Exploiting Temporal Coherence in Final Gathering for Dynamic Scenes

T. Tawara, K. Myszkowski, and H.-P. Seidel

MPI Informatik Saarbruecken, Germany
{tawara, karol, hpseidel}@mpi-sb.mpg.de

Abstract

Efficient global illumination computation in dynamically changing environments is an important practical problem. In high-quality animation rendering costly "final gathering" technique is commonly used. We extend this technique into temporal domain by exploiting coherence between the subsequent frames. For this purpose we store previously computed incoming radiance samples and refresh them evenly in space and time using some aging criteria. The approach is based upon a two-pass photon mapping algorithm with irradiance cache, but it can be applied also in other gathering methods. The algorithm significantly reduces the cost of expensive indirect lighting computation and suppresses temporal aliasing with respect to the state of the art frame-by-frame rendering techniques.

Keywords: Global illumination, photon mapping, temporal coherence, final gathering, irradiance cache.

Please note that this paper is accompanied by electronic materials.

1. Introduction

Producing high quality animations featuring compelling lighting effects is very time consuming using traditional rendering approaches in which each frame is computed separately. This means that a vast majority of computation must be started from scratch for each frame, leading to unnecessary redundant computation. Since temporal coherence is typically not exploited (some algorithms are based on pixel reprojection from frame to frame [16]), temporal aliasing problems are also more difficult to combat. Many small errors in lighting distribution cannot be perceived by the human observer when they are coherent in the temporal domain, but when such a coherence is lost, the result will often be unpleasant flickering effects. An efficient solution to reduce flickering for ray tracing was proposed by Martin [11]. In this paper we are concerned about the reduction of temporal aliasing in the lighting function.

In the rendering of production quality animation, global illumination computations[†] are usually performed using two-pass methods. In the first (preprocessing) pass, the lighting distribution over scene surfaces is sparsely computed using radiosity [10, 15, 4] or photon mapping [8, 3] methods. In the second (rendering) pass more exact global illumination computation is performed on a per pixel basis using the results obtained in the first pass. To improve the spatial resolution of lighting details for a given camera view the so-called *final gathering* [12, 10, 15, 4] is commonly used. Usually the direct lighting is explicitly computed for each pixel, and the indirect lighting is obtained through the integration of incoming radiances, which is computationally expensive.

More efficient versions of this final gathering recently have been proposed specifically for Hierarchical Radiosity with Clustering [13, 14]. The final gathering costs also can be reduced by using the *irradiance cache* data structure [19, 20] which is more suitable for ray tracing methods. Within this method irradiance samples are lazily computed and sparsely cached in object space for a given camera position (a view-dependent process). The indirect illumination is interpolated for each pixel based on those cached irradiance values, which is significantly faster than the final gathering computation for each pixel. The irradiance cache technique efficiently removes shading artifacts which are very difficult to avoid if the indirect lighting is directly reconstructed based on the radiosity mesh or the photon maps. However, the price to be paid for this high quality lighting reconstruction is long computation times, which are mostly caused by

† For a more complete overview of global illumination techniques developed specifically for animation rendering refer to a recent survey on this topic [5].

the irradiance integration that is performed for each cache location in the scene.

The irradiance caches can be reused for the efficient rendering of walkthrough animations in static environments [19]. However, for dynamic environments such a simple reusing may lead to invalid lighting. In this paper we address this problem and we propose some extensions of irradiance cache management into the temporal domain, specifically in the context of a photon mapping technique. Whenever possible we try to re-use not only the irradiance cache locations, but also we update the cache values required in dynamic environments. For each frame and each cache location, a certain percentage of stored incoming radiance samples is updated, potentially for those scene regions in which lighting changes are the most significant. Also, in response to changing lighting and camera positions, new cache locations are lazily inserted and unnecessary cache locations are removed. This results in a significant reduction in computation times compared to frame-by-frame rendering. Also, an even better animation quality can be obtained due to the temporal coherence between cache locations and cached incoming radiance samples.

It should be pointed out that our space-time approach presented in this paper focuses only upon efficient computation of soft indirect lighting. Direct illumination and caustics as well as all directional effects such as specular reflections and refractions are computed from scratch for each frame.

2. Photon Mapping

In the newly developed temporally coherent animation rendering we use the photon mapping algorithm proposed by Jensen [7, 8]. The method works well even for complex environments, and can easily simulate all possible light paths. In particular, the method outperforms all other techniques in high quality rendering of caustic effects. The method consists of two passes: (1) lighting simulation through photon tracing and (2) rendering with the re-computation of direct lighting and view-dependent specular effects.

In the first stage of this method, photons are traced from light sources toward the scene and photon hit points are registered in a kd-tree structure, the so called photon map. This stage is so fast (when compared to the second pass) that photons can be easily recomputed for each frame. To our best knowledge the only attempt at reusing photons for several frames was done to perform motion blur [1] for caustics. In our approach the photon map is computed from scratch for each frame, which guarantees that selectively updated incoming radiance samples in the final gathering computation are correct.

Soft indirect lighting is usually reconstructed using the irradiance cache technique [19, 20, 8]. For each cache location, irradiance is integrated over the scene by sampling incoming energy for selected directions as illustrated in Figure

Symbol	Quantity	Unit
E	Irradiance	Wm^{-2}
Li	Incident radiance	$Wm^{-2}sr^{-1}$

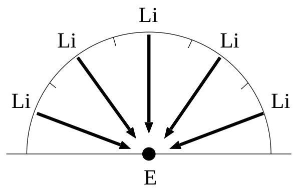


Figure 1: *Irradiance E and incoming radiance samples Li (one sample per strata) at an irradiance cache location marked by the black dot.*

2. To reduce the variance of such sampling, the hemisphere of all possible directions is split into strata and a small number of sample directions (usually one) are randomly chosen for each stratum. Each sample involves a costly intersection computation performed by tracing a ray and the estimate of energy incoming from the point hit by the ray. The photon map is used for the incoming lighting reconstruction using the nearest neighbor density estimation method [6]. To reduce the density estimation cost Christensen [3] proposed to precompute irradiance and store the resulting values at sparsely selected photon locations on diffuse surfaces. When the final gather ray hits some object, a precomputed irradiance value for the nearest photon location is simply used.

In the following section we extend the concept of irradiance cache into the temporal domain, in which case the cached irradiance value may change from frame to frame due to dynamic changes in the scene.

3. Temporally Coherent Gathering

In this section we describe our approach to updating irradiance cache values for the subsequent animation frames. Our goal is to exploit temporal coherence of incoming radiance samples contributing to each cache value. This requires sharing information on the samples between neighboring frames and selectively replacing those samples that become invalid due to changes in the dynamic environment.

In Section 3.1 we describe data structures used for the incoming radiance samples. Then in Section 3.2 we present our strategies to update those samples selectively. In Section 3.3 we discuss the problem of choosing the ratio of samples to be updated and we propose an algorithm for adaptive selection of such a ratio for each cache.

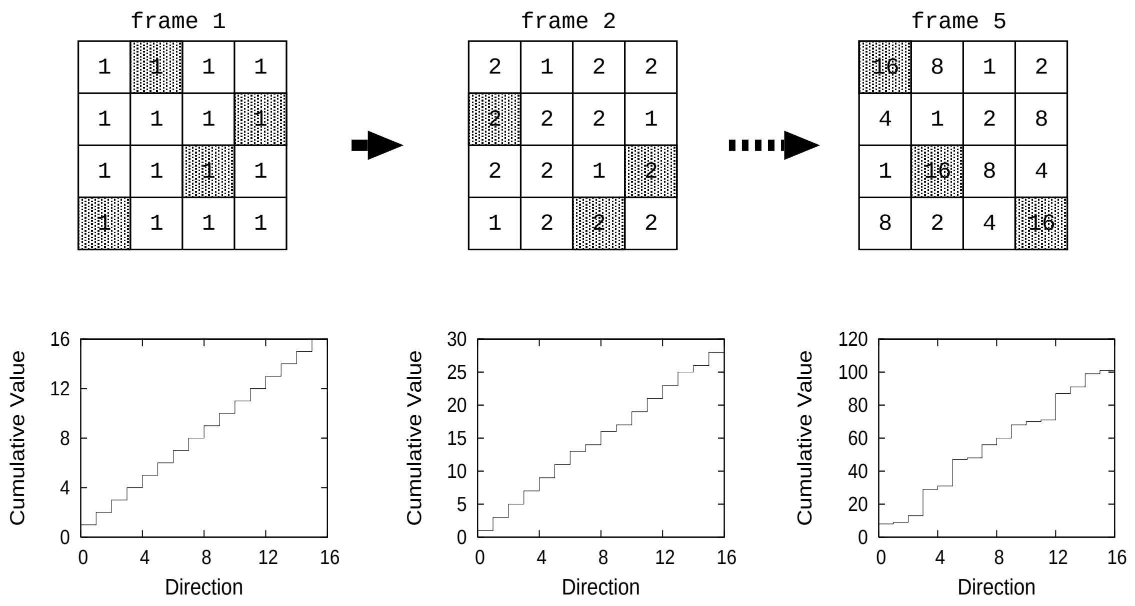


Figure 2: *Sampling scheme for the irradiance cache. Upper row: Internal cache structure with 16 stratified sampling directions in the upper hemisphere for frames 1, 2, and 5. The values in the grid cells are computed as* 2^{age}. *Lower row: The corresponding cumulative distribution functions (c.d.f.).*

3.1. Cache Data Structures

In the traditional irradiance cache algorithm [19, 20] positioning each cache is a view-dependent process. Then the numerical integration of incoming radiance is performed by tracing rays towards the environment and gathering lighting information. To produce high quality images a huge number of rays is traced, and the incoming radiance samples are immediately discarded after each cache value is computed.

In this research we show that storing incoming radiance samples is not only feasible, but also offers many advantages for efficient animation rendering. For each sampling direction, we store incoming radiance packed in four bytes using the common exponent method [18] which reduces the storage to one third of that when using the standard floating-point format. The error of the irradiance estimation caused by the inaccuracy of this format is negligible because an irradiance value is the sum of a large number of incoming radiance samples. The distance to the nearest intersection point is stored to reevaluate the harmonic mean distance [19] (the reciprocal of the sum of reciprocal distances) in upper hemispherical directions, which is required to compute optimal cache locations. The age of each sample, which is a function of the number of frames since the sample was computed, is stored in a flag. This value is used to create an approximated probability density function which is used to decide in which order directions should be replaced (refer to Section 3.2). To save storage, we use the same flag to indicate whether the sample hits a moving object. This information is used to adaptively estimate the number of rays to be updated (refer to Section 3.3). The data structure for the incoming radiance sample is as follows:

```
struct RadianceSample {
  RGBE    Li;   // incoming radiance
                // packed in 4 bytes
  float16 Di;   // distance to the nearest
                // intersection point
                // packed in 2 bytes
  ushort  flag; // number of frames and
                // the flag of hitting on
                // dynamic objects
};
```

Each incoming radiance sample occupies 8 bytes. Usually, 200–1,000 samples per cache are required to remove artifacts for a still image and the number of irradiance values varies in each scene. We store the incoming radiance samples in a hard disk. The overhead of the disk IO is negligible because the samples are accessed only once per frame when they are updated. In Section 5 we provide information on storage requirements for our test scenes.

We use the kd-tree data structure to store cache locations in the object space.

3.2. Age Driven Cache Update

In this section we describe our solution for updating incoming radiance samples based on their age. For the purpose of illustration we show in Figure 2 our sampling scheme for a simple cache, which is composed of 16 samples. The grid depicts the stratified sampling directions over the upper hemisphere. The number in each cell shows the corresponding sample age which is measured using the exponential function 2^{age}, where the *age* value is stored in the structure `RadianceSample`. The graphs in the bottom row show the corresponding cumulative distribution functions (*c.d.f.*) of the sampling direction based on the value of 2^{age}. This value for the updated cells is reset to zero for the current frame. Because all directions are computed from scratch for *frame0* (the first frame in each animation) the age values are set to 0 for all cells. The age of all cells is increased when the subsequent frame is processed. Shaded cells in the grid in Figure 2 indicate the selected cells for which incoming radiance value as well as other records in the structure `RadianceSample` (refer to Section 3.1) are updated by shooting a new random ray within the cell for that frame.

Our purpose is to pick a number of sampling directions and replace the old samples by re-shooting the ray for each selected direction. The random permutation algorithm that randomly changes the order of elements is well suited to our goals because we want to randomly pick directions with the constraint that they should not be the same for each frame. When we have m elements $v_0, v_1, \ldots, v_{m-1}$, a random integer value X in $0 \leq X \leq m-1$ is chosen and the last element v_{m-1} is swapped with v_X. This process is repeated for the remaining elements $v_0, v_1, \ldots, v_{m-2}$ until the size of the remaining elements becomes one. Figure 3 shows the pseudo-code of the random permutation algorithm.

```
randomperm ()
  elements v[size]
  m = size
  while (m > 1)
    X = uniform random integer number
        in the range of 0 and m-1
    swap v[m-1] and v[X]
    m--
```

Figure 3: *Pseudo-code of the random permutation algorithm*

It is straightforward to apply this algorithm to the elements which have non-uniform probability. Each time a random cell is selected, the *c.d.f.* is rebuilt and the uniform random value X in $0 \leq X \leq total\_cumulative\_value$ is mapped to the cell in the grid.

Figure 4 illustrates how the random value is mapped to the index of a cell for frame 5 in Figure 2. Because the *c.d.f.* is already sorted, the binary search can be efficiently exploited for this complete balanced data. After one cell is selected, this cell is excluded from the *c.d.f.* and a new *c.d.f.* is built for the remaining cells as shown in the bold dashed line in Figure 4. This process is repeated until a sufficient number

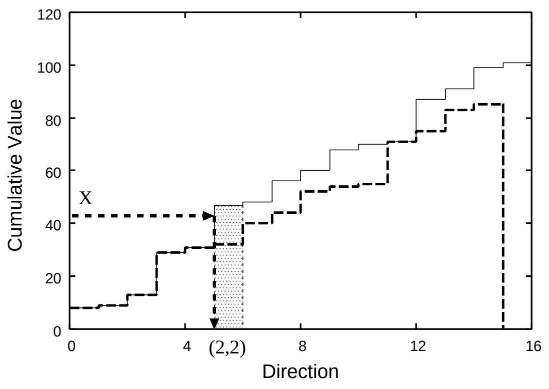


Figure 4: *A random number X is mapped to the index of a cell (2, 2) and a new c.d.f. (the bold dashed line) is rebuilt after the selected cell (the shaded area) is removed.*

of directions are chosen. The pseudo-code in Figure 5 summarizes the overall algorithm.

This scheme makes sample directions uniformly distributed in space and time. The recently selected cells are less likely to be selected than others in the subsequent frames.

```
render_animation ()
  render the first frame using traditional
  irradiance cache
  for all remaining frames
    photon tracing
    update_irradiance_cache()
    render the current frame

update_irradiance_cache ()
  for every irradiance cache E
    create a cdf
    n = the number of updated samples
        (refer to Equation (2))
    update_incoming_samples(E, cdf, n)

update_incoming_samples (E, cdf, n)
  while (n > 0)
    pick a cell based on the cdf
    shoot a gathering ray to the cell
    delete the entry of the selected cell
    from the cdf and rebuild it
    n--
  evaluate irradiance values
```

Figure 5: *Pseudo-code of the overall algorithm*

3.3. Adaptive Cache Update

So far we described in which order directions should be replaced. Another question is how many rays should be replaced for each cache. Intuitively, this should depend on the magnitude of changes in lighting: More rays should be replaced when and where lighting quickly changes and less rays may be enough for slowly changing environments. We tried both uniform and adaptive number of replacement rays. The uniform number approach is very intuitive and leads to refreshing the same number of rays for every cache. For example, when we set the number of replacement rays to 10% of the total number of strata, the rendering speed is roughly ten times faster. Moreover, it ensures that every direction is likely to be refreshed after about 10 frames (the reciprocal of 10%). This means that temporal error propagation by reusing invalid samples for more than 10 frames is not very likely.

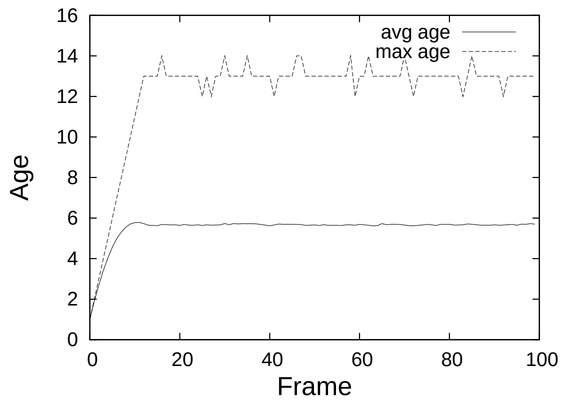


Figure 6: *The average and maximum age of samples measured for an animation composed of 100 frames. The total number of considered sample directions for each cache was 500 and 10% of samples were replaced for each frame. The average age is about 5.6 frames and the maximum age is 14 frames.*

Figure 6 shows the graph of the average and maximum age of all directions when 10% of samples is replaced. We assumed that 500 directions are stored for each cache and 50 directions are updated per frame. The average age of about 5.6 frames is obtained experimentally for an animation composed of 100 frames and it differs only slightly from the theoretical average age 5.5 frames computed as:

$$\frac{50\sum_{x=1}^{10} x}{500} = 5.5 \qquad (1)$$

The graph of the maximum age indicates that every ray is refreshed at most after 14 frames compared with the optimal 10 frames case. These experimental data show that our algorithm works well in practice.

The reason of the low average and maximum age at the leftmost part of the graph is that all directions at *frame0* are computed from scratch.

We also tried an empirical adaptive approach which adjusts the number of the replacement rays based on the number of rays which hit dynamic objects. We estimate the number of directions to be updated as:

$$N_{update}(x) = (\tau_{\max} - \tau_{\min}) \cdot x + \tau_{\min} \cdot N \qquad (2)$$

where N is the total number of gathering rays, the $\tau_{\min}$ and $\tau_{\max}$ are user specified percentages of the minimum and the maximum of updated rays and, x is the number of rays which hit on the dynamic objects. This simple empirical strategy works very well, especially in the scene regions near moving objects. Figure 7 shows an example where a red ball leaves the corner and is rolling on the floor. Figure 7a represents the reference solution of the soft indirect illumination. Figure 7b is rendered using a uniform number of refreshing rays for every cache. This solution leads to incorrect results at the corner near the red moving ball due to a too small number of refreshing rays in this region. Figure 7c shows the result of the adaptive scheme which is very similar to the reference solution.

4. Handling Irradiance Caches

In the previous section we discussed the issues of single cache update and the goal of this section is the problem of cache locations. We focus on two issues specific to our approach that arise when the irradiance cache data structures are reused between frames. In Section 4.1 we discuss how to handle caches located on moving objects. Then in Section 4.2 we present our solution to remove redundant caches as camera and lighting change.

4.1. Transforming Caches on Animating Objects

Separating the irradiance cache data structure from the geometric one is advantageous because optimal cache locations can be independently selected from the geometry. However, it is cumbersome in dynamic scenes because caches on the moving objects may no longer lay in the same position on a given surface for subsequent frames. To solve this problem, we store the object ID for each cache. This allows us to move caches to different locations in the next frame. This requires transformation of a local coordinate system for each cache to preserve the directional distribution of incoming radiance samples. Although the sampling coordinates on the moving objects are not precisely the same, artifacts were not visible in our test scenes if a reasonable number of refreshing rays is chosen (e.g., 10% of total gathering rays) even for the scene in which the ball is rolling and the normals on the ball change quickly (refer to Figure 7). For fast moving objects a complete update of all incoming radiance samples can be also considered.

a) b) c)

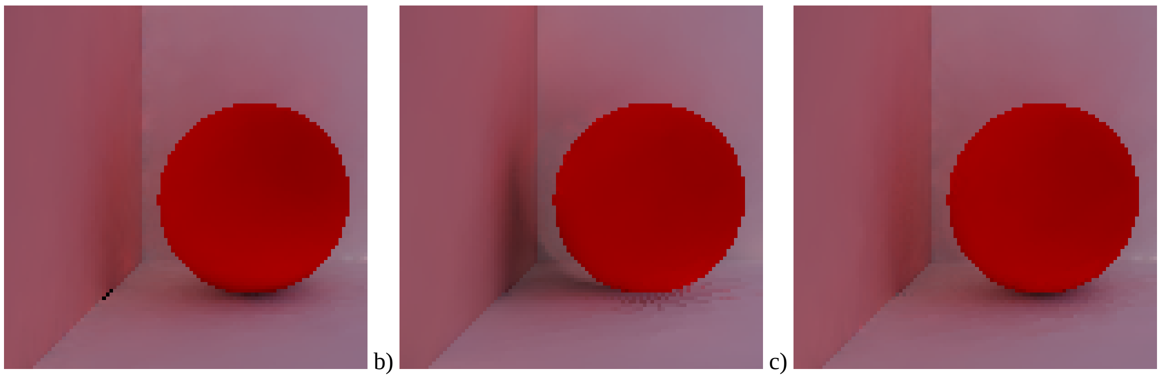


Figure 7: *Adaptive cache update for scene regions in which indirect illumination changes rapidly: a) reference frame-by-frame solution, and temporal update solutions for b) uniform and c) adaptive number of refreshing rays. Notice the incorrect shadow which is cast by the ball in the corner in b). This shadow mostly disappears in c).*

4.2. Removing Redundant Caches

When objects are moving, the optimal distribution of cache locations is changing. If an object approaches another surface the value of a harmonic mean distance becomes small and the valid domain of the irradiance cache is decreased, which leads to denser cache locations in such regions. Because irradiance cache algorithm has a unique lazy construction mechanism, new caches are simply inserted when valid caches are not found near the query location. The problem occurs when the object moves away from a surface. In this case, the harmonic mean distance of each cache becomes large and the valid cache domain increases. In such regions, some caches become redundant. This problem is illustrated in Figure 8a where the red rolling ball leaves along its motion trajectory many redundant caches.

It is difficult to find an optimal layout of caches which completely covers all visible surfaces and leads to a minimal number of caches without affecting image quality. Our solution is inspired by neighborhood-based stratification approach [17] developed in point-based rendering to remove superfluous points. We simply remove caches when too many valid caches are found at some locations. For each cache location, a nearest neighbor search is done. If the number of found neighbors is bigger than some threshold number (for example 10), this cache is removed. Figure 8b shows the result of applying this procedure and as can be seen many redundant caches are successfully removed. Updating the data structure for each cache to be removed is not efficient, so at first we mark all redundant caches and hide them for the subsequent cache density queries. After all caches are examined, marked caches are removed and the kd-tree is balanced in a packed heap data structure.

a) b)

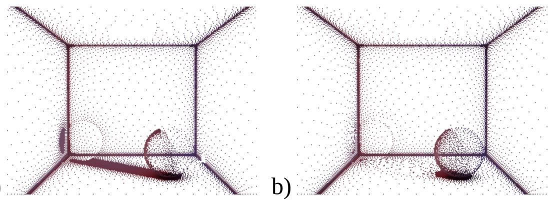


Figure 8: *Irradiance cache locations: a) redundant irradiance caches resulting from the ball motion on the floor, b) result of removing the redundant caches.*

5. Results

We tested our algorithm for three different scenes BOX, LIGHT and ROOM (refer to Figures 9, 10 and 11, respectively). For the BOX scene indirect lighting changes significantly in the proximity of regions traversed by the object. As a result of the motion of the red box towards the light source, strong color bleeding effects can be seen on the ceiling. For the LIGHT scene the light source turns toward the left red wall, which results in strong color bleeding from that wall. The ROOM scene is substantially more complex and we consider both the motion of light (sun position) and of objects (rotating fan) simultaneously. In all our test scenes indirect lighting changes quickly and those changes affect either a large portion of the scene, so these test scenes are very difficult cases for our algorithm. We expect that in many practical applications the indirect lighting changes will be more moderate.

The animation sequences were rendered by our experimental renderer on a Pentium 4 Xeon 1.7 GHz, 1 GB memory, Debian GNU/Linux. In both BOX and LIGHT scenes 192 incoming radiance samples are considered for each cache, and 768 samples are used in the ROOM scene. Incoming radiances are refreshed using the aging method. The $\tau_{\min}$ and

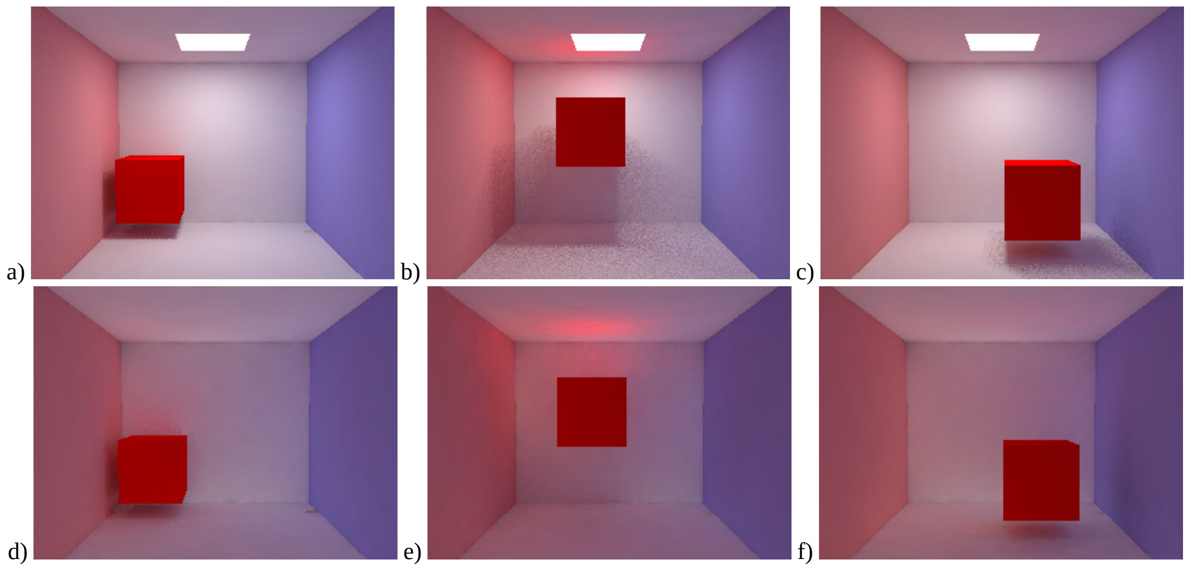


Figure 9: *Selected frames for the* BOX *animation sequence. In the upper row the final frames are shown while the bottom row shows the corresponding indirect lighting solutions computed using temporally coherent gathering.*

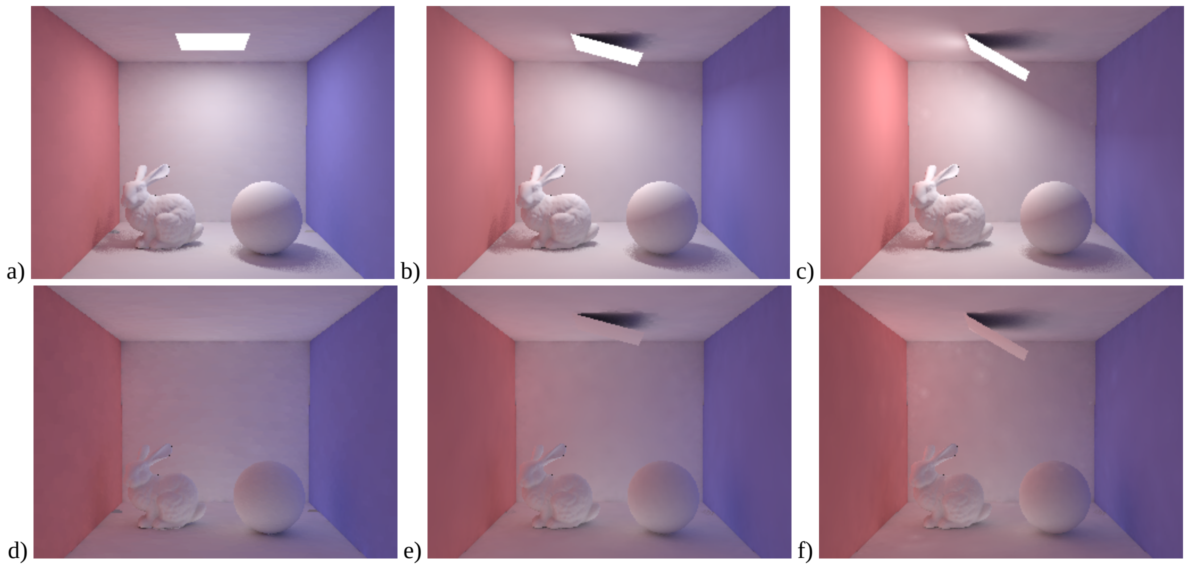


Figure 10: *Selected frames for the* LIGHT *animation sequence. In the upper row the final frames are shown while the bottom row shows the corresponding indirect lighting solutions computed using temporally coherent gathering.*

$\tau_{\max}$ parameters for adaptive control in Equation (2) are set to 0.05 and 1.0, respectively. The image resolution for both animations BOX and LIGHT is 320×240 and the one for ROOM is 564×240.

Detailed timings are shown in Table 1. The photon tracing and the precomputation of irradiance [2] (refer to the timings in column T_{pt}) are repeated for each frame. The precomputation of irradiance is time consuming (about 40–60 % of T_{pt}), but it vastly accelerates the computation of the indirect illumination in both the reference solution and ours. So we used this technique to render all animations. Column T_d shows timings for the direct lighting computation which is repeated for each frame from scratch. T_{pt} and T_d are usually a small fraction of the total computation time for complex scenes. We did not optimize our experimental code to speed up those computations. Column T_i shows timings for indi-

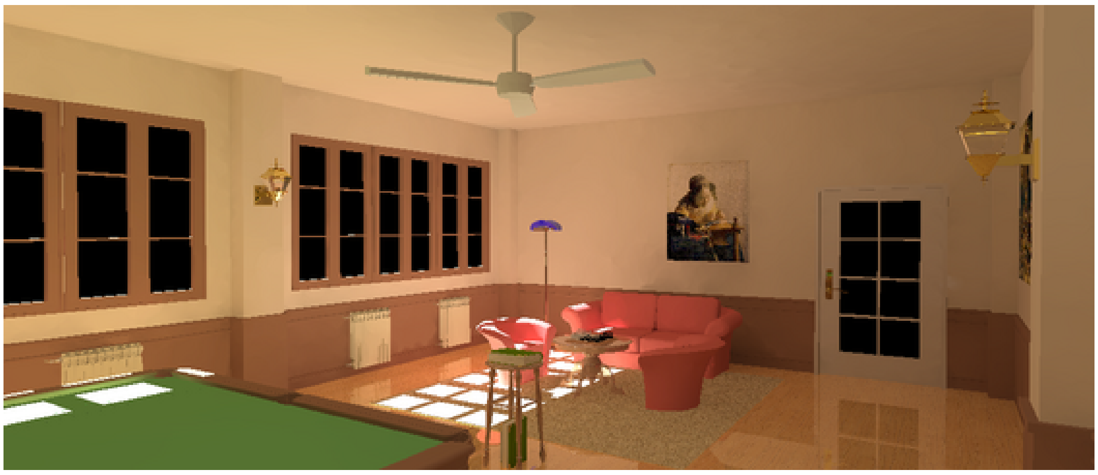
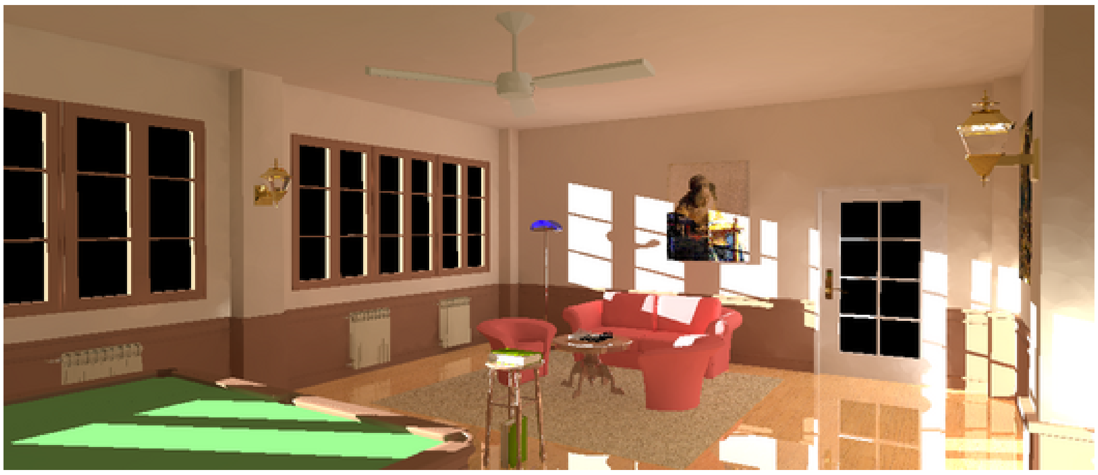
a)

b)


Figure 11: *Selected frames for the* ROOM *animation sequence.*

rect lighting reconstruction including the overhead of temporal processing. Significant speedup 3–9 times with respect to the reference frame-by-frame solution was achieved. Column T summarizes the overall processing time per frame. Note that our algorithm performs much better for the more complex ROOM scene.

The number of irradiance samples and the resulting storage requirements are shown in Table 2. Since the irradiance cache data are stored in the object space the storage requirements weakly depend on the frame resolution. For example, in the ROOM scene the number of irradiances becomes 1.7 times bigger size when the image resolution is quadrupled.

The visual quality of an animation produced by our technique is better than for the reference solution (obtained with traditional irradiance cache) because temporal flickering is significantly reduced. When single frames are compared they look almost perfectly the same as the reference solution (the RMS errors are 0.6%, 0.5% and 2.7% for the BOX, LIGHT and ROOM scenes respectively). All frames are of similar quality because our algorithm does not accumulate error and refreshes all gathering rays. Table 3 summarizes changes of the RMS error as a function of $\tau_{\min}$ for the BOX scene. Similar results have been also obtained for the other test scenes.

Figure 12 depicts the values of incident radiance samples over the hemisphere for a selected cache location. The samples are captured in the middle of an animation sequence in order to check whether errors in their value do not accumulate as a function of time. As can be seen the directional distribution of samples is very similar for the frame-by-frame computation and our method. The graphs in Figure 13 show changes of the irradiance value as a function of time for both approaches. The irradiance is measured for 90 subsequent frames. The same cache location as in Figure 12 is considered. Again, the correspondence between the two graphs is good. A small time lag between the two graphs can be observed because samples are reused only in the "chronological" order.

Scene		T_{pt}	T_d	T_i		T		total
BOX	Ref	21.6	1.0	19.3		41.9		1 h 3 min
	Our	21.6	1.0	6.7	(x2.9)	29.3	(x1.4)	45 min
LIGHT	Ref	24.3	1.1	34.3		59.8		1 h 31 min
	Our	24.4	1.1	6.6	(x5.2)	32.2	(x1.9)	49 min
ROOM	Ref	34.6	11.6	674.6		720.9		18 h 14 min
	Our	34.6	11.6	74.3	(x9.1)	120.8	(x6.0)	3 h 14 min

Table 1: *Timings for scenes* BOX, LIGHT *and* ROOM *shown in Figures 9, 10 and 11, respectively. All timings except the last column are given in seconds. The row "Ref" presents timings of the reference solution, in which all frames are computed independently. The row "Our" presents timings of our solution. The columns show the average time per frame for each component: T_{pt} - photon tracing and precomputation of irradiance [2], T_d - direct illumination, T_i - indirect illumination. The column T is the average time per frame for all components, i.e. $T = T_{pt} + T_d + T_i$. The values in the parentheses show the scale factors of speeding up compared to the corresponding reference solution. The last column shows the total timings for rendering all frames.*

Scene	N	#E	Size
BOX	192	4,009	6.2 MB
LIGHT	192	4,377	6.7 MB
ROOM	768	11,599	71.3 MB

Table 2: *The size of the irradiance cache: N - the number of incoming radiance samples per an irradiance E (refer to Equation 2), #E - the number of irradiance values, Size - the storage requirement of incoming radiance samples.*

$\tau_{\min}$	RMS Error	T_i
10 %	0.5 %	7.3
5 %	0.6 %	6.7
2.5 %	0.8 %	6.1
1.25 %	1.1 %	5.9

Table 3: *The RMS Error measured in respect to the reference frame-by-frame animation for the* BOX *scene for various settings of $\tau_{\min}$ which decides upon the minimal number of updated rays (refer to Equation 2). $\tau_{\max} = 100\%$ was assumed. As in Table 1 T_i denotes the average time per frame (measured in seconds) for the indirect illumination computation during rendering.*

6. Conclusions

We presented a simple, general, and effective method for computing soft indirect illumination in dynamic scenes. Our algorithm requires to build a cumulative distribution function (*c.d.f.*) for each irradiance cache to decide the order of directional incoming radiance samples updates based on their age. The gathering rays are updated taking into account the probability in the *c.d.f.* within the limits of user-set minimum and maximum refreshing sampling ratios ($\tau_{\min}$ and $\tau_{\max}$). By contrast to traditional approaches which compute global illumination for every frame from scratch, our algorithm updates a rather small number of incoming radiances. It speeds up expensive indirect illumination computation 3–9 times compared to the currently fastest rendering algorithm. Also, temporal flickering of indirect lighting component is substantially reduced due to the use of temporal coherence. This is achieved by sharing the same cache locations and incoming radiance samples between subsequent frames. Our algorithm can handle general animations including light source motion. Camera animation does not affect cached indirect illumination and new irradiance caches are inserted only when occluded parts become visible for subsequent frames. Our algorithm fits well to the photon mapping algorithm [8] in which direct illumination, caustics, specular reflections and refractions are computed on a frame-by-frame basis while diffuse interreflections can be accelerated using our technique.

As future work we plan to experiment with more compact representations for incoming radiance using wavelets or spherical harmonics. Also, we plan to investigate the applicability of our technique to render moderately glossy surfaces using directional information which we store in our cache data structure. This will require the development of new criteria controlling the density of cache locations, which will be sensitive to surface glossiness. To reduce the number of caches required for high quality rendering of moderately glossy surfaces we plan to use *the final gather reprojection* technique [9].

References

1. M. Cammarano and H.W. Jensen. Time dependent photon mapping. In *Proceedings of the 13th Eurographics*

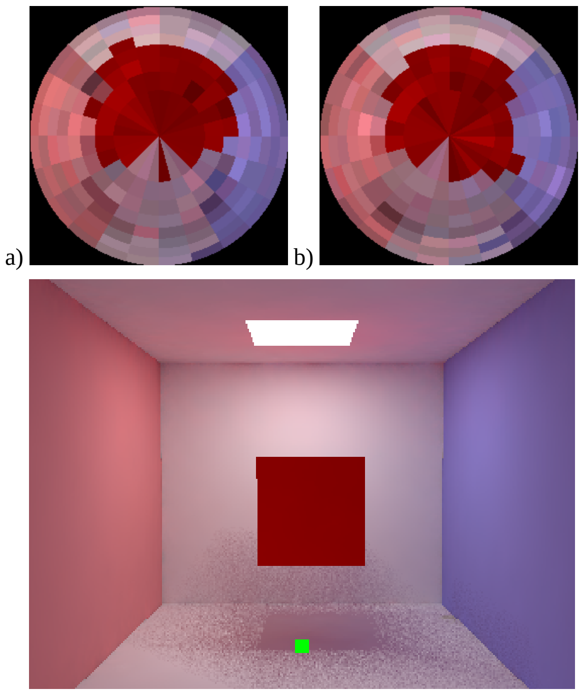


Figure 12: *Distribution of incoming radiance samples over the hemisphere for a selected cache location at the floor which is shown as the green dot in the bottom image: a) the frame-by-frame computation and b) our method (10% of samples is refreshed for each frame according to the aging criterion).*

Workshop on Rendering, pages 135–144, 2002. 2

2. P.H. Christensen. Faster photon map global illumination. In *Journal of Graphics Tools*, volume 4, pages 1–10, 1999. 7, 9

3. P.H. Christensen. Photon mapping tricks. In *Siggraph 2002, Course Notes No. 43, A Practical Guide to Global Illumination using Photon Mapping organized by Jensen, H.W.*, pages 93–121, 2002. 1, 2

4. P.H. Christensen, D. Lischinski, E.J. Stollnitz, and D.H. Salesin. Clustering for glossy global illumination. *ACM Transactions on Graphics*, 16(1):3–33, 1997. 1

5. C. Damez, K. Dmitriev, and K. Myszkowski. State of the art in global illumination for interactive applications and high-quality animations. *Computer Graphics Forum*, 22(1):55–78, 2003. 1

6. P. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. In *Computer Graphics (ACM SIGGRAPH '90 Proceedings)*, pages 145–154, August 1990. 2

7. H.W. Jensen. Global illumination using photon maps.

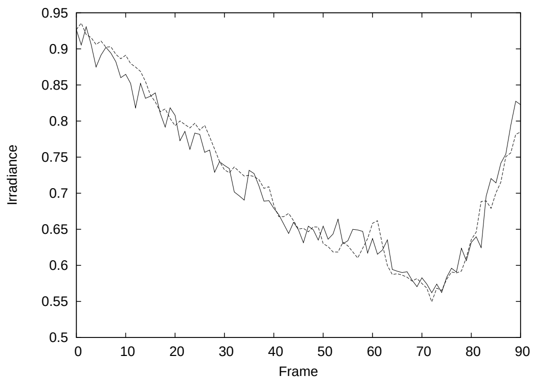


Figure 13: *Changes of irradiance value as a function of time for the* BOX *animation at the cache location shown in Figure 12. The graph for the frame-by-frame approach is drawn as the solid line, while the dashed line is used for our approach.*

In *Eurographics Workshop on Rendering*, pages 21–30, 1996. 2

8. H.W. Jensen. *Realistic Image Synthesis Using Photon Mapping*. A. K. Peters, Natick, MA, 2001. 1, 2, 9

9. T. Kato. Photon mapping in kilauea. In *Siggraph 2002, Course Notes No. 43, A Practical Guide to Global Illumination using Photon Mapping organized by Jensen, H.W.*, pages 122–191, 2002. 9

10. D. Lischinski, F. Tampieri, and D. P. Greenberg. Combining hierarchical radiosity and discontinuity meshing. In *Proc. of Siggraph'93*, pages 199–208, 1993. 1

11. William Martin, Erik Reinhard, Peter Shirley, Steven Parker, and William Thompson. Temporally coherent interactive ray tracing. *Journal of Graphics Tools*, 2002. 1

12. M.C. Reichert. *A Two-Pass Radiosity Method to Transmitting and Specularly Reflecting Surfaces*. M.Sc. thesis, Cornell University, 1992. 1

13. A. Scheel, M. Stamminger, and H.-P. Seidel. Thrifty final gather for radiosity. In *Rendering Techniques 2001: 12th Eurographics Workshop on Rendering*, pages 1–12. Eurographics, June 2001. 1

14. A. Scheel, M. Stamminger, and H.-P. Seidel. Grid based final gather for radiosity on complex clustered scenes. In *Eurographics*, 2002. 1

15. B. Smits. *Efficient Hierarchical Radiosity in Complex Environments*. Ph.D. thesis, Cornell University, 1994. 1

16. Bruce Walter, George Drettakis, and Steven Parker. Interactive rendering using the render cache. In *Eurographics Rendering Workshop 1999*, June 1999. 1

17. M. Wand and W. Straßer. Multi-resolution rendering of complex animated scenes. volume 21, pages 483–491, 2002. 6

18. G. Ward. Real pixels. In *Graphics Gems II*, pages 80–83, 1991. 3

19. G. Ward, Rubinstein F., and R. Clear. A ray tracing solution to diffuse interreflection. In *Computer Graphics (SIGGRAPH 88 Conference Proceedings)*, pages 85–92, 1988. 1, 2, 3

20. G. Ward and P. Heckbert. Irradiance gradients. In *Eurographics Workshop on Rendering*, pages 85–98, 1992. 1, 2, 3