaluated by hemispherical sampling at the centroids, and irradiance is interpolated for the remaining points.

Cuomo et al. [16] present a thorough analysis of an hybrid CPU–GPU implementation of the K-means algorithm using CUDA. They distinguish two stages that are iterated repeatedly in a loop until convergence: the *labeling* stage, where data points are assigned to clusters and the *reduction* stage, where new centroids are computed. The former stage runs entirely in the GPU, whereas the latter runs on the CPU; this requires, for each iteration, expensive data transfers between the CPU and GPU address spaces. For N data points, each with d attributes, a total of $N \times d$ CUDA threads are created. Each computes, per attribute, the square difference between a data point and a centroid. The squared Euclidean distance is then computed in parallel per data point using a parallel reduction algorithm. This process is repeated K times, once per centroid, and data points are assigned to the centroid at minimum distance. The parallel square differences evaluation and parallel reduction for summing only achieve interesting speedups for datasets with large dimensionality, such as $d = 128$. Bhimani et al. [17] assess a CUDA K-means implementation, which also performs the *reduction* stage on the CPU,



(a) 2K VPL clusters and ≈ 8K shading clusters (b) 1K VPL clusters and ≈ 20K shading clusters (c) 4K VPL clusters and ≈ 8K shading clusters

Fig. 1. Results when clustering both VPLs and shading points. The top row shows the reference images. The results have been generated with three different configurations and scenes, trying to find the best trade-off for each one of them: (a) The Conference Room configuration, with 2k VPL clusters and 8k shading clusters; (b) The Cornell box configuration, with 1k VPL clusters and 20k shading clusters; and (c), The Living Room configuration, with 4k VPL clusters and 8k shading clusters. The rendering times for (a), (b) and (c) were 106 ms (MSE = 1.80E−3), 162 ms (MSE = 1.10E−3) and 206 ms (MSE = 8.13E−4), respectively. The reference images required 1715 ms, 2348 ms and 1195 ms respectively.

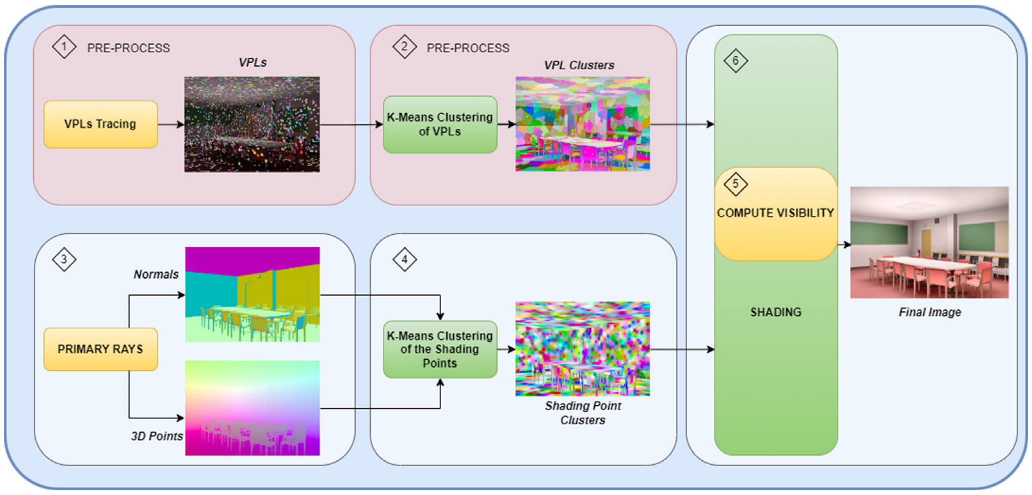


Fig. 2. A diagram showing our algorithm workflow. The green boxes indicate the part of the algorithm in which rays are traced whereas the yellow ones point out the steps that are computed in CUDA.

thus implying the same data transfer overheads. Their algorithm does not evaluate the distance metric in parallel across the data dimensions, thus they use N threads, i.e., as many as the number of data points. They do compare the CUDA version with OpenMP and MPI implementations.

We cluster SPs and VPLs, each with 6 geometric attributes (3D position and normal), therefore dispensing with the parallel decomposition across attributes. Our K-means approach (Section 3.3) runs entirely in the GPU, requiring no data transfers between the CPU and the GPU. The *reduction* stage provides a reduced degree of parallelism, since updates to the same centroid require using some access control mechanism, partially sequencializing the workload execution. We use hardware efficient atomic adds, thus minimizing access control overheads. A regular grid over the centroids, built in the GPU, is used to avoid computing distances between each data point and all K centroids.

3. Proposed approach

This section provides a detailed description of the proposed approach. We start by giving an overview of the rendering algorithm in Section 3.1. Section 3.2 presents the most relevant theoretical aspects of our approach to clustering SPs and VPLs. Then, in Section 3.3, we present the details of our GPU-based massively parallel implementation.

3.1. Algorithm description

As shown in Fig. 2, our proposed algorithm can be divided in six main steps:

1. VPLs tracing and creation;
2. VPLs clustering;
3. Generation of the shading points by tracing paths from the camera;
4. Shading points clustering;
5. Visibility evaluation between representatives of VPLs and shading points' clusters;
6. Rendering the final image's indirect lighting using the coarse visibility data.

Rendering algorithm. In Step 1 the VPLs are stochastically generated by shooting light particles from the light sources. Once the VPLs are scattered over the scene, we cluster them based on their

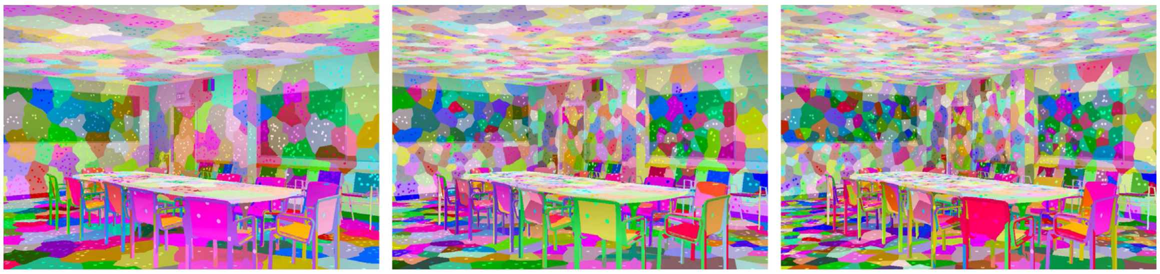

(a) 0.75K VPL clusters (b) 2K VPL clusters (c) 4K VPL clusters

Fig. 3. Visualization of the result of clustering 8k VPLs with a varying numbers of VPL clusters. Note that, as the number of cluster increases, the clusters become smaller thus allowing for more detailed information.

spatial similarities. Fig. 2 depicts these as two pre-processing stages since they are performed only once per scene; scene geometry/lighting and materials are not allowed to change, thus VPLs and the respective clustering can be reused across frames. Each resulting VPL cluster thus approximates the contribution of all its VPLs. In Step 3 we shoot one primary ray per pixel to collect the 3D position and the normal of each shading point. These shading points are then clustered in Step 4. In Step 5 we compute the visibility of each VPL cluster as seen from each shading point cluster. This is achieved by launching shadow rays from each shading points' cluster representative towards the representative of each cluster of VPLs. Finally, the image is rendered in Step 6 using the information produced in the previous steps.

Shading. To render the final image in Step 6 of Fig. 2 we compute the reflected radiance at each shading point p towards its corresponding pixel direction ω_o using the following equation:

$$\tilde{L}_r(p, \omega_o) = \sum_{c \in C_{VPL}} I_c(p)\, \rho(p, \omega_i, \omega_o) G(p, p_c) \tilde{V}(p, p_c; C_{SP}), \tag{6}$$

where c iterates over the set C_{VPL} of all VPLs clusters, $I_c(p)$ is the contribution of the VPL cluster c to the illumination at point p, p_c is the position of the representative VPL of cluster c, and $\tilde{V}$ is an approximated visibility function between p and p_c which resorts to the visibility information stored in the clusters of shading points C_{SP}. Using Eq. (6), the most computationally-intensive task (i.e., the explicit computation of VPLs visibility, which implies shooting shadow rays to evaluate V) is only performed per each cluster of shading points, while $\tilde{V}$, I_c, ρ, and the geometric term G are evaluated per shading point. As demonstrated by our results, this approach allows an effective trade-off between efficiency and image quality.

3.2. Theoretical aspects

3.2.1. Hard clustering

In order to perform clustering of both VPLs and shading points, a Distance Metric that quantifies similarities between different data points is required. Given a general surface data point $d = (p, n)$ where $p \in \mathbb{R}^3$ represents the 3D location of the data point, and n the normal at the surface where p lies, the distance between two data points d_1 and d_2 can be defined as:

$$\text{dist}(d_1, d_2) = \begin{cases} |p_1 - p_2|, & \text{if } n_1 \cdot n_2 > n_t \\ \infty, & \text{otherwise}, \end{cases} \tag{7}$$

where n_t is an orientation threshold below which $\text{dist}(d_1, d_2)$ is forced to be ∞. This threshold forces two data points that have an angle between their respective normals larger than n_t to be distributed into different clusters. This is an interesting feature for our application case since it is well known that illumination between two surfaces is highly dependent on their mutual orientation. In our experiments, we have empirically found that a threshold of $\sqrt{3}/2$ (corresponding to the cosine of a maximum angle of $\pi/6$) leads to good results.

Let us now consider we are provided with a dataset D of M data points, such that $D = \{d_m | m = 1 \dots M\}$, and with the number of clusters to generate (K). Then, the first step of the K-means algorithm is to select a set $C = \{c_k | k = 1, \dots, K\}$ of K centroids as initial cluster centers. This selection is often random, but it can also leverage some structure existing in the data. Follows an iterative phase with two steps: (i) the data points $d_m \in D$ are assigned to the nearest centroid $c_k \in C$ using Eq. (7); (ii) the set C of K centroids is updated by re-computing the centroids c_k based on the attributes of the data points d_m assigned to each cluster. Typically, this process is repeated until a maximum number of iterations is reached or until C remains unchanged in two consecutive iterations. At the end of the process, K-means yields a set of centroids C as well as information regarding to which centroid c_k each data point d_m belongs. In this work, we will assume this information is represented by a vector A of size M mapping each point $d_m \in D$ to its respective cluster index $k \in \{1, \dots, K\}$.

Fig. 3 shows the results of clustering the VPLs for the Conference Room scene. As expected, the size of the VPL clusters is reduced as the number of clusters increases. Note however that the cluster size reduction is not uniform across the scene. Indeed, areas of the image for which the VPLs' features change rapidly (such as, for example, the farthest wall) tend to have smaller cluster sizes, whereas other areas with smoother geometric variations (such as, for example, the back of the chairs) generally keep a larger cluster size.

Fig. 4(a) shows the results of clustering the shading points for the Conference Room scene. It is interesting to note that, due to the used Distance Metric, the resulting clusters do not group together data with significantly different normals or 3D positions. This is a desired feature for our rendering algorithm, since it is well known that illumination conditions might vary drastically with the normal and 3D position of shading points. However, as shown in Fig. 5(a), directly using these clusters for rendering the final image can have negative effects in the image quality. This is because, for each shading point, the used visibility is that of the corresponding cluster representative and, for neighboring shading points belonging to different clusters, the border between clusters becomes apparent. This problem is efficiently mitigated using *distance-based fuzzy clustering*, as proposed in the next section.

3.2.2. Distance-based fuzzy clustering

In general, clustering techniques can be classified as *hard* or *fuzzy* (aka soft) clustering algorithms. The former are characterized by the fact that each data point can only belong to a single

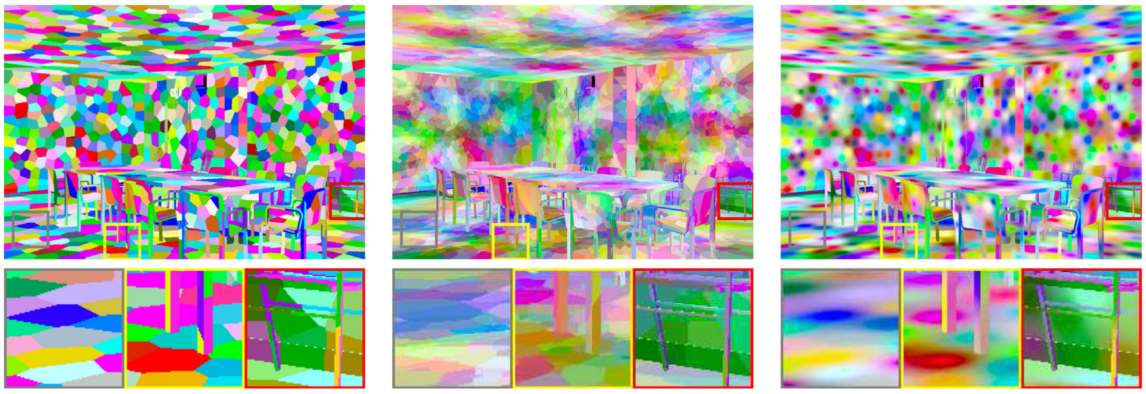

(a) Hard clustering (b) Fuzzy clustering (c) Fuzzy weighted clustering

Fig. 4. Visualization of the different tested clustering techniques when clustering the shading points (3D spatial clustering). Note that the border between neighboring clusters is progressively faded from left to right. The corresponding rendering results are shown in Fig. 5.

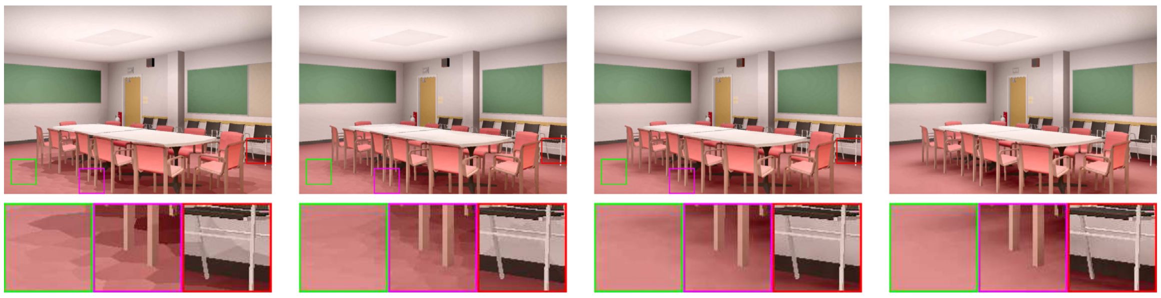

(a) Hard clustering (b) Fuzzy clustering (c) Fuzzy weighted clustering (d) Reference

Fig. 5. The effect of hard clustering (a), fuzzy clustering (b) and distance-based fuzzy clustering (c) on the final rendered image. Comparing (a) and (b) allows assessing the improvement brought by fuzzy clustering with respect to the typical hard clustering. The comparison between (b) and (c) shows the improvement brought by distance-based weighting of the nearest clusters.

cluster, while the latter allow data points to belong to more than one cluster. The degree of cluster fuzziness can be controlled by a *fuzziness parameter* F that determines to how many clusters a given data point can belong. Using this approach, the vector A representing the map between each data point d_m and the cluster c_k it belongs to (see Section 3.2.1) becomes a matrix with size $M \times F$, with M being the number of data points in the dataset D of points to be clustered. Each row of the A matrix can thus be interpreted as a vector I_m containing the indexes of the F cluster indexes $k \in [1, \dots, K]$ to which each data point d_m belongs.

When using soft clustering, the approximated visibility $\tilde{V}$ between a shading point p and a VPL (or a VPL cluster representative) p_c is given by the average visibility between p_c and the representative points p_f of all the F clusters to which p belongs, yielding:

$$\tilde{V}(p, p_c; C_{SP}) = \frac{1}{F} \sum_{f=1}^{F} V(p_f, p_c)\,, \tag{8}$$

where C_{SP} is the set of shading point clusters containing the F closest clusters to p. Using $F > 1$ provides a softer transition between neighboring clusters due to the smoothing effect of the averaging operation in Eq. (8), which makes the clusters' borders less sharp (see Fig. 4(b)). As shown in Fig. 5(b), this approach has a positive effect in the final image quality, even though clustering artifacts are still perceivable in the final image. An efficient solution to this problem is achieved by replacing the simple average in Eq. (8) by a weighted average given by:

$$\tilde{V}(p, p_c; C_{SP}) = \sum_{f=1}^{F} w_f V(p_f, p_c)\,, \tag{9}$$

where w_f is a normalized weight assigned to the fth cluster such that $\sum_{f=1}^{F} w_f = 1$. The weight of each SP cluster f is inversely proportional to the distance between its representative p_f and the shading point p, such that:

$$w_f = \frac{w'_f}{W}\,, \tag{10}$$

where $w'_f = \left(\text{dist}(p, p_f)\right)^{-1}$ cf. Eq. (7), and $W = \sum_{f=1}^{F} w'_f$. Such an approach successfully eliminates the hard borders between neighboring pixels (see Fig. 4(c)) and allows efficient rendering of images with high visual quality (see Fig. 5(c)). We empirically found that using $F = 6$ provides a good balance between image quality and computation time.

3.3. GPU implementation

3.3.1. Hard clustering

Algorithm 1 depicts the massively multi-threaded version of K-means clustering proposed for efficient execution in the GPU. It is parameterized with: a dataset D (c.f. Section 3.2.1); the number of clusters to generate (K) and a downsampling factor γ. In this section we will only consider the case where the fuzziness parameter $F = 1$, i.e., hard clustering. The case where $F > 1$ (i.e., soft clustering) will be treated in Section 3.3.2. Algorithm 1 returns a vector C of K centroids, as well as a matrix A mapping each data point to the F cluster indexes to which it belongs. Using $F = 1$ results in matrix A becoming a column vector with size M. Finally, the role of the variable W will become clear in Section 3.3.2, and can be overseen at this stage.

Algorithm 1: K-Means: GPU parallel

```
input  : $D$ (input dataset),
         $K$ (number of clusters),
         $\gamma$ (downsampling factor),
         $F$ (clustering fuzziness)
output: $A$ (matrix mapping each data point $d_m \in D$ to a
         vector $I_m$ of $F$ cluster indexes),
         $C$ (vector of K centroids),
         $W$ (matrix with inverse distances for each pair
         $(d_m, I_m) \in A$)
   /* Downsample the dataset */
1  $\mathcal{D} \leftarrow$ downsample($D, \gamma$) ;
   /* Initialize centroids */
2  $C \leftarrow$ initialCentroids($K, \mathcal{D}$) ;
   /* Initialize vector $\mathcal{A}$ */
3  $\mathcal{A} \leftarrow [-1] \times$ len($\mathcal{D}$) ;
4  for $i \leftarrow 1$ to iterations do
       /* Acceleration structure */
5      $G \leftarrow$ grid($C$) ;
       /* Assignment step */
6      do in parallel for $d_s \in \mathcal{D}$
7          $minDist \leftarrow \infty$;
           /* Iterate over neighborhood */
8          for $c_k \in$ neighborhood($d_s, C, G$) do
9              if dist($d_s, c_k$) $\leq minDist$ then
10                 $minDist \leftarrow$ dist($d_s, c_k$) ;
11         $\mathcal{A}[s] \leftarrow k$ ;
       /* Recompute centroids */
12     $counter \leftarrow value \leftarrow [0] \times K$ ;
13     do in parallel for $d_s \in \mathcal{D}$
14         $k \leftarrow \mathcal{A}[s]$ ;
15         atomicAdd($value[k], d_s$);
16         atomicAdd($counter[k]$, 1);
17     do in parallel for $k \leftarrow 1$ to $K$
18         $C[k] \leftarrow value[k]/counter[k]$ ;
   /* Final assignment step */
19 $A, W \leftarrow$ clusterAssign($C, D, F$) ;
20 return $C, A, W$
```

The algorithm starts by downsampling the M points of the input dataset D, yielding a reduced dataset $\mathcal{D}$ with S points such that $S \ll M$ and $\mathcal{D} = \{d_s | s = 1 \dots S\}$ (line 1). This allows the main loop, which computes the C vector of K centroids, to operate over a smaller set of data points (i.e., with those stored in $\mathcal{D}$), therefore saving execution time. The downsampling factor is given by the parameter γ. Then, a subset of K points of $\mathcal{D}$ is selected as initial centroids and stored in vector C (line 2); this selection is often random, but it can also leverage some structure existing in the data. The loop starting at line 4 repeatedly assigns data points $d_s \in \mathcal{D}$ to the current clusters (lines 6 to 11) and updates the respective centroids (lines 13 to 18) based on the new assignment. Finally, when the main loop concludes, the M data points of the original (i.e., full resolution) dataset D are assigned to the K final centroids stored in vector C (line 19). In the simplest case, corresponding to $F = 1$, each data point $d_m \in D$ is assigned to a single cluster $c_k \in C$.

Note that the two inner loops at lines 6 and 13 are **parallelized** over the S data items $d_s \in \mathcal{D}$. Parallel decomposition, whenever possible, is performed over the largest collection of data available, such that maximum degree of parallelism is made available to the GPU. In practice, this means that, in most cases, the decomposition is data parallel over the dataset $\mathcal{D}$ instead of C or I_m, since $M \gg K \gg F$, respectively. Another important property of Alg. 1 is the use of a **regular grid** G (line 5) creating a partial 3D ordering over the centroids in C. The assignment of data points to clusters can therefore be optimized, in the sense that only the centroids c_k in the neighborhood of each data point d_s are tested for the minimum distance (line 8). This neighborhood was selected to be a cube of 27 voxels centered on the voxel containing the data point whose nearest neighbors we wish to find. Finally, we would like to emphasize the **atomic add** operations that are used to compute the average value (i.e., position and normal) of all members of a cluster (line 15), in order to maximize parallelism while minimizing overheads associated with shared data updates.

3.3.2. *Fuzzy clustering*

Algorithm 2: Clustering Assignment

```
input  : $D$ (input dataset),
         $C$ (vector of K centroids),
         $F$ (clustering fuzziness)
output: $A$ (matrix mapping each datapoint $d_m \in D$ to a
         vector $I_m$ of $F$ cluster indexes),
         $W$ (matrix with inverse distances for each pair
         $(d_m, I_m) \in A$)
1  $A \leftarrow W \leftarrow [[\,]] \times$ len($D$) ;
   /* Acceleration structure */
2  $G \leftarrow$ grid($C$) ;
   /* Final assigning step */
3  do in parallel for $d_m \in D$
       /* Array F Distances */
4      $R_m \leftarrow [\infty] \times F$;
       /* Array F centroids indexes */
5      $I_m \leftarrow [-1] \times F$;
6      $n_f \leftarrow 0$;
       /* Iterate over the neighborhood */
7      for $c_k \in$ neighborhood($d_m, C, G$) do
8          $r_{m,k} \leftarrow$ dist($d_m, c_k$) ;
9          $f \leftarrow 1$ ;
10         while $f \leq n_f$ and $r_{m,k} > R_m[f]$ do $f++$;
11         if $f \leq F$ then
12             if $n_f < F$ then $n_f++$;
13             for $j \leftarrow n_f$ downto $f+1$ do
14                 $R_m[j] \leftarrow R[j-1]$ ;
15                 $I_m[j] \leftarrow I[j-1]$ ;
16             $R_m[f] \leftarrow r_{m,k}$ ;
17             $I_m[f] \leftarrow k$ ;
18     $A[m,:] \leftarrow I_m$ ;
19     $W[m,:] \leftarrow 1/R_m$ ;
20 return $A, W$
```

Fuzzy clustering is efficiently implemented in our framework by considering a fuzziness value $F > 1$ in the cluster assignment step of Alg. 1 (line 19). Alg. 2 shows how the data points of the original dataset D are assigned to the final K clusters $c_k \in C$ (line 19, Alg. 1). Its input is set D of data points which must be assigned to clusters; the vector C of K centroids to which the data points will be assigned; and the clustering fuzziness factor F. The output of Alg. 2 is a matrix A mapping each data point d_m to its corresponding cluster indexes; and a matrix W, containing, for each data point d_m, the inverse of the distance between d_m and

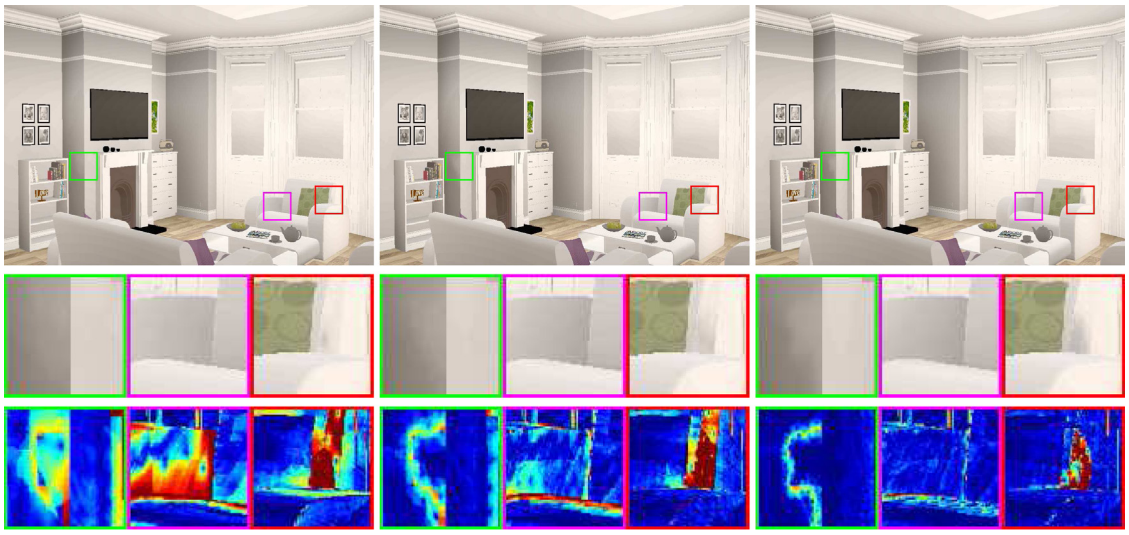

(a) ≈ 2K clusters (0.3%) (b) ≈ 8K clusters (1.1%) (c) ≈ 30K clusters (4.1%)

Fig. 6. Rendering results for the Living Room scene when clustering shading points only with a varying number of clusters (from approximately 2k clusters to approximately 30k clusters). The number in-between parentheses shows the percentage of used clusters with respect to the total number of shading points. The rendering times for (a), (b) and (c) were 319 ms (MSE = 7.42E−4), 349 ms (MSE = 4.54E−4) and 472 ms (MSE = 3.00E−4), respectively. The reference image rendering time was 1195 ms.

Table 1
Scenes' characteristics.

Scene	Triangles	VPL	Resolution
CornellBox	1.9M	5000	1024 × 1024
Conference	331k	8000	1024 × 720
LivingRoom	313k	10 000	1024 × 720

the representatives of the clusters it belongs to. The algorithm starts by initializing matrix A (line 1), as well as creating the regular grid G which acts as an acceleration structure to search over C (line 2). The main loop (line 3), executed in parallel for all data points $d_m \in D$, assigns each d_m to the F closest centroids $c_k \in C$. To this end, two auxiliary vectors are maintained: R_m, an array containing the distances to the F closest centroids found, which is initialized at line 4; and I_m, an array with the indexes of the F closest centroids found, which is initialized at line 5. Then, at lines 7 to 19, we use the regular grid G to efficiently iterate over the centroids c_k which are in the neighborhood of d_m. At each iteration, we use the distance information in R_m to determine whether the current centroid c_k is closer to any of the closest centroids found so far. In that case we discard the farthest centroid and replace it with the current one.

4. Rendering results

In this section we present and analyze experimental results for the proposed approach. We start by describing the experimental setup and methodology. Then experimental results are presented incrementally, as in an ablative study. First the impact of clustering only shading points is analyzed. Then the same analysis is performed to clustering only the VPLs, followed by the final approach, which clusters both.

4.1. Experimental setup

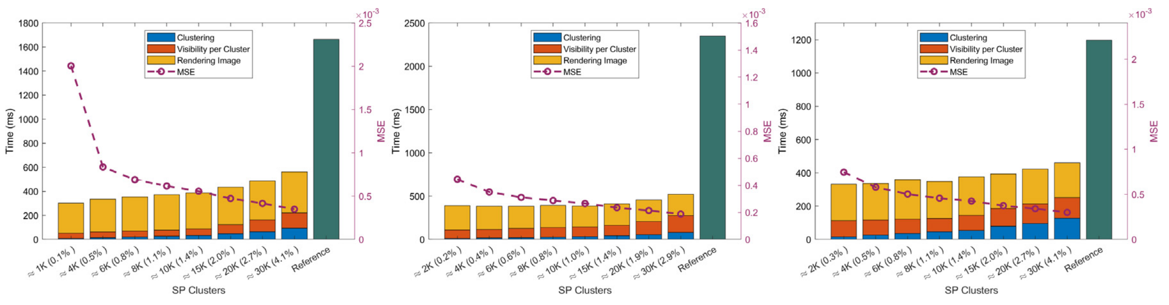
In the following we present results for three scenes with different characteristics (details are shown in Table 1, and reference images in Fig. 1). The Cornell Box scene provides a benchmark commonly used in rendering. The Conference Room scene allows testing difficult soft shadowing conditions, due to the visibility complexity caused by the presence of a large number of chair's legs. Finally, the Living Room scene provides an alternative test setting from the previous two scenes, with large areas of extremely smooth indirect light conditions. For all scenes, and when applying fuzzy clustering, each SP is assigned to 6 clusters ($F = 6$).

The results have been generated using an MSI Leopard with 16 GB of RAM, an Intel i7 processor and an NVIDIA RTX 2060 laptop graphic card with 6 GB of memory. The algorithm has been implemented on the NVIDIA OptiX 7.5 ray tracing engine [1]. The quality of the results is assessed by evaluating the MSE (mean squared error) of each generated image, by proving error maps that complement subjective visual inspections, and through measurements of the execution time. In order to avoid inaccuracies in the measured rendering times all timings have been computed using a 3-best approach: we measure the execution time 10 times, and present the execution time as the average of the three smallest execution times. The time required to shoot and to cluster the VPLs are presented separately in the figures' caption.

4.2. Clustering shading points

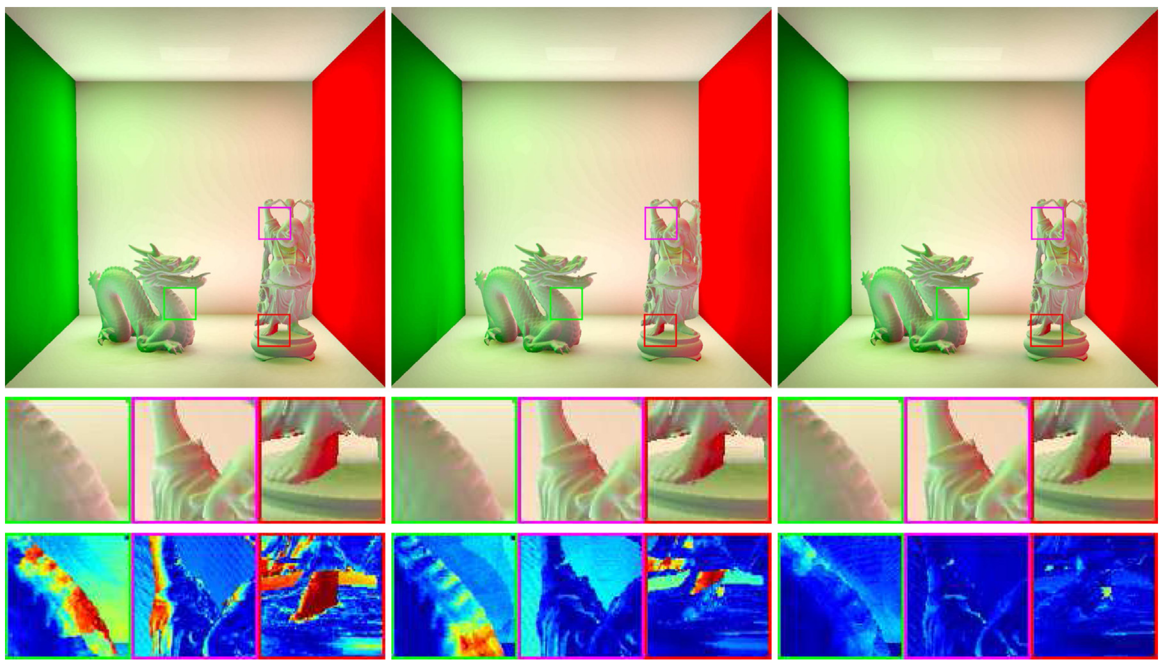
In this section, we present rendering results concerning clustering of shading points only, meaning that visibility is computed between each shading points cluster representative and each VPL. At the rendering stage, and for each shading point, indirect illumination is thus computed using all VPLs as in Eq. (3), but replacing the original visibility function V by the approximation given by Eq. (9).

Fig. 6 shows the rendering results for the Living Room scene using different numbers of clusters, ranging from 2k to 30k (corresponding to 0.3% and 4.1% to the total shading points, respectively). We can observe that as the number of clusters increases, the visual artifacts progressively vanish, and the MSE is reduced. Conversely the rendering time increases with the number of used clusters. This is explained by the fact that, as the number of used clusters increases, the visibility information becomes more refined, but also more costly to compute. With this regard, the



(a) Conference Room (b) Cornell Box (c) Living Room

Fig. 7. Timings and mean squared error (MSE) as a function of the number of used clusters. The vertical bars show the decomposition of the total rendering time into: clustering, evaluating the visibility term (for each cluster), and rendering the final image. The MSE values (in purple) are plotted with respect to the right-hand axis (also in purple).



(a) 500 VPL clusters (9.4%) (b) 1500 VPL clusters (25.0%) (c) 2500 VPL clusters (50.0%)

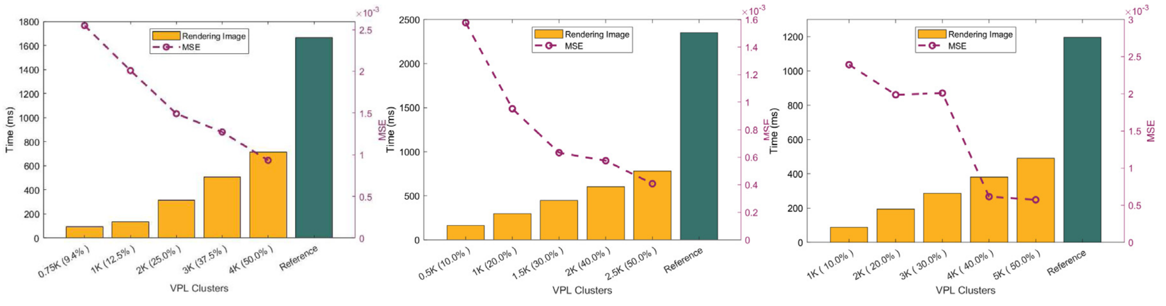
Fig. 8. Rendering results for the Cornell Box scene when clustering VPLs only with a varying number of clusters (from 500 clusters to 2500 clusters). A total of 7500 VPLs are clustered. The number in-between parenthesis shows the percentage of used clusters with respect to the total number of VPLs. The rendering times for (a), (b) and (c) were 168 ms (MSE = 1.80E−3), 439 ms (MSE = 5.42E−4) and 760 ms (MSE = 4.20E−4), respectively. The reference image took 2348 ms to render. The VPL tracing and clustering times for (a), (b) and (c) (not included in the rendering times) were 8.0 ms, 9.4 ms and 10.5 ms, respectively.

intermediate number of used clusters (8k, corresponding to 1.1% of the total shading points) seems to be a good compromise between rendering time and incurred error. Note also the drastic reduction in rendering time brought by our proposed method when compared to the reference, even in the case of 30k shading points clusters (from 1195 ms to 472 ms, respectively). Extended results shown in supplemental material (including the Cornell Box and the Conference Room scenes) fully agree with these observations.

The timings and MSE values for the three tested scenes when varying the number of SP clusters are depicted in Fig. 7. The rendering time is divided into clustering time (in blue, corresponding to Steps 3 and 4 in Fig. 2); time for computing the visibility of each VPL as seen from each representative point of the shading clusters (in red, Step 5 in Fig. 2); and finally, in yellow, the rendering time that consists of shading each SP using the data from the previous stages. Corresponding MSEs with respect to the reference image is shown in purple, with numerical values given by the right-hand axis. The results suggest the existence of a sweet spot corresponding to a good trade-off between image quality and rendering time. Indeed, for the three scenes, increasing the number of clusters to more than ≈1% of the total shading points brings relatively small reductions in terms of MSE, whereas rendering time can increase significantly.

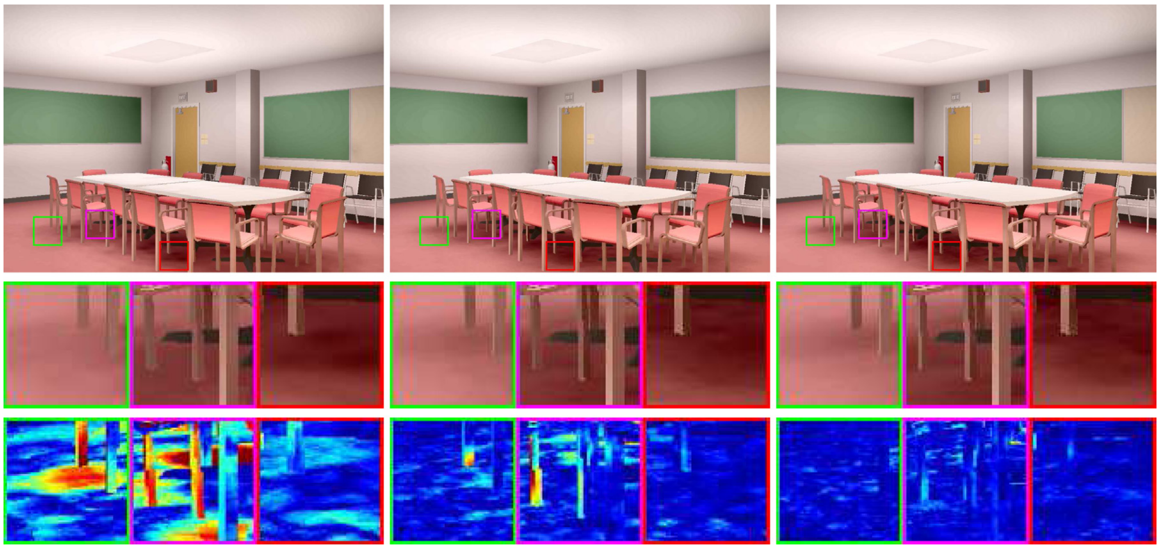
4.3. Clustering VPLs

In this section we present the results obtained when clustering VPLs only. The shading points remain unclustered, and the visibility of each VPL cluster representative is thus explicitly tested for each shading point. The rendering results for the Cornell Box scene with a varying number of VPL clusters are shown in Fig. 8 (an extended version of these results as well as results for the two other tested scenes are provided in supplementary material). We can observe that, similarly to the case of clustering shading points only, the error decreases with the number of used VPL clusters, whereas the rendering time increases. Furthermore, when comparing Fig. 8(b) and (c), we can observe that their MSE values



(a) Conference Room (b) Cornell Box (c) Living Room

Fig. 9. Timings and mean squared error (MSE) as a function of the number of used VPL clusters. The MSE values (in purple) are plotted with respect to the right-hand axis (also in purple).



(a) 0.75K VPL clusters and ≈ 1K shading clusters (b) 2K VPL clusters and ≈ 8K shading clusters (c) 4K VPL clusters and ≈ 30K shading clusters

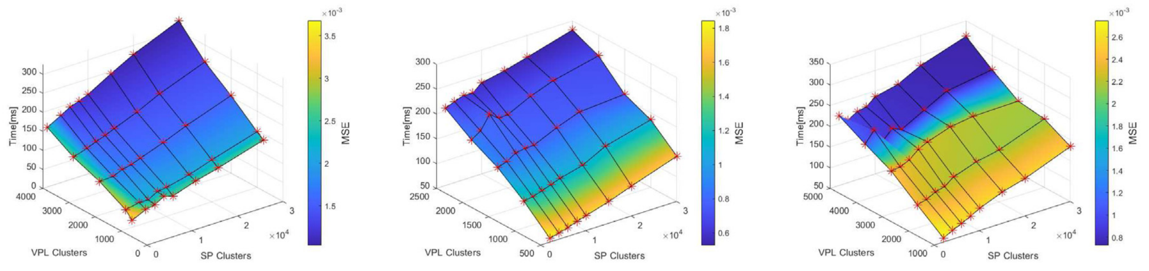
Fig. 10. Rendering results when clustering both VPLs and shading points. The results have been generated with three different configurations: (a) a light-weight configuration, with only 0.75k VPL clusters and 1k shading clusters; (b) an intermediate configuration, with 2k VPL clusters and 8k shading clusters; and (c), a high quality configuration, with 4k VPL clusters and 30k shading clusters. Note the progressive decrease of the rendering error, from left to right, as the number of used clusters increases. The rendering times for (a), (b) and (c) were 34 ms (MSE = 3.80E−3), 109 ms (MSE = 1.80E−3) and 328 ms (MSE = 8.55E−4), respectively. The reference image rendering time was 1715 ms. The VPL tracing and clustering times for (a), (b) and (c) (not included in the rendering times) were 5 ms, 6.3 ms and 10 ms, respectively.

are relatively similar, whereas the rendering time increases by a factor close to two. This indicates that there is a limit from which increasing the number of VPL clusters brings no significant gains in image quality, despite the increase in rendering time. It is important to note the almost negligible time required for VPLs shooting and clustering (8.0 ms, 9.4 ms and 10.4 ms for 0.5k, 1.5k and 2.5k VPL clusters, respectively). This shows the efficiency of our parallel clustering algorithm, and opens the path for a direct application of our method to dynamic scenes, where interactive clustering of VPLs would also be required. Finally, Fig. 9 shows the rendering times for different numbers of VPL clusters, for the three tested scenes. In all cases, our method offers an important reduction in terms of rendering time when compared to the reference.

4.4. Clustering both VPLs and shading points

Fig. 10 presents the rendering results when clustering both the VPLs and the shading points. Similarly to the previous sections, three different settings are considered corresponding to a low, an intermediate and a high number of used clusters. The resulting images confirm the previously observed trend: the error is progressively reduced as the number of clusters increases, while the rendering time increases with the number of clusters. In the three settings, the results show that our method allows a significant reduction in terms of rendering time, ranging from 50 to 5 times faster than the reference image (computed without clustering). Note that in all settings, clustering both VPLs and shading points always yields faster rendering times compared to the cases of clustering only shading points or VPLs.

Fig. 1 shows results for the three scenes when using an intermediate number of VPL and SP clusters (the actual number of used clusters varies from scene to scene, due to differences in incident light complexity). More detailed results can be found in supplemental material. For all scenes, the results show an important reduction in terms of rendering time, while keeping the approximation error (MSE) at low levels. The final image quality is also generally comparable to that of the reference image taking into account the drastic reduction obtained in rendering time. Finally, a detailed depiction of the relationship between



(a) Conference Room (b) Cornell Box (c) Living Room

Fig. 11. Time in function of the number of clusters for the shading points and the number of clusters for the VPLs. The color of the surface depends on the MSE. The reference images required 1715 ms, 2348 ms and 1195 ms respectively.

rendering time and MSE as a function of the used VPLs and shading points (SP) clusters is provided in Fig. 11, for the three tested scenes.

5. Conclusions and future work

In this paper we have proposed and thoroughly assessed a massively parallel algorithm for global illumination at interactive frame rates. Our solution is based on the use of Virtual Point Lights and on soft clustering algorithms. It is built using CUDA and the NVidia OptiX ray-tracing framework and thus runs entirely in the GPU. Thanks to the clustering of both the shading points and the VPLs, we have shown that the rendering time of the original algorithm can be drastically reduced, while keeping the final image quality (and MSE) at satisfactory values. As shown by our results, our parallel algorithm for soft K-means clustering can be used to cluster shading points interactively, which allows for dynamically changing the camera setting. Furthermore, the sheer cost of parallel clustering indicates that an application of our algorithm to dynamic scenes (i.e., with moving objects and light sources) for which interactive clustering of VPLs is also required is perfectly feasible.

With respect to future work, several research lines could be pursued in order to address some of the limitations of our method. Perhaps the most obvious one is extending the method so as to handle glossy reflections. This could be achieved by storing the incoming direction of the VPL, but would likely require the total number of VPLs in the scene to increase significantly so as to capture the directional distribution over the glossy materials. In order to assure the coherence of the VPLs within the resulting clusters, one possibility would be to take into consideration the material glossiness and the direction of incidence of the VPL in the clustering Distance Metric, so as to create clusters of VPLs with similar features.

Exploiting temporal coherence across consecutive frames, which is currently not considered, is another interesting possible extension. For example, the result of clustering the shading points in a given frame could be used as initial condition for K-means clustering of the shading points in the subsequent frame. Furthermore, our method requires a user-specified number of shading points and VPL clusters, which is scene dependent. Future work could address this limitation by automatically determining the number of clusters required to achieve a given frame rate or image quality value (which would be more intuitive to provide than the number of clusters).

Inspired by hierarchical approaches, such as lightcuts [6], the clustering could be generalized into a hierarchy, allowing improved scalability to large scenes with large numbers of VPLs. Additionally, VPL clusters could be importance sampled, such that only the clusters that potentially contribute to the indirect illumination at each SP cluster are sampled using shadow rays. Inspired by the LGH [12] and SST [13] approaches, it would be interesting to evaluate whether each cluster representative (both SPs and VPLs) can be stochastically selected whenever a visibility assessment or a shading operation is performed; this would allow trading structured artifacts by high frequency noise, eventually allowing a reduction on the number of required clusters for the same image quality. Stochastic sampling would incur the cost of building the required probability distributions; advantages and disadvantages would have to be thoroughly assessed.

As regards the weighted interpolation, which acts as a smoothing operation, we have shown that it successfully removes most low frequency artifacts. However it might eventually have a negative impact on existing sharp shadows, although we did not observe this issue in our results. One way to cope with this potential problem would be to replace our weighting function (which is currently given by the inverse of the Euclidean distance) by a function with a faster fall-off, possibly controlled by some parameter. This would allow explicitly controlling the degree of smoothness provided by the weighted interpolation.

CRediT authorship contribution statement

Arnau Colom: Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Visualization. **Ricardo Marques:** Conceptualization, Methodology, Visualization, Writing – original draft, Writing – review & editing, Supervision, Project administration. **Luís Paulo Santos:** Conceptualization, Methodology, Supervision, Writing – original draft, Writing – review & editing, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

No data was used for the research described in the article.

Acknowledgments

As Serra Húnter Fellow, Ricardo Marques acknowledges the support of the Serra Húnter Programme. This work is partially funded by TIN2018-095232-B-C21, MUSAE Horizon Europe project (#101070421) and CERCA Programme/Generalitat de Catalunya. This work is financed by National Funds through the Portuguese funding agency, FCT - Fundação para a Ciência e a Tecnologia, within project LA/P/0063/2020. Arnau Colom and Ricardo Marques acknowledge the support of the European Union's Horizon 2020 Research and Innovation Programme under Grant Agreement No 856879 (Present Project).

Appendix A. Supplementary data

Supplementary material related to this article can be found online at https://doi.org/10.1016/j.cag.2022.09.008.

References

[1] Parker SG, Bigler J, Dietrich A, Friedrich H, Hoberock J, Luebke D, McAllister D, McGuire M, Morley K, Robison A, Stich M. OptiX: A general purpose ray tracing engine. ACM Trans Graph 2010;29(4). http://dx.doi.org/10.1145/1778765.1778803.

[2] Dachsbacher C, Křivánek J, Hašan M, Arbree A, Walter B, Novák J. Scalable realistic rendering with many-light methods. Comput Graph Forum 2014;33(1):88–104. http://dx.doi.org/10.1111/cgf.12256.

[3] Kajiya JT. The rendering equation. SIGGRAPH Comput Graph 1986;20(4):143–50. http://dx.doi.org/10.1145/15886.15902.

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

[5] Walter B, Fernandez S, Arbree A, Bala K, Donikian M, Greenberg DP. Lightcuts: A scalable approach to illumination. ACM Trans Graph 2005;24(3):1098–107. http://dx.doi.org/10.1145/1073204.1073318.

[6] Walter B, Arbree A, Bala K, Greenberg DP. Multidimensional lightcuts. ACM Trans Graph 2006;25(3):1081–8. http://dx.doi.org/10.1145/1141911.1141997.

[7] Dong Z, Grosch T, Ritschel T, Seidel H-P. Real-time indirect illumination with clustered visibility, In: Proc. vision modeling and visualization, 2009.

[8] Hašan M, Pellacini F, Bala K. Matrix row-column sampling for the many-light problem. In: ACM SIGGRAPH 2007 Papers. SIGGRAPH '07, New York, NY, USA: Association for Computing Machinery; 2007, p. 26–es. http://dx.doi.org/10.1145/1275808.1276410.

[9] Ou J, Pellacini F. LightSlice: Matrix slice sampling for the many-lights problem. In: Proceedings of the 2011 SIGGRAPH Asia Conference. SA '11, New York, NY, USA: Association for Computing Machinery; 2011, http://dx.doi.org/10.1145/2024156.2024213.

[10] Davidovič T, Křivánek J, Hašan M, Slusallek P, Bala K. Combining global and local virtual lights for detailed glossy illumination. ACM Trans Graph 2010;29(6). http://dx.doi.org/10.1145/1882261.1866169.

[11] Jarabo A, Buisan R, Gutierrez D. Bidirectional clustering for scalable VPL-based global illumination. In: CEIG 2015 - Spanish computer graphics conference. 2015.

[12] Lin D, Yuksel C. Real-time rendering with lighting grid hierarchy. Proc ACM Comput Graph Interact Tech 2019;2(1). http://dx.doi.org/10.1145/3321361.

[13] Tatzgern W, Mayr B, Kerbl B, Steinberger M. Stochastic substitute trees for real-time global illumination. In: Symposium on interactive 3D graphics and games. I3D '20, New York, NY, USA: Association for Computing Machinery; 2020, http://dx.doi.org/10.1145/3384382.3384521.

[14] Wang R, Wang R, Zhou K, Pan M, Bao H. An efficient GPU-based approach for interactive global illumination. ACM Trans Graph 2009;28(3). http://dx.doi.org/10.1145/1531326.1531397.

[15] Ward GJ, Rubinstein FM, Clear RD. A ray tracing solution for diffuse interreflection. SIGGRAPH Comput Graph 1988;22(4):85–92. http://dx.doi.org/10.1145/378456.378490.

[16] Cuomo S, De Angelis V, Farina G, Marcellino L, Toraldo G. A GPU-accelerated parallel K-means algorithm. Comput Electr Eng 2019;75:262–74, URL https://www.sciencedirect.com/science/article/pii/S0045790617327994.

[17] Bhimani J, Leeser M, Mi N. Accelerating k-means clustering with parallel implementations and GPU computing. In: 2015 IEEE high performance extreme computing conference (HPEC). 2015, p. 1–6. http://dx.doi.org/10.1109/HPEC.2015.7322467.