# Reducing Memory Requirements for Interactive Radiosity Using Movement Prediction

Frank Schöffel
Fraunhofer Institute for Computer Graphics
Darmstadt, Germany
schoeffe@igd.fhg.de

Andreas Pomi
vrcom GmbH
Darmstadt, Germany
apomi@vrcom.de

**Abstract.** The line-space hierarchy is a very powerful approach for the efficient update of radiosity solutions according to geometry changes. However, it suffers from its enormous memory consumption when storing shafts for the entire scene. We propose a method for reducing the memory requirements of the line-space hierarchy by the dynamic management of shaft storage. We store shaft information only locally for those parts of the scene that are currently affected by the geometry change. When the dynamic object enters new regions, new shaft data has to be computed, but on the other hand we can get rid of outdated data 'behind' the dynamic object. Simple movement prediction schemes are applied, so that we can provide shaft data to the radiosity update process in time when needed. We show how storage management and pre-calculation of shafts can be efficiently performed in parallel to the radiosity update process itself.

## 1 Introduction

Realistic global illumination simulations are applied more and more in three-dimensional computer-generated environments, e.g., in Virtual Reality applications. Due to its view-independent nature, the radiosity method is very well suited for illumination simulation in those applications. However, updating the illumination in interactive environments according to modifications in scene geometry is a demanding task, since radiosity updates are expensive to calculate. Interactive update rates are hard to achieve.

Several methods for updating radiosity solutions according to scene modifications have been presented in literature, both for progressive refinement radiosity and for hierarchical radiosity, amongst which the line-space hierarchy approach [5] is one of the most powerful ones. Although this method can provide very fast update rates in the context of hierarchical radiosity, it still has several drawbacks. Its high storage demand is the most important disadvantage, prohibiting a wide-spread use of the line-space approach for complex real-world scenes.

In this paper, we address this problem and propose a method for the dynamic management of storage, thus reducing storage requirements significantly and enabling the line-space hierarchy method to be applied even to complex scenes. In the original line-space hierarchy method, shafts are stored within the entire scene. Our method limits

the spatial regions for which this data is stored to those regions in which interaction currently takes place. When an object is moved into new regions, missing shafts can be computed during radiosity update. To avoid this additional calculation, movement prediction schemes can be applied: The object movement is extrapolated for future frames, thus allowing to pre-compute shaft data so that it is available to the update process in time when needed. Furthermore, a garbage collector removes shafts that are no longer necessary behind the object. The method is scalable according to the storage available and can reduce storage requirements dramatically. It is especially useful when running on a multi-processor system, but can also be applied on single-processor machines when doing without movement prediction, taking into account slightly lower performance.

In the remainder of this paper, we briefly review previous work and then present our method for dynamic shaft management with movement prediction. We proceed with results of a first implementation and, finally, conclude and give an outlook on directions of future work.

## 2 Context and Previous Work

### 2.1 The radiosity method

The radiosity method simulates global illumination on a physical basis within diffuse environments. Improvements on the original algorithm [8] reduced the quadratic storage requirements: Progressive refinement radiosity [4] requires storage that is only linear in the number of patches, allowing radiosity simulations to be applied to complex environments. Hierarchical radiosity [10] simulates light transfer at different levels of accuracy and has also significantly less than quadratic storage costs. It subdivides the input polygons into *hierarchical elements* and creates *links*, across which energy is transferred between elements at appropriate levels. Links are initially established between all pairs of input polygons and refined according to the amount of energy transferred and other criteria. The quadratic cost of the initial linking phase is a major drawback of this method, which can be avoided by applying 'lazy linking' mechanisms [11] or clustering. The clustering approach [15][17] extends the hierarchy above the polygon level up to one single root element for the scene, and allows the efficient treatment of very complex environments.

### 2.2 Radiosity in dynamic environments

Since the radiosity process takes into account the whole scene geometry, the simulation has to be repeated whenever the geometry is modified, in order to maintain a consistent solution. However, the most important effects of geometry changes are often spatially limited, and coherence can be exploited in order to obtain an efficiently updated radiosity solution. Several approaches for fast radiosity updates have been proposed, some of which require the path of the dynamic object to be known in advance (e.g., [1]). These methods are obviously not suited for interactive applications, where the user may freely modify the scene, and therefore are not discussed in this paper.

Other approaches have been developed for both progressive refinement and hierarchical radiosity. Two very similar algorithms based on progressive refinement have been proposed by Chen [2] and George et al. [7]. These methods update an existing radiosity solution by shooting (possibly negative) 'correction values' to patches on which illumination has changed due to object movements, and they account for indirect effects by applying further iterations. In [13], an efficient data structure has been proposed by

Müller et al., enabling the efficient exploitation of coherence. However, these methods still cannot provide feedback at interactive rates for moderately complex environments.

For hierarchical radiosity, a first approach on dynamic updates was presented by Forsyth [6]. He proposed to move links up and down in the hierarchy, according to changes in occlusions and energy transfers. This idea was further developed by Shaw [14] to the idea of keeping track of link refinement and storing visibility information in 'ghost links' and 'shadow links'. Eventually, an efficient update algorithm for hierarchical radiosity and clustering, based on a line-space hierarchy, has been presented by Drettakis and Sillion [5].

### 2.3 The line-space hierarchy

This approach uses the line-space hierarchy for the rapid identification of links affected by a scene modification. The line-space between two hierarchical elements (which may be either patches or clusters) is represented by *shafts* [9] associated with the links.

While in traditional hierarchical radiosity a link that has been subdivided may be discarded, those links are kept in the line-space hierarchy but marked as *passive*. In contrast to *active* links, across which energy is transferred, passive links do not participate in energy transfer, but maintain a history of the link subdivision. The form-factor associated with these links is kept, in order to easily re-establish the links when needed.

Once an object is moved within the scene, one has to check the object's bounding boxes at its old and new position for intersection with the shafts in order to find which links are affected, descending in the shaft hierarchy. For the affected links, new form-factors have to be calculated and the links possibly have to be subdivided. On the other hand, some previous subdivisions may have become too fine and therefore passive links at higher levels have to be activated again. Finally, energy has to be gathered across the modified links, and a limited push-pull operation ensures a consistent global solution.

## 3 Reducing Memory Consumption by Movement Prediction

### 3.1 Memory requirements of the line-space hierarchy

Hierarchical radiosity approaches require a lot of memory when storing all links across which energy is transferred. For the line-space approach, memory requirements are even higher, since a complete hierarchy including passive links is kept, and usually shafts are stored for all links. While for hierarchical radiosity, link caching schemes can significantly reduce link storage [18], these methods cannot be applied to the line-space method for fast radiosity updates, where the complete link hierarchy is essential.

For fast identification of affected regions, shafts are needed to be available throughout the scene. Unfortunately, shafts are much more expensive in terms of storage than links: While a link just consists of a pointer to the element it transfers energy from and a form-factor, a shaft is made up of bounding boxes for the receiver and the sender, as well as a compound bounding box, a slab counter and up to eight slabs, altogether easily requiring more than 400 bytes per shaft, when assuming a float to be 8 bytes wide. Although this data of course can be compressed, the shafts still require a significant amount of memory. Thus, reducing the number of shafts will significantly reduce the overall memory consumption. Deleted shafts, of course, have to be re-calculated when needed for intersection tests during line-space traversal. A possible way to save this additional calculation time is to have a separate process providing shafts *just before* they are needed. Such a process could predict the regions in which shafts will be required in future frames and pre-calculate missing data.

### 3.2 Predicting object movement

For predicting future positions of a dynamic object that is moved interactively by a user just by looking at recent positions, literature offers many different methods. These approaches range from simple translation extrapolation and transformation matrix extrapolation to complicated prediction schemes taking into account kinematics and other constraints. For the purpose of roughly predicting regions into which the dynamic object will move, we do not need sophisticated methods like, e.g., Kalman filtering [12], but instead will use a fast and simple linear extrapolation of object movement.

**Extrapolation.** We define the translation vector $\vec{t}_0$ from the previous to the current object location by $\vec{t}_0 = \vec{c}_0 - \vec{c}_{-1}$, where $\vec{c}_0$ and $\vec{c}_{-1}$ are the bounding box centers of the dynamic object at the current and the previous position, respectively. $\vec{t}_0$ is added to the current position $\vec{p}_0$ of the dynamic object in order to obtain its position $\vec{p}_{+1}$ in the next frame. When adding $\vec{t}_0$ multiple times, we get the future positions for the next $n$ frames:

$$\vec{p}_{+n} = \vec{p}_0 + n \cdot \vec{t}_0 \tag{1}$$

Jittered motion in interactive systems can be smoothened by considering not only *one* previous position, but the last $m$ positions and translation vectors $\vec{t}_{-i}, i = 0, \ldots, m-1$:

$$\vec{p}_{+n} = \vec{p}_0 + \frac{1}{m} \sum_{i=0}^{m-1} \vec{t}_{-i} \tag{2}$$

In addition, the previous positions do not have to be weighted equally. For example, exponential weighting leads to the most recent positions being considered more important than older ones [3]. If not only previous bounding box positions are known, but also the object transformation matrices $T_{-i}$, then object rotation can be considered, too. For example, Eq. 1 then can be extended to:

$$\vec{p}_{+n} = T_0^n \cdot \vec{p}_0 \tag{3}$$

**Compensating for prediction errors.** The presented simple prediction scheme is fast to apply, but it may introduce errors since non-linear object movements cannot be covered very well. These errors, resulting in the predicted bounding box position being different from the actual one, are not crucial, since missing shafts can be generated *on-the-fly* during shaft testing (see Section 3.3). But, since missing shafts reduce update rates, we want to compensate for the prediction error: In order to keep things simple and fast, we scale the predicted object's bounding box by some factor $s > 1$.

Thus, the probability of missing affected shafts is reduced, but the appropriate values for $s$ have to be chosen carefully: If $s$ is too great, lots of unnecessary shafts will be generated, and if it is chosen too small, one might miss many affected shafts. An example for predicted bounding volumes is depicted in Fig. 4 (see Appendix). Note that for simplifying intersection tests we always use axis-aligned bounding boxes, even if the object is being rotated.

### 3.3 Dynamic shaft management

Since we do not store shafts for *all* links any longer, mechanisms are needed for calculating shafts when required. There are two reasons for a link not having shaft data available:

- The link has never been asked for its shaft before. This is true especially at initialisation time.
- The shaft once was available, but has been removed for saving memory.

Initially, there are virtually no shafts; a shaft is calculated only when required (or when predicted to be required). In a clustering environment, the only shaft available at startup time is the one associated with the self-link of the root cluster, which is identical with the scene's total bounding box (*root shaft*). For hierarchical radiosity without clustering, all top-level shafts (associated with the links resulting from the initial linking phase) are available and should be kept in storage. In the following, we describe the algorithm for a clustering environment, but it can also be applied directly to non-clustering hierarchical radiosity when considering all top-level shafts instead of just a single root shaft.

**Shaft generation.** One possible approach is to generate shafts at exactly the time when needed during line-space traversal. Moreover, it is not necessary to store a shaft at all—we can destroy it immediately after the intersection test has been finished and recalculate it again when required. This approach minimises storage demands, but slows down line-space traversal significantly.

Therefore, it is preferable to minimise the number of shafts to be calculated on-the-fly by the update process, and to have missing shafts be provided automatically by the movement prediction process running in parallel. We use the bounding boxes predicted as described in Section 3.2 and check for intersection with existing shafts, starting with the root shaft and traversing the shaft hierarchy. For reducing the number of line-space traversals, we check for intersection with *all* predicted volumes for future frames in *one* traversal step. If more than one dynamic object exists, all predicted bounding boxes of all dynamic objects are used during the intersection test. If an intersection between any of the bounding boxes and a shaft is found, we store that shaft, and for passive links we descend in the hierarchy and test child links accordingly. Shafts that are not available for a child link are generated before testing. If such a newly created shaft intersects with a bounding box, we store it. Otherwise, we stop descending and may discard the shaft. This process is depicted in Fig. 1.

```
CreateShaftsOnPrediction (Helem p, IndexRange idx, BBoxList bbl)
{
    for each link L of p
    {
        Helem q = L→src

        if (TestAndCreateShaft (L→shaft, bbl))
            if ((L is passive) and (q→idx ⊆ idx))
                for each child node c of p
                    CreateShaftsOnPrediction (c, q→idx, bbl)
    }
}
```

**Fig. 1.** Checking for affected links and creating associated shafts.

Links of the dynamic object itself are always affected by the modification and therefore are calculated without checking. These shafts are trivial to detect since the dynamic object is known. However, care has to be taken to keep these shafts' geometry up-to-date during object movement.

**Shaft deletion.** Shafts which are not needed any longer should be deleted in order to save storage. However, it is not easy to decide which shafts are good candidates for deletion. Shafts can be outdated for several reasons:

- If a passive link is being re-established as *active*, all its child links, including the shafts, can be removed.
- Amongst the remaining links, many shafts can be deleted, too. For example, shafts located behind the dynamic object are very unlikely to be needed in the near future, when we assume the object not to turn around suddenly.
- Wrongly predicted shafts will never be used and therefore have to be deleted. This occurs, for example, when the object moves differently from the predicted path or when the user selects another object for interaction.

While the first category of outdated shafts is trivial to identify, shafts behind the dynamic object have to be searched for: These shafts intersect previous dynamic object bounding boxes, but not future ones. Identification of wrongly predicted shafts is quite hard. We suggest to establish a *garbage collector* that deletes shafts which have not been used for a certain time.

**Garbage collection.** In order to keep the total size of shaft storage approximately constant, about the same number of old shafts should be deleted when creating new shafts. We introduce a shaft counter $N_{shafts}$ and a link counter $N_{links}$, and denote the rate of links with shaft data available by $P_{shafts} = \frac{N_{shafts}}{N_{links}}$. Defining a threshold $T_{shafts} \in [0, 1]$, we start the garbage collector whenever $P_{shafts} > T_{shafts}$.[1] This is checked after each line-space update.

The garbage collector removes those shafts which have not been used for the longest time. We add a new counter $age$ to each shaft, and we reset this counter to its inital value 0 whenever the shaft is used for an intersection test. Once the garbage collector is triggered, it traverses the link hierarchy, incrementing ages of all existing shafts. Any shaft reaching a certain threshold $age_{max}$ is deleted. The garbage collector may be stopped as soon as $T_{shafts} > P_{shafts}$, or when a lower threshold $T_{min,shafts}$ is reached by $P_{shafts}$, or it may resume, traversing the hierarchy completely. The process of garbage collection is outlined in Fig. 2.

```
GarbageCollect (Helem p, int age_max)
{
    for each child node c of p
        GarbageCollect (c, age_max)

    for each link L of p
    {
        if (L→shaft exists)
        {
            increment age_{L→shaft}
            if (age_{L→shaft} > age_max)
                DeleteShaft (L→shaft)
        }
    }
}
```

**Fig. 2.** The garbage collector.

[1] Alternatively, one could check for $N_{shafts}$ exceeding some fixed maximum number of shafts. This allows for a fixed storage size being ensured and therefore should be preferred if memory is very limited.

**Algorithm overview and parallelisation.** After each radiosity update the dynamic shaft management is triggered. Firstly, the garbage collector is started if too many shafts are stored. In a second step, the movement prediction calculates future positions of the dynamic object. If more than one object is moving, we come up with a list of predicted bounding volumes for all dynamic objects for the next $n$ frames. The line-space is traversed, and shafts are calculated as discussed above for the predicted object positions.

Although the proposed method can be applied on single processor machines for reducing memory consumption, it is especially useful if movement prediction and shaft management can be performed on a separate processor in parallel to the line-space update itself. Only in this case, shaft prediction can show to advantage, when compared to on-the-fly shaft generation. Since line-space update and shaft prediction/garbage collection can work independently, these tasks can be performed simultaneously by two processes, which we will refer to as *update process* and *shaft management process*. On the other hand, since both the shaft prediction and the update process traverse the same hierarchy, data access conflicts can occur. Therefore, data access has to be synchronised, e.g., by locking sub-trees of the hierarchy. But when both processes traverse the hierarchy in the same manner, there may occur many situations where the processes have to wait for each other, thus slowing down the whole update. To reduce the number of possible conflicts, we propose to organise hierarchy traversal in a way that both processes traverse the hierachy in opposite directions, as shown in Fig. 3 (left). The simultaneous execution of the update process and the shaft management process is depicted in Fig. 3 (right).

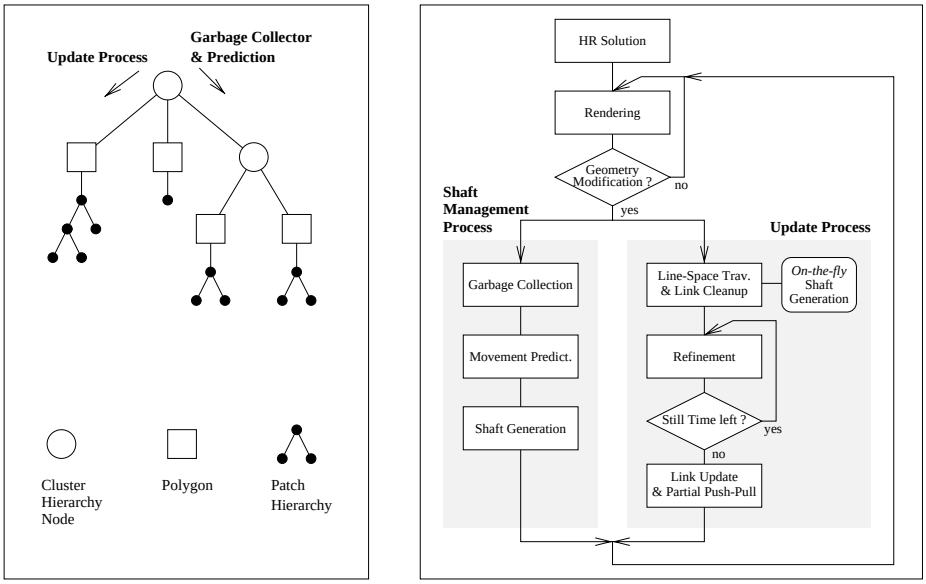


**Fig. 3.** Parallelisation of movement prediction and update process. Left: For avoiding conflicts, the link hierarchy is traversed by the update process and the shaft management process in opposite directions. Right: Performing shaft management in parallel to the line-space update. Missing shafts are generated *on-the-fly* during line-space traversal in the update process.

## 4 Results

We realised a first implementation in our hierarchical radiosity system without clustering. For movement prediction, we implemented a transformation matrix extrapolation scheme according to Eq. 3, accounting for both translation and rotation. We took into account only one previous object position and predict only one future position. Predicting future positions for $n > 1$ is difficult in practice in interactive environments, where users often change movement directions. For the scaling factor $s$ of the predicted bounding boxes, values slightly greater than 1 usually proved to be useful ($s \in (1, 1.2]$); we applied a value of $s = 1.1$ to obtain the results presented in this paper.

Dynamic shaft generation and reduction is shown in Fig. 5 (see Appendix): Shafts do only exist in the region of the chair that forms the dynamic object. New shafts are added in the region into which the chair is moved, and outdated shafts are finally deleted by the garbage collector.

In the following, we report on shaft statistics for two test scenes. Scene 1 is the test environment shown in Fig. 5 (see Appendix), the more complex scene 2 is shown in Fig. 6 (see Appendix). We have simulated only one iteration of hierarchical radiosity in both example scenes. Statistics for the hierarchical solution are given in Table 1.

**Table 1.** Number of links for the two test scenes after initial hierarchical radiosity solution.

| | scene 1 | scene 2 |
|---|---|---|
| # input polygons | 227 | 1339 |
| # links (total) | 31130 | 43679 |
| # passive links | 7932 | 10715 |
| # hierarchy elements | 32108 | 31414 |
| # hierarchy leaves | 23679 | 23447 |
| time for first HR iteration (sec) | 81.4 | 129.6 |

Calculation times reported have been measured on an SGI Onyx IR with a 195 MHz R10000 processor. All computation was performed on the same processor, i.e., shafts were computed on-the-fly during line-space update when needed. Table 2 lists the number of shafts for both test scenes. In each scene, an object has been moved three times after initial hierarchical radiosity computation (in scene 1 the chair has been moved, in scene 2 the seat was selected as dynamic object). Initially, virtually no shafts exist and, thus, for the first movement step all necessary shafts are calculated. The number of shafts to compute is significantly reduced in subsequent steps, where most of the shafts are already available. Significant memory savings (79–95 %) are achieved, compared to the storage needed when storing shafts for the complete link hierarchy.

## 5 Conclusions and Future Work

We have presented a method for controlling the memory consumption for the line-space approach. We identified the shafts as the most memory-consuming part of the line-space data structure, which can be efficiently re-calculated instead of storing. In contrast to the straight-forward approach of not storing any shafts and re-calculating them when needed during line-space traversal, we have presented an approach for the dynamical management of shaft storage. A simple movement prediction is applied to

**Table 2.** Shaft statistics for test scenes 1 and 2, where objects have been moved in three steps.

| | scene 1 | | | scene 2 | | |
|---|---|---|---|---|---|---|
| move | 1 | 2 | 3 | 1 | 2 | 3 |
| # shafts at old pos. | 1516 | 1392 | 1270 | 6208 | 6239 | 492 |
| # shafts at new pos. | 1392 | 1270 | 915 | 6239 | 492 | 485 |
| # shaft tests | 3128 | 2912 | 2728 | 18320 | 17586 | 6748 |
| # shafts generated *on-the-fly* | 1564 | 4 | 48 | 9160 | 152 | 355 |
| # shafts already existing | 0 | 1452 | 1316 | 0 | 8641 | 3019 |
| # shafts deleted by garbage coll. | 0 | 167 | 128 | 0 | 874 | 5538 |
| # shafts (total) | 1564 | 1401 | 1321 | 9160 | 8438 | 3255 |
| shaft memory needed (MB) | 0.88 | 0.79 | 0.74 | 5.15 | 4.75 | 1.83 |
| shaft memory for *all* links (MB) | 14.90 | 14.90 | 14.90 | 24.58 | 24.58 | 24.58 |
| memory savings | 94.1% | 94.7% | 95.0% | 79.0% | 80.7% | 92.6% |
| time for line-space trav. (sec) | 0.078 | 0.033 | 0.032 | 0.448 | 0.191 | 0.083 |
| time for shaft generation (sec) | 0.041 | $<$0.001 | 0.002 | 0.239 | 0.004 | 0.011 |

let us know the necessary shafts for the next frames and prepare this data in time so that they are available to the line-space traversal when needed. Furthermore, a garbage collection is introduced to get rid of outdated shaft information. The presented approach is adjustable to the available memory size. Memory consumption can be reduced significantly, which allows to apply the line-space hierarchy method also to complex environments. We believe that even greater savings will be obtained when applying our method to more complex scenes than the test scenes described in the previous Section, and we therefore intend to do extensive tests on more complex real-world scenes.

While it is possible to apply this approach on a single processor machine for a better control of shaft storage, it comes to full advantage when the shaft pre-calculation and shaft deletion can be performed simultaneously with the line-space update itself on a separate processor. We have outlined how parallelisation can be organised. Even when performing shaft calculation on-the-fly, additional computation time turned out to be very small compared to line-space update time. Nevertheless, we intend to study in more detail the improvements achieved by parallel execution.

We feel that better prediction methods will not have great benefits in this context, since movements are not always smooth in interactive applications. Therefore, we chose a simple prediction method which is fast to perform. However, it should be investigated whether more exact movement prediction methods can improve the ratio of correctly predicted shafts, or if a rough but fast method performs better when combined with an efficient garbage collector.

An interesting direction for further research is to combine the presented approach with importance-driven approaches similar to [16]. Thus, update rates can be increased by focussing on the most 'important' regions of the scene first, and accordingly storage can be further reduced when only 'important' shafts are generated and held in memory.

Finally, of course, further parallelisation efforts can help to speed up the update process. For example, the line-space traversal itself could be split into parallel subtasks, and load balancing issues would then have to be investigated in order to gain maximum speed-up.

## 6 Acknowledgments

This work was funded in part by the European Union (Esprit LTR 24944: ARCADE). The authors wish to thank George Drettakis and François Sillion for fruitful dicussions and for providing helpful information about details on the line-space hierarchy.

## References

1. Baum, D., Wallace, J., Cohen, M., Greenberg, D.: The back-buffer algorithm: an extension of the radiosity method to dynamic environments. *The Visual Computer*, 2(5):298–306, 1986.
2. Chen, S.: Incremental radiosity: An extension of progressive refinement radiosity to an interactive image synthesis system. *Computer Graphics (Proc. SIGGRAPH '90)*, 24(4):135–144, August 1990.
3. Chim, J., Lau R., Si, A., Leong, H., To, D., Green, M., Lam, M.: Multi-resolution model transmission in Distributed Virtual Environments. *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST '98)*, pages 25–33, November 1998.
4. Cohen, M., Chen, S., Wallace, J., Greenberg, D.: A progressive refinement approach to fast radiosity image generation. *Computer Graphics (Proc. SIGGRAPH '88)*, 22(4):75–84, August 1988.
5. Drettakis, G., Sillion, F.: Interactive update of global illumination using a line-space hierarchy. *Computer Graphics (Proc. SIGGRAPH '97)*, 31(3):57–64, August 1997.
6. Forsyth, D., Yang, C., Teo, K.: Efficient radiosity in dynamic environments. In Sakas, G. et al. (eds.): *Photorealistic Rendering Techniques*, pages 313–323, Springer-Verlag, 1995. Proc. 5th Eurographics Workshop on Rendering (Darmstadt, 1994).
7. George, D., Sillion, F., Greenberg, D.: Radiosity redistribution for dynamic environments. *IEEE Computer Graphics and Applications*, 10(4):26–34, July 1990.
8. Goral, C., Torrance, K., Greenberg, D., Battaile, B.: Modeling the interaction of light between diffuse surfaces. *Computer Graphics (Proc. SIGGRAPH '84)*, 18(3):213–222, July 1984.
9. Haines, E., Wallace, J.: Shaft culling for efficient ray-traced radiosity. Proc. 2nd Eurographics Workshop on Rendering (Barcelona, 1991). Proc. 2nd Eurographics Workshop on Rendering (Barcelona, 1991).
10. Hanrahan, P., Saltzman, D., Aupperle, L.: A rapid hierarchical radiosity algorithm. *Computer Graphics (Proc. SIGGRAPH '91)*, 25(4):197–206, August 1991.
11. Holzschuch, N., Sillion, F., Drettakis, G.: An efficient progressive refinement strategy for hierarchical radiosity. In Sakas, G. et al. (eds.): *Photorealistic Rendering Techniques*, pages 357–372, Springer-Verlag, 1995. Proc. 5th Eurographics Workshop on Rendering (Darmstadt, 1994).
12. Kalman, R.: A new approach to linear filtering and prediction problems. *J. Basic Eng., Series 82D*, pages 35–45, 1960.
13. Müller, S., Schöffel, F.: Fast radiosity repropagation for interactive virtual environments using a shadow-form-factor-list. In Sakas, G. et al. (eds.): *Photorealistic Rendering Techniques*, pages 339–356, Springer-Verlag, 1995. Proc. 5th Eurographics Workshop on Rendering (Darmstadt, 1994).
14. Shaw, E.: Hierarchical radiosity for dynamic environments. *Computer Graphics Forum*, 16(2)107–118, 1997.
15. Sillion, F.: A unified hierarchical algorithm for global illumination with scattering volumes and object clusters. *IEEE Trans. on Vis. and Comp. Graphics*, 1(3):240–254, Sep. 1995.
16. Smits, B., Arvo, J., Salesin, D.: An importance-driven radiosity algorithm. *Computer Graphics (Proc. SIGGRAPH '92)*, 26(4):273–282, July 1992.
17. Smits, B., Arvo, J., Greenberg, D.: A clustering algorithm for radiosity in complex environments. *Computer Graphics (Proc. SIGGRAPH '94)*, 28(2):435–442, July 1994.
18. Stamminger, M., Schirmacher, H., Slusallek, Ph., Seidel, H.-P.: Getting rid of links in hierarchical radiosity. In Ferreira, N., Göbel, M. (eds.): *Computer Graphics Forum (Proc. Eurographics '98)*, 17(3):C165–C174, 1998.

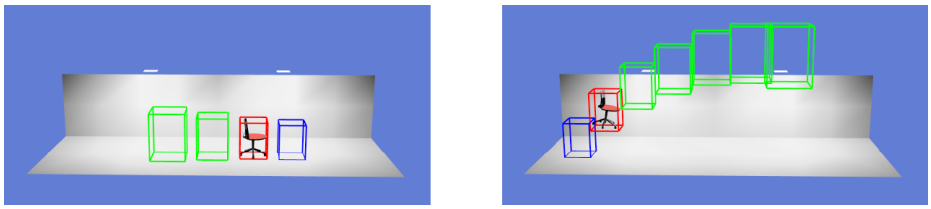

**Fig. 4.** Examples of predicted bounding volumes: The red wireframe box indicates the current position of the dynamic object, the blue box outlines the previous one. Predicted volumes are drawn in green. In the left example, two future positions have been calculated, while in the right image, five predicted positions are shown, taking into account object translation and rotation.

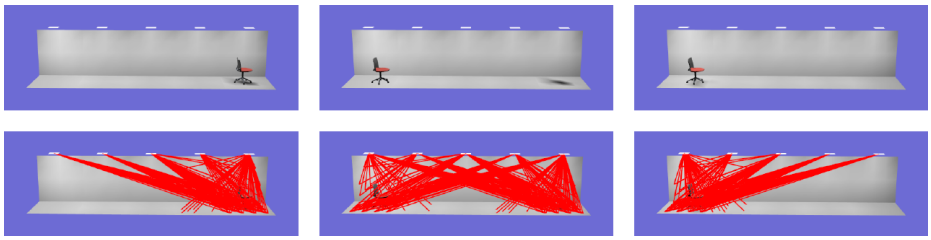

**Fig. 5.** Dynamic shaft management: In the bottom row, links are displayed for those shafts that are stored (only links for the light sources are shown); the same scene without links is shown above. The chair is being moved from the right to the left. Left: Shafts at the old position of the chair. Middle: Shafts for the new position are added. Right: After line-space update, outdated shafts are removed by the garbage collector.

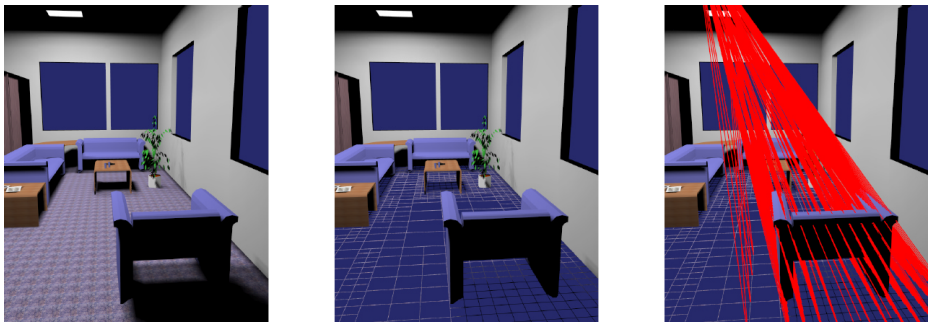

**Fig. 6.** A more complex test environment. Only links from the light source to the floor that are affected by moving the seat are shown. Note that due to balancing, mesh resolution on the floor appears finer than link resolution.