Eurographics Symposium on Rendering 2012
Fredo Durand and Diego Gutierrez
(Guest Editors)

*Volume 31 (2012), Number 4*

# Quantized Point-Based Global Illumination

Bert Buchholz Tamy Boubekeur
Telecom ParisTech – CNRS

**Abstract**

*Point-based global illumination (PBGI) uses a dense point sampling of the scene's surfaces to approximate indirect light transport and is intensively used in 3D motion pictures and special effects. Each point caches the reflected light using a spherical function and is typically used in a subsequent rasterization process to compute color bleeding and ambient occlusion in an economic, noise-free fashion. The entire point set is organized in a spatial tree structure which models the light transport hierarchically, enabling fast adaptive shading on receivers (e. g., unprojected pixels). One of the major limitations of PBGI is related to the size of this tree, which can quickly become too large to fit in memory for complex scenes. However, we observe that, just as with natural images, this point data set is extremely redundant. In this paper, we present a new method exploiting this redundancy by factorizing PBGI data over the tree nodes. In particular, we show that a k-means clustering in the parameter space of the spherical functions allows to define a small number of representative nodes against which any new one can be classified. These representative functions, gathered in a pre-process over a subset of the actual points, form a look-up table which allows to substitute node's data by quantized integers in a streaming process, avoiding building the full tree before compressing it. Depending on the nodes' spherical function variance in the scene and the desired accuracy, our indexed PBGI representation achieves between one and two orders of magnitude compression of the nodes spherical functions, with negligible numerical and perceptual error in the final image. In the case of a binary tree with one surfel per leaf and no spherical functions in the leaves, this leads to compression rates ranging from 3x to 5x for the whole tree.*

## 1. Introduction

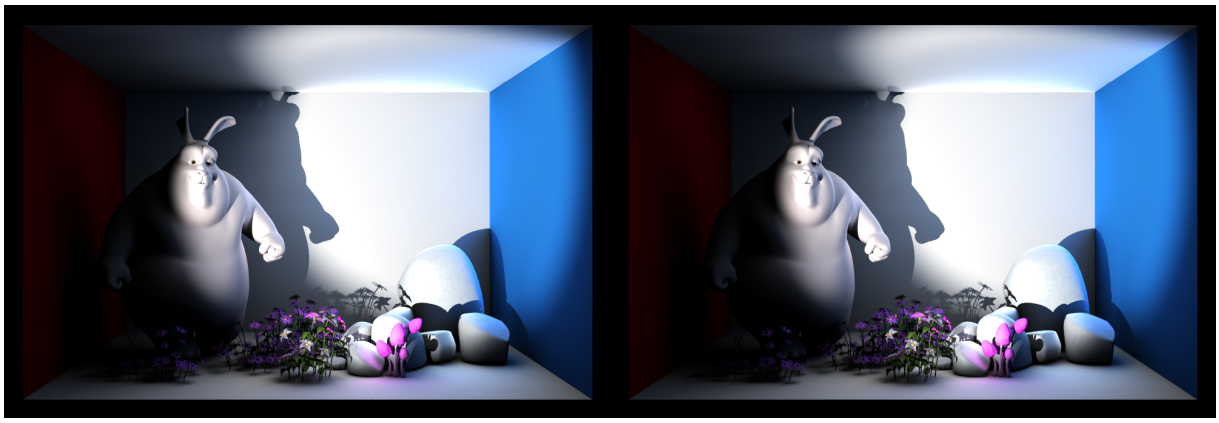
Point-based global illumination (PBGI) is a popular indirect lighting technique, intensively used in film production and recently adapted to real-time scenarios. On the contrary to unbiased, physically-based methods such as path tracing, PBGI cannot easily reproduce all indirect lighting effects but is rather used to approximate a subset of the most critical ones in a fast, noise-free fashion. This includes ambient and directional occlusion effects, as well as color bleeding, all of which stem from one-bounce diffuse light transport.

### 1.1. PBGI in a nutshell

PBGI starts by distributing a dense point set on the scene's surfaces before shading them according to the scene's primary light emitters, taking into account direct visibility only. This point sampling can be performed for example using Poisson Disk distributions or surface tessellation. Starting from this colored point set, a spatial tree is constructed bottom-up by computing at each node a spherical function capturing the diffuse directional reflection of its related subtree. Octrees and Bounding Sphere Hierarchies (BSH) have been successfully used as such PBGI structures, while Spherical Harmonics (SH) are often used as the nodes' spherical color functions. In the second part of the algorithm, a framebuffer is initialized for any receiver, which typically correspond to the unprojected 3D locations of the image pixels and may either be implemented as Lambert-warped 2D buffers or clamped cube map buffers. Finally, the shaded point set is rasterized adaptively against the receiver buffers, solving for visibility using variants of the z-buffer algorithm for each receiver independently, and the resulting filled buffers are convolved with the BRDF at receiver location to produce the final (e. g., pixel) color. Such a process can be iterated several times to simulate multiple diffuse bounces, but is usually bounded to one bounce to capture the most critical indirect lighting effects (e.g., color bleeding, directional/ambient occlusion).

### 1.2. Memory Issues

One of the main problems PBGI applications are currently facing is the large amount of data that needs to be stored in the tree nodes. Even when using spherical function approximations with a small memory footprint (e. g., SH), the sheer amount of nodes needed to correctly approximate a large scene makes it often impossible to keep all nodes in memory. For instance, it is quite usual to use SH with 3 bands only – a very coarse approximation able to capture diffuse BRDFs only. However, this already results in a coefficients vector of 27 floats per node (9 floating-point coefficients per RGB channel).



(a) (b)

**Figure 1:** ***Visual comparison***. *In the ground truth image (left), the nodes' reflectance is approximated using 3 bands of spherical harmonics resulting in 108 bytes per node. In our quantized result (right), nodes reflectance data is indexed over a LUT with 10 000 entries optimized using k-means in the parameter space, resulting in a 14 bits per node data coding scheme, giving an effective compression ratio for the reflectance data of approximately 60. The quantized result captures most of the nodes' spherical function space and only leads to slight discretization errors. The overall result is perceptually and numerically very close to the ground truth (PSNR 66dB).*

### 1.3. Overview

Studying the node values in practical scenes, we observe a significant coherence between nodes, independent of their position in space (e. g., two similar distant buildings lit by the sun). Many nodes share indeed very similar spherical functions and a large portion of the memory is wasted in replicating again and again similar data chunks. Therefore, we propose to exploit this scattered redundancy by quantizing the nodes' spherical function data over a small look-up table (LUT), optimized in a fast pre-process. We use a k-means clustering in the SH parameter space over a small subset of the scene's nodes to define the LUT entries and progressively quantize all nodes' data entries at first bounce shading time by substituting the nodes' spherical functions with a simple index over the LUT. This substitution is performed in a streaming algorithm to avoid having the full, non-quantized tree in memory at any time. As a result, our PBGI tree memory footprint is significantly smaller and allows to process larger scenes without resorting to out-of-core methods and with almost no visual differences in the final rendering. Note however that our approach can be combined with out-of-core methods to reach even larger scene sizes.

## 2. Previous Work

PBGI has been originally introduced by Christensen [Chr08] to compute ambient occlusion and color bleeding, before quickly becoming a reliable solution for fast global illumination. Its principle builds upon surfel-based ambient occlusion for real time applications [Bun05] as well as direct point-based rendering techniques [GP07], which first substituted point hierarchies to polygons in a rasterization process.

PBGI can be implemented on the GPU for final gathering [REG*09] and even reach real-time performances [HREB11] with high resolution dynamic scenes using adaptive, per-receiver level-of-detail extraction in the scene. These efficient implementations usually replace the original octree with a more flexible bounding volume hierarchy [RL00] while simplifying the internal node structure (e. g., single scalar value). Memory issues have so far been mostly tackled using out-of-core frameworks maximizing cache usage and uniform quantization to code nodes' data with half-float precision [KTO11].

The approach we propose in this paper is orthogonal to such methods. Our focus on factorizing PBGI data within a scene is inspired by recent trends in geometric modeling [PMW*08] and image representation [WWOH08], which develop object-/category-specific compression spaces, while our particular choice of a LUT-based approach relates to popular fast image and shape retrieval methods [SZ03].

The general problem of GI data compression has been extensively studied over the last decade and the popular SH basis [SKS02] is already a form of local compression, easy to combine with subspace analysis (e. g., PCA-based methods [SHHS03]). Our scene-aware quantization scheme is orthogonal to such methods and, from an implementation point of view, much simpler.

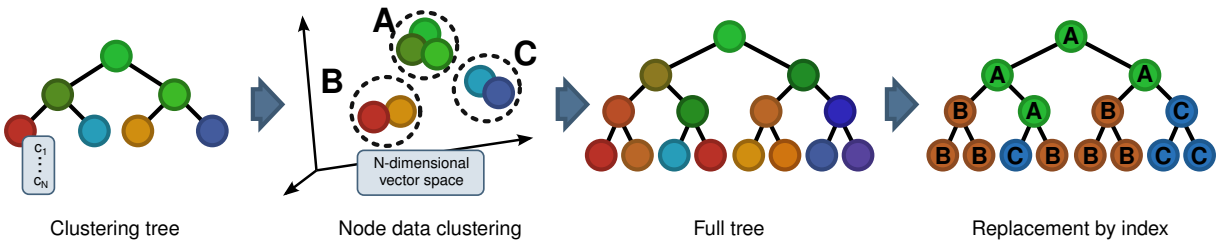



**Figure 2:** Left*: A quantizing tree is constructed from a subsampling of the scene's points. Each node in this tree contains spherical function approximations, usually in the form of coefficients vectors (NDV) which typically exhibit high redundancy. The resulting set is clustered using k-means in the NDV space (*middle-left*), leading to a NDV LUT. At rendering time, the full tree is constructed, quantizing progressively each NDV to the closest LUT entry (*right*). Note that leaf nodes are omitted as, usually, they do not contain NDVs during the rendering step.*

## 3. Tree Data Compression

### 3.1. LUT Construction

Our quantization scheme starts by defining a spherical function LUT, learned from the scene at initialization (see Fig. 2 for an overview). Our approach is generic in the sense that we can apply it on arbitrary spherical functions, as long as it takes the form of a $N$-dimensional node data vector (NDV). In practice, we use SH with 3 bands per color channel, resulting in 27-dimensional NDVs. Let $w \in R^N$ be such a NDV, with individual entries $w_{i,k}$ (e. g., SH coefficients). We express similarity between two NDVs as a real-valued distance function $d(w_i, w_j)$ In practice, we use the L2-norm of their difference:

$$d_{L2}(w_i, w_j) = \sqrt{\sum_{k=1}^{N} \left(w_{i,k} - w_{j,k}\right)^2},$$

We use RGB as the color value space of our functions, but more perceptually motivated spaces (e. g., Lab space) can also be used.

We control the quantization process by specifying the size $l$ of the LUT for which we need to determine the most representative NDV in the scene. We adopt a variational approach in the form of a k-means clustering in $R^N$. Given $S_f$, the initial point-sampling of the scene, we start by randomly selecting a small subset $S_d$, typically one to two orders of magnitude smaller than $S_f$. Then, in order to capture NDVs at various scales and properly cover the parameter space for our LUT optimization, we shade $S_d$ before constructing a temporary PBGI tree over it. We use **all** its NDVs as a N-dimensional point set over which we compute a k-means clustering. To do so, we use the Lloyd algorithm [Llo82], starting with $l$ random centers, and relocating them to minimize $d_{L2}$ in an iterative way: at each step, we move all centers to the barycenter of the samples located in their associated Voronoï cell. Ten to twenty iterations are usually enough to converge. Moreover, a dimensionality of $N = 27$ is low enough to allow the efficient use of acceleration structures like kD-trees for the nearest-neighbor (i. e., nearest cluster center) search. Finally, we extract the $l$ stabilized centers and use them as LUT entries. Later on, they are indexed using $\log_2(l)$ bits by the full PBGI tree nodes.

### 3.2. On-the-fly Quantization

As soon as the LUT is computed, the full PBGI tree can be constructed for indirect lighting evaluations. Once the basic tree structure is initialized (e. g., bounding sphere hierarchy), without NDV, the leaves are shaded (i. e., the spherical functions for the reflected light are constructed) from the scene's light sources using spherical sampling and the NDVs are propagated bottom-up to the root, averaging children NDVs at each internal node.

As our initial problem was that a whole PBGI tree may possibly not be held in memory, it is often not possible to build the whole tree first and then quantize NDVs to their respective LUT indices. Instead, we quantize the NDVs on the fly. More precisely, once the NDV of a node is computed from its children, it is classified against the LUT centers in $R^N$ and replaced by the LUT index of the closest center accord-

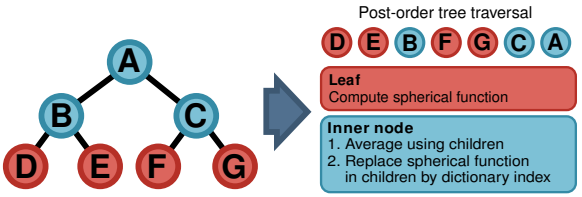



**Figure 3:** *On-the-fly quantization of the full rendering tree. To avoid having the full, unquantized tree in memory, we substitute the spherical functions in each node on the fly. This is done by traversing the full tree in a post-order fashion where the spherical functions are only temporarily computed (i. e., when they need to be averaged in the parent node) and substituted by the index as soon as they are no longer needed.*

| Clusters | Index size (in bits) | Memory usage (in MB) / **Compression rate** | | | | |
|---|---|---|---|---|---|---|
| | | SH only | Quantized SH only | Full Tree | Quantized Full Tree | |
| | | | | | Leaf method A | Leaf method B |
| 100 | 7 | 1080 | 8.75 / **123.4x** | 1460 | 388.75 / **3.76x** | 288.75 / **5.05x** |
| 1000 | 10 | 1080 | 12.50 / **86.4x** | 1460 | 392.50 / **3.72x** | 292.50 / **4.99x** |
| 10 000 | 14 | 1080 | 17.50 / **61.7x** | 1460 | 397.50 / **3.67x** | 392.50 / **4.91x** |

**Table 1:** *Compression rates for different amounts of cluster centers. This amount induces the number of bits required to encode each index. The memory usage is given for a scene with 10M surfels using a binary tree with one leaf per surfel, resulting in 10M leaves and 10M inner nodes. The compression corresponds to two different leaf formats, as shown in Table 2.*

ing to $d_{L2}$. We speed this step up by using a kD-tree for the closest-cluster search.

As summing up quantized NDVs would lead to error amplification in the bottom-up propagation process, each NDV must be computed from actual **non-quantized** children NDVs. We solve this issue by traversing the tree in post-order (see Fig.3), so that the children of a node are always traversed first and can be safely quantized by the node itself, which is traversed immediately after (see Alg. 1). This avoids maintaining more than $q+1$ non-quantized NDV in memory, with $q$ being the tree node's arity (e. g., 2 for a BSH, 8 for an octree).

**Algorithm 1** Post-order node quantization.

```
nodes ← post_order(tree)
for each n ∈ nodes do
    if n is leaf then
        n.NDV ← compute_spherical_function(n)
    else
        n.NDV ← average_children(n.children)
        for each c ∈ n.children do
            c.NDV ← LUT_Quantize(c.NDV)
        end for
    end if
end for
```

### 3.3. Compression

The number of centers dictates the compression ratio. For up to 65k different LUT entries (or cluster centers), each index can be encoded in 2 bytes. Therefore, considering our initial scenario, each 27-floats NDV (i. e., 108 bytes, assuming 4 bytes per float) can be quantized to 2 bytes, resulting in a compression factor of more than 50 for the spherical functions, the LUT size being negligible for large enough scenes. Of course, the tree structure still needs to be encoded (e. g., positions), but efficient solutions exist [RL00]. Besides our primary focus on spherical function compression, the full tree memory footprint depends on the leaf format and the arity of the tree. Usually, no spherical functions are stored in the leaves. In some cases this is beneficial though, for instance when the surfels' BRDF is not purely diffuse but slightly directional. In such cases, storing a spherical function in the leaves improves the results and they benefit from our quantization scheme.

In the case of a binary tree with one surfel per leaf and no spherical functions in the leaves, we achieve compression rates of 61x (10k LUT entries) to 123x (100 LUT entries) for the nodes' spherical functions and 3x to 5x for the whole tree (see Tab. 1), mostly depending on the way the data is stored in the leaves (see Tab. 2). In the case of an octree, the full tree compression is lower due to the higher leaves-to-inner-nodes ratio.

## 4. Results

We implemented our quantization technique in the Yafaray raytracing engine using Poisson Disk sampling to generate the initial point set. In all quality comparisons, the ground truth stands for the original (unquantized) PBGI algorithm [Chr08].

In Fig. 5a we compare the k-means clustering against a scene-oblivious random sampling of cluster centers. We can observe that the k-means clustering provides a significantly smaller error. This is numerically reported in Fig 4. The cluster error is a measure of the difference of each node's quantized spherical function from its original, non-quantized value and is given as an absolute value. The overall error is computed as the mean over the absolute errors in each node. Using k-means clustering decreases the probability of missing frequently repeating spherical functions, leading to a smaller mean error.

| | | Position | Color | Normal | Area |
|---|---|---|---|---|---|
| Memory (in bytes) | A | 12 | 6 | 6 | 2 |
| | B | 2 | 6 | 6 | 2 |

**Table 2:** ***Leaf formats***. *"A" refers to storing the data using half-floats [KTO11] except for the position which keeps full precision. "B" refers to storing the position in a parent-relative fashion as done in [RL00].*

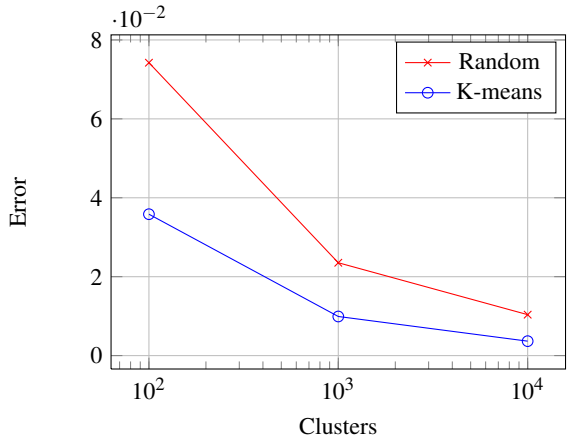



**Figure 4:** *Approximation quality for random and k-means cluster LUT in the "Big Buck Bunny" scene. The* error *is the average of the absolute mean error between each node's actual data and its approximation.*

The influence of the LUT size can be observed in Fig. 5b: using 100 clusters only, the red curtains color bleeding is not captured correctly. This introduces a significant error when compared to the ground truth, which is fixed by increasing the number of cluster/LUT entries. Similarly, a too sparse subset for the initial LUT construction can produce artifacts. In particular, when some very small surfaces (i.e., undersampled by the point generation process) have unique or rare spherical functions that are not otherwise sampled in the scene, the closest cluster can be far away in NDV space and the node cluster error is high. In the case where such a surface is strongly lit and other surfaces are close-by, the expected light bounce may not occur with the correct color or intensity (see Fig. 6).

The different perceptual error measures [Yee04] presented in this paper show that, overall, a negligible quantization error is introduced, even in complex scenes with highly varying color distributions and under strong quantization rate.

About temporal coherence, slight flickering artifacts occured in our experiments when using too few LUT entries (e.g., 100 to 1000 depending on the scene), which were all fixed by increasing this number.

Finally, we report the LUT and full tree construction time in Tab. 3 (Intel Core2Quad, 2.83MHz, 8GB), using $32 \times 32 \times 6$ precomputed SH coefficients for uniformly distributed normal directions. Note, however, that speed was not the prime focus of this study.

| | No LUT | Number of clusters | | |
|---|---|---|---|---|
| | | 100 | 1000 | 10 000 |
| LUT constr. | — | 12.25 | 17.03 | 22.60 |
| Tree constr. | 49.72 | 101.89 | 145.37 | 193.89 |

**Table 3:** *Clustering and tree construction timings (in sec) for the "Bunny and Bird" scene using 5M surfels (resulting in 5M inner nodes) and a representative node fraction of 10% on a single CPU thread. A kD-tree is used to speed up the closest-cluster search in the LUT and tree construction, resulting in sublinear computation time growth.*

## 5. Conclusion

We have introduced a new scene-aware quantization scheme for PBGI data which exploits its redundancy. By learning a small set of representative spherical functions in the parameter space, we are able to substitute full node data with accurate quantized values in a memory-efficient streaming process, leading to significant compression ratios. Our approach is simple, easy to implement in any existing PBGI system and intuitive to control. We experimented with various scenes, showing that it introduces almost no visual difference. So far, our work is mostly focused on one-bounce indirect diffuse lighting and the low frequency nature of band-limited SH certainly helps the quantization process. Generalization to glossy and specular PBGI remains an open question. Other research directions include factorized PBGI LUT among scenes, sampling strategies accounting for surface proximity, combination with out-of-core schemes, symmetry analysis, and application to real time PBGI systems. Last, our quantization scheme may be helpful in other SH applications.

**Acknowledgement.** This work has been partially funded by MEDIAGPU and iSpace&Time ANR Projects, the E.U. 3DLife N.o.E. and the E.U. REVERIE project.

## References

[Bun05] BUNNELL M.: *GPU Gems 2*. 2005, ch. Dynamic ambient occlusion and indirect lighting, pp. 223–233. 2

[Chr08] CHRISTENSEN P.: *Point-based approximate color bleeding*. Tech. Rep. 08-01, Pixar, 2008. 2, 4

[GP07] GROSS M., PFISTER H. (Eds.): *Point-Based Graphics*. Morgan Kaufmann Series on Computer Graphics, 2007. 2

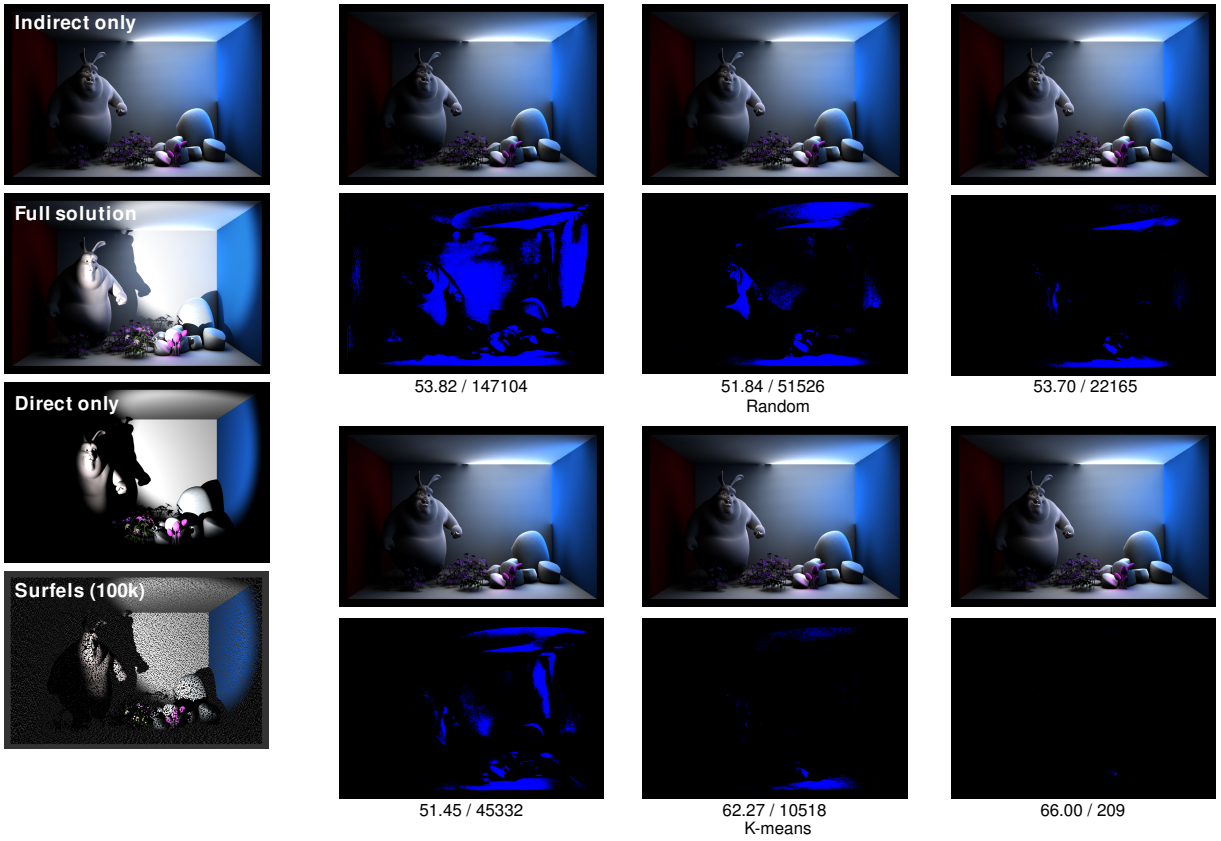
[HREB11] HOLLÄNDER M., RITSCHEL T., EISEMANN E., BOUBEKEUR T.: Manylods: Parallel many-view level-of-detail selection for real-time global illumination. *Comp. Graph. Forum (Proc. EGSR) 30*, 4 (2011), 1233–1240. 2

[KTO11] KONTKANEN J., TABELLION E., OVERBECK R. S.: Coherent out-of-core point-based global illumination. *Comp. Graph. Forum (Proc. EGSR) 30*, 4 (2011), 1353–1360. 2, 4

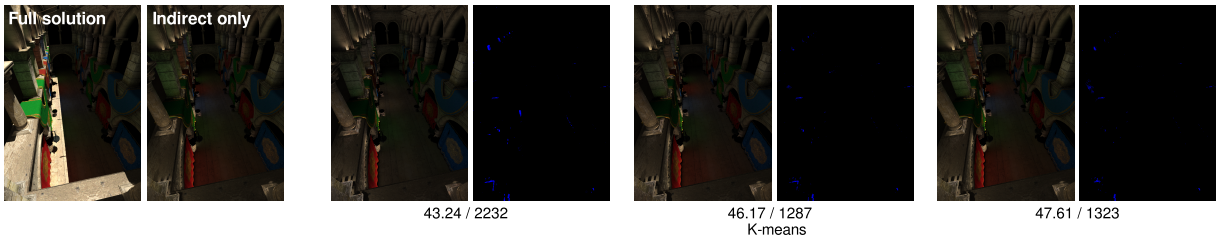
[Llo82] LLOYD S. P.: Least squares quantization in pcm. *IEEE Transactions on Information Theory 28*, 2 (1982), 129–137. 3

[PMW*08] PAULY M., MITRA N. J., WALLNER J., POTTMANN H., GUIBAS L.: Discovering structural regularity in 3D geometry. *ACM Trans. Graph. 27*, 3 (2008), 43:1–43:11. 2

[REG*09] RITSCHEL T., ENGELHARDT T., GROSCH T., SEIDEL H., KAUTZ J., DACHSBACHER C.: Micro-rendering for scalable, parallel final gathering. In *ACM Trans. Graph.* (2009), vol. 28, ACM, pp. 132:1,132:8. 2




**(a)** *"Big Buck Bunny" scene, random and k-means optimized LUT*




**(b)** *"Sponza Yard"*

**Figure 5:** *Visual comparison (indirect contribution only) between ground truth (top left) and our quantization, using 100 (top middle left), 1000 (top middle right) and 10 000 clusters (top right). The error images (bottom) are the perceptual differences in the Lab color space [Yee04], depicted in black (no visible difference) and blue (visible difference). For numerical comparison, we give the PSNR for RGB images and the number of Perceptually Different Pixels [Yee04] (PDP) below each error image (<PSNR>/<PDP>). The full rendering is displayed on the bottom left. Figure continued on the next page.*

[RL00] RUSINKIEWICZ S., LEVOY M.: Qsplat: A multiresolution point rendering system for large meshes. In *Proc. SIGGRAPH* (2000), pp. 343–352. 2, 4

[SHHS03] SLOAN P.-P., HALL J., HART J., SNYDER J.: Clustered principal components for precomputed radiance transfer. *ACM Trans. Graph. 22*, 3 (2003), 382–391. 2

[SKS02] SLOAN P.-P., KAUTZ J., SNYDER J.: Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *Trans. Graph. 21*, 3 (2002), 527–536. 2

[SZ03] SIVIC J., ZISSERMAN A.: Video Google: a text retrieval approach to object matching in videos. In *Proc. Int'l Conf. Computer Vision* (2003), pp. 1470–1477. 2

[WWOH08] WANG H., WEXLER Y., OFEK E., HOPPE H.: Factoring repeated content within and among images. *ACM Trans. Graph. 27*, 3 (August 2008), 14:1–14:10. 2

[Yee04] YEE H.: A perceptual metric for production testing. *Journal of Graphics Tools 9*, 4 (2004), 33–40. 5, 6

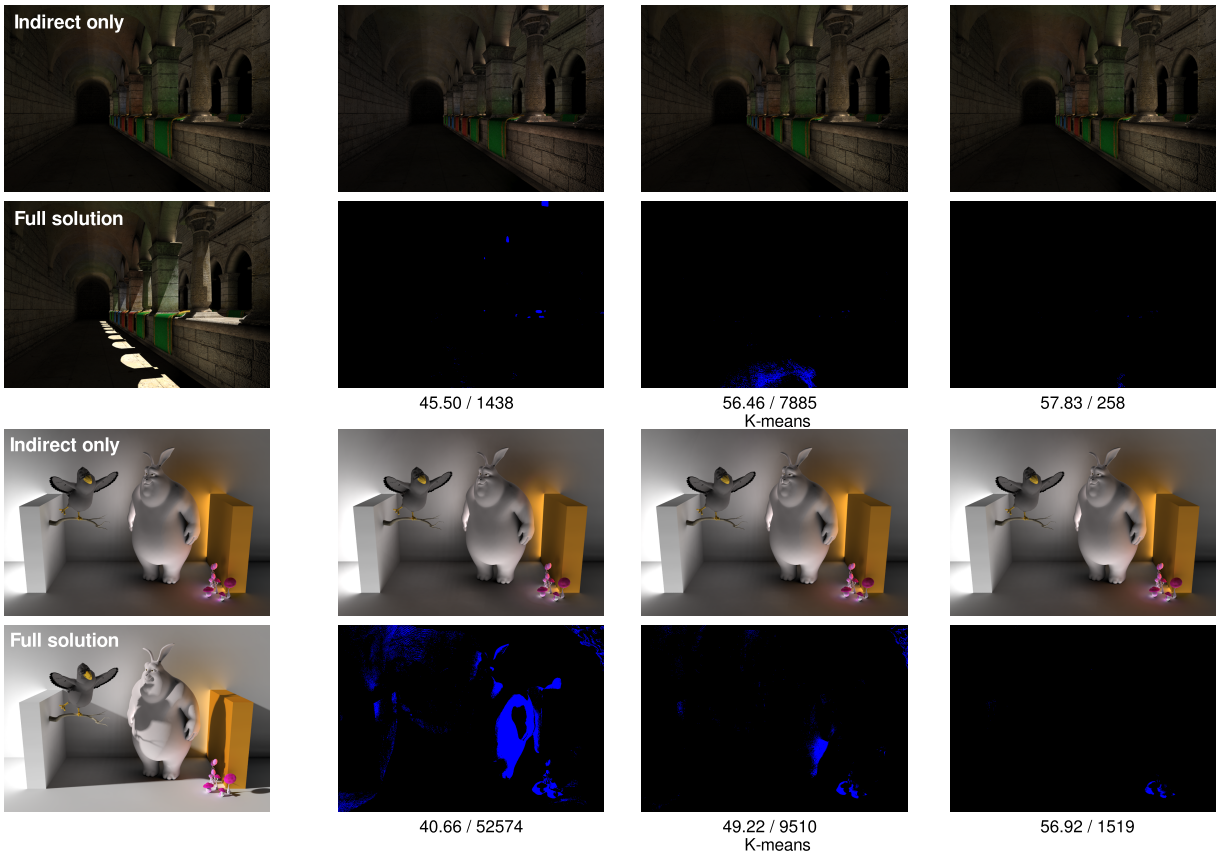



**Figure 5:** *Figure continued from previous page: "Sponza" and "Bunny and Bird" scenes.*

**Figure 6:** *We vary the number of initial nodes for the LUT construction between 0.1% (left), 1% and 10% (right) of the final node count (5M here). While only a little visual difference appears when dropping from 10% to 1%, using 0.1% starts to introduce visible artifacts (close-up), missing the pink spherical functions of the mushroom tops and damaging color bleed on the ground.*

**(a)** *Direct illumination only* **(b)** *With PBGI* **(c)** *With Quantized PBGI* **(d)** *Perceptual Error*

**Figure 7:** *A more complex example (LUT with 10 000 clusters).*