Contents

1

Divide and Conquer Radiosity

R. van Liere

ABSTRACT
This paper presents a coarse-grain parallel algorithm for solving the radiosity method. It describes a technique that partitions a large scene into a number of independent subscenes. The well known progressive refinement solution process will be applied to each subscene. After a fixed number of iterations each subscene will transfer accumulated energy to its neighbor.
Although we have limited ourselves to only diffuse scenes, the algorithm can easily be extended to specular scenes.

1.1 Introduction

During the past few years the field of image synthesis has been dominated by two trends : visual realism and interactivity. On one hand, considerable research is being done on producing increasingly realistic global illumination models. On the other hand, mainly due to the high expectations of future workstations, a trend towards interactive algorithms has been set. Within this context, novel techniques are being developed that will allow global illumination models to be solved at interactive speeds.

a limited, but in many cases adequate, global illumination model. Originally, the radiosity method was restricted to environments consisting of only ideal diffuse reflectors and emitters. Later extensions, [6] and [8], have relaxed these restrictions somewhat so that specular and translucent reflectors and emitters can now also be taken into account.

Implementations of the initial radiosity method resulted in the so-called hemi-cube methods, [4], and its superior progressive reformulation, [3]. Subsequent extensions resulted in various two pass approaches which use hemi-cube method to determine the diffuse to diffuse surface reflections and ray-tracing techniques to take the specular surfaces of the environment into account, [9] and [8].

Recently, some researchers have also focussed their attention to parallel implementations of the radiosity method. Reker et. al., [7], use multiple workstations to solve a number of steps of the progressive refinement method in parallel. This method applies the server/client approach in implementing a coarse-grain parallel solution. It relies on future network technology to solve the high communication band-width between clients and server. Baum et. al., [2], is an example of a finer grain parallel solution. This approach is unique in that it allows the end user to steer the radiosity calculations during the solution process. Finally, a completely alternative approach has been taken in a recent publication by Xu et. al., [10], in which a mathematical formulation is given of a method that subdivides a scene into multiple smaller scenes.

In this paper we present a parallel algorithm for solving the radiosity equations. We describe a technique to partition the complete set of radiosity equations into a number of independent smaller sets. Although we have limited ourselves to only diffuse environments, the algorithm can easily be extended to specular environments.

This paper is organized as follows: we first review the governing radiosity equations which are relevant for our environment. Next, we discuss the algorithm. Finally, we discuss convergence and error analysis of the method.

1.2 The Radiosity Equations

In [8], Rushmeier and Torrance extend the radiosity method to include specularly reflecting and translucent surfaces. Applying their results to surfaces that are *only* diffuse or ideal specular transmitting, we arrive at the following intensity equations:

- for ideal specular transmitting surfaces :

$$I_{o,n}(\theta_{o,n}, \phi_{o,n}) = I_{o,n(t)}(\theta_{o,n(t)}, \phi_{o,n(t)})$$

  Intuitively, this equation says that the outgoing intensity of surface n is equal to the outgoing intensity of the surface, denoted as $n(t)$, that intersects a ray specularly transmitted through surface n.

  Note that the intensities of ideal specular transmitting surfaces are direction dependent. This is indicated by the two angles, $\theta_{o,n}$ and $\phi_{o,n}$, denoting the polar and azimuthal angles.

the used notation.

- for diffuse surfaces :

$$I_{o,m} = I_{e,m} + \rho_m \sum_{n=1}^{N} \{I_{o,n} F_{mn} + \tau_n \sum_{q=1}^{N} I_{o,q} T_{f,nmq}\}$$

Intuitively, this equation states that the outgoing intensity of diffuse surface m is equal to its emission plus a term due to other diffuse surfaces plus a term due to the other ideal specular transmitting surfaces.

The term due to other diffuse surfaces is the same as in the original radiosity equations for diffuse surfaces, [3]. It includes the form-factor, denoted as F_{mn}, that specifies the fraction of the energy leaving one surface which lands on another.

The term due to the ideal specular transmitting surfaces is equal to the sum of the outgoing intensities times the so-called forward window form factor, denoted as $T_{f,nmq}$. Window form factors denote the fraction of energy leaving the transmitting surface, n, from its *back* side which impings on surface m. Window form factors are direction dependent.

The terms, ρ_n and τ_n, denotes the reflectance resp. transmittance function of surface n. Since we consider only diffuse surfaces this functions are independent of direction.

These equations are diagrammed as follows :

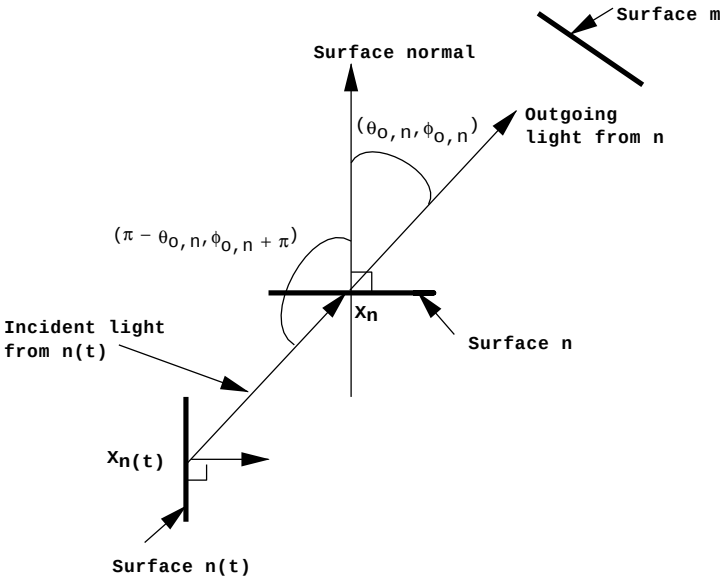

Figure 1. - Surface $n(t)$ from which light is specularly transmitted by surface n into the direction $\phi_{o,n}$ and impinging onto surface m. (Adapted from [8]).

1.3 The Algorithm

1.3.1 An intuitive view

Before going into the details, we first give a step-wise intuitive idea of how the algorithm works :

number of disjunct subscenes. Intuitively, this is done by placing a number of ideal translucent surfaces into to the scene. These translucent surfaces act as boundaries for individual subscenes. Each boundary has two sides, the front and back sides.

2. Next, apply in each subscene a number of iterations of the progressive refinement solution process. All energy that impings a boundary surface is stored in the front side of the boundary surface and will not be shot back into this subscene. Both the impinging energy and impinging angle are stored in the accumulation buffer. Note that since the subscenes are independent of each other that this step can be done in parallel for each subscene.

3. Transfer the accumulated boundary energy from one subscene to the boundary in the adjacent subscene. This is done by transferring the impinging energy from one accumulation buffer (stored in the front side) to the corresponding angle in the adjacent buffer (which will be stored in the back side of the boundary). The energy stored in the back side of a boundary will subsequently be shot into the subscene.

4. Display all subscenes without displaying the boundary surfaces. Strictly speaking, this step can be done in parallel with the energy transferring step, or after each iteration of the progressive refinement solution process in each subscene. We choose to display the scenes sequentially simply for clarity reasons.

5. Go back to step two and repeat the process.

1.3.2 A closer look

The more detailed discussion of the algorithm can best be described through the following fragments of pseudo-code.

```
subDivideScene ()

while (notConverged)
{
    foreach subScene
       doIterations (n, subScene)

    foreach boundary
       transferEnergy (boundary)

    displayScene ();
}
```

Pseudo-code fragment 1. - The main loop.

This fragment of code contains the main loop which forks off a number of "workers" each executing the function *doIterations*. Note that during execution of *doIterations* there is no communication between workers. Once the

boundary is passed on to the adjacent subscene, *transferEnergy*.

```
doIterations (n, subScene)
{
    while (n--)
    {
        shootPatch (selectPatch (subScene))
    }
}
```

Pseudo-code fragment 2. - The worker.

doIterations is a straightforward extension to the original sort and shooting algorithm. It selects a patch for shooting (the one with the greatest energy), and shoots its energy into the scene. The difference is that directions will have to be taken into account for boundary patches.

```
transferEnergy (boundary)
{
    foreach patch
    {
        transferEnergy (patch-> twin1, patch-> twin2)
        transferEnergy (patch-> twin2, patch-> twin1)
    }
}
```

Pseudo-code fragment 3. - The "synchronizer".

A boundary is divided in a number of patches. Each patch has two parts called *twins* . Twins are defined as : [1]

```
struct twin
{
    Radiosity incoming [SOLIDANGLES];
    Radiosity outgoing  [SOLIDANGLES];
}
```

in which *SOLIDANGLES* is a constant that determines the number of solid angles of the hemisphere. Note that the amount of memory used in a *twin* is proportional to the discretization accuracy of the hemisphere.

The following figure depicts a two dimensional view of the basic idea. The (progressively accumulated) incoming radiosity of *twin1* will be transferred to *twin2* which, in turn, will become a "shooting" candidate in its subscene.

[1] There is substantial room for memory optimization here.

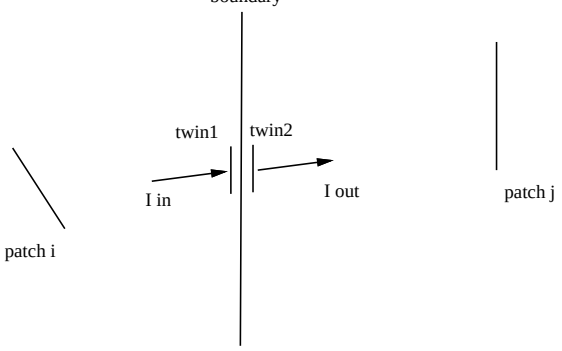

Figure 2. - Twins on a boundary.

Immel et. al., [6], have published an implementation technique that determines the radiosity in a non-diffuse environment. Aside from the convergence speed, Immel's method suffers from the vast amounts of storage needed to store directional form-factors and directional radiosity. Our method also requires a vast amount of memory. Two comments must, however, be made :

1. memory needed for the boundaries can be distributed over a number processors/workstations instead of only one central workstation.

2. additional memory is needed only for boundaries. This allows the additional memory needed to be a function of the available hardware resources and is independent of the complexity of the scene itself.

1.4 Convergence Analysis

In [3] , Cohen et al. introduce a metric for comparing convergence rates of various radiosity solution methods. The given metric provides a quantitative measure of the overall radiosity inaccuracy after each iteration in the progressive refinement solution process. The square root of the area weighted mean of the square of the the individual errors (RMS error) was used as a indication of the error made at each iteration; i.e.

$$RMS\ error = \sqrt{\frac{\sum_{i \in Env} ((I_i^* - I_i)^2 A_i)}{\sum_{i \in Env} A_i}}$$

where I_i^* is the converged intensity and I_i is the intermediate intensity of a patch i in an environment Env.

The following graph shows a hypothetical graph of the normalized RMS error set out against the number of iterations during the solution process. Function S denotes the convergence curve of the progressive refinement solution process in one environment as a function of the number of iterations; i.e. the sequential case. Function P denotes the convergence curve of the progressive refinement solution process in multiple subenvironments as a function of the number of iterations; i.e. the parallel case. Both functions will converge to the same result.

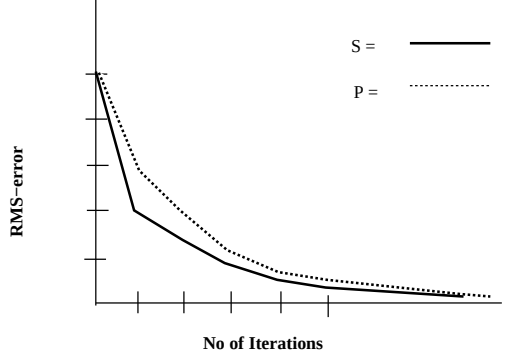

Figure 3. - Plot of normalized RMS error for the sequential (thick) and parallel (dotted) cases.

In general, the following statements can be made about the behavior of S and P :

1. For small n : $S(n) < P(n)$; i.e. after a small number of iterations the mean error made by the single environment method is smaller than in the multiple subenvironment case. This is due to a) extra boundary patches that are introduced in the subenvironment method, and b) a partial ordering is used in the case of multiple subenvironments to determine the shooting patch. Recall that patches in each subenvironment are sorted in increasing order. Partial orderings do not guarantee that a patch with the maximum accumulated energy is chosen as the shooting patch.

   Note that n is the sum of the iterations in all subenvironments.

2. For large n : $S(n) = P(n)$; i.e. both cases converge to the same mean error. This is because both methods converge to the same result.

3. For small n and $k > 1$: $S(n) > P(kn)$ with k as the number subenvironments; i.e. after n iterations in a single environment the mean error is greater than after n iterations in each subenvironment in the multiple environment case. This is because k patches are shot simultaneously. Hence, k times the number of iterations are made.

1.5 Error Analysis

The assumption made by the divide and conquer method is that introducing additional boundaries does not influence the results of light energy calculations in any way. In practice, however, additional errors are made during the form-factor calculations. The quantity of the errors depend in a complicated way on the scene being rendered. In practice, worst case errors are not very useful and "average case" errors are not well defined. We will, therefore, restrict ourselves to a qualitative analysis of errors that may occur.

Baum et al. [1] have shown that three types of errors can occur when the hemi-cube method is used to calculate form-factors :

tance between patches i and j is small compared to the diameter of patch i. The proximity assumption is violated when, for example, i and j are adjacent patches that share a common edge. Since introduction of boundaries decreases the average distance between patches, the proximity assumption will be violated more often.

2. errors due to the violation of the *visibility* assumption; i.e. the visibility of a projected patch j does not alter across the shooting patch i. The visibility assumption is violated if another patch partially occludes the projected patch. Due to the additional boundaries, the visibility assumption will tend to be violated less frequently since there are a reduced number of patches per subenvironment.

3. errors due to the violation of the *aliasing* assumption; i.e. that patch j projects exactly onto whole pixels of patch i's hemi-cube. Introduction of boundary patches cause additional violations of the aliasing assumption. These errors can be minimized by, for example, using a larger hemi-cube.

Note that the divide and conquer algorithm does not mandate that form-factors be calculated with the hemi-cube method. Other form-factor calculation methods should be equally applicable.

1.6 Conclusion

We have presented a parallel algorithm for applying the progressive refinement method to the radiosity equations. The implementation of the algorithm is well suited for multi-processor architectures because of the limited synchronization between processes.

We have also discussed convergence / error analysis of the algorithm. The convergence curve is far better than the sequential case. Additional errors are due to artifacts caused by the hemi-cube method for calculating form-factors.

The algorithm has been implemented in C++ on a network of Personal Iris workstations using SGI's graphics library for the low level rendering.

Future work will concentrate on : determining heuristics for optimal subscene division, dynamic subscene reconfiguration, minimizing memory consumption of boundaries.

Acknowledgements: Many thanks to Henk Schouten who contributed substantially to initial ideas and implementation issues. Thanks go also to Paul ten Hagen who provided me with the time and patience to work on these ideas.

1.7 References

[1] D.R. Baum, H.E. Rushmeier, and J.M. Winget. Improved radiosity solutions through the use of analytically determined form-factors. *Computer Graphics (SIGGRAPH '89 Proceedings)*, 23(3):325–334, 1989.

based lighting simulation programs. *Computer Graphics (Interactive 3D Graphics)*, 24(4):51–57, 1990.

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

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

[5] C.M. Goral, K.E. Torrance, D.P. Greenberg, and B. Bataille. Modelling the interaction of light between diffuse surfaces. *Computer Graphics (SIGGRAPH '84 Proceedings)*, 18(3):213–222, 1984.

[6] D.S. Immel, M.F. Cohen, and D.P. Greenberg. A radiosity method for non-diffuse environments. *Computer Graphics (SIGGRAPH '86 Proceedings)*, 20(4):133–142, 1986.

[7] R.J. Recker, D.W. George, and D.P. Greenberg. Acceleration techniques for progressive refinement radiosity. *Computer Graphics (Interactive 3D Graphics)*, 24(4):59–66, 1990.

[8] H.E. Rushmeier and K.E. Torrance. Extending the radiosity method to include specularly reflected and transluctent materials. *A.C.M. Transactions on Graphics*, 9(1):1–27, 1990.

[9] 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 techniques. *Computer Graphics (SIGGRAPH '87 Proceedings)*, 21(4):311–328, 1987.

[10] H. Xu, Q-S. Peng, and Y-D. Liang. Accelerated radiosity method for complex environments. In W. Strasser W. Hansman, F.R.A. Hopgood, editor, *EUROGRAPHICS '89*, pages 51–61, Hamburg, 1989. North-Holland.