Approximate Visibility for Illumination Computations using Point Clouds

Philip Dutré
Parag Tole
Donald P. Greenberg

PCG-00-1 June, 2000

In this report, we present a simple technique to evaluate the visibility between pairs of points in a scene. In most current rendering algorithms, visibility queries are evaluated exactly. Our approach approximates the visibility value between two points using a point cloud representation of the surfaces in the scene. The computed value is a function of the distance and orientation of points in the point cloud relative to the line segment connecting the two query points.

Approximate Visibility for Illumination Computations using Point Clouds

Abstract. In this paper, we present a simple technique to evaluate the visibility between pairs of points in a scene. In most current rendering algorithms, visibility queries are evaluated exactly. Our approach approximates the visibility value between two points using a point cloud representation of the surfaces in the scene. The computed value is a function of the distance and orientation of points in the point cloud relative to the line segment connecting the two query points.

1 Introduction

When rendering scenes, there are two types of possible visibility queries (figure 1). The first type of query can be formulated as: "Looking in a specific direction from a given point, what is the closest surface point in that direction?" The point produced by this query (e.g. primary visibility through a pixel; hitpoints for particle tracing) can then be used as a parameter in other functions. A second type of visibility query can be formulated as: "Are two given (surface) points mutually visible or is there an intervening surface that blocks the line segment connecting them?" The resulting answer can numerically be expressed as 0 or 1 and can subsequently be used in further computations (e.g. form-factor computations in radiosity algorithms; shadows in direct illumination computations).

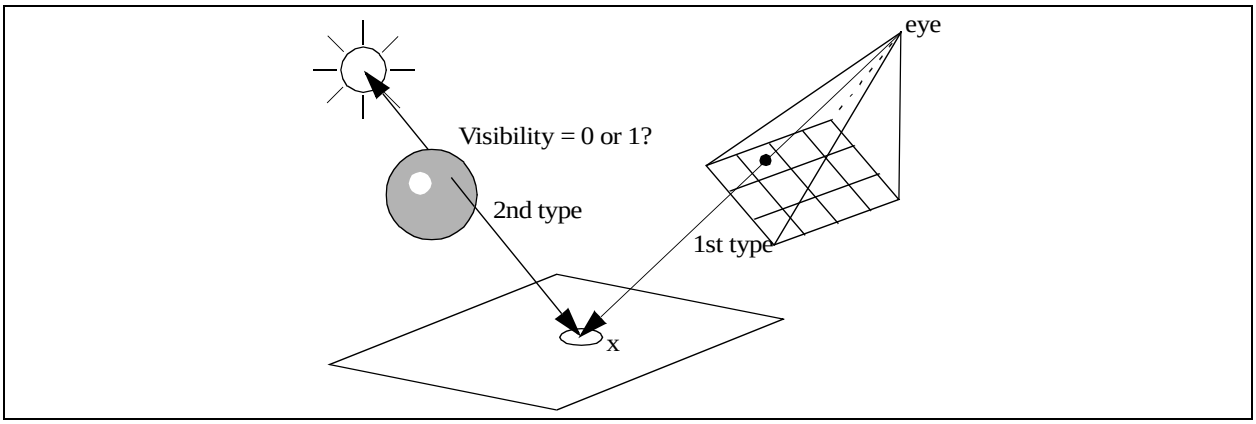


Fig. 1. Two different types of visibility queries. The query to locate *x* from the eye is of the first type; the queries to shade *x* due to the light source are of the second type.

In this work, the second type of visibility query will be computed using an approximate method, which can be summarized as follows: First, the surfaces in the scene are sampled to construct a collection of oriented surface points, which we call the point cloud. Then, when evaluating the visibility between a pair of arbitrary points, we look at the position and orientation of the points in the cloud relative to the line segment connecting the two query points. A value between 0 and 1 is computed which is an approximation of the exact visibility between the two query points. We then use this visibility value in illumination computations. In the case of direct illumination, visibility queries are necessary between every point to be shaded and the light source. With global illumination, visibility queries are dealt with between pairs of surface points, which can be located anywhere in the scene. Note that we still keep the original geometry in order to answer visibility queries of the first type.

The motivation for using an approximate technique to compute visibility stems from the observation that visibility queries are still the major bottleneck in illumination computations. By substituting complex

objects, composed of many polygons, by a simpler point cloud, we hope to get a faster way of evaluating the visibility function usable for illumination computations.

2 Previous Work

Both modeling and rendering algorithms have used collection of points. Points, or particles, have been used to model “fuzzy” objects such as smoke, clouds, fire, water, and even trees [12, 13]. Solid objects have also been modeled using oriented points [3, 8, 4], or have been used as primitive elements to model surfaces [15, 19].

Since an image basically is a collection of points in space, point clouds also have been used to render new views from existing images [2, 9, 16], or from images rendered specifically with this purpose in mind [10, 11]. Various techniques have been proposed to organize and structure these collections of points in such a way that the generation of new views becomes efficient. Tracing rays using depth maps has also been used for generating soft shadows [1].

The approach we follow in this work is related to the field of geometric probability and integral geometry [6,14], although since we are interested in the properties of individual lines and points in space instead of the statistical averages of these properties, the use of results from this field seems to be limited for our purposes.

3 Approximate Visibility

The principle of our approach is as follows:

- N surface points x_i are generated from the original geometry, which are stored together with their respective surface normals n_{x_i}. No connectivity information between the points is stored, and thus all points can be considered independent entities. The number of points N can be smaller or greater than the number of original surfaces in the scene (figure 2).

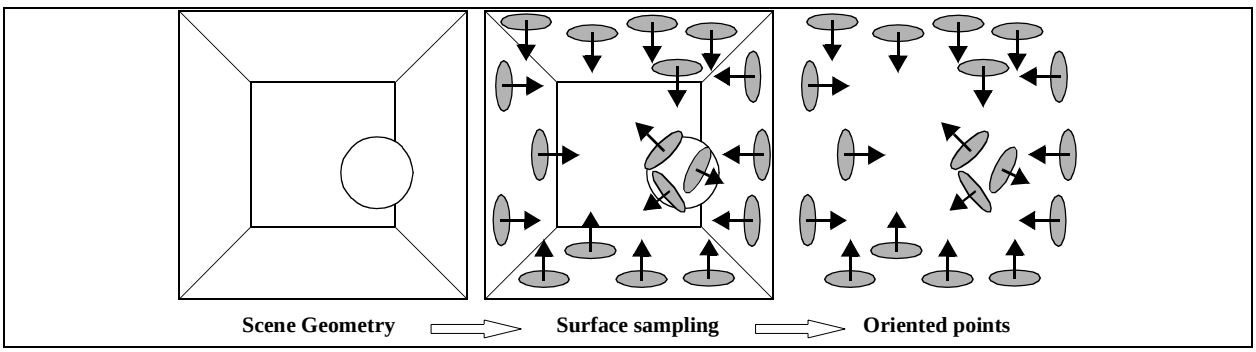


Fig. 2. Surfaces are replaced by oriented points to answer visibility queries of type 2.

- To compute the approximate visibility between two surface points p and q in the scene (which do not themselves have to belong to the point cloud), we look for points x in the cloud which are close to the line segment $\overline{pq}$. We then use a heuristic based on the distance between x and $\overline{pq}$ in the tangent plane Π_x (the plane perpendicular to n_x containing x). This heuristic produces a value between 0 and 1, with 1 indicating full visibility, and 0 indicating the line segment being blocked.

The heuristic starts from the notion that we can construct a small surface-element with shape S_x, located in Π_x and containing x, which can be considered a first-order approximation of the original surface. Since x

is randomly sampled, we approximate the original surface by assuming that S_x can have many different positions which are given by a probability density function $dns(S_x)$ (figure 3).

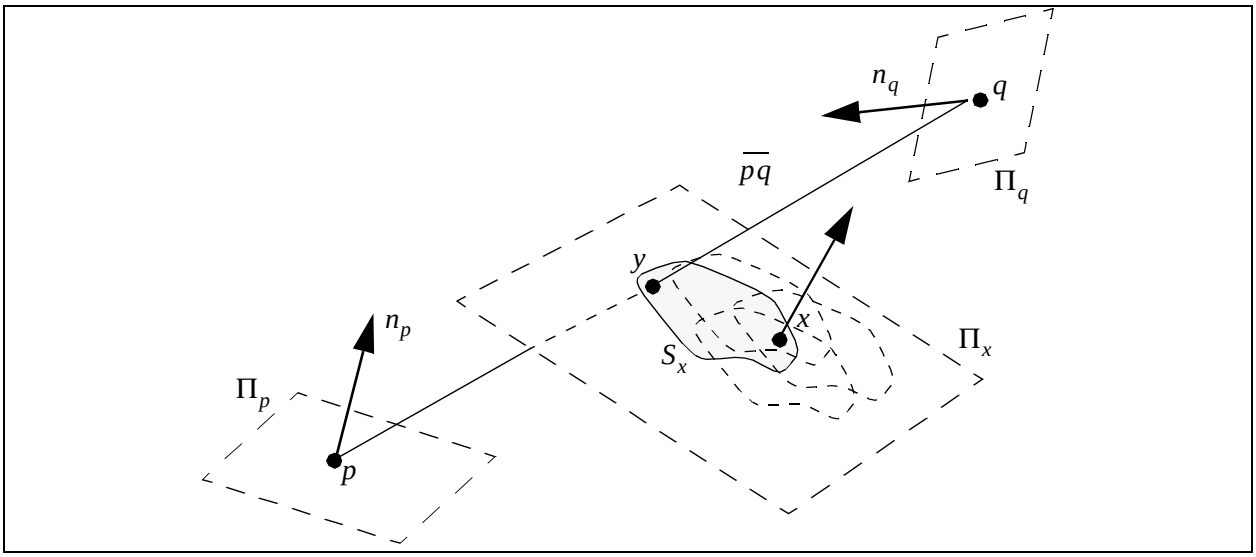


Fig. 3. All possible positions of S_x are checked to see whether they contain y.

Let y be the point of intersection between $\overline{pq}$ and Π_x. The probability $p(S_x, y)$ with which y belongs to S_x is then given by:

$$p(S_x, y) = \int_{\substack{(x \in S_x) \\ (y \in S_x)}} dns(S_x) \cdot dS_x \tag{1}$$

The approximate visibility value, due to the presence of x, is then set to:

$$vis(x, \overline{pq}) = 1 - p(S_x, y) \tag{2}$$

In other words, the approximate visibility value equals the probability that y does not fall within S_x given the distribution $dns(S_x)$. If C points $(x_1, x_2, \ldots, x_C)$ in the point cloud are considered to be potential occluders of $\overline{pq}$, the total visibility value $vis(\overline{pq})$ is computed as a combined probability value:

$$vis(\overline{pq}) = \prod_{i=1}^{C} vis(x_i, \overline{pq}) \tag{3}$$

If the point cloud is large, it is impractical and inefficient to consider all points when computing (3). We use several techniques to keep the cost of computing $vis(\overline{pq})$ low:

- Assuming S_x is finite in size, $dns(S_x)$ has finite support, and $p(S_x, y)$ equals 0 outside this support. Including points in the set $(x_1, x_2, \ldots, x_C)$ which fall outside this support has no effect on the value of $vis(\overline{pq})$.
- Only the C closest points (according to the distance between x and their respective y) are considered as potential occluders. In order to search efficiently for the closest points, the point cloud is stored in an octree structure. This allows for an efficient search algorithm and guarantees that the points are found in order, closest distance first. We can cut off the search when we have found the C closest points, or when we have exceeded the support of S_x. The specific search procedure we use is based on the technique described by Hjaltason and Samet [5]. This algorithm uses a recursive search. By evaluating the distance between the query object (in our case the line $\overline{pq}$) and all cells of the spatial subdivision structure or

objects at the current level of the spatial subdivision structure, a sorted stack is maintained which guarantees that the objects will be found in increasing distance from the query object.

- We also reject points whose corresponding intersection point with $\overline{pq}$ lies outside the range of segment $\overline{pq}$. It is assumed that the original surface will also not intersect the segment $\overline{pq}$ (figure 4).

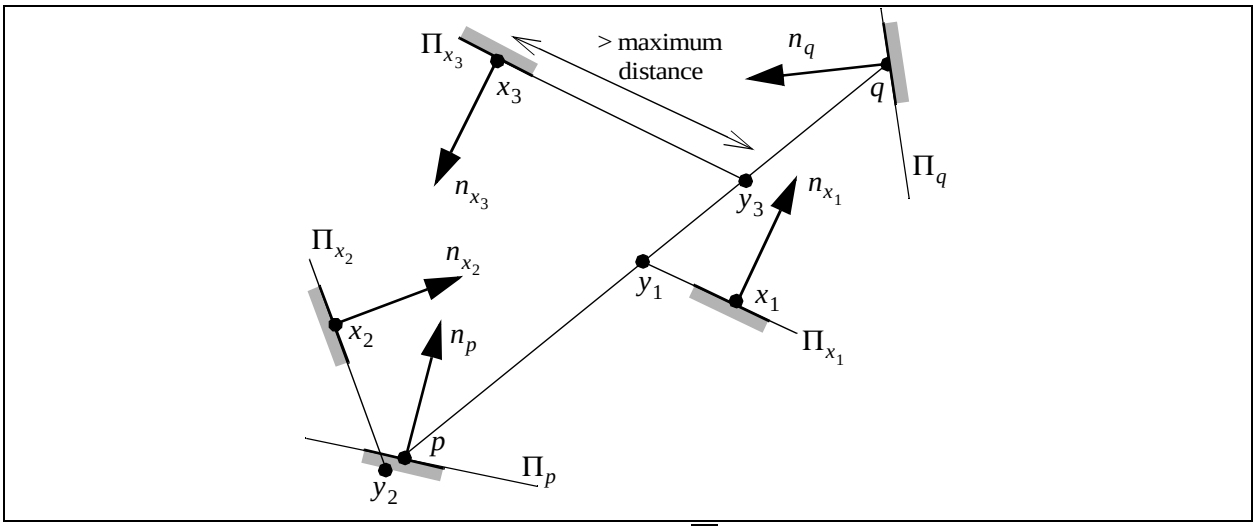


Fig. 4. Only x_1 is selected as potentially blocking the segment $\overline{pq}$. x_2 is rejected because Π_{x2} does not cut the line segment between p and q; x_3 because it is outside the support of S_{x3}.

In our implementation, we use a square with length $L = \sqrt{A/N}$ for the shape of S_x. The orientation of the square is such that it is aligned with the line $\overline{xy}$. The distribution function $dns(S_x)$ for the centre of the square is chosen as $(1 - r_{xy}/L)^d$, where r_{xy} is the distance between x and y, and d is a parameter to control the shape of the distribution. $p(S_x, y)$ can then be computed as (figure 5):

$$p(S_x, y) = 2^d\left(1 - \frac{r_{xy}}{L}\right)^{d+1} \text{ if } \quad \frac{r_{xy}}{L} \geq \frac{1}{2}$$
$$p(S_x, y) = 1 - 2^d\left(\frac{r_{xy}}{L}\right)^{d+1} \text{ if } \quad \frac{r_{xy}}{L} < \frac{1}{2} \qquad (4)$$

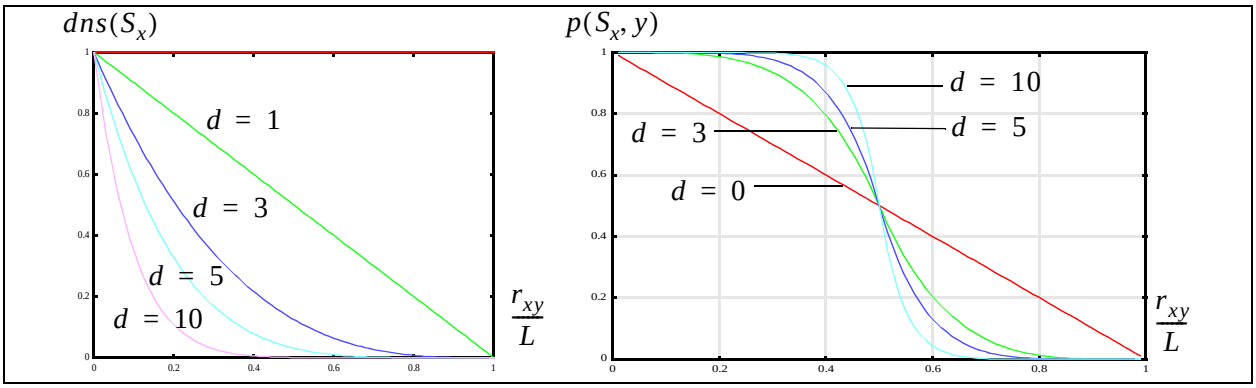


Fig. 5. The distribution functions $dns(S_x)$ and the resulting probability $p(S_x, y)$.

A large value of d makes a distribution where S_x is closely centered around x. As a result, if the distance between x and the query line is large, the probability that the line will intersect with S_x is small, and $p(S_x, y)$ will have very small values. This effect is less outspoken for small values of d. Experiments showed that replacing the square by a circle did not change results significantly, but resulted in more complex expressions and therefore a slightly higher computation time.

4 Validation

To validate our model, we generated a point cloud, using uniform area sampling, from the surfaces of several scenes. 50,000 line segments connecting two random surface points p and q (excluding trivial cases such as self-occlusion) were generated. $vis(pq)$ was then computed, and was considered to be the probability that the line segment pq was visible. This probabilistic outcome for the visibility was then compared to the exact visibility of the line segment. The results are summarized in figure 6. The influence of different parameters was examined:

Size of S_x: Increasing the size of the square L by a factor f, will result generally result in lower values for $vis(pq)$. This is logical, since a larger S_x means that unblocked line segments have a higher probability of being classified as not visible, and the percentage of correctly detected visible line segments decreases. Decreasing the value of L has the reverse effect. The total percentage of line segments detected correctly reaches a maximum for a specific scene-dependent value of f. For most scenes, the optimal value of f varied between 1.0 and 2.0.

The number of points N **in the point cloud**: Increasing N increases the percentage of correctly detected visibility events. In all experiments, the curve flattened rather quickly. For simple scenes, a few thousand points are sufficient to get an accuracy as high as 95 percent. The same trend was noticed in some more complex scenes (e.g. the tree scene, see section 5), where the number of points is the same order of magnitude as the number of polygons. The main reason for this behavior is that the size of S_x is dependent on N. Increasing N decreases the size of S_x, in such a way that the collection of all S_x is a better approximation for the actual surfaces.

Number of closest points C: Increasing C only increases the percentage of correctly detected invisible line segments. For all scenes, there is no significant influence when $C \geq 3$. This is explained by two observations: the closest points have the most influence on the value of $vis(pq)$, so adding more points will not alter the probability of the line segment being blocked much; and there are only a limited number of points within the support of $distr(S_x)$. Increasing C beyond this bound will not change the numerical value of $vis(pq)$.

Choice of $dns(S_x)$: Increasing d, and thus making the distribution more and more centered around the surface point, improved the percentage of correctly detected line segments. However, this did not translate in better image quality. The best quality was achieved using values, ranging from 2 to 4, such that the visibility value had a continuous value, and not a discreet one, as would be the case when d goes to infinity. This is because we compute soft shadows, for which continuous values for visibility are less objectionable than sharp cut-off values, since we are interested in integrated values of visibility and not in discrete values.

5 Application to Direct Illumination

For direct illumination, the following equation for each surface point x visible through a pixel needs to be evaluated:

$$L(x \rightarrow \Theta) = \int_{A_L} L_e(z \rightarrow x) f_r(z \leftrightarrow x \leftrightarrow \Theta) \frac{\cos\theta_x \cos\theta_z}{r_{xz}^2} V(x, z)\, dz \qquad (5)$$

where $L(x \rightarrow \Theta)$ is the radiance reflected from x in direction Θ of the camera, $L_e(z \rightarrow x)$ is the radiance emitted by the light source, A_L is the area of all light sources, $f_r(z \leftrightarrow x \leftrightarrow \Theta)$ is the BRDF, and $V(x, z)$ is the visibility (0 or 1) between x and z. In our tests, a standard Monte Carlo approach for computing this

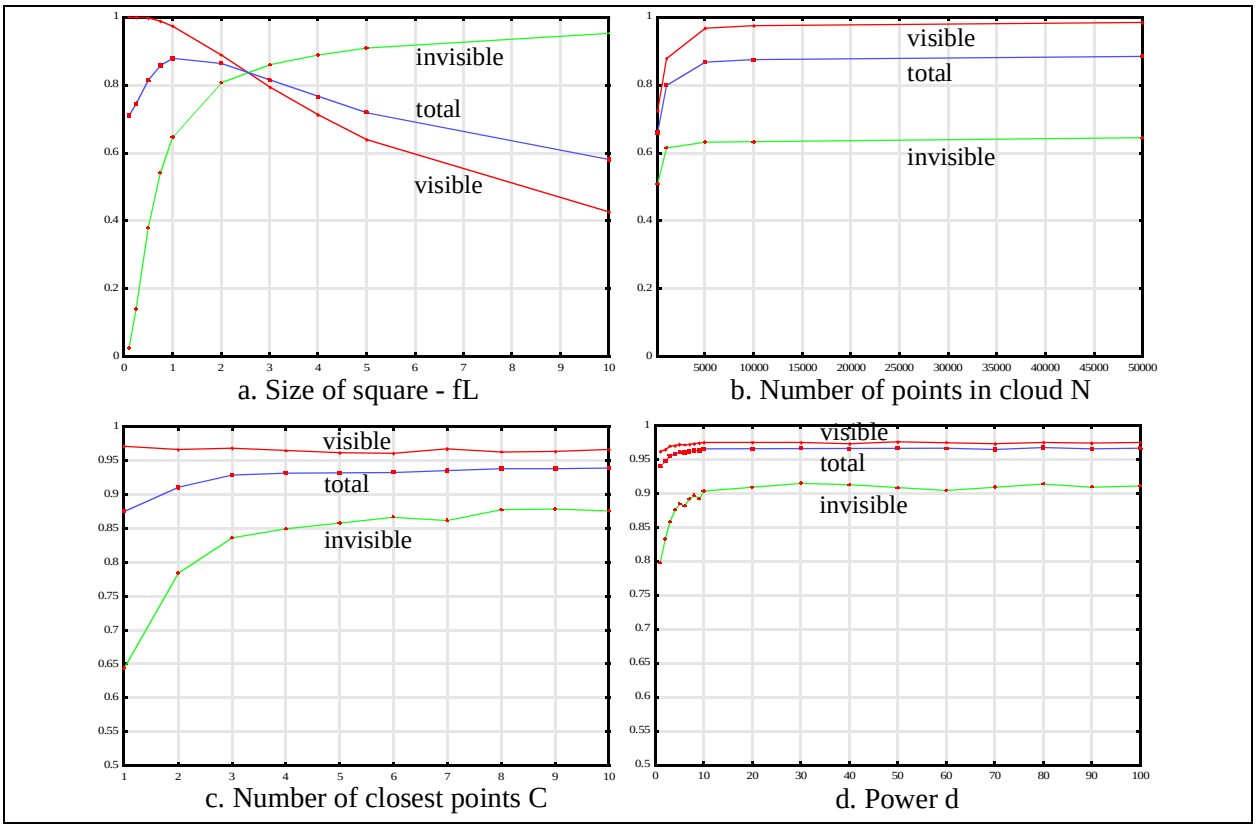


Fig. 6. Influence of different parameters on the percentage of line segments correctly classified.

integral is used to compute $L(x \rightarrow \Theta)$ by generating different points on the light source [17]. All factors of the integrand are evaluated exactly, except the visibility term $V(x, z)$, for which we use the numerical value of $vis(\overline{xz})$, which is a number between 0 and 1, as described in the previous section.

In essence, we could also use $vis(\overline{xz})$ as the probability that $V(x, z)$ equals 1. However, this would introduce extra variance. The expected value of this stochastic process is $vis(\overline{xz})$ itself, so it is this value which we use in the evaluation of the direct illumination integral.

Figure 7 shows two trees casting a shadow on a ground plane, due to an area light source that subtends a solid angle of 0.005 steradian as seen from the center of the tree. The scene consists of 18,000 polygons. The middle and bottom image are generated with approximate visibility using 10,000 and 20,000 points respectively. The quality of the shadow is comparable to the reference image, although there are some light leaks near the stem in the middle image, caused by insufficient sampling of the polygons making up the stem. This test scene illustrates that it is possible to obtain faster execution times for environments with a large number of polygons (1 point is faster to process in our approach than 1 polygon). A spatial subdivision acceleration scheme was used to determine visibility in the reference solutions, such that both search algorithms have the same complexity.

6 Application to global illumination

In order to compute a global illumination solution, we need to consider all possible light paths with multiple reflections between the light source and the point to be shaded. We use bidirectional path tracing [7, 18] to compute our global illumination solution. One problem is that we do not use the original geometry, apart from the primary visibility queries, so we cannot generate eye paths with length greater than 1. Therefore,

a large number of light paths are generated and stored in a pre-processing phase. The complete algorithm is as follows:

- Generate light paths starting from the light sources. The direction of each light path after a reflective bounce is chosen at random over the hemisphere of possible directions. The length of each light path is determined using Russian Roulette sampling. All generated light paths are stored, such that they can be re-used for the actual global illumination computations.
- Generate the point cloud which will be used for visibility testing using uniform area sampling.
- For each pixel, find the closest point using the original geometry and compute the illumination value by connecting this point to a random selection of light paths. The visibility of this connecting segment is determined using our point cloud visibility algorithm.

The right-hand column of figure 7 shows a reference image, computed with classic bidirectional tracing, and two images computed with probabilistic visibility, with respectively 20,000 and 100,000 points in the point cloud. One can see that the soft shadows are reproduced correctly (e.g. shadows on left and rear wall, shadows cast by the chair), but that the sharp shadows contain artefacts (e.g. shadows of the pedestals cast on the floor). Due to the small number of polygons, execution times are larger using our approach, since a large number of points is necessary to approximate the visibility correctly.

7 Future Work and Conclusions

Using this method, the original geometry is still used for primary visibility queries, but geometry outside the viewing frustum could be culled and replaced with a point cloud model. This might save on storage requirements. However, one could imagine an approach where all geometry is replaced by a point cloud, and using reprojection schemes to solve the primary visibility [4,2,16].

The model performs best for soft shadows. Problems arise with hard and small shadows, where the structure of the point cloud becomes visible in the shadow. This can be seen in some of the shadows of the global illumination solutions. The global illumination algorithm stores two different sets of points: those used for the point cloud, and those that are points along the light paths. Some experiments that used the points of the light paths as the point cloud to evaluate visibility did not produce satisfactory images. The main problem is the difference in local sampling densities. The collection of light paths generates a large number of points in areas which are directly lit by the light sources, but less in indirectly lit areas. In some parts of the scene this resulted in very crude approximations of visibility, and consequently bad estimates for the radiance values.

The technique is obviously slower for a scenes with very few polygons (e.g. large rooms with mostly large flat surfaces). Thus, it might be a good idea to replace only parts of a model with a point cloud. The large surfaces would still be represented as polygons, but complex objects such as trees or plants would be represented by points.

We can tune our model such that a better shading is reached by increasing the parameters N and C, at the expense of more computation time. It is therefore possible to design a ‘quality’ parameter, that allows the user to choose a trade-off between execution time and shading quality.

The rendered images indicate that the algorithm is capable of producing images which are comparable in quality to a reference solution. For some scenes, the number of points in the point cloud can be smaller than the number of polygons. Investigating this techniques further might prove to be a worthwhile area of research.

8 References

[1] M. Agrawala, R. Ramamoorthi, A. Heirich, L. Moll. Efficient Image-Based Methods for Rendering Soft Shadows. *To appear in SIGGRAPH 2000 Conference Proceedings*, 2000.

[2] Chun-Fa Chang, Gary Bishop, Anselmo Lastra. LDI Tree: A Hierarchical Representation for Image-based Rendering. *SIGGRAPH 99 Conference Proceedings*, pp. 291-298, 1999

[3] C. Csuri, R. Hackathorn, R. Parent, W. Carlson, M. Howard. Towards an Interactive High Visual Complexity Animation System. In *Computer Graphics (SIGGRAPH 79 Conference Proceedings)*, volume 13-2, pp. 289-299, 1979

[4] J.P. Grossman, William J. Dally. Point Sample Rendering. *Rendering Techniques '98*, G. Drettakis, N. Max (eds.) Springer-Verlag 1998 (Proceedings of the 9th Eurographics Workshop on Rendering, Vienna, Austria, June 1998)

[5] Gisli R. Hjaltason and Hanan Samet. Ranking in Spatial Databases. In *Lecture Notes in Computer Science 951*, Egenhofer and Herring eds., pp. 83-95 (Proceedings of the 4th Symposium on Spatial Databases, Portland, Maine, August 1995)

[6] M.G. Kendall. Geometrical Probability, Hafner Publishing Company, New York, 1963.

[7] E. Lafortune and Y. Willems. Bi-directional Path Tracing. In *CompuGraphics Proceedings* (Portugal, Dec. 1993), pp. 145-153

[8] M. Levoy, T. Whitted. The Use of Points as a Display Primitive. T*echnical Report TR 85-022*, The University of North Carolina at Chapel Hill, Dept. of Computer Science, 1985

[9] W. Mark, L. McMillan and G. Bishop. Post-rendering 3D Warping. In 1*997 Symposum on Interactive 3D Graphics*, pp. 7-16, ACM SIGGRAPH, April 1997.

[10] N. Max, K. Ohsaki. Rendering Trees from Precomputed Z-buffer Views. In Rendering Techniques '95, P.M. Hanrahan, W. Purgathofer (eds.), Springer-Verlag 1995 (Proceedings of the 6th Eurographics Workshop on Rendering, Dublin, 1995)

[11] N. Max. Hierarchical Rendering of Trees from Precomputed Multi-Layer Z-buffers. In *Rendering Techniques '96*, X. Pueyo, P. Schröder (eds.), Springer-Verlag 1996 (Proceedings of the 7th Eurographics Workshop on Rendering, Porto, 1996)

[12] W. Reeves. Particle Systems: A Technique for Modeling a Class of Fuzzy Objects. In *Computer Graphics (SIGGRAPH 83 Conference Proceedings)*, volume 17, pp. 359-376, 1983

[13] W. Reeves. Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems. In *Computer Graphics (SIGGRAPH 85 Conference Proceedings)*, volume 19, pp. 313-322, 1985

[14] L.A. Santaló. Integral Geometry and Geometric Probability, Addison Wesley, 1976

[15] R. Szeliski and D. Tonnesen. Surface Modeling with Oriented Particle Systems. In *Computer Graphics (SIGGRAPH 92 Conference Proceedings)*, volume 26, pp. 185-194, 1992

[16] J. Shade, S. Gortler, L. He, R. Szeliski. Layered Depth Images. *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pp. 231-242, 1998

[17] P. Shirley, C.Y. Wang, K. Zimmerman. Monte Carlo methods for direct lighting calculation. *ACM Transactions on Graphics*, January 1996.

[18] E. Veach and L. Guibas. Bidirectional Estimators for Light Transport. *Photorealistic Rendering Techniques*, Springer-Verlag, New York 1995 (Also in Eurographics Rendering Workshop 1994 Proceedings, pp. 147-162)

[19] A. Witkin and P. Heckbert. Using Particles to Sample and Control Implicit Surfaces. *SIGGRAPH 94 Conference Proceedings*, pp. 269-277, 1994

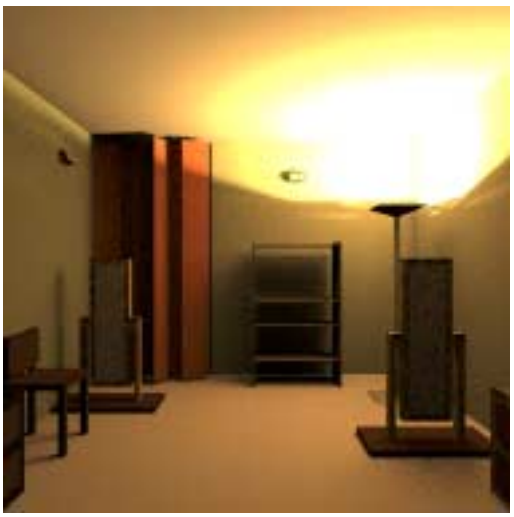
Reference Solution - 97 minutes

Reference Solution - 37 minutes

10,000 points - 50 minutes

20,000 points - 229 minutes

20,000 points - 59 minutes



100,000 points - 334 minutes

Fig. 7. Direct and global illumination solutions using the point cloud for visibility (parameters: C = 3; f = 2; d = 3).