Vis Comput (2011) 27: 759–768
DOI 10.1007/s00371-011-0562-2

ORIGINAL ARTICLE

Image-space hierarchical coherence buffer

Kien T. Nguyen · Hanyoung Jang · JungHyun Han

Published online: 14 April 2011

Abstract Indirect illumination plays an important role in global illumination. However, computing indirect illumination is a time-consuming process and needs to be approximated to achieve interactive performance. Indirect illumination varies rather slowly across the surface. This leads to the idea of computing indirect illumination sparsely in the scene and interpolating the result. This paper presents a hierarchical structure, which enables efficient sampling. The hierarchy is constructed in the image space by exploiting coherences among the screen-space pixels. From the hierarchy, samples are chosen, each of which represents a group of coherent pixels. This paper presents two methods of utilizing the samples for indirect lighting computation. The methods produce plausible lighting results and show high performances. The proposed algorithms run entirely in the image space and are easy to implement in contemporary graphic hardware.

Keywords Image-based algorithm · Indirect lighting · Global illumination

1 Introduction

Global illumination (GI) plays a key role in adding realistic lighting to a 3D scene. With the increasing computing power of GPUs, much attention has been directed to exploiting the GPUs in order to achieve GI at interactive frame rates. In general, some approximations are needed to achieve the interactive GI.

K.T. Nguyen · H. Jang · J.H. Han (✉)
Korea University, Seoul, Korea
e-mail: jhan@korea.ac.kr

Indirect illumination varies rather slowly across the surfaces, especially the diffuse surfaces, and is view-independent. These features enable us to compute indirect illumination sparsely for the samples of the scene and then interpolate the results over the entire scene. This idea is widely applied in irradiance caching [7, 9, 17, 20] and some image-based lighting techniques [1, 12, 13].

This paper first of all proposes to compute the coherences among the screen-space pixels. They are stored in a data structure. It enables efficient identification of the samples for which indirect illumination needs to be computed. Once indirect lighting is computed for every sample, the result is used for interpolation. For the purpose, this paper proposes two methods: One computes samples at each frame, and the other caches the samples from the previous frames and reuses them in the subsequent frames.

2 Related work

A problem in GI is the high cost of computing indirect lighting. Approximation of the physically correct illumination is usually needed. Sampling and interpolation are widely adopted in GI based on the fact that lighting smoothly lays down over object surfaces.

Ward et al. [20] introduced irradiance caching. This method computes irradiance at the samples. Each sample is defined with a radius indicating its influence region. Irradiance at a point is interpolated from the irradiance at the samples if the point is in the influence regions of those samples. Tabellion and Lamorlette [17] applied irradiance caching to film production. Gautron [6] presented (ir)radiance cache implementation on GPU through splatting samples into the image space. Herzog et al. [7] introduced anisotropic cache

splatting in the 3D object space. Debattista et al. [3] proposed instant caching which is a combination of irradiance caching and instant radiosity [8]. (Ir)radiance caching is also extended to work with glossy surfaces [4, 9]. Most of these methods add a new point to the cache whenever there is no information to interpolate illumination on it from the previously computed samples in the cache.

Instant radiosity [8] computes single-bounce indirect lighting using a set of virtual point lights (VPLs) which are sampled from the visible surfaces seen from the light source. Dachsbacher and Stamminger [1] proposed reflective shadow map that holds information for VPLs. The samples are chosen uniformly from a lower-resolution representation of the screen. The illumination at samples is gathered from VPLs. Illumination at a screen pixel is computed by interpolating nearby samples if it is coherent to those samples. Ritschel et al. [15] proposed imperfect shadow maps to produce occlusion from VPLs. A crude point-based representation of geometry is used to compute shadow map from each VPL. The lightcuts framework [19] represents lighting as a hierarchical collection of point lights. For each pixel sample, it computes the pixel radiance by adaptively choosing a subset of important VPLs. Recently, Ritschel et al. [14] proposed to rasterize a hierarchy of point samples into a micro-buffer to accelerate final illumination gathering.

Nichols and Wyman [13] detect mipmap-based discontinuities in the screen. Illumination at a point of a discontinuous region is gathered from VPLs. Nichols et al. [12] improve the work by using stencil-based multiresolution techniques, and also extended their work for dynamic area lighting [11]. The sample points are chosen based on discontinuities, and there often exist too many samples. Consequently, the computation cost is increased at the gathering stage.

Most previous methods detect coherences in object space or discontinuities in image space. In contrast, our method is image-based, and exploits the coherences among pixels in screen space.

3 Hierarchical coherence buffer and samples

For rendering, the scene is transformed from the world space to the screen space. Then various per-pixel attributes of the visible surfaces, such as positions, normals, and materials, can be stored in an off-screen buffer named *G-buffer* [16]. This section presents how to build a data structure that stores the coherences among the screen-space pixels in the G-buffer. It accelerates finding a group of samples, each of which is a representative of a set of coherent texels.

3.1 Coherence measurements

Coherence between two texels in the G-buffer corresponds to the coherence between two world-space points which

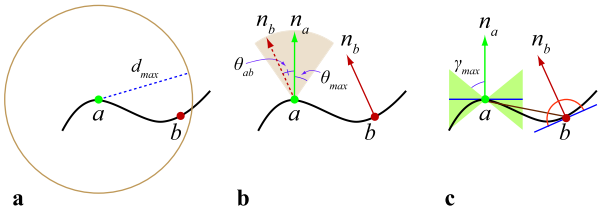

Fig. 1 Three types of coherence: (**a**) distance coherence; (**b**) normal coherence; (**c**) pose coherence

are projected at the texels. Coherence between two world-space points is measured using their positions and normals [17, 18, 20]. Consider two points, a and b, in Fig. 1(a). They are *distance-coherent* if they are within a pre-defined distance $d_{\max}$:

$$\|a - b\| \leq d_{\max} \tag{1}$$

Let us denote the surface normals at a and b by n_a and n_b, respectively, as illustrated in Fig. 1(b). Two points are *normal-coherent* if the normal divergence, θ_{ab}, is less than or equal to a pre-defined angle $\theta_{\max}$. The condition for normal coherence can be redefined using the dot product of n_a and n_b:

$$n_a \cdot n_b \geq \cos\theta_{\max} \tag{2}$$

Now consider Fig. 1(c). Surface point a may contribute to lighting at b that is located at a's back side. Inversely, lighting at a may be affected by b. This is often called *backward lighting*. Let us connect a to b and denote the connecting vector by $\overrightarrow{ab}$. (The vector of the opposite direction is denoted by $\overrightarrow{ba}$.) We say b is *pose-coherent* to a if the following condition is satisfied:

$$\begin{cases} \overrightarrow{ab} \cdot n_a \leq 0 \quad \text{or} \quad \overrightarrow{ba} \cdot n_b \leq 0 \\ \left|\overrightarrow{ab} \cdot n_a\right| \leq \cos\gamma_{\max} \end{cases} \tag{3}$$

The first term tests for the presence of backward lighting, and the second term uses a pre-defined angle $\gamma_{\max}$ to restrict the pose-coherent angles of backward lighting.

3.2 Hierarchical coherence buffer

We call the data structure storing the coherences among the G-buffer texels the *hierarchical coherence buffer* (HCB). It has a mipmap-like structure, where level k in the hierarchy is of a quarter size of level $(k-1)$. The base level (level 0) of the HCB is initialized to the G-buffer. Then the upper levels are recursively constructed in a bottom-up fashion. See Fig. 2. A texel located at (x, y) of level k holds the coherence information of 3×3 child texels of level $(k-1)$, which are located at $(2x, 2y) + (i, j)$, where $i, j = \{-1, 0, 1\}$.

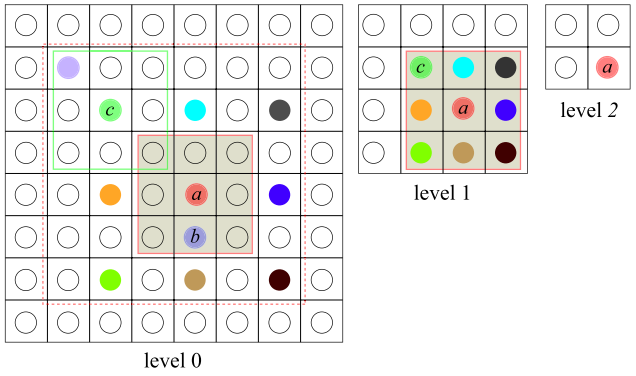

Fig. 2 Hierarchically organized coherences. To compute information of a texel at the coordinates (x, y) of level k, coherences are checked within a 3×3 block of texels centered at $(2x, 2y)$ of level $(k-1)$. Not only the coherences but also the position and normal data are stored at each texel

Let us first present how the "normal coherence" is recorded in the HCB. At level 0 of Fig. 2, consider the 3×3 texels centered at a and one of a's neighbors denoted by b. If (2) is satisfied, a and b are normal-coherent. If all of the eight neighbors of a are normal-coherent to a, the 3×3 texel block is called *group-coherent* (more precisely, group-coherent with respect to normals), and the minimum of $(n_a \cdot n_b)$s is stored at level-1 texel a in the figure. It is the least value of group coherence. On the other hand, if any of the eight neighbors is not normal-coherent to a, the block of 3×3 texels is not considered group-coherent, and zero denoting 'incoherence' is stored at level-1 texel a.

Each texel of the HCB contains not only the coherence information but also the position and normal of the corresponding world-space point. For level 0 of the HCB, the position and normal data stored at the G-buffer are simply copied. For the upper levels, the position and normal stored at the center of the 3×3 texel block at level $(k-1)$ are copied to level k. For example, in Fig. 2, the normal stored at level-0 texel a is copied to level-1 texel a. Let us name it n_a. To save the memory cost, the group coherence information (either the minimum of $(n_a \cdot n_b)$s or zero) is stored at the w-component of n_a at level 1, which we denote by n_a^w.

Once all of the texels at level 1 are filled, we start to build level 2. In Fig. 2, consider the 3×3 texel block centered at a of level 1. We check the coherence between a and its neighbors, one of which is denoted by c in the figure. Then the group coherence information will be recorded at level-2 texel a. It is important to note that the group coherence we want to measure is the one among the children of a and c at level 0 because the level-2 texel a needs to store the coherence information among all of its grandchildren at level 0. Thus, we need to combine the group coherence information stored at n_a^w and n_c^w. The next subsection presents a general idea for combining them.

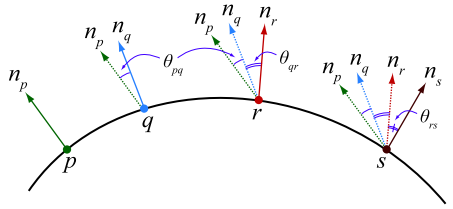

Fig. 3 Combining normal coherences

3.3 Combining normal coherences

Consider three points, p, q, and r, in Fig. 3. Assume that p is found to be normal-coherent to q, i.e., $\theta_{pq} \le \theta_{\max}$ (or equivalently $n_p \cdot n_q \ge \cos\theta_{\max}$). Now suppose that r needs to be checked for normal coherence with both p and q but is allowed to be compared with q only. Then a conservative solution is to compute θ_{qr}, add it to θ_{pq}, and then compare the sum with $\theta_{\max}$. Let us denote θ_{pq} and θ_{qr} by α and β, respectively. Then the coherence condition among p, q, and r can be listed as follows:

$$\begin{cases} \alpha \le \theta_{\max} \\ \beta \le \theta_{\max} \\ \alpha + \beta \le \theta_{\max} \end{cases} \tag{4}$$

Note that, given the last term, the second term is redundant and can be deleted. Further, the first and last terms of (4) can be rephrased as follows:

$$\begin{cases} \cos\alpha \ge \cos\theta_{\max} \\ \cos(\alpha+\beta) \ge \cos\theta_{\max} \end{cases} \tag{5}$$

Assuming that the first term in (5) is satisfied for points p and q, i.e., p and q are known to be normal-coherent, only the second term defines the condition for point r to be normal-coherent to both p and q.

Using the trigonometric cosine function, we can derive the following from the second term of (5):

$$\cos\alpha\cos\beta \ge \cos\theta_{\max} \tag{6}$$

Equation (6) significantly reduces the computing cost. For the last term of (4), we would have to compute the angle β using inverse cosine function, i.e., $\beta = \cos^{-1}(n_q \cdot n_r)$. For (6), it is replaced by cheaper dot product operation, i.e., $\cos\beta = n_q \cdot n_r$.

Equation (6) asserts that determining the coherence of $\{p, q, r\}$ is reduced to checking the multiplication of the known coherence of $\{p, q\}$ and the newly-observed coherence of $\{q, r\}$. Now suppose that another point s needs to be checked for normal coherence with $\{p, q, r\}$ but is allowed to be compared with r only. Then $\cos\theta_{rs}$ is set to the dot product of n_r and n_s, and the following is tested:

$$(\cos\alpha\cos\beta)\cos\theta_{rs} \ge \cos\theta_{\max} \tag{7}$$

where $(\cos\alpha\cos\beta)$ represents the known coherence of $\{p, q, r\}$.

3.4 Coherence exploitation

Using the idea presented in the previous subsection, the problem raised in Sect. 3.2 can be solved, i.e., the group coherence information stored at n_a^w can be combined with n_c^w at level 1 (in Fig. 2). It is presented in the pseudo-code of Algorithm 1.

Algorithm 1 HCB construction

```
INPUT: {a, n_a, n_a^w}, {c_i, n_{c_i}, n_{c_i}^w : i = 1..8}
coherence = n_a^w
bCoherence = True
for i = 1 to 8 do
    temp = n_{c_i}^w * dot(n_a, n_{c_i})
    n_coherence = temp ≥ cos θ_max
    p_coherence = test_pose_coherence(a, n_a, c_i, n_{c_i})
    if n_coherence == True and p_coherence == True
    then
        coherence = min(coherence, temp)
    else
        bCoherence = False
    end if
end for
n_a^w = (bCoherence == True) ? coherence : 0
OUTPUT: {a, n_a, n_a^w}
```

In Algorithm 1, c_is denote eight neighbors of a. The world-space normal and the group coherence information of c_i are denoted by n_{c_i} and $n_{c_i}^w$, respectively. First, we compute $n_a \cdot n_{c_i}$, then multiply it with $n_{c_i}^w$. This implements $\cos\alpha\cos\beta$ in (6), where $n_{c_i}^w$ corresponds to $\cos\alpha$, and $n_a \cdot n_{c_i}$ corresponds to $\cos\beta$. If $n_{c_i}^w(n_a \cdot n_{c_i}) \geq \cos\theta_{\max}$, c_i's child texels are determined to be all normal-coherent to a. Then an additional test for pose coherence is invoked. If both normal and pose coherences are satisfied, c_i is said to be coherent to a.

The above procedure is applied to every c_i. If all c_is are found to be coherent to a, the minimum of n_a^w and $n_{c_i}^w(n_a \cdot n_{c_i})$s is selected, and stored at n_a^w of level-2 texel a. The position and normal of level-1 texel a are also copied to level-2 texel a.

Algorithm 1 recursively constructs the HCB in a bottom-up fashion, i.e., from level 1 to level 2, then from level 2 to level 3, and so on. Further, Algorithm 1 also constructs level 1 from level 0. For this purpose, the w-component of each texel's normal is set to 1 at the base level of the HCB.

3.5 Distance accumulation

At the base level of the HCB, the *distance* of a texel is initialized to be the texel spacing in the world space. It is the square root of the texel area back-projected to the world space, i.e., the area that the texel covers in the world space.

While exploiting the coherences within a 3×3 texel block, we also compute the *accumulated distance*. Suppose that, at level 0 of Fig. 2, R_a and R_b denote the distances stored at a and b, respectively. If b is found to be coherent to a, we compute the distance $R'_b = \|a - b\| + R_b$. Let b_is denote the eight neighbors of a. If all b_is are coherent to a, the maximum among R_a and R'_{b_i}s is stored at level-1 texel a. More specifically, it is stored at the w-component of a's position, which we denote by a^w.

This process is done recursively, i.e., the distance for every texel of level k is determined by accumulating the distances stored at the texels of level $(k-1)$. Then, for texel a at any level of the HCB, a^w defines a *sphere* that encompasses all the world-space points corresponding to the G-buffer texels coherent to a.

3.6 Sample identification

Suppose that the HCB has been constructed. Then, for each texel a of level k, n_a^w holds the group coherence information of 3×3 texels at level $(k-1)$. Recall that n_a^w is positive if the 3×3 texel block is group-coherent but is zero if the block is not.

For identifying the *samples* for which indirect lighting needs to be computed, the HCB is traversed from the top level. During the top-down traversal, each texel is checked to see if its coherence information is greater than zero. If so, it is *splatted* into the 3×3 texels of the child level. This is done recursively. The result of splatting at each level is stored in a *mask buffer* which has the same structure as the HCB. The texels in the mask buffer are classified as 'inside' or 'outside' the splat regions. The 'inside' texel implies that it is coherent to at least a texel of the parent level.

Once the top-down traversal is completed, the samples can be identified by using the HCB and the mask buffer. At the top level, a texel whose coherence information is greater than zero is taken as a sample. At the other levels, a texel is taken as a sample if its coherence information is greater than zero *and* it is 'outside' the splat regions. The samples identified at the various levels of the HCB are copied into a single buffer, called *sample buffer*, which has the same resolution as the G-buffer.

4 Splatting indirect illumination

Indirect lighting is *gathered* only at the samples. Then the results are used to determine the indirect lighting of the remaining G-buffer texels.

4.1 Gathering at the samples

There exist many algorithms for gathering indirect illumination, and the samples stored in our sample buffer are independent of the algorithms. In the current implementation, we adopted an algorithm based on the reflective shadow map (RSM) [1]. A subset of the RSM texels defines the virtual point lights. Then, at each sample of the sample buffer, indirect lighting is gathered from the virtual point lights [1, 12]. The result of indirect lighting is stored at each sample.

4.2 Weights for indirect illumination

Consider a sample a and a point b. The *coherence error* between a and b is defined as follows:

$$\epsilon_a(b, n_b) = \max\left\{\epsilon_a^1(b), \epsilon_a^2(n_b), \epsilon_a^3(b, n_b)\right\} \tag{8}$$

where

$$\epsilon_a^1(b) = \frac{\|a - b\|}{a^w}$$

$$\epsilon_a^2(n_b) = \frac{\sqrt{1 - n_a \cdot n_b}}{\sqrt{1 - \cos\theta_{\max}}}$$

$$\epsilon_a^3(b, n_b) = \rho\frac{\sqrt{|\overrightarrow{ab} \cdot n_a|}}{\sqrt{\cos\gamma_{\max}}}$$

$$\rho = \begin{cases} 1 & \text{if } \overrightarrow{ab} \cdot n_a \le 0 \text{ or } \overrightarrow{ba} \cdot n_b \le 0 \\ 0 & \text{else} \end{cases}$$

In (8), $\epsilon_a^1(b)$, $\epsilon_a^2(n_b)$, and $\epsilon_a^3(b, n_b)$ measure the distance, normal, and pose coherences, respectively, which are presented in Sect. 3.1. (Recall that a^w denotes the accumulated distance presented in Sect. 3.5.)

Point b is coherent to a if the following holds:

$$\epsilon_a(b, n_b) \le 1 \tag{9}$$

Then, in order to determine how much the indirect lighting computed at a contributes to lighting at b, the *weight* of a toward b is defined as follows:

$$w_a(b, n_b) = \left(1 - \epsilon_a^1(b)\right)\left(1 - \epsilon_a^2(n_b)\right)\left(1 - \epsilon_a^3(b, n_b)\right) \tag{10}$$

4.3 Splatting

Each sample dispatches its lighting to the texels located within the sphere, the radius of which has been computed at the time of creating the HCB. For the purpose, a *splatting* technique can be used [2, 6].

The splatted sphere is often approximated by a quad. The error between a point in the quad and the sample is measured using (8). If the coherence error is less than or equal to 1 as shown in (9), i.e., the point is coherent to the sample, the weight of the sample toward the point is computed using (10).

Multiple samples can contribute lighting to a point, and the lighting accumulated at a point is computed as follows:

$$E_b = \frac{\sum_{i,\epsilon_{s_i}(b,n_b)\le 1} w_{s_i}(b, n_b) E_{s_i}}{\sum_{i,\epsilon_{s_i}(b,n_b)\le 1} w_{s_i}(b, n_b)} \tag{11}$$

where s_i and E_{s_i} denote the ith sample's position and lighting, respectively. (The floating-point blending capabilities of graphics hardware enable us to efficiently implement (11), i.e., the numerator is accumulated in the RGB channels whereas the denominator is simultaneously accumulated in the alpha channel.)

Direct illumination is computed separately, and then is combined with indirect illumination to produce the final image of the scene. Figure 4 shows the result of rendering a scene.

5 Caching indirect illumination

Computing indirect lighting at a sample is expensive. Observe that a sample identified at a frame may still be visible in the successive frames. Then lighting computed at the sample can be cached and reused. This idea was originally proposed by Gautron et al. [6], and related discussion is made in Sect. 6.

The flow of the proposed algorithm for caching and reusing the indirect lighting is illustrated in Fig. 5. The G-buffer and RSM are refreshed at every frame and are independent of the proposed algorithm. Therefore, they are not shown in Fig. 5.

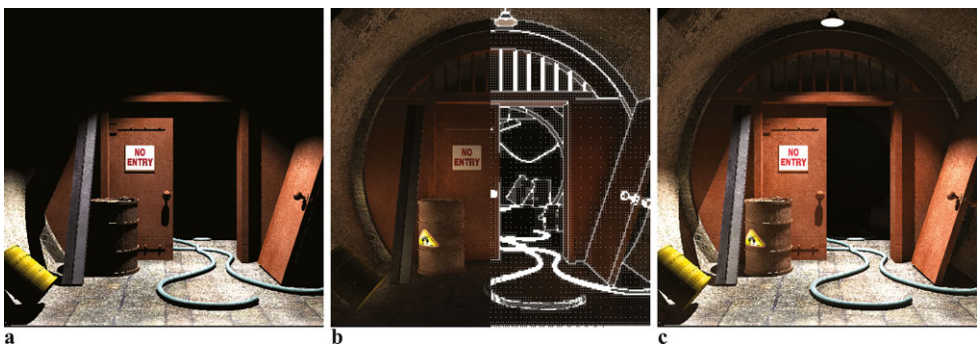

Fig. 4 Rendering result: (**a**) direct lighting; (**b**) indirect lighting (*left half*) and samples in *white dots* (*right half*); (**c**) combination of direct and indirect lighting (with 1024 VPLs)

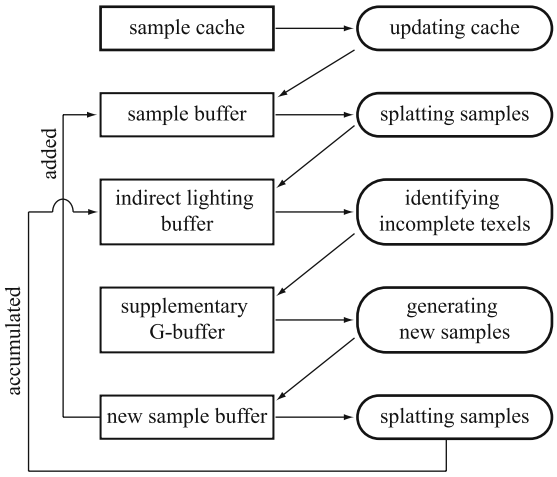

Fig. 5 Indirect illumination caching algorithm

5.1 Reusing samples

Suppose that, for the previous frame, indirect lighting has been computed for all samples. The samples and their lighting stored in the sample buffer will be reused for the current frame. Let us call the sample buffer *sample cache*. (A kind of *double buffering* will be used to store the samples. The sample cache stores the samples of the previous frame whereas the samples of the current frame will be recorded in the sample buffer.)

The scene may be dynamic, i.e., the objects in the scene may move. Then a subset of the cached samples may be found useless for the current frame and should be removed. For this purpose, the samples are transformed into the current view of the camera, and then the position and normal of each cached sample are compared to those stored in the G-buffer, which contains the geometry information of the current frame. If the difference in distance or normal divergence is greater than a pre-defined threshold, the sample is taken as *invalid* and is removed. An example can be found in Fig. 6(a) and (b).

Figure 6(b) shows that many invalid samples have been removed, but also reveals that unnecessarily many samples are still located at the low-frequency wall region. They are all valid samples, but we do not need such many samples, i.e., there exist *surplus* samples. When an invalid sample is removed, the nearby samples are removed to reduce the number of surplus samples. A predefined-size quad, centered at the invalid sample, is used to delimit the range of the surplus samples. Figure 6(a) illustrates the quad in red, and Fig. 6(c) shows that the number of surplus samples is significantly reduced and the low-frequency region contains a reasonable number of samples. In order to avoid taking a valid sample from the cleared quad, depth testing is adopted. A larger depth value is used for valid samples whereas a smaller value is used for invalid samples.

In Fig. 5, the process of removing invalid and surplus samples is performed by the module named "updating cache." The samples surviving "updating cache" are projected into the current view of the camera and stored into the sample buffer. The sample buffer is used to produce indirect lighting, as presented in Sect. 4.3, and the result is stored in the so-called *indirect lighting buffer*.

5.2 New sample generation

When the samples are splatted to the G-buffer, the weights presented in Sect. 4.3 are accumulated at each texel of the G-buffer. If the accumulated weight is less than a threshold value, the texel is taken as *incomplete*. See Fig. 7(a). The white dots are the samples in the sample buffer. The indirect lighting is computed using the samples, and the black area is composed of the incomplete texels.

Lighting needs to be computed for the incomplete texels. For the purpose, the G-buffer data at the incomplete texels' locations are copied to another G-buffer, which we call *supplementary G-buffer*. See the flowchart of Fig. 5. Then indirect lighting is computed using it, i.e., the HCB is constructed for the supplementary G-buffer, a new set of samples is identified, and indirect lighting is gathered at the new samples. Then the new samples are splatted to the original G-buffer, not to the supplementary G-buffer, such that lighting using the new samples is accumulated to lighting stored in the indirect lighting buffer. See Fig. 7(b). The green dots in the upper image represent the new samples. The lower

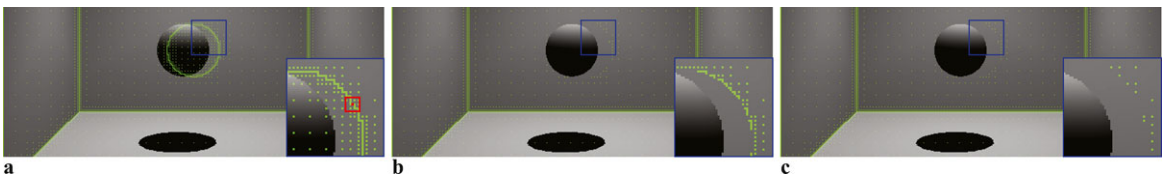

Fig. 6 Sample removal: (**a**) the sphere is moving to the left, and the samples cached from the previous frame are illustrated in *green dots*. Note that the samples are densely located at the high-frequency region; (**b**) as can be clearly observed in the zoomed-in box, the invalid samples on the sphere surface have been removed in the current frame; (**c**) the surplus samples have been removed. Returning to (**a**), observe the *red quad* centered at an invalid sample (also colored in *red*). Not only the invalid sample but also the surplus samples (within the red quad) have been removed

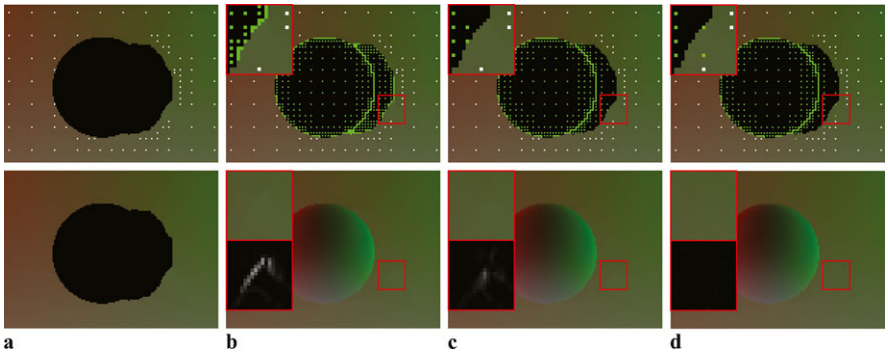

Fig. 7 Handling incomplete samples. The *upper row* shows samples. The cached samples are colored in *white*, and the new samples are in *green*. The *lower row* shows the rendering result using the samples of the upper row: (**a**) the *white dots* represent the cached samples surviving "updating cache" and the incomplete texels compose the *black area*; (**b**) new samples (in *green dots*) are generated and no incomplete texel exits any longer; (**c**) the number of new samples is decreased and the discontinuity artifact is reduced; (**d**) new samples can now locate outside the area of the incomplete texels, and the discontinuity artifact is further reduced

image is obtained by accumulating the contribution from the new samples to the lower image of Fig. 7(a).

Note that the new samples contribute only to the incomplete texels. Consequently, discontinuities often occur between the areas lit by the cached samples and that by the new samples. Such discontinuities can be found in Fig. 7(b).

In order to resolve the discontinuity artifact, first of all, let us fill the supplementary G-buffer with *all* data of the G-buffer, not with the data only at the incomplete texels' locations. Recall that, as presented in Sect. 3.4, coherence information of HCB texels is stored at the w-component of their normals, denoted as n_a^ws, and initialized to 1 at the base level of the HCB. It is modified: n_a^ws are assigned to 1 for incomplete texels and -1 for others. Then Algorithm 1 is accordingly modified such that the absolute value of n_a^w is used for computing the normal coherence but the sign of n_a^w is preserved when it is transferred to the upper level.

The procedure presented in Sect. 3.6 will identify only the samples whose n_a^ws are greater than zero, i.e., only the samples related with the incomplete texels. See the upper image of Fig. 7(c), and compare the number of new samples (green dots) with that of Fig. 7(b). It is decreased because the coherence is computed not only with the incomplete texels but also with the complete texels. Further, see the lower image of Fig. 7(c), and find that the discontinuity artifact is reduced because some samples contribute lighting to complete texels nearby the incomplete texels. However, note that the locations of new samples are restricted within incomplete texels.

Let us move one step further. Suppose that when the group coherence is computed in a 3×3 texel block at level $(k-1)$ for constructing the HCB, incomplete texel is detected from the block. Then the coherence information to be stored at level k is set to positive. By doing so, the center texel of the block can be taken as incomplete even though it is in fact complete. Consequently, the samples can be located outside the area of the incomplete texels, as can be found in Fig. 7(d), and the discontinuity artifact is further reduced.

The result of indirect lighting (stored in the indirect lighting buffer shown in Fig. 5) is combined with direct lighting to produce the final image of the scene. The new samples are added to the sample buffer such that the sample buffer can be provided as the sample cache for the next frame.

5.3 Recomputing indirect lighting

When lighting conditions are changed, indirect lighting at the samples should be recomputed. Fortunately, indirect lighting changes in a low frequency and, therefore, we do not have to recompute it every frame [5, 18]. In the current implementation, indirect lighting is recomputed at every four frames, and about a quarter of the samples is processed per frame. For this purpose, the sample buffer is partitioned into tiles, each of which is of 4×4-resolution index pattern shown in Fig. 8(a). When a new sample is added to the sample buffer, an index is assigned using the pattern. It works as a counter. For each frame, the counter is incremented by one at every sample, and indirect lighting is recomputed only for the samples whose counters reach four. Then their counters are reset to zero.

6 Experimental results and discussions

In the algorithm proposed in this paper, the samples are identified using only the geometry of the scene. Thus, it is straightforward to handle multiple light sources. Figure 9 shows the result of rendering a scene lit by multiple light sources. Compare the image with the one shown in Fig. 4 which is lit by a single light source.

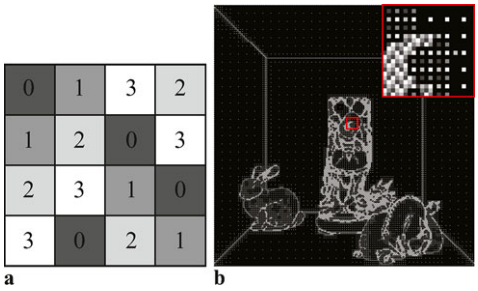

Fig. 8 Index pattern for recomputing the indirect lighting: (**a**) randomized index pattern; (**b**) using the index pattern, 33924 samples in a frame are divided into four groups of 10184, 7908, 8215 and 7617 samples. They are colored in *gray scales*

Figure 10 shows the final images together with the samples for two scenes. Table 1 shows the numbers and percentages (per level and per G-buffer) of the samples for the scenes in Fig. 10.

Our method could be considered an extension of the image-based lighting algorithms [1, 11–13]. However, our approach is different from them. Most of the previous methods detect discontinuities at a block of pixels in the screen in order to exploit coherences among the pixels. Nychols et al. [12] used a hierarchical structure for identifying the discontinuities. Our method also used a hierarchical structure, but directly checked the coherences among the pixels. Detecting the coherences helps reduce the number of samples compared to discontinuity detection. Figure 11 compares the

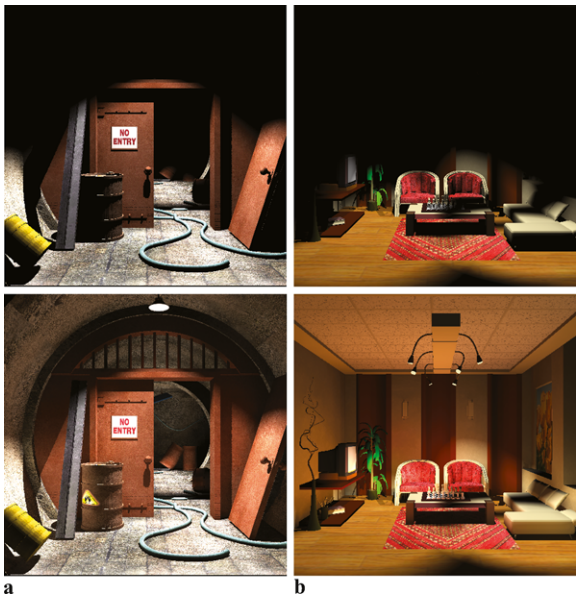

Fig. 9 Lighting with multiple light sources. The *upper row* shows the result of direct lighting, and the *lower row* shows the result of adding indirect lighting. (**a**) a scene with two light sources; (**b**) another scene with six light sources

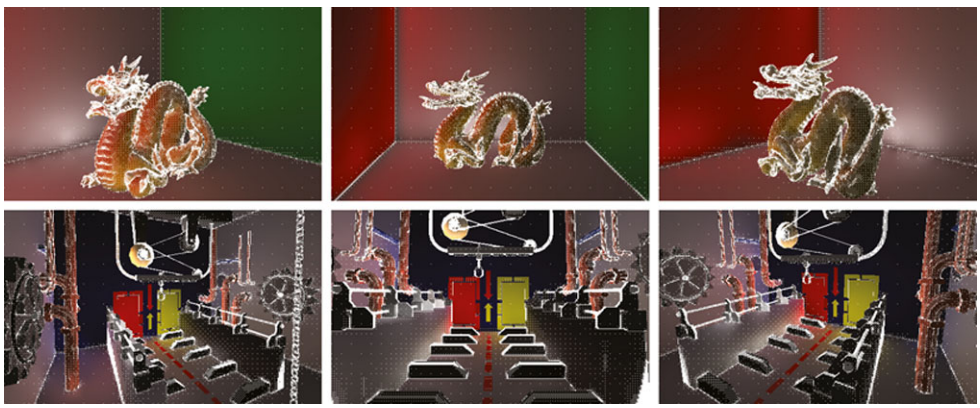

Fig. 10 Each scene is captured from three different viewpoints, and the samples are illustrated as *white dots*

Table 1 Statistics for the scenes of Fig. 10

Level (resolution)	Dragon scene			Factory scene			% of samples
	left	middle	right	left	middle	right	(per level)
0 (1024×1024)	6,976	7,736	7,035	16,635	29,321	14,669	0.9%
1 (512×512)	7,080	6,233	6,205	10,386	9,023	9,233	1.1%
2 (256×256)	3,702	3,087	3,310	5,817	4,209	5,071	1.3%
3 (128×128)	1,467	1,184	1,368	2,632	1,841	2,460	1.3%
4 (64×64)	478	470	473	1,003	715	1,023	1.4%
5 (32×32)	1,024	1,024	1,024	1,023	620	1,020	1.4%
Total	20,727	19,734	19,415	37,496	45,729	33,476	
% of samples (per G-buffer)	2.0%	1.9%	1.9%	3.6%	4.4%	3.2%	

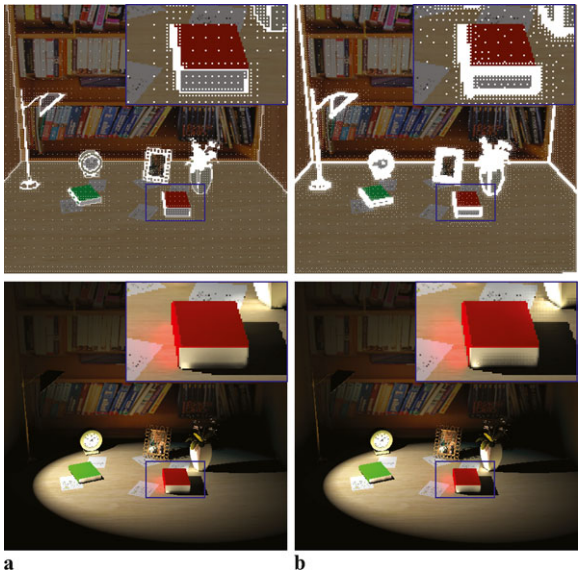

Fig. 11 Comparison of two methods: (**a**) our method (11742 samples, 128 fps); (**b**) Nychols et al. [12] (26170 samples, 56 fps). 1024 VPLs are used for both methods

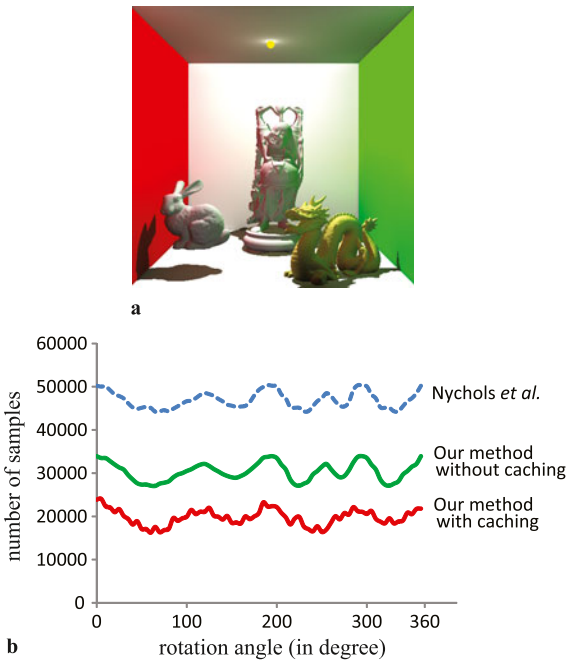

Fig. 12 Comparison of three methods; (**a**) a scene with rotating Bunny, Dragon, and Buddha; (**b**) the numbers of samples, where indirect lighting is computed, for a sequence of frames

numbers of samples between our method and Nychols et al. [12].

Note also that, unlike Nychols et al. [12], our method uses sample splatting which is less dependent on the current view and therefore the samples can be cached and reused. Figure 12 compares the per-frame numbers of samples, where indirect lighting is computed, of our methods and Nychols et al. [12] for a scene with three rotating objects.

Our caching framework was inspired by Gautron et al. [6], especially by the splatting technique of the cached samples. However, our work is distinct in many aspects. Gautron et al. [6] identified a sample first by using a software rasterizer in CPU and then used ray tracing to compute its influence region. In contrast, we used a hierarchical structure to exploit the coherences among the screen-space pixels and then identified the samples. (However, our method relied on roughly approximating the influence region of a sample.) Gautron et al. [6] transferred data between GPU and CPU, leading to degraded performance. In contrast, our method performs all computation within GPU.

In the previous image-based lighting algorithms [1, 11–13], all samples are recomputed per frame, and thus the different sets of samples between two adjacent frames may often cause flickering artifact. Our caching method does not suffer from this artifact.

7 Conclusions

This paper presented an image-based hierarchical coherence structure, which enables effective sampling for indirect illumination computation. Using the set of samples, two indirect lighting algorithms are presented: the algorithm presented in Sect. 4 is suitable for a highly dynamic scene, and the caching algorithm presented in Sect. 5 significantly increases the rendering performance for a relatively static scene. Our methods also has drawbacks. It is an image-space algorithm, and works with only the scene points visible from the camera. We are currently investigating the possibility of combining our method with voxelization [10, 11].

Acknowledgements The Buddha, Bunny, and Dragon models were provided by the Stanford University Computer Graphics Laboratory. The underground scene was designed by Tony Hayes. The home scene was created by Manjeet Chakrvarty. The factory and table scene were downloaded from www.archive3d.net. This research was supported by MKE, Korea under ITRC NIPA-2010-(C1090-1021-0008) (NTIS-2010-(1415109527)) and the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No. 2009-0086684).

References

1. Dachsbacher, C., Stamminger, M.: Reflective shadow maps. In: I3D '05, pp. 203–231 (2005)
2. Dachsbacher, C., Stamminger, M.: Splatting indirect illumination. In: I3D '06, pp. 93–100 (2006)
3. Debattista, K., Dubla, P., Banterle, F., Santos, L.P., Chalmers, A.: Instant caching for interactive global illumination. Comput. Graph. Forum **28**(8), 2216–2228 (2009)
4. Gassenbauer, V., Křivánek, J., Bouatouch, K.: Spatial directional radiance caching. Comput. Graph. Forum **28**(4), 1189–1198 (2009)

5. Gautron, P., Bouatouch, K., Pattanaik, S.: Temporal radiance caching. IEEE Trans. Vis. Comput. Graph. **13**(5), 891–901 (2007)
6. Gautron, P., Křivánek, J., Bouatouch, K., Pattanaik, S.: Radiance cache splatting: a GPU-friendly global illumination algorithm. In: SIGGRAPH '05 Sketches, p. 36 (2005)
7. Herzog, R., Myszkowski, K., Seidel, H.P.: Anisotropic radiance-cache splatting for efficiently computing high-quality global illumination with lightcuts. Comput. Graph. Forum **28**(2), 259–268 (2009)
8. Keller, A.: Instant radiosity. In: SIGGRAPH '97, pp. 49–56 (1997)
9. Křivánek, J., Gautron, P., Pattanaik, S., Bouatouch, K.: Radiance caching for efficient global illumination computation. IEEE Trans. Vis. Comput. Graph. **11**(5), 550–561 (2005)
10. Laine, S., Karras, T.: Efficient sparse voxel octrees. In: I3D '10, pp. 55–63 (2010)
11. Nichols, G., Penmatsa, R., Wyman, C.: Interactive, multiresolution image-space rendering for dynamic area lighting. Comput. Graph. Forum **29**(4), 1279–1288 (2010)
12. Nichols, G., Shopf, J., Wyman, C.: Hierarchical image-space radiosity for interactive global illumination. Comput. Graph. Forum **28**(4), 1141–1149 (2009)
13. Nichols, G., Wyman, C.: Multiresolution splatting for indirect illumination. In: I3D '09, pp. 83–90 (2009)
14. Ritschel, T., Engelhardt, T., Grosch, T., Seidel, H.P., Kautz, J., Dachsbacher, C.: Micro-rendering for scalable, parallel final gathering. In: SIGGRAPH Asia '09, pp. 1–8 (2009)
15. Ritschel, T., Grosch, T., Kim, M.H., Seidel, H.P., Dachsbacher, C., Kautz, J.: Imperfect shadow maps for efficient computation of indirect illumination. ACM Trans. Graph. **27**(5), 1–8 (2008)
16. Saito, T., Takahashi, T.: Comprehensible rendering of 3-d shapes. SIGGRAPH Comput. Graph. **24**(4), 197–206 (1990)
17. Tabellion, E., Lamorlette, A.: An approximate global illumination system for computer generated films. ACM Trans. Graph. **23**(3), 469–476 (2004)
18. Tawara, T., Myszkowski, K., Seidel, H.P.: Exploiting temporal coherence in final gathering for dynamic scenes. In: CGI '04, pp. 110–119 (2004)
19. Walter, B., Fernandez, S., Arbree, A., Bala, K., Donikian, M., Greenberg, D.P.: Lightcuts: a scalable approach to illumination. ACM Trans. Graph. **24**(3), 1098–1107 (2005)
20. Ward, G.J., Rubinstein, F.M., Clear, R.D.: A ray tracing solution for diffuse interreflection. In: SIGGRAPH '88, pp. 85–92 (1988)

Kien T. Nguyen is a Ph.D. candidate at Korea University. He received his B.S. degree in Applied Mathematics from Hanoi University of Science in 1999 and M.S. degree in Computer Science from Le-Quy-Don Technical University in 2003. His research interests include real-time shadows, global illumination, and image-based rendering.

Hanyoung Jang is a Ph.D. candidate at Korea University. He received both of M.S. and B.S. degrees in the College of Information and Communications at Korea University. Since 2005, he has been working in the fields of collision detection and accessibility analysis for robotics. Currently, his primary research interest lies in real-time rendering of complex scenes.

JungHyun Han is a professor at Korea University, where he directs the Interactive 3D Media Laboratory and Game Research Center. Prior to joining Korea University, he worked at Sungkyunkwan University in Korea and at the US Department of Commerce National Institute of Standards and Technology (NIST). He received a B.S. degree in Computer Engineering at Seoul National University, an M.S. degree in Computer Science at the University of Cincinnati, and a Ph.D. degree in Computer Science at USC. His research interests include real-time rendering and animation for games.