Parallel Hierarchical Radiosity: the PIT Approach

F.Baiardi, P.Mori and L.Ricci

Dipartimento di Informatica, Università di Pisa
Via F.Buonarroti, 56125 - Pisa (Italy)
{ricci,baiardi,mori}@di.unipi.it

Abstract. Radiosity is a method to compute the global illumination of a scene characterized by high computational requirements. To reduce this complexity, hierarchical radiosity decomposes the scene into a hierarchy of patches and computes the light exchange at different levels, according to the distance between patches and/or to the amount of light emitted by the patches. A distributed memory implementation of this hierarchical method has been developed through *PIT*, a problem independent library. *PIT* functions are defined in term of a distributed version of the tree representing the hierarchical decomposition. Experimental results confirm the effectiveness of this approach.

1 Introduction

Radiosity [5] defines the global illumination of a scene by decomposing objects into a set of patches and by computing the forms factor of each pair of patches, i.e. the fraction of the light exchanged between the patches. To reduce the overall complexity a hierarchical approach exploits hierarchical patches and it computes the light exchange at distinct levels according to the distance between the patches and/or to the amount of light they emit. One of the problems posed by a parallel implementation of the method is the *irregular* and *dynamic* distribution of the patches in the scene. Furthermore, the computational loads of the patches may be very different because the light exchange is computed at distinct levels. This paper evaluates a parallel version of a hierarchical radiosity method developed through *PIT* [3,1,2] a problem independent programming library to support the development of irregular hierarchical applications on distributed memory architectures. Its key assumption is that both the sequential and the parallel application can be built around a tree that represents the distribution of elements in the domain. The interactions among the elements are computed by applying proper operators to the tree. In the parallel version, the tree is mapped into the local memories of the processing nodes, pnodes. A new data type, the *Parallel Irregular Tree*, describes the mapping of the tree onto the pnodes, and it defines the operations to access information mapped onto another pnode, to insert new nodes into the tree and to balance the computational load. To focus on the adoption of *PIT* to develop hierarchical radiosity on a distributed memory architecture, we have considered scenes in *flatland*, i.e. a bi-dimensional world. Sect.

2 describes the main functions of *PIT*. The sequential hierarchical approach to radiosity in flatland is revised in Sect. 3. Sect. 4 describes the parallelization of the method by *PIT*. Experimental results are discussed in Sect. 5.

2 The PIT library

Parallel Irregular Tree, *PIT*, is a library addressed to the parallelization of irregular problems on distributed memory architectures. This library adopts a hierarchical method to solve irregular problems, and it is based upon the assumption that both the sequential and the parallel versions of this hierarchical method may be structured in terms of operations on a hierarchical tree, *Htree*. The *Htree* represents both the subdomains resulting from the domain decomposition and the hierarchical relation among them and any element they include. Any hierarchical method, to solve irregular problems, iteratively visits the *Htree* and it updates the properties of its elements. For each *hnode* hn , the neighbors of $element(hn)$, the element paired with hn , are collected to update the properties of $element(hn)$. The operations to update the properties of an element depend upon the nature of the problem to be solved, *target problem*, and are denoted *target problem operators*, or simply operators. A goal of *PIT* is to preserve the code of these operators but, while in a sequential application the *Htree* is recorded in just one memory, in a distributed memory architecture the *Htree* is distributed across the local memories of all the pnodes. The functions of *PIT* support the application of these operators in spite of the mapping into distinct memories. These functions are defined in terms of the *Parallel Hierarchical Tree*, *PITree*, the distributed version of the *Htree*. A *PITree* $P(H)$ is a tuple $\langle h_0, \dots, h_{np-1}, mht \rangle$, where np is the number of pnodes, that describes the mapping of a *Htree* H . Any h_i is the private *Htree* of P_i , the i -th pnode, i.e. the subset of all and only the *hnodes* of H that represent domains or elements mapped into the local memory of P_i . *mht*, instead, is the *mappingHtree* that represents the hierarchical relations among $h_0, \dots, h_{np-1}$. *mht* includes the root R of H and all the *hnodes* on the paths from R to the root of any h_j . Each *hnode* of *mht* corresponds to a *hnode* hp of a private *Htree* h_j and it records j , the identifier of the pnode where hp has been mapped. *mht* is the minimal amount of information to be replicated in all the pnodes to enable the *PIT* functions to determine where any *hnode* of H has been mapped.

In the following, we focus on the functions of *PIT* that have been used to define the parallel version of hierarchical radiosity methods. *PIT* is fully described in [3], together with the algorithm to map the *Htree* onto the pnodes. *PITree_Creation* is the function that partitions the initial set of elements among the pnodes and builds the *PITree*, i.e. both the private and the mapping *Htrees*. The strategy adopted to determine the element partition takes into account the locality property to reduce the communications among the pnodes in the *PITree_Completion* function. It returns to each pnode a reference to the root of its private *Htree*. This reference is used to invoke the *PIT* functions that support the application of the target problem operators. Since *PIT* is independent of the

target problem and, in particular, of the element properties, the user defines both a structure to record these properties and three functions, *include_el*, *decompose_el*, *remove_el*, to handle the elements. These functions are exploited by *PITree_Creation* to properly partition the domain elements.

Each pnode P_i applies the target problem operators to each hnode in its private Htree h_i , in a SPMD style. However, h_i does not include all the elements the operator requires, because some neighbours of a hnode hn of h_i , the remote neighbours of hn , may have been mapped onto distinct pnodes. *PITree_Completion* is the *PIT* function that collects all the elements required by an operator and creates, for each private Htree h_i , the *essential Htree* eh_i . eh_i is the union of h_i with all the remote neighbours of any element in h_i . Any operator can be correctly applied to $eh_i \cap h_i$, as in the sequential version of the application. Hence, in the most general case, this function is invoked before the execution of each target problem operator. The parameters of *PITree_Completion* are the root of the private Htree and a user defined function *stencil* to compute the neighborhood stencil, i.e. the set of neighbors of each element. The *stencil* function depends upon the target problem operator, and distinct operators of the same problems may have distinct neighborhood stencils. *PITree_Completion* implements a fault prevention strategy [1], where each pnode P_k determines, through the neighborhood stencil, which of its elements are required by P_h , $\forall h \neq k$ and sends these elements to P_h without any explicit request from it. This strategy reduces the number of communications with respect to the request and answer one. *PITree_Balance* updates the mapping of the *PITree* onto the pnodes to balance the computational load among the pnodes, and it is invoked anytime the distribution of the elements changes. Hence, in the simplest case, it is invoked after the execution of each operator. Its input parameters are a reference to the roots of the private Htrees and a threshold to prevent the recovery of very low unbalances. It returns a reference to the root of the updated private Htree.

3 Hierarchical Radiosity in Flatland

The hierarchical radiosity method is applied in computer graphic to compute the global illumination of a set of objects in a scene. This method has been introduced to compute the heat transfer between surfaces in [9], and it has been applied in computer graphics in [6] to compute the illumination of unoccluded environments and in [4] to compute the illumination of scenes with occlusions. Hierarchical radiosity [7] discretizes the objects of the scene through small elements, denoted as *patches*, whose brightness is assumed to be constant. The radiosity B_i of a patch p_i depends upon those of all the other patches p_j and it is defined as follows:

$$B_i A_i = E_i A_i + \rho_i \sum_{j=0}^{n-1} B_j A_j F_{j,i}$$

where A_i is the area of p_i , E_i is its emissivity, i.e. the light emitted by p_i , ρ_i is its reflectance, and $F_{j,i}$ is the percentage of light leaving the patch p_j that reaches p_i . $F_{i,j}$ is the *form factor* between p_j and p_i .

In *flatland* [8, 10], the bidimensional world we consider, a scene is a set of polygons. Each polygon is decomposed into a set of segments, each with two sides. Only one side interacts with the other polygons, because the other one is internal to the polygon. Let us consider a pair of segments S_i and S_j . A popular and efficient formula for the form factor $F_{i,j}$ is the *the string rule* [9]. Let us consider the *strings* stretched from the endpoints of S_i to the corresponding endpoints of S_j , i.e. the *uncrossed strings*, and those that links up the opposite endpoints, i.e. the *crossed strings*. $F_{i,j}$ is proportional to the difference between the lengths of the crossed strings and that of the uncrossed ones. If some parts of S_i and S_j are not mutually visible, the strings are “stretched” around the obstruction so that the string rule can consider an arbitrary number of occluding segments.

Hierarchical radiosity recursively partitions each segment into a hierarchy of patches and computes the interactions among patches so that a segment interacts with the other ones at distinct levels. Approximated interactions are computed among low level patches, while accurate interactions are computed among high level ones. An *interaction list* paired with each patch p_i records the patches of the other segments that interacts with p_i . By pairing each element of this list with the list of the related occlusions, we produce the visibility list that includes the information to compute the radiosity of the corresponding patch.

Hierarchical radiosity computes a sequence of approximations starting from a domain that includes all the segments of the polygons in the scene. The algorithm computes the visibility list of each segment and it sets the initial radiosity of each segment to its emissivity. Then, an iterative procedure computes the global radiosity of each segment. Each iteration applies the functions *Gather*, *PushPull* and *RefineLink*. For each segment S_i , *Gather* collects the light emitted by the other segments that reaches S_i , according to the current decomposition. The function is recursively applied to S_i and to any patch produced by the decomposition of S_i and it applies the *string rule* to compute the light contributions received by any patch in the visibility list. For each segment S_i , *PushPull* redistributes among the patches of S_i the light contributions received at all the levels of the hierarchy as computed by *Gather*. *PushPull* includes two steps. The *Push* step pushes down into the hierarchy the radiosity of all the patches of S_i from the root patch that represent the segment S_i until the leaf ones. The radiosity of a patch p_i is updated by the current value of the radiosity of the patch p_i belongs to. The *Pull* step pulls the radiosity of each patch of S_i up in the hierarchy, from the leaf patches until the root patch. The radiosity of p_i is the weighted average of the radiosity of each of the subpatches of p_i . *PushPull* computes the overall radiosity of the scene as well. The method terminates if the difference between the current radiosity of the scene and the one at the previous iteration is lower than a user defined threshold. *RefineLink* further partitions a patch p_i whose interactions have to be computed with a better accuracy, i.e. if its radiosity is larger than a predefined threshold. p_i is partitioned into patches according to its geometric properties. *RefineLink* updates the visibility lists of p_i and that of each patch p_j in its interaction list.

Fig. 1. Domain decomposition and PITree

4 Parallelizing Hierarchical Radiosity: the PIT Approach

While radiosity represents distinct objects of the domain by distinct trees, where the nodes of the tree represent the patches, in our approach, a single hierarchical tree *Htree* represents the whole domain and it includes all the objects. The *hnodes* represent the spaces that include the patches. Fig. 1 shows the *Htree* and the domain it represents.

The scene includes two segments, S_0 and S_1 , partitioned into patches. The white *hnodes* of the *Htree* represent the spaces of the domain, while the black ones represent the elements, i.e. the patches. The nodes corresponding to the patches store the properties of the corresponding patch, i.e. the coordinates of the segment, its orientation with respect to the polygon that includes it, the emissivity, the reflection and the current radiosity.

The initial domain is partitioned until each space includes one element only. For instance, the segment S_1 of Fig. 1 has been partitioned into two subsegments, p_0 and p_1 , at level $l = 1$, because the smallest space that includes it, that is the space that represents the whole domain, also includes the segment S_0 . This agrees with the philosophy of this method that partitions a segment S_i before the computation if another segment S_j belongs to the smallest space including S_i . In this case S_i and S_j are close in the domain, and they will be decomposed anyway by the refinement function to compute their interactions. Fig. 1 shows a simple domain decomposition and the related *PITree*.

Fig. 2 describes the parallel version of the hierarchical radiosity method developed through the *PIT* library by inserting the proper *PIT* functions into the sequential code. The first *PIT* function invoked by the parallel code is *PITree_creation* that, starting from the initial set of elements of the domain, creates the *PITree* and returns to each *p-node* a reference to its *private Htree*. For instance in Fig. 1 the segment S_0 and all its patches have been assigned to the *p-node* P_0 , while the segment S_1 and all its patches have been assigned to P_1 . Before the first iteration, *PITree_completion* gather information to compute the visibility list of each segment. In this case, the neighborhood stencil of a segment

S_i , i.e. *vis_stencil*, includes all the spaces that are visible from the external side of S_i .

The computation of the radiosity of each patch p_i requires those of some other patches of the scene. Since some of the patches that p-node P_a needs to compute the radiosity of p_i may have been mapped onto another p-node P_b $a \neq b$, *PITree_completion* is applied before each operator that requires the remote neighbours of a patch. *Gather* and *PushPull* compute the radiosity of a local patch in terms of that of local and of remote ones, while *RefineLink* requires local values only. However, if a patch and all the subpatches it has been recursively partitioned into are mapped onto the same p-node, also *PushPull* requires local values only. The patches required by *Gather* to compute the contributions to the radiosity of p_i are a function of the visibility list of p_i that includes the references to all the patches that interact with p_i and to all those that occlude these interactions. To collect the remote neighbors *Gather* requires, the fault prevention strategy can be applied. The neighborhood stencil of *PushPull* is very simple because, in the *Push* step, the stencil of p_i includes only the patches which p_i belongs to. In the *Pull* step, instead, the stencil of p_i includes any patch resulting from the decomposition of p_i . To update the radiosity of p_i , *PushPull* requires the radiosity of the neighbor patches that have been already updated by *PushPull*. For this reason, in the parallel version of the method, both steps of *PushPull* are implemented through a breadth first visit of the tree. If an anticipated visit is adopted, the execution of the *Push* step and/or the *Pull* one on the objects whose patches are not mapped onto the same p-node strongly serializes the parallel computation.

```

hierarchical_radiosity_method(element_list *scene) {
    pht_root = PITree_creation(scene, include_el, decompose_el, remove_el);
    PITree_completion(vis_stencil, all_levels);
    Visib_list_determ_H(pht_root);
    while (not end) {
        PITree_completion(interaction_list, all_levels);
        Gather_H(pht_root)
        for level from L_min to L_max
            PITree_completion(push_stencil, level);
            Push(pht_root, level)
        for level from L_max to L_min
            PITree_completion(pull_stencil, level);
            Pull(pht_root, level)
        end = RefineLink_H(pht_root);
        pht_root = PITree_balance(pht_root); }}

```

Fig. 2. Hierarchical Radiosity PIT Parallel Code

RefineLink updates the *Htree* by inserting a new *hnode* for each new patch of the scene. Since the new patches created by a p-node P_i partition the patches

already mapped onto P_i , they are mapped onto P_i as well. This is coherent with the adopted methodology because, if the *PIT* data mapping strategy assigns a space A to a p-node P_i , then it assigns to P_i all the subspaces of A before the other spaces of the domain. To determine the computational load of each element, we observe that most of the computation is devoted to compute the form factors. Hence, $l(p_i)$ the load of a patch p_i can be estimated through the number of interactions executed by p_i and it is proportional to the size of the visibility list of p_i . *RefineLink* updates the load of the p-nodes, because it partitions the existing patches and the new patches generate new interactions. The partitioning of p_i , affects both the visibility list of p_i and that of any patch p_j that interacts with p_i . This decreases $l(p_i)$, because the interaction with p_j has been removed from its visibility list, and it increases $l(p_j)$, because the interaction with p_i has been replaced by those with the subpatches of p_i . Moreover, new elements, i.e. the subpatches of p_i , have been added to the domain. The initial load of these elements depend upon the visibility list they inherit from p_i . For these reasons the *PITree_balance* is invoked after *RefineLink*.

5 Experimental Results

To evaluate both the effectiveness and the performances of the PIT approach, this section presents some experimental results of the PIT parallel version of the hierarchical radiosity method. The first scene we consider in our experiments is very simple, and includes 224 initial segments that compose 48 polygons. This scene is shown in figure 3.

Fig. 3. Test scene: 224 segments

After 5 iterations only of the method, the total number of patches of the scene is about 3500% of the initial one. Figure 4 shows the subdomain assigned to pnode 0 after 5 iterations in an experiment executed on 10 pnodes.

We can notice that the hierarchical decomposition of the domain is irregular, because some segments, or part of it, have been decomposed more deeply than

Fig. 4. Subdomain of pnode 0 after 5 iterations

others. We recall that this decomposition depends upon both the geometrical features of the scene and the exchange of light among the segments.

The first parallel architecture we consider to execute our experiments is a cluster of 10 workstations. Each workstation is a PC with an Intel Pentium II CPU (266 MHz) and 256 Mbyte of local memory and the interconnection network is a switched 100Mbit Fast Ethernet.

Fig. 5. Completion time with alternative load balancing thresholds

The first set of experimental results concerns the importance of the load balancing to achieve a good efficiency. Figure 5 shows the completion time of 30 iterations of the parallel hierarchical radiosity method applied to the scene of figure 3 for different values of the load balance threshold. We can notice that the lowest completion time is achieved by adopting a balance threshold of 20%. In this case, the largest unbalance that is tolerated by the parallel application is 20% of the average load of the pnodes. Moreover, the figure shows that, by adopting a load balance threshold of 100%, the completion time is 116% of the optimal one. This demonstrates that considerable performance improvements can be achieved by updating the data mapping during the computation to re-

cover the load unbalances. Moreover, this confirms the effectiveness of the PIT approach.

The next set of experimental results evaluates the performance of the parallel implementation. Figure 6 shows the efficiency of the hierarchical radiosity method for a variable number of pnodes. The scene adopted for this experiment is that of figure 3, and 30 iterations have been executed.

Fig. 6. Efficiency for the Hierarchical Radiosity: 224 segments

The figure shows that, even on 10 pnodes, the efficiency achieved by our implementation is about 60%, and is larger for a lower number of pnodes. For instance, it is about 80% if we consider 4 pnodes. If we adopt larger scenes, the efficiency of our implementation increases.

The other scene considered in our experiments is shown in figure 7 and it includes 192 polygons defined through 896 segments.

Fig. 7. Test scene: 896 segments

Fig.8 shows the performance of the *PIT* hierarchical radiosity method, on two architectures. The left part of the figure shows the experiments executed on

Fig. 8. Efficiency for Hierarchical Radiosity: 896 segments

the architecture previously described. In this case, the efficiency is larger than 80%, on 10 p-nodes. One of the main reasons of this efficiency is the high locality of the scene. In fact, the large number of polygons of the scene results in a large number of occlusions among segments, so that each segment interacts mainly with those that are close in the domain. The mapping strategy of *PIT* exploits at best this locality to minimize the communication overhead.

We have also considered an IBM Linux Cluster including 64 nodes, each with 2 Intel Pentium III (1.133 GHz) and 1Gbyte of local memory. The interconnection network is a Miricom LAN "C" Version. The right side of figure 8 shows the performance on this architecture. These results confirm the previous ones because the efficiency is larger than 70% even on 32 pnodes.

References

1. F. Baiardi, S. Chiti, P. Mori, and L. Ricci. Parallelization of irregular problems based on hierarchical domain representation. *Proceedings of HPCN 2000: Lecture Notes in Computer Science*, 1823:71–80, May 2000.
2. F. Baiardi, S. Chiti, P. Mori, and L. Ricci. Integrating load balancing and locality in the parallelization of irregular problems. *Future Generation Computer Systems: Elsevier Science*, 17:969–975, June 2001.
3. F. Baiardi, P. Mori, and L. Ricci. Pit: A library for the parallelization of irregular problems. *Proceedings of the 6th International Conference on Applied Parallel Computing (PARA 2002): Lecture Notes in Computer Science*, 2367:185–194, 2002.
4. M.F. Cohen and D.P. Greenberg. The hemi-cube: a radiosity approach for complex environment. *Computer Graphics*, 19(3):31–40, 1985.
5. C.Puech F.X.Sillion. *Radiosity and Global Illumination*. Kaufmann Publ., 1994.
6. C.M. Goral, K.E. Torrance, D.P. Greenberg, and B. Battaill. Modelling the interaction of light between diffuse surfaces. *Computer Graphics*, 18(3):213–222, 1984.
7. P. Hanrahan, D. Salzman, and L. Aupperle. A rapid hierarchical radiosity algorithm. *Computer Graphics*, 25(4):197–206, 1991.
8. P.S. Heckbert. Radiosity in flatland. *Eurographics 92*, 11(3):181–192, 1992.
9. H.C. Hottel. *Radiant heat transmission*, chapter 4. McGraw Hill, 1954.
10. R. Orti, S. Rivière, F. Durand, and C. Puech. Radiosity for dynamic scenes in flatland with the visibility complex. *Computer Graphics Forum*, 15(3):237–248, 1996.