Controlling Memory Consumption of Hierarchical Radiosity with Clustering

Xavier Granier, George Drettakis

Controlling Memory Consumption of Hierarchical Radiosity with Clustering

Xavier Granier George Drettakis
iMAGIS-GRAVIR/IMAG-INRIA, BP 53, F-38041 Grenoble, France.
Email:{Xavier.Granier|George.Drettakis}@imag.fr
iMAGIS is a joint project of CNRS/INRIA/UJF/INPG

Abstract

Memory consumption is a major limitation of current hierarchical radiosity algorithms, including those using clustering. To overcome this drawback we present a new algorithm which reduces the storage required for both the hierarchy of subdivided elements and the links representing light transfers. Our algorithm is based on a link hierarchy, combined with a progressive shooting algorithm. Links are thus stored only when they might transfer energy at subsequent iterations. The push-pull and refine/gather steps of hierarchical radiosity are then combined, allowing the simplification of subtrees of the element hierarchy during refinement. Subdivided polygons replaced by textures and groups of input objects contained in clusters may be deleted. A memory control strategy is then used, forcing links to be established higher in the link hierarchy, limiting the overall memory used. Results of our implementation show significant reduction in memory required for a simulation, without much loss of accuracy or visual quality.

Key words: global illumination, hierarchical radiosity with clustering, memory consumption.

1 Introduction

Global illumination algorithms have made great progress in recent years. With the use of hierarchical radiosity (HR) [8] and clustering [14, 12], global illumination can be simulated for large models, resulting in solutions which are appropriate for real-time walkthroughs. These solutions are typically used for interior design, television and entertainment (virtual sets etc.), and "digital mock-ups". The widespread use of radiosity solutions results in the need to simulate large environments; scenes of millions of initial (input) polygons are now common, and present a challenge to existing radiosity algorithms.

Despite the impressive advances in computation time and control of the simulation precision, hierarchical radiosity methods still require large amounts of memory. Hierarchical radiosity with clustering requires the use of memory-intensive data-structures: *hierarchical elements* which contain numerous fields and the *links* which represent the light transfers at different levels of precision. An initial polygonal model which fits in main memory before the simulation may require several times more memory for the global illumination solution, making such a solution infeasible.

Despite important efforts to reduce memory used by global illumination solutions [18, 5, 17], no overall framework has been proposed which allows the control of memory usage for both the element hierarchy and the links, and which maintains the advantage of the representation of global light transfer.

The new approach we present here first develops a novel framework, based on the line-space, or link, hierarchy [4], and a progressive shooting approach. By modifying the link refine, light gather and push-pull steps of hierarchical radiosity, we reduce both the memory used by the links, and the memory of the *element hierarchy* itself. Subdivision on surfaces is replaced by textures, and initial polygons contained in clusters can be removed in a view-dependent manner. A memory control mechanism is applied to the link hierarchy during the modified refine/gather/push-pull: based on user defined memory limits, links are established at higher levels in the link hierarchy, reducing the storage due to links, and permitting the removal of subdivided elements on polygons.

2 Previous Work

The treatment of complex scenes has been a challenge for global illumination since its outset. Progressive refinement radiosity algorithms [3] have produced simulations of complex environments [1]. The main drawback of progressive refinement is the lack of control of global error, since it is impossible to determine whether the global illumination has been properly accounted for.

Hierarchical radiosity [8] is an efficient algorithm which performs light transfers at appropriate levels of accuracy, by subdividing the environment into hierarchical elements and establishing light-transfer *links* between the elements at the appropriate level. The n^2 "initial link-

ing" step (n is the number of polygons in the scene) renders the approach impractical for large scenes. For certain environments (especially building interiors), partitioning schemes based on visibility can allow the treatment of very large databases [18]. The advent of clustering [14, 12, 7] made the treatment of large databases possible in the general case. In particular the largest model treated by a radiosity algorithm to date (to the authors knowledge) is in the order of 258,000 input polygons using hierarchical radiosity with clustering, coupled with a parallel and distributed system [5]. A completely different approach involves the use of stochastic methods. One recent example is [2], which also includes a thorough bibliography. Due to the fundamental differences compared to finite-element approaches, we do not consider this direction further.

The storage of links for hierarchical radiosity is required since at each iteration light is transported across *all* links. Typically, irradiance on all elements is set to zero, and a gather step follows using the radiosity on each element. Irradiance is thus gathered at all levels of the hierarchy from *all* links, and then pushed to the leaves. Radiosity is then pulled up the hierarchy. Willmott and Heckbert [19] observed that wavelet radiosity (which is hierarchical radiosity possibly with higher-order basis functions) uses a prohibitively large amount of memory, especially for the storage of links.

As a response to this observation, Stamminger et al. [17] developed an approach to "get rid of links". They use a "shooting" approach, which stores an additional "unshot radiosity" variable at each hierarchy element, in the spirit of progressive refinement. The storage of all links is thus no longer required once unshot radiosity is transported across it. A fixed-size cache of links is created, and links are inserted in a sorted manner into the cache based on the energy they transport. If a link is not in the cache, it is recomputed, possibly leading to expensive recursive refinement. In addition, the global representation of all light exchanges is lost. Complex secondary interactions may thus result in significant additional refinement operations.

3 Motivation and Overview

By running "standard" hierarchical radiosity, we have observed that the memory used by the element hierarchy is often on a par or more than that required by the link data structures. This is particularly true for cases in which the link refinement ε threshold value is high, and thus many links remain at the level of clusters. In general, the storage of links *vs.* hierarchy elements is heavily dependent on scene type and the values of the various simulation parameters.

As a consequence, we present a new formulation allowing the reduction of memory for both links and the element hierarchy. The reduction of memory used by links is achieved while maintaining a coherent representation of overall light transfer (in Section 4). This representation, achieved through the line-space hierarchy, enables the introduction of an algorithm which allows us to replace entire subtrees of the element hierarchy during subdivision (Section 5). This allows the replacement of subdivided elements by textures and of entire groups of clustered input elements by a simpler representation. The memory control mechanism is then described (Section 6), which enables our approach to significantly reduce memory used by links and subdivided elements by forcing links to be established high in the link hierarchy. Implementation issues and results are presented in Section 7, and we conclude.

4 Controlling Memory used for Links

As mentioned above (Section 2), links in hierarchical radiosity need to be stored since they are used at each iteration for the gather step. If we use "unshot radiosity", there is no longer a need for all these links. Intuitively, links from the sources, which are often numerous, do not need to be stored. Once the energy has been transferred from the light sources to the receiver surfaces, their utility ends. For subsequent links, we have two choices: we either can "predict" when links would be useful in the future, and thus not store them, or delete links in the subsequent iteration when we determine that they no longer transport energy. We choose the latter approach, since it is unclear how to achieve reliable prediction of link utility.

To be able to remove unnecessary links, i.e. links from sources and links deleted in subsequent iterations, we will use the line-space hierarchy, which stores the history of link creation. This has the advantage that a full description of light exchanges in the scene always exists, albeit at lower accuracy. This is an important difference between our approach and that of [17], and can be particularly useful in the context of dynamic environments [4] or when using importance [15].

4.1 A "Shooting" Algorithm

To achieve a hierarchical radiosity algorithm in which we can avoid the storage of links, we store an additional "unshot" radiosity field, in the spirit of the progressive refinement algorithm [3]. To initialise the system, we set the radiosity and the unshot radiosity equal to the emittance of each leaf element (i.e. not the clusters). Thus initially only sources have radiosity and unshot radiosity. Unshot radiosities are then pulled up the hierarchy. Finally, radiosities are set to the value of unshot radiosity at

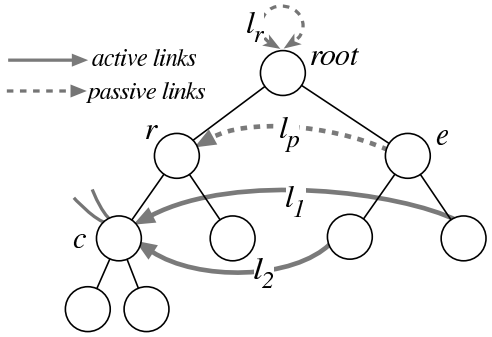


Figure 1: Line-space hierarchy basics

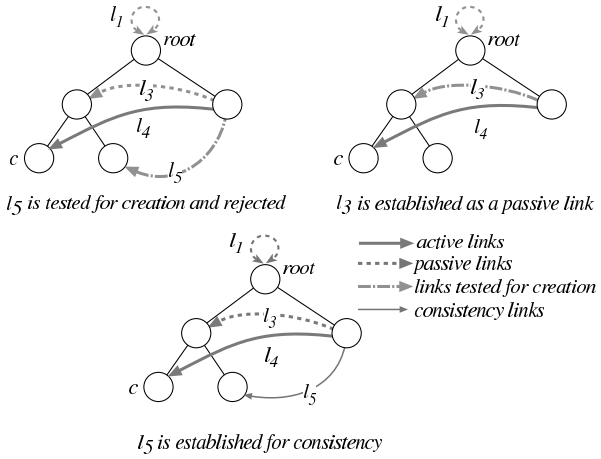


Figure 2: Creating links for consistency at recursion return. l_4 and l_5 are children of the passive link l_3 in the link hierarchy sense.

every level.

At each iteration, we gather unshot radiosity into irradiance, and push irradiance down the hierarchy, setting unshot radiosity to zero. Unshot radiosity is then reflected at the leaves, and added into the radiosity values. Unshot radiosities are averaged while pulling, and added to the radiosity values at every level. This algorithm and the error analysis are similar to those developed by Stamminger et al. [17], showing that this algorithm is equivalent to that of "standard" hierarchical radiosity.

4.2 The line-space hierarchy

The line-space hierarchy as defined in [4], is a hierarchy in link space. We use the terms line-space hierarchy and link hierarchy interchangeably, since we are not interested in the shafts attached to links as in [4].

The link hierarchy maintains the history of link subdivision, in the form of *passive* links. For example, in Figure 1 link l_p, between hierarchy elements r and e is subdivided. In a traditional hierarchical radiosity context, this link is deleted. When maintaining a link hierarchy, we store the link l_p, and the links l_1 and l_2 are considered "children links" of l_p. Links transferring energy will be called "active links". In our example l_1 and l_2 are active links. The passive link l_p no longer corresponds to a real light transfer, but still maintains its form-factor (which required an expensive visibility query to obtain its value), so that if it is re-established in the future as active, it incurs no expense.

4.3 Refinement with Link Removal

Recall that we want to maintain the representation of all energy exchanges and allow the deletion of unnecessary links. To achieve this, we modify the refine and gather steps of traditional hierarchical radiosity. As in previous work [4], we combine refine and gather steps.

In our algorithm, each potential link is first tested against the *refinement criterion*. If we use the ΔBFA criterion, we decide whether the link has enough (unshot) energy to perform the transfer at this level. A new criterion is added which is the *creation criterion*, deciding whether this link should be created, i.e., stored. Currently, the creation criterion simply does not create links from lights sources. More involved criteria are possible. Using the memory control mechanism discussed later, many other links are also not created, with the light transfer occuring on-the-fly during the recursion.

In addition, we remove links which no longer transfer energy. During refine/gather, instead of traversing the element hierarchy itself, we traverse the line-space hierarchy. While descending this hierarchy, we examine the current link. First we test the link against the refinement criterion. If it does not require further refinement, we gather irradiance across this link. We then test the link against the creation criterion. If it satisfies this test, we establish the link as an *active* link.

If refinement is required, we descend to the children of the link (subdividing when necessary), until the refinement condition test allows us to stop. The important step of our approach is performed when returning from this recursion. If no child link has been created, we test whether the current link L verifies the creation criterion. If it does, L is established as an active link; all links below L, created in previous iterations, are now deleted.

If children links exist, we establish L as a passive link. To maintain consistency, we traverse the immediate children in the line-space hierarchy to ensure that a consistent representation is formed, and create additional links if required. This may be necessary due to link removal. See Figure 2 for an illustration of this case, while the entire algorithm is summarised in Figure 3.

```
RefineAndGather(Link L)
  // Descent
  if L satisfies the refinement criterion then
    // end recursion
    add the irradiance transported to the receiver
    if L verifies creation criterion then
      L is stored as an active link
  else
    for all children l of L do
      RefineAndGather(l)
      // return of the recursion
      calculate Form-Factor of L from its children
      if there is no child link of L then
        if L verifies creation criterion then
          L is stored as an active link
          delete all previous children links
        else
          L is established as a passive link
          children links are established
            as active links for consistency
```

Figure 3: The RefineAndGather Algorithm

Recursive form-factor calculation

An interesting advantage of the **RefineAndGather** algorithm is that we can recursively calculate form-factors, and in particular form-factors of clusters.

We can consider two different cases, depending on whether the receiver r or the emitter e is subdivided:

1. The emitter is subdivided: $F_{re} = \sum_{j \in child(e)} F_{rj}$
2. The receiver is subdivided: $F_{re} = \frac{1}{A_r} \sum_{j \in child(r)} A_j F_{je}$,

where child-parent relationships of form-factors are those of corresponding links in the line-space hierarchy sense.

For clusters which do not contain participating media, the following formulation can be applied:

$$F_{rr} = \frac{1}{A_r} \sum_{i \in child(r)} \sum_{j \in child(r)} A_i F_{ij}$$

In this case, the self-link corresponding to F_{rr} (see also [12]), will be quite precise, and in certain cases will actually be exact (e.g. polygons which are "backfacing" each other as in the case of a sphere for example). As seen in the algorithm of Figure 3 this calculation can be integrated simply into the **RefineAndGather** algorithm, and thus may significantly improve the precision of computation, especially at the level of clusters.

5 Controlling Memory of Element Hierarchies

As mentioned in the introduction, our goal is to reduce memory for both links and the *element hierarchy*. In addition, since we have removed a large number of links, the actual subdivision of the hierarchy is often no longer useful for the light simulation itself. It remains nonetheless necessary for the display of the lighting simulation result; however all the information related to the hierarchical representation for the simulation can be discarded and a much cheaper representation can be adopted, which is appropriate for display.

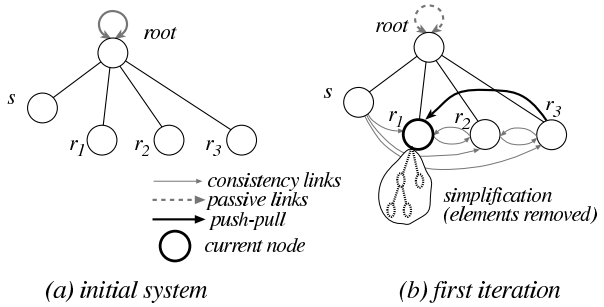


Figure 4: Self-link subdivision: the root self-link is subdivided. Note that some links are not drawn for clarity.

At an abstract level, we require a mechanism allowing the removal of parts of the hierarchy *during* refinement. The alternative, which would involve complete refinement and subdivision followed by the removal of the hierarchy, is inappropriate. Such an approach suffers from the fact that the subdivided element hierarchy may consume all available memory. A new refine-gather-push/pull loop is required, integrating the push-pull step with the **RefineAndGather** approach described previously.

Once the abstract mechanism is in place, we can replace the hierarchy subtree marked as "disposable" with an appropriate representation. In our case, we have chosen to replace the subdivision of surfaces by textures, and, as a first approximation, simplify clusters by removing their contained children.

5.1 Hierarchy Simplification/Refinement Algorithm

We perform an integrated refine-gather-push/pull step with a traversal of link space. We thus start at the root of the link hierarchy, which is the root self-link, and proceed with link refinement.

We need to ensure two basic conditions: (a) that we perform a push-pull operation only *once* at each hierarchy node, and (b) that when we perform a push-pull operation, all links arriving at this node must have been treated.

To do this we identify three different cases which we treat separately: (i) self-links, (ii) receiver subdivision and (iii) emitter subdivision. We use an example to illustrate how each case is treated.

Consider Figure 4(a), showing a scene consisting of sender s and three receivers r_1-r_3. At the outset, only the root self-link exists. This link is split, and all pairs of objects are linked for consistency (see discussion above in

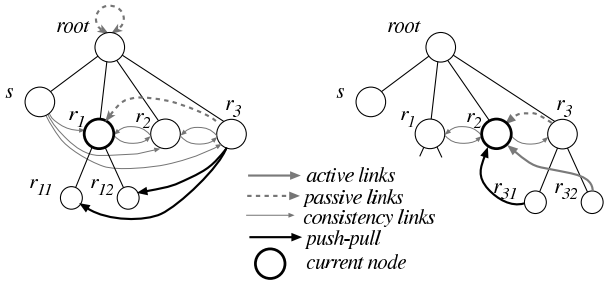


Figure 5: (a) The split receiver case and (b) the split sender case.

Section 4.3). We visit the children links in order; when r_1 is being considered as a receiver, we examine all links arriving at r_1. A **RefineAndGather** operation will be performed on the link $s \rightarrow r_1$, resulting in the creation of subdivided nodes to represent the illumination.

All other links arriving at r_1 will be treated in order. Assume that the last link is the link $r_3 \rightarrow r_1$, as in Figure 4(b). At this point a push-pull operation is performed on node r_1 and its children. Since no links are created, the subdivision of r_1 is removed and replaced by a suitable representation.

At the second iteration, all receivers r_i have positive unshot radiosity and thus will also act as emitters. Consider the case shown in Figure 5(a). The current node is r_1, and we are considering the last link arriving at r_1 which transfers light from r_3. The refinement criterion decides that r_1 should be subdivided. In this case, when we treat the last link $r_3 \rightarrow r_1$ arriving at the receiver r_1, we perform a push-pull operation on all the children of the current receiver r_1.

The remaining case is that of a split emitter. In Figure 5(b) the current receiver node is r_2 and the emitter is r_3. The refinement criterion required r_3 to be subdivided. Assume that the link $r_3 \rightarrow r_2$ is the last link arriving at r_2 (Fig. 5(b)). The push-pull operation will be performed on r_2 when we treat the link $r_{31} \rightarrow r_2$ (i.e., the last link arriving at r_2). Thus all links arriving at r_2 have been treated.

This process is summarised in Figure 6. Note that in practice a temporary variable is used during the return of the pull procedure, so that the current radiosity values are not changed.

This abstract framework allows the replacement of *any* subtree of the hierarchy by an appropriate representation during subdivision. Since the hierarchy of the scene is *complete*, i.e. surfaces and clusters are all contained in a single root cluster, in principle we can replace as much of the hierarchy as we see fit, assuming that this replacement is appropriate.

```
RefineGatherAndPushPull(
      Link L, Element R, Element S, IrradianceDown I)
   if L must be refined then
      choose element to split
      if R is split then
         return RefineGatherAndPushPullRcv(L,R,S,I)
      else if S is split then
         return RefineGatherAndPushPullEmit(L,R,S,I)
   else
      return R → PushPull(I);

RefineGatherAndPushPullRcv(
      Link L, Element R, Element S,IrradianceDown I)
   Radiosity of R is set to Zero();
   for r child of R
      l = potential link from S to r;
      R → Radiosity += r → AreaFactor ∗
         RefineGatherAndPushPull(l,r,S,I+R→Irradiance);
   R → Radiosity /= R → AreaFactor;
   if R has links but its children do not
      replace subdivision of R
   return R → Radiosity;

RefineGatherAndPushPullEmit(
      Link L, Element R, Element S, IrradianceDown I)
   while c not the last child of S
      l = potential link from c to R;
      RefineAndGather(r,R,c);
   l = potential link from c to R;
   return RefineGatherAndPushPull(l,R,c,I);
```

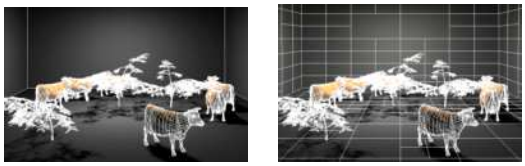
Figure 6: Refine Gather and Push-Pull algorithm

In practice, we can easily replace the hierarchy which is due to subdivision on a polygon surface, since the underlying geometry remains the same. We will show next how to do this by replacing subdivided elements by textures (recall the case of r_1 in Figure 6(a)). Related ideas have been used in different contexts in [16, 9, 10].

Replacing initial scene geometry (i.e., simplifying clusters and their contents) is much more involved. Many potential solutions can be found, all of which require the use of some geometry simplification technique (i.e. image-based [13], volumetric [11] or mult-resolution representations [6] etc.). Investigating these alternatives is beyond the scope of this paper. Instead, we will present a simple first solution, by representing the cluster as a point for ligh-transfer, and as a shaded bounded box for display.

5.2 Replacing Polygon Subdivision by Textures

The data structure of a hierarchical element corresponding to an input polygon contains information about its children, radiosity, unshot radiosity and irradiance as well as link information. We create such a node which has a



(a) (b)

Figure 7: (a) 1st iteration: the illumination detail of the floor and wall polygons is represented by a texture (i.e. no hierarchical subdivision exists on these polygons). (b) 2nd iteration, some re-splitting occurrs. The elements share the same texture.

special texture attached to it. This texture corresponds to the values of the subdivided initial polygon.

The texture is a two-dimensional array of the floating point values of radiosity. In addition a second array which represents unshot radiosity is created in the same manner. These two arrays are stored with the input polygon. When displaying the polygon, the floating point array is converted to a texture appropriate for display on the graphics hardware. In subsequent iterations, the polygon may be resubdivided (see Figure 7), but the texture array is not replicated. The sub-elements are simply assigned correct texture (sub)coordinates. Note that we could gain memory by using a more compact representation for these textures.

Irradiance need not be explicitly stored as a texture, since it is only needed during push-pull, and thus is a temporary variable created during the push phase and freed at the end of the pull step for a given subtree of the hierarchy.

The gather and push-pull operations are applied in the normal manner to each entry of the radiosity and unshot radiosity arrays, thus maintaining the same quality solution, without the overhead of the hierarchical data structures. In our system, ignoring the memory required for the original geometry, a quadrilateral hierarchical element costs around 200 bytes, including children pointers, parametric coordinates, radiosity and irradiance fields, list of links pointer, and vertex geometry and color, used for display of intermediate results. Note that our implementation is in C++, incurring additional storage overhead.

5.3 Replacing Cluster Contents

As a first approach, we have chosen a simple replacement strategy for clusters. For the lighting simulation [12], clusters are considered to be a single point sample of radiosity (typically at the center of the cluster). If the **RefineGatherAndPushPull** algorithm decides that a subtree based at a cluster is suitable for simplification, we remove the children surfaces from the cluster. Clusters are simplified as for polygons and if their projection covers less than a pre-defined number of pixels.

The simplified cluster stores a single value for each of radiosity, irradiance and unshot radiosity. For display, we use the bounding box shaded with the colour value of the clusters radiosity. Since the simplified node is a leaf of the element hierarchy, it stores a reflectance value, and irradiance pushed down to it is reflected at the level of the cluster. Finally, we treat the simplified cluster bounding box as a volumetric element, with a (scalar) extinction coefficient in the manner of [12].

In contrast to polygon subdivision replacement, this approach is view-dependent and does not produce exactly the same solution as the “standard” approach. It is presented here simply to demonstrate that the element replacement can be performed at *any* level of the hierarchy. More involved simplification could be used to achieve better results (see Section 8).

6 Memory Control Mechanism

The algorithm presented above will reduce memory used by links and the hierarchy. A certain number of links are however still created (in particular all links for consistency and links transporting indirect light), resulting in a peak in memory usage, typically at the second iteration (recall that no links from sources are stored). It is thus important to be able to control the total amount of memory required as much as possible, in the same spirit as the cache strategy of [17].

Instead of an explicit cache, we simply use the link hierarchy to directly limit the memory used by links. This is done by modifying the link creation criterion (see Section 4.3): before creating a link we test to see whether the link is above a certain level in the link hierarchy. The cut-off level is estimated based on the memory the user wishes to use, and an estimation of the average number of links in the link hierarchy below this level. This has the consequence of moving active links higher up in the hierarchy. Recall that the light transfers of links which are not stored are performed on-the-fly during the traversal of the link hierarchy.

Nonetheless, a complete representation of all light exchanges is maintained, albeit at a coarser level. Clearly, a more involved method could be used to tightly control the actual memory used by the links and appropriately modify the cut-off level of link creation.

An important consequence of moving links up the hierarchy is that large element hierarchy subtrees become candidates for removal. If the texture replacement only is used, very few surfaces are actually subdivided. Removing the clusters will further remove hierarchy subtrees, but in a view-dependent manner.

Figure 8: Image generated by the original HRC algorithm for the Medium Hall scene.

Figure 9: Image generated by the replacement by textures algorithm for the Complex Hall scene.

7 Implementation and Results

We have implemented the algorithm described previously as part of our hierarchical radiosity system. We present results of our implementation, and compare them to a "standard" BFA hierarchical radiosity with clustering (HRC) approach.

We have tested our approach of three scenes. This is a sequence of three "halls" containing 85,000 to 676,000 initial polygons each. These are called "Simple", "Medium" and "Complex Hall" respectively and abbreviated *SH*, *MH* and *CH*. Examples of the hall scenes are shown in Figures (8, 9). We first show the statistics for the reference "standard" HRC solutions in Table 1.

Test Scene	*SH*	*MH*	*CH*
input polys	65K	169K	676K
mem_{init}	39MB	77MB	309MB
mem_{clust}	7MB	15MB	59MB
links	3.5M	7.5M	4.4M
$poly_{sub}$	137K	203K	2.5M
mem_{hier}	23MB	40MB	500MB
mem_{link}	94MB	202MB	118MB
Time	1h30mn	4h07mn	4h15mn
mem_{tot}	156MB	320MB	927MB

Table 1: Statistics for the reference "standard" radiosity solution. mem_{init} is the total initial memory *before* the solution, mem_{clust} is the part used by clusters, *links* is the number of links at the end of the solution, and $poly_{sub}$ is the number of hierarchical polygonal elements. mem_{hier} is the memory used by the hierarchy (excluding initial polygons), and mem_{link} that used by the links. *Time* is the total computation time; mem_{tot} is the total memory used, including the initial memory.

In Table 2 we show the results of our algorithm with texture replacement of subdivision only (no cluster reduction is performed), where the memory control target has been set to about 30% of initial memory mem_{init}. Clearly, our algorithm achieves significant overall savings in memory, the overall gains varying from 73 to 88% (excluding initial memory). It is important to note that

Test Scene	*SH*	*MH*	*CH*
mem_{link}	1.5MB	1.8MB	11.3MB
$mem_{hier+tex}$	13.3MB	29MB	155MB
links	55K	67K	426K
$poly_{sub}$	22K	3K	7.7K
$gain_{link}$	98%	99%	90%
$gain_{hier}$	42%	27%	70%
$gain_{mem}$	87%	88%	73%
Time	1h07mn	2h51mn	5h39mn
mem_{tot}	58.3MB	112MB	501MB

Table 2: Results of our algorithm. Notation is as in Table 1, except $mem_{hier+tex}$ which is the memory used by the hierarchy and the textures (excluding initial polygons), and $gain_{link}$, $gain_{hier}$, $gain_{mem}$ which are the percent memory gains for link, hierarchy and overall respectively (excluding initial memory).

scenes with very large memory consumption, can now be treated with much less memory (e.g., 501MB instead of 927MB; initial memory is 309MB). The computation time is comparable or even less than that of the standard solution. This is mainly due to the improved form-factor calculation, which reduces the values of many cluster self-link form-factors resulting in a slightly lower number of gather operations overall. The complex *CH* scene has fewer light sources in each room, resulting in a lower number of overall links, and a faster computation time.

Finally we present an example of the cluster reduction algorithm (Figure 10) showing very few artifacts. Here the memory gain was 80% mainly due to the links, since most of the objects are not subdivided in this scene.

(a) (b)

Figure 10: Scene containing 43,000 polygons. (a) the original image and (b) the image generated with cluster reduction.

8 Conclusions and Future Work

We have presented a novel algorithm for controlling the memory consumption for hierarchical radiosity with clustering. We show how we can reduce the memory used by both the element hierarchy and the light-transport links. To do this, we introduce a new algorithm for the refine-gather-push/pull loop, which removes unnecessary links and permits the removal of a subtree of the element hierarchy during subdivision. The algorithm is based on the link or line-space [4] hierarchy, thus preserving the overall representation of global light transfers. Using this representation, our memory control algorithm can calculate global illumination solutions on a limited memory budget, by moving light transfers higher in the link hierarchy (in effect representing them in a more imprecise manner). The results of our implementation show that we can achieve significant savings in real memory consumption with little loss of visual quality or precision in the simulation of light.

In future work, we need to investigate more sophisticated memory control mechanisms, taking the memory consumption of the hierarchy directly into account. This would require storing parts of the hierarchy (clusters and surfaces) to disk during the computation, when the algorithm requires removal of the corresponding sub-tree. If, in addition, we can perform the first iteration of light transfer while reading in the scene, we could apply the simplification techniques described in this paper "on-the-fly" during the loading of the scene file. As a result, it should be possible to simulate environments of arbitrary size on any computer. This entire approach will probably require reorganizing the the order of lighting computations and sophisticated disk handling routines.

We believe that the ideas in this paper can be used with more appropriate representations for simplified clusters. These could take the form of image-based or volumetric primitives, or multi-resolution geometric simplifications. Finally, it will also be interesting to develop strategies to predict the utility of a link in future iterations reducing storage requirements of links.

Acknowledgements

Thanks to Frédo Durand for proof-reading and careful insights. This research was partly funded by Commission of the European Communities ESPRIT Reactive LTR project ARCADE (#24944).

9 References

[1] 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. In *Proc of. SIGGRAPH '91*, July 1991.

[2] P. Bekaert, L. Neumann, A. Neumann, M. Sbert, and Y. Willems. Hierarchical monte carlo radiosity. In *9th EG Workshop on Rendering*, June 1998.

[3] M. F. Cohen, S. E. Chen, J. R. Wallace, and D. P. Greenberg. A progressive refinement approach to fast radiosity image generation. In *Proc. of SIGGRAPH '88*, pages 75–84, August 1988.

[4] G. Drettakis and F. X. Sillion. Interactive update of global illumination using a line-space hierarchy. In *Proc. of SIGGRAPH 97*, pages 57–64, August 1997.

[5] T. A. Funkhouser. Coarse-grained parallelism for hierarchical radiosity using group iterative methods. In *Proc. of SIGGRAPH 96*, pages 343–352, August 1996.

[6] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proc. of SIGGRAPH 97*, pages 209–216, August 1997.

[7] S. Gibson and R. J. Hubbold. Efficient hierarchical refinement and clustering for radiosity in complex environments. *Comp. Graphics Forum*, 15(5):297–310, 1996.

[8] P. Hanrahan, D. Salzman, and L. Aupperle. A rapid hierarchical radiosity algorithm. In *Proc. of SIGGRAPH '91*, pages 197–206, July 1991.

[9] P. S. Heckbert. Adaptive radiosity textures for bidirectional ray tracing. In *Proc. of SIGGRAPH '90*, pages 145–154, August 1990.

[10] K. Myszkowski and T. L. Kunii. Texture Mapping as an Alternative for Meshing During Walkthrough Animation. In *5th EG Workshop on Rendering*, pages 375–388, 1994.

[11] F. Neyret. Modeling, Animating, and Rendering Complex Scenes Using Volumetric Textures. *IEEE Trans. on Visualization and Comp. Graphics*, 4(1):55–70, January 1998.

[12] F. X. Sillion. A unified hierarchical algorithm for global illumination with scattering volumes and object clusters. *IEEE Transactions on Visualization and Comp. Graphics*, 1(3):240–254, September 1995.

[13] F. X. Sillion, G. Drettakis, and B. Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. *Comp. Graphics Forum*, 16(3), September 1997.

[14] B. Smits, J. Arvo, and D. Greenberg. A clustering algorithm for radiosity in complex environments. In *Proc. of SIGGRAPH '94*, pages 435–442, July 1994.

[15] B. E. Smits, J. R. Arvo, and D. H. Salesin. An importance-driven radiosity algorithm. In *Proc. of SIGGRAPH '92*, pages 273–282, July 1992.

[16] C. Soler and F. X. Sillion. Automatic calculation of soft shadow textures for fast, high-quality radiosity. In *9th EG Workshop on Rendering*, June 1998.

[17] M. Stamminger, H. Schirmacher, P. Slusallek, and H-P. Seidel. Getting rid of links in hierarchical radiosity. *Comp. Graphics Forum (EUROGRAPHICS '98)*, 17(3), September 1998.

[18] S. Teller, C. Fowler, T. Funkhouser, and P. Hanrahan. Partitioning and ordering large radiosity computations. In *Proc. of SIGGRAPH '94*, pages 443–450, July 1994.

[19] A. J. Willmott and P. S. Heckbert. An empirical comparison of progressive and wavelet radiosity. In *8th EG Workshop on Rendering*, pages 175–186, June 1997.