COMPUTER SCIENCE PUBLICATION

# THREE IMPROVEMENTS IN THE RAY TRACING ALGORITHM FOR PROGRESSIVE RADIOSITY

K.W. Wong and W.W. Tsang

Technical Report TR-93-05

May 1993

DEPARTMENT OF COMPUTER SCIENCE
FACULTY OF ENGINEERING
UNIVERSITY OF HONG KONG
POKFULAM ROAD
HONG KONG

# Three Improvements in the Ray Tracing Algorithm For Progressive Radiosity

K.W.Wong and W.W.Tsang

Department of Computer Science

The University of Hong Kong

( email: kwwong@csd.hku.hk tsang@csd.hku.hk )

*ABSTRACT*

*Three improvements in the ray tracing algorithm for progressive radiosity are presented. Firstly, a modified hemicube method is used to perform a visibility pre-test before the rays are cast. As a result, the number of rays traced is reduced substantially. Secondly, the ray tracing step is speeded up by using a binary space partitioning tree to determine the order of patches in finding the closest intersection. Thirdly, the Wallace's form-factor formula is replaced by an analytical formula for accuracy. Experiments show that our method runs much faster than the original one without sacrificing the quality of the pictures produced.*

**Key Words:** radiosity, ray tracing, progressive radiosity, form-factor, depth-buffer, BSP tree.

## 1. Introduction

Nowadays most computer graphics workstations support illumination calculation for point light sources and parallel light sources. These illumination models produce attractive and useful pictures with little computation. Generating more realistic pictures, however, requires more accurate modelling of the physical behavior of visible light. The radiosity method [10,11] is proved to be useful in the synthesis of realistic images for in-door scenes by considering the inter-reflection of light between all surfaces.

The most critical step in the radiosity method is the computation of *form-factors*. The form-factor from small surface (called *patch*) $i$ to patch $j$, denoted as $F_{i-j}$, is defined as the fraction of energy leaving patch $i$ that arrives at patch $j$. The illumination of a receiving patch due to a source patch depends on the radiosity of the source and the form-factor from the source to the patch. The computation of form-factors is the most time consuming step in the radiosity method because it involves visibility determination. In addition, a matrix of form-factors must be computed and stored even though most of them have little effect on the quality of the final image. To solve this problem, a progressive radiosity algorithm [6] has been proposed. The algorithm proceeds in iterations. In each iteration, a patch is chosen to *shoot* its energy out. The form-factors from this patch to all the other patches are calculated and the radiosities of these patches are updated. Experiments showed that satisfactory images can be obtained after the bright patches are processed.

The hemicube method [5] is widely used for computing the form-factors between patches. In the progressive radiosity algorithm, when a patch is chosen to shoot its light out, other patches are projected onto a hemicube placed above the center of the source. The cube is discretized into small elements called pixels. The form-factor from the source to a receiving patch is equal to the sum of the weight associated with the pixels covered by the projected image of the patch. Visibility of patches are determined using the depth-buffer technique for hidden surface removal. Although the hemicube algorithm is efficient and simple, it suffers from two major limitations. Firstly, aliasing appears in the projected images due to the discrete nature of the hemicube pixels. As a result, the form-factors computed are inaccurate. Secondly, Gouraud shading is commonly used to render the images in most graphics workstations. This shading algorithm makes use of the vertex intensities of the polygon being rendered. However, the hemicube algorithm computes only the patch radiosities. The vertex intensities are obtained by interpolating the patch radiosities. This interpolation may introduce another level of errors.

Wallace et al. [13] have suggested a new progressive radiosity algorithm which overcomes the above problems of the hemicube method. To shoot the light of a source patch, their method approximates the source by several circular subsources. Visibility between a subsource and a vertex is determined by ray tracing. A ray is cast from the center of the subsource to the vertex. If the ray reaches the vertex without being blocked, the form-factor from the subsource to the vertex is computed using an approximate formula, and the intensity of the vertex is updated. One of the major advantages of Wallace's method is that it computes the vertex intensities instead of the patch radiosities.

We appreciate the framework of the ray tracing radiosity algorithm. In this paper, we suggest three ways for enhancing the algorithm. Firstly, we observed that ray tracing is not necessary for most vertices. A modified hemicube method can be used to perform a visibility pre-test before the rays are cast. As a result, the number of rays traced is reduced substantially. Secondly, when ray tracing is performed, we use a binary space partitioning (BSP) tree [8] to sort the patches in a front-to-back order according to the orientation of the ray being traced. This order helps us to reduce the number of object-ray intersection calculations. Thirdly, there is an aliasing problem appeared in Wallace's algorithm due to the inaccuracy in their form-factor formula. This problem is solved by replacing the Wallace's form-factor formula with an analytical formula.

The three improvements in the ray tracing radiosity algorithm are presented in detail in Sections 2, 3 and 4 respectively. Experiments and results are presented in Section 5. We give our conclusion in the last section.

## 2. Hemicube Visibility Test

In Wallace's algorithm, the visibility between a subsource and a vertex is determined by ray tracing. Rays are cast from the center of the subsource to each vertex. Although this method accurately determine the visibility between the center of a subsource and a vertex, the ray tracing step is a slow process.

We suggest to perform a hemicube visibility pre-test for each subsource. Before rays are cast from a subsource, a hemicube is placed above the subsource and all the other patches are projected onto the hemicube. Each hemicube pixel covered by a patch's projected image stores the *id* of the patch. When two projected images occupy the same pixel, the *id* of the one which is closer to the subsource.

To determine the visibility of a vertex, we locate the hemicube pixel where this vertex is projected onto. The value (the patch *id*) stored in that pixel is checked against the *id* of the patch containing the vertex. If they are equal, we conclude that the vertex is visible from the subsource. If they are not equal, the eight neighbouring pixels are checked. When none of them has stored the *id* of the patch containing the vertex, we conclude that the vertex is invisible from the subsource. Otherwise, the visibility of this vertex is determined by ray tracing.

The eight neighbouring pixels have to be checked because we want to avoid the aliasing errors. If one of the eight neighbouring pixels has stored the *id* of the patch containing the vertex, the vertex is marginally visible or invisible from the subsource. In this case ray tracing is needed to determine the visibility.

In most of the radiosity methods, in order to obtain a graduate change of intensities on a large surface, we usually divide the surface into smaller surfaces, even though the whole surface is totally visible from a light source. These newly created surfaces can still share the same surface *id*. Therefore, the visibility of the vertices created can be easily determined by the hemicube mechanism.

Experiments showed that in most of the cases, the visibility of a vertex can be determined by the hemicube pre-test without ray tracing. In addition, the hemicube can be implemented by the graphics hardware. We have implemented our method on a Silicon Graphics IRIS Indigo/XS24 workstation (all statistics quoted in this paper are based on this machine). Two simple scenes were tried, and the time used to shoot the light of one subsource is recorded and shown in Figure 2.1. These data confirm that the hemicube pre-test does speed up the visibility determination process. However, the set-up overhead in the software implementation of the pre-test is not justified when the number of vertices is small.

| Number of vertices in the scene | 796 | 4311 |
|---|---|---|
| pure ray tracing visibility test | 1.2 | 21.6 |
| with hardware hemicube pre-test | 1.2 | 6.8 |
| with software hemicube pre-test | 5.3 | 12.0 |

Figure 2.1. Time (in sec) used to shoot the light of one subsource.

## 3. Ray Tracing Using a BSP Tree

There are many speed-up methods for ray tracing suggested in the literature [2,7]. Since in the radiosity algorithm all the objects are represented with polygonal faces, we propose a new speed-up method which uses a BSP tree.

A BSP tree is a binary tree which represents a partitioning of space by planes. The planes are induced by a set polygons in the scene. The root of the tree is a polygon selected from the scene. This polygon's plane divides the space into two half-spaces. Each subtree of the root node represents a half-space. All the polygons lying in front of the root's plane are assigned to the left subtree of the root. Similarly, all the polygons lying behind the root's plane are assigned to the right subtree of the root. A polygon that lies on both sides of the root's plane is split by the plane and each piece is assigned to the appropriate half-space. Both the left subtree and the right subtree of the root are constructed recursively in the same fashion. Figure 3.1(a) shows a 2D example of a scene and Figure 3.1(b) shows one possible BSP tree of the scene. Given an arbitrary viewpoint, a modified inorder traversal of the BSP tree provides a front-to-back ordering of the polygons with respect to this viewpoint [4]. Figure 3.1(c) shows the front-to-back order of the polygons obtained by an inorder traversal of the BSP tree guided by the view point *S*. Further details of BSP tree can be found in [4,8].

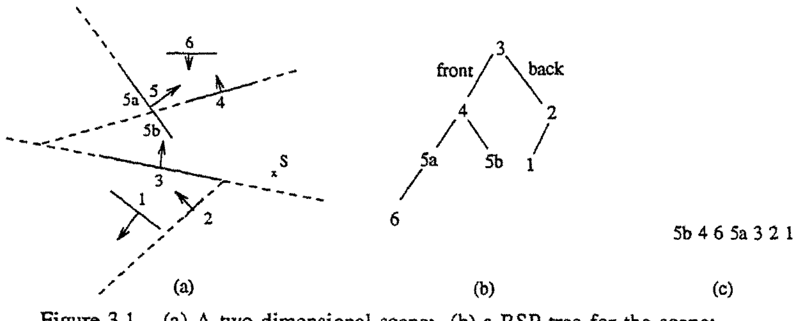
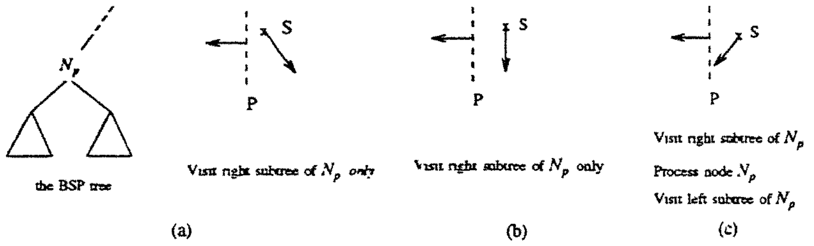


Figure 3.1. (a) A two dimensional scene; (b) a BSP tree for the scene; (c) a front-to-back ordering obtained with respect to *S*.

In our algorithm, a BSP tree is built for the input scene at the beginning. In each shooting process, when a ray is cast from a subsource to a vertex, we use the ray's starting position to guide the BSP tree traversal and obtain a front-to-back order of the patches. The object-ray intersection calculations are performed according to the order obtained until the first intersection is found. If the first intersection occurs at the vertex, the vertex is visible from the subsource. Otherwise, it is invisible. Following the order obtained from the BSP tree traversal ensures that the first intersection found is the closest intersection along the ray.

We can use the direction of the ray to modify the BSP tree traversal such that the number of intersection calculations is further reduced. For example, as shown in Figure 3.2(a), a ray starting from point $S$ is being traced. At a certain step the BSP tree traversal will reach the node containing the plane $P$. Let $N_p$ be the node. Since $S$ lies on the back of the plane $P$, the ordinary BSP tree traversal will visit the right subtree of $N_p$ first, and then process $N_p$ (i.e. test the patches lying on $P$ for intersection), and finally visit the left subtree of $N_p$. However, as the ray is going away from $P$, the receiving vertex will not locate on the plane $P$ or in the left subtree of $N_p$. Moreover, the patches on plane $P$ or in the left subtree of $N_p$ will never block the ray. Therefore, those patches can be ignored. After the right subtree of $N_p$ is visited, the BSP tree traversal can back-track two levels to the parent of $N_p$. This reduces the number of patches to be tested. Figure 3.2(b) and (c) show the other possible orientation between a plane and a ray, and state the corresponding actions. Empirical results show that this BSP tree speed-up method reduces nearly half of the time needed by the straightforward ray tracing process.



$N_p$
the BSP tree
S
P
Visit right subtree of $N_p$ only
S
P
Visit right subtree of $N_p$ only
S
P
Visit right subtree of $N_p$
Process node $N_p$
Visit left subtree of $N_p$
(a)
(b)
(c)

Figure 3.2. Different cases which will be encountered and the corresponding actions should be done when doing ray tracing visibility test with a BSP tree.

Note that this BSP tree speed-up method is a special case of the object space subdivision method suggested in the literature [9]. In that method, the object space is equally divided into enough subspaces until each subspace contains only a few number of objects. When a ray being traced enters a subspace, only the objects inside the subspace are tested for intersection. If the ray does not hit any objects, the algorithm should find the next subspace into which the ray moves. The process of finding the next subspace is difficult and time consuming. Moreover, using octree to divide the object space usually results in too many subdivisions. In our BSP tree speed-up method, the space subdivision is determined by the geometry of the polygonal faces. Paterson and Yao [12] showed that an $O(n^2)$-sized BSP tree can be built using the randomized method, where $n$ is the number of planar facets.

It should be pointed out that this BSP tree speed-up method is suitable for a scene which contains only polygonal faces. If an object (such as a sphere or a cylinder) is not represented with polygons, the algorithm should use other ray tracing speed-up methods. In a scene which contains only polygonal faces, this BSP tree speed-up method is simple and efficient. Another advantage of this method is that the BSP tree built can be used for other purposes, such as the *adaptive mesh generation* method proposed in [3].

## 4. Analytical Form-Factor Formula

In Wallace's paper, the form-factor between circular subsource $k$ and vertex $j$ (treated as a differential area) is approximated as (Figure 4.1a):

$$F_{k\,j} = \frac{dA_j \cos\theta_j \cos\theta_k}{\pi r^2 + A_k} \qquad (1)$$

where

$A_k$ = area of disk $k$
$dA_j$ = differential area located at vertex $j$

The form-factor between source patch $i$ and vertex $j$ (Figure 4.1b) is:

$$F_{i\,j} = dA_j * \frac{1}{N} \sum_k \delta_k \frac{\cos\theta_{jk} \cos\theta_{ik}}{\pi r_k^2 + \frac{A_i}{N}}$$

where

$N$ = number of sample disks used on the source patch
$\delta_k$ = 1 if the ray can reach $j$ from the center of disk $k$; 0 otherwise

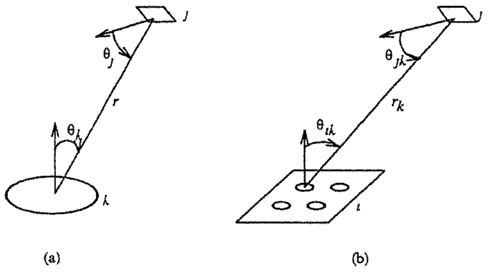


Figure 4.1. (a) Geometry for form-factor between an arbitrarily oriented disk and a differential area.
(b) Geometry for form-factor between a source patch (approximated by multiple disks) and a differential area.

When a receiving vertex is far away from a source, the errors introduced by this approximate formula are negligible. However, when a receiving vertex is close to the surface of a source patch, this method introduced relatively large errors. The errors are caused by two levels of approximation. The first level is the approximate formula from a circular disk to a vertex (Formula 1). Figure 4.2 shows the errors of the form-factors obtained using Formula 1.

| Distance from source to receiving area (r) | 1R | 5R | 10R |
|---|---|---|---|
| Average relative errors of Wallace's formula (Formula 1) | 55.38% | 5.55% | 1.44% |
| Max. relative errors | 99.97% | 11.38% | 2.96% |

Figure 4.2. Statistics of errors introduced by Wallace's formula over the range $0 \le \theta_k \le 89$, $0 \le \theta_j \le 45$. $R$ is radius of the source. ($\theta_k$, $\theta_j$ are defined in Figure 4.1)

The second level of approximation occurs when Wallace's algorithm treats a polygonal sub-source as a circular disk. The shape the source patch is not equal to the union of the circular disks. Some regions of the source are not covered and some regions are covered more than once. As a result, aliasing effect is introduced. In order to visualize the errors, we computed the form-factors from a unit square source to a differential area at different locations of a cross section of space right above the source. The magnitude of the form-factors were rendered using a color scale. The cross section above the source is shown in Figure 4.3. The left picture in the figure shows the correct values. It is obtained by using the original form-factor formula [10]. The right picture in the figure shows the values computed by Wallace's formula, using 3 by 3 circular disks approximating the source. The darken areas on the surface of the source are caused by the aliasing errors. Wallace et al. have mentioned this kind of errors in their paper [13]. They suggested to subdivide the source adaptively for each receiving vertex such that more sample disks are placed near the vertex if it is too close from the source.

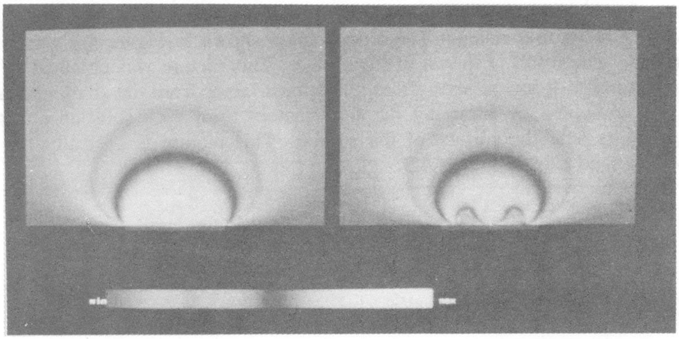


Figure 4.3. Aliasing effect of Wallace's approximate form-factor calculation.

Baum et al. [1] suggested an analytical form-factor formula from source $i$ to differential area $j$ (Figure 4.4) :

$$F_{i\,j} = \frac{dA_j}{2\pi A_i} \sum_{g \in G_i} N_j \quad \Gamma_g^{\,\cdot} \tag{2}$$

where

$G_i$ is the set of edges in patch $i$;
$N_j$ is the surface normal for differential area $j$;
$\Gamma_k$ is a vector with magnitude equal to the angle $\gamma$ (in radians) and direction equal to the cross product of the vectors $R_g$ and $R_{g+1}$ as illustrated in Figure 4.4.

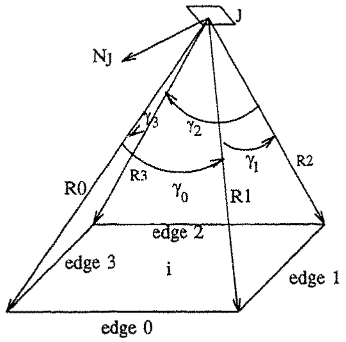


Figure 4.4. Geometry for evaluating analytical form-factor.

In our algorithm, the Baum's analytical formula is used to replace the Wallace's formula when computing the form-factor between a subsource and a vertex. In this case, the actual shape of the subsource is accurately taken into account The cross section shown in Figure 4.3 was rendered again using Baum's formula. The result is shown in Figure 4.5. This picture was obtained by dividing the unit square source patch into 9 square subsources. The form-factor from the source to each receiving differential area was computed by summing the form-factors from each subsource to the receiving area. No aliasing occurs near the surface of the source. This picture is basically the same as the correct one on the left hand side of Figure 4.3.

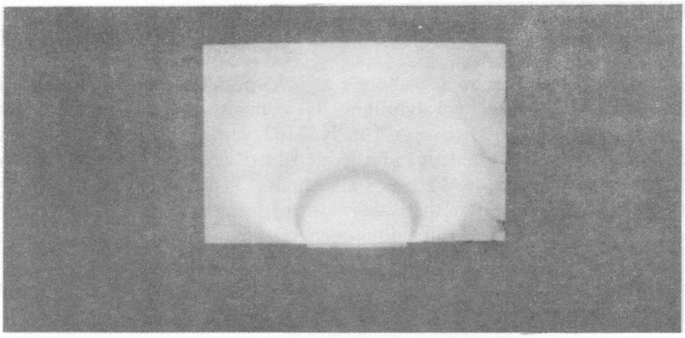


Figure 4.5. Form-factors computed by the analytical formula.

To investigate the execution overhead of using Baum's formula, we computed the form-factors from a square source patch to a differential area in 4140 different orientations and positions. The computation took 0.9 second when Baum's formula was used. When the source was assumed to be a circular disk and Wallace's formula was used, the computation took 0.1 second. Although the Baum's formula is 9 times slower than Wallace's one, this part of computation is only a small portion in the entire process of the radiosity algorithm.

We have derived another approximate formula for estimate the form-factor from a disk to an differential area. The computation of form-factors using this formula runs as fast as Wallace's one, but more accurate. This formula will be useful when execution time is a major concern:

$$F_{k\text{-}j} \approx \frac{dA_j \cos\theta_j \cos\theta_k}{\pi r^2 + A_k - 2\pi r R \sin\theta_k} \times \frac{\sqrt{r^2 + R^2 - 2rR\sin\theta_k}}{\sqrt{r^2 + R^2 + 2rR\sin\theta_k}} \tag{3}$$

A detailed derivation is included in the appendix. The errors of this formula is shown in Figure 4.6.

| Distance from source to receiving area (r) | 1R | 5R | 10R |
|---|---|---|---|
| Average relative errors of Formula 3 | 32.71% | 1.13% | 0.28% |
| Max. relative errors | 98.25% | 4.0% | 1.0% |

Figure 4.6. Error analysis of our approximate form-factor formula.

## 5. Experiments and Results

We have implemented both the original Wallace's algorithm and our improved version. In Figure 5.1 a room is rendered using these two algorithms. It contains 181 patches before subdivision and contains 5562 subpatches after subdivision. Totally 7167 vertices are rendered (because most subpatches share vertices). Estimated ambient radiosity is added when displaying the pictures. Pictures on the right column are produced by Wallace's algorithm. Visibility is determined by ray tracing, and the form-factors are calculated by their approximate formula. Pictures on the left column are produced by our algorithm. All the three improvements mentioned in this paper are implemented. There is no obvious difference between these two set of pictures. Figure 5.2 shows the execution time used for producing these picture. Our algorithm runs much faster than the original one.

| Number of subsources shot | 9 | 48 | 72 |
|---|---|---|---|
| Time (sec) used by Wallace's algorithm | 116.8 | 554.7 | 826.0 |
| Time (sec) used by our algorithm | 55.1 | 243.0 | 363.5 |

Figure 5.2. Time used for producing Figure 5.1.

## 6. Conclusion

We have suggested three ways for enhancing the ray tracing radiosity algorithm. Firstly, a hemicube visibility pre-test is performed to reduce the number of rays cast. Secondly, the ray tracing step is speeded up by using a BSP tree. Thirdly, an analytical form-factor formula is used to increase the accuracy. The major advantages of Wallace's algorithm: approximating an area source with several subsources, and computing the vertex intensities instead of the patch radiosities are preserved. Experiments show that our algorithm runs 4 times faster than the original one without sacrificing the quality of the pictures produced.

(a)

(b)

(c)

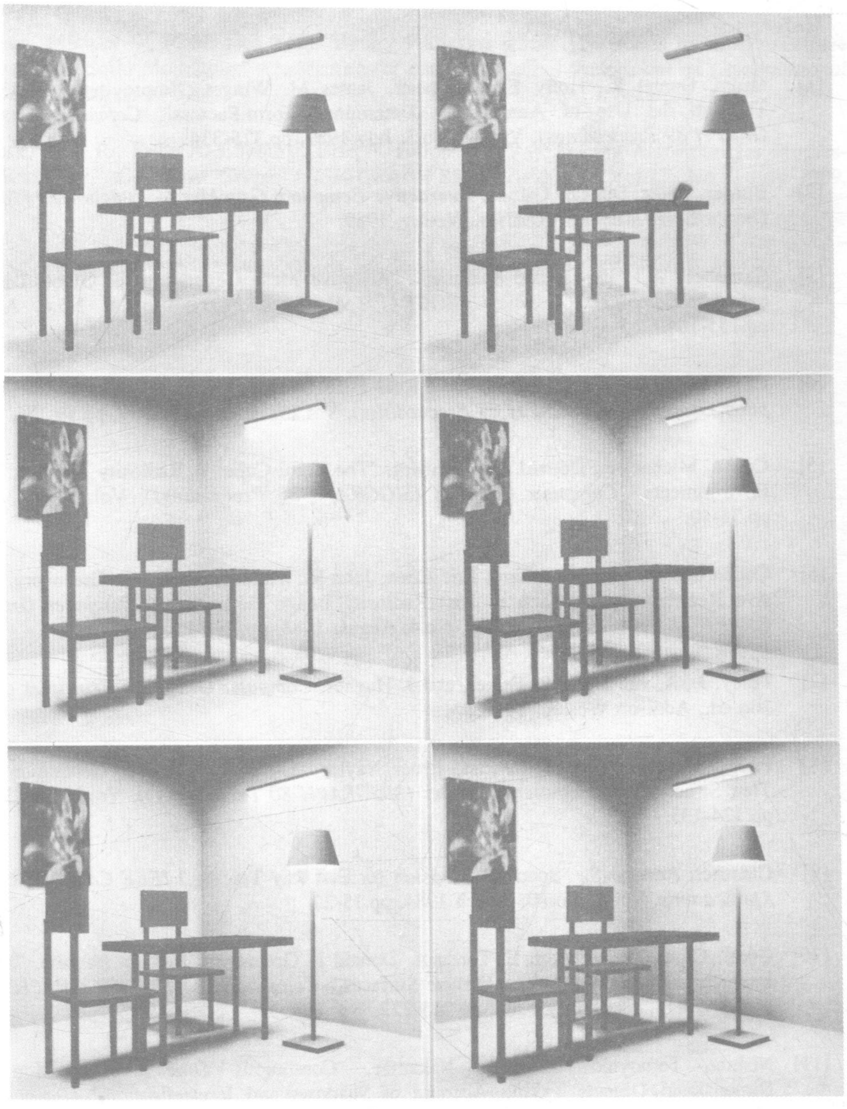


Figure 5.1. A room after the light of
(a) 1 source (totally 9 subsources) is shot;
(b) 16 sources (totally 48 subsources) are shot;
(c) 29 sources (totally 72 subsources) are shot.

## Reference

[1] Baum, Daniel R., Holly E. Rushmeier, James M. Winget, "Improving Radiosity Solutions Through the Use of Analytically Determined Form-Factors," *Computer Graphics (SIGGRAPH'89 Proceedings)*, Vol.23, No.3, July 1989, pp.325-334.

[2] Burger, Peter, Duncan Gillies, *Interactive Computer Graphics -- Functional, Procedural and Device-Level Methods*, Addison Wesley, 1989.

[3] Campbell, A.T. III, Donald S. Fussell, "Adaptive Mesh Generation for Global Diffuse Illumination," *Computer Graphics (SIGGRAPH'90 Proceedings)*, Vol.24, No.4, August 1990, pp.155-164.

[4] Chin, Norman, Steven Feiner, "Near Real-Time Shadow Generation Using BSP Trees," *Computer Graphics (SIGGRAPH'89 Proceedings)*, Vol.23, No.3, July 1989, pp.99-106.

[5] Cohen, Michael F., Donald P. Greenberg, "The Hemi-Cube: A Radiosity Solution for Complex Environments," *Computer Graphics (SIGGRAPH'85 Proceedings)*, Vol.19, No.3, July 1985, pp.31-40.

[6] Cohen, Michael F., Shenchang Eric Chen, John R. Wallace, Donald P. Greenberg, "A Progressive Refinement Approach to Fast Radiosity Image Generation," *Computer Graphics (SIGGRAPH'88 Proceedings)*, Vol.22, No.4, August 1988, pp.75-84.

[7] Foley, J., A. van Dam, S. Feiner, and J. Hughes, *Computer Graphics: Principles and Practice*, 2nd ed., Addison Wesley, 1990.

[8] Fuchs, Henry, Zvi M. Kedem and Bruce Naylor, "On Visible Surface Generation by A Priori Tree Structures," *Computer Graphics (SIGGRAPH'80 Proceedings)*, Vol.14, No.3, July 1980, pp.124-133.

[9] Glassner, Andrew S., "Space Subdivision for Fast Ray Tracing," *IEEE Computer Graphics and Applications*, Vol.4, No.10, March 1984, pp.15-22.

[10] Goral, Cindy M., Kenneth E. Torrance, Donald P. Greenberg, Bennett Battaile, "Modeling the Interaction of Light Between Diffuse Surfaces," *Computer Graphics (SIGGRAPH'84 Proceedings)*, Vol.18, No.3, July 1984, pp.213-222.

[11] Nishita, Tomoyuki, Eihachiro Nakamae, "Continuous Tone Representation of Three-Dimensional Objects Taking Account of Shadows and Interreflection," *Computer Graphics (SIGGRAPH'85 Proceedings)*, Vol.19, No.3, July 1985, pp.23-30.

[12] Paterson, Michael S., F. Frances Yao, "Binary Partitions with Applications to Hidden-Surface Removal and Solid Modelling," *Proceedings of the Fifth Annual Symposium on Computational Geometry*, 1989, pp.23-32.

[13] Wallace, John R., Kells A. Elmquist, Eric A. Haines, "A Ray Tracing Algorithm for Progressive Radiosity," *Computer Graphics (SIGGRAPH'89 Proceedings)*, Vol.23, No.3, July 1989, pp.315-324.

## Appendix

Derivation of the form-factor formula from a disk to a differential area (Formula 3) :

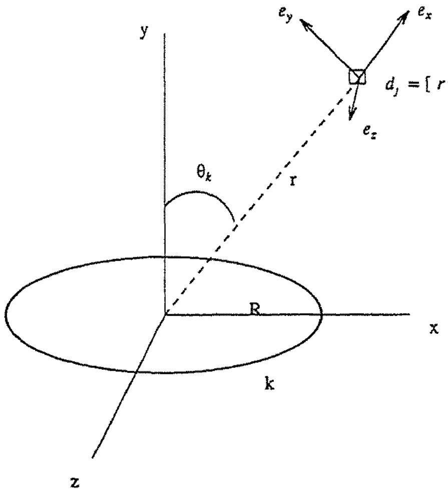


Figure A.1

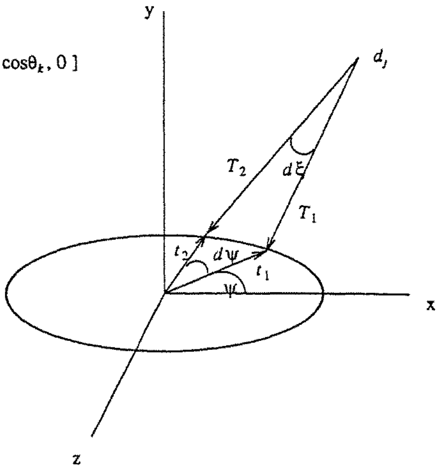


Figure A.2

WLOG, assume center of disk $k$ is located at the origin and $k$ lies on the $x$-$z$ plane. Furthermore, assume differential area j is located at $\vec{dj} = [\ r\sin\theta_k, r\cos\theta_k, 0\ ]$ (Figure A.1).

Refer to Figure A.2:

$$\vec{t_1} = [\ R\cos\psi,\ 0,\ R\sin\psi\ ]$$
$$\vec{t_2} = [\ R\cos(\psi+d\psi), 0, R\sin(\psi+d\psi)\ ]$$

$$\vec{T_1} = \vec{t_1} - \vec{dj} = [\ R\cos\psi - r\sin\theta_k,\ -r\cos\theta_k,\ R\sin\psi\ ]$$
$$\vec{T_2} = \vec{t_2} - \vec{dj} = [\ R\cos(\psi+d\psi) - r\sin\theta_k, -r\cos\theta_k, R\sin(\psi+d\psi)\ ]$$

Let $\vec{\Gamma}_{d\psi}$ is a vector with magnitude equal to the angle $d\xi$ (in radians) and direction equal to the cross product of the vector $\vec{T}_1$ and $\vec{T}_2$.

$$\vec{\Gamma}_{d\psi} = \frac{\vec{T}_1 \times \vec{T}_2}{|\vec{T}_1|\,|\vec{T}_2| \sin d\xi} \times d\xi$$

$$= \frac{\vec{T}_1 \times \vec{T}_2}{|\vec{T}_1|\,|\vec{T}_2|} \qquad \textit{when } d\xi \to 0$$

$$= \begin{bmatrix} \dfrac{-Rr\cos\theta_k\,[\sin(\psi+d\psi)-\sin\psi]}{\sqrt{r^2+R^2-2rR\sin\theta_k\cos\psi}\,\sqrt{r^2+R^2-2rR\sin\theta_k\cos(\psi+d\psi)}} \\ \dfrac{Rr\sin\theta_k\,[\sin(\psi+d\psi)-\sin\psi] - R^2\sin d\psi}{\sqrt{r^2+R^2-2rR\sin\theta_k\cos\psi}\,\sqrt{r^2+R^2-2rR\sin\theta_k\cos(\psi+d\psi)}} \\ \dfrac{Rr\cos\theta_k\,[\cos(\psi+d\psi)-\cos\psi]}{\sqrt{r^2+R^2-2rR\sin\theta_k\cos\psi}\,\sqrt{r^2+R^2-2rR\sin\theta_k\cos(\psi+d\psi)}} \end{bmatrix}^T$$

Using the definitions above, we can write down the form-factor formula from disk $k$ to differential area $j$ by rewriting the Baum's form-factor formula [1] as:

$$F_{k\text{-}j} = \frac{-dA_j}{2\pi A_k} \int_{\psi} \vec{N}_j \cdot \vec{\Gamma}_{d\psi}$$

$$= \frac{-dA_j}{2\pi A_k} \times \vec{N}_j \cdot \left( \int_{\psi} \vec{\Gamma}_{d\psi} \right)$$

Before we evaluate $\int_{\psi} \vec{\Gamma}_{d\psi}$ , we have made an assumption that when projecting $\int_{\psi} \vec{\Gamma}_{d\psi}$ onto the coordinate system defined by $[\, \vec{e}_x, \vec{e}_y, \vec{e}_z \,]$ as shown in Figure A.1, the $\vec{e}_y$ and $\vec{e}_z$ component of $\int_{\psi} \vec{\Gamma}_{d\psi}$ are negligible.

$$\begin{aligned} \vec{e}_x &= [\ \sin\theta_k,\ \cos\theta_k,\ 0\ ] \\ \vec{e}_y &= [\ -\cos\theta_k,\ \sin\theta_k,\ 0\ ] \\ \vec{e}_z &= [\ 0,\ 0,\ 1\ ] \end{aligned}$$

Project the vector $\vec{\Gamma}_{d\psi}$ onto the coordinate system [ $\vec{e_x}, \vec{e_y}, \vec{e_z}$ ], we get:

$$\vec{\Gamma}_{d\psi}^{[e]} = \begin{bmatrix} \dfrac{-R^2\cos\theta_k \sin d\psi}{\sqrt{r^2+R^2-2rR\sin\theta_k\cos\psi}\,\sqrt{r^2+R^2-2rR\sin\theta_k\cos(\psi+d\psi)}} \\ \dfrac{[Rr\cos\psi - R^2\sin\theta_k]d\psi}{\sqrt{r^2+R^2-2rR\sin\theta_k\cos\psi}\,\sqrt{r^2+R^2-2rR\sin\theta_k\cos(\psi+d\psi)}} \\ \dfrac{-Rr\cos\theta_k[\cos(\psi+d\psi)-\cos\psi]}{\sqrt{r^2+R^2-2rR\sin\theta_k\cos\psi}\,\sqrt{r^2+R^2-2rR\sin\theta_k\cos(\psi+d\psi)}} \end{bmatrix}^T$$

Therefore, based on our assumption,

$$\int_\psi \vec{\Gamma}_{d\psi}^{[e]} = [\ \int_\psi \frac{-R^2\cos\theta_k \sin d\psi}{\sqrt{r^2+R^2-2rR\sin\theta_k\cos\psi}\,\sqrt{r^2+R^2-2rR\sin\theta_k\cos(\psi+d\psi)}},\ 0,\ 0\ ]$$

$$= [\ \frac{-2\pi R^2\cos\theta_k}{r^2+R^2-2rR\sin\theta_k} \times \frac{\sqrt{r^2+R^2-2rR\sin\theta_k}}{\sqrt{r^2+R^2+2rR\sin\theta_k}},\ 0,\ 0\ ]$$

and the form-factor from disk $k$ to differential area $j$ can be written as:

$$F_{k\,j} = \frac{-dA_j}{2\pi A_k} \times \vec{N_j}\ \ (\int_\psi \vec{\Gamma}_{d\psi})$$

$$= \frac{-dA_j}{2\pi A_k} \times \|\vec{N_j}\| \times \|\int_\psi \vec{\Gamma}_{d\psi}\| \times cos(angle\ between\ \vec{N_j}\ \text{and} \int_\psi \vec{\Gamma}_{d\psi})$$

$$= \frac{-dA_j}{2\pi A_k} \times \frac{-2\pi R^2\cos\theta_k}{r^2+R^2-2rR\sin\theta_k} \times \frac{\sqrt{r^2+R^2-2rR\sin\theta_k}}{\sqrt{r^2+R^2+2rR\sin\theta_k}} \times \cos\theta_j \qquad \textit{based on our assumption}$$

$$= \frac{dA_j\cos\theta_k\cos\theta_j}{\pi r^2 + A_k - 2\pi rR\sin\theta_k} \times \frac{\sqrt{r^2+R^2-2rR\sin\theta_k}}{\sqrt{r^2+R^2+2rR\sin\theta_k}}$$