Real Time Global Illumination for Quasi-Static Scenes

Rubén J. García Hernández Carlos Ureña Almagro
Miguel Lastra Leidinger

Departamento de Lenguajes y Sistemas Informáticos (Software Engineering)
University of Granada

September 18, 2003

Contents

1 Introduction and State of the Art **9**

1.1 Introduction 9
1.1.1 Uses of Global Illumination 9
1.1.2 Goal 9
1.1.3 Description of the article 10
1.2 Density Estimation Methods 10
1.2.1 Optimizations of Density Estimation on the Tangent Plane 11
1.2.2 Remarks 13
1.3 Interactive Global Illumination 13
1.3.1 Finite Element Methods 13
1.3.2 Real Time Ray Tracing 13
1.3.3 Interactive Global Illumination using Selective Photon Tracing (Dmitriev) 14

2 Real Time Optimizations **15**

2.1 Algorithm overview 15
2.2 Simplifications 17
2.2.1 Model Geometry 17
2.2.2 Scene change 17
2.3 Computing and updating the ray set 17
2.4 Computing and updating radiosity 18
2.4.1 Calculation of the first frame 18
2.4.2 Updating radiosity in static vertexes 18
2.5 Computing and updating radiosity on dynamic vertexes 21
2.5.1 Impact Count 21
2.5.2 Photon Maps and Density Estimation on the Tangent Plane 21
2.5.3 Comparison between Impact Count and DETP 22
2.6 Reuse of Point ordering 23
2.7 Disc Indexing 24
2.8 Study of Pre-intersection tests 26
2.9 Comparison between Photon Maps and DETP 26
2.9.1 Error and bias analysis of PM and DETP 27
2.9.2 Performance Tables 28
2.10 Glossy Surfaces 30

2.10.1 QuadTree 31

3 Implemented System **39**
3.1 Goals 39
3.2 Further Characteristics 39
3.2.1 Supported Surface Types 39
3.3 Supported Estimation Methods 40
3.4 Classic optimizations present in our system 40
3.4.1 General Purpose Optimizations 41
3.4.2 Radiosity Specific Optimizations 41

4 Future Work **43**
4.1 Hardware accelerated direct illumination 43
4.2 Transparent surfaces 43
4.3 Ray Tracer 43

List of Figures

1.1 Performance of Photon Maps and DETP for 50 000 photons . . . 11

2.1 Pseudocode for the Initial Photosimulation Calculation 18
2.2 Pseudocode for the Photosimulation Recalculus 19
2.3 Pseudocode for the Initial Radiosity Calculation 20
2.4 Pseudocode for the Recalculation of Radiosity for the Impact Count method 22
2.5 Density Estimation on the Tangent Plane, 50 000 photons 23
2.6 Density Estimation using Impact Count, 1 000 000 photons . . . 24
2.7 Second Tree Scene, 20 000 photons 28
2.8 Flat Shaded Render of the Tree Scene 30
2.9 Photon Maps Bias: Detail of a border vertex 31
2.10 Measurement of the border angle 32
2.11 Density Estimation on the Tangent Plane, 1 000 photons 33
2.12 Density Estimation on the Tangent Plane, 10 000 photons 34
2.13 Photon Maps Density Estimation, 10 000 photons 35
2.14 Photon Maps Density Estimation, 50 000 photons 36
2.15 Photon Maps Density Estimation, 200 000 photons 37

List of Tables

2.1 Comparison between DETPR and Impact Count 22
2.2 Recalculation time for 10 000 photons, tree scene 25
2.3 Radiosity recalculation time for 20 000 photons for the Tree Scene 26
2.4 Radiosity recalculation time for 20 000 photons for the Second Tree Scene 27
2.5 Times to simulate the Tree Scene using different preintersection tests 29
2.6 DETP, 1 000 photons 31
2.7 DETP, 10 000 photons 32
2.8 Photon Maps 33

Chapter 1

Introduction and State of the Art

1.1 Introduction

1.1.1 Uses of Global Illumination

Global illumination techniques provide computer generated images with realistic appearence. This is useful in many applications, such as building design, interior design, light design (both for lamps and for car headlights) and movies, for example. Interactive global illumination provides these applications with a more comfortable interface.

Other applications, such as video games, do not require interactive global illumination. Nevertheless, it is a very important bonus in their goal towards realism.

1.1.2 Goal

Our goal is solving the problem of calculating global illumination in a quasi-static scene in real time. We define a quasi-static scene as the combination of a static scenario and a dynamic object, when the complexity of the dynamic object is much smaller than the complexity of the scenario. Two facts must be taken into account:

- We are creating an animation. Each Photogram is seen during approximately 40 milliseconds (at 25 fps, which is the refresh rate of television in Europe). Therefore the exact solution is not needed; an approximation would be enough, even in the hypotetic case of no real time requirements.

- We wish for interactive refresh rates. This means we only have 40 ms worth of computing power per frame, so some simplifications will have to be made in order to fit performance requirements.

These two facts lead us to an approximate calculation of illumination.

This is the most usual scene type, especially in videogames, which are the most common interactive graphical applications.

1.1.3 Description of the article

This document describes the new algorithms created in the Global Illumination Subgroup of the Computer Graphics Research Group for the development of a Real-Time version of the Density Estimation on the Tangent Plane algorithm. The program developed using these improvements is also described.

Firstly, (section 1.2) a quick survey of different density estimation methods for radiosity is made. Then published algorithms for Real-Time global illumination are commented.

In the following chapter, the reasons for our new algorithm are shown. In chapter 2.1 the Real Time algorithms and techniques are explained. These algorithms include improvements in photon tracing, density estimation, and non diffuse surfaces. Some timing results are included, showing the increased performance of new algorithms, and the interactivity of the results. Some more improvements are needed to reach Real-Time performance, though.

Later on in chapter 3 our system is described, including a description of surface types supported (such as diffuse or glossy) and classic optimizations (such as Space Partitioning) present in the system. Finally a description of our future work is mentioned.

1.2 Density Estimation Methods

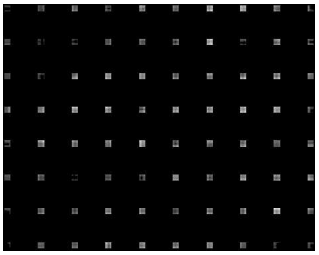
In order to obtain a radiosity value in a point, we have to integrate the incident radiance for all the directions in that point. Since we are using numeric approximation, the most straightforward way to obtain an estimate of the integral are the so called Density Estimation Methods.

Different estimation methods have been described in Literature. The following methods are relevant to our paper.

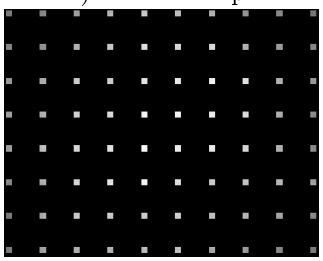
The most basic method is Impact Count, described by Patanaik in [12] and Arvo in [2]. It is based on the particle model of light. Photons are traced from the light sources and interact with the surfaces until eventually they are absorbed or leave the scene. The interaction between photons and surfaces are stored in order to reconstruct radiosity on surfaces. In order to obtain the radiosity on a surface, the energy of the impacts on the faces are added. Then, for each vertex, a mean value is computed taking into accout the faces it belongs to.

Henrik van Jensen, [6] explains a method to calculate the radiosity of a scene, which he calls Photon Maps. A spatial data structure which keeps the photon impacts on surfaces is created, and a fixed number of impacts (the closest to the point where radiosity is being calculated) is used to estimate the radiosity.

There is a problem with this method, namely that if relativelly very small surfaces exist in the scene, these zones tend to appear too dark if the number of photons is not large enough. [8] explains this problem in detail. See figure 1.2 for an example.



a) Photon Maps



b) Tangent Plane estimation

Figure 1.1: Performance of Photon Maps and DETP for 50 000 photons

Miguel Lastra et all, in [7] describes a method to correct this problem storing the rays in the scene and using a fixed size disc centered in the point where radiance is being calculated, and contained in the plane tangent to the surface. Rays intersecting a given disc are used to calculate the radiance in the point the disc is centered on.

The algorithm is called Density Estimation on the Tangent Plane. We will refer to it by its initials DETP in latter sections of the article. Since our work is based on this last method, we will explain it in more detail here.

1.2.1 Optimizations of Density Estimation on the Tangent Plane

Some optimizations have been described which increase performance or reduce the bias of the algorithm in certain situations. This section provides a quick overview.

Artifact Control

The basic DETPR algorithm does not take into account the fact that, when a disc is generated, it may have an unreachable zone. An obvious example is

places near walls and corners in a room. The disc intersects the wall, and no photon can reach the part of the disc behind the wall. The solution is using the reachable area of the circle in the irradiance estimation. Another similar problem appears on concave surfaces. The corresponding solution is taking into account rays which intersect the disc after intersecting the mesh the vertex is in, but before any other intersection with the scene. This algorithm is explained in detail in [8].

This algorithm produces a better solution but requires some processing. In the case of Real-Time requirements, it should be studied whether the increased quality of the solution is worth the extra effort. If both solutions are visually similar, disabling Artifact Control frees some time for other processing tasks, so for example more photons can be shot.

Ray Caché

There exists a huge number of rays and only a small subset intersects a given disc. Therefore, a data structure which allows to query for nearby (and therefore posibly intersecting) rays of a disc is useful. Rays that cannot intersect are discarded. This is useful when vertices are ordered in a way which preserves spatial coherence.

Ray Caché ([8]) is a method which tries to use this idea. It also uses the fact that nearby points in a scene share many intersecting rays (i.e: the rays intersect both points), so some calculations can be reused for different vertices. The coherence is created by means of an optional sorting step. This algorithm increases the performance of the system and therefore is good for our Real Time purposes.

Pre-Intersection Test

Intersecting a ray against a disc is a relatively complex operation. It is useful, in some circumstances, to create a polygon which contains the disc and intersect the ray and the polygon. If the ray does not intersect the polygon, it may be discarded, since it cannot intersect the disc.

There are some algorithms based in this idea. We have implemented the most common ones:

- Badouel
  This ray-triangle intersection method is explained in [3]

- Plücker
  This one is based in using the Plücker coordinates of a ray and a polygon which contains the disc. This intersection method is computationally faster than the geometric standard method [16].

- Möller
  This is the well known Möller-Trumbore method for ray-triangle intersection, described in [10]

Currently, we are using small scenes with a large radius for the discs, which means that an important fraction of the rays intersect the discs. Since these pre-tests are useful when most of the rays do not intersect, these pre-tests are not being very useful. Results indicate increases in performance of around 1% in these circumstances (see section 2.8 for details). Our expectations are larger increases in performance when we use larger scenes.

1.2.2 Remarks

As we will see later, the recalculation procedure for Photon Maps and Density Estimation on the Tangent Plane is more time consuming than that of Impact Count, but the resulting quality is higher (lower noise). See section 2.5 for an analysis.

1.3 Interactive Global Illumination

There are some algorithms to calculate Interactive global illumination. This section cites the characteristics of the most important ones.

1.3.1 Finite Element Methods

These methods were the first ones developed for the calculation of radiance. They divide the scene in a set of patches, and calculate the form factor between pairs of patches. The form factor is the fraction of energy leaving the first patch and arriving at the second patch. Then a system of equations is solved in order to know the radiance in each patch.

Different optimizations have been proposed for this schema. The state of the art in this method is [14], which uses the fact that nearby surfaces have comparatively large form factors. This allows the use of group iterative methods to increase convergence speed. The interactivity is achieved using ideas from Progressive Refinement, i.e., solutions are displayed before convergence is complete.

1.3.2 Real Time Ray Tracing

Wald, in [17] explains a method which applies techniques from Distributed Programming to RayTracing. It uses a 16-node cluster of PCs to obtain global illumination in real time.

He mentions merging results from nearby points, i.e, a blur filter is used when nearby pixels belong to the same object, decreasing noise.

Hand-made assembler optimizations are also mentioned as a source of the incremented performance of their system. Other low level optimizations include optimizing the layout and alignment of data structures, prefetching, cache management, SIMD and changing the algorithms to be able to follow various (typically four) rays at the same time to bump cache performance.

An extension to caustics photon maps (hashing photon impacts) is used to be able to use transparent surfaces. Specular reflections are calculated with a raytracing pass. Quasi Monte-Carlo Integration is used in his system to speed up convergence.

1.3.3 Interactive Global Illumination using Selective Photon Tracing (Dmitriev)

We are now going to focus in algorithms which simulate illumination by shooting photons from the light source.

Dmitriev, in [4] proposes a method which allows for the recalculation of illumination and radiosity along a number of frames, and for the fast finding of the photons which need to be recalculated. This method is based in the multidimensional Halton sequence, which due to its quasi-periodicity, allows for easy finding of photons with a similar path to the one being recalculated. Most of these photons will also have to be recalculated. He uses raytracing from the light sources, so his solution is view-independent.

The hardware requisites for his method are a mid-end computer. He discusses the illumination update procedure, in which he uses only the recalculated rays to update illumination. Graphic hardware is used to calculate shadows, and then only indirect illumination is calculated and added to the resulting image.

Chapter 2

Real Time Optimizations

2.1 Algorithm overview

All the algorithms proposed in the previous section have some drawbacks when applied to rendering of quasi-static scenes in Real-Time. The Density Estimation algorithms present in section 1.2 were designed for the rendering of a static scene. Therefore, they do not reuse information from the previous frame when used to compute an animation. We have developed techniques which allow for the reuse of information from previous frames, therefore increasing performance.

Finite element methods have the problem that the movement of one patch affects the form factor of the patch with all the other patches. This means a whole line and column of coefficients of the system of equations must be recomputed, and the whole system must be resolved again. Taking into account the fact that Progressive Refinement shows non-converged solutions, the net result is that shadows and indirect illumination lag behind current object positions. This is visually much more unrealistic than using, for example, softening of shadows from direct illumination.

On the other hand, our algorithm must recalculate the photons which hit the mobile object, and use the recalculated photons to update illumination. The displayed frame has indirect illumination and shadows corresponding to geometry.

The algorithm Wald proposes has a series of drawbacks:

We are focusing in using a standard PC for Real-Time global illumination, since this is the most common platform. His distributed algorithm is not suited for our chosen platform.

Our use of an independent-of-view solution, justified since it makes changes in camera position less expensive, means we cannot make use of interpolation in nearby pixels. We use a similar idea: estimate radiance on vertices and interpolate inside the polygons.

No low-level optimizations were added for our program. State of the art compilers are used for the optimization. Regarding the low level optimizations Wald mentions, only the last one (following batches of rays concurrently) is not implemented by compilers, since it requires a change to the interface of the ray-primitive intersection functions. However, it can be seen as a loose form of loop unrolling, and some compilers can create a similar algorithm when the intersection function is inlined. We are confident this form of optimization will be inserted into compilers in the near future.

Even though interactive refresh rates can be obtained by Wald while objects are moving, the correct illumination takes two to three seconds to compute after objects stop moving. The problem with this approach is that shadows are calculated from information from previous frames, and object positions have changed. This makes the shadow position lag behind their correct position. A couple of seconds after the object has settled, the shadows gain their correct position. The problem is analogous to the one present in Finite Element Methods. This divergence comes from the fact that Wald is using his algorithm for the preview of global illumination in scenes, so waiting a few seconts for illumination to settle is not an issue. On the other hand, we are focusing on obtaining a realistic approach to global illumination in all frames, which is needed for example for 3D games.

We have not included support for transparent surfaces due to the small number of scenes which have them. Future plans in our work include extending our support for glossy surfaces to take into account transparencies, though.

Quasi Monte-Carlo is not being used, but it would integrate easily in our system. We have found that radiosity density estimation time is much higher than the raytracing phase, and are focusing on optimizing the Density Estimation part.

The multidimensional Halton sequence mentioned in Dmitriev's paper would integrate easily in our system, but we are not using it. As was mentioned above, we are firstly optimizing the Density Estimation phase.

Dmitriev's use of hardware to calculate shadows could be used in our system in order to decrease the number of photons, but is not yet being used. Future plans include implementing this technique.

For our density estimation methods, using only recalculated rays as Dmitriev does is not enough. The result is incorrect for moving objects, which is fine by Dmitriev because photons are removed when they reach a certain age, while we keep all photons which have a valid path. Our algorithm is described in the following sections.

The Density Estimation methods were designed for the rendering of a still image. However, when rendering an animation in real time, most of the information from the previous frame can be used to speed up the calculation of the current frame.

In order to obtain a Real-Time algorithm, we have restricted ourselves to some simplifications:

2.2 Simplifications

2.2.1 Model Geometry

The most widespread way to model a scene is a set of triangle meshes. Most of the Graphic Design utilities can work with these meshes. Other utilities work with implicit surfaces, but usually export them as triangle meshes.

Therefore, we are going to restrict ourselves to triangle meshes. The scene is made of a set of triangles. Our algorithm computes radiosity the vertexes and interpolates inside the triangles.

2.2.2 Scene change

Animations are usually made of a static background and a moving character. This is most clear in videogames. Another example are films.

Most of the vertexes in the scene are static. A minority of the vertexes move. We have separated thus the scene in two parts, the 'static' scene and the 'mobile', which can move and rotate with respect to the static scene. Conceptually, our algorithm can take into account a set of moving characters, since it only checks the previous and actual position of the triangles the moving objects are made of.

Taking into account mobile light sources introduces many difficulties in the treatment of recalculation of radiosity. Solving these difficulties requieres more processing power. We are going to restrict ourselves to static light sources, since we are trying to obtain a Real-Time algorithm. These scenes have wide applications, such as interior design, so our decision is justified.

2.3 Computing and updating the ray set

The algorithm's first step is the generation of the rays and storage of the intersection points of each ray with the scene. The pseudocode can be seen in figure 2.1

In latter frames, the ray set is updated like this: We know which triangle intersected the ray in the previous frame. If this triangle belongs to the mobile object, the ray is recalculated, taking into account the new position of the mobile object. If the triangle does not belong to the mobile object (or if there was no hit at all in the previous frame) the ray is tested against the mobile object's current position. If it hits, its reflections are recalculated. Otherwise, the ray is still valid, but the procedure should be repeated for its reflections. Note that for most rays, no intersection test against the scene is needed, which is where most of the complexity lies.

In order to implement this approach efficiently, a two-level ray list is used: The first level contains the primary rays; these rays allow access to a simply linked list of rays produced by subsequent reflections of themselves. Recalculating a ray means storing a copy of the ray (drawing along a reference to the rest of the reflections of this ray) in a helper list, referred to as "Old Rays" from now

on, and rechecking the original ray against the scene and mobile object. This will generate a list of new reflections, which are stored where the reference to the old reflections was. They also have to be stored in another helper list, "New Rays" With this approach, we have three ray lists: Old Rays, New Rays and Ray List. These lists are later used to calculate radiosity values in the vertexes. The pseudocode of the algorithm can be seen in figure 2.2

Algorithm 2.3.1: INITIALPHOTOSIMULATIONCALCULATION (*Photons*)

```
Let numLights be the number of lights of a scene.
Let Photons be the number of photons to be simulated.
Calculate number of photons for each light,
according to the percentage of energy emited from that light.
for Light ← 1 to numLights
   for i ← 1 to number of photons for Light
      Ray ← Lights[Light].GETPHOTON()
      Insert Ray in RayList
      while Ray is not absorbed
         IntersectedTriangle ←First intersection of Ray
          against the scene
         Ray ← IntersectedTriangle.REFLECT(Ray)
```

Figure 2.1: Pseudocode for the Initial Photosimulation Calculation

2.4 Computing and updating radiosity

2.4.1 Calculation of the first frame

The first time the radiosity algorithm is run, all the rays in the scene must be taken into account. The procedure is shown in figure 2.3. This algorithm is expressed in a very high abstraction level, because the specific algorithm (Impact Count, Photon Maps or Density Estimation on the Tangent Plane) is passed as an argument to the program.

2.4.2 Updating radiosity in static vertexes

On the next run of the algorithm, the mobile object is located in a new position. The rays which intersected the object in the previous frame have been recalculated in the Photosimulation phase. Obviously, the rays which did not intersect before and now do have also been recalculated. However, most of the rays have the same contribution as before to the radiosity on a given point.

Algorithm 2.3.2: FOLLOWRAY (*Ray*)

while Ray is not absorbed
  Intersect *Ray* against the scene
  Ray.next ←next reflection of *Ray*
  Ray ← *Ray.next*

Algorithm 2.3.3: PHOTOSIMULATIONRECALCULUS ()

OldRayList ← *EmptyList*
NewRayList ← *EmptyList*
for $i \leftarrow 1$ **to** *Photons*
  Ray ← *RayList*[i]
  while Ray is not absorbed
    if *Ray* intersected the mobile object **or**
    Ray intersects now the mobile object
      Insert *Ray* in *OldRayList*
      FOLLOWRAY(*Ray*)
      Insert *Ray* in *NewRayList*
      break
    Ray ←next reflection of *Ray*

Figure 2.2: Pseudocode for the Photosimulation Recalculus

It is therefore interesting to try and conserve the old valid radiosity information.

This is the expression of the radiance leaving a point x in direction ω_0:

$$L_r(x,\omega_0) = \int_\Omega f_r(x,\omega_0,\omega) L_i(x,\omega) cos(\theta) d\omega \tag{2.1}$$

For diffuse surfaces, it can be simplified into:

$$L_r(x,\omega_0) = \frac{\rho(x)}{\pi} \int_\Omega L_i(x,\omega) cos(\theta) d\omega = \frac{\rho(x)}{\pi} E(x) \tag{2.2}$$

where $\rho(x)$ is the reflectivity at x and $E(x)$ the irradiance at x.

When we have a dynamic scene, the time has to be taken into account: Let $E(x,t)$ be the irradiance at x at instant t. Let $\tilde{E}(x;S)$ be the irradiance at x due to the set of rays S. Let S^t be the set of rays at instant t.

$$\tilde{E}(x,t) = \tilde{E}(x;S^t) \tag{2.3}$$

is an approximation of $E(x,t)$

Algorithm 2.4.1: INITIALRADIOSITYCALCULATION (DensityEstimatonAlgorithmDE)

for each $Vertex$ in the scene
  $DE.CalculateRadiosity(Vertex, RayList)$

Figure 2.3: Pseudocode for the Initial Radiosity Calculation

Then,

$$E(x,t) \approx \tilde{E}(x,t-1) + \tilde{E}(x;N^t) - \tilde{E}(x;O^t) \tag{2.4}$$

where

$$N^t = S^t - S^{t-1} \tag{2.5}$$

(new Rays) and

$$O^t = S^{t-1} - S^t \tag{2.6}$$

(old Rays).

This formula can be extracted from the following reasoning:

Let I^t be the invariant indices in instant t:

$$I^t = S^t \cap S^{t-1} \tag{2.7}$$

Since the contribution of each ray to radiosity is additive,

$$\tilde{E}(x;S^t) = \sum_{i \in S^t} \tilde{E}(x;\{i\}) \tag{2.8}$$

$$S^t = I^t \cup N^t \tag{2.9}$$

follows from set theory from 2.7 and 2.5

$$S^{t-1} = I^t \cup O^t \tag{2.10}$$

can be extracted in a similar way from 2.7 and 2.6

$$\tilde{E}(x;S^t) = \sum_{i \in I^t} \tilde{E}(x;\{i\}) + \sum_{i \in N^t} \tilde{E}(x;\{i\}) \tag{2.11}$$

from 2.8 and 2.9

$$\tilde{E}(x;S^{t-1}) = \sum_{i \in I^t} \tilde{E}(x;\{i\}) + \sum_{i \in O^t} \tilde{E}(x;\{i\}) \tag{2.12}$$

from 2.8 and 2.10

$$\tilde{E}(x;S^t) - \tilde{E}(x;S^{t-1}) = \sum_{i \in N^t} \tilde{E}(x;\{i\}) - \sum_{i \in O^t} \tilde{E}(x;\{i\}) \tag{2.13}$$

(substraction of 2.11 and 2.12)

$$\tilde{E}(x;S^t) = \sum_{i \in N^t} \tilde{E}(x;\{i\}) - \sum_{i \in O^t} \tilde{E}(x;\{i\}) + \tilde{E}(x;S^{t-1}) \tag{2.14}$$

(from 2.13, moving $E(x;S^{t-1})$ to the right hand side)

$$\tilde{E}(x;S^t) = \tilde{E}(x;N^t) - \tilde{E}(x;O^t) + \tilde{E}(x;S^{t-1}) \tag{2.15}$$

(Substituting 2.8 in 2.14). This equation is 2.4 with the right hand side terms in a different order.

In order to recalculate the radiosity, we run the density estimation algorithm with the old ray set generated in the previous step. The calculated radiance is substracted to the previous value in each vertex. Then the algorithm is run again with the new ray set, and the calculated radiance estimate is added to the value in the vertex. The result is the identical to calculating the radiance estimate on the whole ray set, but much faster. Note that this calculation does not require the use of the whole ray set.

2.5 Computing and updating radiosity on dynamic vertexes

The algorithm for dynamic vertexes depends on the density estimation method. We will first discuss the algorithm for Impact Count. Then we will discuss the algorithm for Photon Maps and Density Estimation on the Tangent Plane.

2.5.1 Impact Count

In order to update the radiosity using Impact Count, we need to know which triangles have received new impacts or no longer have the impacts they had. Fortunately, the Photon tracking phase calculates this information, so we can remove the contribution of the rays which do not hit the triangles any longer and add the new impacts.

The radiosity on a vertex is finally calculated as an average of the radiosity of the triangles it belongs to. Therefore, the algorithm for recalculation of the mobile objects with Impact Count is identical to the recalculation of the static scene. The pseudocode can be seen in figure 2.4

2.5.2 Photon Maps and Density Estimation on the Tangent Plane

Since the update of radiosity using these methods depend on impact information on non-local points (the discs in DETP are unknown to the Photon Tracking Phase; and the vertices which are affected in Photon Maps are unknown unless a specific search is made) the algorithm used in Impact Count cannot be used for these methods.

Algorithm 2.5.1: RadiosityRecalculationForImpactCount ()

```
for each Ray in OldRayList
   TriangleIntersected = Ray.GetHit()
   TriangleIntersected.Substract(Ray)
for each Ray in NewRayList
   TriangleIntersected = Ray.GetHit()
   TriangleIntersected.Add(Ray)
```

Figure 2.4: Pseudocode for the Recalculation of Radiosity for the Impact Count method

To calculate the radiosity estimate in the points of the mobile object, we run the initial algorithm, checking all the rays in the scene. Refer to Figure 2.3, where the pseudocode for the initial calculation of radiosity is shown. Please note this algorithm is different from the one in [4].

2.5.3 Comparison between Impact Count and DETP

We have seen that the algorithm for Impact Count in intrinsically faster than the one for DETP, since only the recalculated rays have to be taken into account. The problem is that the noise and variance parameters are much higher. It is therefore necessary to increase the number of photons, and the final illumination time for a given scene is higher.

Although it has been noted in the literature that the parameters of noise and variance are much higher in Impact Count than in Photon Maps ([6], page 52) or DETP ([7], section 2), they have studied the time for the whole calculation of irradiance. In this case, however, we are measuring the time of recalculating the whole scene in Photon Maps or DETP, versus recalculating rays which intersect the mobile only in Impact Count. As an example we'll use the complex scene studied in detail in section 2.9.

Algorithm	Fotosimulation	Ilumination	Total	Error
DETPR	0.05	0.80	0.85	16.10 %
ImpactCount	1.09	0.02	1.11	139.54 %

Table 2.1: Comparison between DETPR and Impact Count

We will use the mean luminance difference between a reference image (Density Estimation on the Tangent Plane, radius 0.01, 100 million photons) and the generated images shown below. The figures 2.5 and 2.6 can be compared to see the improved quality of DETP. Table 2.1 shows the time in seconds for the recalculation of the rays (Fotosimulation), the density estimation (Illumination), and the total time. Finally the error is shown. It can be seen that while