Real Time Global Illumination for Quasi-Static Scenes

Rubén J. García Hernández, Carlos Ureña Almagro, Miguel Lastra Leidinger

July 16, 2003

Contents

List of Figures

List of Tables

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.

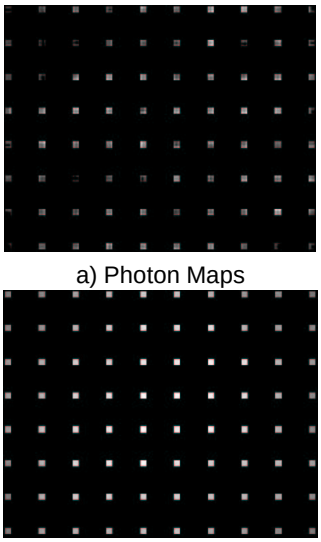
Different estimation methods have been described in Literature. The following methods are relevant to our paper.

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.

Miguel Lastra et all, in [7] describes a method to correct this problem storing the



a) Photon Maps

b) Tangent Plane estimation

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

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 $Light \leftarrow 1$ **to** *numLights*
  for $i \leftarrow 1$ **to** number of photons for *Light*
    $Ray \leftarrow Lights[Light]$.GETPHOTON()
    Insert *Ray* in *RayList*
    while *Ray* is not absorbed
      $IntersectedTriangle \leftarrow$First intersection of *Ray*
       against the scene
      $Ray \leftarrow IntersectedTriangle$.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.

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:

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

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

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 $E(x,t)$

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:

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

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

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

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.

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.

Figure 2.5: Density Estimation on the Tangent Plane, 50 000 photons

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

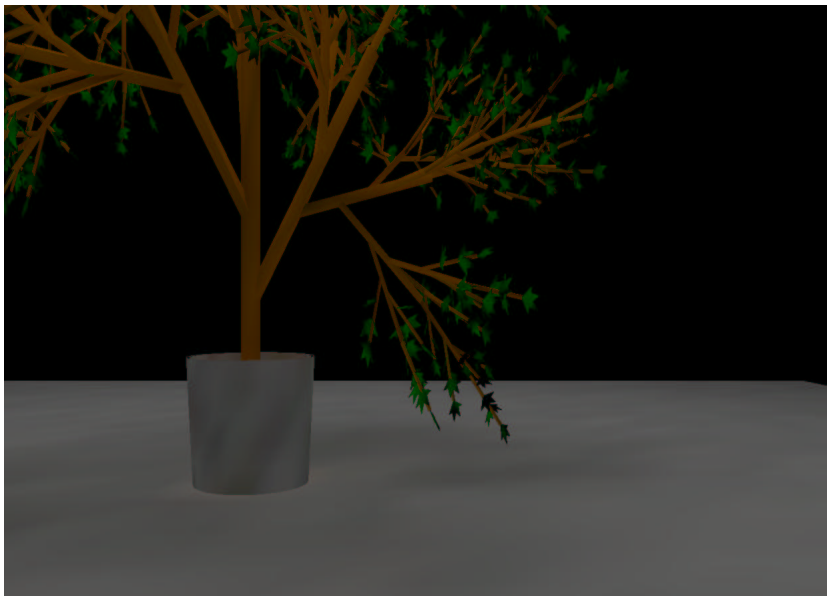


Figure 2.6: Density Estimation using Impact Count, 1 000 000 photons

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 the simulation time is similar (slightly lower for DETP), the error is an order of magnitude higher in Impact Count.

2.6 Reuse of Point ordering

In [8], after describing Ray Caché, the fact that its performance depends on the ordering of the vertices is remarked. This is due to the fact that a linear hierarchical data structure is maintained, composed of spheres of decreasing radius. These spheres keep track of the rays which traverse them. When Density Estimation is made on a point which is inside the current sphere (actually when the whole disc used for the density estimation is), finding the rays which affect its radiance is fast. When the point is outside the sphere, (or at least part of the disc is) the sphere is discarded and the parent sphere in the structure is checked until a valid sphere is found. Then the hierarchy of spheres is rebuilt.

A preordering scheme is presented in [8] which gets the most performance of the algorithm by lowering the number of times the spheres must be rebuilt. Due to the static nature of part of (most of, actually) the scene and the fact that the mobile is rigid, the optimum order for a minimum of caché misses can be studied in a first step. On following steps, this order is reused. In the case of a non rigid mobile, it should be studied whether the preprocess time to recalculate the order is less than the caché miss

increment (It should be noted that the mobile deformation follows rules which conserve part nearbyness; a walking person is a useful example)

2.7 Disc Indexing

A spatial indexing technique for the discs in the Density Estimation on the Tangent Plane was implemented. The algorithm calculates the discs located in the vertices of the scene, and orders them using a spatial indexing method. The radiance is calculated by intersecting each ray against the discs, using the indexing to speed the process.

This method has higher performance than the original Ray Caché intersection method when the discs have a radius of a similar size to the triangle sides, and for a small number of rays. In other situations, Ray Caché's performance is higher.

The maximum number of primitives per voxel should be adjusted as the disc size change, since when discs grow, they intersect one another; dividing the voxel creates voxels in which most of the discs belongs to both voxels. This makes the intersection time with the new voxels higher than the original combined voxel.

The timing results and optimum number of primitives per voxel are shown in table 2.2 for 10 000 photons. Table 2.3 has the timing information for 20 000 photons.

a) Comparison between Ray Caché and Disc Indexing for different scenes. Time as a function of disc radius.

Algorithm \ Radius	0.01	0.02	0.03	0.04
Ray Caché (Whole scene)	4.45	5.70	7.32	9.24
Disc Indexing (Whole scene)	1.46	3.43	7.18	17.79
Ray Caché (Tree)	4.03	5.34	6.78	8.38
Disc Indexing (Tree)	0.89	2.78	7.15	18.37
Ray Caché (Ground)	0.39	0.46	0.62	0.74
Disc Indexing (Ground)	0.21	0.33	0.50	0.68
Circle Area	1.57	6.28	14.14	25.13

b) Best number of primitives per voxel for the Disc Indexing as a function of disc radius.

Scene \ Radius	0.01	0.02	0.03	0.04
Whole Scene	20	20	50	800
Tree	20	10	100	1600
Ground	20	80	120	160

Table 2.2: Recalculation time for 10 000 photons, tree scene

Triangles in the ground have a 0.02 x 0.02 size. Triangles in the tree have a size around 0.01 x 0.01. Circle area is measured using 1 for the area of a triangle in the ground. It can be seen that for sizes under 0.04, the new method is faster than Ray

a) Comparison between Ray Caché and Disc Indexing for the Tree Scene. Time as a function of disc radius.

Algorithm \ Radius	0.01	0.02	0.03	0.04
Ray Caché (Whole scene)	5.50	8.16	11.38	15.89
Disc Indexing (Whole scene)	2.89	6.82	14.12	41.95

b) Best number of primitives per voxel for the Disc Indexing as a function of disc radius.

Scene \ Radius	0.01	0.02	0.03	0.04
Whole scene	20	20	200	1600

Table 2.3: Radiosity recalculation time for 20 000 photons for the Tree Scene

a) Comparison between Ray Caché and Disc Indexing for the Second Tree Scene. Time as a function of disc radius.

Algorithm \ Radius	0.03	0.04	0.05	0.10
Ray Caché (Whole scene)	2.28	2.86	3.79	7.83
Disc Indexing (Whole scene)	1.34	2.17	3.22	10.29

b) Best number of primitives per voxel for the Disc Indexing as a function of disc radius.

Scene \ Radius	0.01	0.02	0.03	0.04
Whole scene	40	80	160	32000

Table 2.4: Radiosity recalculation time for 20 000 photons for the Second Tree Scene

Caché. Since the triangles in the ground are bigger than the ones in the tree, the scene with only the ground has better results also for disc radius of 0.04.

For 20 000 photons, the results are only better for sizes of 0.01 and 0.02. The problem is that this method is linear in the number of rays.

Another scene was generated with a different sort of tree and ground, with bigger triangles, which is shown in figure 2.7. The timing information for this scene can be seen in table 2.4.

These tables show Disc Indexing's performance decrease as the area of the discs increases. It can be seen that the results are better using larger discs than in the previous scene, due to the fact that the triangle size has increased.

For large discs, the normal procedure of dividing the voxels until few objects remain in the voxel or a maximum depth is reached does not work. Since the discs are so large, they belong to most or all of the newly created nodes. This makes the algorithm go through the list of all discs for each node, so the resulting time is higher instead of lower. In order to avoid this, the maximum depth must be decreased as the disc radius

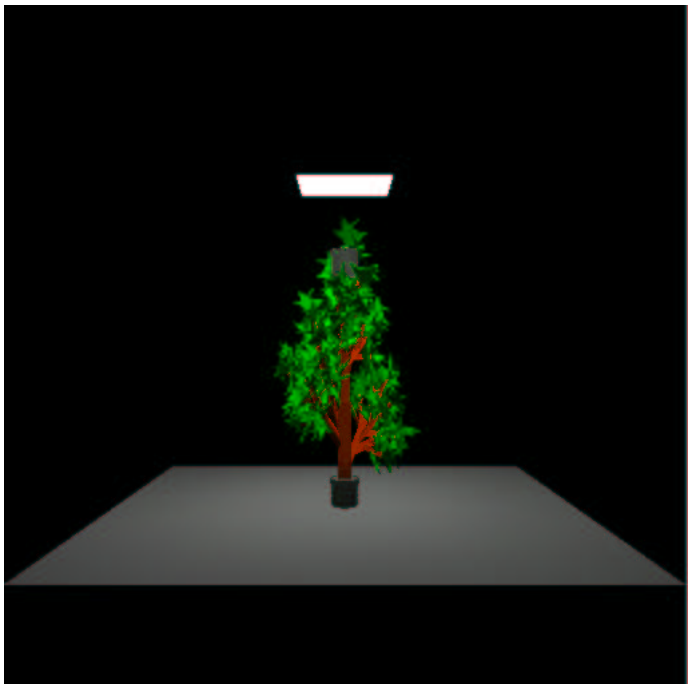


Figure 2.7: Second Tree Scene, 20 000 photons

increases.

One advantage of the Disc Indexing method is an easy parallelization, since the disc indexing tree is static, and can be accessed with no contention problems. On the other hand, synchronizing access and update of the ray caché is complex, and lessens the caché performance.

2.8 Study of Pre-intersection tests

The Badouel, Plücker and Möller pre-intersection tests were studied with respect to the ray-circle intersection tests. An equilateral triangle which is tangent to the disc is created, and rays are intersected first against the triangle, and if hit, against the circle. While these tests on their own achieve a performance boost of 34%, in combination with Ray Caché they just have an increase in performance of arount 1 %. These tests' performance depends on the size of the disc, since for small discs most rays do not intersect either the triangle or the disc, which means the pre-test will speed up the calculation, and for big discs a significant fraction of the discs intersect both the triangle and the disc, so the pre-test will only be a burden. For Real-Time scenes, we are interested in big discs, since that makes possible to have a smaller number of rays. Unfortunately, it means that the pre-test increase in performance is almost negligible.

The available pre-intersection tests were tested using the scene shown in section 2.9. The results for the different tests can be seen in table 2.8 (times in seconds)

a) Times to simulate the Tree Scene using Badouel Preintersection test

Phase	Initialization	Frame
PhotoSimulation	10.34	1.02
Illumination	679.44	67.90
Total	689.78	68.92

b) Times to simulate the Tree Scene using Möller Preintersection test

Phase	Initialization	Frame
PhotoSimulation	10.29	0.98
Illumination	666.34	67.27
Total	676.63	68.25

c) Times to simulate the Tree Scene using Plücker Preintersection test

Phase	Initialization	Frame
PhotoSimulation	10.40	1.05
Illumination	716.47	68.66
Total	726.87	69.71

d) Times to simulate the Tree Scene using No Preintersection test

Phase	Initialization	Frame
PhotoSimulation	10.38	1.05
Illumination	678.75	68.00
Total	689.13	69.05

Table 2.5: Times to simulate the Tree Scene using different preintersection tests

The tests show that the pre-intersection tests Badouel and Möller are slightly faster than no pre-test. Plücker is slower because it needs the Plücker coordinates to be stored.

2.9 Comparison between Photon Maps and DETP

We are now going to compare the time needed for the update of radiosity information for the Photon Maps density estimation method and Density Estimation on the Tangent Plane.

We will use a scene with 71966 triangles and a 500-triengle mobile object. (See figure 2.8) This scene has also been used in [9], although there was no mobile object in that article. We will call this the “Tree Scene”.

We will use as a reference the image created by using 100 million rays with the DETP estimation method (radius is 1 % of the scene).

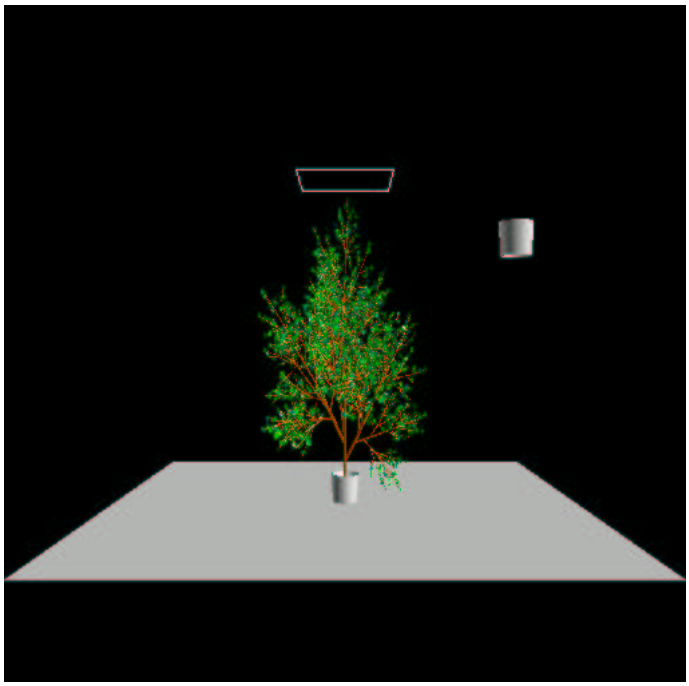


Figure 2.8: Flat Shaded Render of the Tree Scene

2.9.1 Error and bias analysis of PM and DETP

Both algorithms have a noise error due to their statistical nature. They also have a bias error due to the intrinsic algorithm. Photon Maps calculates the mean radiance on a semisphere surrounding the point. This estimator tends to the correct value when the point is not a border. Otherwise, (see figure 2.9, which shows a vertex of a triangle from the direction of the normal, and the corresponding circle), the calculated value has a bias which depends on the angle of the border vertex This angle is defined as the angle of the two sides of the triangle converging in the vertex when these sides belong to the border. When the vertex can be considered the invariant vertex in a triangle fan (i.e. the vertex has edges from different triangles converging in it), the angle is the one between the first and the last edges of the triangle fan. For non planar surfaces, the angle is measured after projection on the tangent plane. Figure 2.10 ilustrates the different situations.

Let α be the angle of the vertex. Photon maps tries to calculate the energy density by adding the energy of the impacts and dividing by the area of the circle. As the picture shows, this gives a much lower estimate of the radiance than the correct one, since

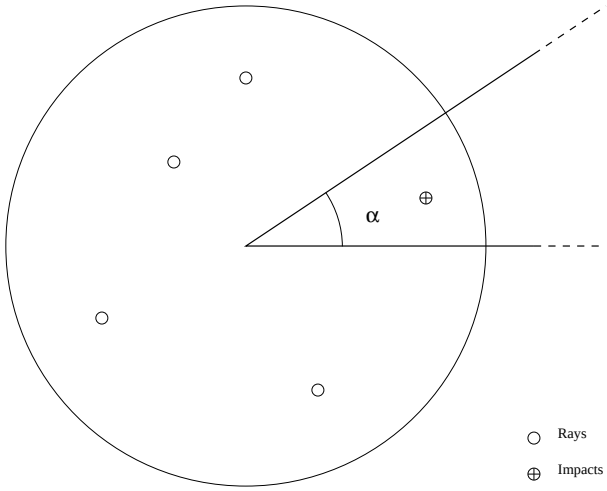


Figure 2.9: Photon Maps Bias: Detail of a border vertex

there is an unreachable zone. There are two solutions. One is taking into account the rays which do not intersect any geometry but intersect the disc (DETP). The other is dividing by the area of the reachable zone.

The bias created by doing neither is $1 - \frac{\alpha}{2\pi}$ (Let E be the total energy of the impacts, and R the radius of the circle. The PM estimator (PME) is $\frac{E}{\pi R^2}$ and the correct estimator (CE) is $\frac{E}{(\alpha/2)R^2}$ The bias is

$$\frac{CE - PME}{CE} = 1 - \frac{PME}{CE} = 1 - \frac{E}{\pi R^2}\frac{\alpha/2R^2}{E} = 1 - \frac{\alpha}{2\pi}$$

The error tends to this bias when the radius of the sphere tends to zero (i.e when the number of impacts increases) Note that for inside vertexes, the angle is 2π, (impacts in adjacent triangles are counted, so the angle spans the whole circle) and the bias is 0.

For DETP, the bias comes from the fact that we are calculating the mean radiance in the disc. This bias tends to zero when the disc size tends to zero.

In this section the resulting images for DETP for 1 000 and 10 000 photons, and for Photon Maps for 10 000 and 50 000 photons are shown, and the timing information is given.

2.9.2 Performance Tables

DETP, 1 000 photons

The timing data for figure 2.11, on page 31, can be seen in table 2.6.

As we can see, activating the ray caché and sorting the queries allow us for 1/0.11=9 frames per second, which might be considered Real-Time.

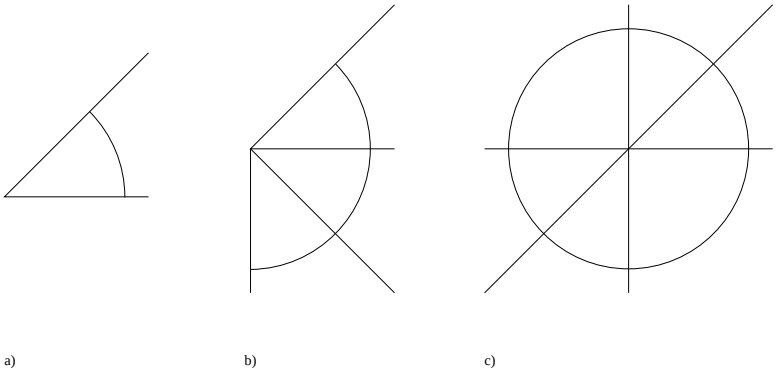


Figure 2.10: Measurement of the border angle
a) Isolated border vertex b) Triangle Fan c) Interior vertex

Photosimulation time (first frame) 0.09
Photosimulation update 0.00

No Ray Caché		Ray Caché			
		No Sort Queries		Sort Queries	
Init	26.14	Init	2.56	Init	2.56
Frame	0.29	Frame	0.26	Frame	0.11

Bias Error	Noise Error	Total Error
14.403 %	24.64 %	28.6912 %

Table 2.6: DETP, 1 000 photons

DETP, 10 000 photons

The timing data for figure 2.12 (page 32) can be seen in table 2.7 For this number of rays, we have a refresh rate of two frames per second, which is interactive, but not Real-Time.

Photon Maps

The timing data and error with respect to the reference file for figures 2.13 (10 000 photons, page 33), 2.14 (50 000 photons, page 34) and 2.15 (200 000 photons, page 35) can be seen in table 2.8

The reference files are an image generated with Photon Maps, 5 million photons, used for the Noise column, and an image with DETP, 5 million photons, for the Total column. The Bias column is the difference between the PM reference image and the DETP reference image. It can be seen that as the number of photons increases, the

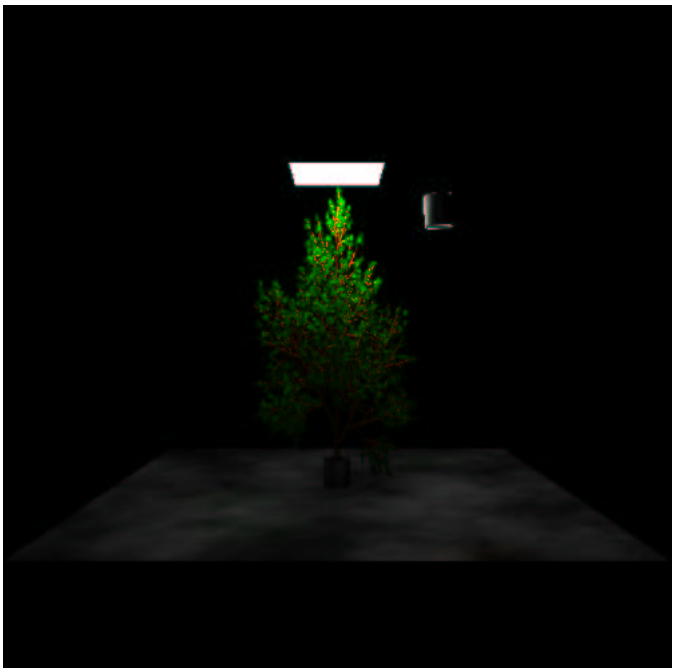


Figure 2.11: Density Estimation on the Tangent Plane, 1 000 photons

image tends to the PM reference image, diverging from the DETP image. This is due to the bias of Photon Maps becoming more noticeable as the size of the sphere decreases.

Conclusion

As we can see, for Photon Maps the refresh rate is 5 fps (10 000 photons), 2 pfs (50 000 photons) and 0.43 fps (200 000 photons).

The quality of the images for the corresponding frame rates of DETP is much lower for Photon Maps, so our preferred method for a real-time application is Density Estimation on the Tangent Plane.

2.10 Glossy Surfaces

Recalculation of glossy surfaces presents some additional problems not present in diffuse surfaces. Camera position changes do not modify the radiance estimate with diffuse surfaces, but do with glossy surfaces. Therefore, the estimation method must be called for camera moves.

The recalculation method is affected, since the radiance estimation depends on the position of the camera. In the glossy case, a number with the radiance estimate in the previous frame is not enough. The actual set of rays which affected the point must be

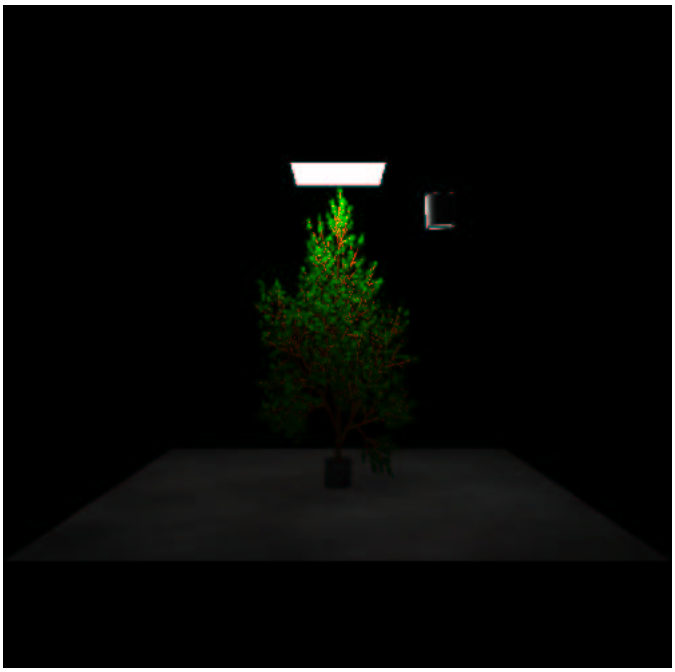


Figure 2.12: Density Estimation on the Tangent Plane, 10 000 photons

stored. Then, they must be used in conjunction with the new camera position to obtain the new radiance estimate.

When the object moves, the new rays and old rays which were used to update the radiance estimate in the diffuse case, now must be checked against the ray list in the vertex, and the ray list must be updated and a new radiance estimate be calculated.

2.10.1 QuadTree

The QuadTree method was developed for the rendering of static scenes. It allows the rays to be summarized in a QuadTree, so that a small number of rays is enough to estimate the radiance in a given direction.

This method does not allow for easy integration with recalculation in non static scenes, since the original rays are lost. Two approaches have been designed in order to integrate QuadTree and recalculation.

Disable summarizing in the QuadTree

The QuadTree can be configured to disable summarizing of rays. Then rays can be added and removed, and the QuadTree is used to speed up the radiance estimate calculation.

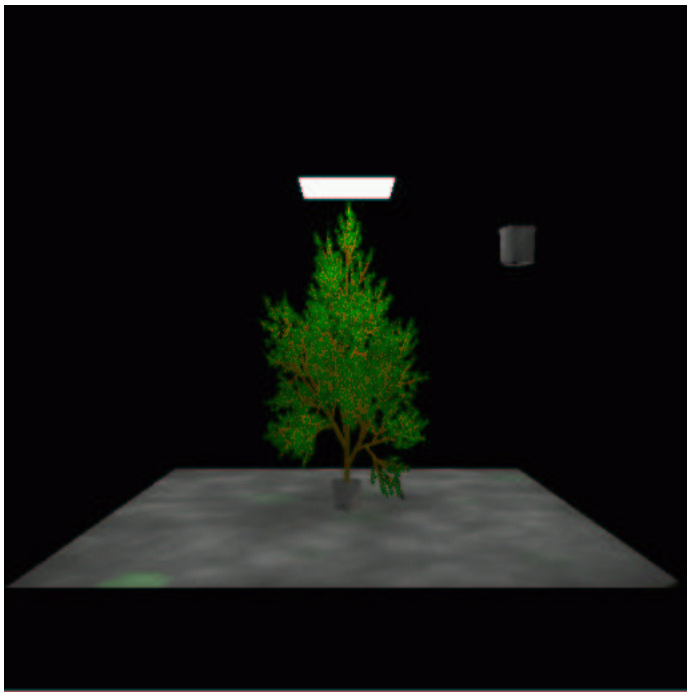


Figure 2.13: Photon Maps Density Estimation, 10 000 photons

Enable summarizing in the QuadTree

In order to do this, the original number of rays in the cell should be stored. Let $\{R_i; 0 \leq i < n\}$ be the original rays. When the QuadTree summarizes a node, the resulting direction of the new ray is calculated with this formula:

$$R = \frac{\sum_{i=0}^{n-1} R_i}{n}$$

In the next frame, some rays will be removed from the QuadTree, and some others will be added. Let $\{N_i; 0 \leq i < n_n\}$ be the new rays.
Let $\{O_i; 0 \leq i < n_o\}$ be the old rays.
The new ray created as a summary of the current rays is:

$$R^{new} = \left(R + \frac{\sum_{i=0}^{n_n-1} N_i}{n} - \frac{\sum_{i=0}^{n_o-1} O_i}{n} \right) \frac{n}{n + n_n - n_o}$$

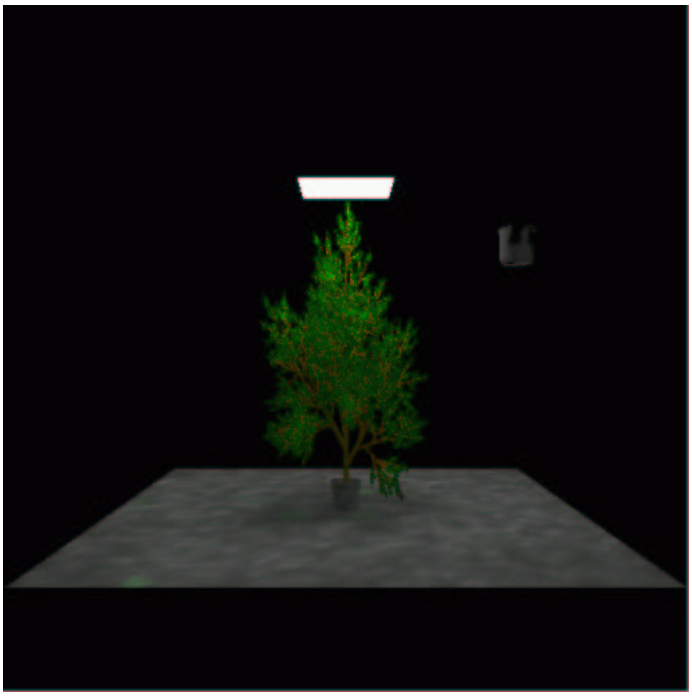


Figure 2.14: Photon Maps Density Estimation, 50 000 photons

This method has one problem: the list of rays in the QuadTree gets degraded after repeated insertion and deletion of rays. This is due to the fact that inserting rays triggers ray summary; some information is lost. Then, rays are removed; the mean direction value is still the same, but fewer rays are used. The radiance estimate for directions relatively far away from the mean value obtain a worse estimate than they would with the original rays.

This method has not yet been implemented.

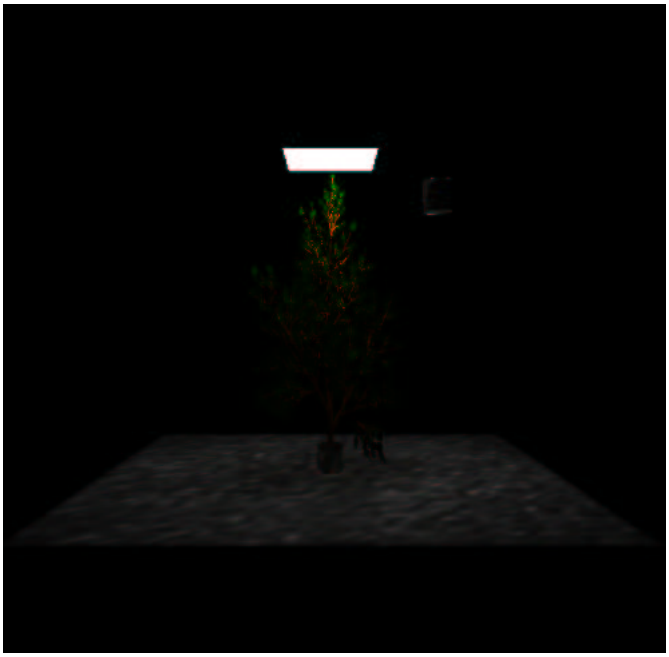


Figure 2.15: Photon Maps Density Estimation, 200 000 photons

Photosimulation time (first frame) 0.8 seconds.
Photosimulation update 0.02-0.03 seconds.

No Ray Caché		**Ray Caché**			
		No Sort Queries		**Sort Queries**	
Init	465.46	Init	66.15	Init	64.0
Frame	3.17	Frame	0.61	Frame	0.46

Bias Error	**Noise Error**	**Total Error**
14.40 %	8.57 %	16.82 %

Table 2.7: DETP, 10 000 photons

Photons	**PhotoSimulation**		**Density Estimation**		**Error**		
	Init	**Frame**	**Init**	**Frame**	**Noise**	**Bias**	**Total**
10 000	0.66	0.02	0.66	0.18	172.5 %	62.80 %	54.39 %
50 000	3.14	0.13	0.86	0.34	93.2 %	62.80 %	62.46 %
200 000	12.51	0.50	2.20	1.79	62.2 %	62.80 %	65.72 %

Table 2.8: Photon Maps

Chapter 3

Implemented System

3.1 Goals

The system's primary goal was the development of a program which were able to calculate the global illumination of a static scene, and which could recalculate the illumination in Real-Time when an object moved through the scene. This was motivated by the fact that most scenarios are mainly static, and changes in illumination are caused by characters or small objects moving. The scene is fed to the program using two files: a (static) scene and a mobile object. For these files, the GRF [11] format was used, because it allows for the description of the global illumination characteristics of the scene.

3.2 Further Characteristics

Using GRF provided us with some characteristics which could improve the usability of the program:

- Textures: They allow us to create high resolution material characteristics, so that the meshing of the geometry can be at a lower resolution. This makes the complexity of the scene smaller, and thus the computing times are decreased.
- Bidirectional Reflectance Distribution Functions (BRDFs): BRDFs provide the reflection parameters of a surface, and are used to compute a realistic (in the Physics sense) image of an object.

3.2.1 Supported Surface Types

Object reflect light depending on the characteristics of the material they are made from. To Realistically simulate light reflection on objects, the way light is reflected on a material must be defined. This is done with the Bidirectional Reflectance Distribution Function (BRDF). This program supports the following BRDFs:

- Perfect Diffuse

- Perfect Specular
- Phong
- Blinn
- Torrance-Sparrow
- Poulin-Fournier
- He-Torrance-Sillion-Greenberg
- Ward
- Lewis
- Schlick
- Oren-Nayar
- Strauss
- Lafortune

There is also the possibility of creating a linear combination of several of the above mentioned BRDFs.

Using a non-diffuse BRDF increases strongly the response time. Therefore, for Real Time purposes, we are using diffuse scenes.

3.3 Supported Estimation Methods

We have implemented the three methods explained in section 1.2: Impact Count, Photon Maps and Density Estimation on the Tangent Plane, with its optimizations, in order to study the increase in performance for these classic methods.

3.4 Classic optimizations present in our system

This chapter describes the optimizations added to the Software.

3.4.1 General Purpose Optimizations

Space partitioning data structures

The first step of the algorithm which calculates the illumination consists in a photon tracking from the light sources. In order to speed up this process, the scene is integrated into a space partitioning data structure. Several methods have been implemented:

- None (Just for debugging purposes)
- Octree [13]

- BinaryTree [5]
- Grid3D [1]
- BSPTree (Based on [15]).

3.4.2 Radiosity Specific Optimizations

Vertex-based calculation

In a triangle mesh, many vertices are shared among different triangles. Since these triangles may have a different color, the emited radiance is different depending on which triangle the vertex is considered to belong to. Nevertheless, the density estimation calculation can be reused, multiplying later on the radiance with the triangle colour to obtain the final value.

Chapter 4

Future Work

4.1 Hardware accelerated direct illumination

In the current implementation, both direct and indirect illumination are calculated by photon tracing. We can make use of graphics hardware to calculate direct illumination, and add indirect illumination in a second pass. This way, the number of photons can be reduced, since indirect illumination is smoother.

4.2 Transparent surfaces

This is the most important type of surface not yet present in our system.

4.3 Ray Tracer

This type of algorithm would allow us to simplify geometry specification, and might be better suited for our goals.

Bibliography

[1] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *EUROGRAPHICS'87*, pages 3–10, Amsterdam, 1987.

[2] J. Arvo. Backward ray-tracing (course developments in ray tracing). *ACM Siggraph Course Notes 12.*, pages 259–263, 1986.

[3] D. Badouel. An efficient ray-polygon intersection. In Andrew Glassner, editor, *Graphics Gems*, pages 390–393. Academic Press, 1990.

[4] Kirill Dmitriev, Stefan Brabec, Karol Myszkowski, and Hans-Peter Seidel. Interactive global illumination using selective photon tracing. 13^{th} *Eurographics Workshop on Rendering*, 2002.

[5] V. Havran, T. Kopal, J. Bittner, and J. Zara. Fast robust BSP tree traversal algorithm for ray tracing. *Journal of Graphics Tools*, 2(4):15–24, 1997.

[6] Henrik Wann Jensen. *Realistic Image Synthesis using Phothon Mapping*. AK Peters, 2001.

[7] M. Lastra, C. Ureña, J. Revelles, and R. Montes. A density estimation technique for radiosity. *1st Ibero-American Symposium in Computer Graphics (SIACG'2002)*, 2002.

[8] M. Lastra, C. Ureña, J. Revelles, and R. Montes. A particle-path based method for Monte-Carlo density estimation. *Poster at: 13th EUROGRAPHICS Workshop on Rendering*, 2002.

[9] Miguel Lastra and Carlos Ureña. Density estimation on the tangent plane for radiosity. *III Jornadas Regionales de Informática Gráfica*, 2002.

[10] T. Möller and B. Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools 1(2)*, pages 21–28, 1997.

[11] Rosana Montes and Carlos Ureña. Wannabe Amazing: Una herramienta de trabajo para iluminación global. *III Jornadas Regionales de Informática Gráfica*, 2002.

[12] S.N. Pattanaik and S.P. Mudur. Computation of global illumination by Monte Carlo simulation of the particle model of light. *Proceedings of 3rd Eurographics Rendering Workshop, Bristol*, 1992.

[13] J. Revelles, C. Ureña, and M. Lastra. An efficient parametric algorithm for octree traversal. *Journal of WSCG (UNION Agency-Science Press)*, 8(2):212–219, 2000.

[14] Franćois Rouselle and Christophe Renaud. Group accelerated shooting methods for radiosity. 10^{th} *Eurographics Workshop on Rendering*, 1999.

[15] Kelvin Sung and Peter Shirley. Ray tracing with the BSP tree. In David Kirk, editor, *Graphics Gems III*, pages 271–274. Academic Press, 1992.

[16] S. Teller. Computing the antipenumbra of an area light source. In *Siggraph '92*, pages 26:139–148, July 1992.

[17] Ingo Wald, Thomas Kollig, Carsten Benthin, Alexander Keller, and Philipp Slusallek. TR-2002-02: Interactive global illumination. Technical report, Computer Graphics Group, Saarland University, 2002.