Global Illumination Using Well-Separated Pair Decomposition
N. Bus, Nabil Mustafa, Venceslas Biri

To cite this version:

HAL Id: hal-01188993
https://hal.archives-ouvertes.fr/hal-01188993v2
Submitted on 30 Oct 2017
Instant radiosity methods rely on using a large number of virtual point lights (VPLs) to approximate global illumination. Efficiency considerations require grouping the VPLs into a small number of clusters that are treated as individual lights with respect to each point to be shaded. Two examples of clustering algorithms are Lightcuts [WFA05] and LightSlice [OPT11]. In this work we use the notion of geometric separatedness of point sets as a basis for a data structure for pre-computing and compactly storing a set of candidate VPL clusterings. Our data structure is created prior to rendering, is view-independent and relies only on geometric and radiometric information. For any point to be shaded, we show that a suitable clustering of the VPLs can be efficiently extracted from this data structure. We develop the above framework into an accurate and efficient clustering algorithm based on well-separated pair decompositions which outperforms earlier work in speed and/or quality for diffuse scenes.

1 Introduction

Several methods have been proposed to solve the global illumination problem, towards the ultimate goal of efficient and realistic rendering of large scenes in the presence of varied and complex lighting effects. Pure unbiased Monte Carlo algorithms such as Metropolis light transport [VG97] or simple bidirectional path tracing [VG95] are considered to be gold standards for reference solutions. However they produce noisy results that are slow to converge in the case of complex scenes. Biased algorithms like Photon Mapping [Jen01], point-based global illumination [Chr08] or the many-lights methods, such as Instant Radiosity [Kel97], provide good performance in many practical applications.

The last method and its improvements have proven to be very useful for approximating global illumination. By tracing light paths from light sources, they create virtual point lights (VPLs) at the intersections of the surface of the scene and the paths. Global illumination is estimated by computing the direct illumination from all of the VPLs (we refer the reader to the SIGGRAPH 2012 course notes on the many-address of the many-lights problem [KHA12] and to the EG state-of-the-art report [DKH13]). Despite some limitations, they provide a unified and scalable approach to the problem of computing global illumination. However, as the number of VPLs needed for a good-fidelity approximation is large, computing the illumination for each point by summing up the contribution of each individual VPL can become prohibitively expensive.

Clustering in the many-lights method. Let \( \mathcal{S} \) be the set of VPLs, henceforth considered the set of input lights and \( \mathcal{P} \) be the set of points to be shaded, i.e., points in the scene hit by the rays traced for all the pixels. One key idea for speeding up computations is to cluster the VPLs. In other words, for each point \( p \in \mathcal{P} \), partition \( \mathcal{S} \) into a small number of clusters (each partition of \( \mathcal{S} \) into clusters is called a clustering of \( \mathcal{S} \)), and then consider each cluster as a single VPL when calculating its contribution to the shading at \( p \).

The goal is to compute, for each \( p \in \mathcal{P} \), a clustering that minimizes the shading error at \( p \). See Figure 1 for the image produced by our clustering algorithm as well as the differ-
ence from the reference image. This forces all algorithms to be *adaptive*, i.e., as one iterates over all the points to be shaded, the clustering has to be recomputed again with respect to the spatial and radiometric properties of each point (though various improvements are possible, at a loss of quality, by exploiting spatial coherence to re-use previous computation).

As the set $S$ of VPLs is view-independent, an idea that improves efficiency is to compute the set of all possible candidate clusters of $S$ before the rendering computation (and store in some hierarchical structure). Then during rendering, a clustering for each point is selected by choosing an appropriate subset of clusters from this pre-computed set. This is an expensive task which forces examining a large number of clusters (again, this can be somewhat ameliorated by exploiting the spatial coherence between neighboring pixels). This has been the basis of previous work; two well-known examples are Lightcuts [WFA∗05] and LightSlice [OP11].

**Our Contributions.** We consider our main contributions to be two-fold. First, we make the next logical step in the many-lights clustering paradigm: rather than pre-computing a set of individual candidate clusters from which clusterings are computed during rendering, we pre-compute a number of clusterings of $S$. Then, during the rendering phase, the clustering for a point $p$ can simply take to be one of these pre-computed clusterings, together with some minor modification. Our data structure stores the clusterings compactly, it is view-independent and computed prior to rendering. This results in a very efficient rendering phase. For a very natural criterion of clustering, we show that

- the total number of these pre-computed clusterings will be *independent* of the number of points in $P$ to be shaded and will only depend on the size of $S$.
- the modification required for each point will be provably small; in fact it will be independent of the size of $S$ or $P$.

Second, we develop the above framework into an accurate and efficient new many-lights clustering method. It computes a clustering relying on geometric and radiometric data for fast and accurate computation. We show that the complexity of our scheme is largely invariant on geometric scenes, and it is easily scaleable with the number of VPLs.

We prove theoretical guarantees as well as experimentally validate the computational efficiency of our scheme, contrasting it with two of the most well-known earlier systems, Lightcuts [WFA∗05] and LightSlice [OP11]. In particular, the advantages of our work include:

- Our pre-computation phase is view-independent, and so are the pre-computed clusterings. Unlike LightSlice and Lightcuts, this allows our algorithm to re-utilize computation with changing camera position.
- As all the clustering computations are moved to the pre-computation phase, the rendering phase takes constant time for each $p \in P$, i.e., independent of the number of VPLs. On the other hand, Lightcuts has to maintain a heap and do clustering computations. Our method is able to produce similar output as Lightcuts with around 3 times average speedup.
- It outperforms LightSlice in speed and quality, achieving, e.g., 2 times speedup with consistently better quality. The errors of our algorithm are smooth, and visually difficult to detect, unlike for LightSlice which suffers from visible blocking effects.
- It also uses a significantly lower amount of memory than LightSlice which requires around 30 GB for scenes with around 0.5 million VPLs while our algorithm runs with 5 GB. This allows the usage of a considerably higher number of VPLs for rendering.

Broadly our work shows that the set of VPLs itself contain enough information such that with intensive preprocessing, a geometrically good clustering can be constructed for each point $p \in P$ with provably little effort.

**Organization.** In Section 2 we review previous work on global illumination with the many-lights method. After preliminaries in Section 3 we present the novel ideas involved in our approach in Section 4. Theoretical details on our clustering representation and computation follow in Section 5. Finally, limitations and future work is discussed in Section 6. Appendix 10 contains all the formal proofs of the statements stated in the paper.

2 Previous Work

Many-lights rendering techniques estimate global illumination using VPLs. In Instant radiosity [Kec97], the original many-lights technique, each point is shaded using all VPLs. Since then, many improvements have been made to avoid the limitations such as clamping and diffuse-only global illumination. Further efforts to optimize the computation include carefully placing the VPLs, speeding up visibility computation or selecting a subset of the VPLs to use for each shaded point.

**Real-time techniques.** Approximate global illumination can be done in real-time with incremental selection of the VPLs [LSK′07], quick shadow computations and deferred shading [RGK′08] or using simple scene descriptors [HREB11]. Clustering VPLS into area lights for real-time GPU based rendering has been proposed in [PKD12]. [SIMP06] uses stochastic sampling of VPLs to achieve interactive frame rates. Our method has different scope from
3 Preliminaries

We use similar notation as in Lightcuts [WFA+05]. The radiance for a point \( p \in \mathcal{P} \) in direction \( \omega \) caused by the direct contribution of the lights in \( \mathcal{S} \) is denoted by \( L(p, \omega) \). It is a function that sums up all lights the product of the material, geometry, visibility and intensity terms, where each product represents the radiance caused by a single light:

\[
L(p, \omega) = \sum_{s \in \mathcal{S}} M_s(p, \omega) \cdot G_s(p) \cdot V_p(s) \cdot I_s
\]

Here \( I_s \) is the intensity of the light \( s \) while the geometric term \( G_s(p) \) captures the light attenuation. The VPLs are spotlights having a direction \( N_p \), and therefore \( G_s(p) = \cos(\phi)/d(p, s)^2 \), where \( d(p, s) \) is the Euclidean distance between \( p \) and \( s \), and \( \phi \) is the angle between \( N_p \) and the vector \( p-s \). The material term \( M_s(p, \omega) \) is the BRDF which depends on the local geometry and material at \( p \). We will use Lambertian and Phong BRDFs, which only depend on the angles \( \theta \) and \( \beta \). The former is the angle between the surface normal \( N_p \) and the light direction \( s-p \) and the latter is the angle between \( \omega' \) (the view direction \( \omega \) reflected on \( N_p \)) and the light direction. \( V_p(s) \) denotes the visibility of \( p \) from the light \( s \).

For a cluster \( C \subseteq \mathcal{S} \), let \( \text{rep}(C) \) denote a representative light \( s \in C \). Then the radiance at \( p \) from lights in \( C \) with representative \( \text{rep}(C) \) can be approximated as:

\[
L_C(p, \omega) = M_{\text{rep}(C)}(p, \omega) \cdot G_{\text{rep}(C)}(p) \cdot V_p(\text{rep}(C)) \cdot \sum_{s \in C} I_s
\]

Let \( C = \{C_1, \ldots, C_k\} \) denote a clustering of \( \mathcal{S} \) into \( k \) clusters. The radiance at \( p \) from all the lights in \( \mathcal{S} \) can be approximated as:

\[
L_C(p, \omega) = \sum_{C \subseteq \mathcal{S}} L_C(p, \omega)
\]

4 Our Approach

For a point \( p \in \mathcal{P} \) to be shaded, consider a clustering of the set of VPLs \( \mathcal{S} \) into \( k \) clusters \( C_1, \ldots, C_k \) such that the radius of the smallest-ball containing each cluster is much smaller than the distance of that ball to \( p \) (this will be formulated precisely in Section 5). Call such a cluster well-separated, or a ws-cluster for brevity, from \( p \), and the clustering \( \{C_1, \ldots, C_k\} \) a well-separated clustering, or a ws-clustering, for \( p \). Intuitively, from the point of view of
$p$, all the points in each cluster behave roughly like a single point. Figure 3 illustrates this for an arbitrary point $p$, shaded bright green, a few of the $\text{ws}$-clusters of $S$ around it. Note that as the $\text{ws}$-cluster’s distance to $p$ decreases, the well-separated criterion automatically ensures that the radius of the cluster decreases as well. Our goal is to compute a $\text{ws}$-clustering of $S$ for each point $p$ to be shaded during the rendering phase.

Our approach has two main components: first we propose a method to pre-compute and compactly store several $\text{ws}$-clusterings of $S$. Then during rendering, we show how to quickly extract a $\text{ws}$-clustering for each point to be shaded using this pre-computed structure. This pre-computed clustering will then be slightly modified to fit the spatial properties of the shaded point. We sketch the main components of the system in Figure 2 and outline them below.

Figure 3: The Museum scene with the well-separated clusters (represented by their enclosing spheres) around a shaded point, shown as a bright green square in the middle of the image. For visualization purposes we only included a quarter of the clusters.

### Pre-compute clusterings

The computation of $\text{ws}$-clustering for each point $p$ is dependent on spatial properties of $p$, and requires individual computation during rendering for each point, an expensive task. Instead, we will do the following in a view-independent pre-rendering phase: compute a $\text{ws}$-clustering of $S$ with respect to each light $s \in S$. In other words, for each light $s$ in $S$, compute the partition of the remaining lights into clusters satisfying the well-separatedness criterion. The key to this construction will be the use of a partitioning data-structure, the well-separated pair decomposition [CK95] (henceforth denoted as WSPD). These $\text{ws}$-clusterings will be stored implicitly in a compact structure from which the $\text{ws}$-clustering for any light $s \in S$ can be extracted quickly.

Well-separated clusters approximate the geometric terms of the rendering equation, but ignore visibility and material properties. To adjust for this, we will further compute two additional structures in this pre-processing phase. First, we further group the lights in each $\text{ws}$-cluster $C$ into a small number of subgroups by similar light normals. This additional grouping will be used to evaluate the illumination from the cluster more precisely. Second, we introduce representative lights that approximate local visibility for each $\text{ws}$-cluster $C$ as follows: sample a number of directions and compute the illumination of the lights in $C$ reaching the boundary of the ball $b(C)$ in the sampled directions, where $b(C)$ is the smallest-ball containing $C$. This will be used to estimate the visibility of the lights in $C$ to $b(C)$; the visibility test from $b(C)$ to $p$ will be performed during the view-dependent rendering phase.

### Retrieve $\text{ws}$-clustering

During the rendering computation, for an arbitrary point $p$, find the closest point in $S$ to $p$ (an approximate nearest-neighbor is sufficient and will be used), and start with its (pre-computed) clusters as the clustering for $p$. Furthermore, refine each cluster by subdividing it into new clusters.
For a cluster to that of between two clusters. Two point sets
well-separated pair decompositions.

We first need to
compute $ws$-clusterings efficiently, we will need to use
a basic structure in the theory of geometric computing, the
well-separated pair decomposition.

**Well-separated pair decompositions.** We first need to
extend the notion of well-separatedness between a point and
a cluster to that of between two clusters. Two point sets
$R$ and $Q$ are well-separated from each other if, for a given
separation parameter $\varepsilon > 0$, the radius of both the balls $b(R)$
and $b(Q)$ is smaller than $\varepsilon \cdot d(R, Q)$, i.e., $\max(r(R), r(Q)) <
\varepsilon \cdot d(R, Q)$ where $d(R, Q)$ is the distance between $b(R)$ and
$b(Q)$.

**Definition 5.2.** A well-separated pair decomposition of $S$
for a given separator parameter $\varepsilon$ is a list of pairs of clusters
$\{(R_1, Q_1), \ldots, (R_s, Q_s)\}$, where each $R_i, Q_i \subseteq S$, and

i) for every pair of points $p, q \in S$, there is a unique index
$s$ such that $p \in R_i$ and $q \in Q_i$, and

ii) for all $i = 1 \ldots s$, the clusters $R_i$ and $Q_i$ are well-
separated from each other, with separation parameter $\varepsilon$.

Here $s$ is called the size of the WSPD. See Figure 5
for some example pairs $\{R_i, Q_i\}$ for a point set in two di-
dimensions. A remarkable fact about WSPDs is that there

Figure 4: For $\varepsilon = 0.5$, $C$ is $ws$ from $p$. In other words, we have $d(p, C) > 2 \cdot r(C)$.

until they are well-separated from $p$. The number of clusters
required to achieve this $ws$-clustering criteria could poten-
tially be quite high, for two reasons: i) the clustering for
the closest point in $S$ to $p$ could have many clusters, and ii)
refinement could add many more new clusters to this initial
clustering. It will be shown that refinement can only add a
constant number of new clusters for any point $p$. This con-
stant is provably independent of the number of lights in $S$
or points in $P$. Also, under some basic assumptions on the
geometry of scenes, we will show that the average size of a
$ws$-clustering for points of $p$ will be logarithmic in the size
of $S$. Experimental evidence will confirm this behavior.

**Calculate $L(p, \omega)$**

Let $C_p = \{C_1, \ldots, C_k\}$ be the final constructed $ws$-clustering
for $p \in P$ during rendering. Furthermore let $\{C^1_1, \ldots, C^k_k\}$ be
the $k_i$ subgroups of each $C_i$ by similar light normals. From
a single subgroup $C_i^j$ of $C_i \in C_p$, we compute:

$$L_{C_i^j}(p, \omega) = M_{\text{rep}(C_i^j)}(p, \omega) \cdot G_{\text{rep}(C_i^j)}(p) \cdot \sum_{s \in C_i^j} I_s$$

(4)

Then we approximate the illumination at $p$ from the cluster
$C_i \in C_p$ as:

$$L_{C_i}(p, \omega) = \left( \sum_{j=1}^{k_i} L_{C_i^j}(p, \omega) \right) \cdot V_p(\partial(b(C_i))) \cdot R(C_i, p)$$

(5)

where $V_p(\partial(b(C_i)))$ denotes the visibility from $p$ to the
boundary of the sphere $b(C_i)$ enclosing cluster $C_i$; this
is computed by a shadow ray with the Embree raytracing ker-
nel. $R(C_i, p)$ is the proportion of the summed intensity of
the lights in $C_i$ reaching the boundary of the ball $b(C_i)$ in
the direction from the center of $b(C_i)$ to $p$; see Section 5
for its precise definition and computation.

**5 Constructing $ws$-clusterings**

For a cluster $C \subseteq S$, define $b(C)$ to be the smallest-enclosing
ball of the points in $C$. Let $r(C)$ be the radius of the ball
$b(C)$. For any point $p$, $d(p, C)$ denotes the Euclidean dis-
tance of $p$ to $b(C)$.

A well-separated cluster. We introduce a necessary condition
that each cluster must satisfy when constructing a
clustering of $S$ w.r.t. a shaded point $p$ – namely that it is
well-separated from $p$, for a given parameter $0 \leq \varepsilon \leq 1$ ($\varepsilon$
will be called the separation parameter).

**Definition 5.1.** A cluster of lights $C$ is well-separated from
a point $p$ if $r(C) < \varepsilon \cdot d(p, C)$, where $\varepsilon$ is the separation
parameter.

The lights in a $ws$-cluster w.r.t. $p$ are ‘far enough’ from
$p$, and concentrated in a small ball (see Figure 4). This condition
implies that from the point of view of $p$, all the lights
in a $ws$-cluster are in a similar direction and the distances
of $p$ to the lights in $C$ are approximately the same. Since
the luminosity reaching $p$ depends on the angle and the dis-
tance of lights in $S$ from $p$ (differences regarding visibility
and light normals will be accounted for later in Section 5)
it can be argued that creating all the lights in a $ws$-cluster as
one point does not introduce significant error. This intuition
is captured in the following theorem:

**Theorem 5.1.** For a point $p$ and a $ws$-cluster $C \subseteq S$, as-
sume that all lights in $C$ face in the same direction and they
have the same visibility from $p$. Then the error from repre-
senting $C$ with any one light in $C$ (which has the cumulative
intensity summed over all the lights in $C$) is bounded by a
function depending only on $\varepsilon$. In case the point to be shaded
has Lambertian BRDF, it is

$$|L(p, \omega) - \omega(C, p)| = O(\varepsilon) \sum_{s \in C} \frac{I_s}{d(p, s)^2}$$

(6)

where $L(p, \omega)$ denotes the exact illumination from lights in $C$.

**Proof is in the Appendix.**

In a $ws$-cluster, the distance between a point and any
cluster is $\varepsilon \cdot d(p, C)$.
that if the three-dimensional space was partitioned into equal-sized cubes of size $\delta$, the boundary of scene objects would intersect on average $O(\varepsilon^{-2})$ cubes in a ball of radius $\delta/\varepsilon$ (note that there are $O((\delta/\varepsilon)^3) = \Theta(\varepsilon^{-3})$ cubes in a ball of radius $\delta/\varepsilon$). Figure 6 shows that, for a variety of scenes, the average number of cells containing VPLs within a ball of radius $\delta/\varepsilon$ in our octree behaves more like $O(\varepsilon^{-2})$ than $O(\varepsilon^{-3})$. This implies better than cubic dependency on $1/\varepsilon$ of WSPD sizes in three dimensions, since the proof [CK95] uses a trivial cubic upper bound for non-empty cubes of size $\delta$ within a ball of radius $\delta/\varepsilon$. Experimental results confirm this: Figure 7 plots, for several scenes, the ratio of the WSPD size for varying separation parameter $\varepsilon$ to WSPD size for separation parameter 1. Observe that the behavior of the WSPD is relatively unchanged from one scene to the next.

**Constructing well-separated pair decomposition of $S$.** We use a compressed octree of $S$ as an underlying data structure to compute the initial $ws$-clusters for $S$, to find approximate nearest neighbors, and for the local refinement for each point $p$ during the rendering computation. The compressed octree is an octree where the non-branching paths are contracted into one edge. The compressed octree can be directly computed in linear time [Sam95]. Each node corresponds to an axis-aligned bounding box. We associate with each node the set of VPLs of $S$ contained in its bounding box. Note that each leaf of the octree contains exactly one unique point of $S$. For a node $w$, denote by $R_w$ the corresponding set of VPLs. Note that the height of the octree is linear in the worst-case, though in practice it is logarithmic. In Table 1 we show the depth of the tree for 300K VPLs, which is logarithmic for a variety of scenes.

<table>
<thead>
<tr>
<th>Scene</th>
<th>Tree depth</th>
<th>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conference</td>
<td>19</td>
<td>14</td>
</tr>
<tr>
<td>Sibenik</td>
<td>20</td>
<td>14</td>
</tr>
<tr>
<td>Museum</td>
<td>18</td>
<td>15</td>
</tr>
<tr>
<td>San Miguel</td>
<td>22</td>
<td>14</td>
</tr>
</tbody>
</table>

Table 1: Octree depths with 320K VPLs, both with and without compression.

After constructing the compressed octree, Algorithm 1 computes the WSPD of $S$ by utilizing a top down search on the tree for $ws$-pairs. Two nodes $w,v$ of the octree will form a $ws$-pair if the corresponding sets $R_w,R_v$ are well-separated. Note that instead of simply storing all the pairs of the WSPD as a list, we directly store the WSPD structure in the octree by storing, for each node $w$ of the octree, a list of pointers to all the other nodes with which $w$ forms $ws$-pairs in the WSPD. We denote this list by $pairs(w)$. In Figure 8 we show a simple example for 2 dimensional data with the red links denoting the pairs of sets. The construction ensures a hierarchical structure on the clusters which
Pre-computing of points.

Algorithm 1 Create WSPD for the set of points $S$

1: function CREATEWSPD($S$)
2:     \textbf{root} $\leftarrow$ create compressed octree on $S$
3:     $S$: stack of pairs, $S.pushback(\{\textbf{root}, \textbf{root}\})$
4:     \textbf{while} notEmpty($S$) \textbf{do}
5:       $(w, v) = S.pop()$
6:       \textbf{if} isWellSeparated($R_w, R_v$) \textbf{then}
7:           \textbf{Insert} v into pairs($w$)
8:           \textbf{Insert} w into pairs($v$)
9:       \textbf{else}
10:          \textbf{if} $r(R_w) < r(R_v)$ \textbf{then}
11:             \textbf{Swap}(w, v)
12:       \textbf{end if}
13:       \textbf{for} all $i \in$ children($w$) \textbf{do}
14:           $S.pushback(\{i, v\})$
15:       \textbf{end for}
16:     \textbf{end if}
17: \textbf{end while}
18: \textbf{return}

Algorithm 2 Constructing $w$-clusters $C_p$ for $p \in P$.

1: function CONSTRUCTWSPDCLUSTERING($p$)
2:     $C_p \leftarrow \emptyset$
3:     $s \leftarrow$ closest point in $S$ to $p$
4:     $C_s \leftarrow$ $w$-clustering of $s$
5: \textbf{for} all $Q \in C_s$ \textbf{do}
6:       \textbf{if} $d(s, Q) \geq d(p, s) - \varepsilon$ \textbf{then}
7:           add $Q$ to $C_p$
8:       \textbf{else}
9:           \textbf{refine} $Q$ into subclusters that are $w$ from $p$
10:          add the resulting clusters to $C_p$
11:       \textbf{end if}
12: \textbf{end for}
13: \textbf{return} $C_p$
14: \textbf{end function}

enables us to easily refine a cluster if needed by simply descending in the octree.

Figure 8: The compressed octree and the WSPD for a set of points.

Pre-computing $w$-clusterings of $S$. From the WSPD of $S$ one can compute a set of clusterings $\{C_s \mid s \in S\}$, where $C_s$ is a $w$-clustering of $S$ for the point $s \in S$. Namely, for each $s \in S$, $C_s$ is a partition of the lights in $S \setminus \{s\}$ into a number of clusters, each of which is well-separated from $s$. Note that if the two sets $\{R, Q\}$ are well-separated, then for every point $p \in R$, $Q$ is a $w$-cluster with respect to $p$. The definition of WSPD ensures that for every pair of points $p_1$ and $p_2$, there is a unique $\{R, Q\}$ such that $p_1 \in R$ and $p_2 \in Q$. Therefore $C_s = \{Q \mid \{R, Q\} \text{ is } w\text{-clustering of } S \text{ for } s\}$ is a $w$-clustering of $S$ for $s$. If the WSPD has been computed, then a $w$-clustering for each point $s \in S$ can be extracted from it efficiently, as follows. Consider the leaf node of the compressed octree corresponding to $s$. Any node $w$ of the octree on the path from this leaf to the root has $s \in R_w$, and so the $w$-clustering of $s$ is simply the union of pairs($w$) for all such nodes $w$, and can be computed by traversing the octree from this leaf to the root.

Computing $w$-clusterings of $P$ during rendering. We now show how to use the clusterings $C_s$, pre-computed for each $s \in S$ before the rendering phase, to quickly compute a $w$-clustering of $S$ with respect to any point $p \in P$.

Consider the case for an arbitrary point $p \in P$. Compute the closest light, say $s \in S$, to $p$. One could use a variety of known optimal algorithms, but for us an approximation will suffice. We find the smallest node of the compressed octree containing $p$ and return an arbitrary light contained in it. A calculation shows that for randomly shifted octrees, the expected distance from the true nearest neighbor is bounded. Point location in a compressed octree takes $O(\log n)$ time with some additional data structures, but for us the naive implementation suffices as the tree has logarithmic depth (see Table 1).

Say $s$ is at distance $d$ from $p$. Take the $w$-clustering $C_s$ of $s$. These were pre-computed, and can be efficiently retrieved. The key idea now is to consider two types of clusters in $C_s$ separately: far clusters in $C_s$ are at distance further than $d/\varepsilon$ from $s$ and close clusters are those closer than $d/\varepsilon$ from $s$. We show that each far cluster in $C_s$ is an approximately $w$-cluster from $p$. For the remaining close clusters in $C_s$, we recursively subdivide them until they are $w$-clusters from $p$. The subdivision uses the same octree that was used for the construction of the WSPD. See Algorithm 2. Note that the above algorithm is adaptive to the local geometry of the scene: for a point $p$ closely surrounded by VPLs, it will refine at a smaller radius.

We now prove that the clusters far from $s$ are approximately $w$-clusters from $p$.
Lemma 5.1. Let \( p \) be an arbitrary point and \( s \) be its nearest-neighbor with \( d(p, s) = d \). Any ws-cluster \( C_0 \in C \) disjoint with the ball of radius \( \frac{d}{2} \) around \( s \) is approximately well-separated from \( p \), namely it is well-separated with separation parameter \( \varepsilon' = \frac{\varepsilon}{1 - \varepsilon} \).

Proof is in the Appendix.

We next prove that the additional number of clusters added in the refinement of a cluster close to \( s \) is low.

Theorem 5.2. Let \( p \) be an arbitrary point and \( s \) be its nearest neighbor with \( d(p, s) = d \). After refining the clusters in the set \( C^* \subseteq C \) which intersect the ball of radius \( \frac{d}{2} \) around \( s \), there are at most \( O\left(\frac{1}{\varepsilon^2}\right) \) new ws-clusters \( C' \) created. The resulting set \( C_i \setminus C^* \cup C' \) is a partition of \( S \) into ws-clusters around \( p \), with separation parameter \( \varepsilon/(1 - \varepsilon) \).

Proof is in the Appendix.

While the above theorem may be surprising at first glance, we hope the following intuition sheds some light: consider the distance of the closest point \( s \in S \) to \( p \). If this distance is small, then it is not hard to argue that the clusters for the closest light provide a good approximation of the clustering for \( p \), and so little refinement is necessary. On the other hand, if this closest distance is large, then all points of \( S \) are ‘far’ from \( p \), and so any ws-cluster from \( s \) is far from \( p \) and thus approximately well-separated from \( p \); again little refinement is needed to approximate the separation (and thus illumination). This intuition is formalized in the proof of the theorem. For empirical validation, see Table 2 for the maximum number of new clusters added per point for a number of scenes with varying values of \( \varepsilon \).

<table>
<thead>
<tr>
<th>Refined clusters for varying ( \varepsilon )</th>
<th>0.9</th>
<th>0.7</th>
<th>0.5</th>
<th>0.3</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Conference</td>
<td>58</td>
<td>75</td>
<td>114</td>
<td>227</td>
<td>1401</td>
</tr>
<tr>
<td>Sibenik</td>
<td>57</td>
<td>74</td>
<td>109</td>
<td>219</td>
<td>1505</td>
</tr>
<tr>
<td>Museum</td>
<td>81</td>
<td>103</td>
<td>159</td>
<td>326</td>
<td>2350</td>
</tr>
<tr>
<td>San Miguel</td>
<td>80</td>
<td>108</td>
<td>167</td>
<td>370</td>
<td>3411</td>
</tr>
</tbody>
</table>

Table 2: Maximum number, over all \( p \in P \), of added clusters during refinement with 320K VPLs.

Cluster sizes. We have proved that the clustering stored in the WSPD can be used to retrieve clustering for arbitrary points without increasing the number of clusters more than an additive constant. It remains to argue that the initial clustering for every light \( s \in S \) is compact. While one can construct examples of arbitrary points where the average number of ws-clusters for a point \( p \) is linear (as a function of \(|S|\)), those are never realized in practice for the set of lights arising in geometric scenes. The spatial partitioning structures (octrees) turn out to be roughly balanced, and so the number of ws-clusters is logarithmic.

6 Additional structures for illumination computation

We enhance the purely geometric WSPD based clustering with the following additional structures that improve the efficiency and accuracy in calculating illumination. Recall that given a ws-clustering \( C_i \) for \( p \), for a ws-cluster \( C_i \in C \), the approximation of the radiance with a single representative has the form:

\[
M_{\text{rep}(C_i)}(p, \omega) \cdot G_{\text{rep}(C_i)}(p) \cdot V_p(\text{rep}(C_i)) \cdot \sum_{s \in C_i} I_s \tag{7}
\]

Clustering refinement by direction. The WSPD data structure is able to efficiently bound angles and distances between points. However, the normals of the lights could vary widely in directions. To overcome this difficulty, the lights in each ws-cluster are grouped into a few subgroups with similar normals. For a ws-cluster \( C_i \), construct the subgroups \( C_i^1, \ldots, C_i^{K_i} \), where all the lights in each \( C_i^j \), \( j = 1 \ldots K_i \), will have similar normal directions. The approximation then becomes

\[
L_{C_i}(p, \omega) = \left( \sum_{j=1}^{K_i} L_{C_i^j}(p, \omega) \right) \cdot V_p(\text{rep}(C_i)) \tag{8}
\]

where \( L_{C_i^j}(p, \omega) \) is defined in Equation 4. The subgroups are constructed by first picking a center \( c^j \in C_i \) for each subgroup \( C_i^j \), and then assigning each VPL \( s \in C_i \) to the subgroups with most similar center. see Algorithm 3

Algorithm 3 Computing subgroups of a given cluster \( C_i \)

1: function CLUSTERNORMALS(\( C_i \))
2: \( j \leftarrow 1; c^1 \leftarrow \text{random light in } C_i \)
3: largestDistance \( \leftarrow \infty \)
4: while largestDistance \( \geq \) threshold do
5: \( \text{For each } s \in C_i, d_{ij} \leftarrow \min_{k \leq j} d(s, c^k) \)
6: \( c^{j+1} \leftarrow q, \text{where } q \text{ has largest } d_{q} \text{ value} \)
7: \( j \leftarrow j + 1 \)
8: largestDistance \( \leftarrow d_{q} \)
9: end while
10: \( \text{For each } i, C_i^j \leftarrow \{ s \in C_i | l = \arg\min_{k < j} d(s, c^k) \} \)
11: end function

The shading of a point \( p \in P \), as before, visibility test will still be performed once for each ws-cluster of \( p \). However, the radiance \( L_{C_i}(p, \omega) \) for a ws-cluster \( C_i \) will be calculated by summing up the radiance contributions separately for each subgroup \( C_i^j \) of \( C_i \), using the normal of the center light for each subgroup. The distance \( d(s, c^j) \) used in the algorithm is the Euclidean distance with threshold set to 0.01, which did not result in too many subgroups on average (see Table 3). The number of subgroups is slightly higher for more complex scenes and decreases with \( \varepsilon \) since the clustering becomes more fine.
Table 3: Average number of subgroups per cluster.

<table>
<thead>
<tr>
<th>Scene</th>
<th>Average number of subgroups</th>
</tr>
</thead>
<tbody>
<tr>
<td>ε</td>
<td>0.9 0.7 0.5 0.3</td>
</tr>
<tr>
<td>Conference</td>
<td>8.3 8.1 6.7 5.3</td>
</tr>
<tr>
<td>Sibenik</td>
<td>9.5 9.3 8.0 5.9</td>
</tr>
<tr>
<td>Museum</td>
<td>23.3 21.2 16.8 11.4</td>
</tr>
<tr>
<td>San Miguel</td>
<td>44.72 39.6 31.2 18.5</td>
</tr>
</tbody>
</table>

Visibility testing. Visibility differences within a cluster can cause errors with a fixed representative light. Consider for example a flat object with VPLs on both sides. Since the representative light is either on one side or the other, visibility queries falsely return occlusion if the pixel is on the other side of the object. To overcome this problem we propose to augment our ws-clusters with additional visibility information.

As stated in Theorem 5.1 without taking into account visibility differences, a ws-clustering with a single representative and clustered normals gives a good approximation to $L$ (as a function of $\epsilon$). The visibility computation for a ws-cluster $C_i$ will be divided into two parts: a simple shadow test from $p$ to the boundary of the ball of $C_i$ for the outside visibility and then shadow testing from the boundary to each light for visibility inside the ball of $C_i$. Here again the geometric well-separated property of the ws-clustering comes useful, as the angles from $p$ to $C_i$ are bounded (as a function of $\epsilon$). We use a new approximation for $L(p, \omega)$, to better handle visibility:

$$L_{C_i}(p, \omega) = \left( \sum_{j=1}^{k_i} L_{C'_j}(p, \omega) \right) \cdot V_p(\partial(b(C'_j))) \cdot R(C_i, p) \quad (9)$$

$$R(C_i, p) = \frac{\sum_{s \in C_i} V_p(s, \partial(b(C'_i))) \cdot \omega(s) \cdot \cos \phi_s}{\sum_{s \in C_i} \omega(s) \cdot \cos \phi_s} \quad (10)$$

where $V_p(s, \partial(b(C'_j)))$ denotes the visibility from $s$ to the boundary of the sphere in the direction of the center of $b(C'_j)$, the shaded point $p$ and $\phi_s$ is the angle between the same direction and the normal of the light. Note that as $\epsilon \to 0$, equation (9) converges to $L(p, \omega)$. Since computing $R(C_i, p)$ during rendering would be expensive, we do the following in the pre-processing phase. For each cluster $C_i$ and for a small uniform set of directions on $\partial b(C'_j)$, $R(C_i, p)$ is pre-computed and stored in a cubemap (with a resolution of $6 \times 6$ on each side). This enables quick lookup of $R(C_i, p)$ for $p$. During the rendering of a point $p$, nearest-neighbor interpolation on the cubemap yields $R(C_i, p)$. A shadow test to $\partial b(C'_j)$ gives the $V_p$ term.

High intensity clusters. To further minimize the error coming from a badly chosen representative for high intensity clusters we limit the radiance from each cluster (by further refining the cluster if necessary) to be less than 1% of the radiance received by a pixel. This refinement happens at the pre-processing phase only using approximate intensities between the cluster pairs.

7 Results and discussion

In this section we present the experimental results on several scenes of varying complexity. Timings are for a workstation equipped with two Xeon X5570 processors each with 4 cores running at 2.93GHz and with 32 GB of memory. We compare our algorithm with two well-known methods: Lightcuts [WFA+05] and LightSlice [OPT]. The authors of LightSlice have made their code publicly available, which also includes an implementation of Lightcuts. In order to do fair comparisons between all three methods, we have ported their implementation of LightSlice (and Lightcuts) into the ray-tracing system INTEL EMBREE [WFWB13] without modifying the core of the algorithms. Our code is also written to use Embree as its ray-tracing engine.

Unless otherwise stated, we run LightSlice and Lightcuts with similar parameters as used in [OPT]: Lightcuts error bound is set to 1% and unbounded the maximum cut size. In order to compare our method to Lightcuts with the parameters set as in [WFA+05], we include results with 2% error and maximum cut size set to 2000. We use the version with 1% error threshold as the reference for equal quality comparisons. LightSlice is run with approximately 1400 slices and 400 columns (the number of slices determines the size of the reduced light transport matrix while the number of columns determines the number of clusters used per point). For our method, the user is free to set the separation parameter $\epsilon$. This parameter closely tracks both theoretically and practically the quality of the resulting image. In general, setting $\epsilon$ to 0.5 gives a good compromise between quality and speed.

One inherent disadvantage of the many-lights technique is the presence of certain artifacts due to the overly-high contribution of some VPLs to their neighboring points. The standard way to avoid these problems is by clamping: limiting the contribution of any one VPL within some small Euclidean distance by applying a clamping threshold. The bias introduced by this technique requires the image rendered with all VPLs to be used as our reference image and not the path traced one. Our experiments use 1 and 4 samples per pixel to compare the quality of clustering obtained by the different methods.

Scenes. We test the algorithms on standard collection of scenes with only moderately glossy materials. The Museum has specular materials like the bones of the dinosaur and the canvas on its stand. Most of the primary lights are facing the ceiling to ensure that the scene is mainly lit by indirect illumination. In Sibenik, the light sources are facing upwards
<table>
<thead>
<tr>
<th>Scenes</th>
<th>Museum</th>
<th>Sibenik</th>
<th>Conference</th>
<th>San Miguel</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Triangles</td>
<td>Resolution</td>
<td>VPLs</td>
<td></td>
</tr>
<tr>
<td>Lightcuts(1%) max cut: $\infty$</td>
<td>1.3M</td>
<td>1024x1024@1</td>
<td>540K</td>
<td>3226</td>
</tr>
<tr>
<td>Render time (s)</td>
<td>0.38</td>
<td>0.47</td>
<td>0.44</td>
<td>0.29</td>
</tr>
<tr>
<td>Avg # of rays</td>
<td>1872</td>
<td>1399</td>
<td>957</td>
<td>3226</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.004674</td>
<td>0.000394</td>
<td>0.002220</td>
<td>0.005828</td>
</tr>
<tr>
<td>LMSE</td>
<td>0.000735</td>
<td>0.001124</td>
<td>0.003228</td>
<td>0.003134</td>
</tr>
<tr>
<td>Rel. Error(%)</td>
<td>1.123480</td>
<td>1.094558</td>
<td>1.273182</td>
<td>2.617376</td>
</tr>
<tr>
<td>Preproc. time (s)</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
<td>1.0</td>
</tr>
<tr>
<td>Render time (s)</td>
<td>515.06</td>
<td>216.29</td>
<td>219.43</td>
<td>1583.63</td>
</tr>
<tr>
<td>Avg # of rays</td>
<td>884</td>
<td>724</td>
<td>556</td>
<td>714</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.006223</td>
<td>0.004401</td>
<td>0.003030</td>
<td>0.021054</td>
</tr>
<tr>
<td>LMSE</td>
<td>0.001155</td>
<td>0.001599</td>
<td>0.002794</td>
<td>0.019569</td>
</tr>
<tr>
<td>Rel. Error(%)</td>
<td>1.727831</td>
<td>1.864828</td>
<td>1.689145</td>
<td>8.528274</td>
</tr>
<tr>
<td>Preproc. time (s)</td>
<td>2.3</td>
<td>2.1</td>
<td>1.8</td>
<td>5.2</td>
</tr>
<tr>
<td>Render time (s)</td>
<td>218.81</td>
<td>103.31</td>
<td>125.98</td>
<td>306.39</td>
</tr>
<tr>
<td>Avg # of rays</td>
<td>665</td>
<td>795</td>
<td>750</td>
<td>463</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.009090</td>
<td>0.006820</td>
<td>0.003515</td>
<td>0.014771</td>
</tr>
<tr>
<td>LMSE</td>
<td>0.002120</td>
<td>0.001457</td>
<td>0.003124</td>
<td>0.028364</td>
</tr>
<tr>
<td>Rel. Error(%)</td>
<td>3.273127</td>
<td>3.142222</td>
<td>3.164744</td>
<td>13.855263</td>
</tr>
<tr>
<td>Preproc. time (s)</td>
<td>8.2</td>
<td>7.0</td>
<td>6.2</td>
<td>18.2</td>
</tr>
<tr>
<td>Render time (s)</td>
<td>62.93</td>
<td>31.10</td>
<td>35.24</td>
<td>86.91</td>
</tr>
<tr>
<td>Avg # of rays</td>
<td>711</td>
<td>532</td>
<td>642</td>
<td>715</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.007049</td>
<td>0.004366</td>
<td>0.003515</td>
<td>0.014771</td>
</tr>
<tr>
<td>LMSE</td>
<td>0.002025</td>
<td>0.001457</td>
<td>0.003124</td>
<td>0.028364</td>
</tr>
<tr>
<td>Rel. Error(%)</td>
<td>2.416696</td>
<td>2.148856</td>
<td>2.392963</td>
<td>7.539747</td>
</tr>
<tr>
<td>Preproc. time (s)</td>
<td>19.28</td>
<td>18.21</td>
<td>18.24</td>
<td>58.11</td>
</tr>
<tr>
<td>Render time (s)</td>
<td>86.72</td>
<td>37.85</td>
<td>42.33</td>
<td>115.87</td>
</tr>
<tr>
<td>Avg # of rays</td>
<td>901</td>
<td>635</td>
<td>665</td>
<td>915</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.005750</td>
<td>0.004254</td>
<td>0.003515</td>
<td>0.014771</td>
</tr>
<tr>
<td>LMSE</td>
<td>0.002120</td>
<td>0.001457</td>
<td>0.003124</td>
<td>0.028364</td>
</tr>
<tr>
<td>Rel. Error(%)</td>
<td>1.928312</td>
<td>1.821888</td>
<td>2.180206</td>
<td>6.760511</td>
</tr>
<tr>
<td>Preproc. time (s)</td>
<td>5.9</td>
<td>5.7</td>
<td>5.2</td>
<td>13.7</td>
</tr>
<tr>
<td>Render time (s)</td>
<td>76.86</td>
<td>64.31</td>
<td>60.41</td>
<td>232.91</td>
</tr>
<tr>
<td>Avg # of rays</td>
<td>1913</td>
<td>1183</td>
<td>1318</td>
<td>6972</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.006452</td>
<td>0.002833</td>
<td>0.002219</td>
<td>0.005864</td>
</tr>
<tr>
<td>LMSE</td>
<td>0.000400</td>
<td>0.001822</td>
<td>0.002624</td>
<td>0.004080</td>
</tr>
<tr>
<td>Rel. Error(%)</td>
<td>1.439894</td>
<td>1.261353</td>
<td>1.597909</td>
<td>2.886908</td>
</tr>
<tr>
<td>Preproc. time (s)</td>
<td>2.7</td>
<td>3.2</td>
<td>2.7</td>
<td>1.6</td>
</tr>
<tr>
<td>Render time (s)</td>
<td>190.27</td>
<td>68.00</td>
<td>80.28</td>
<td>950.88</td>
</tr>
<tr>
<td>Avg # of rays</td>
<td>1913</td>
<td>1183</td>
<td>1318</td>
<td>6972</td>
</tr>
<tr>
<td>RMSE</td>
<td>0.004652</td>
<td>0.002833</td>
<td>0.002219</td>
<td>0.005864</td>
</tr>
<tr>
<td>LMSE</td>
<td>0.000400</td>
<td>0.001822</td>
<td>0.002624</td>
<td>0.004080</td>
</tr>
<tr>
<td>Rel. Error(%)</td>
<td>1.439894</td>
<td>1.261353</td>
<td>1.597909</td>
<td>2.886908</td>
</tr>
<tr>
<td>Preproc. time (s)</td>
<td>2.7</td>
<td>3.2</td>
<td>2.7</td>
<td>1.6</td>
</tr>
<tr>
<td>Render time (s)</td>
<td>190.27</td>
<td>68.00</td>
<td>80.28</td>
<td>950.88</td>
</tr>
<tr>
<td>Avg # of rays</td>
<td>1913</td>
<td>1183</td>
<td>1318</td>
<td>6972</td>
</tr>
</tbody>
</table>

Table 4: Rendering statistics and images for the three different methods, Lightcuts, LightSlice and the WSPD algorithm.
in the dome such that the scene is again lit by indirect illumination, and is made up of purely diffuse materials. The big uniformly colored surfaces in the Conference are challenging since the clustering methods have to be spatially consistent, with moderately shiny materials. The outdoor scene San Miguel is our largest scene consisting of 10M triangles lit by an environment map of sunset. This is our most challenging scene since the area under the tree and in the corridors is mostly lit by indirect illumination with lots of smooth shadows.

**Performance.** The images are rendered at a 1024 × 1024 resolution with 1 sample per pixel (spp) and with approximately 500K VPLs. We provide the running times for the average number of shadow rays per pixel.

We provide three different error metrics. Denote by $F(x,y,c)$ the value of a color channel $c$ in the image at coordinate $(x,y)$ and by $\hat{F}(x,y,c)$ the same value in the reference image. The number of pixels multiplied by the number of color channels is $m = 3 \cdot 1024^2 \cdot 1024$ and each value of $F$ and $\hat{F}$ is between 0 and 1.

- The normalized $\text{RMSE}$ provides numerical difference against the VPL reference image:
  \[
  \text{RMSE} = \sqrt{\frac{1}{m} \sum_{x,y,c} (F(x,y,c) - \hat{F}(x,y,c))^2}
  \]
  where the summation is over all pixels and color channels of the images.

- The $\text{LMSE}$ represents the average squared difference of the gradients between the rendered image and the reference $\text{SPA07}$:
  \[
  \text{LMSE} = \sum_{x,y,c} \left( \nabla F(x,y,c) - \nabla \hat{F}(x,y,c) \right)^2 / \sum_{x,y,c} \nabla F(x,y,c)
  \]
  where $\nabla F(x,y,c) = (F(x+1,y,c) + F(x-1,y,c) + F(x,y+1,c) + F(x,y-1,c) - 4F(x,y,c))$. A high $\text{LMSE}$ error implies sharp discontinuities (e.g., sharp error edges), identifying more noticeable errors.

- Average relative error is given by:
  \[
  \text{Rel. Error} = \frac{100}{m} \sum_{x,y,c} \frac{|F(x,y,c) - \hat{F}(x,y,c)|}{\hat{F}(x,y,c)}.
  \]
  The error images are calculated by taking the channel-wise Euclidean distance between the image, and the VPL reference image, and multiplying it by a factor of 32.

Table 4 shows the results with all these statistics. In general, we find that with $\varepsilon = 0.25$, the quality of our results is similar to Lightcuts with around 3× speedup (the last row in Table 4). The average number of shadow rays for WSBD can be larger than that of Lightcuts; however, as proved earlier, almost no other computation except visibility testing is done by the WSBD algorithm. The WSBD method solely relies on the pre-computed pairs which results in a shorter rendering time since there is no additional work done. Lightcuts, on the other hand, has to descend the tree, maintaining an expensive heap data structure during this traversal. The cost of calculating upper bounds during rendering is also significant. Bounding the maximum cut size can result in significant loss of quality unless the ideal cut size is known a priori (e.g., as in San Miguel). Considering this run as a reference for equal quality comparison, our method with $\varepsilon = 0.5$ still shows similar or even better quality with around 3 times speed-up (e.g., as in Museum).

LightSlice is able to explore the structure of VPLs and adapt to it more efficiently than Lightcuts, but dividing the image into slices results in visually disturbing blocking artifacts if the error is not low enough. This is captured by the high errors (especially the LMSE) of LightSlice compared with WSBD in all the scenes. The WSBD method locally adjusts the cluster radius based on the well-separated criteria. Thus the errors in the resulting image are smoothly distributed, with visually minimal artifacts. The value of the parameter $\varepsilon$ closely tracks the quality; for scenes with complex shadows like San Miguel, $\varepsilon$ has to be set lower (0.1) for comparable quality to Lightcuts. LightSlice relies also on a fixed parameter (number of columns, set to 400). For San Miguel it is unable to adapt to the complexity of the shadows with such a low number of columns, resulting in faster running times with high errors.

**Scalability.** See Table 5 for the total memory (GB) used by the three methods. Lightcuts is the most efficient on memory consumption, followed by WSBD. LightSlice, due to the light transport matrix storage, has prohibitively high memory consumption.

<table>
<thead>
<tr>
<th># VPLs:</th>
<th>75K</th>
<th>115K</th>
<th>200K</th>
<th>375K</th>
<th>776K</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lightcuts</td>
<td>0.59</td>
<td>0.63</td>
<td>0.74</td>
<td>0.84</td>
<td></td>
</tr>
<tr>
<td>LightSlice</td>
<td>4.53</td>
<td>6.31</td>
<td>9.85</td>
<td>18.81</td>
<td>36.58</td>
</tr>
<tr>
<td>WSBD</td>
<td>0.5</td>
<td>1.14</td>
<td>1.40</td>
<td>1.98</td>
<td>3.38</td>
</tr>
</tbody>
</table>

Table 5: Memory requirements for the Museum scene (GB).

In Figure 10, we plot the rendering times of the three methods with varying number of VPLs in the Museum scene. Note how our algorithm consistently outperforms Lightcuts even with 3M VPLs and both of them scale sub-linearly in the number of lights. LightSlice is only usable as long as it does not allocate more memory than the system has.

**Trade-off:** In Figure 11, we plot the relative error and the rendering time against $\varepsilon$ in the Museum scene. The curve is not strictly monotonic since our algorithm is not deterministic (e.g., approximate nearest neighbor).

**Blocking artifacts.** In contrast to LightSlice, our method and Lightcuts currently do not take advantage of using different representative lights for each sample per pixel. We have found that increasing the number of samples increases...
Figure 9: Varying number of VPLs for the Museum scene.

Figure 10: Relative error (red) and render time (green) with varying ε for the Museum scene.

Figure 11: Part of the Museum scene rendered with different values of samples per pixel.

8 Limitation and future work

We have showed that the WSPD structure is suitable for compactly storing clustering information and for providing fast extraction of clusters during render time. We showed theoretical bounds on the error for diffuse materials. The data structure proposed in this paper gives a new perspective on how to efficiently store and retrieve a view-independent clustering for scenes. This framework is very flexible and leaves several possibilities for improving and enhancing the current solution.

One of the main limitations of our method is that it is suited towards diffuse surfaces, and the quality decreases with highly glossy surfaces. The upper-bound proved in Theorem 5.1 increases as a function of glossiness, and so higher glossiness requires smaller values of ε in the WSPD construction for similar error upper-bounds. One can compensate for it with decreasing ε, at the loss of efficiency. In order to demonstrate this, we have replaced the Phong BRDFs in the Conference scene with Blinn microfacet BRDFs. This BRDF results in a very significant contribution from the direction of the reflected view ray. See Figure 13 for the results with 1 sample per pixel. The most prominent error is around the shadows of the chairs on the highly glossy floor where the exponent for the Blinn microfacet BRDF is set to 100. Lightcuts can more efficiently adapt to highly glossy materials than our method.

A similar problem is the need to use a small ε for scenes with highly varying visibility properties since in this case the number of clusters increases globally without only refining the clustering where it is necessary. To overcome these
Table 6: Rendering statistics and images for 4 samples per pixel.

<table>
<thead>
<tr>
<th>Scene</th>
<th>Museum</th>
<th>Conference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangles</td>
<td>1.5M</td>
<td>0.33M</td>
</tr>
<tr>
<td>Resolution</td>
<td>1024x1024@4</td>
<td>1024x1024@4</td>
</tr>
<tr>
<td>VPLs</td>
<td>474K</td>
<td>516K</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lightcuts</th>
<th>RMSE</th>
<th>LMSE</th>
<th>Rel. Error(%)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preproc. time (s)</td>
<td>0.48</td>
<td>0.44</td>
<td>0.905316</td>
<td>1.0</td>
</tr>
<tr>
<td>Render time (s)</td>
<td>2080.97</td>
<td>876.96</td>
<td>1.208336</td>
<td>1.0</td>
</tr>
<tr>
<td>Avg # of rays</td>
<td>7544</td>
<td>3828</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMSE</td>
<td>0.003374</td>
<td>0.002119</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMSE</td>
<td>0.001008</td>
<td>0.003789</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rel. Error(%)</td>
<td>0.905316</td>
<td>1.208336</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speedup</td>
<td>1.0</td>
<td>1.0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Lightslice</th>
<th>RMSE</th>
<th>LMSE</th>
<th>Rel. Error(%)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preproc. time (s)</td>
<td>0.01</td>
<td>0.02</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Render time (s)</td>
<td>344.73</td>
<td>262.89</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg # of rays</td>
<td>1696</td>
<td>1792</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMSE</td>
<td>0.005662</td>
<td>0.003025</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMSE</td>
<td>0.005431</td>
<td>0.005524</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rel. Error(%)</td>
<td>1.836033</td>
<td>2.328994</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speedup</td>
<td>6.0</td>
<td>3.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WSPD 0.7</th>
<th>RMSE</th>
<th>LMSE</th>
<th>Rel. Error(%)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preproc. time (s)</td>
<td>63.47</td>
<td>40.27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Render time (s)</td>
<td>265.25</td>
<td>158.72</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg # of rays</td>
<td>2660</td>
<td>2528</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMSE</td>
<td>0.006984</td>
<td>0.003378</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMSE</td>
<td>0.002270</td>
<td>0.004733</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rel. Error(%)</td>
<td>2.318444</td>
<td>2.425038</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speedup</td>
<td>7.8</td>
<td>5.5</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WSPD 0.5</th>
<th>RMSE</th>
<th>LMSE</th>
<th>Rel. Error(%)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preproc. time (s)</td>
<td>68.38</td>
<td>43.25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Render time (s)</td>
<td>356.20</td>
<td>165.58</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg # of rays</td>
<td>3728</td>
<td>2588</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMSE</td>
<td>0.005661</td>
<td>0.003029</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMSE</td>
<td>0.001876</td>
<td>0.005271</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rel. Error(%)</td>
<td>1.809967</td>
<td>2.099148</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speedup</td>
<td>5.8</td>
<td>5.3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WSPD 0.3</th>
<th>RMSE</th>
<th>LMSE</th>
<th>Rel. Error(%)</th>
<th>Speedup</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preproc. time (s)</td>
<td>84.68</td>
<td>56.42</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Render time (s)</td>
<td>646.19</td>
<td>260.38</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Avg # of rays</td>
<td>6228</td>
<td>4192</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RMSE</td>
<td>0.005458</td>
<td>0.002664</td>
<td></td>
<td></td>
</tr>
<tr>
<td>LMSE</td>
<td>0.001481</td>
<td>0.004486</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rel. Error(%)</td>
<td>1.563184</td>
<td>1.717990</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Speedup</td>
<td>3.2</td>
<td>3.4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 13: Part of the Conference scene with highly glossy floor using 1 sample per pixel.

limitations one might consider the following improvements:

- Currently our definition of well-separatedness is purely geometric, without taking into account the visibility properties of the clusters. It would be interesting if the pre-processing phase can use visibility queries to adaptatively guide the construction of cluster pairs.

- One of the strengths of our approach is that the pre-processing phase already computes pairs of clusters, which are then modified in a limited manner during the rendering phase. Furthermore this computation is view-independent. This opens up the possibility of reusing this computation with changing camera position,
towards animations, interactive rendering, or even the final goal of real-time global illumination.

9 Acknowledgments

We thank the following people for the models. Greg Ward for the Conference scene, Alvaro Luna Bautista and Joel Andersdon for the Museum scene, Guillermo M. Leal Laguno for the San Miguel scene and Marko Dabrovic for the Sibenik Cathedral scene.

References


10 Appendix

10.1 Proofs

In this section we give the proofs of Theorem 5.1, 5.2 and Lemma 5.3.

Theorem 5.1. First we show $\varepsilon$-dependent bounds of the changes of angles and distances between $p$ and the lights in a $ws$-cluster $C$. See Figure 14 for the interaction of $p$ with two lights $s, r$ in $C$ where $r$ denotes the light chosen as the representative of the cluster.

![Figure 14: Changes in the angles and distances are bounded by a function of $\varepsilon$.](image)

Estimating the change in the distances follows immediately from the $ws$ property:

\[
d(p, s) \leq d(p, r) + 2r(C) \leq d(p, r) + 2\varepsilon d(p, s) \tag{11}
\]

For bounding the change of angles $\Delta \theta$, it is clear that the angle between the two tangents of the circle from $p$ is the upper bound. The center of the circle, $p$, and the point on tangency form a right angle triangle:

\[
\sin \frac{\Delta \theta}{2} = \frac{r(C)}{d(p, C) + r(C)} \leq \frac{\varepsilon d(p, C)}{d(p, C)} \leq \varepsilon \tag{12}
\]

Hence using the approximation for the $\sin$ function with Taylor series we have $\Delta \theta = O(\varepsilon)$.

In the following we approximate the error for one cluster using the previous results. We assume that the visibility is always 1 (the case of 0 is trivial). Therefore the error is:

\[
|L(p, \omega) - L_C(p, \omega)| = \left| \sum_{s \in C} V_p(s) I_M(p, \omega) G_s(p) - V_p(r) M_r(p, \omega) G_r(p) \right| \tag{13}
\]

\[
= \left| \sum_{s \in C} I_s [M_s(p, \omega) G_s(p) - M_r(p, \omega) G_r(p)] \right| \tag{14}
\]

The above formula has the important property that the $M_s$ and $G_s$ functions are dependent on the cosines of angles and distances which are closely bounded because of the $ws$ property. This result intuitively means that the error cannot be too big for a BRDF that relies on distances and angles. In the case of the diffuse BRDF, $M_s(p, \omega) = k_d(p) \cos \theta_s$ where $k_d(p)$ is the diffuse reflection coefficient and $\theta_s$ is the angle between the surface normal at $p$ and $s - p$. Denote by $\phi_s$...
the angle between the light normal and \( p - s \) and for brevity denote \( d(p,s) \) by \( r_s \), then the formula becomes:

\[
\sum_{s \in C} I_s \left| \frac{k_d(p) \cos \theta_s \cos \phi_s}{r_s^2} - \frac{k_d(p) \cos \theta_s \cos \phi_s}{r_s^2} \right| = (16)
\]

\[
k_d(p) \sum_{s \in C} I_s \left| \frac{\cos \theta_s \cos \phi_s}{r_s^2} - \frac{\cos(\theta_s + \Delta \theta_s) \cos(\phi_s + \Delta \phi_s)}{(r_s + \Delta r_s)^2} \right| \]

(17)

We omit a complete analysis and just minimize the subtrahend assuming that \( \Delta \theta, \Delta \phi, \Delta r \geq 0 \) (the other cases are similar). Using the\( ws \) property and that \( \cos(\theta + \Delta \theta) \geq \cos \theta - \Delta \theta \),

\[
\leq k_d(p) \sum_{s \in C} I_s \left| \frac{\cos \theta_s \cos \phi_s}{r_s^2} - \frac{(\cos \theta_s - \Delta \theta_s)(\cos \phi_s - \Delta \phi_s)}{(1 + 2\varepsilon)^2 r_s^2} \right|
\]

(18)

Using that \( \cos x \leq 1 \) and the bound on \( \Delta \theta \)

\[
\leq k_d(p) \sum_{s \in C} I_s \left( \frac{4\varepsilon + 4\varepsilon^2 + \Delta \theta_s + \Delta \phi_s - \Delta \theta_s \Delta \phi_s}{(1 + 2\varepsilon)^2 r_s^2} \right) = O(\varepsilon)k_d(p) \sum_{s \in C} I_s
\]

(19)

(20)

In other words, the error is proportional to the complete intensity received by a pixel. Note that this does not account for visibility difference of course, which we address with other methods.

Remark: In the case of Phong BRDF the \( M_r(p, \omega) \) function becomes \( k_r(p) \cos^2 \beta_n \cos \theta_n \) where \( \beta_n \) is the angle between \( \omega \) and the direction of \( s - p \) reflected on the surface normal (the subscript of \( k \) now refers to the word specular). In this case the above calculations are as follows:

\[
\sum_{s \in C} I_s \left| \frac{k_r(p) \cos^2 \beta_n \cos \theta_n \cos \phi_s}{r_s^2} - \frac{k_r(p) \cos^2 \beta_n \cos \theta_n \cos \phi_s}{r_s^2} \right| = k_r(p) \sum_{s \in C} I_s \left| \frac{\cos^2 \beta_n \cos \theta_n \cos \phi_s}{r_s^2} - \frac{\cos^2(\beta_n + \Delta \beta_n) \cos(\theta_n + \Delta \theta_n) \cos(\phi_s + \Delta \phi_s)}{(r_s + \Delta r_s)^2} \right|
\]

(21)

\[
\leq k_r(p) \sum_{s \in C} I_s \left| \frac{\cos^2 \beta_n \cos \theta_n \cos \phi_s}{r_s^2} - \frac{(\cos \beta_n - \Delta \beta_n)^2(\cos \theta_n - \Delta \theta_n)(\cos \phi_s - \Delta \phi_s)}{(1 + 2\varepsilon)^2 r_s^2} \right|
\]

(22)

One could upper bound this error but the subtrahend converges to 0 as \( n \rightarrow \infty \) hence in this case one could only give a weak bound which depends on \( n \) as well. If we were to set \( \varepsilon \) according to such a bound one would have to build a too fine WSPD which would result in a prohibitively big runtime. We remark that this could be possibly overcome by refining the WSPD with a smaller epsilon during the rendering phase but only for those clusters that have a high value for \( M_r(p, \omega) \). However we have not experimented with this approach.

\[\text{Figure 15: A close } ws\text{-cluster } C_i \text{ and a far } ws\text{-cluster } C_o \text{ from } s.\]

Lemma[5.7] See Figure [15] As \( C_o \) is \( ws \) from \( s \), it follows that \( ed_o \geq r(C_o) \), where \( r(C_o) \) is the radius of the cluster \( C_o \). Also \( d \leq ed_o \) since the cluster is disjoint. Triangle inequality implies \( d(s,C_o) = d_o \leq d + d(p,C_o) \), and so \( d(p,C_o) \geq d_o - d \geq d_o(1 - \varepsilon) \geq r(C_o)(1 - \varepsilon) \). Thus for the slightly larger value of the separation parameter \( \varepsilon' = \frac{r(C_o)}{1 - \frac{1}{\varepsilon}} \).

\[ C_o \text{ is } ws \text{ from } p. \]

Theorem[5.8] The second statement of the proof comes from Lemma[5.1] Take a cluster \( C_i \in C \) lying inside the ball of radius \( d/\varepsilon \) around \( s \). If it is not \( ws \)-separated from \( p \), partition the bounding-box of \( C_i \) into 8 equal-volume bounding boxes (the children of the node in the octree), and recursively check the \( ws \)-separated property of these new refined clusters with \( p \). Eventually when a newly refined cluster is finally a \( ws \)-cluster from \( p \), add it to \( C' \). To count the total number of new \( ws \)-clusters added to \( C' \), consider a cluster \( C \in C' \). It exists because its parent, say cluster \( D \), was not a \( ws \)-cluster with \( p \). i.e., \( r(D) > \varepsilon d_p \), where \( d_p \) is the distance of the ball of \( D \) to \( p \). Because \( s \) is the nearest neighbor of \( p \) we know that no other point is in the small ball of radius \( d \) around \( p \); in particular it cannot completely contain the ball of \( D \) and hence \( r(D) / \varepsilon \geq d_p \geq d - 2r(D) \), which implies that \( r(D) \geq \frac{d}{2(1 - \frac{1}{\varepsilon})} \). So each cluster added to \( C' \) has a parent with radius at least the above value. Grouping the parents by size (higher level parents are the same size but multiplied by some power of two) we can give an upper bound on their number by a simple packing argument since parents with the same size are disjoint (since octree nodes can either contain each other or be disjoint):

\[
\sum_{i=0}^{\infty} \left( \frac{d/\varepsilon}{2(1 - \frac{1}{\varepsilon})} \right)^3 = O\left( \frac{1}{\varepsilon^3} \right)
\]

(23)

Since we have bounded the number of parents and each of them can have at most 8 children, this finishes the proof.