Thirteenth Eurographics Workshop on Rendering (2002)
P. Debevec and S. Gibson (Editors)

A Particle-Path based Method for Monte Carlo Density Estimation

M. Lastra and C. Ureña and J. Revelles and R. Montes

Departamento de Lenguajes y Sistemas Informáticos
E.T.S.I. Informática. Universidad de Granada.
C/Periodista Daniel Saucedo Aranda s/n E-18071 (Granada-Spain)
{mlastral,almagro,jrevelle,rosana}@ugr.es

Abstract

Radiosity computation on scenes including objects with a complex geometry, or with a large number of faces and meshes with very different sizes, is very complex. We present a new method (based on the photon maps method [6]) where density estimation on the tangent plane at each surface point is performed for irradiance computation by using photon paths instead of photon impacts. Other density estimation techniques require locally simple and flat surfaces in order to be able to compute approximations to irradiance. This requirement is not present in this method, because it does not use photon impacts. Thus we claim the algorithm is more independent of geometry.
The algorithm requires fast ray-disc intersection tests, so we propose techniques for speeding up this part of the process. We compare this approach with the photon map technique, both qualitatively (in terms of its ability to handle complex geometry), and quantitatively (in terms of time and storage complexity). We obtained results which show that better images are obtained with some extra cost both in time and memory.

1. Introduction and Previous Work

In this article we try to find a solution for radiosity computation in the case of complex scenes. In this work complexity is seen as the presence of a large number of objects with non planar boundaries and very different sizes.

In order to compute radiosity values, hierarchical radiosity [5] with adaptive meshing could be used, but the number of small faces makes it difficult to control the mesh resolution inside a mesh, also, the minimum number of top-level patches can not be smaller than the number of different meshes, which might be large. If hierarchical radiosity with clustering is used [13], then, small faces would still make it very difficult to create a set of patches with adaptive resolution inside large meshes.

The photon tracking algorithm is a well known technique which simulates the electromagnetic radiation energy flow in a scene using the particle model of light. By using this method we can count hits on each face [1]. The main problem of this technique is that variance is inversely proportional to face size [10]. Thus, small faces or small meshes either cause high variance or require a huge number of photons. The hits density might be enough for larger faces, but not for the smaller ones. Therefore,few faces will appear quite bright but most of them will appear almost black.

In order to overcome these limitation, other authors proposed new techniques such as photon maps or density estimation [11, 6]. However, there is again a drawback of this technique. Small objects cause high variance or require a huge number of photons. A certain hits density might be enough for larger objects but not for smaller ones [16].A technique suitable for complex scenes is presented in [4] but only a low resolution approximation to radiosity is achieved, because of the storage requirements imposed by the 3D grid used.

In this context we propose a new method. This method uses the photon tracking algorithm, however, in our case we do not use the photon impacts on surfaces. Instead use the straight segments joining two consecutive impacts of a photon.

By using this method radiosity can be computed at any surface point x using a disc centered at x. This disc is tangent to the surface at x, that is, perpendicular to the normal n_x at x (see figure 1).

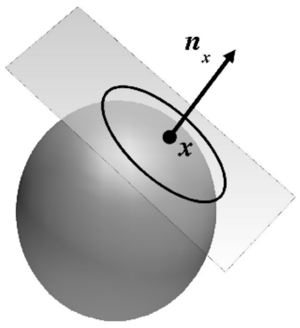


Figure 1: *Disc tangent to surface at x*

After a photon is created or reflected, it follows a straight path until it hits another surface or leaves the scene. We call *ray* each of these segments visited by a particle. A *ray* is defined by a origin o and a direction vector v. During the first phase, a vector of m rays are generated $\{(o_1,v_1),(o_2,v_2),\ldots(o_m,v_m)\}$

Let $P = \{1\ldots m\}$ be the set of indexes of all the rays generated during the photon tracking process. $I_r(x) \subseteq P$ is defined as the set of the indexes of all the rays that intersect the disc centered at x with radius r . After these definitions, the expression of the estimated irradiance is:

$$E(x) \approx \frac{\sum_{i\in I_r(x)} \phi_i}{\pi r^2} \tag{1}$$

where ϕ_i is the energy carried by the i-th ray.

The accuracy of the method we present does not depend on the size of the object which contains x. It is also not necessary to assume that the surface is locally flat because the geometry of the disc is independent on this property. This is not the case of the photon maps because of the use of the photon impacts on the surface of the objects.

2. Speed up of ray-disc intersection computation

As has been stated above, the bottleneck in the system is clearly in the computation of the set of rays intersecting a disc or disc.

The raw method for computing those intersections requires explicitly checking each ray for intersection with the disc. Even by using a highly optimized ray-disc intersection test, this approach is not efficient at all, because the number of rays is usually very high. We have performed tests which yield times in the order of one second per disc, for a set of one million rays.

A more efficient method is obviously necessary to obtain a feasible implementation of the algorithm. In this section we describe a method that reduces the number of ray-disc intersection computed. It is based on the observation that for almost all of the cases, only a very small fraction of the rays hit any given disc. This technique perform an *a priori* selection of candidate rays for intersection. After that, each of these candidate rays is explicitly tested by using a fast ray-disc intersection test.

In order to accelerate the ray-disc intersection test, a ray-triangle intersection test is done first. If the ray does not intersect a triangle containing the disc, then it cannot intersect the disc and therefore the ray-disc intersection test is not necessary. Ray-triangle intersection test is computed much faster than the ray-disc one. Several ray-triangle intersection algorithms have been tested: one based on Plücker coordinates [14, 3, 12] and the algorithms developed by Möller [8], Segura [9] and Badouel [2]. Plücker coordinates proved to be the best method for our application. We tested the pre-intersection test with Plücker coordinates and observed that the time reduction ranged from 40% to 70%.

2.1. The *ray cache*

We have designed a technique that we call *ray-cache*. This technique is inspired in the multilevel cache memory found in current computer hardware. This speeds up main memory access times in the assumption that memory references are made with some degree of coherence.

2.1.1. Properties and management of the structure.

We use a multilevel cache structure which consists in a dynamic list of spheres. For each of these spheres, we include a list with all the rays intersecting it. Spheres are numbered, starting from 1. Center of i-th sphere is called c_i, while its radius is r_i. The first sphere (with index 1) has its center c_1 located at the center of scene bounding box. Radius r_1 is chosen in such a way that any disc falls completely inside first sphere. The list associated to this first sphere contains all the rays intersecting it (that is, the complete list of rays to process). The list of spheres has n spheres and both its length and the spheres it contains do change on demand during processing of discs. Before starting to process discs, the list is built with just one sphere (the first one).

At any time during the computation, the following invariants hold:

1. Sphere list is never empty ($n > 0$)
2. The first sphere contains all rays and is fixed (its center and radius never change).
3. The distance from c_i to c_{i+1} is smaller or equal than r_i. That is, each sphere center is always inside the previous sphere.
4. A real value f exists (with $0 < f < 1$) such that $r_{i+1} = f r_i$. This means that the ratio of radius between successive spheres is constant.
5. Before a disc is processed, the sphere list must be updated so that the disc is completely included in the last sphere (the n-th sphere).

6. If $n > 1$ then $f\,r_n$ must be smaller than disc radius (this implies we could not insert a new sphere between last one and the disc).

These conditions imply that the set of rays associated to one sphere (beyond the first) is always included in the set of rays associated to the previous one. This property is, of course, essential to the algorithm.

When a new disc is processed, we check whether it is included in n-th sphere. If this is the case, then we do not need to update the sphere list and the list of candidate rays is exactly the list of rays intersecting last sphere. This list is already computed and all we need to do is return it for explicit ray-disc intersection test.

In the case a disc is not included in last sphere, we compute the maximal integer value i such that the disc is included in i-th sphere (i is at least 1). Then we remove and delete all spheres beyond that one, and n is made equal to i. We call this a *cache fault* at level i, because this means we need to process a disc which is not included in the last level of our cache structure.

After that we check last condition from the above list. If it does not hold, this means we must create a new sphere (which will be the $(n+1)$-th sphere) and add it at the end of the list. Its radius is fixed by condition (4), while its center must be chosen so both (3) and (5) hold. In fact, we choose a point on the line between c_n and disc center, exactly the closest possible to disc center (it may be equal in some cases) such that new sphere is still included in previous one. The list of rays for this new sphere is obtained by explicitly checking every ray in the n sphere for intersection against $(n+1)$-th sphere. When this new sphere is created, it is added to the list and n is increased in one.

We then go back to the point where we checked for last condition, entering a loop which ends when it holds. This happens when we cannot add new spheres to the list, because they would be smaller than the query disc radius.

2.1.2. Discs coherency and sorting

The efficiency of the previous algorithm is completely dependent on coherency of disc locations. If a disc is very near the previous one (discs orientation does not matter here), the probability of a cache fault is low, and we reuse the candidate ray set, resulting in a great saving of time.

We have tested the algorithm for polygon mesh models, performing irradiance computation on each mesh vertex. The list of vertexes obtained from the model exhibits coherency, as each group of vertexes of a same object surface are stored usually in sequence. In most of the cases, a vertex and the next share one edge, which is usually short for complex scenes. Results obtained outperform the other technique described in this paper (see result sections).

However the usage of this technique imposes a requirements (coherency) on the geometry in order to be efficient, which is in contradiction of our goal expressed in the introduction (that is, to produce an algorithm as much independent of geometry as possible). Moreover, it may happen that in other applications (such as ray-tracing), the coherency can be lower than in the one we have tested.

In order to improve this, we have implemented a pre-ordering of the discs which yields a sequence with increased coherency. This pre-ordering is based on the fact that we can know and store in advance the whole set of discs to process. For each disc we store the position of its center. This position is expressed as a tuple (x,y,z) of three 16 bit unsigned integer values. Floating point disc center coordinates are linearly transformed so the whole range of available integer values are used. Any of these a values can be seen as a vector of 16 bits, and we use $a[i]$ to mean the i-th bit in a, with i in the range from 0 (the less significative one) to 15 (the most one).

The sorting algorithm uses a comparison between discs which is based on these integer coordinates. Lets suppose we want to compute whether i-th disc with coordinates (x_i,y_i,z_i) is smaller (goes first) than j-th disc with coordinates (x_j,y_j,z_j). We fist check if $x_i[15] < x_j[15]$, in that case the test is positive (i goes first). If $x_j[15] < x_i[15]$, the test is negative and j-th disc does not go before (is not smaller) than i-th one. If both bits are equal, then we do the same check with $y_i[15]$ and $y_j[15]$, and if they are equal again, we compare MSB bits of z values. If MSB bit is not enough we move to the next bit (the 14th one), and we continue down to LSB bit (0) until the comparison can be decided or both tuples are equal (which may happen because we are using discrete values, but in that case both discs are very near).

The algorithm is based on examining bits and can be easily implemented. By using an efficient quicksort algorithm, the whole sorting of discs can be done very fast, typically in times in the order of one or two seconds for sets of 100000 discs.

The proposed sorting increases coherency because it tends to put together discs which only differ in less significative bits. In fact it can be shown that the ordering is the same we would obtain if we used an octree structure to sort discs. When using this technique, the number of cache faults is greatly reduced and thus computing time also reduces (see result section).

3. Avoiding Artifacts

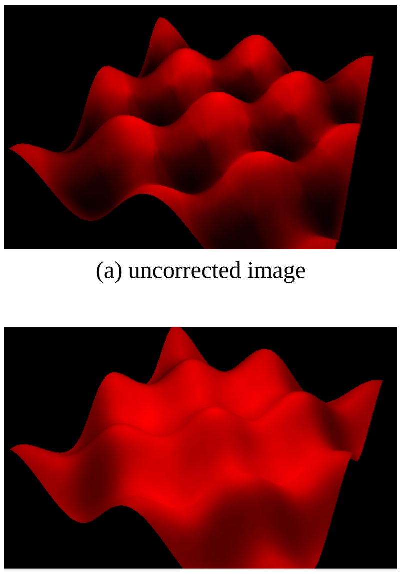
The density estimation technique presented may produce certain errors in the irradiance estimation, and therefore in the image obtained, due to the way it works. The result are some visible artifacts in the final image.

Two kind of artifacts will be discussed here. One of them appeared in non convex parts of a mesh and the other one arose when part of the disc was unreachable for any ray.

We propose solutions to these problems and biased and corrected images will be shown to illustrate each of them.

3.1. Concave meshes

This artifact produced a bias in concave meshes or parts of a mesh. Then non convex parts of the scene resulted much darker as they should as if no rays, or only a small amount of them reached these zones. An example is shown in figure 2 on top. On the bottom of this figure the correct image is also shown. Figure 3 will be used to explain why this happens. Point *x* lies within a concavity and most of the rays mainly intersect the mesh before they can intersect the disc. Therefore all these rays are treated like blocked rays, as if a shadow is projected on *x*. In order to avoid this undesired effect, not only the ray parameter value of the first intersection of each ray is stored, but also the second one (in case it exists). If a ray intersects with the disc before the second intersection point (the second intersection with an object of the scene in the direction of the ray) and the first intersection point is in the same mesh as *x*, then the ray is counted as intersecting the disc. Otherwise it is considered that the ray does not intersect the disc.


(a) uncorrected image

(b) corrected image

Figure 2: *Concave meshes bias*

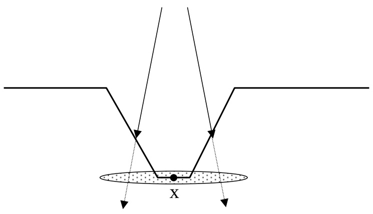


Figure 3: *Concave meshes bias*

In figure 4 the intersection of the ray and the disc associated to point *x* happens after the second intersection point. If intersections at the back of a face are not considered, then it could be said that the intersection of the ray and the disc happens exactly at the second intersection of the ray and the mesh. This intersection is rejected anyway because, as it has been explained before, it does not happen before the second intersection with the mesh. In this case the energy carried by the ray will not have any influence on *x* because it is in a shadowed region for rays coming from that direction.

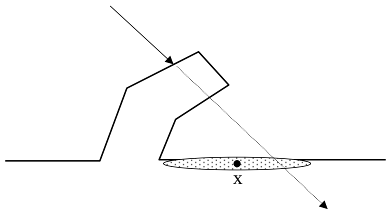


Figure 4: *Another concave area*

3.2. Unreachable regions

There is another kind of artifact that was detected in a scene which was formed by a room and a light source on the ceiling. In this scene, the boundaries of the walls and the floor were much darker than the rest of the scene. The situation is in someway similar to the classical boundary bias as described in [15]. In figure 5 this situation is illustrated. Any ray reflected or originated on the plane of figure 5 can not reach the shaded part of the disc, but, on the other hand, the energy of these rays gets divided by the total area of the disc. This produces a biased lower value for the irradiance in these areas. In general, this happens when a proportion of the disc is not visible from the origin of the ray, because part of it is

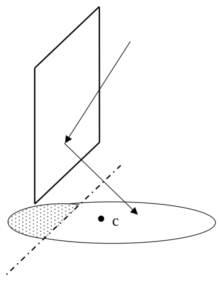
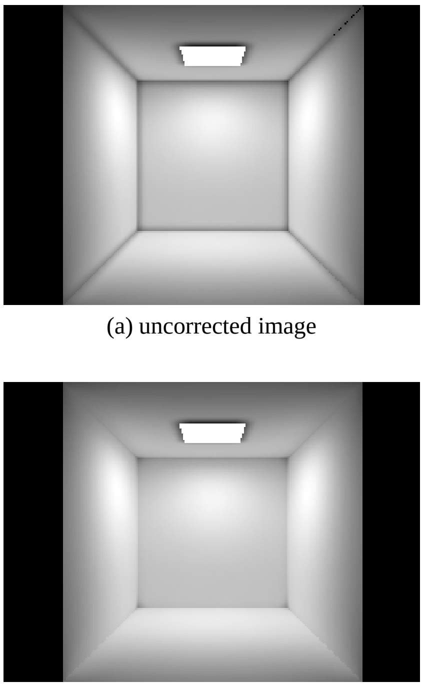


Figure 5: *Unreachable regions bias*

under the plane tangent to the surface at the ray origin. This line divides the area of the disc that can be reached by a ray and the area where that can not happen. This kind of situation has to be detected and for each ray, the area of the disc where such a ray could intersect that disc is calculated and used instead of the whole area.

In figure 6 an example of this kind of bias is shown, as well as the correct image.


(a) uncorrected image

(b) corrected image

Figure 6: *Unreachable regions bias*

4. Comparison with Photon Maps

The photon maps technique [6] is a density estimation method which uses photon impacts on the objects stored in a data structure called photon map. The search for photons is performed inside a sphere. The density estimation method presented in this article has been compared with the photon map method using the source code published in [7]. Both algorithms were integrated in the same rendering system and also the same photon tracking implementation was used in both cases.

There is a important difference about the density estimation used in each method. When using photon maps, the n nearest photons inside a sphere of radius r centered at point x, where irradiance needs to be estimated, are located. If this sphere contains m photons, where $m > n$, the rest of the photons, $m - n$, are not used. Because in the method we present all the rays that intersect the disc centered at x are used, always a high value for n was used to force the photon maps method to use all the photons contained in the sphere.

The comparison with the photon maps method will be done considering the errors each method produces in the images, the storage requirements and time consumption.

All the images were obtained using density estimation on the vertexes of each mesh. This way view-independent results were achieved. Ray tracing could have also been used to select the points for irradiance computation. In fact, the method that we present allows the irradiance computation at any point of the space and only a normal vector is needed.

4.1. Errors

The artifacts or errors produced by the density estimation on the tangent plane, along with its solutions have already been discussed. The photon maps method also produces images with errors under certain circumstances. Some of these situations will be studied.

In figure 7 a plane is shown and density estimation is performed at a point x close to one of its edges. There is a region R of the space where no photon impacts can be found but the irradiance estimate gets finally divided by the volume of the whole sphere. This produces objects with too dark edges [7]. In figure 10(a) an example can be seen. The effect is similar to the unreachable regions artifact introduced before (darkening near the edges).

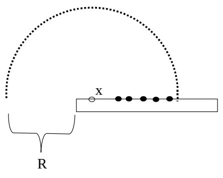


Figure 7: *Darkening near the edges*

Another situation where a biased image is obtained is illustrated in figure 8. On the top of the figure a light source L can be seen and in the middle an obstacle A, which should produce some kind of shadow on the plane below, has been placed. If the distance between this plane and A is less than the radius of the sphere used for irradiance estimation on a point like x, then the shadow will not be obtained. This happens because photons on A will be used to compute the irradiance estimate at x. This kind of problems do not arise when using rays instead of photon impacts. An example of this situation can be seen in figure 9

Finally, as stated in the introduction, when small objects are part of a scene, they might appear much darker than they should, because they do not receive enough photon impacts. In figure 10 a scene is shown where a set of little rectangles

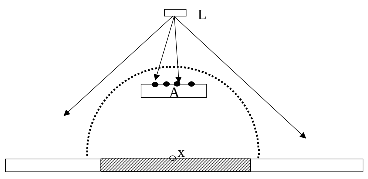


Figure 8: *Photon maps missing shadow bias*

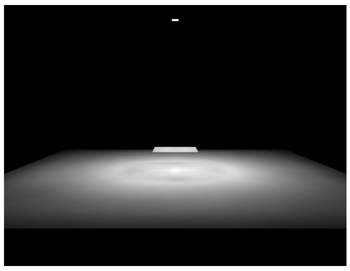


Figure 9: *Photon maps missing shadow bias example*

that are being illuminated from above. The observer is approximately situated under the center of the light source. In figure 10(a), obtained using photon maps, the rectangles result much darker and roughly illuminated as in figure 10(b) which was obtained applying the tangent plane density estimation.

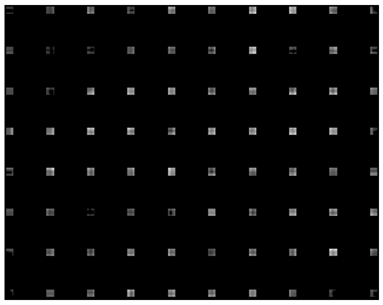
4.2. Storage requirements

According to the photon maps implementation in [7], each photon is stored as follows:

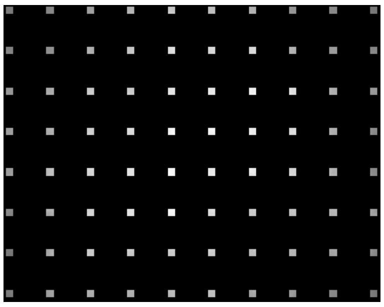
```
typedef struct Photon {
 // photon position
 float pos[3];
 // splitting plane for kd-tree
 short plane;
 // incoming direction
 unsigned char theta, phi;
 // photon power (uncompressed)
 float power[3];
} Photon;
```

The information stored for each ray in our method has the following structure:

```
typedef struct Ray{
  //ray origin
  float ray_origin[3];
  //ray direction
  float ray_direction[3];
  //distance from origin to 1st intersection
  float inter;
  //distance from origin to 2nd intersection
  float inter2;
```



(a) Photon Maps



(b) Tangent plane estimation

Figure 10: *Qualitative comparison with photon maps in a scene with small objects*

```
  //photon power
  float rgb[3];
  //normal at the origin of the ray
  float normal[3];
} Ray;
```

As can be seen the storage requirements for our method are greater. Each photon is stored in the photon map using 28 bytes and we need 56 bytes for each ray. If the ray direction would be coded as in the photon maps method, then, each ray could be represented using 46 bytes. This overhead is caused by the fact that we need the distances from the ray origin to the first and second intersection of the ray and also the normal vector at the ray origin.

If the ray-triangle intersection test is used, then Plücker coordinates of each ray need also to be stored. This means 6 floating point numbers (24 bytes) more for each ray.

4.3. Time consumption

The main advantage of the photon maps method is its speed during the photon location process. Some tests have been done using different scenes.

In figure 11 the first scene is shown. This scene is composed of 46701 vertexes and 4.500.000 photons were shot during the particle tracing phase. The radius used was 4 % of the radius of the bounding sphere of the scene.

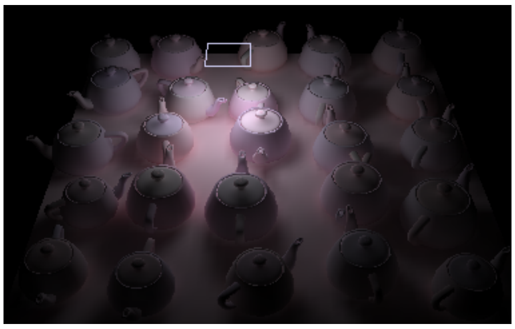


Figure 11: *Teapots test scene*

To obtain the results that are shown in table 1 about this scene, only 200.000 photons were shot and the radius used was 2 % of radius of the bounding sphere of the scene. In this table the first column is related to the density estimation method used, the second column has the density estimation time per sample point expressed in hundredths of a second, and the third column has the total density estimation time expressed in seconds.

Method	per point (secs./100)	Total time (secs.)
1	0.549	256.0
2	0.330	154.0
3	0.003	1.4

	Method
1	Ray cache without sorting and with Plücker coord.
2	Ray cache with sorting and Plücker coordinates
3	Photon maps

Table 1: *Time consumption for irradiance computation for the teapots test scene*

Another scene used can be seen in figure 12 (obtained with density estimation on the tangent plane) and in figure 13 (obtained using photon maps). The scene represents the Tower of Pisa with four colored lights around it. It is composed of 169.831 vertexes and 2.000.000 photons (divided into four passes of 500.000 particles) were shot during the particle tracing phase. The radius used in this case was 3 % of radius of the bounding sphere of the scene. The irradiance estimation time obtained for this scene can be seen in table 2. Again, in this table the first column is related to the density estimation method used, the second column has the density estimation time per sample point expressed in hundredths of a second, and the third column has the total density estimation time expressed in seconds.

The photon maps method outperforms in terms of time

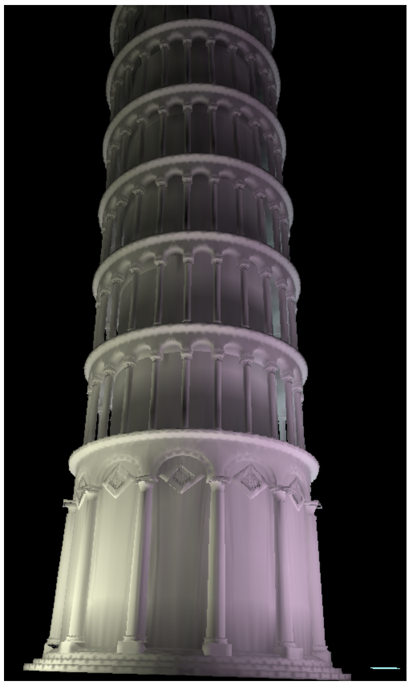


Figure 12: *Obtained with density estimation on the tangent plane*

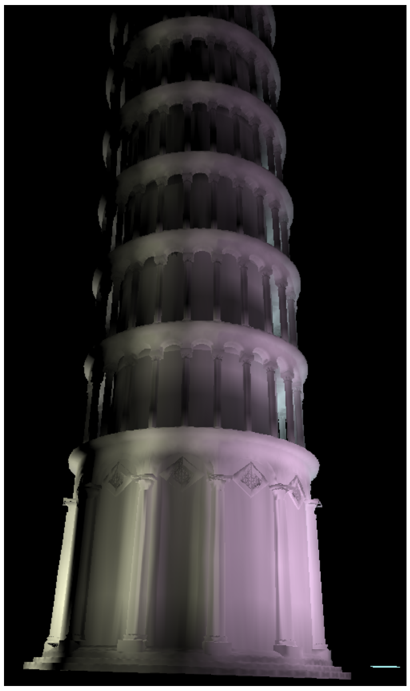


Figure 13: *Obtained with photon maps*

Method	per point (secs./100)	Total time (secs.)
4	0.27	1836
5	0.08	544

	Method
4	Ray cache with sorting and Plücker coordinates
5	Photon maps

Table 2: *Time consumption for irradiance computation for the Tower of Pisa scene*

consumption the density estimation on the tangent plane because it is more complicated to work in the lines space than in the 3D points one. There has also to be taken into account that the implementation of photon maps used does not include any error control code as the code used for our method.

Finally, it is also important to state that the density estimation time difference between the two methods is much smaller for the Tower of Pisa scene than in the other one, and also the resulting image of this scene is of a higher quality using the density estimation on the tangent plane than using photon maps.

5. Future Work

The storage requirements for the rays should be decreased in order to allow more rays to be created. On the other hand, some extensions to the basic algorithm are also planned: non purely diffuse surfaces and other geometric models should be allowed, the use of more advanced density estimation algorithms and obtention of iterative solutions.

6. Acknowledgments

This work has been supported by the research project coded TIC2001-2392-C03-03 (Spanish Commission for Science and Technology)

References

1. J. Arvo. Backward Ray-Tracing (course Developments in Ray Tracing).ACM Siggraph Course Notes 12. 259-263, 1986. 1
2. D. Badouel. An Efficient Ray-Polygon Intersection. Graphics Gems, ed. Andrew Glassner. Academic Press pp. 390-393. 1990 http://www.acm.org/tog/GraphicsGems/. 2
3. J. Erickson. Plücker Coordinates. Ray Tracing News 10(3). December 2, 1997. 2
4. G. Greger, P. Shirley, P. Hubbard, D.P. Greenberg.The Irradiance Volume. IEEE Computer Graphics and Applications 18(2) pp.32-43, 1998. 1
5. P. Hanrahan, D. Salzman, L. Aupperle. A Rapid Hierarchical Radiosity Algorithm. Computer Graphics, 25(4), July 1991. ACM Siggraph'91 Conference Proceedings. 1
6. H.W. Jensen.Global Illumination Using Photon Maps. Rendering Techniques'96 (Pueyo, Schroeder, eds.) pp.21-30, Springer-Verlag, 1996. 1, 5
7. H.W. Jensen. Realistic Image Synthesis Using Photon Mapping. A K Peters. 2001. 5, 6
8. T. Möller, B. Trumbore. Fast, Minimum Storage Ray-TRiangle Intersection. Journal of Graphics Tools 1(2). pp21-28. 1997. http://www.acm.org/jgt/papers/MollerTrumbore97/ 2
9. R.J. Segura, F.R. Feito. Algorithms to Test Ray-Triangle Intersection. Comparative Study. WSCG 2001.Conference Proceedings. Pilsen. 2001. 2
10. P. Shirley. Time Complexity of Monte-Carlo Radiosity. Computer & Graphics 16(1), pp.117-120, 1992. 1
11. P. Shirley, B. Wade, P. Hubbard, D. Zareski, B. Walter, D.P. Greenberg. Global Illumination via Density Estimation. Rendering Techniques'95 (hanrahan, Purgathofer eds.) pp.219-23, Springer-Verlag, 1995. 1
12. K. Shoemake. Plücker Coordinate Tutorial. Ray Tracing News 11(1). July 11, 1998. 2
13. F. X. Sillion. A Unified Hierarchical Algorithm for Global Illumination with Scattering Volumes and Object Clusters. IEEE Transactions on Visualization and Computer Graphics 3(1). 1995. 1
14. S. Teller. Computing the Antipenumbra of an Area Light Source. Computer Graphics (Proc. Siggraph '92), 26:139-148, July 1992. 2
15. B. Walter, P. Hubbard, P. Shirley, D.P. Greenberg. Global Illumination Using Local Linear Density Estimation. ACM Transactions on Graphics 16(3) pp.217-259, 1997 . 4
16. A.Wilkie (advisor: W. Purgathofer). Photon Tracing for Complex Environments. PhD. dissertation. The Institute of Computer Graphics and Algorithms, Technique University of Vienna. 1