UNITÉ DE RECHERCHE
INRIA-RENNES

Institut National
de Recherche
en Informatique
et en Automatique

Domaine de Voluceau
Rocquencourt
B.P.105
78153 Le Chesnay Cedex
France
Tél.:(1) 39 63 55 11

Rapports de Recherche

N° 1563

Programme 4
Robotique, Image et Vision

USING COHERENCE TO ACCELERATE RADIOSITY

Pierre TELLIER
Eric MAISEL
Kadi BOUATOUCH
Eric LANGUÉNOU

Décembre 1991

RR_1563

IRISA

INSTITUT DE RECHERCHE EN INFORMATIQUE
ET SYSTEMES ALEATOIRES

Campus Universitaire de Beaulieu
35042 - RENNES CEDEX FRANCE
Tél. : 99 84 71 00 - Telex : UNIRISA 950 473 F
Télécopie : 99 38 38 32

Using Coherence to Accelerate Radiosity

Utilisation de la Notion de Cohérence pour Accélérer la Méthode de Radiosité

Pierre Tellier, Eric Maisel, Kadi Bouatouch, Eric Languénou

IRISA
Campus de Beaulieu
35042 Rennes Cedex
FRANCE
tellier@irisa.fr

Publication Interne n°616 - Novembre 1991, 16 pages, Programme 4

Abstract

This paper shows how the temporal coherence can be exploited to accelerate the form factors calculation for the radiosity method. A certain number of form factors are calculated by a new technique combining hemisphere and ray tracing, while the other form factors are estimated by using a motion information. This estimation method has reduced the form factor calculation by a factor greater than three, for some test scenes.

Résumé

Ce papier montre comment la cohérence temporelle peut être exploitée afin d'accélérer le calcul des facteurs de forme dans la méthode de radiosité. Un certain nombre de facteurs de forme sont calculés en utilisant une technique nouvelle qui combine hémisphère et lancer de rayon, alors que les autres facteurs de forme sont estimés à l'aide d'informations de mouvement. Les tests montrent que cette méthode d'estimation permet de diviser le temps de calcul des facteurs de forme par un facteur supérieur à trois.

1 Introduction

The production of realistic computer generated pictures requires the use of global illumination models, to accurately simulate the light interactions within a 3D scene. In the case of scenes made up of perfectly diffuse surfaces, the relation between these surfaces depends on purely geometrical terms, called *form factors* [8].

To efficiently compute these form factors, some methods have been developed, among which the hemicube technique [5], the plane [11] and the hemisphere [12, 10]. Actually, these methods only evaluate point-to-surface form factors. The surface-to-surface form factors are simply assumed to be equal to these point-to-surface form factors. Generally, the point is the center of a surface.

All these algorithms are based on the same principle. They allow the computation, of all the point-to-surface form factors from one point to all the surfaces of the scene. The computation scheme of the form factors related to a point consists of the following steps. First, a projection surface (hemicube, plane or hemisphere) is associated with the point. Then this surface is discretized into *pixels*, assuming that the form factors from the point to each pixel have been precomputed. These form factors are called *delta form factors*. In a second step, the scene is projected on the discrete surface. After this step, each pixel contains a pointer to the surface seen through it. Finally, the discrete surface is scanned, and the delta form factors related to the pixels through which is seen a surface are summed so as to determine the form factor associated with this surface.

These form factors allow to obtain a system of *light energy balance equations*, which describes the light interactions between surfaces. The diffuse radiance of each surface will be obtained by solving this system. To build this system, the form factor computation algorithm must be successively applied at all the points (centers of surfaces) of the scene.

The visibility computation can be performed in two ways: either by a z-buffer algorithm or by ray tracing, independently of the type of the projection surface.

Since the radiosity algorithms spend most of time in performing form factors computations, we will focus only on these computations. This observation led us to propose two ways to reduce this cost.

The first one consists in applying a new form factor computation algorithm which is based on an efficient ray tracing, takes advantage of several kinds of spatial coherencies (patch textures, voxels adaptive grid), and uses an hemisphere as a projection surface [10]. The result of this projection will be called, from now on, *hemisphere value*.

The other strategy consists in reducing the number of projections. Indeed, instead of completely projecting the scene onto two hemispheres located at the centers of two adjacent surfaces, one of the two hemispheres can be estimated from the other by using a motion information. These two centers can be considered as the position of an animated point at two consecutive instants, the velocity of which is calculated.

This paper is organized as follows. After an introduction, our new form factor calculation method is described. Then, the used form factor estimation technique is given, and the implementation aspects are shown. Finally, experimental results are discussed, followed by a conclusion.

2 Form Factors Computation

This section deals with a new method of form factor calculation combining hemisphere and ray tracing.

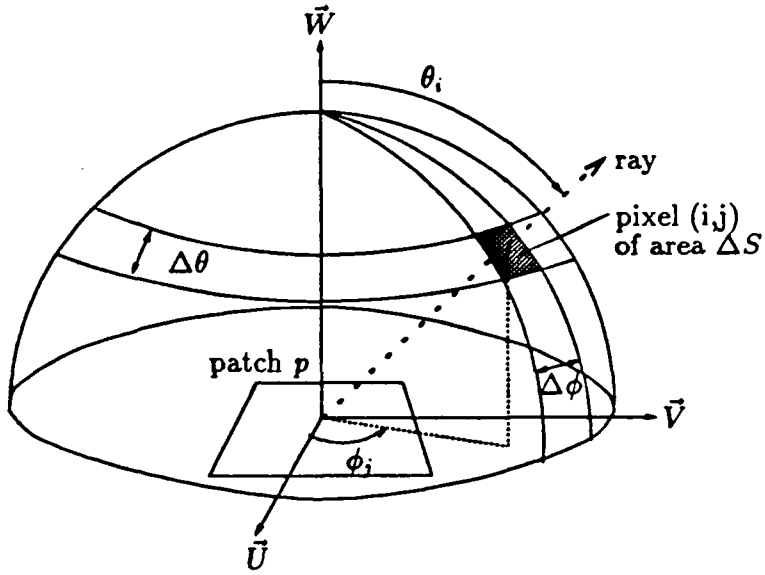

Figure 1: Hemisphere

2.1 The Method

A hemisphere is placed at the center of a patch p and is discretized by uniformly sampling the two polar angles θ and ϕ as shown in figure 1. The hemisphere is then discretized into surface elements ΔS of different solid angles.

To each surface element ΔS on the hemisphere of a patch corresponds a solid angle subtented by ΔS and whose appex is the center of the patch. These solid angles are precomputed and equal to $\sin\theta\Delta\theta\Delta\phi$, the corresponding delta form-factors are given by $(\cos\theta\sin\theta\Delta\theta\Delta\phi)/\pi$. To calculate the form-factors, a ray is cast from the hemisphere center and through each surface element ΔS of this hemisphere, i.e. in directions (θ_i, ϕ_j). Note that each ray corresponds to a delta form-factor. The intersection process between a ray and the scene may result in several points. Only the point closest to the ray origin is considered, and the identifier of the patch containing it is stored in an item buffer. Once all the rays have been cast from the center of a patch p toward all directions (θ_i, ϕ_j), the form-factors from this patch are calculated by scanning the item buffer and summing the delta form-factors associated with the rays along which a particular patch is visible.

This form factor calculation technique is simpler and faster than the hemicube approach, since it avoids several processings such as polygon clipping, polygon filling and geometric transformations.

2.2 Improvements

To speed up the ray-object intersection process implied by the form factor calculation, a spatial subdivision is performed [3]. This subdivision results in a set of $3D$ regions called *voxels*. In addition, this process is still accelerated by the use of another data structure named *patch texture* [10].

Let us now describe this data structure. Usually, when using a radiosity method, the scene is modeled by a set of planar polygons, each of them is discretized into small patches. To avoid calculating the intersection between a ray and all the patches of a same polygon belonging to the current voxel, only the intersection with the relevant polygon is performed. Once the intersection point has been found, a simple test determines the patch containing it. This is made possible thanks to the *patch texture* data structure. Indeed, the scene is not described by a collection of patches, but by a set of polygons. With each

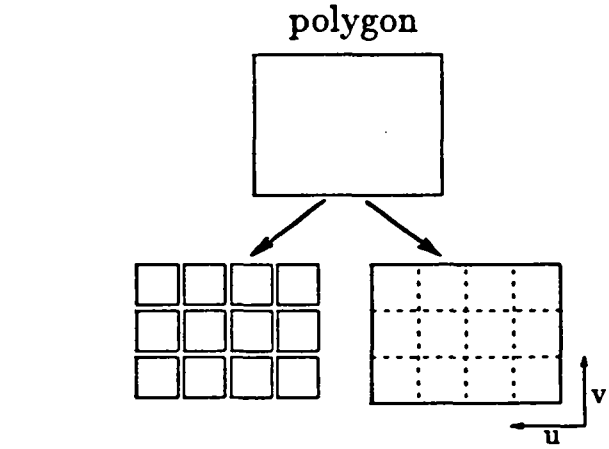

Figure 2: Patch texture data structure

polygon, is associated a $2D$ array (*patch texture*) as shown in figure 2. This array results from uniformly sampling a polygon along two axes corresponding to the parameters u and v. Each element of this array corresponds to a patch. A ray-polygon intersection provides two parameter values (u_I, v_I) of the intersection point I. A simple test determines the element (i, j) of the $2D$ array which corresponds to the patch containing I. Experiments [10] have shown that this strategy reduces drastically the form factor calculation.

2.3 Use of Coherence

Another way to reduce the amount of calculations is to take advantage of temporal coherence. Indeed, a certain number of form factors can be evaluated by a simple estimation, as shown hereafter.

3 Form Factors Estimation

This section shows how certain form factors can be deduced from the exact evaluation of other form factors, by simple estimation based on motion information. This estimation exploits the spatial coherence of the scene. Actually, time is implicitly used, allowing thus to apply techniques that exploit the temporal coherence to estimate the form factors.

3.1 Background

In the field of image synthesis, a certain number of algorithms have been already developed to take advantage of the temporal coherence. This latter can be exploited in, either the image plane (intensity) [1, 4], or the $3D$ scene (positions, forms, displacements, shading) [7], or the light flux domain (ray origin, ray direction, light power, visibility) [9, 2]. Such algorithms fall into three different classes:

1. the fact that certain data structures used by the rendering process (form factor calculation, visibility graph, ...) are not modified over time, is exploited;

2. a boolean information showing the temporal difference between two instances of a same data structure allows to compute again only the portions of the data structures which have changed;

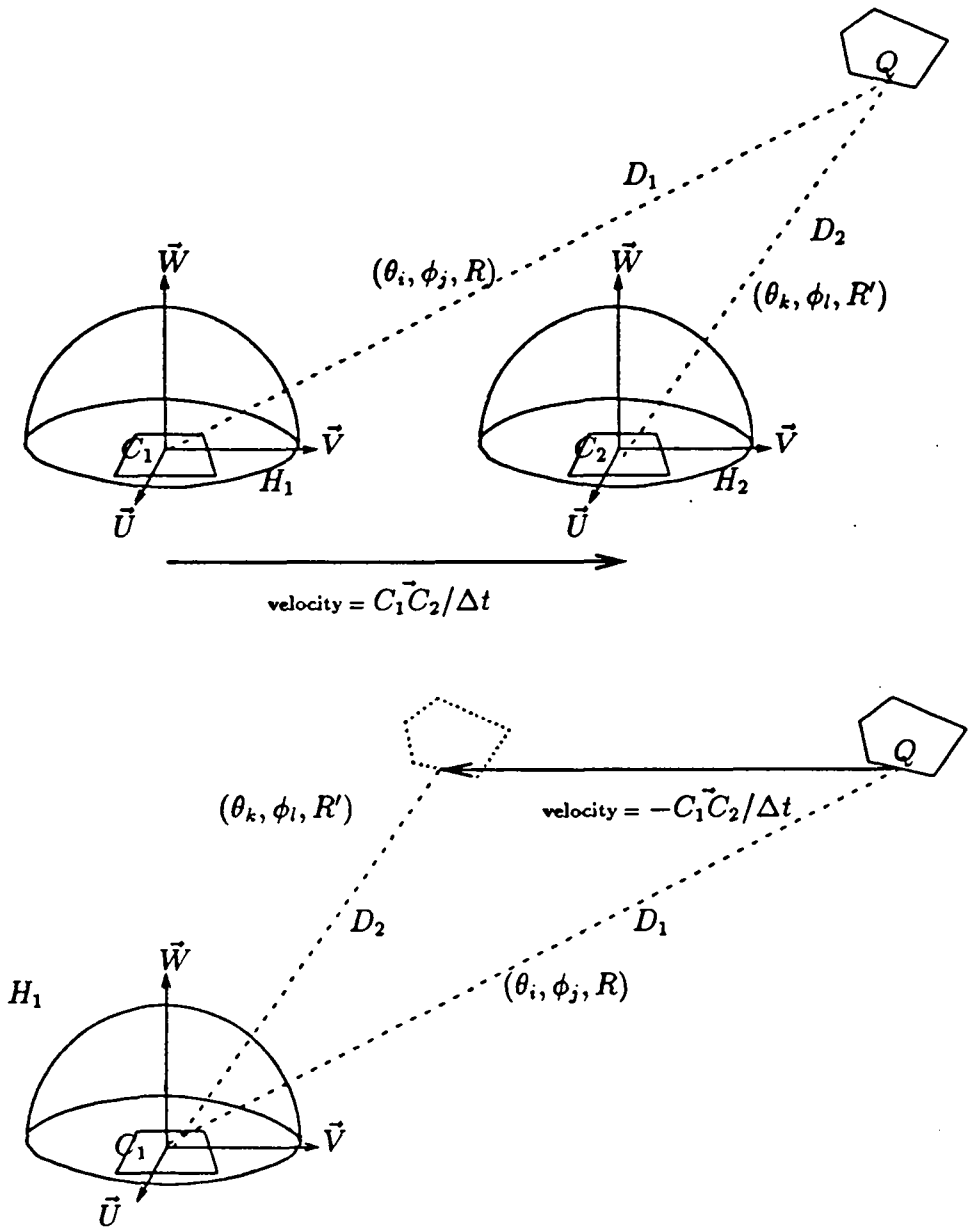

Figure 3: Relative movement

3. a motion information is used to update data structures.

As seen below, our estimation algorithm which exploits the temporal coherence falls into the third class.

3.2 Goal

Let H_1 and H_2 be the hemispheres associated with the two neighbouring patches P_1 and P_2 respectively. Let C_1 and C_2 be the centers of these patches. Assume that H_1 has been evaluated by ray tracing as explained before. The aim is to estimate H_2 from H_1. Moving from C_1 to C_2 is equivalent to apply a motion to H_1 toward H_2. Let D_1 be a ray cast through a pixel (θ_i, ϕ_j) of H_1, and Q the closest intersection point (figure 3). We would like to calculate the position (θ_k, ϕ_l) of Q in the H_2 polar coordinate system. To this end, instead of moving H_1 toward H_2, we consider the relative displacement of the

3.5 Limits of the Method

As our estimation method relies on a certain number of assumptions (small displacement, constant speed), it has some limitations to which we bring solutions.

3.5.1 Proximity

As mentioned earlier, the formulas (1) can only be applied in case of small movement. Actually, the displacement must be small enough so that the movement of objects seems small as seen from any observer which is in our case the center of a patch. Unfortunately, this assumption is not valid for the patches that are close to the center of a hemisphere (i.e. when the patch corresponding to the current hemisphere and these patches are not coplanar). For such patches, their relative movement may be important and cannot be assumed to be uniform. Applying the formulas (1) to these patches would give a wrong destination pixel. Consequently, the patches close to the current hemisphere center are normally projected on this hemisphere without estimation. The used proximity criterion is: the ratio of the distance of a patch to the displacement is under a certain threshold. The algorithm accounting for these close patches is the following:

```
for all the pixels (i,j) of H1 {
   R = H1[i][j].R; id = H1[i][j].patch_id;
   if (R/||C1C2|| > Threshold) {
       /* apply formulas (1) */
   }
   else {
      /* transform the polar coordinates of the patch id */
      /* into the Cartesian coordinate system associated with P1.  */
      u1 = R sin(θi)cos(φj);
      v1 = R sin(θi)sin(φj);
      w1 = R cos(θi);
      /* apply to this patch the displacement */
      /* in the H2 Cartesian coordinate system */
      u2 = u1 + Δu;
      v2 = v1 + Δv;
      w2 = w1 + Δw;
      /* project on hemisphere H2 */
      R' = √(u2² + v2² + w2²);
      θl = arcos(w2/R');
      φk = arctan(v2/u2);
   }
}
```

3.5.2 New Visible Patches

In our algorithm, the motion estimation, is applied only to the set of patches contained in the current hemisphere. Unfortunately, when the observer moves, some polygons (let us recall that polygons are discretized into patches) may become visible (see figure 4). If the movements are small, this situation may happen only near the polygon edges. Our solution to cope with this lack of information consists in detecting on the hemisphere the edges of the projected polygons, and to cast new rays toward these edges. However, this solution appears to be expensive, since it requires an edge detection and the casting of extra rays.

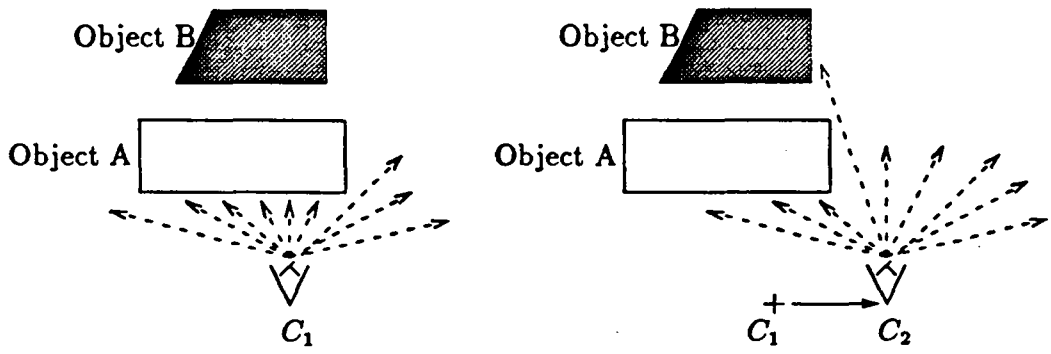

Figure 4: Apparition of new objects

To avoid casting new rays, the knowledge of the hidden polygons that could become visible is needed. This information is obtained by propagating a ray after the first intersection, until the bounding volume of the scene is reached. The list of the intersected patches (as well as their distance to the center of the current hemisphere) is then associated with the pixel corresponding to this ray. This technique will increase the cost of the evaluation of the hemispheres H_1 and H_2. However, the extra cost due to the propagation of rays and list storage can be limited. Indeed, since the displacement is assumed to be small, the problem of missing information arises only near the polygon edges. Consequently the rays are propagated when they hit a patch lying on a polygon boundary (see figure 5). The estimation process becomes then:

```
for all the pixels of the hemisphere
   for all the patches in the list
      estimate their new position and project them
```

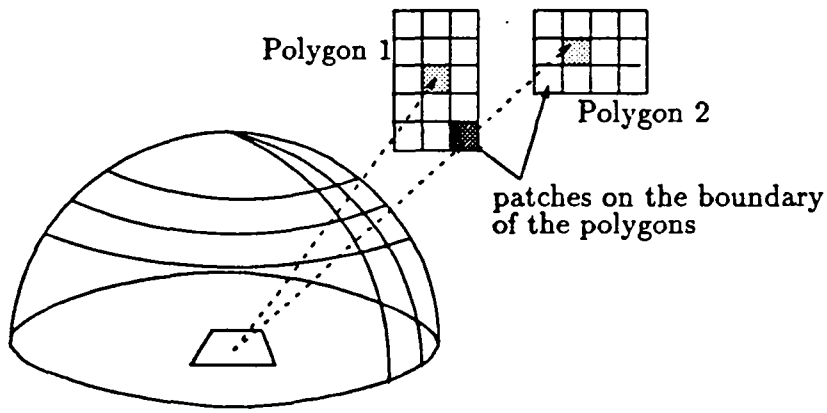

Figure 5: Getting information on hidden objects

3.6 Chosing Neighbouring Patches for Estimation

This is very important to pay attention to the error due to successive estimations. To keep this error as small as possible, while estimating hemispheres as often as possible, we propose the following strategy (see figure 6) allowing an efficient choice of the neighbouring patches for which the hemisphere has to be estimated.

Let P be a polygon discretized into patches. P is scanned by a 3×3 patches window W (see figure 6). The first window is placed at the top-left corner of the polygon. Then, this

window is moved without overlapping between two successive windows. The hemisphere of the patch located at the center of W is exactly computed by ray tracing, while the hemispheres of the other window's patches are estimated. This avoids an important estimation error and greatly exploits the coherence.

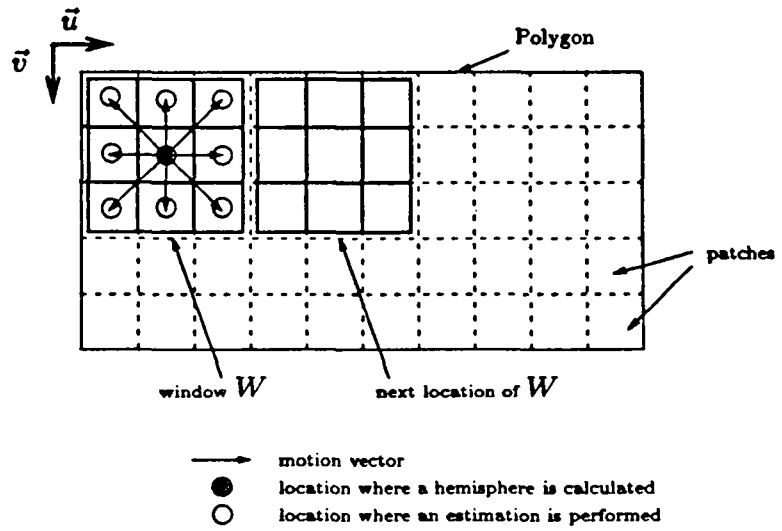

Figure 6: Efficient choice of the neighbouring patches

4 Results

Two test scenes have been considered. The first one is a simple scene, which allowed us to easily evaluate the estimation error, while the second, more complex, shows that our method works.

The first scene is an empty cube whose each face has been subdivided into 10 × 10 patches. The resolution of the hemisphere is 180 × 720 pixels. For a given patch P, we have computed the exact form factors FF and the estimated ones FF_e between this patch and the other ones. The average relative error $|(FF_e - FF)|/FF$, for this scene is less than 5%.

The second scene, named *loft*, is made up of 185 polygons, which corresponds to 14400 patches. The resolution of the hemisphere is 180 × 720 pixels. Figures 8 and 7 are the images obtained with estimation and exact calculation, respectively. The difference between these two images is hardly visible. Figure 9 shows the relative difference between the two previous images. Table 1 shows that the form factor computation time is divided by 3, with our estimation method. This gain factor is greater than 5 for other test scenes. The results given in table 2 show the efficiency of our method.

	with estimation	without estimation
time (500 iterations)	103 min. 55 sec.	313 min. 46 sec.
average time per iteration	12.47 sec.	37.65 sec.

Table 1: Acceleration with test scene *loft*

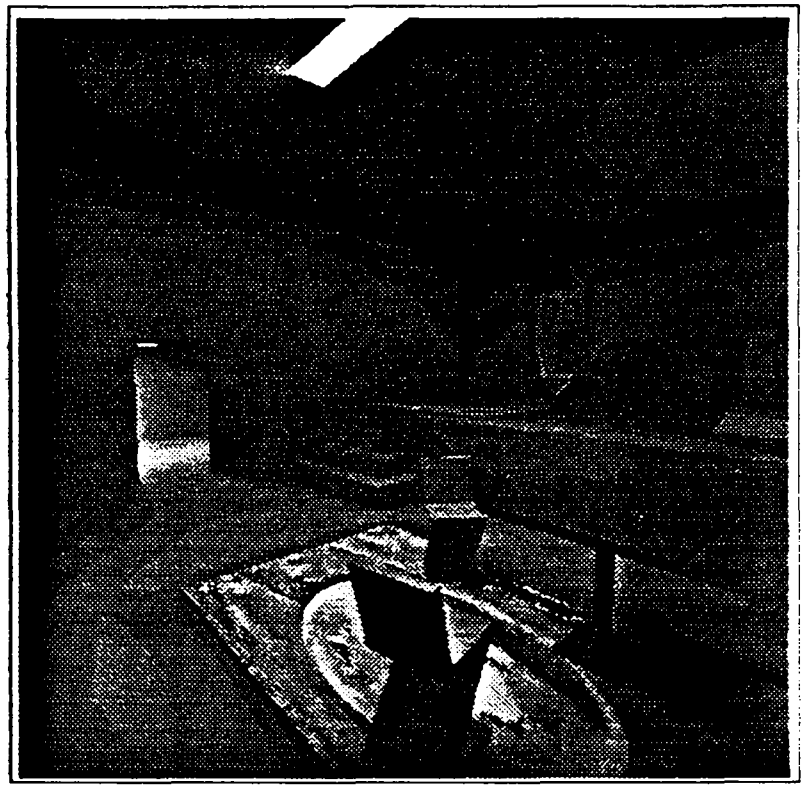

Figure 7: Image obtained without estimation

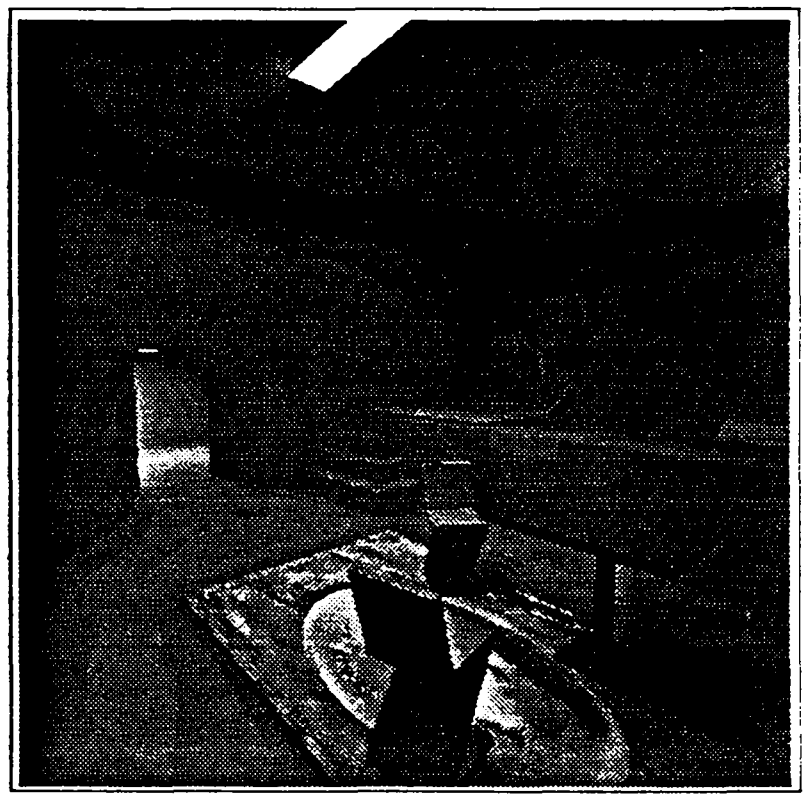

Figure 8: Image obtained with estimation

Figure 9: Relative difference image (white=10%)

5 Conclusion

Our form factor calculation method based on estimation has reduced the computing time, while keeping the same visible quality of images. With this method, the hemisphere resolution as well as the scene discretization can be increased without a substantial extra cost in time. The more refined the scene discretization, the more accurate the estimated form factors. In addition, this estimation method may be used to accurately evaluate the form factors between two patches. Indeed, if a patch is subdivided into elements [6], the form factors FF_{ep} between one element e and another patch p can be evaluated by ray tracing, while the form factors between the other elements and this patch p are estimated from FF_{ep}. Note that our estimation method can be easily applied to any another kind of projection surface such as the hemicube.

number of exact calculations	66
number of estimations	434
average time per exact calculation	37.65 sec.
average time per estimation	5.73 sec.

Table 2: Results on the use of estimation

References

[1] S. Badt. Two algorithms for taking advantage of temporal coherence in ray tracing. *The Visual Computer*, 4(3):123–132, September 1988.

[2] D.R. Baum, J.R. Wallace, M.F. Cohen, and D.P. Greenberg. The back-buffer algorithm: an extension of the radiosity method to dynamic environment. *The Visual Computer*, 2(5):298–306, February 1986.

[3] K. Bouatouch, M.O Madani, T. Priol, and B. Arnaldi. A new algorithm of space tracing using a CSG model. In *EUROGRAPHICS'87 Conference Proceedings*, pages 65–78, Amsterdam, The Netherlands, August 1987.

[4] J. Chapman, T.W. Calvert, and J. Dill. Exploiting temporal coherence in ray tracing. In *Proc. of Graphics Interface'90*, pages 196–204, 1990.

[5] M. Cohen and D. Greenberg. The hemi-cube, a radiosity solution for complex environments. *Computer Graphics*, 19(3):31–40, July 1985.

[6] M.F. Cohen, D.P. Greenberg, D.S. Immel, and P.J. Brock. An efficient radiosity approach for realistic image synthesis. *IEEE Computer Graphics & Applications*, 6(2):26–35, March 1986.

[7] A. Glassner. Spacetime ray tracing for animation. *IEEE Computer Graphics & Applications*, 8(2):60–70, March 1988.

[8] C. M. Goral, D. P. Greenberg K. E. Torrance, and B. Battaile. Modeling the interaction of light between diffuse surfaces. *Computer Graphics*, 18(3):213–222, July 1984.

[9] K. Murakami and K. Hirota. Incremental ray tracing. In *Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 15–29, Rennes, France, June 1990.

[10] E. Languenou, K. Bouatouch, and P. Tellier. *Une Nouvelle Approche Réaliste de Simulation d'Eclairage dans un Environnement Diffus.* Research Report 608, IRISA, Rennes, France, October 1991.

[11] F. Sillion and C. Puech. A general two-pass method integrating specular and diffuse reflection. *Computer Graphics*, 23(3):335–344, July 1989.

[12] Steven Spencer. The hemisphere radiosity method: a tale of two algorithms. In *Eurographics Workshop on Photosimulation, Realism and Physics in Computer Graphics*, pages 127–135, Rennes, France, June 1990.

LISTE DES DERNIERES PUBLICATIONS INTERNES IRISA

PI 607 ABOUT LOGICAL CLOCKS FOR DISTRIBUTED SYSTEMS
Michel RAYNAL
Octobre 1991, 16 pages.

PI 608 UNE NOUVELLE APPROCHE REALISTE DE SIMULATION D'ECLAIRAGE DANS UN ENVIRONNEMENT DIFFUS
Eric LANGUENOU, Kadi BOUATOUCH, Pierre TELLIER
Octobre 1991, 64 pages.

PI 609 INTEGRATION D'UN CORRECTEUR ORTHOGRAPHIQUE DANS L'EDITEUR STRUCTURE GRIF
Patrice FRISON, Eric PICHERAL, Hélène RICHY
Octobre 1991, 22 pages.

PI 610 SYNCHRONIZATION AND CONCURRENCY MEASURES FOR DISTRIBUTED COMPUTATIONS
Michel RAYNAL
Octobre 1991, 20 pages.

PI 611 MALI v06 - TUTORIAL AND REFERENCE MANUAL
Olivier RIDOUX
Octobre 1991, 86 pages.

PI 612 SENSITIVITY COMPUTATION IN NETWORK RELIABILITY ANALYSIS
Gerardo RUBINO
Octobre 1991, 38 pages.

PI 613 OPAC : A FLOATING-POINT COPROCESSOR DEDICATED TO COMPUTE-BOUND KERNELS
André SEZNEC, Karl COURTEL
Octobre 1991, 28 pages.

PI 614 CONTROLLING AND SEQUENCING AN HEAVILY PIPELINED FLOATING-POINT OPERATOR
André SEZNEC, Karl COURTEL
Octobre 1991, 28 pages.

PI 615 ON FAULT-TOLERANT SYMBOLIC COMPUTATIONS
Bernard DELYON, Oded MALER
Novembre 1991, 18 pages.

PI 616 USING COHERENCE TO ACCELERATE RADIOSITY
Pierre TELLIER, Eric MAISEL, Kadi BOUATOUCH, Eric LANGUENOU
Novembre 1991, 16 pages.

ISSN 0249-6399