Author's Draft. The final version will be presented at the Eurographics Symposium on Rendering and published a spcial issue of the Computer Graphcis Forum journal. 2013

Volume 32 (2013), Number 4

Factorized Point Based Global Illumination

Beibei Wang[+*] Jing Huang[*] Bert Buchholz[*#] Xiangxu Meng[+] Tamy Boubekeur[*]

[*]Institut Mines-Telecom; Telecom ParisTech; CNRS LTCI [+]Shandong University [#]NYU Polytechnic Institute

Abstract

The Point-Based Global Illumination (PBGI) algorithm is composed of two major steps: a caching step and a multiview rasterization step. At caching time, a dense point-sampling of the scene is shaded and organized in a spatial hierarchy, with internal nodes approximating the radiance of their subtrees using spherical harmonics. At rasterization time, a microbuffer is instantiated at the unprojected position of each image pixel (receiver). Then, a view-adaptive level-of-detail of the scene is extracted in the form of a tree cut and rasterized in the receiver's microbuffer, solving for visibility using a local variant of the z-buffer. Finally, the pixel color is computed by convolving its filled microbuffer with the surface BRDF. This noise-free indirect lighting method is widely used in the industry and captures several critical lighting effects, including ambient occlusion, color bleeding, (indirect) soft-shadows and environment lighting. However, we observe a large redundancy in this algorithm, both in cuts and receivers'microbuffers, which stems from their relatively low resolution. In this paper, we propose an evolution of PBGI which exploits spatial coherence to reduce these redundant computations. Starting from a similarity-based variational clustering of the receivers, we compute a single tree cut and rasterize a single microbuffer for each cluster. This per-cluster microbuffer provides a faithful approximation of the incident radiance for distant nodes and is composited over a receiver-specific microbuffer rasterizing only the closest nodes of the cluster's cut. This factorized approach is easy to integrate in any existing PBGI implementation and offers a significant rendering speed-up for a negligible and controllable approximation error.

1. Introduction

The visual impact of global illumination (GI) in a synthesized picture is the sum of a number of lighting effects stemming from indirect light bounces. Among them, one-bounce diffuse effects, such as ambient occlusion, directional occlusion, color bleeding and indirect soft shadows, carry a large portion of the visual realism that typical GI solutions bring. Point-based global illumination (PBGI) is a popular rendering technique which captures such a subset of GI effects for a moderate amount of time and is intensively used in special effects and computer animation productions. This GI approximation model can be seen as a generalized forward rendering method which combines a fast adaptive approximation of the scene with a multiview rasterization. The resulting algorithm is noise-free, amenable to a parallel implementation and can even be extended to other GI effects (e.g., multiple bounces), although still away from a full GI solution, in particular when it comes to specular indirect phenomena (i.e., caustics).

1.1. Basic Algorithm

PBGI [Chr08] runs in a two-step process: a caching step and a multiview rasterization step. At caching time, the scene is densely point-sampled (e.g., using Poisson disks), the points are shaded from the light sources – accounting for direct shadows only – and structured in a hierarchical data structure (e.g., octree, BSH). This tree is constructed bottom-up from the shaded points, with internal nodes carrying approximations of their related sub-trees (e.g., bounding sphere, normal cone, low-degrees spherical harmonics modeling the outgoing diffuse radiance).

At rasterization time, each pixel of the final picture is shaded using a so-called *microbuffer*, which is a small hemispherical RGBZ image instantiated at the unprojected position of the pixel (or *receiver*) in the scene. For each microbuffer, a specific level-of-detail (LoD) of the scene is extracted in the form of an adaptive cut in the PBGI tree. The resulting nodes are rasterized in the microbuffer using a local variant of the z-buffer algorithm to solve for visibility. The filled microbuffer is finally convolved with the point's BRDF to shade the pixel.

The two key ideas of this algorithm are (i) the point's hierarchy, which acts as an economic substitute to the actual scene when it comes to the many adaptive LoDs which have to be extracted; (ii) the microbuffers, which extend the concept of rasterization to a per-pixel/receiver level.

1.2. Redundancy Issue

Looking back at the rasterization step, we observe that a specific cut is computed from the entire scene for each single receiver. However, the resolution of their microbuffers is typically low (from 4x4 to 64x64 in practice) which immediately translates into tree cuts having a large number of coarse nodes, therefore being highly similar for nearby receivers. As we will show later, this abundant redundancy has a significant impact on the overall rendering time.

1.3. Overview

We tackle this problem by exploiting the microbuffers'spatial coherence to factorize both cut computations and rasterizations. Our factorized PBGI technique (or FPBGI) works in three steps (see Fig. 1):

1. we cluster the receivers based on their similarity and select a per-cluster *active* receiver,
2. for each cluster, we compute a single (coarser) cut from the active receiver and rasterize it in a microbuffer shared by all receivers of the cluster
3. for any receiver, we start the tree traversal from its cluster cut and rasterize only the newly added (i.e., closer) nodes in a receiver-specific microbuffer, which is composited with the cluster one before final BRDF convolution.

As a result, a large part of tree traversals and cut rasterizations are factorized among nearby receivers, which leads to an overall rendering speed-up ranging from 2x to 4x on the typical scenes illustrating this paper.

2. Previous Work

PBGI. PBGI was first introduced by Christensen [Chr08], who proposed the idea of microbuffers and exploited the notion of point-based substitutes introduced by Bunnell [Bun05] for real time ambient occlusion and indirect illumination. Ritschel et al. [REG*09] then replaced cube microbuffers with 2D Lambert-warped ones, introducing importance sampling to PBGI together with an efficient GPU implementation. Holländer et al. [HREB11] later improved on the fine-grained parallelism of the adaptive cut computation by pairing nodes and receivers in a low-scale GPU data amplification mechanism. The cut definition itself has been addressed by Maletz and Wang [MW11] who used an importance-driven point projection based on an initial clustering, by Wang et al. [WMXS11] who grouped together close points with similar normals and computed average cuts for a subset of the receivers, and by Tabellion [Tab12] who recently exposed a set of cut picking algorithms suitable for HDR imaging. The PBGI accuracy entirely depends on the density of the initial sampling and the related memory issue has been tackled by Kontkanen et al. [KTO11], who proposed an out-of-core framework for PBGI with cache-coherent tree construction and traversal. Buchholz and Boubekeur [BB12] proposed an in-core solution to this problem, learning a reduced set of node data vectors in high dimension and quantizing all tree nodes against the resulting look-up table.

Coherence in rendering. Coherence through some form of "reuse" mechanism has been widely studied in GI research. Such techniques try to avoid redundancy at different levels of the GI solution computation, including irradiance, radiance, shading and even tree-cuts in a closer context to ours. Ward et al. [WRC88] reused illumination computation by computing scalar (diffuse) irradiance on a subset of pixels and interpolating for the others, eventually using gradients [WH92] for smoother results. Wang et al. [WWZ*09] used k-means to subsample receiving points and interpolate irradiance, reaching interactive framerates but missing small geometric details. *Radiance Caching* [KGPB05] overcomes the limitation to diffuse reflectance by storing incoming radiance as a directional function, interpolating it between pixels and convolving with the BRDF for every pixel. Closer to PBGI methods, Holländer et al. [HREB11] proposed a **time**-coherent cut update, together with a lazy scheme bounding the amount of time dedicated to this update. Our approach is inspired by this method, but acts in the spatial domain.

Near-far decomposition. The idea of near-far irradiance decomposition has been previously studied in the context of hardware ambient occlusion [SA07] and final gathering [AFO05]. Acting in a PBGI context, our approach differs in the sense that the near-far split is entirely formulated through the cluster/receiver cut, the far component being shared by numerous receivers.

3. Factorized Point Based Global Illumination

3.1. Variational Receiver Clustering

Our basic assumption is that receivers with similar positions and normals have similar cuts: we propose to model this position/normal similarity by computing a variational clustering of the receivers based on a specific metric $\mathcal{D}$. To ease parallel computation, we start by regularly tiling the image space and work independently on each tile. Within a tile, we group spatially coherent receivers in k clusters using a variant of the *k-means* algorithm:

1. we initialize k centers from randomly selected receivers in the tile,
2. we cluster the tile's receivers by associating each of them to its *closest* center w.r.t. $\mathcal{D}$
3. we update clusters' centers and restart in (2).

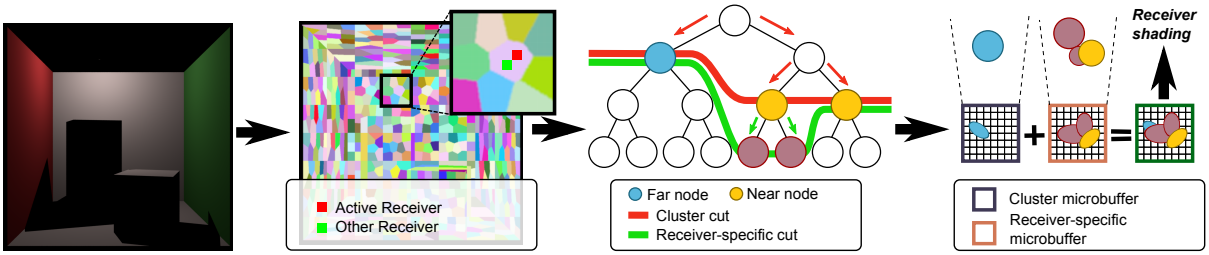

Figure 1: ***Principle***. *Starting from a tiled set of image pixels/receivers (left), we perform a variational clustering based on positional and normal similarity (middle left). For a given cluster, we compute a shared cut (middle right, red) later reused by each individual receiver to further refine their own cuts (middle right, green). The far nodes of the cluster are rasterized into a shared cluster microbuffer (right, purple) and refined nodes (added on a per-receiver basis) are rasterized in a receiver-specific microbuffer (right, orange), which is composited into one cluster for final BRDF convolution (pixel indirect shading).*

We perform this procedure for a prescribed number of iterations and search, for each cluster, the closest receiver to the resulting center: in the following, we call it the *active* receiver of a cluster.

Following Cohen-Steiner et al. [CSAD04], we define our position/normal metric $\mathcal{D}$ as a Sobolev summed metric:

$$\mathcal{D}(\mathbf{x},\mathbf{c}) = ||\mathbf{p_c} - \mathbf{p_x}||^2 + \alpha||\mathbf{n_c} - \mathbf{n_x}||^2$$

with $\mathbf{x}$ being a receiver, $\mathbf{c}$ a cluster center, $\mathbf{p}$ (resp. $\mathbf{n}$) their position (resp. normal) in $\mathbb{R}^3$. The weight α trades cluster flatness for spatial extent. We typically set it to the length of the tile's receivers' bounding box diagonal. Last, at each iteration, the center position and normal of a cluster $\mathcal{C}$ are updated as follows:

$$\mathbf{p_c} = \frac{\sum_{\mathbf{x}\in\mathcal{C}} \mathbf{p}_x}{card(\mathcal{C})} \quad \mathbf{n_c} = \frac{\sum_{\mathbf{x}\in\mathcal{C}} \mathbf{n_x}}{||\sum_{\mathbf{x}\in\mathcal{C}} \mathbf{n_x}||}$$

3.2. Cluster Cut and Microbuffer

Within a cluster $\mathcal{C}$, the factorized workload among receivers takes the form of a single shared cut and a single microbuffer which are computed w.r.t. the *active* receiver $\mathbf{x}_\mathcal{C}$.

In the next step of the rasterization phase, we start by traversing the PBGI tree from the root for $\mathbf{x}_\mathcal{C}$ but stop early to produce a cut which is coarser than required in the vicinity of $\mathbf{x}_\mathcal{C}$. Indeed, we assume that the significant difference between two nearby microbuffers only appears at fine scale (i.e., closer nodes) and deal with it later.

During the top-down PBGI tree traversal, we use a *far/near* classification of the tree's nodes based on a measure γ for each node/receiver pair: far nodes ($\gamma > \varepsilon$) are traversed as usual, while near nodes ($\gamma \leq \varepsilon$) stop the traversal immediatly. The node/receiver measure is defined as $\gamma = \frac{r}{d}$ with r being the cluster's radius r and d the distance between the node and $\mathbf{x}_\mathcal{C}$. The resulting cluster cut contains two types of nodes: *far* nodes, which are rasterized in the shared cluster microbuffer, and *near* nodes, which will be concurrently refined for each individual receiver in the next step. At this stage, the cluster microbuffer already carries the distant irradiance shared by all cluster receivers.

3.3. Receiver Cut, Microbuffer and Shading

In the last part of our algorithm, we process each individual receiver in parallel. For a given receiver, we compute its specific cut starting from the cluster cut (instead of the tree's root) and traversing the hierarchy down to the classical microbuffer-dependent solid angle threshold. Only the newly added nodes to the cut are marked as *refined*. Once the cut is completed, we rasterize its *refined* and *near* nodes into a receiver-specific microbuffer. Basically, only the closer nodes are rasterized and we obtain a sparse microbuffer.

Last, we composite this receiver microbuffer with the corresponding cluster one, using the depth component of both microbuffers to properly cull the microbuffer pixels which are hidden by this combination. The resulting composited microbuffer is finally convolved with the receiver's BRDF to shade the receiver/pixel.

4. Results

We implemented our technique in the Mitsuba Renderer [Jak10], with the initial point set being generated using Poisson Disk sampling. Comparisons are performed against the original PBGI algorithm [Chr08] and performances are measured on a 2.67GHz Intel i7 (8 cores) with 9GB of main memory. Images are rendered with one-bounce indirect illumination at a 1280×1000 pixels resolution (except for the Cornell Box, at 1024×1024) with 32×32 tiles.

In all comparisons, we measure numerical differences with the Mean Squared Error (MSE) and visual differences by counting the number of Perceptually Different Pixels (PDP), as proposed by Yee [Yee04]. This perceptual error metric acts in the Lab space and is plotted in black and blue.

Scene	Points	Individual Timings					Total Time		Error	
		Clustering	Cut Computation		Micro-Rasterization		Full Rendering			
		Time(s)	*PBGI*	*FPBGI*	*PBGI*	*FPBGI*	*PBGI*	*FPBGI*	*PDP*	*MSE*
CornellBox	88.88K	0.68s	2.46m	0.65m	1.73m	1.19m	5.72m	2.60m	103	8.79e-6
Bunny	1.00M	0.87s	2.38m	0.56m	1.55m	1.08m	4.49m	2.03m	61	1.31e-4
ItalianCity	8.38M	0.87s	3.65m	1.08m	1.23m	0.64m	5.52m	2.34m	235	8.15e-5
Sponza	16.19M	0.87s	19.71m	3.77m	5.40m	1.68m	26.72m	7.04m	15	2.51e-5

Table 1: *Performance measures.*

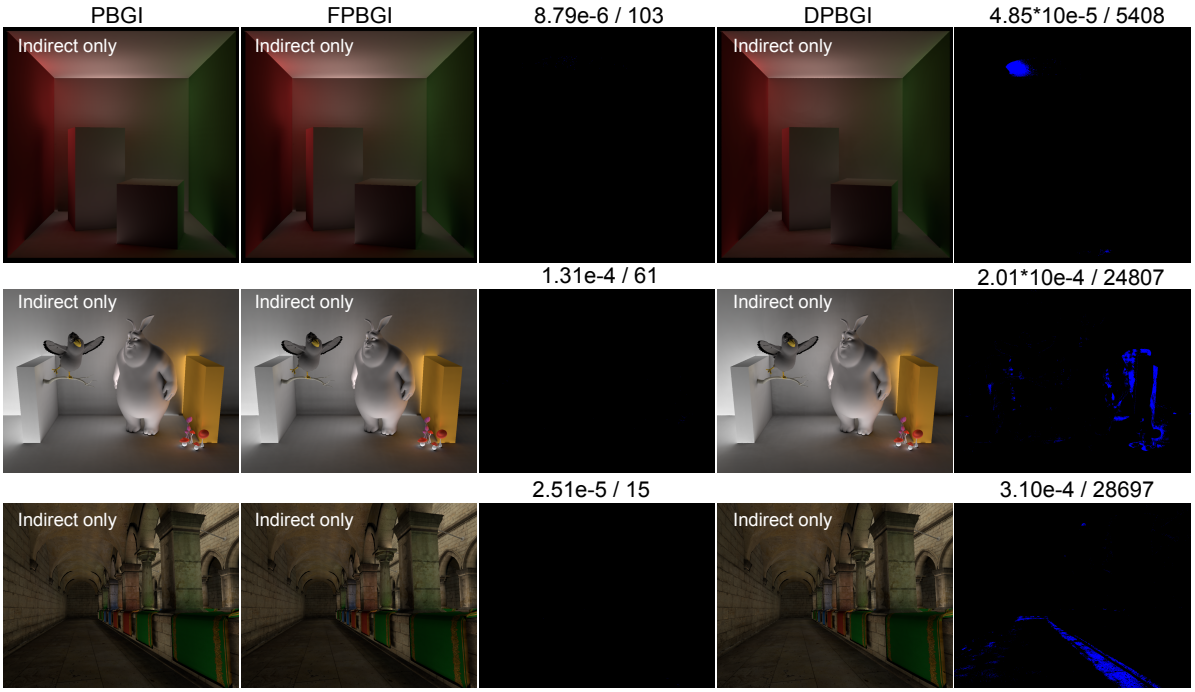

Figure 2: *Error analysis on the indirect lighting contribution for FPBGI and DPBGI against PBGI. Perceptual differences [Yee04] are plotted in black (no visible difference) and blue (visible difference). The MSE between RGB images and the number of Perceptually Different Pixels [Yee04] (PDP) are indicated in the format <MSE>/<PDP> on top of difference images.*

In Fig. 2, we compare FPBGI with the original PBGI algorithm on three different scenes. Overall, we observe a negligible error, both from a perceptual and numerical point of view. The original PBGI algorithm can indeed be trivially sped-up by reducing the resolution of the microbuffers (i.e., higher solid angle threshold in the tree traversal), which immediately translates into coarser cuts for each receiver and reduced rasterization time. Therefore, we also compare to such a *degraded* PBGI setting (DPBGI), with microbuffer resolution decreased so that the total rendering time is as close as possible to our FPBGI. In this case, DPBGI produces significantly stronger errors, with noticeable aliasing appearing.

In Table 1, we report timings and errors for the four different examples shown in Fig. 2 and Fig. 3. Here, we can assess the benefit of our factorized approach, with a speed-up ratio for the total rendering time (including BRDF evaluation and initial set up) ranging from 2.2 to 3.8 compared to the original PBGI algorithm. Looking at the specific portion of the algorithm that we target (rasterization), the speed-up ratio ranges from 3.4 to 5.2 for the cut computation and from 1.4 to 3.2 for the micro-rasterization. In all cases the receiver clustering time is negligible.

In Fig. 3, we provide a visual comparison of the final rendering (direct+indirect illumination) between a fully path-traced solution (PTS), PBGI and FPBGI. We can observe that PBGI and FPBGI provide similarly good approxima-

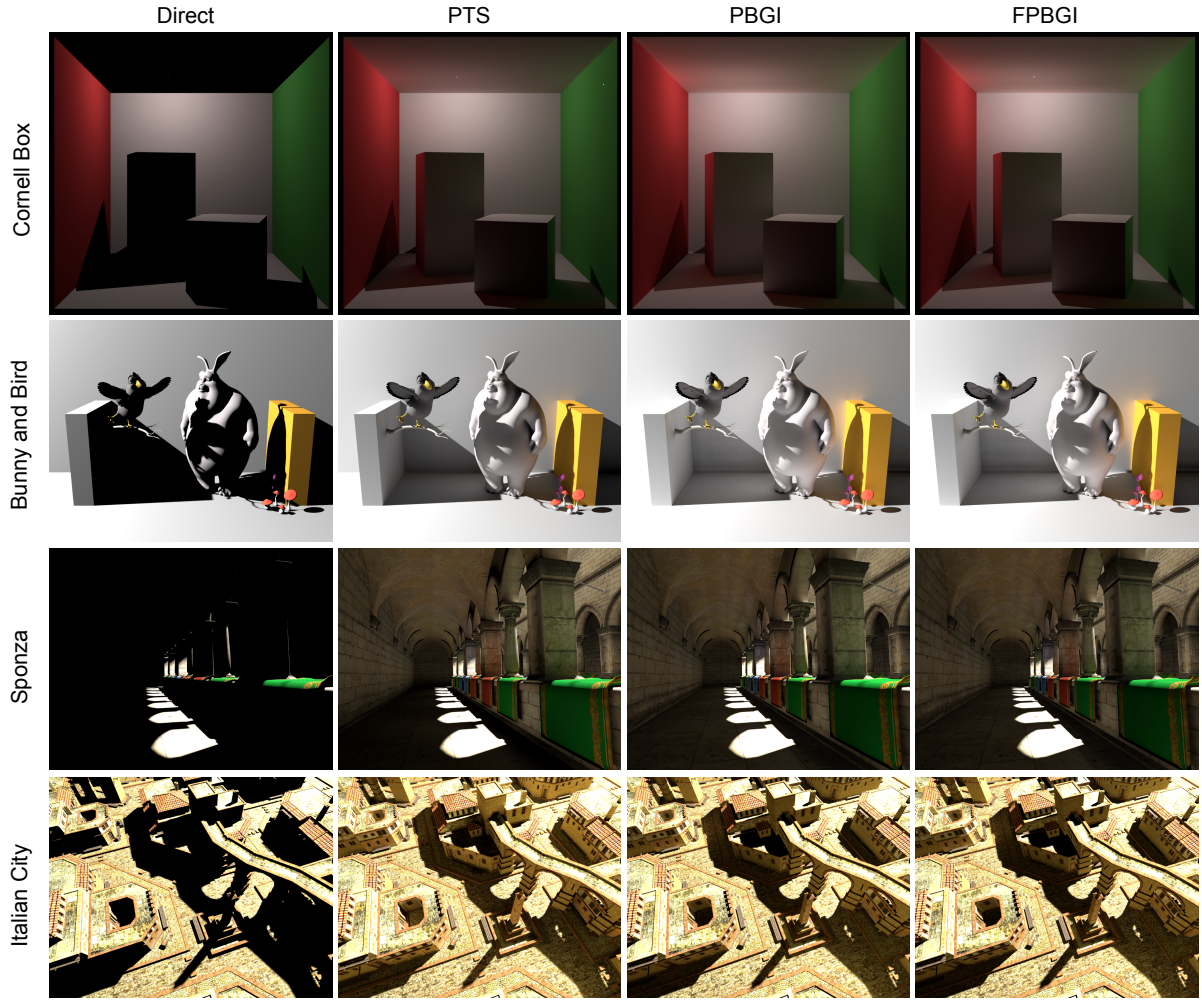

Figure 3: *Visual comparison of final renderings.*

tions of the PTS, which is typically an order of magnitude slower than FPBGI.

We also analyze the influence of the two main parameters of FPBGI: the number of clusters per-tile k and the far-near threshold ε. In Fig. 4, we illustrate their influence on the Sponza scene. We observe that the influence of k clearly dominates on the approximation quality, as measured by numerical and perceptual errors. However, looking closely at the result, we can see that, under a very small k value, large values of ε cause large visible artifacts. In Fig. 5, we plot the speed-up evolution under variations of these two parameters. We empirically determine $k = 100$ and $\varepsilon = 10e^{-2}$ as good default values for all the scenes we experimented with. Last, with visually invisible differences, FPBGI inherits the temporal coherence of PBGI: we illustrate this behavior in an accompanying video with three sequences showing animated lighting, camera and models.

Discussion. Alternatively to our approximation technique, recent approaches [REG*09,HREB11] propose to maximize the fine-grained parallelism of the PBGI algorithm in order to map it efficiently on GPU architectures. Clearly, our approach is orthogonal to such methods, but preserves the natural parallelism of PBGI. However, compared to such methods, an additional specific preliminary pass would be required to gather the shared microbuffers. As future work, at least two alternative solutions may be further developed to combine our factorization with an efficient GPU implementation: first, the typical number of clusters is large enough to load the numerous GPU computing units with cluster cuts computations using a naive implementation (i.e., one thread per-cluster first, then one thread per-receiver); sec-

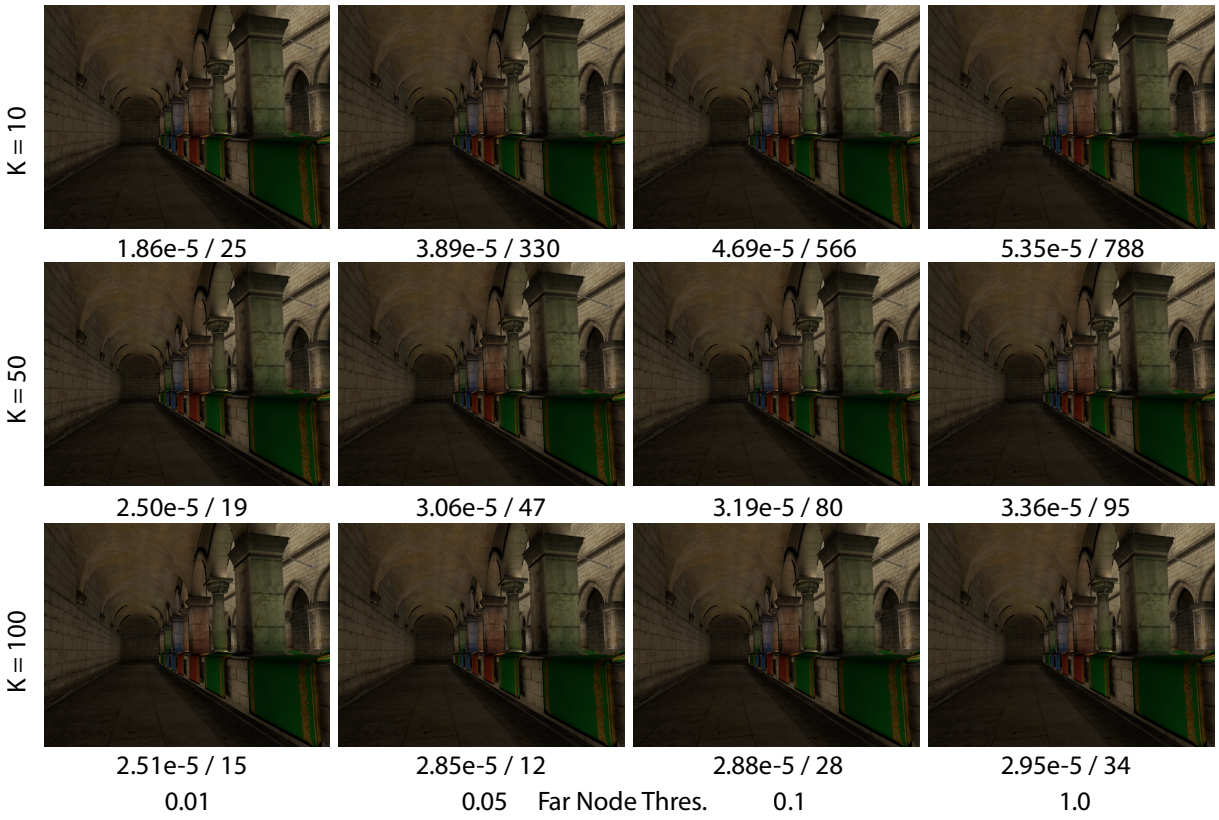

Figure 4: *Parameter influence with <MSE>/<PDP> to the PBGI solution for each pair.*

ond, a more evolved solution could use the two-layer GPU computing model (blocks and threads) to make threads belonging to the same block define concurrently the cluster cut and microbuffers in shared memory before synchronizing them and letting them processing their receiver-specific components, using the ManyLoDs algorithm [HREB11] at both stages.

Interestingly, Holländer et al. [HREB11] proposed an acceleration exploiting *temporal* coherence only, the *lazy* scheme which reuses cuts over consecutive frames, while our factorized approach exploits *spatial* coherence. Combining both approaches could help exploiting *spatio-temporal* coherence to its full extent.

Our approach has two main limitations. First, the cluster cut may be over-conservative and the resulting per-receiver cut can be too refined. Although this does not influence the rendering quality, this remains sub-optimal. A solution could be to allow receivers to "walk-up" the tree while refining their cut. Second, the two main parameters of the algorithm have fixed values. These values could be optimized dynamically and vary spatially by using the PBGI tree to perform a quick scene analysis. Last, our approach can be seen as a simplified *hierarchy* of receivers. It would then be interesting to determine how to reformulate the PBGI algorithm to rasterize, adaptively, the PBGI tree against the receiver/pixel one to reach a fully adaptive solution.

5. Conclusion

We have proposed a factorized evolution of the PBGI algorithm which exploits spatial coherence to significantly speed up the computation of indirect diffuse illumination effects. By combining an initial variational clustering with per-cluster cuts and microbuffers, we showed that the individual receiver workload boils down to a local geometry rasterization followed by a microbuffer compositing. As a result, we obtain a speed-up ranging from 2x to 4x, without any visible image degradation. Our approach is easy to implement in any PBGI framework and has a reduced set of intuitive control parameters.

Acknowledgements. This work has been partially supported by the China Scholarship Council and 863 Program of China under Grant No. 2012AA01A306, the European Commission under contract FP7-287723 REVERIE and the ANR iSpace&Time project.

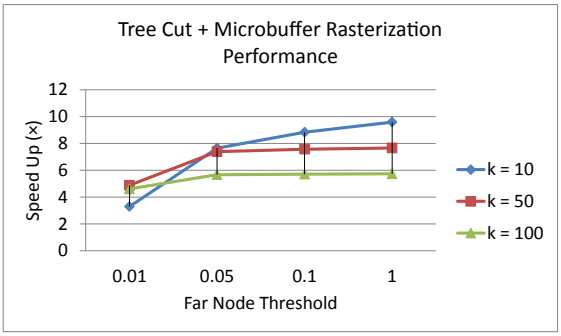

Figure 5: *Parameters influence on the speed-up.*

References

[AFO05] ARIKAN O., FORSYTH D. A., O'BRIEN J. F.: Fast and detailed approximate global illumination by irradiance decomposition. In *ACM SIGGRAPH 2005* (2005), pp. 1108–1114. 2

[BB12] BUCHHOLZ B., BOUBEKEUR T.: Quantized point-based global illumination. *Comp. Graph. Forum (Proc. EGSR 2012) 31*, 4 (2012), 1399–1405. 2

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

[Chr08] CHRISTENSEN P.: *Point-based approximate color bleeding*. Tech. Rep. 08-01, Pixar Technical Notes, 2008. 1, 2, 3

[CSAD04] COHEN-STEINER D., ALLIEZ P., DESBRUN M.: Variational shape approximation. *ACM Trans. Graph. 23*, 3 (2004), 905–914. 3

[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 2011) 30*, 4 (2011), 1233–1240. 2, 5, 6

[Jak10] JAKOB W.: Mitsuba renderer. http://www.mitsuba-renderer.org/, 2010. 3

[KGPB05] KRIVANEK J., GAUTRON P., PATTANAIK S., BOUATOUCH K.: Radiance caching for efficient global illumination computation. *IEEE TVCG 11*, 5 (2005), 550–561. 2

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

[MW11] MALETZ D., WANG R.: Importance point projection for GPU-based final gathering. *Comp. Graph. Forum 30*, 4 (2011), 1327–1336. 2

[REG*09] RITSCHEL T., ENGELHARDT T., GROSCH T., SEIDEL H.-P., KAUTZ J., DACHSBACHER C.: Micro-rendering for scalable, parallel final gathering. *ACM Trans. Graph. (Proc. SIGGRAPH Asia 2009) 28*, 5 (2009). 2, 5

[SA07] SHANMUGAM P., ARIKAN O.: Hardware accelerated ambient occlusion techniques on GPUs. In *ACM I3D* (2007), pp. 73–80. 2

[Tab12] TABELLION E.: Point-based global illumination directional importance mapping. In *ACM SIGGRAPH Talk* (2012). 2

[WH92] WARD G. J., HECKBERT P.: Irradiance gradients. In *Eurographics Workshop on Rendering* (1992). 2

[WMXS11] WANG B., MENG X., XU Y., SONG X.: Fast point based global illumination. In *Computer-Aided Design and Computer Graphics* (2011), pp. 93–98. 2

[WRC88] WARD G. J., RUBINSTEIN F. M., CLEAR R. D.: A ray tracing solution for diffuse interreflection. In *ACM SIGGRAPH Computer Graphics* (1988), vol. 22, pp. 85–92. 2

[WWZ*09] WANG R., WANG R., ZHOU K., PAN M., BAO H.: An efficient GPU-based approach for interactive global illumination. In *ACM Trans. Graph. (Proc. SIGGRAPH 2009)* (2009), pp. 91:1–91:8. 2

[Yee04] YEE H.: A perceptual metric for production testing. *Journal of Graphics Tools 9*, 4 (2004), 33–40. 3, 4