Monte Carlo Methods Appl. Vol. No. (), pp. 1–16
DOI 10.1515 / MCMA.2007.

Fast GPU-based reuse of paths in radiosity

Francesc Castro, Gustavo Patow, Mateu Sbert and John H. Halton

Abstract. We present in this paper a GPU-based strategy that allows a fast reuse of paths in the context of shooting random walk applied to radiosity. Given an environment with diffuse surfaces, we aim at computing a basis of n radiosity solutions, corresponding to n light-source positions. Thanks to the reuse, paths originated at each of the positions are used to also distribute power from every other position. The visibility computations needed to make possible the reuse of paths are drastically accelerated using graphic hardware, resulting in a theoretical speed-up factor of n with regard to the computation of the independent solutions. Our contribution has application to the fields of interior design, animation, and videogames.

Keywords. Computer graphics, animation, global illumination, radiosity, random walks, graphic hardware.

AMS classification. 65C05,65C35.

1. Introduction

Radiosity techniques [5, 20] were introduced to the computer graphics field in 1984 [6]. They provide accurate realistic solutions for the global illumination problem, especially when dealing with diffuse environments.

The kernel of the radiosity equation includes the form-factors, which are geometric terms in which the visibility among the surfaces in the environment is contained. The explicit computation or the simulation of the form-factors happens to be the most costly part of the radiosity algorithms, basically because of the visibility computations.

Monte Carlo random walk algorithms [14] have a wide application to radiosity [15, 1]. In such algorithms, the paths are built by tracing rays that move among the patches in which the environment has been discretized with form-factors as transition probabilities. The present paper deals specifically with *shooting* random walk, in which the paths start at the light sources, and simulate the light trajectory.

The main drawback of radiosity random walk is the large number of paths needed to obtain an acceptable result, that is a noiseless image, due to the error behavior in Monte Carlo simulations, $O(\frac{1}{\sqrt{N}})$, N being the number of paths.

We can reduce the cost of such a random walk by reusing the paths [2, 16]. We present in this paper a GPU-based algorithm that allows the fast reuse of the shooting paths for different light-source positions. Paths starting at each of the light positions

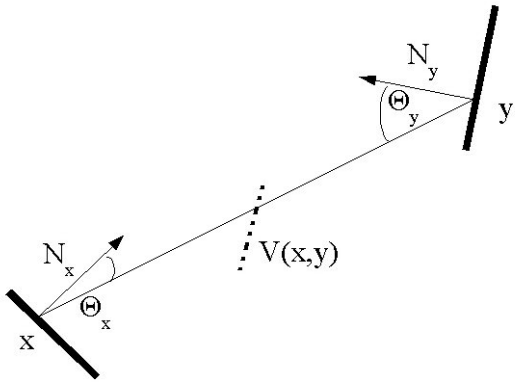

Figure 1. *Point-to-point form-factor geometry.* $V(x, y)$ *will be* 1 *if points* x *and* y *are mutually visible, and* 0 *otherwise.*

serve for distributing power from all of them, being reused from the first hit on. Our algorithm permits this reuse with practically no additional cost, resulting in a theoretical maximum speed-up factor equal to the number of light positions considered. This algorithm has applications to the fields of animation, interior design, and videogames.

This paper is structured as follows: section 2 deals with the previous work; section 3 presents our contribution; section 4 reports the results obtained in our experiments; finally, in section 5, the conclusions and future research related to this work are presented.

2. Previous work

2.1. Radiosity techniques

The radiosity (either for a patch or a point) is the combination of the emitting and the reflected power per unit area. The equation for the radiosity $B(x)$ of a point x is:

$$B(x) = \int_D \rho(x)F(x, y)B(y)dy + E(x), \tag{2.1}$$

where $\rho(x)$ and $E(x)$ respectively stand for the reflectance (fraction of the incoming light that is reflected) and emittance (emitting light) of point x, and $F(x, y)$ stands for the form-factor between point x and point y. The form-factor between two points (see Fig. 1) is a geometric term, integrating to one for either x or y. It is given by:

$$F(x, y) = \frac{\cos\theta_x \cos\theta_y V(x, y)}{\pi r^2}, \tag{2.2}$$

where θ_x, θ_y are the angles between the line joining x and y and the respective normal vectors, r is the distance between both points and $V(x, y)$ is the visibility function (with value 0 or 1).

In practice, a discrete version of the radiosity equation is considered, corresponding to a discretization of the environment in small polygons or patches. We have then, for each patch i,

$$B_i = E_i + \rho_i \sum_{j=1}^{N_p} F_{ij} B_j, \tag{2.3}$$

where B_i, E_i, and ρ_i stand respectively for the radiosity, emittance, and reflectance of patch i (assumed to be constant along the patch), F_{ij} stands for the form-factor from patch i to patch j, and N_p stands for the number of patches in the scene. Considering all the patches in which the environment is discretized, we have a linear system of equations.

2.2. Shooting Random Walk in radiosity

Monte Carlo methods have been widely applied to the radiosity techniques [15, 1, 3], for both computing the form-factors or simulating their distribution. Unbiased Monte Carlo estimators allow to obtain results as accurate as desired by simply taking a large enough number of samples.

Shooting random walk (also known as adjoint method in MC literature) is a common Monte Carlo technique valid to solve linear systems of equations [14]. In the case of the radiosity, each walk (shooting path) starts at a point uniformly sampled on a light-source surface. Importance sampling [8, 14, 11] is applied to the transition computation, which in radiosity case involves taking form-factors (not explicitly computed) as transition probabilities among the patches. Each shooting path can be seen as a light particle that carries Φ/N power, where Φ is the total source power, and N is the total number of shooting paths. Estimators for the incoming power to each patch are built from the contributions of the shooting paths (see [1]).

2.3. Reuse of paths in random walk

The idea of reusing full paths for different states (pixels, patches, or light sources) was introduced by Halton in [7], in the context of sequential Monte Carlo methods for solving of linear systems.

Bekaert et al. [2] applied the idea in [7] to the context of path-tracing, combined with multiple importance sampling [21] to avoid bias. Pixels were grouped in tiles and paths from one pixel were reused for the rest of the pixels in the tile from the second hit on. This algorithm produced an speed-up factor of one order of magnitude for fairly complex scenes. Also, Sbert et al. [16] presented a theoretical framework

valid to reuse paths in gathering random walks, and outlined applications to radiosity and global illumination.

Regarding to light-source animation, Sbert et al. [18] presented a fast algorithm based on the virtual light sources illumination method [22]. They reused paths in all frames (and not only in the frame where paths were obtained). This algorithm presented a speed up closely proportional to the number of frames of the animation.

Regarding to camera animation, Havran et al. presented [10] the reuse of paths in a walk-through, namely when the observer changes position. Paths cast from one observer's position were reused for other neighbor positions. This method produced an important speed-up factor, but it remained biased since the samples were not weighted with the respective probability. Also Havran et al. [9] presented a technique that aimed at exploiting temporal coherence of ray casted walkthrough. They reused ray/object intersections computed in the last frame of the walkthrough for accelerating the ray casting in the current frame. Méndez et al. [12] recently presented an unbiased solution for frame reuse in camera animation, showing how the different contributions can be combined into a unbiased solution using multiple importance sampling [21].

2.3.1 Applications to radiosity

Sbert et al. [17] introduced an algorithm valid for the reuse of full shooting paths in the case of moving light sources. Given n positions on a light-source trajectory, they aimed at computing one radiosity solution for each position by considering the paths started at all of the positions, in a sort of each-for-all strategy that allowed to reuse each path from the first hit on. Next, we will describe in more detail this algorithm.

First, let us consider N points from a uniform distribution on a light source with emitting power Φ. Considering the emission intensity to be constant on the light-source surface, each point has to distribute equal power, that is Φ/N. Let x be one of these points. The next integral expresses the total amount of power to be distributed from x to the environment (considering a closed environment):

$$\Phi(x) = \int_D F(x,y)\frac{\Phi}{N}dy = \frac{\Phi}{N}. \tag{2.4}$$

Now, consider the Monte Carlo estimation of (2.4) [1], using n samples y^k generated from the pdf $p(y)$:

$$\Phi(x) \approx \widehat{\Phi(x)} = \frac{\Phi}{nN}\sum_{k=1}^{n}\frac{F(x,y^k)}{p(y^k)}. \tag{2.5}$$

This expression can be seen as a sum of n terms, the k-th term corresponding to the power that y^k (receiver) receives from x (shooter), or, in terms of shooting paths,

[1] As $\int_D F(x,y)dy = 1$ for closed environments, this is a trivial integral; the purpose of this is to obtain an estimator for the received power at y.

to the power that carries the k-th such a path. Note that if we simply take as pdf the form-factor, that is $p(y^k) = F(x, y^k)$, then, according to (2.5), each path will carry a constant power $\Phi/(Nn)$, the classic case where each shooting path carries the same amount of power.

In order to make the reuse of paths possible, let us consider the n light positions which we referred to before. Let $x_1, ..., x_n$ be the point x taken at each of the positions. Points y will be sampled from any of the points $x_1, ..., x_n$, corresponding to using a mixture pdf [21] which is just the average of the n form-factors:

$$mix(y) = \frac{F(x_1, y) + F(x_2, y) + ... + F(x_n, y)}{n}. \tag{2.6}$$

Let j be one of the n positions, and let x_j be the point taken at this position. If points y are generated according to the mixture pdf (2.6), the power to be distributed from x_j to the environment is, according to (2.5):

$$\widehat{\Phi(x_j)} = \frac{1}{n} \sum_{i=1}^{n} \frac{F(x_j, y_i)}{mix(y_i)} \times \frac{\Phi}{N}, \tag{2.7}$$

This estimator is more accurate the more similar is the pdf corresponding to x_j, i.e., $F(x_j, y)$, to $mix(y)$ (the optimal case would be when all the points $x_1, ..., x_n$ and the corresponding orientations were the same). This implies that the more grouped the positions and the more similar their orientations, the more accurate the results. This also means that, for a given set of light positions, the solutions corresponding to central positions are in general more accurate that the ones corresponding to peripheral positions.

The estimator in (2.7) can be seen as a sum of n terms, being the i-th term the amount of the power from x_j carried by the shooting path Γ_i, which has started at x_i. Let $\Phi_j(\Gamma_i)$ be this power:

$$\Phi_j(\Gamma_i) = \frac{F(x_j, y_i)}{\sum_{l=1}^{n} F(x_l, y_i)} \times \frac{\Phi}{N}, \tag{2.8}$$

where y_i corresponds to the first hit point of the path, $x_1, ..., x_n$ corresponds to point x_i at each of the n light-source positions, and the $F(x, y)$ are the point-to-point form-factors (2.2).

Summing up, this algorithm assigns to each path, and for each position j, a weight in $[0, 1]$ that, multiplied by Φ/N, gives the amount of power that this path has to carry from x_j to the scene. Fig. 2 illustrates such a reusing strategy.

This algorithm was applied to light-source animation. All the n frames were computed at the same time with a noticeable reduction of the cost with respect to computing them independently. Moreover, using the same paths for all the frames produced a

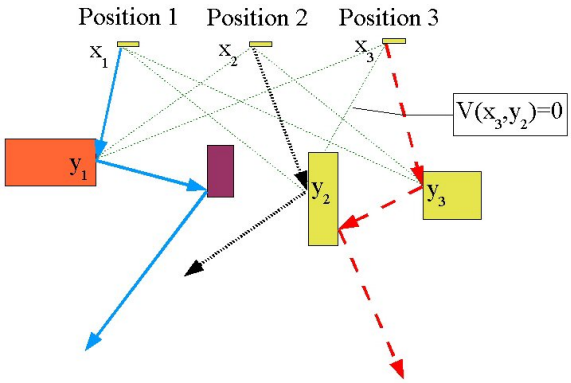

Figure 2. *Reuse of shooting paths. Consider a point x in 3 diferent positions x_1, x_2, x_3, one for each light-source position. The path started at x_1 (blue/continuous) is reused to distribute power from x_2 and x_3 too, and the same with the paths started at x_2 (black/dotted) and x_3 (red/dashed). Note that point y_2 is not visible from x_3, so the path started at x_2 will not contribute to expand any power from x_3.*

clear reduction of the flickering effect, resulting in a smoother animation. By considering the n frames as a basis of n radiosity solutions, this algorithm can also be useful for the computation of n static scenes in order to choose the best light position or to linearly combine some of this solutions [4].

3. Accelerating the reuse of shooting paths in light-source animation

The reusing technique described in section 2.3.1involved the computation, for each path, of n point-to-point form factors (2.2) (n being the number of light-source positions), needed to compute the weights for each path (see equation (2.8)). Thus, such a technique involved an additional cost that practically corresponded to the point-to-point form factor computation cost, which, at the same time, was basically due to the visibility computation, that is, to determine whether both points were mutually visible or not.

The cost above can be reduced by approximating the point-to-point visibility functions instead of computing them exactly. This can be done by discretizing in angular regions the directions over each light-source position and considering the polygon seen from each position through each angular region, like a kind of shadow mapping technique [23, 13, 19].

We first describe an algorithm, already sketched in [4], to carry out the strategy above, and, later, we introduce an acceleration for this algorithm based on the use of the GPU.

3.1. Fast visibility computation in the CPU

We consider the directions over each of the n light-source positions. We introduce a preprocess in which such directions are discretized in angular regions, and aim at computing, for each angular region, some information that will be stored and used later to approximate the visibilities.

Let us examine the preprocess in detail. For each light-source position, we build a structure, referred to from here on as *visibility map*, in the following way. We consider a hemisphere centered on the light-source center, and discretize it in angular regions, corresponding to the directions over the position. For each angular region, a ray is deterministically generated taking one direction as a representative of the region, and the nearest intersected polygon identifier (we assume all polygons in the environment planar) is recorded, and the intersection distance is computed and stored. The algorithm in Fig. 3 describes this preprocess.

Preprocess: building visibility maps

for each *light-source position* **do**
  Build an hemisphere discretized in angular regions
  for each *angular region* **do**
    Cast a ray (from the source center)
    Compute and store:
     nearest intersected polygon identifier
     intersection distance
  endfor
endfor

Figure 3. *Building the visibility maps.*

The information gathered during this preprocess is used during the shooting stage in order to avoid the costly visibility computations. The algorithm in Fig. 4 shows how it works (considering a random path started at position i). Fig. 5 shows an example of the application of the algorithm.

This optimization entails a reduction of the cost: approximating the $n-1$ visibilities per path, according to the algorithm in Fig. 4, is much faster than computing them exactly, and the preprocess happens to be computationally cheap in relation to the full process, since only one ray per angular region is needed, and, in general, not many angular regions are required.

We have to mention that, since we do not use the exact values of the visibilities but approximations, this optimization results in a biased solution. However, for a small enough light source and an appropriate discretization, the bias happens to be visually

Approximating the visibility computations

Sample a point x on position i
Sample the direction for a shooting path from x
Search the nearest intersection for the path:
  ** Let P be the nearest intersected polygon*
  ** Let y be the intersection point*
  ** Let $\overline{xy}$ be the distance between x and y*
for each *position k ($k \neq i$)* **do**
  Compute vector v from the center of k to y
  Determine r, the angular region over k corresponding to v
  Obtain the pre-stored information corresponding to r:
    Let $p[r]$ be the nearest intersected polygon for r
    Let $d[r]$ be the intersection distance for r
  if $p[r] \neq P$ **and** $d[r] < \overline{xy}$ **then**
    y non-visible from position k (occlusion)
  else
    y visible from position k
endfor

Figure 4. *Using the visibility maps to approximate the visibility in the shooting stage, for a path started at the light-source position i.*

acceptable (see experiments in section 4.3).

3.2. GPU acceleration for the visibility computation

Usage of modern graphics hardware, with its increasing power and computation capabilities, seems to be a logical next step to improve the efficiency of the above described algorithm. The advantages of including graphics processing unit (GPU) in the computations are twofold:

- On the one hand, it will allow a faster preprocessing of the scene, building the visibility maps at a fraction of the computational cost of the CPU solution, due to the inherent parallelism of the GPU. This will enable us to increase the discretization of the visibility maps with almost no impact on the overall computations.
- On the other hand, the computations of the form factors for each first intersection will also be much faster, as again we will take advantage of the high capabilities of modern GPUs, allowing the approximation of the visibilities as a simple point-wise rasterization process.

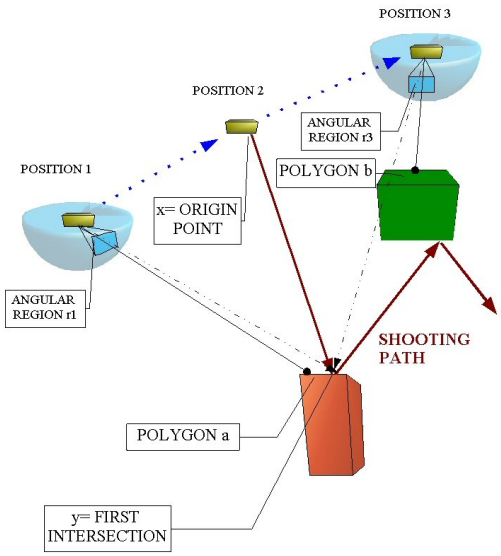

Figure 5. *Approximating the visibility computations via visibility maps. Since the pre-stored polygon for angular region* $r3$ *is* $b \neq a$*, and* b *is closer to position* 3 *than* a*, we consider point* y *to be non-visible from position* 3*.*

3.2.1 The algorithm

The proposed GPU-based approach moves two of the most time consuming steps from the CPU to the GPU: the computation of the visibility maps (described in Fig. 3) and the form factors computation, including the approximation of the visibilities (described in Fig. 4). For a given scene (see Fig. 7 (left)), our GPU-based approach proceeds as follows:

- **Step 1: Generation of the visibility maps (at the GPU).** For each light-source position, we build an hemicube at its center, generating a visibility map by taking the images of the surrounding environment, and storing the polygon identifiers (see Fig. 7 (center)). Note that the distances are not stored in this approach, in order to adapt the algorithm in section 3.1 to the GPU features.

- **Step 2: First hit point generation (at the CPU).** This stage consists of finding the first hit points y for each path, which is achieved by starting a given number of paths, according to the form factor distribution (see [15]), at each light-source position, and recording for each path the resulting hit point, the normal at this point, and the identifier for the hit polygon. This information is sent to the GPU (see Fig. 7 (right)).

- **Step 3: Visibility and form-factor computations (at the GPU).** For each first hit point y for a path, represented by a vertex in the GPU, its visibility from each light-source position i is computed. In order to do so, the vertices, stored in the previous step in the corresponding data structure, are projected onto the cube maps. Then, the polygon identifiers stored in Step 1 are compared to the ones corresponding to the first hit polygons. If they are the same, the visibility function is considered to be 1 and the form-factor $F(c_i, y)$ (c_i being the center of the light at position i) is computed and stored. If not, a zero value is stored, corresponding to $F(c_i, y)$, since in this case point y is considered to be non-visible from position i. Once finished, the process is iterated for the next light-source position (see Fig. 8). We also accumulate, for each point y, the sum of the form factors $F(c_i, y)$ through all the positions i. In Fig. 6 we have the pseudocode for this step:

Computing the form factors in the GPU

SFF := 0 //clean the sums of the FF data structure
For each *light-source position* i, $i = 1..n$
  Project the $\{y\}$ *on the map for* i
  For the ones that pass the visibility test
    compute the form factor $F(c_i, y)$
    SFF[y] += $F(c_i, y)$
    FF[y] = $F(c_i, y)$
  endfor
  Send the FF data structure to the CPU
endfor
Send the SFF data structure to the CPU

Figure 6. *Form-factor computation via hardware. FF stands for the data structure recording the form-factors, SFF for the data structure recording the sums of form-factors, and* $\{y\}$ *for the set of first hit points.*

- **Step 4: Path continuation (at the CPU).** Once the form factors for all vertices as seen from all light-source positions have been approximated, the process follows by computing the paths continuations.

4. Results

In our experiments, we will consider 4 different shooting random walk approaches valid for computing a set of radiosity solutions corresponding to n light-source posi-

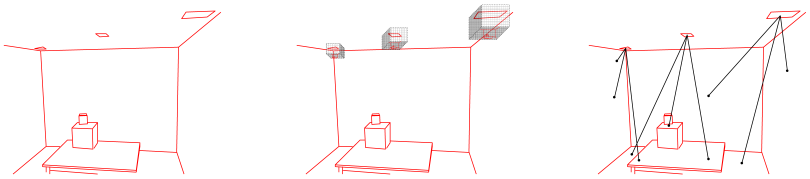

Figure 7. *(left) Scene with three light-source positions near the ceiling. (center) Precomputing the hemicubes to store the scene visibility information. (right) A set of first hit points* $\{y\}_i$ *is generated for each light-source position.*

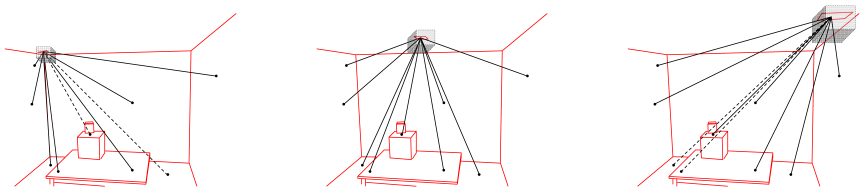

Figure 8. *Computing the visibilities and form-factors for the* $\{y\}$ *set (*$3 \times 3 = 9$ *points) by projecting them onto each cube map.*

tions (from here on, we refer to such approaches using the corresponding capital letters *A, B, C,* and *D*):

- *A*- Computing n independent solutions (without reuse).
- *B*- Reusing paths with the exact computation of the visibility functions.
- *C*- Reusing paths, and approximating the visibility functions in the CPU (see section 3.1).
- *D*- Reusing paths, and approximating the visibility functions via GPU (see section 3.2).

4.1. Analysis of the cost

Let us assume, for the sake of simplicity in the analysis, the cost of generating a ray from a point in a surface and finding the nearest intersection to be constant, and let K be this cost. Note that K practically corresponds to the cost of computing the nearest intersection. Since a point-to-point visibility computation can be reduced to find the nearest intersection in a given direction for one of both points, K can be also taken as the cost of a visibility computation.

Let n be the number of light-source positions. If we aim at taking N effective paths per position, in the approach *A* (without reuse), it involves generating N paths at each of the n positions. By contrast, if we reuse the shooting paths (*B, C,* and *D*), only N/n paths starting at each position are needed, thanks to the path amortization.

Without reuse, the average cost of computing each of the Nn paths is Kl (l being the average length of a shooting path), while involving reuse we have to consider, for each path, the computation of n point-to-point form factors (see equation 2.2. If we compute the visibility functions $V(x, y)$ exactly, as in *B*, such form factor computations practically reduce to the computation of $n - 1$ visibilities per path (see Fig. 2), resulting in an average cost, for each of the N paths, of $Kl + K(n - 1)$. This gives a maximum theoretical speed-up factor of $\frac{ln}{l+n-1}$, which, if n becomes large, gives a theoretical bound of l for the speed up (see [17]).

In the approaches *C* and *D*, where the visibilities are not computed but approximated, the computation of the n form factors per path is clearly accelerated with respect to *B*. Let $K'n$ be the average cost of computing the n form factors per path, where $K' << K$. In such approaches, we have also to add the cost of the pre-computations needed to build the visibility maps. Let $K''n$ be the average cost of such pre-computations (note that such a cost does not depend on the number of paths). Thus, the speed-up factor results, in average:

$$\frac{\text{time for classical method}}{\text{time for reusing method}} = \frac{NnKl}{NKl + NK'n + K''n}. \tag{4.1}$$

In the approach *D*, the use of the GPU for the pre-computations drastically reduces the values of K' and K''. Taking the values of N needed to obtain acceptable results, $NK'n + K''n$ happens to be nearly negligible compared to NlK, the cost of the shooting paths. Thus, according to (4.1), it gives a theoretical speed-up factor close to n. In other words, the approach *D* allows the reuse of paths nearly for free, driving to a maximum speed-up equal to the number of light-source positions.

4.2. Error and bias

In order to express the accuracy of the results in our experiments, we have used the mean square error (MSE) of our results with respect to reference solutions obtained taking a large number of paths per position.

We have used the *relative efficiency*, i.e. the ratio of efficiencies, throughout this article in order to compare the results for the different approaches. The efficiency can be estimated as the inverse of the product of the execution time and the MSE.

We have to remind (see section 3.1) that approaches *C* and *D*, in which visibilities are approximated instead of computed exactly, incorporate a bias given by the discretization of the directions over each position, and for the use of non-point light sources. Such a bias, which for an appropriate discretization and a small-enough light source happens to be visually negligible (see section 4.3), has been taken away from the MSE in the results presented in this article.

4.3. Experiments

We have tested a scene *(experiment* 1*)* in which the light source moves in a curved trajectory with changing orientation along $n = 30$ positions. Different executions, varying the number of paths started at each position, have been done for each of the 4 approaches. Results are summarized in the graphs in Fig. 9 for 2 out of the 30 frames: the first one (top), and the 15-th one (bottom). From this graphs, we note that, on the one hand, the performance of the techniques involving reuse is better in frame 15 than in frame 1: this is in general valid for the central frames regarding to the extreme ones, as seen in Section 2.3.1 On the other hand, we also note that the performance of approaches C and D is clearly superior to the one for the approach B, in the sense that using the same number of paths per position in the three approaches produces a similar error (once taken the bias), but the computational cost is drastically reduced in C and specially in D, thanks to the visibility approximations and to the use of the GPU.

In the graph in Fig. 10 we aim at showing the gain due to the use of the GPU, comparing the cost for building the visibility maps plus the cost for computing the form factors for both approaches *C* and *D*. We can observe a speed-up factor of about 15 when using the GPU (D) regarding to the same computations done in the CPU (C).

Animation sequences 1A, 1B, 1C, and 1D in the additional material (see `ima.udg.edu/~castro/videosMCMA`) have been respectively obtained with approaches *A, B, C,* and *D*. For all of these animations, 200000 shooting paths have been generated at each light-source position, meaning, when reusing the paths (approaches *B, C*, and *D*, a total of 6 milions of effective paths per frame. Table 1 shows the total time for the 30 frames and the MSE and relative efficiencies for frames 1 and 15. In Fig. 11 we can see the 15-th frame from the 3 animation sequences involving reuse.

In *experiment* 2 we have tested a second scene where the light source has a diagonal straight trajectory near the ceiling. Here, the light orientation does not change during the animation. 3 sequences, labelled as 2B, 2C, and 2D (see the additional material) have been generated, obtained with the approaches *B, C*, and *D*, respectively. The number of paths from each position is the same as in experiment 1, and the gain obtained thanks to the visibility approximation is similar to the one in experiment 1. In Fig. 12, we can see one of the frames from the 3 animation sequences. The relative efficiencies with respect to a solution obtained without reuse are, for this frame, of about 1.8, 7.9, and 9.8 respectively.

The relative efficiencies obtained in both experiments with approach D are far from the theoretical speed-up factor (regarding to the approach without reuse), which is about the number of positions (see Section 4.1), 30 in our tests. As seen in Table 1, the MSE does not divide by the number of positions when reusing the paths (it would divide if all the light positions and orientations were the same), but it reduces less, especially in the extreme frames. However, this fact is compensated by the reduction of the flickering effect in the whole sequence of frames when reuse is involved, resulting in such cases in softer animations. This has to be considered as an important added

APPROACH	TIME	MSE f.1	MSE f.15	rel.efficiency f.1	rel.efficiency f.15
A	345”	0.111	0.0781		
B	1354”	0.0462	0.0123	0.61	1.62
C	438”	0.0444	0.0109	1.97	5.64
D	347”	0.0499	0.0112	2.21	6.93

Table 1. *Experiment 1:* 200000 *paths generated at each of the positions in all cases, meaning, in approaches* B*,* C*, and* D*, a total of* $6M$ *of effective paths per frame, thanks to the reuse. Mean square error (MSE) regarding to the corresponding reference solutions has ben computed for frames 1 and 15 in a* 30*-frames animation. In (B) all the visibilities have been explicitly calculated; in (C) they have been approximated using the CPU; (D) corresponds to the GPU-based acceleration described in Section 3.2. Relative efficiencies regarding to the solution without reuse (approach A) are also shown.*

value to the reuse techniques when applied to animation (see this effect in animations corresponding to *Experiment 1* in the additional material).

We also have to note that, in both experiments, the 3 animations involving reuse look very similar, meaning that the bias introduced by the discretization of the directions in approaches *C* and *D* is not visually noticeable. On the other hand, the MSE obtained after subtracting the bias in the solutions corresponding to C and D is practically the same for most of the frames, meaning that the approximation using visibility maps does not affect the error convergence.

5. Conclusions and future work

We have presented an algorithm that allows the reuse of shooting paths nearly for free in the radiosity random walk context. This algorithm is valid for computing a set of radiosity solutions corresponding to different source positions and orientations.

The strategy that makes possible the reuse of the shooting paths needs the computation of point-to-point visibilities. The contribution presented in this article is based on replacing the exact computation of such visibilities by an approximation using visibility maps, reducing in this way the computational cost. The use of the GPU in order to build the visibility maps, approximate the visibilities, and compute the form factors, happens to be an added value to our algorithm, allowing the reuse of the paths practically for free. As shown in our experiments, this drives us to a speed-up factor with respect to the computation without reuse close to the number of positions. The relative efficiency obtained depended on the frame considered: in general, the more central the light-source position, the lower the error and thus the higher the relative efficiency.

Our contribution is appropriate for light-source animation, allowing the computation of all the frames at the same time. In this case, the reduction of the flickering in the animation, due to the use of the same paths for all the frames, happens to be an added value to the reuse techniques. In static environments, our contribution can be applied to compute a basis of radiosity solutions in order to choose, from a set of positions, the best one to place the light-source, or to test different light configurations.

Some future lines of research can be outlined. First, we could study non-uniform distributions of the number of paths to be generated at each light-source position (for instance, generating more paths the more extreme the position is, in order to compensate for the higher error of these frames). Also, the reuse frame could be adapted to environments with moving objects (in addition to moving light sources). Finally, we can consider the use of the GPU not only for the form-factor computation but also for the path tracing.

Acknowledgments. We wish to thank Jordi Rovira for his collaboration with the GPU code. This project has been funded in part with grant number TIN2007-68066-C04-01 from the Spanish Government, and GameTools Project (number IST-2-004363) from the European Union.

References

1. P. Bekaert, *Hierarchical and Stochastic Algorithms for Radiosity*, Ph.D. thesis. Katholic Univ. of Leuven (1999).
2. P. Bekaert, M. Sbert and J. Halton, *Accelerating Path Tracing by Re-Using Paths*, Proceedings of Workshop on Rendering (2002), pp. 125–134.
3. F. Castro, *Efficient Techniques in Global Line Radiosity*, Ph.D. thesis. Universitat Politècnica de Catalunya, Barcelona (2002).
4. F. Castro, M. Sbert and J. Halton, *Efficient reuse of paths for random walk radiosity*, To appear in Computers and Graphics journal.
5. M. Cohen and J. Wallace, *Radiosity and Realistic Image Synthesis*. Academic Press Professional, 1993.
6. C.M. Goral, K.E. Torrance, D.P. Greenberg and B. Battaile, *Modeling the Interaction of Light between Diffuse Surfaces*, Computer Graphics (ACM SIGGRAP Conf.Proc.) 18, N.3 (1984), pp. 213–222.
7. J. Halton, *Sequential Monte Carlo techniques for the solution of linear systems*, Journal of Scientific Computing, 9(2) (1994), pp. 213–257.
8. J.M. Hammersley and D.C. Handscomb, *Monte Carlo Methods*. Methuen and Co. Ltd., 1975.
9. V. Havran, J. Bittner and H.P. Seidel, *Exploiting Temporal Coherence in Ray Casted Walkthroughs*. Proceedings of the 19th Spring Conference on Computer Graphics 2003 (SCCG 03) (Kenneth I. Joy, ed.), pp. 164–172. ACM, Budmerice, Slovakia, April 2003.
10. V. Havran, C. Damez, K. Myszkowsky and H.P. Seidel, *An Efficient Spatio-Temporal Architecture for Animation Rendering*, Rendering techniques '03 (2003), pp. 106–117.

11. M.H. Kalos and P. Withlock, *Monte Carlo Methods, Volume I*. John Wiley and Sons, 1984.
12. A. Méndez Feliu, M. Sbert and L. Szirmay-Kalos, *Reusing Frames in Camera Animation*, Journal of Winter School of Computer Graphics, ISSN 1231-6972, Vol. 14 (2006).
13. W. Reeves, D. Salesin and R. Cook, *Rendering antialiased shadows with depth maps*, ACM SIGGRAPH'87 proceedings (1987).
14. Reuven Y. Rubinstein, *Simulation and the Monte Carlo Method*, (1981).
15. M. Sbert, *The Use of Global Random Directions to Compute Radiosity. Global Monte Carlo Methods*, Ph.D. thesis. Universitat Politècnica de Catalunya, Barcelona (1997).
16. M. Sbert, P. Bekaert and J. Halton, *Reusing paths in radiosity and global illumination*, Proceedings of 4th IMACS Seminar on Monte Carlo Methods, Berlin (Germany) (2004).
17. M. Sbert, F. Castro and J. Halton, *Reuse of paths in light source animation*, Proceedings of CGI'2004, Crete (Greece) (2004).
18. M. Sbert, L. Szecsi and L. Szirmay-Kalos, *Real-time Light Animation*, Computer Graphics Forum (proc. EG'04) (2004).
19. M. Segal, C. Korobkin, R. van Widenfelt, J. Foran and P. Haeberli, *Fast Shadows and Lighting Effects Using Texture Mapping*, ACM SIGGRAPH'92 proceedings (1992).
20. F.X. Sillion and C. Puech, *Radiosity and Global Illumination*. Morgan Kaufmann Pubilishers, Inc., 1994.
21. E. Veach, *Robust Monte Carlo Methods for Light Transport Simulation*, Ph.D. thesis. Stanford University (1997).
22. I. Wald, T. Kollig, C. Benthin, A. Keller and P. Slussalek, *Interactive global illumination using fast ray tracing*, Rendering Techniques'02 (2002).
23. L. Williams, *Casting Curved Shadows on Curved Surfaces*, ACM SIGGRAPH'78 proceedings (1978).

Received

Author information

Francesc Castro, Institut d'Informàtica i Aplicacions. Universitat de Girona. Campus Montilivi. Edifici P4. 17071 Girona., Spain.
Email: castro@ima.udg.edu

Gustavo Patow, Institut d'Informàtica i Aplicacions. Universitat de Girona. Campus Montilivi. Edifici P4. 17071 Girona., Spain.
Email: dagush@ima.udg.edu

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

John H. Halton, University of North Carolina at Chapel Hill., USA.
Email: halton@cs.unc.edu

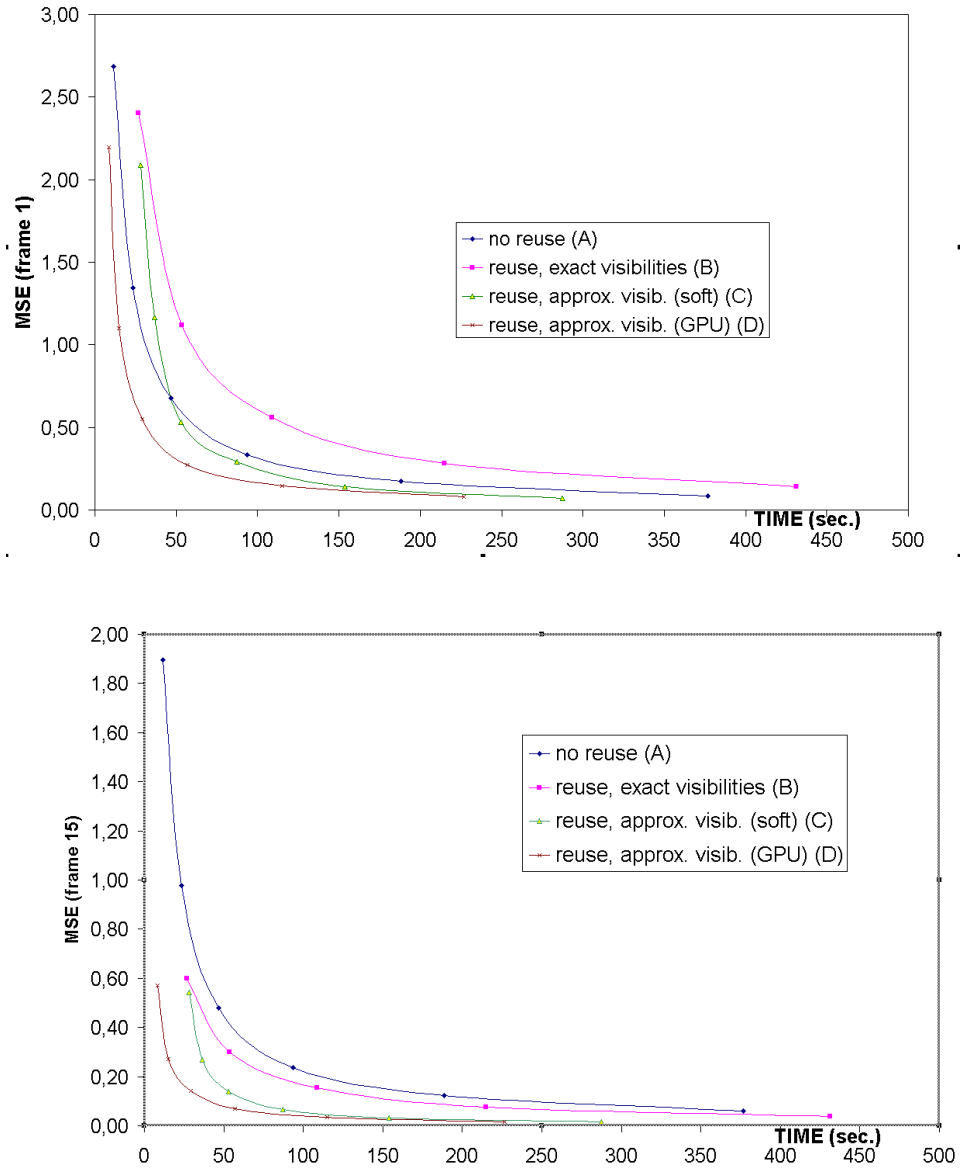

Figure 9. *Experiment 1. (top) Time vs MSE for the first frame in the animation. (bottom) Time vs MSE for the* 15*-th frame in the animation.*

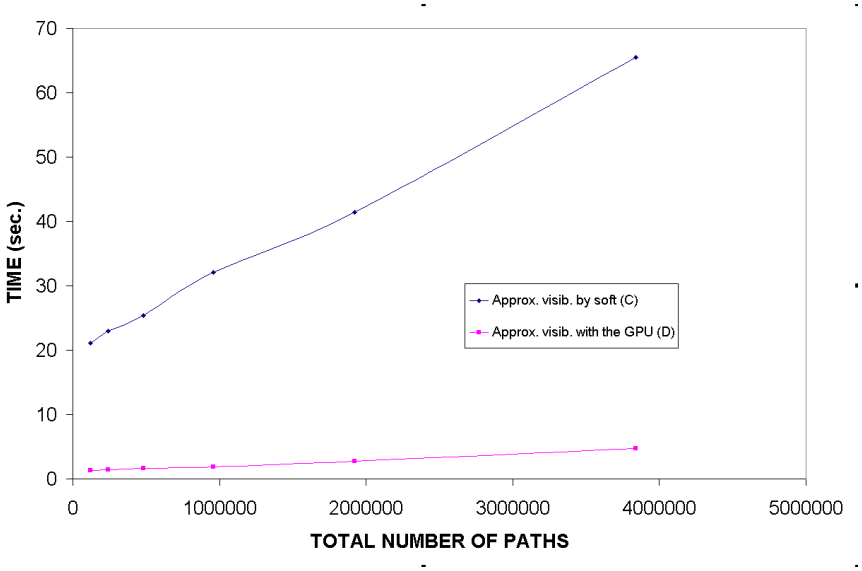

Figure 10. *Experiment 1. Time for visibility maps construction plus form factor computation for approaches C (in the CPU) and D (in the GPU).*

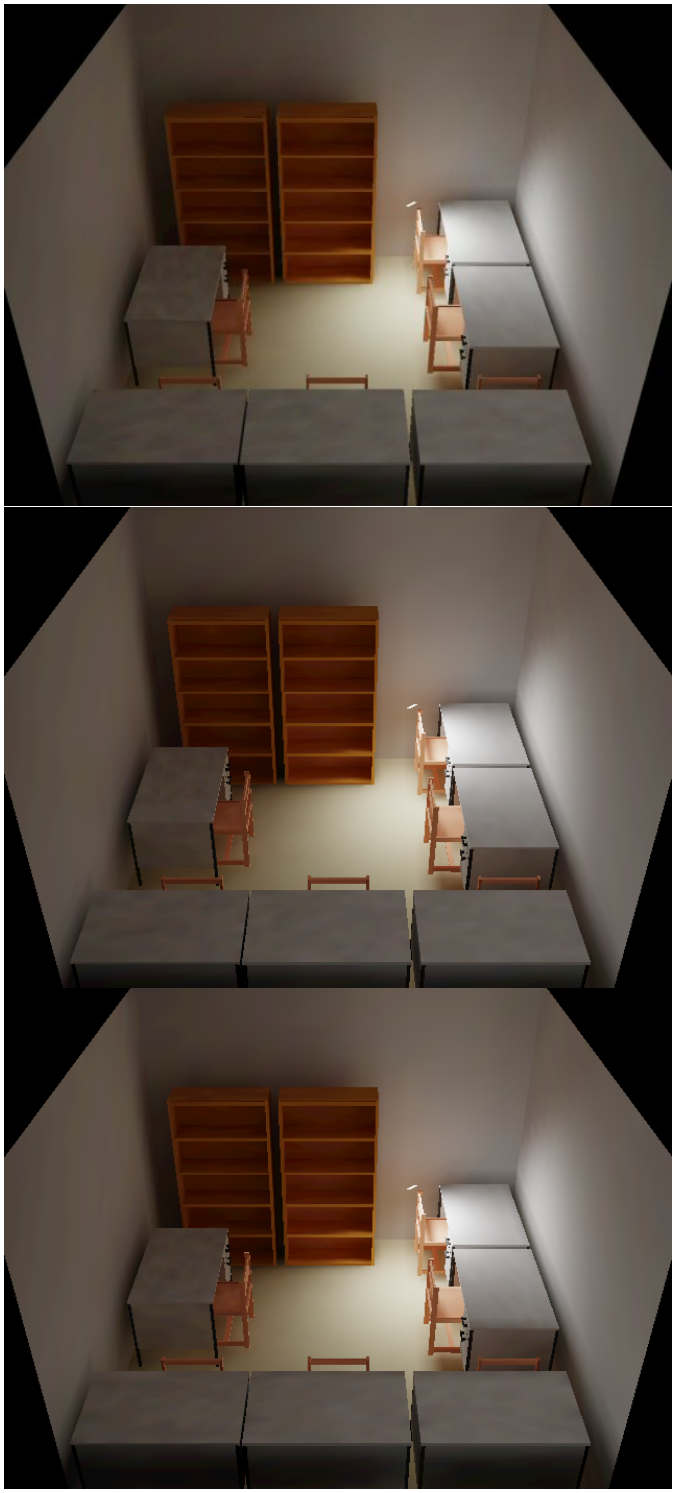

Figure 11. *Experiment* 1. *(left)* 15*-th frame (out of* 30*) obtained with reuse of paths and exact visibility computation. Total time for the full animation* 1354 *sec. (center)* 15*-th frame (out of* 30*) obtained with reuse and visibilities approximated using the CPU. Total time for the full animation* 438 *sec. (right)* 15*-th frame (out of* 30*) obtained with reuse and visibilities approximated with the GPU. Total time for the full animation* 346 *sec.* 200000 *paths have been generated at each position for the* 3 *animation sequences, resulting in a total of* $6M$ *of effective paths per frame, thanks to the reuse.*

Figure 12. *Experiment* 2. *(top)* 20*-th frame (out of* 30*) obtained with reuse of paths and exact visibility computation. Total time for the full animation* 1092 *sec. (center)* 20*-th frame (out of* 30*) obtained with reuse and visibilities approximated using the CPU. Total time for the full animation* 331 *sec. (bottom)* 20*-th frame (out of* 30*) obtained with reuse and visibilities approximated with the GPU. Total time for the full animation* 245 *sec.* 200000 *paths have been generated at each position for the* 3 *animation sequences, resulting in a total of* $6M$ *of effective paths per frame, thanks to the reuse.*