Using A-Buffer in Radiosity

Q. Wang, J. H. Tang, C. H. Lim, H. C. Teh, and Z. Huang
Department of Computer Science, School of Computing
National University of Singapore, Singapore 117543
Email: {wangqius, tehhc, huangzy}@comp.nus.edu.sg

Abstract

Anti-aliasing in form factor computation and precise determination of shadow boundaries are important issues for Radiosity method. A-Buffer is a technique introduced by Carpenter for anti-aliasing in hidden surface removal. In this paper, we propose a method based on the A-Buffer to replace the Z-buffer in the computation of form factors using the Hemicube for anti-aliasing. The method directly computes shadow boundaries in contrast to the adaptive re-meshing approach. Though the processing of shadow boundaries is not fast, only one discontinuity meshing (no iteration) is required. We implement the method in an infrastructure building design system.

Keywords: *Radiosity, shadow boundary, anti-aliasing, A-Buffer*

1. Introduction

In the quest for visual realism in computer graphics, the understanding of illumination properties and the development of global illumination algorithms for photo realistic images are an important step towards developing wider graphical engineering applications. One of the popular global illumination methods is radiosity.

Radiosity method was first proposed by Goral *et al.* [Gora84]. It is based on the concept of radiative heat transfer [Sieg92]. Radiative heat transfer was developed in the 1950s for computing radiant interchange between surfaces. Thus, in image rendering, radiosity method is applicable to solving for the inter-reflection of light between ideal diffuse (Lambertian) surfaces in a closed environment. This way of light treatment produces view-independent global illumination information.

The radiosity solution is described by the following equation:

$$B_i = E_i + {}_i\sum_{j=1}^{n} B_j F_{i-j}\ ,$$

where B_i is the radiosity of patch i (W/m^2), E_i the emittance of patch i (W/ m^2), A_i the area of patch i (m^2), ρ_i the reflectivity of patch i, and F_{i-j} the form factor which is the fraction of light energy leaving patch i that arrives at patch j.

In the progressive refinement approach proposed by Cohen *et al.* [Cohe88], illumination is calculated one step at a time. At each step, the most influential illuminator is selected and the energy shot to other surfaces. The process is repeated until the next selected illuminator has little or no effect on the scene. Thus the radiosity of every patch can be approximated with O(n) time and space complexity. The use of this algorithm, when accompanied by the Hemicube method [Cohe85], considerably speeds up the radiosity method.

The calculation of form factors requires appropriate *meshing* (subdivision of surface into elements) to produce accurate results for the radiosity method. Early methods performed simple uniform meshing that results in jaggy shadow boundaries and other artifacts. In fact, radiosity functions have D^0 discontinuities at touching surfaces, intersections, creases, and along shadow boundaries from point light sources, and D^1 discontinuities along shadow boundaries from linear light source. Uniform meshing is unable to resolve these discontinuities.

It is desirable that meshing follows the distribution of light in the scene, with a higher density of elements in areas where the illumination changes rapidly. Since the distribution of illumination is precisely the unknown of the problem, it is unrealistic to rely on the user to guess where and how the mesh elements should be placed. Instead discontinuity meshing should be generated automatically based on the actual shadow boundaries.

We propose a method based on the direct computation of shadow boundaries in a scene using A-buffer, an anti-alising technique introduced by Carpenter [Carp84]. A-buffer replaces the Z-buffer used in the Hemicube method. Depending on the context of the scene and the methods used, surfaces, polygons, patches, and elements have been labeled for the basic components during the computation of radiosity values. For simplicity, we denote all basic components as patches since adaptive meshing is not involved in our method. During scan-converting the patches with respect to the viewpoint at a transmitted patch, a list of fragments, that represents parts of patches contributed to a particular pixel, in Z-increasing order is created during the propagation of the form factors. A list of occluding patches for each patch is extracted and shadow edges computed. Discontinuity meshing (DM) is then carried out using a triangulation algorithm [Shew96]. Finally, radiosity values at all vertices (from original mesh and shadow edges) are calculated and the scene is rendered with Gouraud shading.

We brief the related work in section 2. Next, we present our method in detail and followed by description of our implementation. Finally, we show some experimental results and conclude the paper.

2. Related previous work

There has been much related work on the issue of meshing. The *posteriori* (adaptive) meshing approach [Cohe86] employs a coarse uniform meshing and then refines the mesh where the gradient of radiosity is large. After a surface is subdivided adaptively, T-vertices may occur at shared edges where the new vertex is introduced. T-vertices need to be removed, otherwise it may cause shading discontinuities. Furthermore, it is not effective to mesh along shadow boundaries. Many subdivisions are needed to approximate the radiosity values at shadow boundaries. This results in high mesh density.

Another approach is *priori* meshing. It employs object space techniques to predict, before radiosity calculation, where shadow edges will occur. A mesh is then constructed accordingly. Some methods have been introduced to detect shadow edges. Campbell's priori meshing method finds some shadow boundary by splitting the scene with planes through light source points and edges of occluding patches [Camp90]. Baum's method [Baum91] finds the discontinuities caused by intersecting and touching of surfaces. Heckbert's method [Heck92] finds shadow boundaries by projecting emitter patch and occluder patch into the scene. A progressive radiosity DM algorithm [Lisch92] based on BSP tree to detect shadow boundaries demonstrates improved photo-realism of the images.

3. A-buffer based method

In this section, we describe the A-buffer based method for form factor computation and DM. It is essentially a *priori* meshing approach. The main idea is to replace the Z-buffer used in the standard Hemicube algorithm by A-buffer. The Hemicube algorithm and the A-buffer will be briefly summarized, then followed by the description of the use of A-buffer in Hemicube algorithm and the shadow boundaries detection.

3.1. Hemicube algorithm

Hemicube algorithm is introduced by Cohen *et al.* [Cohe85] to calculate the form factor between a pair of patches. When using Z-buffer to implement the Hemicube method, the Z-buffer is maintained as usual. But rather than storing color values into the frame buffer, the patch *IDs* are stored. Only partial information of the form factor could be captured if this were implemented. This is because information concerning the relative angles of orientation between each pair of emitting and receiving patches are not fully taken into account.

The amount of radiant energy leaving an emitting patch and intersected by a receiving patch in fact depends on the relative angles of both the emitting and receiving patch, in a way similar to Lambertian reflection. This angular dependency is handled by a look-up table which stores the delta form factors. The pre-calculated delta form factors are used to properly scale the amount of transmitted energy associated with each hemicube pixel according to its angular orientation. The total form factor between the pair of patches is finally obtained by summing the relevant delta form factors.

Although the Hemicube method has made evaluation of form factors numerically an easy process, it still encounters other problem. The accuracy of the form factors calculated is dependant on the hemicube's grid spacing. A coarse resolution may result in visible aliasing artifacts in the output images for small patches. While this aliasing artifact may be reduced with higher resolution it will significantly slow down form factor calculation. Doubling the size of the resolution approximately quadruples the execution time.

One approach to solve the problem is to use jittering method. This will convert the aliasing to noise [Tell92][Cook86]. This method assumes that noise at pixel level is not interpreted by eyes as structure in image. However, the proper shading of objects in radiosity method is driven by correct meshing instead of pixel shading and thus the problem may not be directly addressed.

Another approach is to use ray tracing with adaptive subdivision [Dipp85]. It uses shot rays to test for shadow edges and adaptively shoot more rays if a shadow boundary is found. This method will still miss objects that fall between two rays and cause aliasing if not applied properly.

We resort to the A-buffer technique for a favorable solution.

3.2. A-buffer

A-buffer is an anti-aliasing technique introduced by Carpenter [Carp84]. It approximates the continuous filtering process by sampling sub-pixel fragments using bitmasks. A-buffer combines the filtering with the standard rendering methodology, essentially a Z-buffer approach. The 'A' stands for anti-alised, area-averaged accumulation buffer. The significant advantage is that at the sub-pixel level floating point geometry calculations are avoided. Coverage and area weighting is accomplished by using bitwise logical operators (AND, OR, and XOR) between the bit patterns or masks representing patch fragments. It is an efficient area sampling technique, where the processing per pixel depends only on the number of visible fragments.

A-buffer is made up of a 2D array grid. Each grid may consist of a list of fragments. Each fragment represents a part of a patch that is rendered to that particular pixel. A fragment consists of the following information:

(1) A 8x8 bit mask represents the coverage of the pixel
(2) The z-depth
(3) A pointer to the patch that creates this fragment

The patch to be scan-converted will first be clipped to the pixel and then scan-converted again to find the mask representation. The clipping process is needed to detect aliasing problem. If after clipping, it is found that there is a patch to be drawn onto the pixel but the mask is found to be empty after scan converting, we can safely deduced that undersampling has occurred. In this case, a finer bit mask can be used (Figure 1).

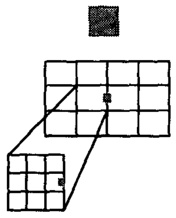


Figure 1: Detection of under sampling.

3.3. Propagation and update of form factors using A-buffer

In this subsection, we describe the computation of form factor using A-buffer. Each fragment list in the A-buffer is sorted in ascending depth order after all patches are scan converted using the Hemicube algorithm. The method starts with a full delta form factor and a full bit mask (first propagated mask) for each pixel list. The following two formulas determine the inside and outside of a mask.

$$mask_{inside} = mask_{propagated} \cap mask_{fragment} \tag{1}$$

$$mask_{outside} = mask_{propagated} \cap (\sim mask_{fragment}) \tag{2}$$

where $mask_{propagated}$ represents the propagated mask, $mask_{fragment}$ the fragment mask, $mask_{inside}$ the portion of the propagated mask falls inside the patch, and $mask_{outside}$ the portion of the propagated mask falls outside the patch

Assuming no transparency at this moment, the amount of delta form factor captured by the inside and outside masks are calculated by the following formula:

$$\Delta F_{inside} = \frac{mask_{area}^{(inside)}}{mask_{area}} \times \Delta F_{propagated} \tag{3}$$

$$\Delta F_{outside} = \frac{mask_{area}^{(outside)}}{mask_{area}} \times \Delta F_{propagated} \tag{4}$$

where $mask_{area}$ represents the area of the propagated mask. We can visualize this by referring to Figure 2.

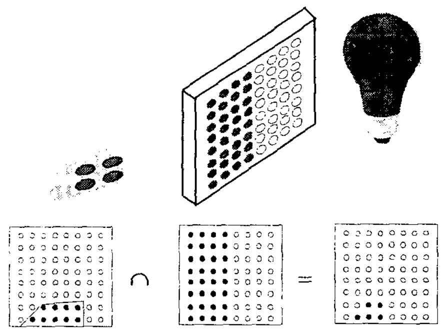


Figure 2: Visualization of the propagation miss object.

Imagine a bulb on the right most that shine through a filter (first propagated mask). Next, a piece of wood (receiving patch) is placed behind the filter. The light that passes through the filter and falls on the piece of wood is calculated with Equation (3) and the light that passes through the filter but falls outside the piece of wood is calculated with Equation (4). If instead of using a piece of wood, a translucent object is used, the light that passes through the filter and falls on the translucent object is calculated using the following equation:

$$\Delta F_{i-j}^{(retained)} = Opacity_j \times \Delta F_{inside} \tag{5}$$

where $Opacity_j$ = Opacity of the translucent object j.

The rest of the light that are not captured by the translucent object will be propagated to the next object that is placed behind the current object. The amount to be propagated is calculated using:

$$\Delta F_{i-j}^{(inside\_propagated)} = (1 - Opacity_j) \times \Delta F_{inside}$$
$$= \Delta F_{inside} - \Delta F_{i-j}^{(retained)} \quad (6)$$
$$\Delta F_{i-j}^{(outside\_propagated)} = \Delta F_{outside} \quad (7)$$

The calculated amounts are then recursively applied until no more objects or the amounts are zero. The propagation algorithm is indicated as followings:

```
Algorithm 1. PropFF (fragment, mask, formfactor)
  Find inside and outside mask (Eqn (1) & (2))
  Find inside and outside form factors (Eqn (3) & (4))
  Calculate form factor retained (Eqn (5))
  Add retained form factor to patch
  Calculate inside (finprog) and outside (foutprog) form
      factors to be propagated (Eqn (6) & (7))
  if there is still fragment in the list
    if inside form factor to be propagated is not zero
        PropFF(next fragment, inside mask, finprog)
    if outside form factor to be propagated is not zero
        PropFF(next fragment, outside mask, foutprog)
```

3.4. Occlusion information

After calculating the required form factors for each patch, the fragment lists are not discarded. Each list represents the occluding information at the particular pixel from the viewpoint of the light-emitting patch. The lists are merged together such that for each patch, we know which are the occluding patches (Figure 3).

The algorithm is a straight-forward comparison of occlusion patches. The opacity of each patch is tested before inserting the next patch in the list as an occluding patch. As is shown in Figure 3, this is equivalent as saying if the patch that is occluding has an opacity equal 1.0 and the testMask (which can be consider as a ray) is totally occluded, then it is known that the rest of the patch in the list will not have shadow boundary. The merging algorithm is as follows:

```
Algorithm 2. MergeOcclusionList ()
for each fragment, f1, in list
  testMask = f1->mask
  for each fragment, f2, further in the list
    if (testMask ∩ f2->mask is not empty)
      insert f1->patch into f2->patch if it does not exist
        if (f2->patch->opacity == 1.0)
          testMask = testMask ∩ ~(f2->mask)
        if (testMask is empty)
          break;
```

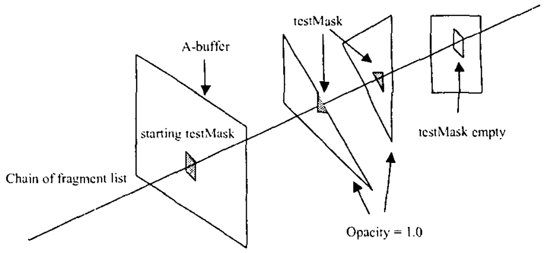


Figure 3: Patch occlusion list creation.

3.5. Shadow boundaries detection

In this subsection, we describe the computation of shadow edges and use them for DM. First, we devise an algorithm (Algorithm 3) to compute the shadow edges for each patch from its occluding patch list derived from Algorithm 2. Next, we conduct DM using a triangulation algorithm [Shew96] with the original mesh and the added shadow edges as input. Finally, we compute radiosity values for this new set of vertices and render the scene with Gouraud shading. For a scene of n light sources, n iterations will be run for the progressive refinement.

To compute the shadow edges for each patch, the opacities of the patches in its occluding patch list are checked recursively. For a patch in the scene, its occluding patch is any patch occluding the light and casting shadow onto it. The occluding patch list of a patch may be empty if no patch in the scene occludes it. If the occluding patch is transparent, the shadow projected from other previous occluding patches will pass through it; otherwise, the projecting wedge, formed by a light source with the edge of the occluding patch, will be split into two parts. Only the wedge that is not occluded will be considered for further occlusion to other patches.

The algorithm *ProjectShadowEdge(e)* given below is executed for each patch in the scene. If an occluding list is not empty, every edge of the patches in the occluding list will be projected and every shared edge calls the algorithm only once.

```
Algorithm 3. ProjectShadowEdge (e)
if (the occluding list is not empty)
    project edge, e, onto next patch, p
    if (e completely lies in p)
        insert e into shadow edge list of p
    else
      if (p is transparent)
        ProjectShadowEdge (e)   // recursively
          project e to next patch of the occluding list
      else // opaque
        let v1 and v2 be the two vertices of e
        // address three cases of e with respect to p
```

```
case 1: v1 is inside p and v2 is outside p
    find v3 such that v1-v3 is inside p
    insert v1-v3 into shadow edge list of p
    ProjectShadowEdge(v2-v3)
    // recursively project edge, v2-v3
case 2: v2 is inside p and v1 is outside p
    find v3 such that v2-v3 is inside p
    insert v2-v3 into shadow edge list of p
    ProjectShadowEdge (v1-v3)
    // recursively project edge, v2-v3
case 3: v1 and v2 are both outside p
    find v3 and v4 between v1 and v2 such that
    edge v3-v4 lies inside p
    if such edges exists
        insert v3-v4 into shadow edge list of p
        ProjectShadowEdge (v1-v3)
        // recursively project edge, v1-v3
        ProjectShadowEdge (v2-v4)
            // recursively project edge, v2-v4
    else
        ProjectShadowEdge (e)
        // recursively project e
```

After the shadow boundaries have been found, DM can be conducted. The triangulation method proposed by Shewchuk [Shew96] meets this purpose. Shewchuk introduced a method for 2D mesh generation and construction of Delaunay triangulations, constrained Delaunay triangulations, and Voronoï diagrams. The method is fast, memory-efficient, and robust (exact arithmetic); it computes Delaunay triangulations and constrained Delaunay triangulations exactly. Guaranteed-quality meshes (having no small angles) are generated using Ruppert's Delaunay refinement algorithm [Rupp95].

The algorithm is implemented in a mesh generator available on the Web [Tria]. As the shadow boundaries may be shared, preprocessing should remove all duplications for shared vertices. After preprocessing, the unique vertex list together with the original patches is used as input to the mesh generator.

4. Test Results

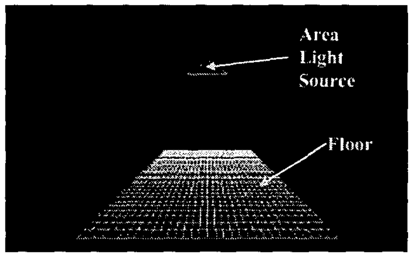
The test results of our method are presented in this section. The correctness of the proposed method is first illustrated with a simple test model. The images of a more complex scene produced from an infrastructure building design system shows more details for the soft shadows.

4.1. A simple test model

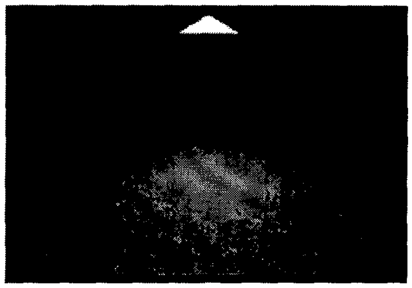
An area light source is placed over a finely meshed floor in Figure 4a. A 50 x 50 A-buffer with 8 x 8 sub-pixel division is used and the image produced is shown in Figure 4b. Next, a 50 x 50 (Figure 5a) and 100 x 100 (Figure 5b) Z-Buffer is used to calculate the radiosity of the floor. The area light source is taken as a point light source at the middle of the patch and one iteration of progressive refinement is made for both tests. It can be seen that some patches are more brightly illuminated than the rest. This is due to the aliasing problem of the Z-Buffer discussed earlier.

Since A-buffer inherently does an adaptive 8 x 8 sub-pixel division, a 400 x 400 Z-Buffer is used to compare the result. Figure 5c shows an image generated with a 400x400 Z-Buffer. Although the aliasing problem is no longer there, but it is done with brute force super-sampling.

An occluding triangle is added in figure 6a and 6b. The mesh and image are generated by assuming the area light as a point light source. For clarity of display, the walls, ceiling and floor of Figure 6a are each represented with only two triangles. The shadow boundary of the occluding patch is shown on the floor. The additional edges resulted from the DM are also given. The image in Figure 6b is rendered from the same environment but with much higher starting triangle density. Figure 7a and 7b are similarly generated with the vertices of the emitting patch treated as point light sources. Figure 7a shows that the umbra and penumbra are detected correctly. Figure 7b also shows a soft shadow generated with Gouraud shading.

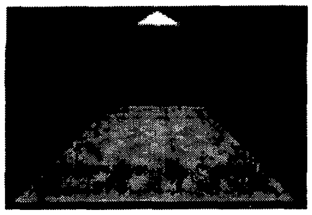


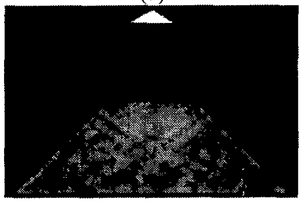
(a)



(b)

Figure 4: (a) Test model; (b) Using 50 x 50 A-buffer.


(a)


(b)

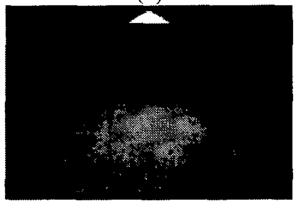
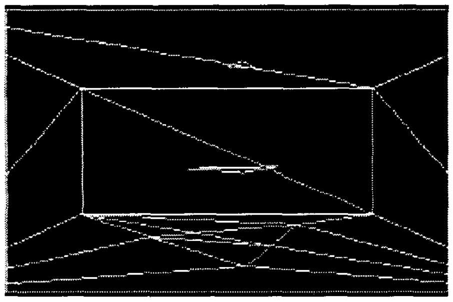
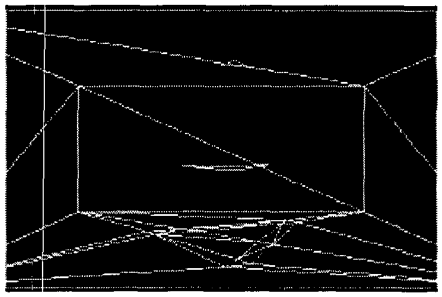

(c)

Figure 5: Using Z-buffer (a) 50 x 50; (b) 100 x 100 (c) 400 x 400.


(a)

(b)

Figure 6: (a) DM assuming the area light as point light source; (b) Image rendered using DM.


(a)

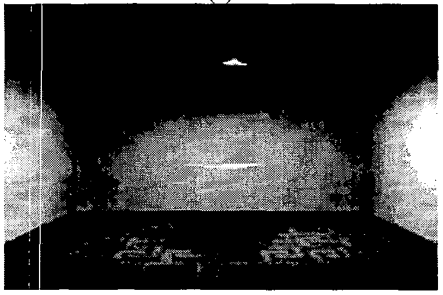

(b)

Figure 7: (a) DM using vertices as point light sources; (b) Image rendered using DM with vertices as point light sources.

4.2. Implementation of Radiosity for Building Interior Design

In this subsection, we describe the implementation of our radiosity method in a previewing system for building interior design. This previewing system with 3D realistic visual effect aims at helping architects to evaluate the visual aspect of the building design with interactive walthrough. It is one of the important issues in a building design life cycle. Both the high quality shading, provided by radiosity, as well as the intermediate quality fast shading, provided by Phong illumination model, are necessary for such an application. The system architecture is illustrated in Figure 8. Besides the graphics user interface (GUI) the system is implemented with a walkthrough engine and a radiosity engine.

The walkthrough engine consists of

(1) Data Converter: It re-organizes the database of the architecture design with reference to its geometric, material, and lighting information into a format suitable for visibility processing.
(2) Visibility Processing: Firstly, a model-space partitioning is carried out using a hierarchical triangulation structure for the purpose of addressing the point-location problem to support

fast data-retrieving. Secondly, an incremental line-stabbing algorithm [Tell91] is employed to compute the cell-to-cell visibilities and organize the geometric and other information (color and opacity) of the geometric model in terms of potentially visible sets. This preprocessing enables interactivity during walkthrough because only the required subset of building data is sent for each frame to achieve fast rendering.

(3) Rendering: It generates the rendering from the current viewpoint based on the shading results computed in the radiosity engine or stored in the visual database. It is supported by the standard graphics card on PC.

The radiosity engine consists of

(1) High Quality Shading: It computes the realistic shades at all vertices of the architecture models that is passed from the architecture database using the A-buffer based radiosity method. The results are either directly retrieved by the walkthrough engine for high quality shading or stored in the visual database for later use.

(2) Fast Shading: It computes the shades similar to that done by the high quality shading module but using the Phong illumination model. This serves as a quick view of the model with intermediate quality shading.

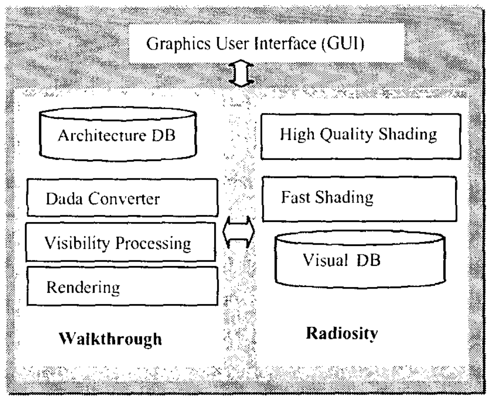


Figure 8: Illustration of the system.

We now look at the overall performance of the A-buffer based radiosity method. We illustrate the DM with a wire frame display of a corner of the room in Figure 9. It shows the final triangular mesh produced. Two snapshots during a walkthrough of the scene are given in Figure 10 and 11. There were three rectangular light sources in the room. Each was partitioned into two triangles as were done for all initial patches. Hence 2x3 progressive refinement iterations were carried out during the radiosity calculation. With a Pentium 3 running at 667MHz and 128MB, the time for the shadow boundaries detection and DM was 20 min 42 seconds and 22 minutes 12 seconds for the radiosity computation. The number of triangles before and after the DM was 2998 and 6192 respectively. Soft shadow has been correctly detected in the vicinity of the table and chair. As only lighting from the light sources is considered, the proximity rule for hemicube method is satisfied for form factor calculation. If the detailed light interactions among the tiny patches produced by DM were taken into account, other sophisticated analytical method such as [Baum89] would have to be appropriately incorporated. The inherent per-pixel base of our method accounts for the relatively long computation time. As in the process of creating the occlusion list in algorithm 2, much time was spent in processing an occluding patch for a pixel that might had already been identified during the creation of the occluding list for other pixels covered by the same patch. The two levels of scan-conversions and the construction of bitmasks in algorithm 1 are also expensive. We should probably investigate other better data structures to improve processing the patch fragments.

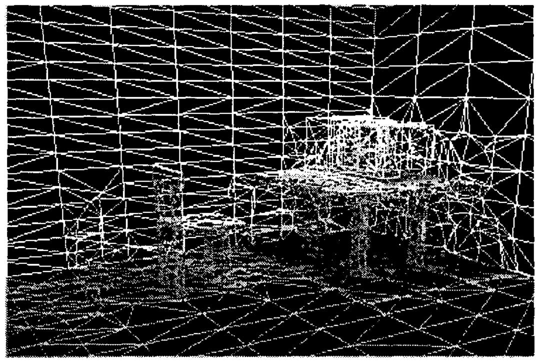


Figure 9: The result of discontinuity meshing for a corner of an interior room.

5. Conclusion

An A-buffer based algorithm that effectively reduces the aliasing artifact and detects shadow boundaries has been proposed. It is incorporated into the Hemicube method. The examples have clearly shown that A-buffer can solve the color aliasing problem in radiosity in a much more graceful manner than Z-buffer. In addition, A-buffer supports transparency for radiosity.

Furthermore, the occluding information created by A-buffer is used to detect the shadow boundaries. The DM is then carried out automatically to generate a final triangle mesh of the scene for better image quality.

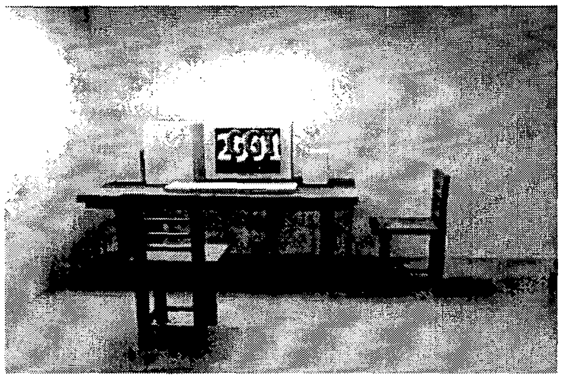


Figure 10: The radiosity-shaded image for the above room interior.

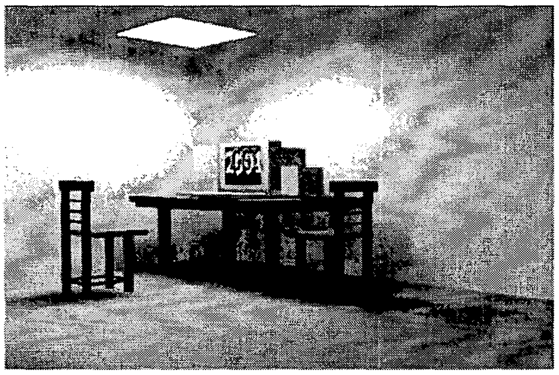


Figure 11: Another view of the radiosity-shaded image.

References

[Baum89] Baum, D. R., Rushmeier, H. E. and Winget, J. M., "Improving Radiosity Solutions through the use of Analytically Determined Form Factors". *Computer Graphics (SIGRAPH '89 Proceedings),* 23(3), 325-34, August 1989.

[Baum91] Baum D. R., Mann S., Smith K. P., and Winget J. M., "Making radiosity usable: Automatic preprocessing and meshing technique for the generation of accurate radiosity solutions". *Computer Graphics (SIGRAPH '91 Proceedings),* 25(4): 51-60, July 1991.

[Camp90] Campbell III A. T. and Donald S. Fussell D. S., "Adaptive mesh generation for global diffuse illumination". *Computer Graphics (SIGRAPH '90 Proceedings),* 24(4): 155-164, Aug. 1990.

[Carp84] Carpenter L., "The A-Buffer, an Antialiased Hidden Surface Method." *Computer Graphics 18(3): 103-108 (ACM SIGGRAPH '84 Proc.),* July 1984.

[Cohe85] Cohen M. F. and Greenberg D. P., "The hemi-cube: A radiosity solution for complex environments." *Computer Graphics 19(3): 31-40 (ACM SIGGRAPH '85 Proc.)*, August 1985.

[Cohe86] Cohen M. F., Donald P. Greenberg D. P., Immel D. S., and Brock P. J., "An efficient radiosity approach for realistic image synthesis". *IEEE Computer Graphics and Applications*, 26-35, Mar. 1986.

[Cohe88] Cohen M. F., Chen S. E., Wallace J. R., Greenberg D. P., "A Progressive Refinement Approach to Fast Radiosity Image Generation," *Computer Graphics 22(4): 75-84 (ACM SIGGRAPH '88 Proc.),* Aug 1988.

[Cook86] Cook R. L., "Stochastic Sampling in Computer Graphics," *ACM Transcations on Graphics* 5, 3 (January 1986), 51-72.

[Dipp85] Dippe M. A. Z. and Wold E. H., "Antialiasing Through Stochastic Sampling", *Computer Graphics (SIGGRAPH '85 Proc.)* 19, 3, 69-78.

[Gigu90] Gigus Z. and Malik J., "Computing the aspect graph for line drawings of polyhedral objects". *IEEE Trans. On Pattern Analysis and Machine Intelligence,* 12(2): 113-122, Feb. 1990.

[Gora84] Goral C. M., Torrance K. E., Greenburg D. O., and Battaile B., "Modeling the Interaction of Light between Diffuse Surfaces," *Computer Graphics 18(3): 213-222 (ACM SIGGRAPH '84 Proc.).*

[Heck92] Heckbert P., "Discontinuity Meshing for Radiosity", *Eurographics Workshop on Rendering*, Bristol, UK, May 1992.

[Lisch92] Lischinski D., Tampieri F., and Greenberg D.P., "Discontinuity Meshing for Accurate Radiosity", *IEEE Computer Graphics and Applications*, 12(6), November 1992, 25-39.

[Rupp95] Jim Ruppert. "A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation". *Journal of Algorithms* **18**(3):548-585, May 1995.

[Shew96] Shewchuk J. R., "Triangle: Engineering a 2D Quality Mesh Generator and Delaunay Triangulator", *First Workshop on Applied Computational Geometry (Philadelphia, Pennsylvania),* pages 124-133, ACM, May 1996.

[Sieg92] Siegel. R. and Howell J. R., "Thermal Radiation Heat Transfer," 3rd Edition. Hemisphere Publishing Corporation, New York, 1992.

[Tell91] Teller S. J. and Carlo H. "Visibility Preprocessing For Interactive Walthroughs" *Computer Graphics (SIGGRAPH '91 Proc.),* 25(4): 61-69, July 1991.

[Tell92] Teller S. J. "Computing the antipenumbra of an area light." *Computer Graphics (SIGGRAPH '92 Proc.),* 26(2): 139-148 ,July, 1992.

[Tria] Shewchuk J. R., "A Two-Dimensional Quality Mesh Generator and Delaunay Triangulator", http://www.cs.cmu.edu/~quake/triangle.html.

[Wall89] Wallace J. R., Elmquist K. A., Haines E. A., "A Ray Tracing Algorithm for Progressive Refinement", *Computer Graphics (SIGGRAPH '89 Proc.)* 23, 3, 315-324.