Pixelcuts: Scalable Approximate Illumination from Many Point Lights

Pramook Khungurn, Thatchaphol Saranurak, and Chakrit Watcharopas
Kasetsart University, Bangkok, 10900, Thailand
Email: fscipmk@ku.ac.th, everyday.is.good@gmail.com, fscickw@ku.ac.th

Abstract

We present pixelcuts*, a scalable algorithm to compute approximate low-frequency illumination from many point lights. Its running time is $\widetilde{O}(n+mk)$ where n is the number of pixels, m is the number of point lights, and k is a constant the user specifies. Our algorithm was inspired by the* lightcuts *approach, which runs in $\widetilde{O}(nk+m)$ [15]. Lightcuts builds a tree of point lights and, for each pixel, gathers contribution from $O(k)$ light groups. On the contrary, pixelcuts builds several trees of pixels, and, for each light, scatters its contribution to $O(k)$ pixel groups. We demonstrate that pixelcuts outperforms lightcuts when there are many more pixels than lights. Nevertheless, pixelcuts is limited to low-frequency illumination. It is also prone to structured noise if too few trees are used or if the pixels are divided into too few clusters.*

Keywords: computer graphics, rendering, illumination, many-light problem

1. Introduction

One goal of computer graphics is to simulate complex illumination effects such as soft shadows, color bleeding, and caustics. To achieve the goal, one typically needs to gather radiance to each visible points from other points in the scene. Gathering is often formulated as a two-dimensional integral over either all the incoming light direction or all the relevant surfaces [6, 3].

One technique to evaluate the integral is to approximate outgoing radiance in the scene with a finite number of point light sources. This reduces the two-dimensional integral to a sum of contribution of each point light. Though a gross simplification, the technique can simulate many types of interesting illumination such as diffuse interreflection [7], environmental lighting, and illumination from area light sources.

To create artifact-free images, however, hundreds and even thousands of point lights are needed. Keller used around 250 lights to render global illumination in diffuse scenes [7]. Agarwal et al. observed that 300 lights were sufficient to approximate environmental maps [1], but Walter et al. suggested 3,000 [15]. Recently, Nichols et al. approximated a large area light with 256 point lights [9].

Exhaustively gathering contribution from each light to each visible point is very expensive. If there are n pixels in the image and m point lights in the scene, computing pairwise contribution takes $O(nm)$ time, making the naive approach unscalable. Many techniques have been invented to reduce the gathering time, and ours is in the same line of research.

In this paper, we present *pixelcuts*, a scalable algorithm to compute approximate low-frequency illumination from many point lights. Our algorithm achieves speedup by dividing the pixels into $O(k)$ groups, where k is a constant the user chooses. It then scatters each light's contribution to the groups in such a way that all pixels in the same group receive the same contribution at once without exhaustively adding the contribution to individual pixels. This process is carried out for each light, and thus the overall time complexity of our algorithm is $\widetilde{O}(n+mk)$ where the $\widetilde{O}(n)$ term is the cost of preprocessing the pixels for easy grouping and "propagating" the lights' contribution.

Pixelcuts can be thought of as the opposite of *lightcuts* [15], which groups lights instead of pixels and runs in $\widetilde{O}(nk+m)$ time. Hence, pixelcuts runs much faster when there are many more pixels than lights. However, it uses more memory and cannot produce high-frequency details such as sharp shadows or high gloss. Pixelcuts also has difficulty in rendering contact shadows, so it is more suitable for indirect illumination than direct illumination.

2. Related Works

Many-Light Problem. Approximating environment maps and area light sources with point lights is considered a standard technique. However, Keller [7] was the first to point out that the rendering equation [6] can be solved by tracing light particles to generate *virtual point lights* (VPLs), and then accumulating their contribution to the pixels.

Because high-quality images require a large number of point lights, researchers have come up with many techniques to speedup gathering. Wald et al. [13] suggested dividing both the pixels and the lights into k (≈ 25) groups, forming k pairs. Each pixel group gathers contribution from its corresponding light group with

the naive algorithm. The resulting image is noisy, but the noise is later removed by filtering. The time complexity of the algorithm is still $O(nm)$, however.

Nichols et al. [9] proposed rendering a very low-resolution image first, and then iteratively upsample the image and reshade pixels at discontinuity until it reaches the desired resolution. The algorithm's running time depends on the visual complexity of the final image but can be $O(nm)$ in the worst case.

Matrix row-column sampling [5] reduces m ($\approx 100,000$) lights to k ($\approx 1,000$) lights. It does so by clustering based on the responses of ℓ (≈ 300) pixels to each light. Depending on how the clustering is performed, its time complexity can be as large as $O(nk + m\ell k)$. The advantage of this approach, however, is that it can be accelerated with the GPU.

Another class of algorithms first builds a tree of lights. For each pixel, they use the tree to arrange lights into a small number of groups, and gather contribution from each group as a whole [10, 15, 8]. We use many concepts from lightcuts [15], so we shall review it in details.

Lightcuts. Let $\mathbb{S}$ be the set of point lights. The outgoing radiance L_o at point x in direction ω_o due to illumination by all point lights is given by

$$L_o^{\mathbb{S}}(x, \omega_o) = \sum_{i \in \mathbb{S}} M(x, \omega_o, \omega_{x,i}) G_i(x) V_i(x) I_i$$

where $V_i(x)$ is the visibility between x and Light i, $G_i(x)$ is the geometry term which depends on the light's type and the relative position between x and the light, and I_i is the light's intensity. The material term $M(x, \omega_o, \omega_{x,i})$ is equal to $f(x, \omega_o, \omega_{x,i}) \cos\theta_i$ where f is the BRDF, $\omega_{x,i}$ is the direction from x to the light, and θ_i is the angle between $\omega_{x,i}$ and the surface normal at x.

For each pixel, the algorithm partitions the lights into at most k clusters, where k is a constant set by the user. For each cluster $\mathbb{C}$, the algorithm randomly picks a representative light j, and computes the cluster intensity $I_{\mathbb{C}} = \sum_{i \in \mathbb{C}} I_i$. It then approximates the outgoing radiance due to $\mathbb{C}$ by

$$\begin{aligned} L_o^{\mathbb{C}}(x, \omega_o) &= \sum_{i \in \mathbb{C}} M(x, \omega_o, \omega_{x,i}) G_i(x) V_i(x) I_i \\ &\approx M(x, \omega_o, \omega_{x,j}) G_j(x) V_j(x) I_{\mathbb{C}}. \end{aligned}$$

As a result, the *total outgoing radiance estimate* at x is given by $L_o^{\mathbb{S}}(x, \omega_o) \approx \sum_{\ell=1}^{k} L_o^{\mathbb{C}_\ell}(x, \omega_o)$ where $\mathbb{C}_\ell$ denote the ℓth cluster.

The algorithm clusters lights by first building a tree of point light sources such that each light belongs to a leaf. Then, for each pixel, it chooses a *cut* of size at most k. A cut is a set of nodes in the tree such that any path from the root to a leaf must pass exactly one node in the cut as illustrated in Figure 1. Lights in the subtree rooted at the same node in the cut are considered to be in the same cluster.

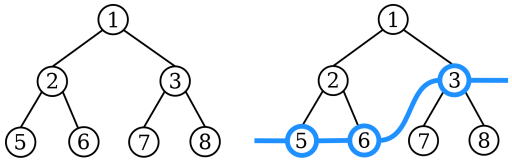

Figure 1: A light tree and a cut.

To choose a cut, the algorithm starts at the root node (the coarsest cut). Then, the node with the largest error upper bound is chosen to be replaced by its two children. The algorithm iterates until all of the nodes' error bounds are below a certain percentage (typically 2%) of the current total outgoing radiance estimate.

Lightcuts was later extended to handle motion blur, depth of field, and participating media [14]. The new approach, called *multidimensional lightcuts*, considers each pixel to be composed of multiple *gather points*, and the pixel's color to be the sum of contribution of every light to every gather point. It creates a tree of gather points and uses this tree together with the light tree to approximate the sum. In a sense, lightcuts can be seen as a special case of multidimensional lightcuts where the gather tree has only one node. However, pixelcuts is not a special case of multidimensional lightcuts because it constructs trees of pixels, not gather points. The trees are not used to approximate a single sum, but a function from pixels to their incoming radiance.

Spherical harmonics. Pixelcuts needs a way to compactly represent incoming radiance, a spherical function. We use the *spherical harmonics* (SH) basis which expands a spherical function $f(\omega)$ into a weighted sum of orthonormal functions $Y^{p,q}(\omega)$:

$$f(\omega) = \sum_{p=0}^{\infty} \sum_{q=-p}^{p} f^{p,q} Y^{p,q}(\omega).$$

Here, $f^{p,q}$ is the *spherical harmonics coefficient* whose value is given by

$$f^{p,q} = \int_{S^2} f(\omega) Y^{p,q}(\omega)\, d\omega$$

where S^2 is the unit sphere. To represent f, we store the coefficients with $p \leq 3$ (16 in total). Details on the use of spherical harmonics in computer graphics can be found in [4].

3. Pixelcuts

High level description. Pixelcuts seeks to approximate the incoming radiance function at each pixel location x due to illumination by all point lights:

$$L_{\text{in}}(x, \omega) = \sum_{i \in \mathbb{S}} G_i(x) V_i(x) I_i \delta(\omega - \omega_{x,i})$$

where δ is the Dirac delta function and G_i, V_i, $\omega_{x,i}$, and I_i are the same as previously discussed in the section on lightcuts. Projecting the function to the SH basis, the (p, q)-coefficient is given by

$$L_{\text{in}}^{p,q}(x) = \sum_{i \in \mathbb{S}} G_i(x) V_i(x) I_i Y^{p,q}(\omega_{x,i}).$$

To approximate $L_{\text{in}}^{p,q}(x)$, for each Light i, we divide the pixels into $O(k)$ clusters. For each cluster, we uniformly pick a representative pixel at random. Let y_i denote the representative pixel of the cluster for Light i that x belongs to. The SH coefficient is approximated as

$$\widetilde{L}_{\text{in}}^{p,q}(x) = \sum_{i \in \mathbb{S}} G_i(y_i) V_i(y_i) I_i Y^{p,q}(\omega_{y_i,i}), \qquad (1)$$

meaning that x receives the same contribution from Light i as the representative pixel of its cluster.

We now view the material term as a spherical function $M(x, \omega_o, \omega)$ where ω varies over the unit sphere. The outgoing radiance $L_o(x, \omega_o)$ is given by the convolution of the material term and the incoming radiance:

$$L_o(x, \omega_o) = \int_{S^2} M(x, \omega_o, \omega) L_{\text{in}}(x, \omega)\, d\omega.$$

To compute the integral, we project the material term to SH basis and let $M^{p,q}(x, \omega_o)$ denote its (p, q)-coefficient. The integral reduces to a dot product between SH vectors of $M(x, \omega_o)$ and $L_{\text{in}}(x)$, the latter function being approximated by $\widetilde{L}_{\text{in}}(x)$. Only the first 16 coefficients are used:

$$\begin{aligned} L_o(x, \omega_o) &= \sum_{p=0}^{\infty} \sum_{q=-p}^{p} M^{p,q}(x, \omega_o) L_{\text{in}}^{p,q}(x) \\ &\approx \sum_{p=0}^{3} \sum_{q=-p}^{p} M^{p,q}(x, \omega_o) \widetilde{L}_{\text{in}}^{p,q}(x) \end{aligned}$$

Pixel tree. We construct a tree of pixels and then use it to choose a cut for each light. We want a tree such that the pixels in the same subtree are spatially near one another. In this way, each pixel x is near its cluster's representative y, implying that the $G_i(y_i)$ and $Y^{p,q}(\omega_{y_i,i})$ do not differ from $G_i(x)$ and $Y^{p,q}(\omega_{x,i})$ much, respectively.

In our implementation, the light tree is a kd-tree of the pixels' 3D positions. For each tree node, we find the axis-aligned bounding box (AABB) of all the positions in its subtree, and choose the splitting plane that cuts the longest axis of the AABB in half. This method suits our need pretty well because it tries to make the AABB small as soon as possible.

In addition to the standard information such as the splitting plane and pointers to the children, each node of the kd-tree stores the AABB of the pixels in its subtree. This information will be used to refine cuts in the next section.

Choosing a cut. The process of choosing a cut is divided into two steps. In the first step, we pick a random cut of size at most k. We start with the coarsest cut, the root node. We then randomly pick a node to refine. The probability of a node being picked is proportional to the number of pixels in its subtree. The node is replaced by its two children if it is not a leaf. We repeat this process until the cut has size k.

In the second step, we refine nodes in the initial cut that might introduce large errors. Our aim is to reduce the range of $G_i(x)V_i(x)I_iY^{p,q}(\omega_{x,i})$ as x takes on the positions of the pixels in the cluster. (We consider only the geometry term because bounding the visibility term and the SH basis functions is hard.) The criteria of when to refine a node are as follows:

- For directional light, the geometry term is always 1. So we never refine any node.
- For omni light, the geometry term is $\frac{1}{\|p_i - x\|^2}$, where p_i is the position of the light. Let x_{near} and x_{far} be the points in the AABB of the node nearest and farthest for the light, respectively. We refine the node if $G_i(x_{\text{near}}) - G_i(x_{\text{far}}) = \frac{1}{\|p_i - x_{\text{near}}\|^2} - \frac{1}{\|p_i - x_{\text{far}}\|^2}$ is greater than 0.01.
- For oriented light, the geometry term is $\frac{\max(\cos\phi_i, 0)}{\|p_i - x\|^2}$ where ϕ_i is the angle between the light's normal and the vector from the light's position to x. Let x_{near} and x_{far} be the same as those for point light. Let $c_{\min}$ be the minimum of $\cos\phi_i$ over all the points in the AABB, and $c_{\max}$ be the maximum. We refine the node if $\frac{c_{\max}}{\|p_i - x_{\text{near}}\|^2} - \frac{c_{\min}}{\|p_i - x_{\text{far}}\|^2}$ is greater than 0.01. Note that $c_{\min}$ and $c_{\max}$ can be found by inspecting the corners of the AABB.

We put all nodes in the initial cut in a binary heap and repeatedly refine the node with the largest range. We do so until no more refinement are needed or until the cut size exceeds $10k$.

Scattering contribution. We still need to compute the sum (1) for each pixel. Doing so naively takes $O(nm)$ time. The crux of our algorithm is to compute the sums for all n pixels in $O(n + mk)$ time.

We linearize the tree by doing an inorder traversal, appending a leaf to an array as we find it. Notice that an internal node corresponds to an interval of cells as in Figure 2, and thus a cut is a partitioning of the array into intervals.

If we write the contribution of a light to $L_{\text{in}}^{p,q}$ of each pixel in the array, we get $O(k)$ blocks of constant values. If we find the difference between consecutive cells, there will be $O(k)$ non-zero elements. Hence the contribution of one light to all pixels can be represented by $O(k)$ numbers instead of n. Note that we can compute these numbers directly from the cut by finding the difference between contribution of cuts that are next to each other in the array. Next, we can accumulate these non-zero elements of all the lights

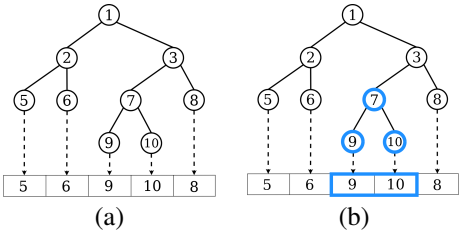

Figure 2: A pixel tree being linearized (a) and the correspondence between an internal node and an interval of cells (b).

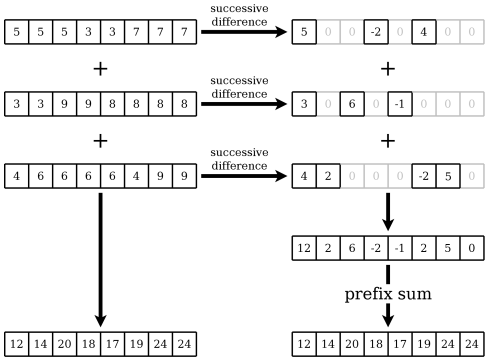

Figure 3: How we accumulate contribution of all the ligths. Each light's contribution is an array with constant blocks. We find non-zero elements of successive difference array directly from the cut. We put non-zero successive differences into an array, and compute the prefix sum to recover the original sum.

in an array, and find the *prefix sum*[1] of the array to accumulate the contribution of all the lights. This process is illustrated in Figure 3. Because there are m lights and each light gives rise to $O(k)$ non-zero elements and performing a prefix sum takes $O(n)$ time, the overall time complexity is $O(n + mk)$.

Removing artifacts. As can be seen in Figure 4a, using only one kd-tree results in images with axis-aligned blocks of constant colors. This is due to the fact that pixels were splitted by axis-aligned planes.

To remove the artifacts, we randomly divide the lights into ℓ groups. For each group, we build a tree and have all lights in the group choose cuts from that tree. To avoid axis-aligned blocks, we pick a random rotation and rotate all the pixels' positions before constructing the kd-tree. From our experience, setting $\ell = 16$ removes most of the artifacts, but few sharp boundaries may still be visible. We hide these corners by bilateral filtering of the incoming light in the image space. Currently, we use a 5×5 window. The SH coefficient $L_{\text{in}}^{p,q}(x_{0,0})$ of the center pixel is given by:

$$L_{\text{in}}^{p,q}(x_{0,0}) = \frac{1}{w} \sum_{-2 \le i,j \le 2} w_{i,j} L_{\text{in}}^{p,q}(x_{i,j})$$

[1]The prefix sum of sequence $a_1, a_2, \ldots$ is a sequence $b_1, b_2, \ldots$ such that $b_i = a_1 + \cdots + a_i$.

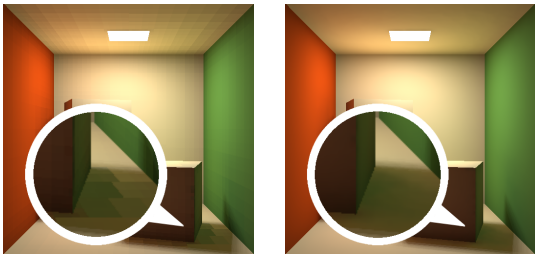

Figure 4: Using only one kd-tree results in the image being blocky (a). Using 16 trees and filtering removes most of the artifacts (b). The images were tone-mapped to make edges visible.

where

$$w_{i,j} = \frac{(n_{i,j} \cdot n_{0,0})^{32}}{1 + 10\|x_{i,j} - x_{0,0}\|},$$

$w = \sum_{i,j} w_{i,j}$, and $n_{i,j}$ is the normal at $x_{i,j}$. These weights are taken from [12].

Complexity analysis. Building a pixel tree takes $O(n \log n)$ time. Selecting a cut for a light takes $O(k \log k)$ due to the use of binary heap. Thus, choosing cuts for m lights takes $O(mk \log k)$ time. Computing the contribution of each light to each pixel cluster takes $O(mk \log P)$ where P is the number of geometric primitives in the scene because we trace rays to determine visibility. Accumulating contribution from all lights takes $O(n + mk)$ time, and filtering takes $O(n)$. Hence, our algorithm takes $\widetilde{O}(n + mk)$ time, ignoring the logarithmic factors. It uses $O(n)$ space to store the pixelxel tree and an SH vector for every pixel.

Limitations and applications. Pixelcuts approximates incoming light with few SH coefficients, resulting in low-frequency incoming light. It also assumes that all pixels in the same cluster receive the same contribution, which might not be true due to visibility. These approximations made pixelcuts unable to capture sharp shadows. So, it cannot be used to render images with high-frequency details such as one with a small but bright light source, and one with highly glossy materials.

Pixelcuts can be used to render approximate illumination in the following situations:

- *Indirect illumination.* Instant radiosity algorithm [7] typically generates hundreds and thousands of VPLs. Pixelcuts can be used to accumulate their contribution fast, and the resulting image will have low error as it has been observed that indirect illumination is low in frequency [11].
- *Environmental maps and area lights.* Pixelcuts will yield plausible illumination. However, it will miss contact shadows of small objects and will produce light leaks. We surmise that these errors might be masked by ambient occlusion or Arikan's technique [2].

4. Results

We rendered images of five test scenes on a laptop with 2.66 GHz Intel Core 2 Duo processor with 4 GB of RAM. All the images are of 640×480 except that of Cornell Box which is 512×512. The reference images were generated by a brute-force $O(nm)$ renderer. In all scenes, point lights are divided into "direct lights" whose contributions are computed exhaustively and "approximate lights" whose contributions are computed by pixelcuts. Direct lights are bright and small light sources whose high-frequency illumination pixelcuts cannot capture.

In the Cornell Box, Sponza, and Sibenik scenes, the approximate lights are VPLs generated by instant radiosity. The Conference Room scene originally had 1,024 point lights to simulate a small area light. Later, 1,252 more VPLs were added to capture indirect illumination. The Flat Panel Box scene contains 4,096 lights arranged as a square grid on the textured wall. We made all the lights in the last two scenes approximate lights to test pixelcuts' handling of area light sources.

Pixelcuts' rendering times are given in Figure 6. Notice that the time in the Process Cut stage is proportional to the number of lights. The overhead of building pixel trees is low compared to computing light contributions. The Sibenik scene took unproportionally long time because there is a direct light to which tracing shadow rays is slow.

Figure 5 compares the total time used and the number of shadow rays traced by pixelcuts, lightcuts, and the reference brute-force renderer. The data shows that pixelcuts can be an order of magnitude faster than lightcuts when there are many more pixels than lights. Since all the algorithms spend most time tracing shadow rays, pixelcuts is faster because it traces much fewer rays.

Pixelcuts produced high quality images when used to render indirect illumination such as that in the first three scenes. However, in the Conference Room scene, shadows of the chair legs are missing and all the shadows are softer than the reference. Worst, some light leaked to the chair legs as a result of pixels on the legs being grouped together with pixels on the floor. Moreover, in the Flat Panel Box scene, the shadows of the dragon's feet are not completely dark. These errors are due to both filtering and the approximations that make light contribution to nearby pixels the same. They demonstrate that pixelcuts cannot handle high-frequency details well.

Scene	Time (s)			Shadow Rays (10^6)		
	PC	LC	Ref	PC	LC	Ref
Cornell	29	290	650	4.8	64.6	439
Sponza	27	576	1616	3.8	83.9	546
Sibenik	52	510	3349	6.3	67.6	902
Conf.	39	345	1775	7.4	70.7	699
Flat	59	157	2435	12.9	26.1	1258

Figure 5: Time used and numbers of shadow rays traced by pixelcuts (PC), lightcuts (LC), and the reference algorithm (Ref).

5. Conclusion

Pixelcuts is a scalable algorithm for rendering approximate low-frequency illumination from many point lights. It is faster than lightcuts when there are much fewer lights than pixels. Its main limitation is the inability to produce high-frequency details, making it suitable for only rendering indirect illumination. Pixelcuts can be used to render illumination from large area lights or environmental maps, but the resulting images will contain light leaks.

Future directions include accelerating pixelcuts with the GPU and combining it with lightcuts to produce an algorithm that traces even fewer rays. Using occlusion by nearby objects to each pixel to prevent light leaks is another interesting direction to take.

Acknowledgements. We thank Toshiya Hachisuka, Marko Drabovic, Greg Ward, and the Stanford 3D Scanning Repository for the 3D models used.

References

[1] S. Agarwal, R. Ramamoorthi, S. Belongie, and H. W. Jensen. Structured importance sampling of environment maps. In *Proc. SIGGRAPH 2003*, pages 605–612, 2003.

[2] O. Arikan, D. A. Forsyth, and J. F. O'Brien. Fast and detailed approximate global illumination by irradiance decomposition. *ACM Trans. Graph.*, 24(3):1108–1114, 2005.

[3] R. L. Cook, T. Porter, and L. Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, 18(3):137–145, 1984.

[4] R. Green. Spherical harmonic lighting: The gritty details. In *Archives of the Game Developers Conf.*, March 2003.

[5] M. Hašan, F. Pellacini, and K. Bala. Matrix row-column sampling for the many-light problem. In *Proc. SIGGRAPH 2007*, page 26, 2007.

[6] J. T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, 20(4):143–150, 1986.

[7] A. Keller. Instant radiosity. In *Proc. SIGGRAPH 1997*, pages 49–56, 1997.

[8] H. Ki and K. Oh. A gpu-based light hierarchy for real-time approximate illumination. *The Visual Computer*, 24(7-9):649–658, July 2008.

[9] G. Nichols, R. Penmatsa, and C. Wyman. Interactive, multiresolution image-space rendering for dynamic area lighting. *Computer Graphics Forum*, 29(4):1279–1288, 2010.

[10] E. Paquette, P. Poulin, and G. Drettakis. A light hierarchy for fast rendering of scenes with many lights. In *Proc. Eurographics 1998*, pages 63–74, 1998.

[11] T. Ritschel, T. Grosch, M. H. Kim, H.-P. Seidel, C. Dachsbacher, and J. Kautz. Imperfect Shadow Maps for Efficient Computation of Indirect Illumination. *ACM Trans. Graph.*, 27(5), 2008.

[12] P.-P. Sloan, N. K. Govindaraju, D. Nowrouzezahrai, and J. Snyder. Image-based proxy accumulation for real-time soft global illumination. In *Proc. Pacific Graphics 2007*, pages 97–105, 2007.

[13] I. Wald, T. Kollig, C. Benthin, A. Keller, and P. Slusallek. Interactive global illumination using fast ray tracing. In *Proc. EGRW 2002*, pages 15–24, 2002.

[14] B. Walter, A. Arbree, K. Bala, and D. P. Greenberg. Multidimensional lightcuts. *ACM Trans. Graph.*, 25(3):1081–1088, 2006.

[15] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. P. Greenberg. Lightcuts: a scalable approach to illumination. In *Proc. SIGGRAPH 2005*, pages 1098–1107, 2005.

Scene	Scene Parameters			Time (s)				
	Polygons	Direct Lights	Approx Lights	Direct Illum.	Tree Build	Process Cut	Filter	Total Time
Cornell Box	86	25	1,674	8.08	3.30	16.06	1.51	29.22
Sponza	66,454	1	1,768	2.19	3.94	18.27	1.78	26.56
Sibenik	80,479	9	2,936	11.29	4.19	34.60	1.72	52.17
Conference Room	190,981	0	2,276	1.08	3.93	32.12	1.70	39.15
Flat Panel Box	115,096	0	4,096	0.76	3.93	53.29	1.69	59.95

Figure 6: The timing of the stages of pixelcuts for the five test scenes. The Process Cut stage includes selecting cuts, tracing shadow rays, and scattering light contribution to pixels. The Direct Illumination stage includes tracing eye rays. So even though there are no direct lights, the time used is not zero. All images were rendered using 16 trees ($\ell = 16$) and 2048 clusters per light before refinement ($k = 2048$).

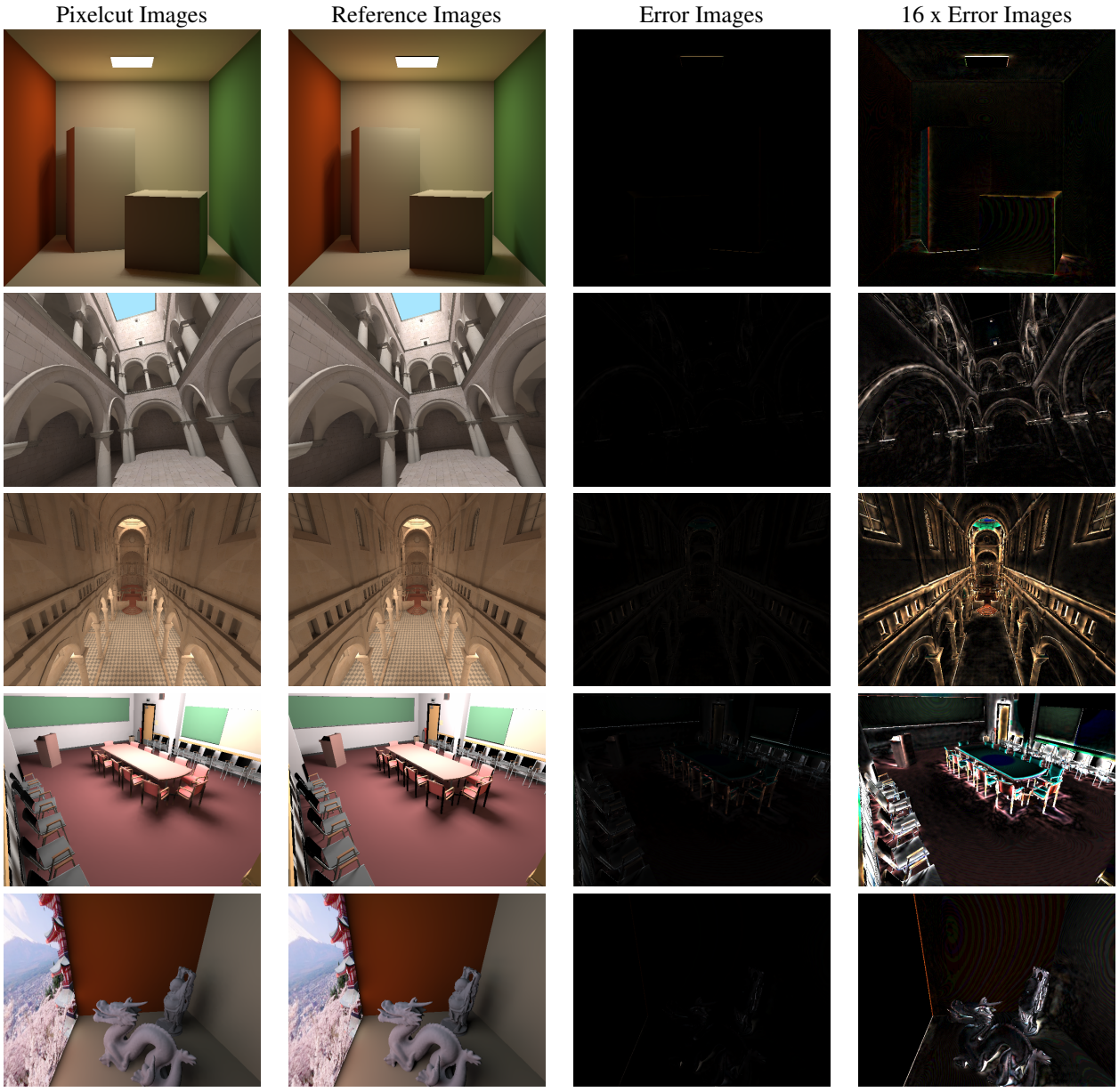

Figure 7: Comparison of images produced by pixelcuts and reference images. The scenes are, from top to bottom, Cornell Box, Sponza, Sibenik, Conference Room, and Flat Panel Box.