Hardware Challenges for Ray Tracing and Radiosity Algorithms

Frederik W. Jansen, Arjan J. F. Kok and Theo Verelst

ABSTRACT
Computer graphics algorithms and graphics hardware have mainly been developed along two lines: real-time display and realistic display. Real-time display has been achieved by developing dedicated hardware for projective, depth-buffer display algorithms. Increased realism has been achieved by ray tracing and radiosity algorithms, which generally are implemented on standard workstations because the complexity of the computation makes it difficult to implement these algorithms in hardware. In this paper we review these different approaches and discuss the feasibility of using special hardware to enhance the radiosity and ray tracing computation. In particular we will explore the use of the intersection of a frustrum of rays with patches in a scene as a basic computational primitive for these algorithms and their implementation in hardware.
Keywords and phrases: rendering, radiosity, ray tracing, graphics hardware, parallel processing.

1.1 Introduction

Over the last decade, computer graphics research has been very successful in achieving two aims: increased real-time display and increased realism of display. However, both aims have not been realized so far within one approach. Real-time display has been successfully achieved by implementing the viewing pipeline of the projective depth-buffer hidden- surface algorithm in special hardware, and increased realism has been achieved by extending the traditional ray tracing algorithm to include also diffuse interreflection and soft shadows.

Since the introduction of the geometry engine [9], the depth- buffer based display systems have shown a steady increase in performance, both in speed and quality. Starting with a display rate of 30,000 polygons per second in the early eighties, current systems are now able to display more than a million polygons per second, allowing display of reasonable complex scenes in real-time [41, 1, 24]. Although the depth-buffer algorithm, inherently a projective algorithm, is not able to handle optical effects such as shadows, highlights and mirroring reflections in a natural way, several techniques have been developed to enhance realism by adding textures, anti-aliasing, motion-blur, depth-of-field, etc. [17]. Also diffuse interreflection and area light sources have been incorporated by adding a radiosity preprocessing pass that subdivides the scene into a mesh of small surface patches and elements, and that calculates the exchange of energy between these patches to account for the diffuse interreflection between surfaces [11, 29]. Rendering these elements makes it possible to display scenes in real-time while still maintaining a high-degree of shading accuracy [5, 6, 4]. Summarizing, within the paradigm of depth-buffer-based projective display of polygons a whole set of techniques have been developed to increase realism without sacrificing real-time performance. However, the projective approach will never be able, despite all clever tricks, to achieve real realism, because it lacks the possibilities of true mirroring reflections and subtle shadowing effects.

The other display paradigm, the family of ray tracing based display techniques, has

always been appreciated for its high-quality rendering capabilities. The initial 'recursive' ray tracing algorithm [45] did effectively model shadows and optical effects such as mirroring reflection and transparancy. With stochastic ray tracing the repertoire of optical effects was further expanded to anti-aliasing, soft shadows, motion blur and depth-of-focus [13, 15, 27, 28]. The addition of Monte Carlo sampling techniques to capture also the indirect light has even further increased the realism and accuracy of the illumination calculation [20], and so did improved reflection models [14] and texture filtering [18]. As realism increased, however, computation times exploded. Efficiency improving techniques that have been developed such as adaptive ray tracing [30] and spatial subdivision techniques [16] are effective, but processing times for complex scenes are still in the order of minutes and not of seconds, not to speak about tenths of seconds. For that reason ray tracing has always been a popular subject for parallel processing. Although good speed-up figures have been reported for many multi-processor systems, interactive image update rates are still not achieved. The alternative of designing special VLSI hardware, the popular route for the depth-buffer approach mentioned above, has not been tried so much for ray tracing. The efforts of Kedem and Ellis [21, 22] and Pulleyblank and Kapenga [31, 32] are the notable exceptions so far.

In line with Pulleyblank and Kapenga [31, 32], Shen et al. [35, 36, 34] have proposed to enhance the radiosity and ray tracing computation by using special VLSI hardware. They extend the earlier approach by considering the intersection computation of a frustrum of rays with a set of patches as the basic computational primitive to be supported by special VLSI hardware. In this paper we explore the feasibility of this approach both for real-time and realistic rendering.

The paper is structured as follows. In section 1.2, the requirements for realistic rendering are summarized and the state-of-the-art for global illumination calculation reviewed. In section 1.3, the different hardware approaches are discussed. In section 1.4, the basic outline of a family of ray tracing algorithms with radiosity preprocessing is given based on the computational primitive of the ray frustrum intersection. In section 1.5 some experiments are reported to give an indication of the total required computation for the different versions of this algorithm for rendering a reasonable complex scene. In section 1.6 the different options are discussed and conclusions are given.

1.2 Realism in computer graphics

Realism can only be achieved by a combination of sophisticated modeling and rendering techniques, including techniques for modeling curved surfaces, specifying procedural models, applying texture sampling and filtering, light source models, local reflection models (isotropic/anisotropic, diffuse/specular reflection, refraction, absorption, etc.) and global illumination (interreflection patterns between surfaces, simulation of soft shadows, mirroring reflections and participating media). Although all of these subjects are important, global illumination is currently considered to be most crucial, in particular in applications for architecture and interior design.

To give an indication of the complexity of the interreflection problem, some of the paths travelled by the light leaving a light source before it reaches the eye are shown in figure 1.1. The situation is simplified in the sense that surfaces are assumed to be either purely diffuse or purely specular. Path 1 represents the direct diffuse reflection, path 2 the diffuse-specular reflection, path 3 the diffuse-diffuse reflection and path 4 the specular-diffuse reflection. Other possible paths, e.g. only specular (highlight) or specular-specular reflection are not included in the figure.

Standard projective algorithms (depth-buffer, scan-line, etc.) will only account for light following path 1 and for the direct specular reflection of light, however without shadow testing. Standard ray tracing does sample light following path 1 (including shadow detection) and 2 but it does not account for the indirect reflection of light as a result of the diffuse interreflection between surfaces in the scene (path 3) and also not for the light that is first reflected by a specular surface before it is diffusely reflected by a visible surface (path 4). To capture this light it will be necessary to cast at each intersection of a viewing ray with a diffuse surface, additional secondary rays into the directions of all other surfaces to capture the light that is reflected by these surfaces, and even then it is very unlikely that the light will exactly hit the light source as suggested by path 4.

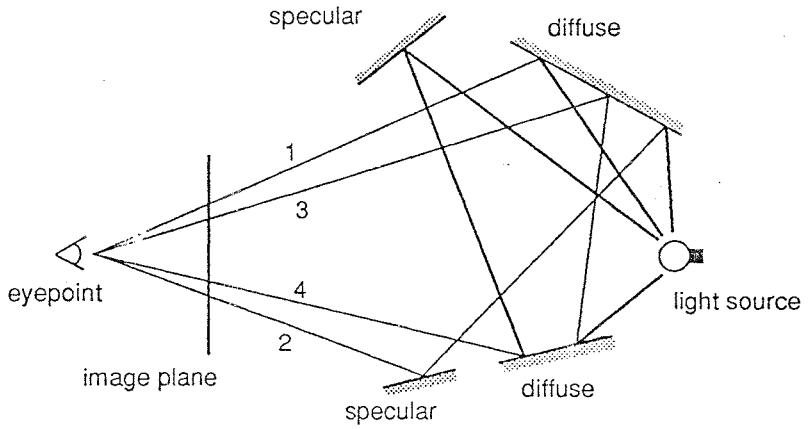

FIGURE 1.1. Different paths of light reflection

A better approach therefore is to do a preprocessing, also known as the radiosity pass, to access the global light distribution in a scene and to precompute the amount of light that each surface receives from its environment [11, 29]. This radiosity pass can be done either by calculating the energy exchange between surfaces in the scene by simultaneous solving a set of linear equations, or by a progressive radiosity method that 'shoots' light from light sources to other patches, light which in turn is re-shot to other surfaces, and so on, recursively, until a good enough approximation of the final light distribution is reached [10]. Because it is not feasible to do such a calculation for each point in the scene, it is done only for a limited number of samples points, and the value of intermediate points is interpolated from these. Alternatively, the surfaces can be subdivided into smaller patches or elements and the energy exchange between these surface elements can be calculated. To avoid seeing the boundaries between the elements, the values over the patches are also smooth interpolated.

The resolution of the sample points (resp. the size of the patch subdivision) is a critical factor in the efficiency and quality of the radiosity calculation. A resolution too high will give a too expensive and too accurate solution while a resolution too low will not be adequate to represent correctly the shading gradients. Therefore adaptive meshing techniques have been developed that provide locally a higher resolution to accommodate shading discontinuities [12]. A further improvement can be obtained by applying an exact meshing technique that lets the boundaries of elements coincide with shading discontinuities [7, 19]. In that case, however, it is inevitable that *a priori* knowledge about shading discontinuities is available, obtained for instance by projecting surface contours onto other patches. This is of course a very expensive and complex kind of (object-space-oriented)

preprocessing, in particular if curved surfaces are involved.

Summarizing, for high-quality rendering there are the following combinations of preprocessing and display (see also table 1.1):

- *projective (depth-buffer) display with a radiosity preprocessing* (alg. 1 in table 1.1); here an extensive preprocessing is needed to accurately represent shading discontinuities because the results of the preprocessing are directly displayed as the 'shading' of the surfaces (including shadows); the mesh should therefore be as accurate as possible, possibly exact. Although the preprocessing may take a considerable amount time, the display can be done at interactive rates, and is therefore attractive for walk-through applications [6].

- *ray tracing algorithms with a radiosity preprocessing*; also known as the two-pass radiosity methods [42, 43]; here several versions are possible (alg. 2a, 2b, 2c in table 1.1); the first one is a ray tracing algorithm that takes the precomputed radiosity value as the diffuse intensity of the patch and only adds the specular reflection component to it by tracing secondary rays [40]; this version requires a preprocessing that is comparable with the projective display algorithms above because the shadows from the major (point) light sources are implicitly included in the radiosity shading; the second version only uses the precomputed radiosity intensity as an improved 'ambient' term and it re-samples the light from the most important light sources and patches to calculate more accurate shadows [37, 8, 26, 25]; this version does a source selection or source classification during the radiosity pass to determine which patches can be considered as important light sources; the contributions of these selected sources are then not included in the precomputed radiosity values; finally, the third version re-samples all the light by shooting secondary rays to all directions [33]; now the radiosity shading is not used at all for display but only to quantify the light that is diffusely reflected by each patch and that is sampled during the rendering by the secondary (shadow) rays.

- *ray tracing algorithms without a radiosity preprocessing* (alg. 3 in table 1.1); these algorithms sample all the light by shooting secondary rays into all directions, but now these secondary rays hit other surfaces for which no radiosity intensity is known, and thus the sampling has to be done recursively [20]; sampling efficiency can be improved by applying importance sampling strategies [38, 2, 23] and by exploiting coherence, for instance in the form of 'illuminance caching' [44].

In fact the last two algorithms can be generalized and merged into one algorithm when the recursive sampling is combined with a radiosity pass. For instance at a certain level of recursion it may be advisable to take a precomputed value instead of continuing the sampling or taking an arbitrary intensity value. Whether recursion is only done to the first level, 'one-level path tracing' [33], or deeper can be made dependent on the intensity of the patch or the chance of shading discontinuities (highlights, shadow boundaries, etc.) in the neighborhood. The radiosity pass can be less extensive if one accepts to sample deeper, and vice versa.

The choice of algorithm for a certain application is of course very much dependent on a mix of the following factors: the quality of the picture (e.g. shadow accuracy), the display rate (e.g. real-time, interactive, or overnight), the amount of preprocessing time and memory use that can be accepted, and finally also the available hardware.

1.3 Hardware approaches

Having reviewed the different preprocessing and display algorithms, in this section we discuss the possible hardware platforms.

Dominant in the market are (high-end) graphics workstations with one or multiple

algorithm	meshing	display	light source sampling	shadow accuracy	time
1	extensive, exact	depth-buffer	no	dependent on mesh	real-time
2a	extensive, exact	ray-tracing	no	dependent on mesh	long
2b	moderate	ray-tracing	yes	good	longer
2c	low	ray-tracing	sampling in all directions	better	even longer
3	no	ray-tracing	recursive sampling	better	longest

TABLE 1.1. Different versions of high quality rendering methods

fast processors and additional hardware support for fast display of polygon models; most workstations with more than one processor work in shared-memory mode. The natural algorithm for this type of workstation is the projective depth-buffer display algorithm with a radiosity preprocessing with accurate or even exact meshing.

A second category of multi-processor workstations comprises the systems with distributed memory (i.e. transputer systems). Ray tracing is a popular subject for this type of system. A large number of implementations of radiosity and ray tracing algorithms have been published in recent years. All have shown good to excellent speed-up rates. Overall performances however are still slow in particular for cases where the object database is too big to be duplicated on each processor and has to be distributed over the processors. Until the communication bandwidth will have been improved, no real-time performances can be expected from these systems.

Special VLSI implementations of ray tracing algorithms have been scarse. Well-known is the ray casting engine of Kedem and Ellis [21, 22] which to our knowledge has been built. Another design was published by Pulleyblank and Kapenga [31, 32]. This work was done at the VLSI-group of the Electrical Engineering department of the TU Delft. Work on this subject has since then continued and was extended to (two-pass) radiosity algorithms [46]. In 1990 a project was started to develop a 'radiosity engine' in the form of a plug-in board to enhance the performances of standard workstations for high-quality rendering. Although not aiming at real-time performances as such, it should give at least an order of magnitude speed-up for the radiosity calculation compared to conventional hardware approaches.

As the basic computational primitive for hardware implemen ation was chosen the intersec ion of a frustrum (hemisphere or part of hemisphere) of rays with a set of patches (polygons or bicubic patches). Given several computational units to calculate the ray-patch intersection, it was assumed that the main bottle-neck would be the communication between the patch database, managed by the host, and the intersection computation units. Assuming coherence among neighboring rays in a frustrum, it was envisioned that if the ray frustum could be subdivided into segments in a way that would reflect the patch distribution, then the number of calls to the database would be minimal. For this purpose a special data structure was devised that segments the ray frustrums into sectors (see figure 1.2).

All rays in one sector are loaded on one intersection computation unit. The size (angle)

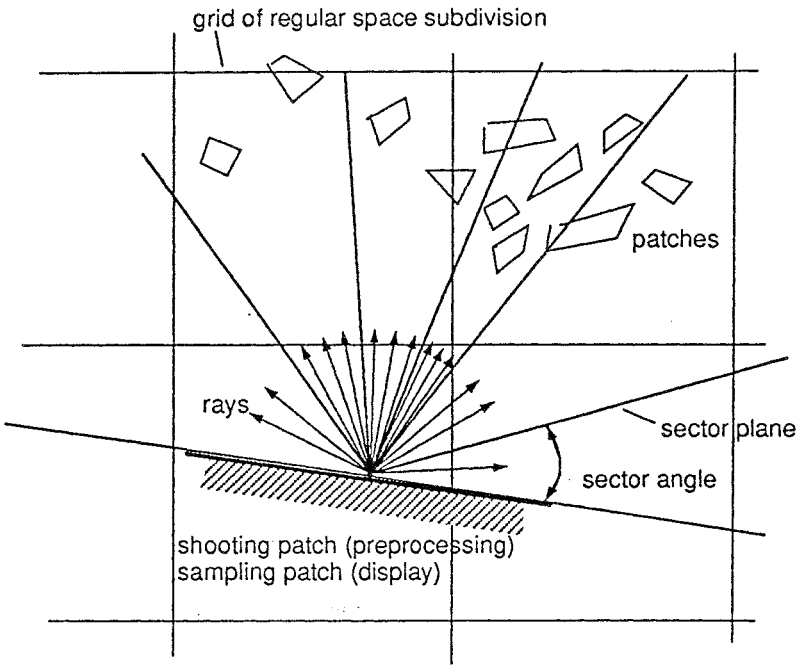

FIGURE 1.2. Ray frustrum and sectors

of the sector is made dependent on the patch density and distribution, and thus on the expected computation load, to ensure a good load balancing over the various intersection units. A further reduction in the communication with the central database has been achieved with a hierarchy of caches. See for further details and simulation results [35, 36, 34].

1.4 Algorithms for a radiosity engine

It is interesting to speculate about possible algorithms that would fit the radiosity engine concept and in particular the computational primitive of 'ray frustrum intersection' as explained above. First of all note that ray frustrums can be used both in the radiosity preprocessing for shooting, as well as in the rendering pass for sampling. The computational primitive of ray frustrum shooting/sampling is thus indeed very versatile in the context of radiosity and ray tracing algorithms and probably will take care of the larger part of the total computation load.

However, there is also a drawback. The ray frustrum method is mainly intended for *undirected* shooting and sampling (see figure 1.3), which is to avoid overhead at the host for determining the number and directions of the rays that would leave the intersection computation units idle for certain moments. This means that rays are cast without aiming at a specific patch or a specific point (e.g. a vertex). This fits well in the context of a Monte Carlo type of sampling (undirected shooting) but not very well in a progressive radiosity method as in [43]. See for a discussion on the advantages and disadvantages of directed and undirected sampling [39] where these are called implicit and explicit sampling.

The undirected sampling poses also some additional constraints on the resolution of the mesh and the resolution of the rays (see again figure 1.3). If the number of rays is too low and the mesh resolution too high then some elements of the mesh will not receive a contribution; this is likely to happen because as the distance over which the rays travel

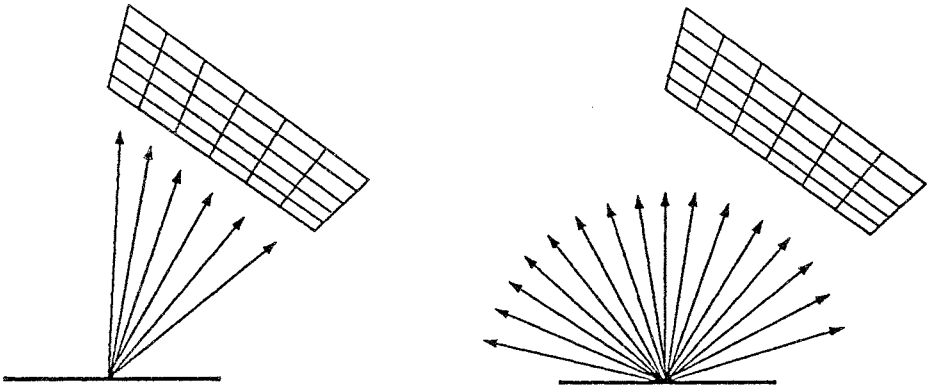

FIGURE 1.3. Directed (left) versus undirected shooting (right)

increases, the rays will get more separated and thus the mesh resolution can never be optimally adapted to the ray density. This can be accommodated by using an hierarchical mesh data structure that assigns intensity values to levels corresponding with the density of the receiving rays [3]. This would require some extra memory. At the end of the radiosity pass, the different levels could then be merged to obtain the radiosity values of the patch or vertices.

Nevertheless, there will always be an order of magnitude difference in efficiency between directed and undirected shooting (see also section 1.5). It will therefore almost be inevitable (unless the hardware is extremely fast) to find some compromise between both extremes. The only control within the undirected shooting is the choice of number of rays for each sector and the distribution of rays within each sector. These parameters can be made dependent on different factors, for instance: the (expected) density of the patches in a sector, the sizes of the patches, the distance of the patches from the ray origin, the intensity of the patches (during display), the chance of shading discontinuities (both during radiosity preprocessing and display), the reflection properties of the patches, the resolution of the mesh on the patches, etc., etc. Also source selection and source classification criteria could be applied here [25, 8]. The ray density would then reflect the 'importance' of the shooting/sampling direction of that sector. In this way the undirected shooting would take over characteristics of the directed shooting and sampling. However, the parameter estimation should not take too much overhead and should not require too much a priori knowledge about which patches actually will be intersected.

Summarizing, and taking into account all considerations, the radiosity engine could be used in the following way (see figure 1.4):

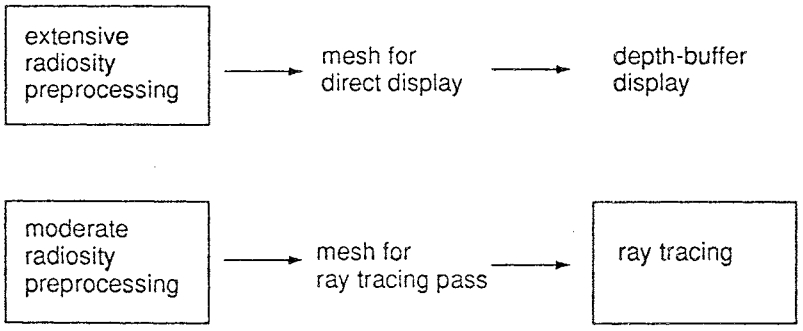

FIGURE 1.4. Combinations of radiosity preprocessing and display algorithms

- *for a radiosity pass only:* this is a standard progressive radiosity algorithm; starting with the patch with the highest intensity/energy level, energy is distributed by shooting frustrums of rays into the scene until the energy distribution converges to a solution; the mesh can dynamically be adapted to the required resolution (which also may be viewpoint dependent); a final result is obtained in the form of explicit elements that can be converted to polygons for direct display with conventional display hardware, if needed in a viewpoint dependent way.

- *for a radiosity pass and a ray tracing pass:* the radiosity pass is similar to the radiosity preprocessing above, except that the preprocessing need not be so extensive and the mesh resolution can be coarse. The mesh is used in the second pass not for display but to provide secondary rays with intensity values. If a secondary ray hits a surface then sampling can be continued (recursive sampling) or the precomputed radiosity value can be taken as the sampling intensity.

For both uses, the number of sectors and the density distribution of the rays within each sector to find an optimal sampling strategy, will be an important factor for efficiency.

1.5 Experiments

The differences between directed and undirected shooting can very well be illustrated with the example of a light source in front of a rectangular patch. The patch is regularly subdivided into 256 elements. Figure 1.5 shows the result of the radiosity pass (shooting from the source to the patch). Directed shooting takes 289 rays (figure 1.5h right-below). A comparable quality can only be obtained with more than 100,000 rays shot in a uniform way (see figure 1.5a-g).

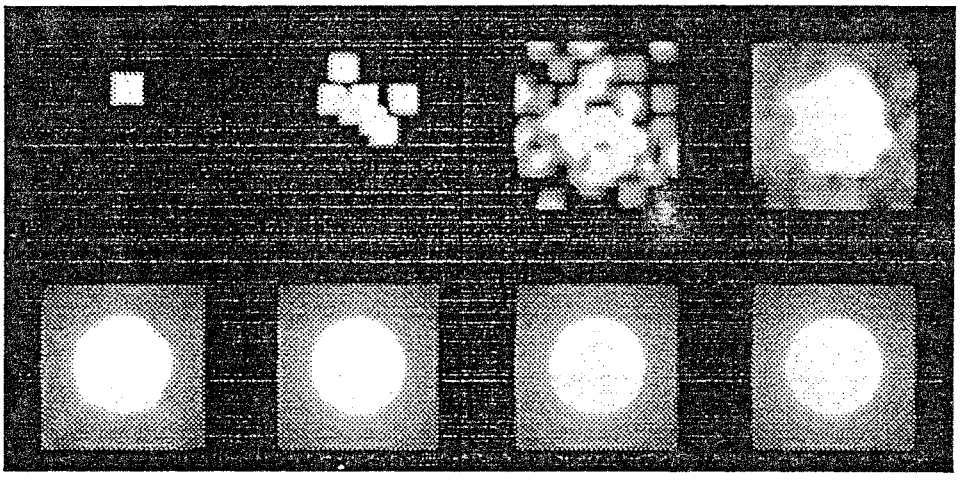

FIGURE 1.5. Results of radiosity pass for a patch in front of a light source. Undirected shooting a: 1, b: 10, c: 100, d: 1000, e: 10000, f: 100,000 rays, and directed shooting: h: 289 rays

A radiosity pass for a simple scene will give the following results (see figure 1.6); for the directed version (figure 1.6a), and for the undirected version (figure 1.6b) almost the same number of rays (approximately 5000) per shooting (per hemisphere in undirected shooting) is used.

The undirected version shows deficiencies due to mismatch between the hemisphere resolution and the meshing of the receiving patches. Also, the positions of the hemispheres are noticeable at those places where the receiving patches are close to the shooting patches

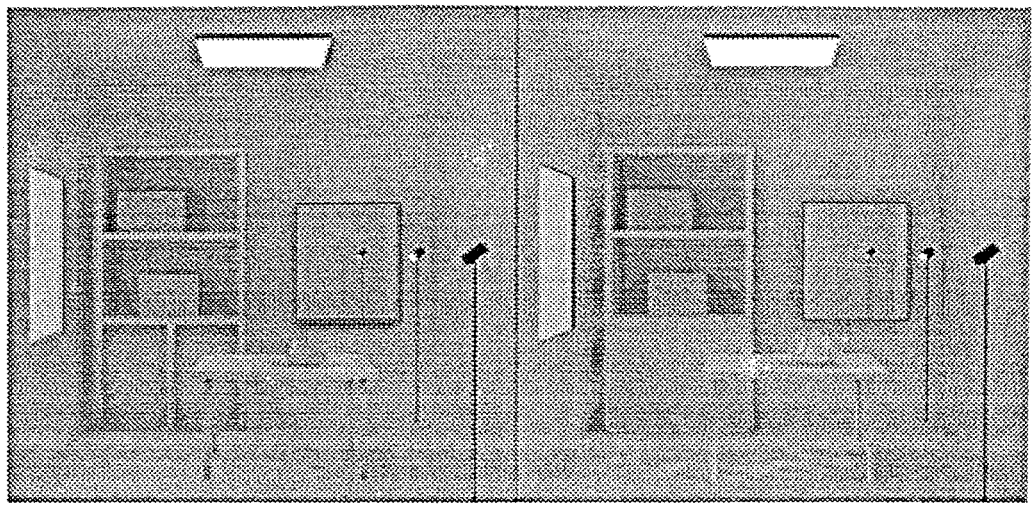

FIGURE 1.6. Results of a radiosity pass for a simple scene; directed shooting (left) and undirected shooting (right)

(see wall left of cabinet). This illustrate the importance of some kind of control over the number of hemispheres, and the ray density and the directions of the rays within a frustrum sector. However, too complicated calculations to adapt the ray directions and density to the patch distribution in a sector does not fit very well within the hardware approach, but some form of directional control will be needed. This will be subject for further research.

1.6 Conclusions

To our opinion a radiosity engine based on the general computational primitive of a ray frustrum can be a versatile piece of hardware to enhance high quality rendering. First of all it can be used to speed up the radiosity calculations in a system that uses depth-buffer hardware for real-time display as well in a system that uses a two pass method, and secondly it can be used as a rendering engine for a two-pass radiosity algorithm.

Comparing the approach with other algorithms and hardware solutions, it has to be taken into account that undirected shooting/sampling is less efficient than directed shooting and therefore is indeed a pure brute-force approach. At the other hand, this approach is better suited for parallel implementation on specialised VLSI hardware. The overall performance of the radiosity engine will depend to a large extent on the ability to tune the ray density and directions within each frustrum sector to the patch distribution.

Acknowledgements

The ideas reported in this paper have grown out of discussions with Ed Deprettere and Li-Sheng Shen of the Radiosity Engine Project of the Networktheory Section at the Faculty of Electrical Engineering of Delft University of Technology.

1.7 References

[1] K. Akeley and T. Jermoluk. High-performance polygon rendering. *Computer Graphics 22(4): 239-246, Siggraph'88*, 1988.

[2] J. Arvo and D. Kirk. Particle transport and image synthesis. *Computer Graphics 24(4):63-66, Siggraph'90*, 1990.

[3] F. Asensio. A hierarchical ray casting algorithm for radiosity shadows. *Proceedings of the 3rd Eurographics Workshop on Rendering, Bristol*, 1992.

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

[5] D.R. Baum, H.E. Rushmeier, and J.M. Winget. Improving radiosity solutions through the use of analytically determined form-factors. *Computer Graphics 23(3): 325-334*, 1989.

[6] D.R. Baum and J.M. Winget. Real time radiosity through parallel processing and hardware acceleration. *Computer Graphics 24(2): 67-75*, 1990.

[7] A.T. Campbell and D.S. Fussell. Adaptive mesh generation for global diffuse illumination. *Computer Graphics 24(4): 155-164, Siggraph'90*, 1990.

[8] S.E. Chen, H. Rushmeier, G. Miller, and D. Turner. A progressive multi-pass method for global illumination. *Computer Graphics 25(4): 165-174, Siggraph'91*, 1991.

[9] J. Clark. The geometric engine: A VLSI geometry system for graphics. *Computer graphics 16(3): 127-133, Siggraph'82*, 1982.

[10] M.F. Cohen, S.E. Chen, J.R. Wallace, and D.P. Greenberg. A progressive refinement approach to fast radiosity image generation. *Computer Graphics 22(4): 75-84, Siggraph'88*, 1988.

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

[12] M.F. Cohen, D.P. Greenberg, D.S. Immel, and P.J. Brock. An efficient radiosity approach for realistic image synthesis. *IEEE Computer Graphics and Applications 6(3): 26-35*, 1986.

[13] R.L. Cook. Stochastic sampling in computer graphics. *ACM Transactions on Graphics 5(1): 51-72*, 1986.

[14] R.L Cook and K.E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics 1(1): 7-24*, 1982.

[15] M. Dippé and E.H. Wold. Antialiasing through stochastic sampling. *Computer Graphics 19(3): 69-78, Siggraph'85*, 1985.

[16] A.S. Glassner. Introduction to ray tracing. *Academic Press*, 1989.

[17] P. Heaberli and K. Akeley. The accumulation buffer: Hardware support for high-quality rendering. *Computer Graphics 24(4): 309-318, Siggraph'90*, 1990.

[18] P.S. Heckbert. Survey of texture mapping. *Computer Graphics and Applications 6(11): 56-67*, 1986.

[19] P.S. Heckbert. Discontinuity meshing for radiosity. *Proceedings of the 3rd Eurographics Workshop on Rendering*, 1992.

[20] J.T. Kajiya. The rendering equation. *Computer Graphics 20(4): 143-150, Siggraph'86*, 1986.

[21] G. Kedem and J.L. Ellis. The ray casting machine. *Proc. IEEE Int. Conf. on Computer Design: VLSI in Computers (ICCD'84), IEEE Computer Society Press, 533-538*, 1984.

[22] G. Kedem and J.L. Ellis. *Chapel Hill Conference on VLSI*, 1985.

[23] D. Kirk and J. Arvo. Unbiased sampling techniques for image synthesis. *Computer Graphics 25(4): 153-156, Siggraph'91*, 1991.

[24] D. Kirk and D. Voorhies. The rendering architecture of the DN10000VS. *Computer Graphics 24(4): 299-307, Siggraph'90*, 1990.

[25] A.J.F. Kok and F.W. Jansen. Source selection for the direct lighting computation in global illumination. *Proceedings of the 2nd Eurographics Workshop on Rendering*, 1991.

[26] A.J.F. Kok, F.W. Jansen, and C. Woodward. Efficient complete radiosity ray tracing using a shadow coherence method. *Report of the Faculty of Technical Mathematics and Informatics, nr. 91-63, 1991. Submitted for publication*, 1991.

[27] M.E. Lee, A. Redner, and S.P. Uselton. Statistically optimized sampling for distributed ray tracing. *Computer Graphics 19(3): 61-67, Siggraph'85*, 1985.

[28] D.P. Mitchell. Generating antialiased images at low sampling densities. *Computer Graphics 21(4): 65-72, Siggraph'87*, 1987.

[29] T. Nishita and Nakamae E. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. *Computer Graphics 19(3): 23-30, Siggraph'85*, 1985.

[30] J. Painter and K. Sloan. Antialiased ray tracing by adaptive progressive refinement. *Computer Graphics 23(3): 281-288, Siggraph'89*, 1989.

[31] R.W. Pulleyblank and J. Kapenga. VLSI chip for ray tracing bicubic patches. *In: Advances in Computer Graphics Hardware I, Springer Verlag, Proceedings First Eurographics Workshop on Hardware, 125-140*, 1986.

[32] R.W. Pulleyblank and J. Kapenga. The feasibility of a VLSI chip for ray tracing bicubic patches. *Computer Graphics and Applications 7(3): 33-44*, 1987.

[33] H. Rushmeier. Realistic image synthesis for scenes with radiatively participating media. *PhD thesis, Cornell University*, 1988.

[34] L.-S. Shen and E. Deprettere. A parallel-pipelined multiprocessor system for the radiosity method. *Seventh Eurographics Workshop on Graphics Hardware*, 1992.

[35] L.-S. Shen, E. Deprettere, and P. Dewilde. A new space partitioning for mapping computations of the radiosity onto a highly pipelined parallel architecture (I). *Fifth Eurographics Workshop on Graphics Hardware*, 1990.

[36] L.-S. Shen, F.A.J. Laarakker, and E. Deprettere. A new space partitioning for mapping computations of the radiosity onto a highly pipelined parallel architecture (II). *Sixth Eurographics Workshop on Graphics Hardware*, 1991.

[37] P. Shirley. A ray tracing method for illumination calculation in diffuse specular scenes. *Proceedings Computer Graphics Interface: 205-212*, 1990.

[38] P. Shirley and C. Wang. Direct lighting calculation by monte carlo integration. *Proceedings of the 2nd Eurographics Workshop on Rendering, Barcelona*, 1991.

[39] P. Shirley and C. Wang. Distributed ray tracing: Theory and practice. *Proceedings of the 3rd Eurographics Workshop on Rendering, Bristol*, 1992.

[40] F. Sillion and C. Puech. A general two pass method integrating specular and diffuse reflection. *Computer Graphics 23(3): 335-344, Siggraph '89*, 1989.

[41] J.G. Torborg. A parallel processor architecture for graphics arithmetic operations. *Computer Graphics 21(4): 197-204, Siggraph'87*, 1987.

[42] J.R. Wallace, M.F. Cohen, and D.P. Greenberg. A two-pass solution to the rendering equation: A synthesis of ray tracing and radiosity methods. *Computer Graphics 21(4): 311-320, Siggraph'87*, 1987.

[43] J.R. Wallace, K.A. Elmquist, and Haines E.A. A ray tracing algorithm for progressive radiosity. *Computer Graphics 23(2):315-324, Siggraph'89*, 1989.

[44] G.J. Ward, F.M. Rubinstein, and R.D. Clear. A ray tracing solution for diffuse interreflection. *Computer Graphics 22(4): 85-92, Siggraph'88*, 1988.

[45] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM 23(6): 343-349*, 1980.

[46] A.C. Yilmaz, C. Hagestein, E. Deprettere, and P. Dewilde. A hardware solution to the generalized two-pass approach for rendering of artificial scenes. *Advances in Graphics Hardware IV, Proceedings Eurographics Hardware Workshop 1989, 65-79*, 1989.