Octant Priority for Radiosity Image Rendering

Yigong Wang and Wayne A. Davis
Department of Computing Science
University of Alberta
Edmonton, Alberta
Canada T6G 2H1

Abstract

This paper presents a new scheme for image display using radiosity methods. An octree technique performs subdivision of polygons which is required in any radiosity method. The octree-based subdivision not only subdivides polygons into patches, but also produces a sequence of leaf octants implicitly sorted in the octree. By traversing the octree, a depth priority of these leaf octants with respect to a given viewer position can be found efficiently. Since each octant contains a certain number of patches, octant priority implies a patch depth priority that is used to determine visibility in computing form-factors by a front-to-back list priority algorithm. This list-priority method has many advantages over the z-buffer and can avoid some unnecessary computation.

Keywords Radiosity, Illumination Models, Octrees Image Synthesis, Hemi-cube, Space Subdivision.

1. Introduction

Radiosity methods for image synthesis have been important for some time. This is due largely to its potential in generating more realistic images. It overcomes several difficulties that are hard for previous approaches to resolve. The major advantage of the method is to allow for inter-reflection of lights in an enclosure, more faithfully simulating the behavior of light based on thermal engineering.

Since Goral et al. first introduced the radiosity method for a perfect diffuse environment in 1984 [1], a number of papers have been published. In these papers, the main focus is placed on efficiency of its implementation. Cohen and Greenberg proposed a method to approximate the calculation of form-factors by using a hemi-cube technique [2]. This hemi-cube method laid down the foundation of radiosity methods including occluded objects. Further, Cohen et al. improved the hemi-cube method using a substructuring strategy in order to more accurately determine radiosity on surfaces with complex intensity gradients [3]. Recently, Cohen again improved radiosity methods and proposed an interactive technique to progressively refine radiosity images [4].

In general, radiosity methods can be implemented in four steps:

- Subdivide surfaces into patches each of which has a constant radiosity;
- Calculate the form-factors from any patch to the remaining patches;
- Solve a set of simultaneous equations for radiosity;
- Render the image using an appropriate rendering algorithm.

Focusing on the first two steps, this paper proposes another radiosity method, again for an ideal diffuse environment. By carefully examining the previous results, it is observed that calculating form-factors involves a lot of visibility computation that requires a depth priority of patches to determine which patch is seen first. Therefore, if an implicit depth priority of patches is set up while polygons are divided into patches in the subdivision step, the second step may take advantage of the patch priority to speed up the visibility computation. This results in a 3-D subdivision scheme. By using an octree technique, this scheme not only subdivides polygons into patches but constructs an implicit octant priority. With the octant priority, in the second step for computing form-factors, a front-to-back list priority algorithm is used to efficiently determine visibility from a patch to the rest.

2. 3-D Subdivision for Polygons

Traditionally, an octree has been used to represent three dimensional objects in a hierarchic structure by subdividing the space into octants [5,6], each being either solid or empty. However, such a scheme is also suitable for subdividing polygons into patches. To derive accurate values for radiosity, a radiosity method

generally needs to subdivide polygons interactively, that is, the user needs to conduct the subdivision process in a way that polygon surfaces with high intensity gradients should be divided into fine patches. In the new radiosity method, interactive preprocessing is required to assure that the 3-D subdivision works properly. This is described first in this section.

2.1 Interactive Preprocessing

In the subdivision method to be stated later, it is assumed that all the polygons must be convex. Therefore, the first task in this preprocessing step is to break all polygon surfaces that are concave into convex subpolygons. This can be done automatically [19,20]; however, since the focus of the new radiosity method is on efficiency of form-factor computation, the decomposition of polygons is performed manually.

In general, polygon surfaces having high intensity gradients are those shadowed by the surfaces of other objects. On these polygons, intensity across the boundaries of shadows changes dramatically. In order to assure that each patch has an approximately equal radiosity, the shadowed polygons should therefore be divided into smaller patches than other polygons. To this end, these shadowed polygons are first divided into subpolygons along the boundaries of the shadows in the preprocessing. Then the 3-D subdivision uses the information of the boundaries between the divided polygons to subdivide them again in an adaptive way.

Computing shadows can be automated; however, it is expensive and difficult. Furthermore, for radiosity image display, accurate computation of shadows is almost impossible before the radiosity solutions are obtained. The objective to calculate shadows is to use the shadow information to guide the following 3-D subdivision. As a result, a user's careful estimate on shadows is sufficient to make the 3-D subdivision work fine.

2.2. Octree in Polygon Subdivision

Subdividing space to improve efficiency of image generation is not a new idea. The fast ray tracing algorithm proposed by Glassner 1984 was based on a space subdivision by an octree technique [11]. Other papers have been present in the literature to deal with space subdivision, especially octree subdivision [12,13,14]. However, papers that tackle radiosity using octree techniques have not been seen. Although Cohen et al. and Samet-Webber mentioned how to take advantage of hierarchical data structures to improve radiosity methods [3, 14], the detail of how to implement the octree technique in radiosity methods was not described.

Without loss of generality, it can be assumed that all polygons in a 3-D environment are enclosed in a unit cube whose edges are parallel to the axes, as shown in Figure 1a. In addition, it is also assumed that all the polygons are convex (see Section 2.1).

To construct an octree, a coding scheme needs to be chosen. Gargantini proposed a numbering scheme [15], in which *0* through *7* are used to name the nodes. Although Glassner had a better numbering system [11], this paper mainly refers to Gargantini's method since its features are adequate.

Starting with the original unit cube, the whole environment is subdivided into *8* sub-cubes by *3* planes P_x (x=*0.5*), P_y (y=*0.5*) and P_z (z=*0.5*), as shown in Figure 1a. Then for each sub-cube, another set of planes, for instance, x=*0.25*, y=*0.25* and z=*0.25*, are applied to subdivide the subcube into *8* smaller sub-cubes. The process continues for subsequent sub-cubes until a given criteria is met. Finally, the evironment space is divided into a sequence of small sub-cubes and an octree is built up. Figures 1b and 1c demonstrate a 2-level octree, where O_k represents a node which is called an octant, and k stands for the code.

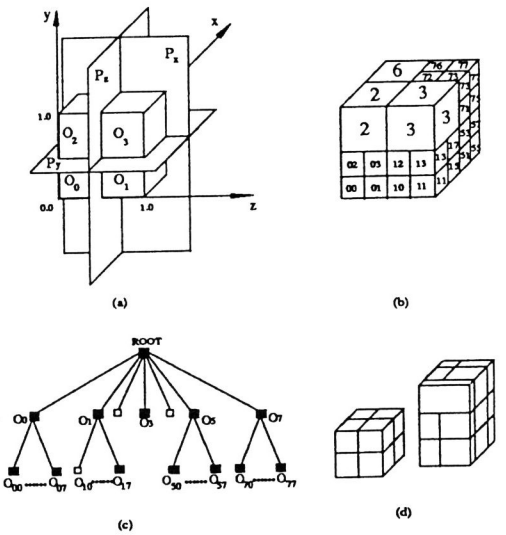


Fig.1. (a) The cube is divided by Px, Py and Pz into 8 smaller cubes, O0, ..., O7, called octants. (b) An example of the numbering system. (c) The octree corresponding to Fig.1b. (d) The 3-D subdivision for a scene based on the octree in Fig.1c.

Constructing an octree is not the whole story, but using the octree to subdivide polygons into patches is the main purpose. Polygon subdivision can be done by the following approach. As an octant is subdivided into 8 smaller octants by the dividing planes, for example, P_x, P_y and P_z in Fig.1., any polygon that crosses the boundaries of these smaller octants is simultaneously clipped against the dividing planes. The polygon subdivision goes on as these smaller octants are subdivided again. Finally, when the octree is completed, each of the resulting sub-polygons will lie entirely in one of the leaf octants. Such a sub-polygon is called a

patch. Once the octree has been formed, each polygon is eventually divided into a number of patches. If the criteria for subdivision termination is chosen accordingly, each of these patches will have an approximately constant radiosity. An example of 3-D subdivision for a simple scene is shown in Figure 1d.

2.3 Criteria for Subdivision termination

As indicated above, the subdivision process terminates when a given criteria is met. Several ways exist to specify the criteria.

One way is to constrain the size of the octant. When the octant is less than a specified size, the process of subdivision stops. In this case, if the specified size is sufficiently small, all polygons involved will eventually be subdivided into patches with the desired size because these patches are restricted in the final octants. However, with such criteria, subdivision may end up with some octants each of which contains more than one patches. In this case, the depth priority of these patches has to be determined in a different way. Fortunately, the Newell-Newell-Sancha algorithm's overlapping tests [8] can be used to accomplish this task and is efficient when the number of patches within an octant is small.

The second method for termination is to constrain both the size of octant and the number of polygons in it. When an octant is less than a specified size and contains only one patch, subdivision would terminate. The constraint may result in a problem — infinite subdivision because some joined polygons may not be separable. Theoretically, this method can be valid and perform a good ploygon subdivision if a very fine subdivision that eventually leads to a 3-D point in an octant is allowed. Fine subdivision is very expensive itself, let alone that it may produce a huge number of patches that the radiosity method can not handle within a reasonable time. As far as this is concerned, the following criteria is better than this one.

The third method, adopted in the new radiosity method, consists of two levels: when an octant contains one patch and its size is less than a number, say **SIZE1**, the subdivision terminates; or when an octant contains more than one patches and its size is less than another number, say **SIZE2**, the subdivision stops, where **SIZE1 > SIZE2**. The reason to use a two-level criteria is to allow for fine subdivision on high intensity gradients. This point can be clearly seen in the following two situations:

Since a shadowed polygon is divided into several subpolygons along the boundaries of shadows in the preprocessing step, the surfaces around the boundaries of shadows are made up of more than two polygons. By this criteria, those surfaces are eventually divided into more patches. Moreover, since a corner of an object consists of at least two polygons, a finer subdivision around the corner is also performed. Shadow boundaries and object corners are generally considered to have high intensity gradients. Therefore, this method allows for adaptive subdivision to some extent, at least in the two cases: boundaries of shadows and corners of objects.

In the last criteria, as dealt with in the first criteria, the depth priority of the patches within an octant is found by using the Newell-Newell-Sancha algorithm [8].

3. Depth Priority of Octants

Octant priority in terms of depth with respect to a viewer is a new concept introduced in this paper. With octant priority, the radiosity method can employ a front-to-back paint algorithm to perform hidden-surface removal in computing form-factors. Establishment of the octant priority is based on both the octree constructed in 3-D subdivision and the Schumacker algorithm.

3.1 Traditional Method for Cluster Priority

Schumacker proposed a fast algorithm for determining hidden surfaces as the viewer position constantly changes [7]. In the Schumacker algorithm, all polygons in a scene are collected into clusters of polygons, which are linearly separable, that is, any two clusters can be separated by a dividing plane. Then cluster priority that is independent of the viewer can be easily established. Fuchs et. al later developed the algorithm and proposed a data structure for tracing the cluster priority [16,17]. The data structure is called a *Binary Space Partitioning* tree (*BSP* tree). A classical example is shown in Figure 2.

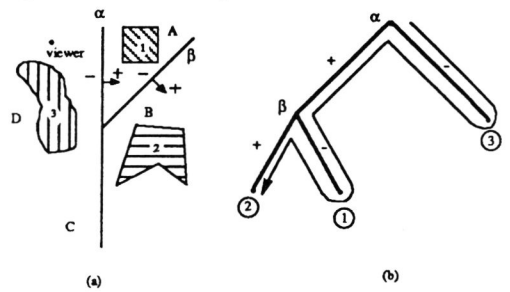


Fig.2. The Schumacker algorithm's cluster priority. (a) Two dividing planes α and β separate three clusters and divide space into three regions. (b) The corresponding *Binary Space Partitioning* tree.

The tree in Fig. 2b is a *BSP* tree of the scene in Fig. 2a. To determine the cluster priority, given a viewer, the *BSP* must be traversed from the root to the leaves. At each node, which branch, + or - , is visited first depends on the order in which clusters are displayed. If a front-to-back paint algorithm is used, the branch associated with the side of the dividing plane in which the viewer lies is visited first. In the example in Fig. 2. where the viewer is supposedly in area *D* the branch - is

visited first and then + at the first level, and at the second level, - first and + later. When the whole tree is visited, the order in which the leaves have been visited is the cluster priority. For the example, the priority is *3*, *1*, and *2*.

3.2 Methods for Octant Priority

As seen in the preceding section, the 3-D subdivision algorithm uses a sequence of planes to partition the space into octants and clip the polygons into patches. To use the Schumacker algorithm, the octants can be regarded as clusters of patches, and the planes as dividing planes for separating the clusters. Unlike the original Schumacker algorithm, however, the *BSP* tree is replaced by the octree produced in the 3-D subdivision, in which each node has *8* children instead of *2*.

To describe the following method conveniently, let us classify three kinds of nodes in an octree.

NULL node — has no child and contains no patch.
INTER node — 8 children one being a non-NULL.
TERMINAL node — has no children but patches.

The task to compute octant priority is to sort the **TERMINAL** nodes according to depth with respect to a given viewer. Based on the principle of the *BSP* tree, computing octant priority needs to traverse the octree in the order that the octants that are closesr to the viewer are visited first. At each **INTER** node, determining the ordering of its 8 children is the key part in forming octant priority.

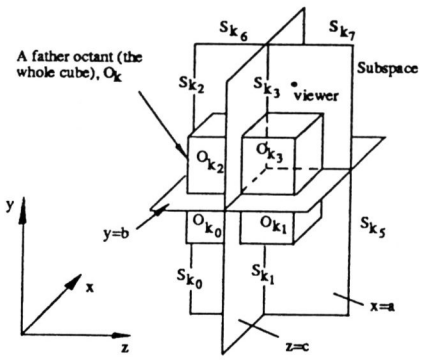


Fig.3. An example of how to determine octant priority given a viewer.

Let $O_k=(k, xm, ym, zm)$ present an **INTER** octant, where k is the code of the octant and (xm, ym, zm) is the center of the octant. Consider Fig.3. When a father octant O_k is subdivided into 8 smaller octants O_{k_0}, O_{k_1}, ..., O_{k_7} by *3* planes $x=a$, $y=b$, $z=c$ (a, b, c are constants), the space is also divided into *8* sub-spaces, S_{k_0}, S_{k_1}, ..., S_{k_7}. Note that according to the numbering scheme of the octree, k_i is in fact equal to $k*8+i$ $(i=0,...,7)$. If the octal system is used and let $k = (b_1b_2...b_r)_8$, then $k_i = (b_1b_2...b_ri)_8$.

Suppose that the viewer is in S_{k_3}. If only the plane $x=a$ is considered ignoring others, it is clear that any patches lying on the side of $x=a$ further from the origin can not obscure the patches on the side closer to the origin; therefore O_{k_0}, O_{k_1}, O_{k_2}, O_{k_3} should be visited first, and O_{k_4}, O_{k_5}, O_{k_6}, O_{k_7} later. Applying the same principle, when the planes $y=b$ and $z=c$ are considered in turn, each of the above two groups is sorted again, respectively, and finally the priority of the *8* octants can be determined as follows: O_{k_3}, O_{k_2}, O_{k_1}, O_{k_0}, O_{k_7}, O_{k_6}, O_{k_5}, O_{k_4}.

Since there are only *8* possibilities that the viewer can be located in the 8 sub-spaces, therefore, all *8* orderings, each corresponding to a sub-space in which the viewer lies, can be precomputed and stored in a table, called the *basic octant priority table* (**BOPT**). The complete table is listed in Appendix A.

At each **INTER** node, the following simple calculation is required to determine in which sub-space the viewer lies.

$$b = (view.x >= xm)*4 + (view.y >= ym)*2 + (view.z >= zm)$$

Then S_{k_b} is the sub-space in which the viewer is. Using b to index **BOPT**, the *8* children under the current **INTER** node will be visited in the order indicated by **BOPT**. When a **TERMINAL** node is visited, it is output and used in the form-factor computation which is stated in the next section.

4. Visibility in Form-factor Computation

To compute the form-factor, the visibility from a patch to the remaining patches must first be identified. Secondly, the form-factor from a patch, say p_e, to one of the remaining patches, say p_r, is calculated by counting the delta form-factors in the area projected by p_r onto the hemi-cube[2]. In this new radiosity method, with octant priority, the hemi-cube concept is kept with a slight modification and a front-to-back algorithm is adopted for the visibility computation.

4.1 Hemi-cube Technique

To calculate form-factors for any patch, an imaginary hemi-cube is first mounted on the patch as traditionally done. (For convention, the patch on which the hemi-cube is constructed is called the emitting patch, and the rest of patches the receiving patches.) Then the receiving patches are projected onto each of the five faces of the hemi-cube, respectively. Let $O_{k_i} = \{k_i, (mx_i, my_i, mz_i), P_{k_i}\}$ present a **TERMINAL** octant, where k_i is the code, (mx_i, my_i, mz_i) is the center coordinates of the octant and P_{k_i} is the set of patches that lie inside the octant. Suppose the emitting patch, p_e, is contained in the **TERMINAL** octant $O_{k_i} = \{k_i, (mx_i, my_i, mz_i), S_{k_i}\}$. Thus p_e belongs to P_{k_i}. Let (x,y,z) be the viewer

position which is located at the centre of p_e. By traversing the octree, the octant priority list with respect to (x,y,z) can be found. Let $OPL=\{O_{k_1},...,O_{k_n}\}$ represent this list where O_{k_i} has a higher priority than O_{k_j} if $i<j$ $(i,j=1,..., n)$.

As required by the hemi-cube technique, two 2-D arrays of delta form-factors are precomputed and any form-factor between two patches can be quickly calculated using a table lookup technique. The two arrays are called the *delta form-factor tables*. One corresponding to the upward-face of the hemi-cube is used to accumulate delta form-factors while patches are projected onto the upward-face of the hemi-cube. The other is used when patches are projected onto any of the sides. In addition, another *1*-bit 2-D array with the same size as the upward array of delta form-factors is used to assist the visibility computation. The array is named the *mask table* which can replace the item buffer used traditionally.

A conventional paint method paints all polygons to the frame buffer in the order of the farthest to the nearest. Instead, this new method adopts the reverse order, that is, patches are projected onto a face of the hemi-cube from the nearest to the farthest, i.e., from the patch with the highest priority to the one with the lowest priority.

When a patch is painted on a face of the hemi-cube, the corresponding pixel that is *0* in the mask table is set to *1* (the table is initialized at *0*). However, it is only at those pixels currently set to *1* in the *mask table* and covered by the patch's projection that the delta form-factors are accumulated. Note that the form-factor from the emitting patch to a receiving patch is computed right after the receiving patch is projected onto the hemi-cube. Thus there is no need to use an item buffer as used usually to store the projection of the receiving patch.

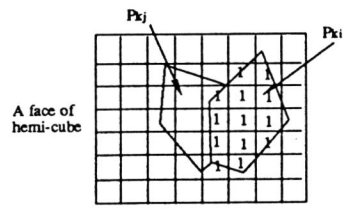
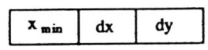


Fig.4. When P_{ki} hides P_{kj}, the delta form-factors only within the visible area of Pkj are counted.

Suppose that O_{k_i} and O_{k_j} are two **TERMINAL** octants in **OPL** each containing a single patch, p_{k_i} and p_{k_j}, respectively (for the more-than-one-patch cases, the Newell-Newell-Sancha algorithm is used), and that p_{k_i} partially hides p_{k_j} $(i<j)$. Then only part of p_{k_j} may be visible to the viewer. Since p_{k_i} is painted onto the hemi-cube first, the pixels covered by p_{k_i}'s projection are already *1*"s in the *mask table* when p_{k_j} is painted. Therefore, only the delta form-factors in the visible area are counted, while the delta form-factors in the overlapping area are not accumulated. Fig.4. illustrates this point.

4.2 Painting Patches onto the Hemi-cube

To accumulate delta form-factors, a scan-line algorithm using an *active edge table* [9] is used to locate the pixels covered by the projection of the 3-D patch on a face of the hemi-cube. In this algorithm, each edge of the polygon is represented by an edge structure as follows:



and, it is placed at the appropriate position in the *edge bucket* according to the maximum y of the edge. An *active edge table* (*AET*) contains those edges in pairs that are intersected by the current scan line. For example, when the current scan line

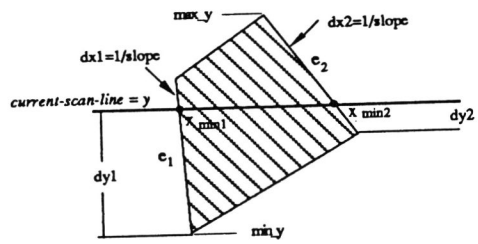
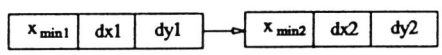


Fig.5. A san line with a pair of active edges.

is equal to y, as shown in Fig. 5., the *AET* includes the two edges entries:



To scan convert a patch, unlike the conventional scan-line method that paints each pixel between x_{min_1} and x_{min_2} to the frame buffer [9,10], the new method accumulates the delta form factors in the *delta form factor table* between the two columns that happen to correspond to x_{min_1} and x_{min_2}. Let $col_1 <=> x_{min_1}$, $col_2 <=> x_{min_2}$, *DFFT* be the precomputed *delta form-factor table* corresponding to the upward-face of the hemi-cube and *MT* be the *mask table*. For any patch, the key part of the painting technique is given in the following algorithm.

For each current-scan-line from the maximum y to the minimum y of the patch;

```
for(col=col1; col≤col2; col++)
    if(MT[current-scan-line][col] == 0){
        sum_delta_FF += DFFT[current-scan-line][col];
        MT[current-scan-line][col] = 1;
    }
```

After the above algorithm is repeated for each face of the hemi-cube, the final result of *sum_delta_FF* is the form-factor from the emitting patch to the patch being projected on the hemi-cube. Note that the delta form-factor table for the upward-face is different from that for the side-face, and normally these tables contain only the values for a quarter of the face. These factors are considered in the algorithm's implementation.

4.3 Advantages

Such a scheme for painting patches from front-to-back has a substantial advantage over a back-to-front painting scheme like the z-buffer. This approach can save computation, especially in a complex environment. According to the painting scheme, when the mask table is full of *1* 's[1], it is obviously unnecessary to compute the form-factors from the emitting patch to the remaining patches that have not been processed. This is because these remaining patches have lower priority and are completely hidden by the patches that have been painted onto the hemi-cube. Therefore, their form-factors are determined to be zero. This situation happens very frequently in an environment with crowded objects. Consider the example in Fig. 6. A big wall or polygon that appears in front of a small patch. All that the small patch can see is part of the wall (perhaps, a few other polygons around the boundary of the wall), and then only the wall receives light from the patch. This implies that the form-factors from the small polygon to the other polygon except the wall are zero. As a result, in this case, only the big polygon is projected and the delta form-factors covered by the projection are accumulated, simply assuming that the form-factors to the other patches are zero.

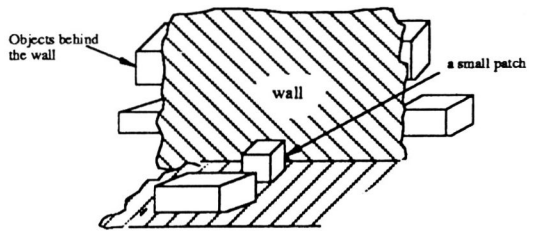


Fig.6. A small patch on a side of the cube can only see the wall that it faces. In other words, other objects receive no light from it.

4.4 Overview of the Algorithm

The following pseudo code outlines the new radiosity method using octant priority established in the 3-D subdivision.

```
Pre-process Polygons;
Construct the octree and
subdivide polygons into N patches ⇒ {p1,...,pN};
for ( i=1; i<=N; i++ ) {
  Form the octant priority list  ⇒ OPL={Ok1,...,OkM}
    with respect to the center of pi;
  for ( j=1; j≤M; j++ ) {
    if ( MT is full ) goto LOOP;
    Using Newell-Newell-Sancha
    sort patches in Pkj  ⇒ {pj1,...,pjL};
    for ( k=1; k≤L; k++ ) {
      project pjk onto each face of the hemi-cube in turn;
      Scan the projection of pjk from max_y to min_y:
      for ( col=col1; col≤col2; col++ )
        if ( MT[current-scan-line][col] == 0 ) {
          FF[i][jk] += DFFT[current-scan-line][col];
          MT[current-scan-line][col] = 1;
        }
      }
    }
    LOOP:
  }
  Solve the radiosity equation to obtain: B[1],...,B[N];
  Render images using B[1],...,B[N];

  MT — mask table
  FF — form-factor matrix
  DFFT — delta form-factor table
```

5. Implementation and Analysis of Results

The new radiosity method proposed in this paper has been implemented on a Color Sun 3/60 and the I^2S color monitor and produced several radiosity images. Five scenes with different complexity have been modelled and rendered by this method. Their images with performance results are shown in Figs. 7 to 12.

To highlight the advantages of this new method, two comparisons are used to analyze its performance. First, the new method using a front-to-back paint algorithm is compared with the hemi-cube method using the z-buffer technique. Second, a scene of crowded objects is compared with a scene of sparse objects. There are two terms to measure performance, the execution time and the number of octants processed.

A comparison of the two methods can be converted to a comparison of a front-to-back paint algorithm with the z-buffer. By careful examination, the paint algorithm has an overhead consisting of returning pointers in traversing the octree to obtain octant priority, and performing a simple calculation at each node to determine which branch to visit next. On the other hand, the z-buffer does not need to determine depth priority of patches, but has the extra computation for the z values, comparing, and updating the frame buffer and the z buffer at each pixel. Since a front-to-back paint algorithm uses a 1-bit mask to determine whether a pixel is occupied, extra comparison operations are needed at

[1] A counter, which counts the number of pixels with 1, can be used for determining the fullness of 1's in the mask table

each pixel too. However, this can be compensated by the saving in the situations where a number of patches are completely obscured by other patches and then discarded (This has been discussed in Section 4.3 and will be emphasized again in the following timing analysis). Thus, both methods have overhead in different places. The trade-off is that if the size of the octree is smaller than the number of pixels, the new method would have less overhead than the z buffer. From the six examples shown in Figs. 7 to 12, in general, a four-level octree is enough to produce a fine polygon subdivision. In the worst case, the size of the octree is $8+8^2+8^3+8^4=4632$. However, the resolution of the hemi-cube is usually from 100*100 to 300*300. Obviously, the former is less than half of the latter.

By further investigation, the overhead imposed by the octant priority algorithm can be still reduced in the following situations.

Recall that in Section 3.2, once a **TERMINAL** octant has been visited in traversing the octree, the octant can be immediately output to the front-to-back paint algorithm since the **TERMINAL** octant must have higher priority than those not visited. This implies that traversal of an octree can dynamically interact with the painting process. By the interaction, an **INTER** octant being visited currently can be checked to see if it is behind the viewer in advance. If so, the whole subtree under the **INTER** node can be discarded without traversing down through it. Definitely, this would reduce a significant amount of overhead due to octree traversal.

Another situation that may lessen the overhead is that when the 1-bit mask is full of 1's, the paint algorithm stops further processing. In this case, there is no need to continue to traverse the remaining portion of the octree. Therefore, a lot of time is saved by cutting off these remaining branches.

Since the view position is located in the center of an emitting patch that appears inside the environment, it is usual that the viewer only can see part of the scene, most objects being either behind it or behind the other nearer objects. As a result, the savings occurring in the above two situations is significant for most scenes.

Furthermore, traversing the octree is performed only once for all five faces of the hemi-cube, while the z buffer has to be gone over five times for different faces of the hemi-cube. This implies that the overhead in traversing the octree is small compared with that in the z buffer.

From the point of view of the number of the octants processed, the advantages of the front-to-back paint algorithm can be seen easily. The Z buffer always has to process all the patches in the environment (except the back faces of an object) even if some patches are completely obscured by the nearer patches. Therefore, its complexity is constant for any emitting patch. By contrast with the z buffer, the complexity of the front-to-back paint algorithm varies depending on the location of the viewer or the emitting patch. In a crowded environment as shown in Fig. 11, the average number of octants per face that have to be processed by the paint algorithm is 125 (for the z buffer, it is always 932, the maximum number. Although the z-buffer does not use the octant concept, the number 932 in effect represents the equivalent number of patches). Even for the environment shown in Figs 8 to 10, in which objects are distributed loosely, approximately half of the emitting patches use only about 230 octants. Moreover, there is always at least one face of the hemi-cube that needs only 20 octants to cover it on average, that is, an average of 20 octants can be seen from this face.

From the Algorithm described in Section 4.4, this kind of saving is reflected by the number of the loop running through **OPL**. If the *mask table* is full very quickly, only a small number of octants in OPL will be processed, the rest being discarded. Obviously, the smaller the number of octants processed, the faster the algorithm. The statistical results corresponding to the images are shown in Figs. 7 to 12.

6. Conclusions

This paper proposes a new radiosity method which subdivides polygons into patches as an octree is established. Based on the Schumacker algorithm and the BSP tree, traversing the octree produces octant priority that can be used for fast visibility computation in computing form-factors. A paint method that paints patches from near to far is developed to accomplish the visibility computation.

As indicated above, the main advantage in this new scheme for radiosity image synthesis is to avoid unnecessary computation. The front-to-back method can save much effort in computing form-factors that are in fact zero, especially in the case of a complex environment with crowded objects.

The major disadvantage is that the 3-D subdivision may result in a number of odd-shaped patches that will create hemi-cube aliasing. To reduce the artifact, a high resolution hemi-cube is required. However, this will unfortunately cancel off the savings gained from using this octant priority method. This difficulty has been overcome in [21].

Furthermore, the new method has also proposed an adaptive subdivision technique for polygons. But it is not sufficient at its current stage. More work needs to be done in order to accommodate the adaptive subdivision efficiently. Cohen's substructure techniques for adaptive subdivision are expected to be added into this new method to accomplish more accurate polygon subdivision.

To conclude, although some problems exist, this proposed radiosity method has attempted to open up an alternative direction for radiosity methods instead of being forced to use the traditional z buffer.

7. Acknowledgements

Special thanks to Myrias Corporation in Edmonton for letting us to use the 64-processor Myrias PSP-2 to test the programs. We would like to thank Mark Green and Chris Shaw in Dept. of Computing Science at University of Alberta for their powerful graphics packages, particularly Chris's nice arm chairs.

REFERENCES

[1] Goral, C.M., Torrance K. E., Greenberg, D. P. and Battaile, B., "Modeling the Interaction of Light Between Diffuse Surfaces", *Proc. of SIGGRAPH' 84*, July 1985, pp. 213-222.

[2] Cohen, M.F. and Greenberg, D.P., "The Hemi-cube: A Radiosity Solution For Complex Environments", *Proc. of SIGGRAPH' 85*, July 1985, pp. 31-40.

[3] Cohen, M.F., Greenberg, D.P., Immel, D.S. and Brock, P.J., "An Efficient Radiosity Approach for Realistic Image Synthesis", *IEEE CG & A*, March 1986, pp. 26-35.

[4] Cohen, M.F., Chen S.E., Wallace, J.R., Greenberg, D.P., "A Progressive Refinement Approach to Fast Radiosity Image Generation," *Proc. of SIGGRAPH' 88*, August 1988, pp. 75-84.

[5] Yamaguchi, K., Kunii, T.L., Fujimura, K., "Octree-related Data Structure and Algorithms", *IEEE CG & A*, January 1984, pp. 53-59.

[6] Doctor, L.J., Torborg, J.G., "Display Techniques for Octree-Encoded Objects", *IEEE CG & A*, July 1981, pp. 29-38.

[7] Schumacker, R.A., Brand, B., Gilliland, M. and Sharp, W., "Study for Applying Computer-generated Images to Visual Simulation", *U.S. Air Force Human Resource Lab Tech. Rep.*, AFHRL-TR-69-14, Sept. 1969, NTIS AD 700 375.

[8] Newell, M.E., Newell, R.G., and Sancha, T.L., "A New Approach to Shaded Picture Problem", *Proc. ACM National Conference*, 1972, pp. 443-450.

[9] Foley, J. D. and Dam, A. V., *Fundamentals of Interactive Computer Graphics*, Addison-Wesley Publishing Company, 1983.

[10] Rogers, D. F., *Procedural Elements for Computer Graphics*, McGraw-Hill Book Company, New York, 1985.

[11] Glassner, A.S., "Space Subdivision for Fast Ray Tracing", *IEEE CG & A*, October 1984, pp. 15-22.

[12] Fujimoto, A., Tanaka, T., and Iwata, K., "ARTS: Accelerated Ray-Tracing System", *IEEE CG & A*, April 1986, pp. 16-26.

[13] Kaplan, M.R., "Space-Tracing, a Constant Time Ray-Tracer", *SIGGRAPH 85 Tutorial*, San Francisco, July 1985.

[14] Samet, H., Webber, R.E., "Hierarchical Data Structures and Algorithms for Computer Graphics", *IEEE CG & A*, July 1988, pp.59-75.

[15] Gargantini, I., "Linear Octree for Fast Processing of Three-Dimensional Objects", *Computer Graphics and Image Processing*, Vol. 20, No. 4, 1982, pp. 265-274.

[16] Fuchs, H., Kedem, Z.M., and Naylor, B.F., "On Visible Surface Generation by A Priori Tree Structures", *Proc. SIGGRAPH 80*, Vol. 14, No. 3, July, 1980, pp. 124-133.

[17] Fuchs, H., Abram, G.D., and Grant, E.D., "Near Real-Time Shaded Display of Rigid Objects", *Proc. SIGGRAPH 83*, Vol. 17, No. 3, July 1983, pp. 65-69.

[18] Reynolds, R.A., Gordon, D., and Chen, L-S, "A Dynamic Screen Technique for Shaded Graphics Display of Slice-Represented Object", *Computer Vision, Graphics, and Image Processing*, Vol. 38, 1987, pp. 275-298.

[19] Schachter, B., "Decomposition of Polygons into Convex Sets", *IEEE Transactions on Computer*, Vol. c-27, No. 11, Nov., 1978, pp.1078-1082.

[20] Joe, B, and Simpson, R.B., "Triangular Meshes for Regions of Complicated Shape", *International Journal for Numerical Methods in Engineering*, Vol. 23, 1986, pp. 751-778.

[21] Wang, Y., "Image Synthesis Using Radiosity Methods", *Ph.D. Thesis*, University of Alberta, 1990.

Appendix A The Basic Octant Priority Table

```
BOPT [8] [8] = {

        {0, 1, 2, 3, 4, 5, 6, 7},
        {1, 0, 3, 2, 5, 4, 7, 6},
        {2, 3, 0, 1, 6, 7, 4, 5},
        {3, 2, 1, 0, 7, 6, 5, 4},
        {4, 5, 6, 7, 0, 1, 2, 3},
        {5, 4, 7, 6, 1, 0, 3, 2},
        {6, 7, 4, 5, 2, 3, 0, 1},
        {7, 6, 5, 4, 3, 2, 1, 0}

}
```

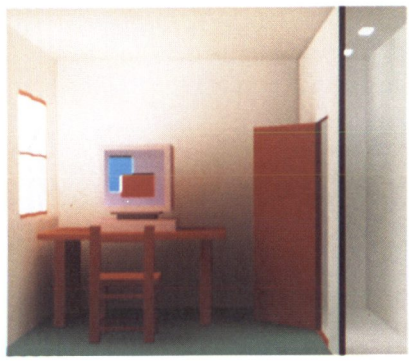


Fig.7.

	# of patches	Time for 3-sub (sec.)	Time for form-factor (min)	Render time (min.)	Max # of octants	Level of octree	Octants per face	Octants per hem-cube
Z Buffer	1098	16	435	5.8	872	5	872	4350
Octant Priority	1098	16	290	4.9	872	5	499	2495

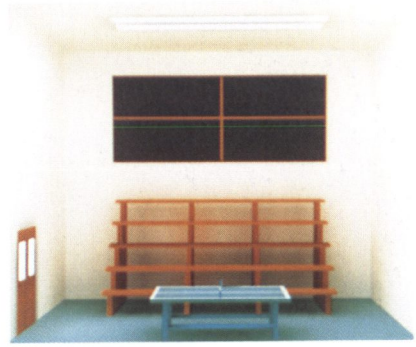


Fig.8.

	# of patches	Time for 3-sub (sec.)	Time for form-factor (min)	Render time (min.)	Max # of octants	Level of octree	Octants per face	Octants per hem-cube
Z Buffer	1059	12	496	5	481	4	481	2405
Octant Priority	1059	12	354	4.8	481	4	230	1150

Fig.9.

	# of patches	Time for 3-sub (sec.)	Time for form-factor (min.)	Render time (min.)	Max # of octants	Level of octree	Octants per face	Octants per hem-cube
Z Buffer	1774	14	594	5.4	492	5	492	2264
Octant Priority	1774	14	384	4.9	492	5	190	950

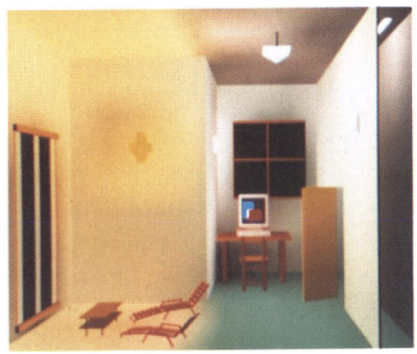


Fig.10.

	# of patches	Time for 3-sub (sec.)	Time for form-factor (min.)	Render time (min.)	Max # of octants	Level of octree	Octants per face	Octants per hem-cube
Z Buffer	1774	14	594	5.4	492	5	492	2264
Octant Priority	1774	14	384	4.9	492	5	190	950

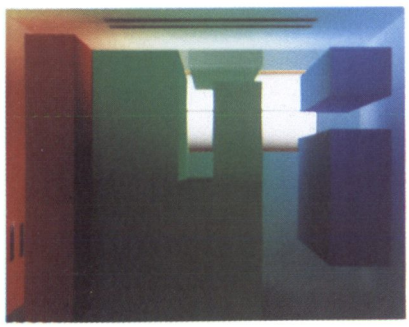


Fig.11.

	# of patches	Time for 3-sub (sec.)	Time for form-factor (min)	Render time (min.)	Max # of octants	Level of octree	Octants per face	Octants per hem-cube
Z Buffer	1170	15	526	6.2	932	4	932	4660
Octant Priority	1170	15	194	4.6	932	4	125	725

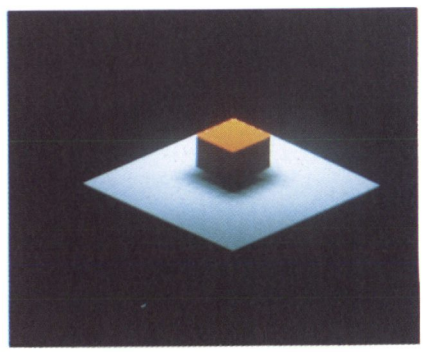


Fig.12.

	# of patches	Time for 3-sub (sec.)	Time for form-factor (min.)	Render time (min.)	Max # of octants	Level of octree	Octants per face	Octants per hem-cube
Z Buffer	441	8	100	2	241	5	241	1205
Octant Priority	441	8	84	2	241	5	184	920