Reusing frames in camera animation

Àlex Méndez-Feliu
Universitat de Girona
Institut d'Informàtica i Aplicacions
Edifici P4, Campus de Montilivi
17071, Girona, Spain
amendez@ima.udg.es

Mateu Sbert
Universitat de Girona
Institut d'Informàtica i Aplicacions
Edifici P4, Campus de Montilivi
17071, Girona, Spain
mateu@ima.udg.es

László Szirmay-Kalos
Technical University of Budapest
Informatics Building, B320
1117 Pázmány P. sétány 1/D.
Budapest, Hungary
szirmay@iit.bme.hu

ABSTRACT

Rendering an animation in a global illumination framework is a very costly process. Each frame has to be computed with high accuracy to avoid both noise in a single frame and flickering from frame to frame. Recently an efficient solution has been presented for camera animation, which reused the results computed in a frame for other frames via reprojection of the first hits of primary rays. This solution, however, is biased since it does not take into account the different probability densities that generated the different contributions to a pixel. In this paper we present a correct, unbiased solution for frame reuse. We show how the different contributions can be combined into an unbiased solution using multiple importance sampling. The validity of our solution is tested with an animation using path-tracing technique, and the results are compared with both the classic independent approach and the previous unweighted, biased, solution.

Keywords: Animation, Ray tracing, Path reuse, Global illumination, Path tracing

1 INTRODUCTION

In global illumination an image can be computed by tracing paths from the eye (or observer position) trough the pixels that compose the image plane towards the surfaces of the scene. In the path-tracing technique, from the hit point in the scene a random walk is followed, gathering the energy at every new hit point. The main drawback of these Monte Carlo random walks is the high number of paths needed to obtain an acceptable result. To obtain an animation or a sequence of frames in a global illumination framework with production quality, we need to cast many rays per pixel. Each frame has to have high accuracy to avoid both noise in the frame and flickering from frame to frame. An efficient solution to reduce this cost has been presented for camera animation [9]. Computation for one frame is reused for other frames via reprojection of the first hit of the eye ray to neighbour eyes positions. Although very computationally efficient, this solution is biased, as it does not take into account the different probability densities that generated the different contributions to a pixel. In this paper we improve on this solution by presenting an unbiased method for frame reuse. Our approach is a follow-up of previous research on path reuse [2, 17, 16]. We show how the different contributions can be combined to an unbiased solution using multiple importance sampling [20]. We test our solution with an animation using the path-tracing algorithm for global illumination, and compare it with a classic independent solution and the previous unweighted, biased, technique. Although the results are shown in this paper for the path-tracing algorithm, the validity of our technique is general.

This paper is organized as follows. In the next section we will refer to previous work on reusing frames in animation and on reusing paths in the context of radiosity and global illumination. Path-tracing is also shortly revisited. The theoretical foundation and algorithms of our techniques for reusing frames are presented in section 3. In section 4 details about the implementation and costs of our algorithm are explained. In section 5 results are presented that illustrate the benefits of our technique, and in the last section we present the conclusions and future work.

2 PREVIOUS WORK

2.1 Path-tracing algorithm

Random walks are a Monte Carlo common tool to solve second kind Fredholm integral equation [15, 7, 12]. For instance, they are used in global illumination and radiosity [1, 5] to solve the *rendering equation* [11]. Path-tracing [11], distributed ray-tracing [4], bidirectional path-tracing [13, 20], photon map [10] and Metropolis [22] are main global illumination techniques using random walk. In the path-tracing technique (see Fig.1a) an image is computed by tracing paths from the eye (or observer position) trough the pixels that compose the image plane towards the surfaces of the scene. To avoid a huge variance, from each hit point ray(s) are directed towards the light sources to gather light. The random walk can be terminated upon different termination conditions. Russian roulette is the most used termination technique. At a hit point, we decide at random whether to terminate or follow the path. Another possibility is to accumulate the albedos of visited hit points and stop when the accumulated albedo is below a given threshold. Path-tracing is a general, unbiased simple technique to compute global illumination, relatively easily to implement and very much appropriate to test improvements in global illumination, although more sofisticated techniques have been introduced like Metropolis and bidirectional path-tracing. On the other hand, shooting random walk solves the adjoint equation by simulating the trajectories of photons from the light sources, and hybrid methods like bidirectional path-tracing combine both shooting and gathering. The cost of a random walk simulation is mainly the cost of computing the next hit point, that is, the point visible from the old hit point in the sampled direction.

The main drawback of these Monte Carlo random walks is however the high number of paths needed to obtain an acceptable result. As the variance of the estimators is proportional to N^{-1} with N the number of paths, it makes necessary the use of many paths, of the order of millions, to obtain an acceptable noiseless image. This is still more dramatic in an animation computation, due to the high number of frames to be computed. Thus achieving some sort of path reusing can reduce the computational cost.

2.2 Reusing paths

Bidirectional path-tracing [13, 20] can be considered as the first attempt to reduce the cost by reusing paths. This method joins sub-paths belonging to the same pixel or to the same source point. However, the idea of the reuse of full paths and for different states (i.e., pixels, patches or light sources) was first presented by John Halton in [6] in the context of the random walk solution of an equation system. This technique was applied by Bekaert et al. in the context of path-tracing in [2, 16] (see Fig.1a), combined with multiple importance sampling [21, 20] to avoid bias. Pixels were grouped in tiles, and paths belonging to a pixel in the tile were reused for the other pixels in the tile from the second hit point of the path. A speed-up of one order of magnitude was reported for fairly complex scenes. Havran et al. [9] presented the reuse of paths in a walk-through, that is, when the observer changes position. Paths cast from one observer position were reused for other neighbor positions. Their technique admitted motion blur and they applied it in the context of bidirectional path-tracing. Although obtaining a high speed-up, the method remained biased as the samples were not weighted with the respective probability. In the radiosity context Besuievsky [3] used the same set of lines to expand direct illumination from different light source positions. The source positions were packed in a bounding box and lines crossing this box expanded the power of all intersected positions. The drawback of this method is that lines are wasted if the source positions are not tightly packed. Moreover, this method is only valid for diffuse sources. Recently path-reuse has been applied to light source animation in [17, 18, 14].

3 FRAME REUSE

In this section the theoretical framework of our algorithm is introduced. We first introduce the basic *native* estimators, then we show how we can estimate the radiance from a different eye and finally how the radiance estimators obtained with paths from different eyes can be combined by multiple importance sampling.

3.1 Estimating the *native* radiance

In global illumination we are interested in the integrals $L^o(i) = \int_{A_i} L(p)dp$ where A_i is the area of pixel i, and $L(p)$ is the radiance from visible scene point x that reaches the observer at point o through point p in pixel i . Introducing a change of integration variable [19] we obtain

$$L^o(i) = \int_S L(x \to o) G(o,x) V_i(x) \frac{\cos^3 \theta_i}{f^2} dx, \quad (1)$$

where the integration extends over all scene surface points S, θ_i is the angle between the normal of the screen plane and direction $\omega(x \to o)$ at the center of the pixel i, and f is the focal distance, i.e. the distance from o to the plane of the screen. $V_i(x)$ takes the value of 1 if x is visible through the pixel i and 0 otherwise. The geometric factor $G(o,x)$ is defined as

$$G(o,x) = vis(o,x) \frac{\cos(N_x, \omega(x \to o))}{d^2(x,o)} \quad (2)$$

where $vis(o,x)$ is 1 if x and o see each other and 0 otherwise, N_x is the normal at point x, $\omega(x \to o)$ is the direction from x to o, and $d(x,o)$ is their distance. The radiance $L(x \to o)$ comes from the global illumination equation [11]:

$$\begin{gathered} L(x \to o) = L^e(x \to o) + \\ \int_\Omega \rho(\omega^{in}, x, x \to o) L(x, \omega^{in}) \cos(N_x, \omega^{in}) d\omega^{in} \end{gathered} \quad (3)$$

where $L^e(x \to o)$ is the self emitted radiance, $\rho(\omega^{in}, x, x \to o)$ is the bidirectional reflectance distribution (brdf) function at point x, incoming direction ω^{in} [1] and outgoing direction $x \to o$. $L(x, \omega^{in})$ is the incoming radiance to x in direction ω^{in}.

[1] In fact, the true incoming direction is $-\omega^{in}$, but we use the opposite one to keep the reciprocity in the brdf.

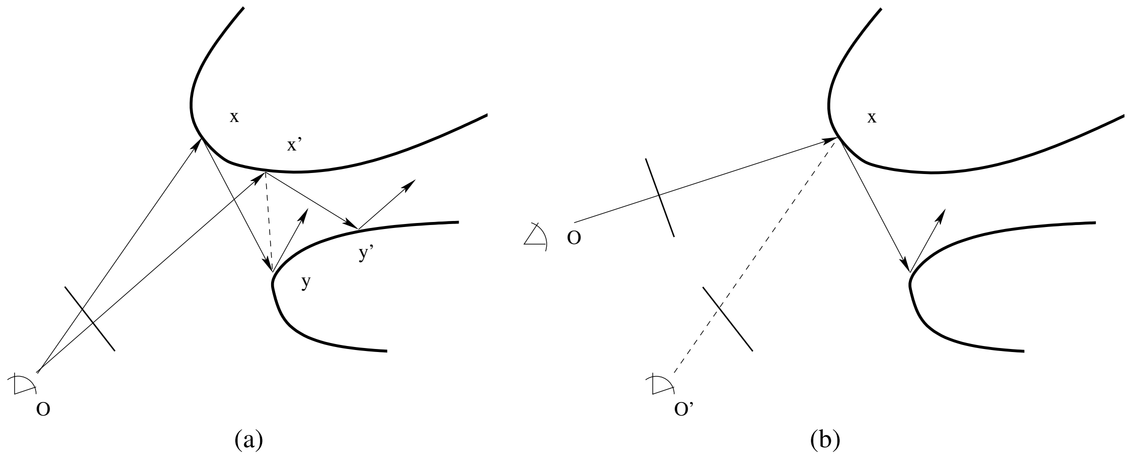


Figure 1: (a) Reusing a path from the second hit point, y, for a single observer O, creating thus the new path $O,x',y,\ldots$ at the cost of the visibility test $vis(x',y)$. (b) Reusing the path from observer O for observer O', at the cost of the visibility test $vis(O',x)$.

Substituting (3) into (1), and dropping constant terms and self-emission [2], we obtain

$$L^o(i) = \int_S \int_\Omega G(o,x)V_i(x)\rho(\omega^{in},x,x \to o) \\ L(x,\omega^{in})\cos(N_x,\omega^{in})d\omega^{in}dx \qquad (4)$$

Primary estimator $\widehat{L^o(i)}$ for $L^o(i)$ is obtained by selecting a point x with probability $p_i^o(x)$ and then direction ω^{in} with probability $p(\omega^{in};x,x \to o)$. An unbiased estimator for $L(x,\omega^{in})$, $L(\widehat{x,\omega^{in}})$, can be obtained by any suitable technique, for instance by the random walk path-tracing technique. Thus

$$\widehat{L^o(i)} = \frac{G(o,x)V_i(x)\rho(\omega^{in},x,x \to o)L(\widehat{x,\omega^{in}})\cos(N_x,\omega^{in})}{p_i^o(x)p(\omega^{in};x,x \to o)} \qquad (5)$$

With importance sampling we select probabilities

$$p_i^o(x) \propto G(o,x)V_i(x)$$

and

$$p(\omega^{in};x,x \to o) \propto \rho(\omega^{in},x,x \to o)\cos(N_x,\omega^{in})$$

Inthis case the estimator becomes:

$$\widehat{L^o(i)} = a(x,x \to o)\Omega_i L(\widehat{x,\omega^{in}}) \qquad (6)$$

where Ω_i (solid angle subtended by pixel i) and $a(x,x \to o)$ (albedo) are the probabilities normalization constants. Estimator (5) is the unbiased *native* estimator for a pixel, that is, the one obtained by sending rays from the observer through the pixel.

[2] Self emission can be easily dealt with separately.

3.2 Estimating the radiance from a different eye

Consider now a different observer o' (see Fig.1b). This observer will see x through a different pixel, j. We can obtain a (biased) estimator for $L^{o'}(j)$ reusing the value obtained for the radiance at x with estimator $L(\widehat{x,\omega^{in}})$ (this comes to reusing the path from x supposing the estimator is a random walk, see Fig. 1b). The estimator for pixel j from eye o' obtained with a ray started from eye o is given by the following expression:

$$\widehat{L^{o',i}(j)} = \frac{G(o',x)V_j(x)\rho(\omega^{in},x,x \to o')L(\widehat{x,\omega^{in}})\cos(N_x,\omega^{in})}{p_i^o(x)p(\omega^{in};x,x \to o)} \qquad (7)$$

Note that the probabilities used in the denominator are the native probabilities used to find point x from eye o through pixel i, but they should be normalized with respect to pixel j as seen from eye o'. We have then to normalize $p_i^o(x)$ with respect to the new eye. Let us drop the assumption that probability $p_i^o(x)$ depends on pixel i as seen from o and through which the ray was generated (this is an approximation, considering pixels on a spherical screen). The corresponding normalization condition should be fulfilled

$$\int_S p^o(x)V_j(x)dx = 1 \qquad (8)$$

where $V_j(x) = 1$ if point x is visible from o' trough pixel j and 0 otherwise. Suppose now that we distribute eye rays from o with probability proportional to $G(o,x)$. To obtain probabilities $p^o(x)$ we have to find the normalization constant given by the integral:

$$I = \int_S G(o,x)V_j(x)dx \qquad (9)$$

Integral (9) can be interpreted as the solid angle from o that *sees* the portion of the scene seen from the solid angle subtended by pixel j from eye o' (Ω_j), see fig.

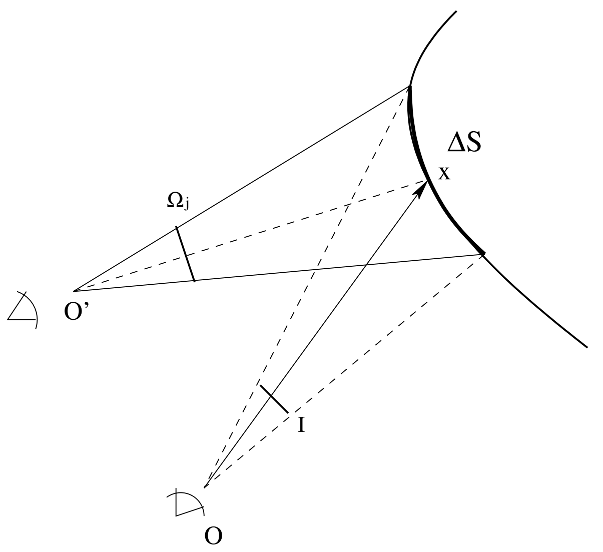


Figure 2: Here is shown in a graphical way the interpretation of equation (9). Ω_j is the solid angle subtended by pixel j, and I is the solid angle through which eye o sees what eye o' can see through Ω_j.

2. Observe that when $o = o'$ integral (9) is equal to Ω_j. Integral (9) is not known and computing it (using Monte Carlo integration) would require sending a lot of rays from o and comparing the visible or unoccluded hit points from o' to the total unoccluded+occluded. Lacking this information, we make the assumption that we have no visibility problems. Thus, given hit x from o obtained with probabilities proportional to $G(o,x)$, and taking into account that without occlusions $\Delta\Omega_j = G(o',x)\Delta S$ and $\Delta I = G(o,x)\Delta S$, the normalization constant (9) is approximated by

$$I \approx \frac{\Omega_j G(o,x)}{G(o',x)} \tag{10}$$

Expression (10) is used to normalize p_i^o and estimator (7) (having dropped the dependence on the pixel i) thus becomes

$$\begin{aligned} \widehat{L^{o'}(j)} &= \frac{G(o',x)\rho(\omega^{in},x,x\rightarrow o')L(\widehat{x,\omega^{in}})\cos(N_x,\omega^{in})}{\frac{G(o',x)}{\Omega_j G(o,x)}G(o,x)p(\omega^{in};x,x\rightarrow o)} \\ &= \frac{\Omega_j\rho(\omega^{in},x,x\rightarrow o')L(\widehat{x,\omega^{in}})\cos(N_x,\omega^{in})}{p(\omega^{in};x,x\rightarrow o)} \end{aligned} \tag{11}$$

and for a diffuse hit point ρ (and thus neither p) depends on the incoming eye ray, thus it becomes simply

$$\widehat{L^{o'}(j)} = \Omega_j \widehat{L(x)} \tag{12}$$

where $\widehat{L(x)}$ is the estimated radiance from hit point x.

3.3 Combining paths

In [9] an unweighted combination of estimators of kind (5) and (7) was done, resulting in a biased estimator. We present now a strategy that gives an unbiased estimator. For each pixel and frame, we keep accumulated radiance value and native ray estimators (among many other data useful for our computation, see 4.3) generated with probability $p_i^o(x)p(\omega^{in};x,x\rightarrow o)$). When hits from neighbour frames can be reused for this pixel (suppose estimator $\widehat{L^{o,j}(i)}$, generated with probability $p_j^{o^j}(x^j)p(\omega^{in,j};x^j,x^j\rightarrow o^j)$), we combine them using multiple importance sampling with the native estimator. This gives the new unbiased estimator:

$$\widehat{L^o(i)} = \sum_j \frac{p_j^{o^j}(x^j)p(\omega^{in,j};x^j,x^j\rightarrow o^j)\widehat{L^{o,j}(i)}}{\sum_k p_k^{o^k}(x^j)p(\omega^{in,j};x^j,x^j\rightarrow o^k)} \tag{13}$$

We show now how this estimator is applied.

For the sake of simplicity, and without loss of generality, consider only two observers O_1 and O_2[3]. In this case estimator (13) becomes estimator (14) for observer O_1, using the approximation (12) for the estimator and also (9) for the normalization constant in the weights. We have taken the approximation that all pixels subtend the same solid angle. Remember also that the visibility boolean function is included in the G function. Consider first the particular case for diffuse hit pixels.

$$\begin{aligned} L(O_1) &= \frac{G(O_1,x_1)}{G(O_1,x_1)+G(O_2,x_1)\frac{G(O_1,x_1)}{G(O_2,x_1)}}L(x_1) \\ &+ \frac{G(O_2,x_2)}{G(O_1,x_2)\frac{G(O_2,x_2)}{G(O_1,x_2)}+G(O_2,x_2)}L(x_2) \\ &= \tfrac{1}{2}L(x_1)+\tfrac{1}{2}L(x_2) \end{aligned} \tag{14}$$

Thus approximation (10) for the normalization constant leads to the simple unweighted estimator when we deal only with diffuse hits, and this is why Havran et al. solution [9] works well for diffuse surfaces.

For the non-diffuse general case, using again approximation (10) allows us to eliminate all G terms, and using estimator form (11) we obtain

$$\begin{aligned} L(O_1) &= \frac{\rho(\omega^{in,1};x_1,x_1\rightarrow O1)L(x_1,\omega^{in,1})\cos(N_{x_1},\omega^{in,1})}{p(\omega^{in,1};x_1,x_1\rightarrow O1)+p(\omega^{in,1};x_1,x_1\rightarrow O2)} \\ &+ \frac{\rho(\omega^{in,2};x_2,x_2\rightarrow O1)L(x_2,\omega^{in,2})\cos(N_{x_2},\omega^{in,2})}{p(\omega^{in,2};x_2,x_2\rightarrow O1)+p(\omega^{in,2};x_2,x_2\rightarrow O2)} \end{aligned} \tag{15}$$

where $L(x_1,\omega^{in,1})$ and $L(x_2,\omega^{in,2})$ are the incoming radiances to x_1 from direction $\omega^{in,1}$ and to x_2 from direction $\omega^{in,2}$.

4 IMPLEMENTATION

4.1 Algorithm

Once we hit a point in the scene from the eye, we can reuse this information for all the other frames in our camera animation, but only if the point is visible from the other eyes. For this reason and also because of memory restrictions, we are only interested in reusing

[3] For a clearer explanation we also drop here the albedo and solid angle Ω

```
for i = firstFrame to lastFrame do
    currentEye = getEye(i)
    for all pixel in images[i] do
        hit=traceRay(currentEye, pixel)
        for j = firstFrame to lastFrame do
            reuseHitinImage(hit,images[j])
```

Algorithm 1: The algorithm for the *hit harvest* phase, considering only the group of neighbouring frames.

the hit with the closest neighbouring frames, as the probability of being visible is much higher.

We will consider two phases in the computation of our animation frames. The first one is the *hit harvest* (see algorithm 1), where we find the native hit points, compute the rest of the gathering path, connect every hit point with the other eyes to see if they can be reused, and finally, if it is the case, we add a pointer to it in the list of outer hits of the corresponding pixel. Meanwhile, additional probabilities and reflectances needed for the computation are kept in memory. The second phase is the *image computation* itself, in which we use all the stored information, including the native and outer hits for every pixel to compute the final pixel color.

As a few seconds animation involves hundreds of frames, it is not feasible to keep all them in memory at the same time. We have two possible strategies. The first one is to reuse a hit in the n previous and subsequent frames keeping in memory all information needed for the final computation, that will be done once we know we are not reusing more hits for that frame, that is, the $(actual-n)$th frame. But as we can see in equations (15) and (13), for every hit that will be used in a pixel computation, we need to use all probabilities in combination with all the eyes that have been used in the other hits for that pixel. This means keeping in memory also information for frames previous to the $(actual-n)$th, or recompute these probabilities every time we need them. This is a waste of time or a waste of memory.

The second strategy consists in considering a group of $2n+1$ neighbour frames and reuse every native hit in all the other frames in the group, no matter if it is the first one, the last one or the one in the middle. When we are done for the group, instead of moving to the next $2n+1$ frames, we can move just one frame, overlapping $2n$ frames, but without deleting the previous results for the frames that are still active. This previous results can be simply averaged with the new ones. This is the strategy we follow.

4.2 Cost analysis

Now we analyze the relative cost of the animations with and without reuse. Suppose we cast n_r rays per pixel and reuse n_f frames at once. The cost of tracing an eye ray is c_e, the cost of computing the illumination at the hit point in the scene is c_l, and the cost of a visibility test is c_v. In the case of no reuse, the cost of computing n_f frames is $n_f n'_r(c_e+c_l)$. In the reusing case the cost is $n_f n_r(c_e+c_l)+n_f(n_f-1)n_r c_v$, where the second term in the sum accounts for reusing all rays. In the optimal case a ray through a pixel would be equivalent to a native ray, and we compare thus the cost of two animations with equal number of rays per pixel, that is, $n'_r \simeq n_f n_r$. The relative cost for this case is:

$$\frac{n_f n_f n_r(c_e+c_l))}{n_f n_r(c_e+c_l)+n_f(n_f-1)n_r c_v} = \frac{n_f n_r(c_e+c_l)}{n_r(c_e+c_l)+(n_f-1)n_r c_v} = \frac{n_f(c_e+c_l)}{(c_e+c_l)+(n_f-1)c_v} \quad (16)$$

This last expression has the limiting value $\frac{(c_e+c_l)}{c_v}$ when n_f tends to infinite. Supposing $c_l \gg c_e$, limit efficiency will be $\frac{c_l}{c_v}$.

Observe that the above limit efficiency is an upper bound, as on the one hand not all rays will be able to be reused, and on the other hand the variance associated with a reusing frames estimator is higher than with an independent one, because in the independent estimator we have the benefit of importance sampling.

A second, and very important, independent increase in efficiency comes from the reduction in flickering from frame to frame. This reduction is due to that reuse of paths for different frames correlates the computations for all them. And in the way we have constructed our algorithm there is no flickering shown when passing from a group of reused frames to the following one, as we interleave them.

4.3 Memory use

We need a huge amount of memory to keep track of all our computation. We have to keep not only all the images for the current group being computed (including native reflectances and hits), but also the lists of outer hits for every pixel and all possible combinations of probabilities and reflectances per pixel and frame.

Same as before, suppose we use n_h hits per pixel and reuse n_f frames at once. Images are made of $w \times h$ pixels, so we have a total number of pixels $n_{pix} = w \times h \times n_f$.

For every pixel we keep the final color and the number of samples to add and average every result. We also need the list of outer hits for every pixel. The total nodes of outer hits are $n_{nod} = n_h \times n_{pix} \times (n_f-1)$ and are distributed in lists among all the pixels. Every node contains four integers: the image number, the pixel (w and h) and the number of sample, thus it can point towards the data we need to reuse from another image. For every native hit in a pixel we have to keep the native direct and indirect gathered radiance, cosinus weighted, and a n_f-vector containing all probabilities and reflectances if we combined the hit with the rest of eyes.

Just to see the numbers in a concrete example, if we are using 2 samples for every one of the 800×600 pixels and reusing groups of 7 images, we get 3360000 pixels ($800 \times 600 \times 7$), each containing a final color (3 floats), the number of samples (one integer) and a pointer to the list of nodes. We have 40320000 nodes ($2 \times 800 \times 600 \times 7 \times 6$), four integers and a pointer each. We also have the 6720000 native radiances to reuse (direct and indirect, 3 floats each) and 47040000 ($2 \times 800 \times 600 \times 7^2$) probabilities (1 float) and reflectances (3 floats). If we assume each float, integer and pointer is 4 bytes, we need a total memory of 1787520000 bytes for this structure. That's almost all the memory of our 2Gb pentium 4.

5 RESULTS

We have applied the proposed algorithm to an animation computed using path tracing. The frame resolution is 800×600 pixels. We rendered 48 frames, for a 2 seconds animation[4]. For every pixel, we used 2 samples and reused them in groups of seven neighbour frames. As these groups are overlapped, we get a maximum average in number of samples of 98 (2×7^2). The actual average is less than that (between 85 and 90) because some reuses are lost (they can lie out of frame or be hidden by other objects) and it mainly depends on distances between different neighbour eyes, i.e. the smoothness of the camera movement. If camera movement is not smooth or the number of neighbour frames to reuse is too high this ratio decreases, and noticeable differences of noise between different parts of the same image might appear. In our example, as our camera movement corresponds to a zooming of a glossy phong brdf vase, the pixels in the center of the images get more samples than those lying near the borders. This can be interpreted as an advantage, as perception focuses in the center of the image when zooming, some kind of perception driven sampling is performed. The first frames and the last ones present more noise because they cannot be overlapped with previous (in the case of the first ones) or subsequent (for the last ones) groups of frames. Computing time was approximately 40 hours, for a PentiumIV with 2 Gigabytes of memory. That is 50 minutes per frame. A single path tracing image without reuse and with 98 samples per pixel takes more than 5 hours to compute. It is more than 6 times faster, and it can be even faster if we reuse more frames. If we compare this animation with the one computed with Havran et al. method [9], we can appreciate almost no differences. This is because we have reused very few frames and they are very close to each other.

We have computed a second animation with a higher number of reusing frames to prove more clearly the differences between the methods. In this case one hit is reused in 17 frames. We used 2 samples per hit, so we get a maximum average in number of samples of 578 (2×17^2). The actual average is about 500 due to loss of reuses. Due to memory restrictions, resolution is now reduced to 320×240 pixels. Computation time is about 16 hours. This is 20 minutes per frame, almost 8 times faster than the computation time of a single path tracing image with 500 samples per pixel (155 minutes). If we look at Havran et al. version of the same animation we clearly appreciate more noise in the vase in the form of glittering.

In Fig. 3 we show the same frame (the middle frame in our second animation) obtained with three different computations. In the first one (image a) an image with no reuse has been computed using 500 samples per pixel. It takes more than two and a half hours to compute. Image b) shows biased Havran et al. [9] version for reuse of frames. Time computation results in about 18 minutes per frame. Image c) is the result of our unbiased version. It takes 20 minutes to compute, a little more time than b), but we need much more memory. Both images b) and c) present more noise than image a) near the border due to loss of reuses. Image b) presents more noise than image c) in the vase and other glossy objects. Diffuse objects look the same in images b) and c).

In Fig. 4, details for the vase are shown. First image a) is computed with no reuse and 15 samples per pixel. Image b) is biased and computed with the Havran et al. [9] version and image c) is computed with our unbiased method. Both are computed with 2 samples per pixel and reuse of three fairly separated frames, i. e., a maximum average in number of samples of 18 (2×3^2). We clearly see much more noise in image b).

6 CONCLUSIONS AND FUTURE WORK

We have presented in this paper an efficient unbiased method to combine frames in camera animation. It consists in reusing the incoming radiance information of a hit point for the neighbouring frames of the animation. The different probabilities are taken into account and multiple importance sampling technique is used to correctly combine the different samples. Our method makes the difference when using non-diffuse materials, because the diffuse ones distribute reflected rays with equal probabilities in all directions, and when the separation between reusing frames increases. In diffuse cases or when reusing frames are very close, other biased methods can work fine. The main drawback of our method is the large amount of memory needed for the computation.

[4] Animations can be found in http://ima.udg.es/˜amendez/TIC2001/gal_hitreuse.html.

The new method has been demonstrated with an animation of a camera in a scene that contains a vase with a glossy brdf, computing the global illumination using the path-tracing technique.

Future work will be addressed to increase the efficiency of our approach using coherence in visibility computation, by guessing on the one hand the visibility for one observer from the results for neighbour observers and on the other hand by using an acceleration schema similar to [8]. We will also try to combine both the benefits of this approach and the reuse of paths for light source animation [18]. Combination with an adaptive sampling technique, i.e., using more native samples for those pixels that come with not enough outer hit samples, because they are occluded by other objects or because they lie near the bounder of the image.

ACKNOWLEDGEMENTS

This project has been funded in part with Gametools project from the European VIth Framework, with grant number TIN2004-07451-C03-01 from the Spanish Government and with Hungarian-Spanish Joint Action number HH2004-0011.

REFERENCES

[1] Philippe Bekaert. *Hierarchical and Stochastic Algorithms for Radiosity*. PhD thesis, Department of Computer Science, Katholieke Universiteit Leuven, Leuven, Belgium, 1999.

[2] Philippe Bekaert, Mateu Sbert, and John Halton. Accelerating path tracing by re-using paths. In *Rendering Techniques 2002 (Proceedings of the Thirteenth Eurographics Workshop on Rendering)*, pages 125–134, June 2002.

[3] Gonzalo Besuievsky. *A Monte Carlo Approach for Animated Radiosity Environments*. PhD thesis, Universitat Politecnica de Catalunya, Barcelona, Spain, 2001.

[4] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed Ray Tracing. In *Computer Graphics (ACM SIGGRAPH '84 Proceedings)*, volume 18, pages 137–145, July 1984.

[5] Philip Dutre, Philippe Bekaert, and Kavita Bala. *Advanced Global Illumination*. AK Peters Limited, 2003.

[6] John Halton. Sequential monte carlo techniques for the solution of linear systems. *Journal of Scientific Computing*, 9(2):213–257, 1994.

[7] J. Hammersley and D. Handscomb. *Monte Carlo Methods*. Chapman and Hall, London, 1979.

[8] Vlastimil Havran, Jiri Bittner, and Hans-Peter Seidel. Exploiting temporal coherence in ray casted walkthroughs. In *Proceedings of the Spring Conference on Computer Graphics 2003 (SCCG 2003)*, April 2003.

[9] Vlastimil Havran, Cyrille Damez, Karol Myszkowski, and Hans-Peter Seidel. An efficient spatio-temporal architecture for animation rendering. In *Proceedings of Eurographics Symposium on Rendering 2003*, pages 106–117. ACM SIGGRAPH, June 2003.

[10] Henrik Wann Jensen. Global Illumination Using Photon Maps. In *Rendering Techniques '96 (Proceedings of the Seventh Eurographics Workshop on Rendering)*, pages 21–30. Springer-Verlag/Wien, 1996.

[11] James T. Kajiya. The Rendering Equation. *Computer Graphics (ACM SIGGRAPH '86 Proceedings)*, 20(4):143–150, August 1986.

[12] M.H. Kalos and P.A. Whitlock. *Monte Carlo Methods*. John Wiley & Sons, 1986.

[13] Eric P. Lafortune and Yves D. Willems. Bi-directional Path Tracing. In H. P. Santo, editor, *Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*, pages 145–153, Alvor, Portugal, December 1993.

[14] Àlex Méndez-Feliu and Mateu Sbert. Combining light animation with obscurances for glossy environments. *Computer Animation and Virtual Worlds*, 15(3-4):463–470, july 2004.

[15] R.Y. Rubinstein. *Simulation and the Monte Carlo Method*. Wiley Series in Probabilities and Mathematical Statistics, 1981.

[16] Mateu Sbert, Philippe Bekaert, and John Halton. Reusing paths in radiosity and global illumination. *Monte Carlo Methods and Applications*, 10(3-4):575–586, 2004.

[17] Mateu Sbert, Francesc Castro, and John Halton. Reuse of paths in light source animation. In *Proceedings of Computer Graphics International 2004 (CGI '04)*, pages 532–535. IEEE Computer Society, June 2004.

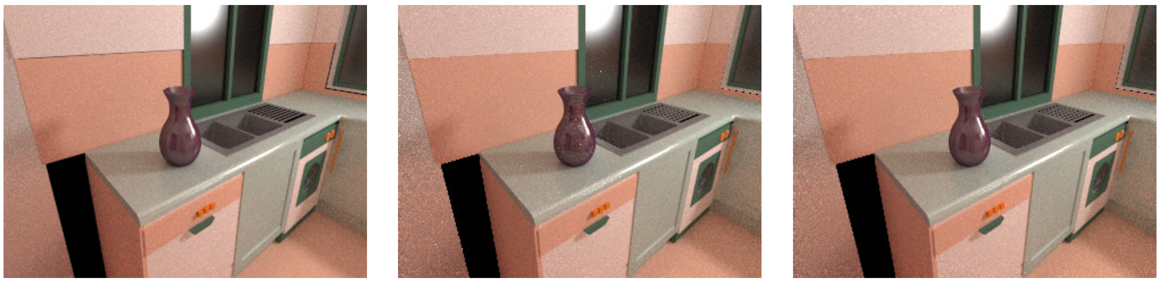
[18] Mateu Sbert, Laszlo Szecsi, and Laszlo Szirmay-Kalos. Real-time light animation. *Computer Graphics Forum (Eurographics 2004 Proceedings)*, 23(3):291–299, September 2004.

[19] László Szirmay-Kalos, Mateu Sbert, Roel Martínez, and Robert F. Tobler. Incoming first-shot for non-diffuse global illumination. In *Spring Conference on Computer Graphics*, Budmerice, Slovakia, 2000. Available from http://www.fsz.bme.hu/˜szirmay/puba.htm.

[20] Eric Veach. *Robust Monte Carlo Methods for Light Transport Simulation*. PhD thesis, Stanford University, December 1997. Available from http://graphics.stanford.edu/papers/veach_thesis.

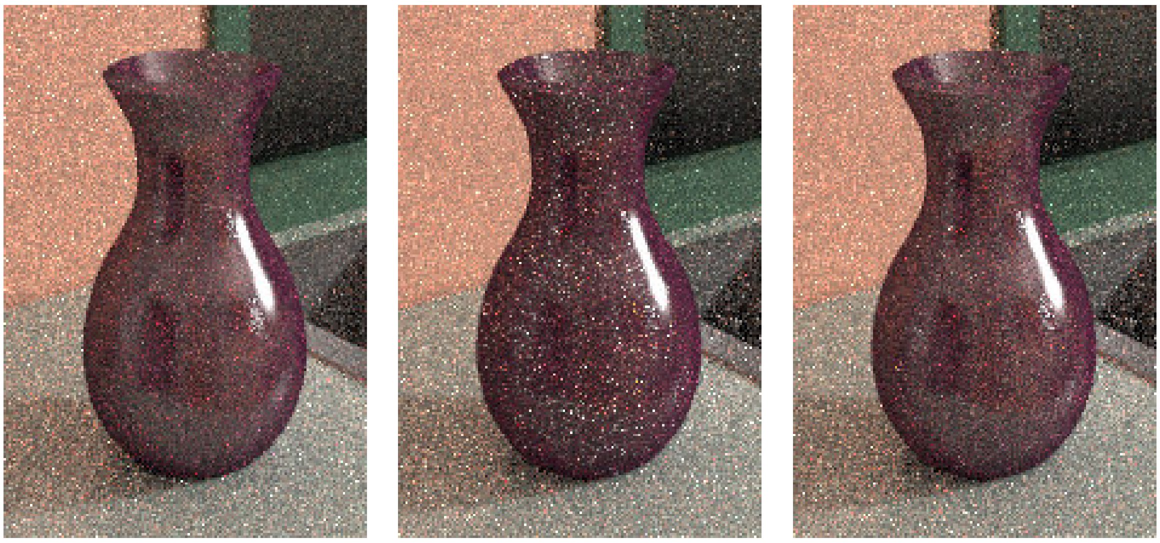
[21] Eric Veach and Leonidas J. Guibas. Optimally Combining Sampling Techniques for Monte Carlo Rendering. In *Computer Graphics Proceedings, Annual Conference Series, 1995 (ACM SIGGRAPH '95 Proceedings)*, pages 419–428, 1995.

[22] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *Computer Graphics (ACM SIGGRAPH '97 Proceedings)*, volume 31, pages 65–76, 1997.



(a) (b) (c)

Figure 3: Here we show the same frame (the middle frame in our animation) obtained with three different computations. In the first one (image a) an image with no reuse has been computed. It takes more than 2,5 hours to be computed. Image b) shows Havran et al. [9] version for reuse of frames and takes 18 minutes to compute. Image c) is the result of our unbiased version and takes 20 minutes. The unbiased c) version uses much more memory. Images b) and c) presents noise near the border due to loss of reuses, and image b) presents noise in glossy objects due to biased computation.



(a) (b) (c)

Figure 4: The differences between the methods are clearly appreciated for non-diffuse materials when we reduce the computation time (noise is higher, consequently) and the separation between frames increases. Here we see the details of the vase for the 800×600 image when reusing only 3 frames fairly separated. First image a) is computed with no reuse. Image b) is biased and computed with Havran et al. [9] version and image c) is computed with our unbiased method. We clearly see much more noise in image b).