A New Space Partitioning for Mapping Computations of the Radiosity Method onto a Highly Pipelined Parallel Architecture

Li-Sheng Shen, E. Deprettere and P. Dewilde

ABSTRACT Despite the fact that realistic images can be generated by ray-tracing and radiosity shading, these techniques are impractical for scenes of high complexity because of the extremely high time cost. Several attempts have been made to reduce image synthesis time by using parallel architectures, but they still suffer from communication problems. In this paper, we present a new space partitioning which is adaptive to the local environment seen by a bundle of rays. Two tracking mechanisms are embedded to guarantee adaptation. When using a shared memory parallel architecture, the communication load between the host and the PEs can be alleviated with this approach. Furthermore, the partitioning provides a better balancing between processing throughput and I/O bandwidth which will enhance the pipelinability of computations, especially when a high speed cache memory is allowed for each PE. Combining those factors, a highly pipelined parallel architecture can be used to accelerate computations in ray-tracing and radiosity methods. The technique has been tested on different scenes with randomly generated patches in a 2D setting. When compared with the conventional technique, promising results have been observed. This technique can be easily extended to 3D.

1 Introduction

In the fields of solid modeling and computer graphics, the most promising method to render realistic images is based on ray-tracing and radiosity shading, because it can account for all important physical effects occurring in the scene. Ray tracing, first described by Appel [1], cannot alone account for global illumination, since that is dependent on all objects in the environment. Although some fairly realistic images have been obtained using solely ray-tracing, this technique ignores the interreflection of light between diffusely reflecting surfaces. In 1984, Goral [2] introduced a radiosity method borrowed from radiative heat transfer to model the interreflection between diffusely reflecting surfaces. After that, substantial efforts have exerted on it: Cohen [3,4] presented the hemi-cube algorithm to calculate form-factors by projecting patches onto a hemi-cube and presented the substructuring technique to render areas possessing high intensity gradients by adaptive patch subdivision. Immel [5] extended the radiosity method to treat an environment consisting of non-diffuse surfaces. Wallace [6] proposed a combined radiosity and ray-tracing method to render a complex environment; they called it the two-pass approach.

Despite the fact that the two-pass approach can generate realistic images, it is impractical for scenes of high complexity because of extremely high time and storage costs. In ray-tracing, a large amount of intersection computations is needed. In fact, it is proportional to the number of screen pixels (even more when supersampling is used to accom-

modate for anti-aliasing, or when secondary rays are taken into account) and the number of surfaces in the environment. This situation becomes even worse if form-factor computations are also based on ray- tracing, because the number of intersections computed for calculating form-factors is at the order of $(N^2 \times R)$, where N is the number of patches in the environment and R is the number of rays on a hemisphere. Apart from this computation cost, the storage capacity needed to store the form-factor matrix is extremely high. Indeed, assuming the form-factor matrix is 90% sparse, it still requires 400 gigabytes of storage for an environment consisting of 1M patches.

To overcome the storage problem in the radiosity method, Cohen [7] proposed a progressive refinement approach to reduce the storage capacity to $O(N)$. To overcome the computation problem in ray-tracing, two main strategies have been devised. The first strategy is to exploit coherence. One kind of coherence is called ray coherence: Heckbert [8] proposed a beam-tracing algorithm in which a beam of rays, instead of a single ray, traces an environment consisting of polygons. Speer [9] also exploit this coherence in their coherent ray-tracing algorithm. Another kind of coherence is called spatial coherence. This sort of coherence is exploited in space partitioning techniques to reduce the number of patches to be considered. Many different encoding schemes for space partitioning have been proposed: octree [10], binary tree [11] and voxel [12]. Basically, the 3D object-space is subdivided into cells based on a specific encoding scheme. One ray is shot and tested against each patch in the first cell. If the ray hits some patches in this cell, a nearest distance test must be invoked to determine the intersection point. Otherwise, the next cell (or cells) will be traversed and tested again until there is a hit or the boundary of the data structure is reached. After that, the procedure proceeds to the next ray and iterates again. One very promising result of the space partitioning technique is that the performance is nearly independent of the scene complexity.

The second strategy relies on parallel processing. Parallel architectures can be classified into three classes: processing without dataflow [13,14], processing with ray dataflow [15,16], and processing with object dataflow [17,18]. The drawback of the first class is that it cannot render complex scenes due to the limited size of local memories. As for the second class, a ray might pass through a number of processors such that the efficiency of the system will be degraded due to this communication overhead. This can be resolved by assigning rays to processors as the third class does, but it requires an efficient way to access the whole database. For comparison purpose, we refer to the third class as the conventional technique throughout this paper.

Ray tracing has already been used as a method for determining the form-factors in the radiosity method. We have capitalized on a hardware oriented two-pass approach [19] in which both the preprocess and the postprocess are ray tracing based. Unfortunately, the two strategies stated above can not be applied successfully in this case. This is due to the following:

- In the radiosity method, each patch in the environment will be chosen as a source to start a ray-tracing algorithm to compute form-factors. For each source patch, a costly depth-sorted list of polygons must be constructed. The time spent on constructing such a list will be at the order of $O(N^2)$ at worst, where N is the number of polygons after splitting which is larger than the original number [20].

- Since any patch in the environment will become the source patch, the architecture with ray dataflow is certainly undesirable. As for the architecture with object dataflow, many regions can be processed in parallel, but each processor still works on a serial ray base within its own region. Consequently, a patch might be loaded

many times from the global or local memory because the neighboring rays are most likely to traverse similar cells and even hit the same patch. This accounts for a lot of waste in the sense of communication.

In this paper we present a new space partitioning technique to alleviate the communication problem stated above. Moreover, two tracking mechanisms are considered to exploit the ray coherence with low cost.

The outline of the paper is as follows. After defining some basic terms in Section 2, we present the main technique in a two-dimensional setting and suggest two possibilities to extend it to 3D in Section 3. We formalize the intersection computation time and the speedup factor of this technique in Section 4. In Section 5, the results from a number of scenes with randomly generated patches are shown and compared with the conventional technique.

2 Basic Terms

In this section we define some basic terms to facilitate the discussion in subsequent sections.

Definition 2.1 (Polygonal Patch) A polygonal patch P with v vertices ($v \geq 3$) is defined by the inner area of a convex planar polygon with simple or non-self-intersecting boundary. It will be represented by the coordinates of its vertices $V_i = (x_i, y_i, z_i)$, $i = 1, 2, \ldots, v$, the coordinates of its center point $O = (x_c, y_c, z_c)$, and the unit normal vector $\vec{n}$ of the plane enclosing P, i.e.,

$$P = \{V_i, O, \vec{n} \mid i = 1, 2, \ldots, v\}.$$

Usually, a polygonal patch can only emit light energy into its front side, i.e., the half-space containing $\vec{n}$. Similarly, a polygonal patch can only receive light energy from its front side.

Definition 2.2 (Backward Polygonal Patch) Given polygonal patches P and P', P' is said to be a backward polygonal patch for P when it is not located at the front side of P.

Definition 2.3 (Spherical Bounding Box of a Polygonal Patch) Let

$$P = \{V_i, O, \vec{n} \mid i = 1, 2, \ldots, v\}$$

be a polygonal patch with v vertices, and let the polar coordinates of vertex V_i be r_i, Θ_i, and Φ_i. The spherical bounding box B of P is defined as the region bounded by six surfaces represented in the polar coordinates: $r = r_{min}, r = r_{max}, \Theta = \Theta_{min}, \Theta = \Theta_{max}, \Phi = \Phi_{min}, and\ \Phi = \Phi_{max}$, where the min and max denote the minimum and maximum value of the corresponding polar coordinates.

Definition 2.4 (Spanning Angle and Flatness of a Polygonal Patch) Let B, which is the region bounded by six surfaces represented in the polar coordinates: $r = r_{min}, r = r_{max}, \Theta = \Theta_{min}, \Theta = \Theta_{max}, \Phi = \Phi_{min}$, and $\Phi = \Phi_{max}$, be the spherical bounding box of a polygonal patch P. The spanning angles ω_θ, and ω_ϕ of P are defined as $\omega_\theta = \Theta_{max} - \Theta_{min}$, and $\omega_\phi = \Phi_{max} - \Phi_{min}$. The flatness η of P is defined as $\eta = r_{max} - r_{min}$.

Definition 2.5 (Sector) Let a hemisphere ray $\vec{r_i}$ be represented by its origin that is the centroid of the hemisphere, and the unit direction vector

$$\vec{a_i} = (\sin\Phi_i \cos\Theta_i, \sin\Phi_i \sin\Theta_i, \cos\Phi_i).$$

A sector bounded by the boundary rays $\vec{rb_1}, \vec{rb_2}, \vec{rb_3}$, and $\vec{rb_4}$, i.e., the hemisphere rays with the following unit direction vectors:

$$(\sin\Phi_1 \cos\Theta_1, \sin\Phi_1 \sin\Theta_1, \cos\Phi_1),$$
$$(\sin\Phi_2 \cos\Theta_2, \sin\Phi_2 \sin\Theta_2, \cos\Phi_2),$$
$$(\sin\Phi_2 \cos\Theta_1, \sin\Phi_2 \sin\Theta_1, \cos\Phi_2),$$
$$(\sin\Phi_1 \cos\Theta_2, \sin\Phi_1 \sin\Theta_2, \cos\Phi_1),$$

is defined as the set of hemisphere rays $\vec{r_i}, i = 1, \ldots, r.$, with the following properties:

- The angular component Θ_i of the unit direction vector $\vec{a_i}$ is bounded by the real numbers Θ_1 and Θ_2, i.e., $\Theta_1 \leq \Theta_i < \Theta_2$.
- The angular component Φ_i of the unit direction vector $\vec{a_i}$ is bounded by the real numbers Φ_1 and Φ_2, i.e., $\Phi_1 \leq \Phi_i < \Phi_2$.

Definition 2.6 (Shell) Let V be a sector bounded by $\vec{rb_1}, \vec{rb_2}, \vec{rb_3}$, and $\vec{rb_4}$ as defined in the definition. A shell ∇ in V is defined as the region bounded by six surfaces represented in the polar coordinates: $r = r_1, r = r_2, \Theta = \Theta_1, \Theta = \Theta_2, \Phi = \Phi_1$, and $\Phi = \Phi_2$. It can be represented as,

$$\nabla = \{\vec{rb_1}, \vec{rb_2}, \vec{rb_3}, \vec{rb_4}, \zeta_1, \zeta_2\}$$

where ζ_1 and ζ_2 denote the surfaces represented in the polar coordinates: $r = r_1$ and $r = r_2$, respectively.

Definition 2.7 (Spanning Angle and Depth of a Shell) Let

$$\nabla = \{\vec{rb_1}, \vec{rb_2}, \vec{rb_3}, \vec{rb_4}, \zeta_1, \zeta_2\}$$

be a shell as defined in the definition. The spanning angles Ω_θ, and Ω_ϕ of ∇ are defined as $\Omega_\theta = \mid \theta_1 - \theta_2 \mid$, and $\Omega_\phi = \mid \Phi_1 - \Phi_2 \mid$. The depth Γ of ∇ is defined as $\Gamma = \mid r_1 - r_2 \mid$.

Definition 2.8 (Degree of Local Coherence) For a prescribed discretization (and hence a prescribed number of rays confined to a hemisphere) of the surface of a hemisphere, the degree of local coherence of a patch P with respect to a point O is defined as the number of rays shot from O through the hemisphere placed over O, that hit P.

Definition 2.9 (Active Patch) Let O be a point over which a hemisphere is placed to start a ray-tracing algorithm. An active patch with respect to the point O is a patch whose degree of local coherence with respect to the point O is nonzero.

Definition 2.10 (Average Degree of Local Coherence) Consider an environment which consists of a number of patches. Let the degree of local coherence of active patches with respect to a point O over which a hemisphere is placed to start a ray-tracing algorithm, be $D_1, D_2, \ldots, D_N$. The average degree of local coherence of the environment with respect to the point O, which we denote $ADLC$, is defined as,

$$ADLC = \frac{\sum_{i=1}^{N} D_i}{N}$$

3 The Main Technique

In this section, a technique relying on an adaptive tracking mechanism to construct a shell structure which can support a highly pipelined parallel architecture for the radiosity method is described. For simplicity, we first explain it in a two-dimensional setting. Then we propose two possibilities to extend it to 3D. We already mentioned the communication problems which may arise from parallel architectures. This motivates us to think out a new space partitioning technique featuring the following properties:

- The partitioning of space should be consistent with the direction of rays such that a ray does not travel through many partitions.
- The partitioning of space should be adaptive to the local environment seen by rays such that each active patch needs to be loaded only once from the memory that stores it.

This is just what we have called the shelling technique. Conceptually, it can be viewed as a method that strives to construct an ideal structure as shown in Figure 1. The half-space seen by the source patch is first subdivided into sectors. Then, a sector is subdivided into a number of shells. Ideally, a shell can be matched with the the active patch which actually defines the shell[1]. When this structure has been constructed, the bundle of rays in a sector is shot and tested against the active patch within the shell currently considered. With the ideal structure considered here, each active patch needs to be loaded only once as desired. Furthermore, when assigning sectors to processors, it is clear that a ray is always processed by one processor.

Besides the saving in communication cost, the shelling technique has a further advantage is that the resulting shell structure is amenable to a highly pipelined parallel architecture. To see this, recall first that for achieving high throughput, the conventional technique can exploit concurrency at the patch side, i.e., each ray traced can be tested against all the patches within a cell in a pipelined fashion. However, the following two factors will jeopardize the pipeline efficiency[2]:

- When using a shared memory parallel architecture, it is very time-consuming to load a patch from the memory. This will preclude the applicability of pipelined computations.
- In general, a ray can only hit one patch at most. This implies that the conventional technique can only fill up pipeline stages with irrelevant hidden patches, which would be meaningless.

On the contrary, the shelling technique exploits concurrency at the ray side, i.e., a bundle of rays can be tested against each relevant patch within a shell in a pipelined fashion. With this approach, the two detrimental factors mentioned above do not exist. This is because:

- When using a shared memory parallel architecture, the shelling technique provides a better balancing between processing time and I/O time such that the pipeline

[1] A shell is said to be matched with a patch when the shell is aligned perfectly with the visible part of the patch. Hence one necessary condition for this definition is that the spanning angle of the shell must be equal to the degree of local coherence of the patch.

[2] The pipeline efficiency is used to measure the performance of a pipeline and is defined as the ratio of busy time-space span over the total time-space span.

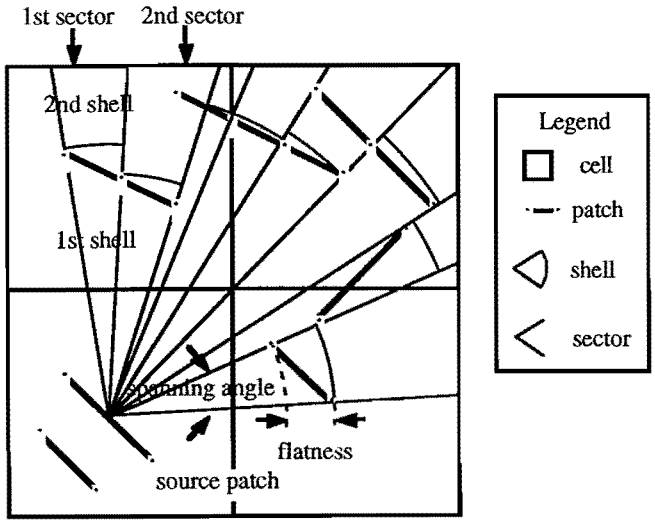

Fig. 1. The ideal shell and sector structures in the shelling technique.

efficiency will be higher than the conventional technique. This property will be very promising when a high speed cache memory is allowed for each PE. This is because the intersection computation time between a bundle of rays and a patch can readily balance with the loading time of the next patch for an environment with reasonable degree of local coherence.

- In general, a patch can be hit by many rays, it is desirable to fill up pipeline stages with those rays.

However, there are two major problems in constructing the shell structure. First, this specific structure cannot be chosen as an original data structure because it varies with the source patches. Second, it is costly to match a shell with the active patch which defines the shell. A simple solution to the first problem is to superimpose this structure on an uniform grid data structure as shown in Figure 1 such that relevant patches can be found by working on this well-developed data structure. As for the second problem, the actual degree of local coherence of an active patch can be estimated from its neighborhood with low cost. This is due to the fact that in realistic situations, neighboring patches appear to possess similar shapes. This can be understood from the following:

- Neighboring patches most likely belong to the surface of the same object. For an object with smooth surface, neighboring patches are often similar in some sense.
- When using the substructuring technique as proposed in [4] to account for the effect of high intensity gradients, it is necessary to provide a reasonable initial subdivision for the surfaces of a scene such that neighboring patches will become similar indeed.

Exploiting these facts, an adaptive tracking mechanism which employs the neighborhood information to estimate the degree of local coherence is devised. In the next section,

we shall discuss the shelling technique with this tracking mechanism. We shall call it the stochastic shelling technique hereafter.

3.1 Main Procedures

From the above discussions, the stochastic shelling technique can be broken into four main procedures: preprocessing, screening, tracking, and intersection computation.

- Preprocessing

  The main purpose of this step is to build a data structure on which the shell structure can be superimposed such that relevant patches can be searched efficiently. We have chosen the uniform grid cells with macro-regions proposed in [21] as our underlying data structure.

- Screening

  Because the intersection computation for each patch will proceed with a bundle of rays in the shelling technique, it is necessary to screen out irrelevant patches before computing intersection points. This step is similar to that of searching small sets of surfaces by cell traversal in the conventional space partitioning technique, but additional angle comparisons are necessary to cull patches which reside within (at least partly) the shell currently considered. Meanwhile, backward patches which are located at the back side of a source patch will be screened out.

- Tracking

  After the screening step, the spanning angle and flatness of relevant patches in the shell are computed and they will contribute to the $ADLC$ and the average flatness of the shell. We use the $ADLC$ of the first shell in a sector to set up the next sector. All the shells in the next sector can be constructed by subdividing the sector along the radial direction. In order to pass neighborhood information smoothly, we choose a constant shell depth, but shells can be further subdivided based on the $ADLC$ and the average flatness coming from one of their neighboring shells to reduce the number of intersection computations. In the case when there is no such neighborhood information, the algorithm will choose default values. Conceptually, the way of tracking is to traverse a region, which is most probably empty as match as possible in the radial direction but as narrow as possible in the angular direction. On the contrary, it will traverse a region which is most probably nonempty as narrow as possible in the radial direction but as match as possible in the angular direction.

- Intersection Computation

  When the local shell structure has been constructed, the bundle of rays in a sector proceeds to compute the intersection points with each relevant patch within the shell currently considered. Only remaining rays leaving one shell will enter the next shell and need to be processed again until there is no ray left or the boundary of the data structure is reached. Intersection computation is the most time-consuming step because a huge number of intersection points need to be computed, but the regular and repeatedly executed nature of this operation makes it very appealing for a pipelined circuit. For polygonal patches, we can adopt the circuit as proposed in [22] which is built from commercial chips or the circuit as proposed in [23] which is built from pipelined Cordic processors.

3.2 Extension to 3D

It is easier to explain the concept and the nature of the shelling technique in 2D. Nonetheless, it is only the 3D version that is of major interest to us. In the following, we suggest two possibilities to extend it to 3D.

- Intuitively, we can extend the 2D version directly to 3D. In 3D, a shell is defined by three polar coordinates instead of two. In order to pass neighborhood information smoothly, we must fix the shell depth and one of the two angles. Just like in 2D, a shell can be subdivided further in radial and two angular directions based on neighborhood information. However, we have found two difficulties for this approach. First, it is difficult to compute the spanning angle wj because max and min do not necessarily occur at vertices of a patch. Second, the spherical bounding box of a patch might be a loose convex hull of this patch.

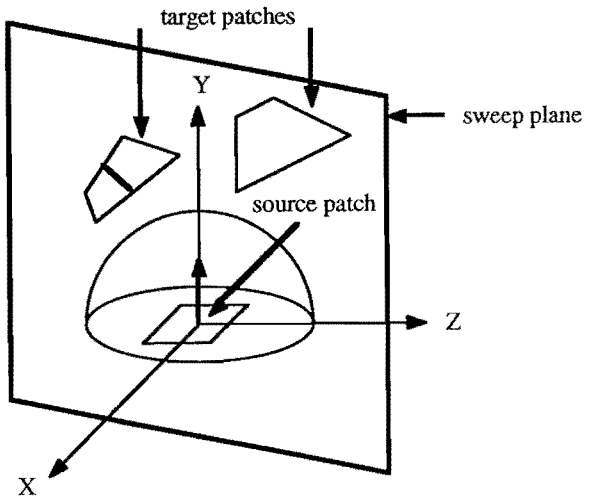

Fig. 2. The geometry of a sweep plane and patches in 3D.

- We can also transform our 3D problem into a 2D problem. A simple way is to use a sweep plane to slice 3D hemisphere rays and polygonal patches (see Figure 2) into 2D rays on the hemicircle and line segments, and then rotate the sweep plane to coincide with the YZ-plane as shown in Figure 3. Now we can use the technique described above to treat it. With this approach, a modification must be made in the tracking step. Instead of using $ADLC$ coming from neighboring shells, the actual degree of local coherence of a patch must be derived. This is because the orientation of line segments originating from slicing a patch will not in general parallel with the orientation of one of the subdivision axes of the patch such that neighboring line segments do not possess similar shapes at all. In order to derive actual degree of local coherence, a sorting procedure is necessary to sort patches by increasing angle. In addition to the overhead introduced by the sorting procedure, another drawback of this approach is that the degree of local coherence captured in 2D is far less than

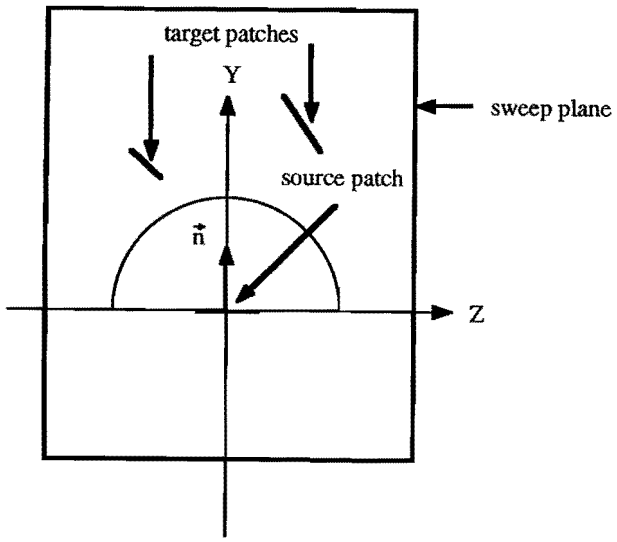

Fig. 3. A coordinate system to represent rays and patches in 2D.

the actual degree of local coherence. For clarity, we refer to the shelling technique with this tracking mechanism as the deterministic shelling technique.

Finally, we want to show the resulting shell structures of the stochastic and deterministic shelling techniques in Figure 4, and 5. In Figure 4, we can see sectors, shells, and subshells which are constructed by using neighborhood information. Some degree of mismatch between a subshell and a patch can be observed from the figure. This is because the stochastic shelling technique can only estimate the degree of local coherence of a patch but not its position. In Figure 5, each shell is exactly matched with the active patch which actually defines the shell. Notice that only a small portion of patches in an environment are necessary to be considered in both cases. This is due to cell traversal procedure.

4 Formal Comparisons

In this section, we derive an estimate of the intersection computation time for the shelling technique and the conventional technique. In order to simplify the derivation, we make the following assumptions:

- The patches are uniformly distributed in the space, and
- For a given number of rays, the percentage of rays leaving a shell will be constant if the product of the number of patches in the shell and the density of the rays in the shell is constant.

Based on those assumptions, we can derive a shell structure as shown in Figure 6 (represented conceptually in 2D) such that the percentage of rays leaving all the shells can be kept the same. Let R be the number of rays in the starting shell, P the number of patches in this shell, and let S be the percentage of rays leaving any shell. Moreover,

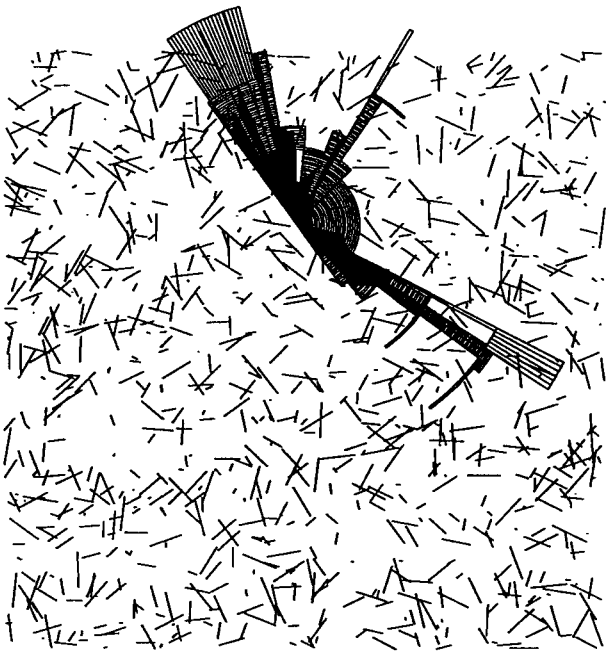

Fig. 4. The actual shell structure of the stochastic shelling technique.

Fig. 5. The actual shell structure of the deterministic shelling technique.

let T_λ be the pipeline period of the intersection computation circuit as proposed in [22] or [23]. The intersection computation time for the shelling technique, T_1, can be estimated to be

$$T_1(RP + SR2^2SP + S^2R3^2S^2P + \ldots)T_\lambda \tag{1}$$

Here we assume that the intersection computation time of one patch with respect to a bundle of rays can balance the arrival time of the next patch. Notice that the remaining rays need to visit only a fraction of patches because of cell traversal procedure. Evaluating the series in Equation 1, we obtain

$$T_1 = \frac{(1+S^2)RPT_\lambda}{(1-S^2)^3} \tag{2}$$

$$= K_1 RPT_\lambda \tag{3}$$

where

$$K_1 = \frac{(1+S^2)}{(1-S^2)^3} \tag{4}$$

The effect of S on K_1 is depicted in Figure 7. From this, we observe that K_1 increases slowly when S is small, but it rises up very quickly when S approaches to 1. This explains the purpose and the necessity of introducing a tracking mechanism in the shelling technique.

From the definition of $ADLC$, we can derive the $ADLC$ of shell i as follows:

$$ADLC_i = \frac{KR}{i^2P} \tag{5}$$

We compute the intersection computation time of the conventional technique in shell i, $T2_i$, as follows:

$$T2_i = \begin{cases} (S^i i^2 P + S^{i-1} KR)T_c & \text{if } ADLC_i > 1, \\ (S^{i-1} i^2 P)T_c & \text{if } ADLC_i \leq 1 \end{cases} \tag{6}$$

where $K = (1-S)$ denotes the percentage of rays screened out in a shell, and T_c is the data arrival time of the intersection computation circuit which will be dominated by the loading time of one patch from the host when using a shared memory parallel architecture.

From Equation 1, we can express the portion of time of T_1 spent on shell i, T_{1_i}, as follows:

$$T_{1_i} = \begin{cases} S^{2(i-1)} Ri^2 PT_\lambda & \text{if } S^{i-1}RT_\lambda > T_c \\ S^{i-1} i^2 PT_c & \text{if } S^{i-1}RT_\lambda \leq T_c \end{cases} \tag{7}$$

Here we have allowed the intersection computation time of one patch not to balance the arrival time of the next patch. If balancing is achieved, then only the first part of Equation 7 will apply as was in Equation 1.

Comparing Equation 6 and Equation 7, we see that:

- When $ADLC_i$ is not larger than 1, there is no advantage at all to use the shelling technique. This is because

$$\begin{cases} T_{1_i} > T_{2_i} & \text{if } S^{i-1}RT_\lambda > T_c \\ T_{1_i} = T_{2_i} & \text{if } S^{i-1}RT_\lambda \leq T_c \end{cases} \tag{8}$$

- From the above, only in case that $ADLC_i$ is larger than 1 is of major interest to us. If we can subdivide a shell into subshells such that the number of rays within a subshell i is equal to $ADLC_i$, then we have

$$T_{1_i} = \begin{cases} KS^{i-1}RT_\lambda & \text{if } \left(\frac{KR}{i^2P}\right)T_\lambda > T_c \\ S^{i-1}i^2PT_c & \text{if } \left(\frac{KR}{i^2P}\right)T_\lambda \leq T_c \end{cases} \tag{9}$$

From Equation 6 and Equation 9, we obtain

$$SF_i = \begin{cases} (1 + \frac{S}{ADLC_i})\left(\frac{T_c}{T_\lambda}\right) & if \left(\frac{KR}{i^2P}\right) T_\lambda > T_c, \\ S + ADLC_i & if \left(\frac{KR}{i^2P}\right) T_\lambda \leq T_c \end{cases} \tag{10}$$

where $SF_i = T_{2_i}/T_{1_i}$ is called the speedup factor in shell i that denotes the ratio of the intersection computation time that spent on shell i in both techniques. The optimum point lies at $S = 0$,i.e., $K = 0$, in which case SF_i is given by

$$SF_i = \begin{cases} \frac{T_c}{T_\lambda} & if \left(\frac{KR}{i^2P}\right) T_\lambda > T_c \\ ADLC_i & if \left(\frac{KR}{i^2P}\right) T_\lambda \leq T_c \end{cases} \tag{11}$$

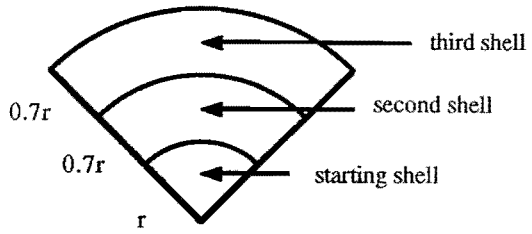

Fig. 6. A shell structure to keep the percentage of rays leaving all the shells the same.

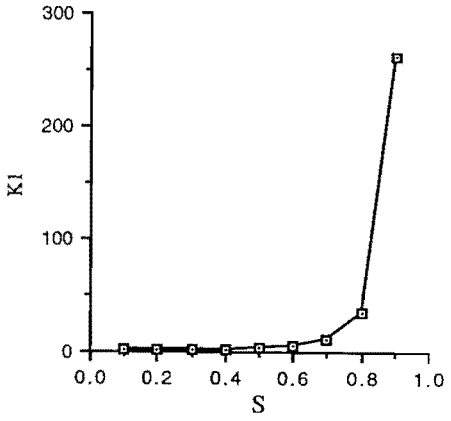

Fig. 7. The effect of S on K1.

We can make some remarks as follows:

- The shelling technique provides a more balanced structure for processing time and I/O time.

- A considerable speedup can be obtained when the environment consists of large patches (actually large $ADLC$). If a shared memory parallel architecture is adopted, the speedup factor will be $ADLC$ because the data arrival time Tc is dominated by the loading time of one patch from the host. However, when a high speed cache memory is allowed for each PE and the next patch can be loaded immediately from that memory, the speedup factor will be the ratio of Tc over T_λ. In general, the speedup factor lies in-between which can be determined only when the detailed statistics about the environment and the underlying architecture are known.

5 The Results

The two shelling techniques as described in Section 3 have been implemented in the C language. Different scenes with randomly generated patches (actually 1000 line segments in 2D) have been tested. The length of the segments is within the following: 4, 8, 16, 32, 64, and 128. In this section, we compare the results of three techniques: the stochastic shelling technique, the deterministic shelling technique, and the conventional technique. For convenience, the stochastic and deterministic shelling techniques are referred to shelling technique S and shelling technique D, respectively.

In the following, three factors will be taken into account for comparison purpose:

- The number of patches read in (refer to Figure 8 and 9)

  When using a shared memory parallel architecture, this is the most important factor in determining the performance. The saving from the two shelling techniques strongly depends on the actual $ADLC$ of an environment. A considerable saving can be observed when the actual $ADLC$ is high. Notice from the figure that the number of patches read in after screening in shelling technique D will approach that of shelling technique S. This motivates us to screen irrelevant patches out before loading them.

- The Number of Cells Traversed (refer to Figure 10)

  The results of three techniques are much the same. This is because we did not use radius comparison to cull relevant patches. As said before, one drawback of shelling technique S is that the prescribed shell depth will limit the span of passing empty regions.

- The Number of Intersection Computations (refer to Figure 11)

  In order to see the overhead of introducing wasteful intersection computations, here we compare results with actual value. It is interesting that the result of shelling technique S is always about twice that of the actual value. This is reasonable because shelling technique S can only estimate the degree of local coherence of a patch in a shell but not its position. So one additional bundle of rays must be introduced to compute intersection points. This bears a similarity with the Nyquist theory in communication. As for the result of shelling technique D, it is close to the actual value when the $ADLC$ is low, but it will approach that of shelling technique S when the $ADLC$ is high. This is due to the fact that it is difficult to isolate large patches by decreasing cell size.

So far we have compared the three factors separately. From a hardware perspective, they are related to each other as will be discussed in the following. In fact, two-level

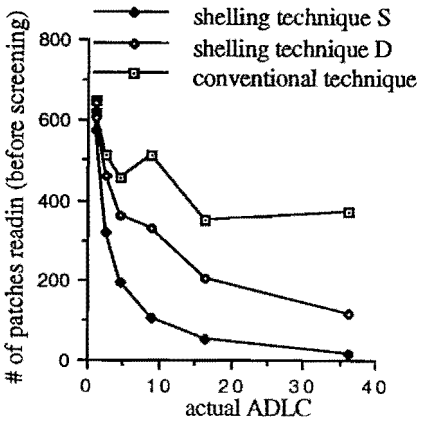

Fig. 8. The number of patches read in before screening.

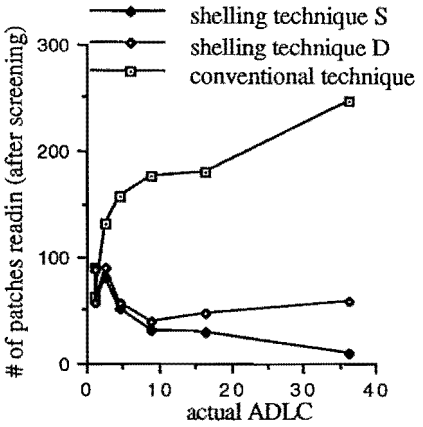

Fig. 9. The number of patches read in after screening.

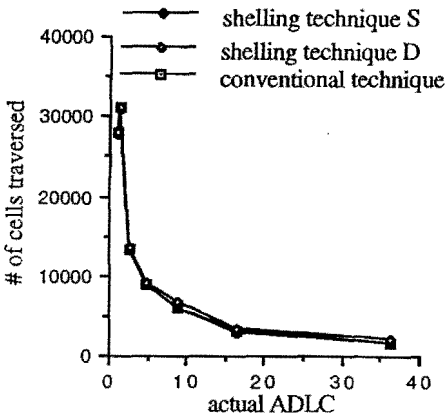

Fig. 10. The number of cells traversed.

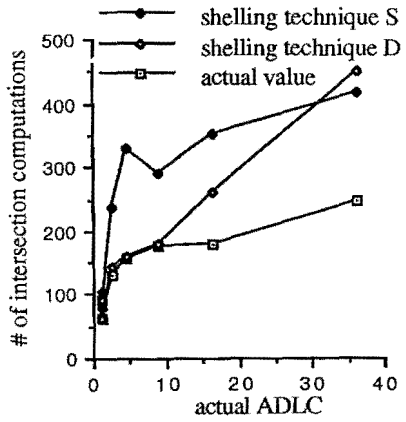

Fig. 11. The number of intersection computations.

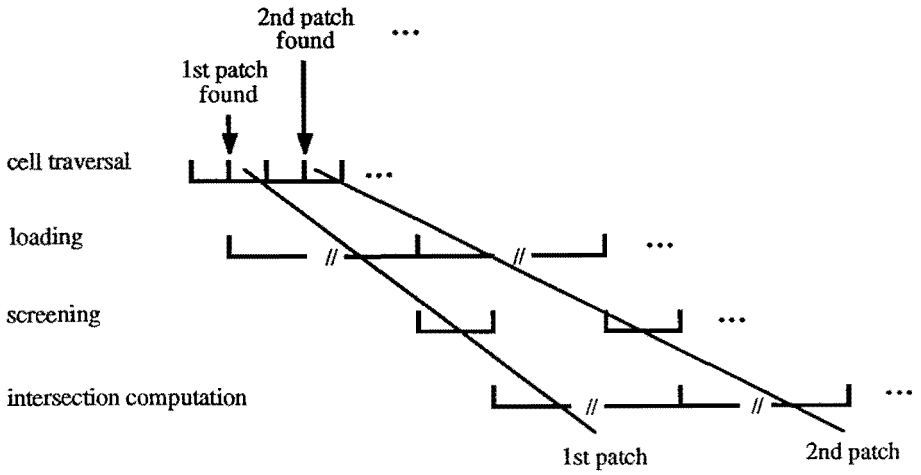

Fig. 12. The four steps in the shelling technique are processed in an overlapped fashion.

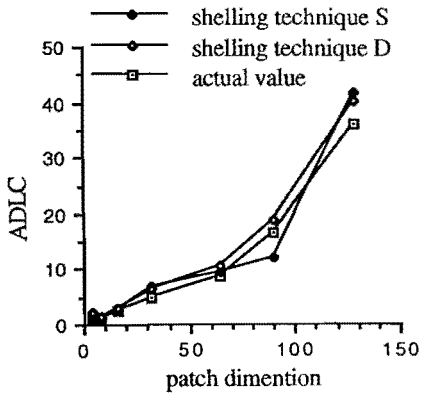

Fig. 13. The average degree of local coherence.

pipelining can be exploited in the shelling technique, in which the first level refers to pipelining at the four steps as shown in Figure 12 and the second refers to pipelining within each step. The second level pipelining can be understood when realizing them with pipelined circuits. As to the first level pipelining, it can be understood as follows:

- When the neighborhood information is known, we can start the cell traversal of the next shell.
- After traversing some empty cells (or macro-regions), a patch is found and then loaded from the host. During this time, the cell traversal still keeps on searching patches.
- After loading a patch from the host, we can start the preprocessing to check whether it is a backward patch or not. During this time, the cell traversal and loading still keep on searching and supplying patches for the preprocessing.
- If it is not a backward patch, then a bundle of rays proceeds to compute the intersection points with this patch. During this time, the cell traversal, loading, and preprocessing still keep on traversing, supplying, and screening out patches.

In the conventional technique, there is only one intersection computation for one patch. Without solving the bottleneck, i.e., the loading, it cannot benefit from the pipelining at all. By contrast, a bundle of rays will proceed to compute the intersection points with one patch in the shelling technique. This provides a more balanced structure for the above-mentioned four steps. From this point of view, we can assert that the shelling technique can outperform the conventional technique. Nonetheless, the comparison can be made only when the detailed statistics about the environment and the underlying architecture are known.

It is known that pipeline efficiency is a measure of the performance of a pipeline. In the shelling technique, the *ADLC* plays an important role in determining the pipeline efficiency. If the *ADLC* of a shell is too small to fill up the pipeline, then the pipeline efficiency will be degraded. This can happen in an environment consisting of small patches or in an environment consisting of large patches when the tracking mechanism cannot capture the actual degree of local coherence. For this purpose, we use the *ADLC* as an index to show how close the coherence of an environment can be captured. In Figure 13, we see that the results of both techniques are desirable.

6 Conclusion

In this paper we have proposed a new space partitioning technique such that the requirement of loading patches can be reduced considerably. It can be used as a replacement of accommodating a large local memory for each PE. Of course, it is favourable to use a reasonable size of cache memory in each PE. This leads to a more balanced structure to fill up the pipeline such that a highly pipelined parallel architecture is applicable to speedup the form-factor computations in the radiosity method.

Acknowledgements

This research has been supported in part by the commission of the EEC under the NANA project (BRA 3280) and by ITRI.

References

[1] Appel, A.: Some techniques for shading machine renderings of solids. *Proc. AFIPS JSCC*, Vol. 23, No. 6, 1968, pp. 37-45.

[2] Goral, C., Torrance, K., Greenberg, D. and Battaile, B.: Modeling the interaction of light between diffuse surfaces. *Computer Graphics (SIGGRAPH '84 Proceedings)*, Vol. 18, No. 3, July 1984, pp. 213-222.

[3] Cohen, M. and Greenberg, D.: The Hemi-Cube: A radiosity solution for complex environments. *Computer Graphics (SIGGRAPH '85 Proceedings)*, Vol. 19, No. 3, July 1985, pp. 31-40.

[4] Cohen, M., Greenberg, D., Immel, D. and Brock, P.: An efficient radiosity approach for realistic image synthesis. *IEEE Computer Graphics and Applications*, Vol. 6, No. 2, March 1986, pp. 26-35.

[5] Immel, D., Cohen, M. and Greenberg, D.: A radiosity method for non-diffuse environments. *Computer Graphics (SIGGRAPH '86 Proceedings)*, Vol. 20, No. 4, August 1986, pp. 133- 142.

[6] Wallace, J. Cohen, M. and Greenberg, D.: A two pass solution to the rendering equation: A Synthesis of Ray-Tracing and Radiosity Methods. *Computer Graphics (SIGGRAPH '87 Proceedings)*, Vol. 21, No. 4, July 1987, pp. 311-320.

[7] Cohen, M., Chen, S., Wallace, R. and Greenberg, D.: A progressive refinement approach to fast radiosity image generation. *Computer Graphics (SIGGRAPH '88 Proceedings)*, Vol. 22, No. 4, August 1988, pp. 75-84.

[8] Heckbert, S. and Hanrahan, P.: Beam tracing polygonal objects. *Computer Graphics*, Vol. 18, No. 3, July 1984, pp. 119-127.

[9] Speer, L., Tony, D. and Barsky, B.: A theoretical and empirical analysis of coherence ray-tracing. *Computer-Generated Images (Proceedings of Graphics Interface '85)*, May 1985, pp. 11-25.

[10] Glassner, A.: Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, Vol. 4, No. 10, October 1984, pp. 15-22.

[11] Tamminnen, M., Karonen, O. and Mantyla, M.: Ray-casting and block model conversion using spatial index. *Computer-Aided Design*, Vol. 16, No. 4, July 1984, pp. 203-208.

[12] Fujimoto, A., Tanaka, T. and Iwata, K.: ARTS: Accelerated ray-tracing system. *IEEE Computer Graphics and Applications*, Vol. 6, No. 4, April 1986, pp. 16-26.

[13] Murakami, K., Hirota, K. and Ishii, M.: Fast ray tracing. *FUJITSU Sci. Technical Journal*, Vol. 24, No. 2, June 1988, pp. 150-159.

[14] Naruse, T., Yoshida, M., Takahashi, T. and Naito, S.: Sight : A dedicated computer graphics machine. *Computer Graphics Forum*, Vol. 6, No. 4, 1987, pp. 327-334.

[15] Cleary, J., Wyvill, B., Birtwistle, G. and Vatti, R.: Multiprocessor ray tracing. *Computer Graphics Forum*, Vol. 5, No. 1, January 1986, pp. 3-12.

[16] Dippé, M. and Swensen, J.: An adaptive subdivision algorithm and parallel architecture for realistic image synthesis. *SIGGRAPH'84*, 1984, pp. 149-157.

[17] Green, S., Paddon, D. and Lewis, E.: A parallel algorithm and tree-based computer architecture for ray traced computer graphics. *Parallel Processing for Computer Vision and Display*, University of Leeds, January 1988.

[18] Priol, T. and Bouatouch, K.: Static load balancing for a parallel ray tracing on a MIMD hypercube. *The Visual Computer*, Vol. 5, 1989, pp. 109-119.

[19] Yilmaz, A., Hagestein, S., Deprettere, E. and Dewilde, P.: A hardware algorithm for fast realistic image synthesis. *Eurographics '89*.

[20] Fuchs, H., Kedem, Z. and Naylor, B.: On visible surface generation by a priori tree structures. *Computer Graphics (SIGGRAPH '80 Proceedings)*, Vol. 14, No. 3, July 1980, pp. 124-133.

[21] Devillers, O.: The macro-regions: an efficient space subdivision structure for ray tracing. *Eurographics '89*, pp. 27-38.

[22] Ullner, M.: Parallel machines for computer graphics. *Doctor Thesis*, California Institute of Technology, January 1983.

[23] Bu, J. and Deprettere, E.: A VLSI system architecture for high-speed radiative transfer 3D image synthesis. *The Visual Computer*, Vol. 5, No. 3, June 1989.