12 Space Partitioning for Mapping Radiosity Computations onto a Pipelined Parallel Architecture (II)

L.S. Shen
F.A.J. Laarakker
E. Deprettere

ABSTRACT A new space partitioning technique is elaborated. In part I of the paper [3], we proposed a shell-like structure which is to be superimposed on a uniform grid data structure and is adaptive to the local environment seen by a bundle of rays. Here we extend this segmentation by embedding it in a static partitioning which is determined by low resolution ray casting. This partitioning is useful in achieving a balanced computation while mapping it onto a pipelined parallel architecture. Moreover, a run-time control of workloads is applied during a subsequent high resolution ray casting so as to adjust low resolution partitioning. The technique has been tested on practical and randomly generated scenes. The performance evaluation of a pipelined parallel architecture has been done by queueing network simulation. Promising results have been obtained.

12.1 Introduction

Recently, several attempts to integrate radiosity and ray tracing have been developed. One approach could be to separate the calculation of direct and indirect lighting into two passes. The first pass calculates the indirect lighting for each diffuse patch and is done by the radiosity method. The shading over a patch changes slowly at this step so that the patch refinement can be kept coarse. The direct lighting is then calculated at the second pass by casting shadow rays as in traditional ray tracing. In this approach, there are two crucial issues to be explored. The first is to select light sources which will contribute to the direct lighting. The other is to reduce the number of shadow rays. More detailed discussions can be found in [1, 4].

To come up with a hardware-oriented version of this approach, it quickly becomes clear that in particular the second pass, which is basically software-oriented, causes serious problems. On the other hand, the first pass is very amenable to VLSI implementation due to the high degree of coherence. This expresses a high parallelism that can be exploited by hardware. Therefore, it would be beneficial to try to replace the second pass with a hardware friendly version. For example, instead of calculating the direct lighting by casting shadow rays and adding it to the indirect lighting calculated in the first pass, the final shading of an intersection point could be computed by gathering the contribution of all the other patches as in the radiosity method. However, this entails new problems because the number of rays might increase overwhelmingly. Such problems, which typically result from attempts to migrate wholly software-oriented algorithms to some kind of hardware environment, can be avoided by reconsidering the problem directly from hardware perspectives. Thus, we advocate a two-pass approach in which the two passes are based on ray casting, both making use of a low density (hence low resolution) preprocessing step to provide ray density estimates for the ultimate high density ray casting. The advantages of this approach are:

- the two passes are very similar, hence a unified set of hardware is sufficient for them;
- the ray explosions due to blind space oversampling can be avoided.

The issues of interest will be now the following. First, there is the problem of how to minimize the number of computations and their execution times as a whole. Our strategy is to combine space partitioning techniques and parallel processing. In [3], we presented a first step along this line. The main idea there was to derive a shell-like structure which is adaptive to the local environment seen by a bundle of rays. While conducting operations with this structure, the number of ray-patch intersections and also the necessity of loading patches can be reduced considerably. Furthermore, pipelinability which is generally high due to the high coherence of patches can be exploited. Second, there is the problem of how to balance workloads and hence improve pipeline efficiency when using a pipelined parallel architecture. To solve this problem, we propose an improved space partitioning based on low resolution ray casting such that each PE can be assigned with roughly equal workload, which can be further adjusted by a run-time control if necessary. This will be the main theme of this paper.

The outline of the paper is as follows. After retrieving the background of the shelling technique proposed in [3], we present an extension of this technique and a new dynamic workload balancing in Section 12.2. In Section 12.3 the queueing network model used for simulation and the results gleaned from simulation are shown.

12.2 The Shelling Technique

There is a high degree of inherent parallelism when applying ray casting to compute form-factors in the radiosity method. The problem is how to exploit this parallelism in an *optimal* way which means: (1) minimum overall computation time, (2) minimum communication overhead, and (3) linear speedup in terms of the number of PEs. We tackle this problem by exploiting benefits from (1) space partitioning, (2) parallel processing, and (3) workload balancing. The emphasis here will be on the workload balancing. The other two issues have been considered in [3]. For the sake of clearness, we shall briefly review in the next Subsection the motivation behind the proposed approach in [3]. See [3] for details. After that, we proceed with our extensions to the present approach.

12.2.1 Motivation

Space partitioning is a technique to reduce the number of ray-patch intersections by exploiting spatial coherence, which is instrumental in achieving a minimum overall computation time. One drawback is that some dependencies will be introduced. While mapping the

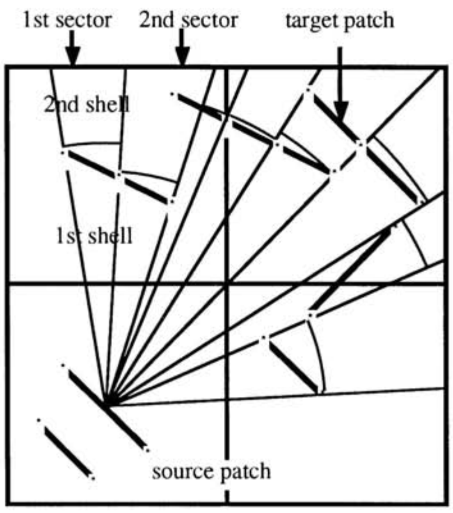

FIGURE 12.1. The ideal structure in the shelling technique.

computations onto a parallel architecture, this might degrade performance when improper partitioning schemes are used:

- The object space is partitioned into regions, and each PE is assigned with a region. The patches residing in a region are stored in the corresponding PE. With this partition, rays are passed from PE to PE via messages and consequently degrade the performance.
- When assigning neighboring rays to PEs, memory contentions which occur when multiple PEs attempt to access the same patch, will deteriorate the performance.

Certainly, it is worthy of preserving the benefits of the space partitioning technique. Now the question is how to solve the above two problems. One solution is to partition rays into sectors and each PE works with one sector on a serial ray base. However, there are two drawbacks in this approach:

- Although many sectors can be processed in parallel, each PE still works on a serial ray base within its own sector. Consequently, a patch might be loaded many times from the global or local memory because the neighboring rays are most likely to traverse similar cells and even hit the same patch. This accounts for a lot of waste in the sense of communication.
- The potential pipelinability available in computations cannot be exploited effectively. This is because cell traversal time and loading time will jeopardize the efficiency of a pipeline.

This motivated us to devise a new space partitioning called the shelling technique. Conceptually, it can be viewed as a method that strives to construct an ideal structure as shown in Figure 12.1. The half-space seen by the source patch is first subdivided into sectors. Then, a sector is subdivided into a number of shells. Ideally, a shell should match with the the active patch which actually defines the shell. When this structure has been constructed, the bundle of rays in a sector is shot and tested against the active patch within the shell currently considered. With this arrangement, each active patch needs to be loaded only once as desired. The shelling technique has a further advantage is that the resulting structure is amenable to a highly pipelined parallel architecture. It is rather straightforward that sectors can be processed in parallel while each shell within a sector can be processed in a pipelined fashion.

To make the explanation a bit easier, we have assumed an ideal structure in the above discussion. In fact, there are two detrimental factors to do this:

- It is very costly to match a shell with the active patch which defines the shell. This is because the degree of ray coherence of this patch is hard to be derived.
- The number of computations within a sector is not enough to fill up the pipeline especially when rendering an environment constituting of smaller patches. This may result in an inefficient usage of the pipeline when mapping these computations onto a pipelined parallel architecture.

In [3], we proposed an adaptive tracking mechanism to overcome the first problem. As to the second problem, which has not been addressed yet, our strategy is to maintain a balanced structure by controlling the granularity of computations through static partitioning and balancing workloads through run-time control. For this purpose, we extend the shelling technique to be a partitioning of space such that a balanced computation can be achieved while mapping this partitioning to a pipelined parallel architecture. Notice that the adaptive coherence tracking is now a local procedure embedded in the partitioning.

12.2.2 Static Partitioning

As shown in Figure 12.2, the object space is partitioned into sections (S), sectors (S'), shells (s), and subshells (s'). A subshell is the primitive construct for conducting computations, which can be set adaptively by neighborhood information, and a shell is a grouping of subshells. Those two constructs are relevant to the scheduling of computations. As for the other two constructs, they are meaningful for processor assignment as can be seen in the following.

While mapping this partitioning onto a pipelined parallel architecture, a section and its corresponding sectors can be mapped onto a cluster which consists of an intersection computation unit (ICU) and a number of cell traversal units (CTUs). Both ICU and CTU are of type pipeline. With this arrangement, the adaptive coherence tracking is only kept within a sector. It is not necessary to track across a sector because the CTU assigned for a sector can derive tracking information of its own. The problem is how to design a computational array in which the computations can be balanced and the communication

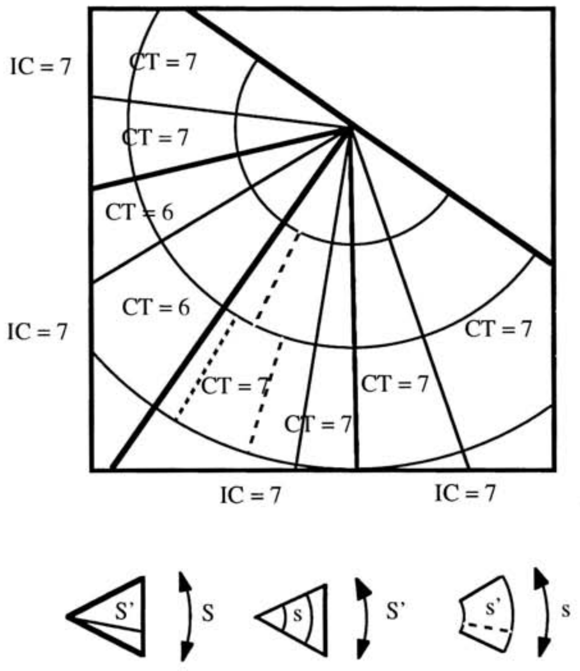

FIGURE 12.2. An example of the static partitioning.

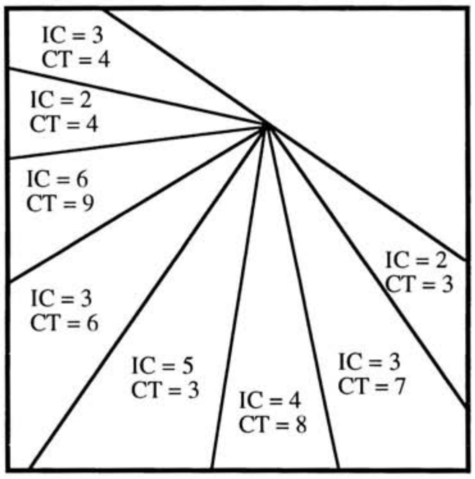

FIGURE 12.3. In 2D case, the object space is subdivided into subregions by equally distributed $\Delta\theta$s.

overhead between PEs and the communication with the host do not have unacceptable impacts on the overall performance of the array. As we explained in [3], the communication overhead between the host and PEs can be alleviated with the shelling technique. In an attempt to achieve a balanced computation, the granularities of the sections should be controlled such that all ICUs have approximately the same workload and have as high pipeline efficiency as possible. Similarly, the granularties of the sectors should be controlled such that all CTUs within a section have approximately the same workload and have as high pipeline efficiency as possible. The difficulty is that workloads cannot be precisely known in advance. Our approach consists of three steps:

- In 2D case, the object space is subdivided into subregions by equally distributed $\Delta\theta$s as shown in Figure 12.3.
- To start with a low resolution ray casting, so that only a small number of rays is shot. Then the number of cell traversals and the number of intersection computations for each subregion are recorded.
- Based on the information of the low resolution ray casting, the object space is reorganized into sections and sectors such that the number of intersection computations for one section should be close to that of other sections and also close to the number of cell traversals for each sector belonging to this section. This is done by a technique called BSP in the field of computer graphics. Firstly, the object space is recursively subdivided into sections by using median constant θ planes. Then each section is recursively subdivided into sectors by using median constant θ planes. An example of this subdivision is shown in Figure 12.2.

This technique has been tested on different scenes. As will be seen in the next Section, a high deviation from that is expected can be observed for a few sections. This entails the necessity of using a dynamic workload balancing to adjust the low resolution partitioning.

12.2.3 Dynamic Workload Balancing

In order to insure a good balancing, the static partitioning should be fine-tuned with a dynamic workload balancing. When a PE completes its own computations, it sends a request to get a work-item from a busy PE. A work-item is a fixed amount of data which can trade for the communication overhead in transmitting them. With this approach, the communication between PEs can also be alleviated because it is not necessary to adjust workloads all the times and the amount of data transmitted can trade for the cost spent on transmitting them.

12.3 Performance Evaluation

The shelling technique as described in 12.2 has been implemented in the C language. It has been tested on practical and randomly generated scenes. While mapping a partitioning onto a pipelined parallel architecture, performance evaluation has been done by queueing

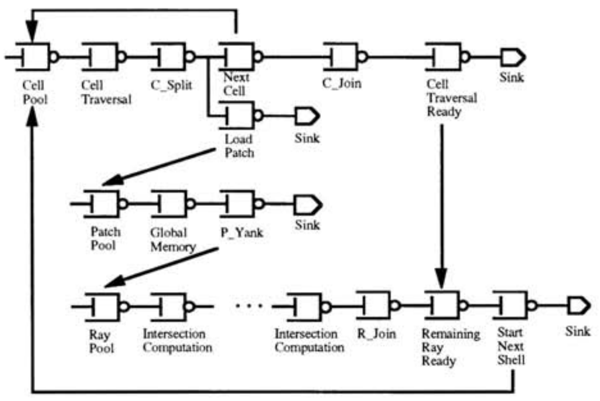

FIGURE 12.4. The queueing network of the shelling technique without workload balancing.

network simulation. In this Section, we first describe the queueing network model of the shelling technique. Then the results of a pratical scene and a random generated scene will be shown.

12.3.1 Queueing Network Model

In queueing network, physical resources are modelled as service centers and each task is modelled as a customer circulating among the service centers. Queueing networks are useful as performance models of systems where performance is principally affected by contention for resources which would be major concerns in our case. We have chosen PAW as our modeling and simulation tool. It is a discrete event simulation system operating in a world of queueing networks in which nodes and transactions are used to represent service centers and customers, respectively. For more detailes can refer to [2]. In the following, we explain the behavioral aspects of the network (refer to Figure 12.4).

- Initially, Cell Pool node holds tokens which represent nonempty cells sorted into the order of being traversed within a sector. When a transaction enters the yank node Start Next Shell, a yanking action is invoked by yanking a token from Cell Pool node and route it to Cell Traversal node. After a service time that denotes the cell traversal time of this nonempty cell is consumed, two yanking actions are invoked to request loading all the patches within this cell and start traversing next nonempty cell. When a batch of transactions, which is equal to the number of nonempty cells within a shell, enter C_Join node, they will be collapsed into one tranaction and routed to Cell Traversal Ready node.

- All the relevant patches are stored as initial tokens in Patch Pool node. When a yanking action initiated by Load Patch node, a number of tokens, which represents all the relevant patches within the nonempty cell, can be yanked from Patch Pool

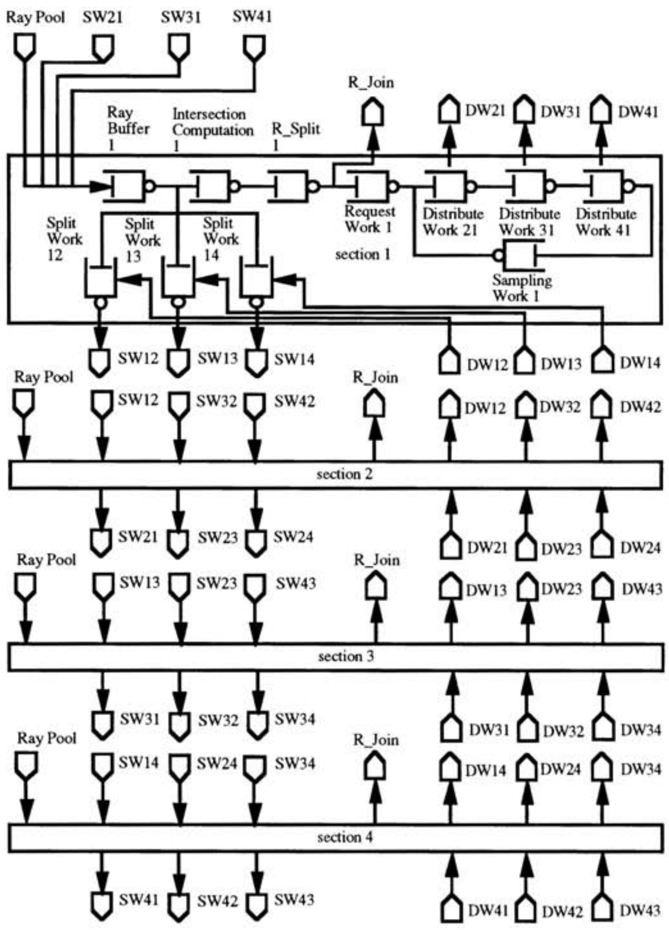

FIGURE 12.5. The queueing network of the shelling technique with workload balancing.

node and routed to Global Memory node. Patches are queued in Global Memory node to model that there is only one unique memory to store the complete database.

- When a patch is loaded from the global memory, a bundle of rays, which is stored as initial tokens in Ray Pool node, can be yanked from that node and routed to Intersection Computation node. In order to model the pipelined behavior of intersection computations, each stage of the pipeline should be modelled by an Intersection Computation node. If a batch of rays, which is equal to the number of intersection computations within a shell, enter R_Join node, they will be collapsed into one transaction and routed to Remaining Ray Ready node. Together with the yanking action initiated by Cell Traversal Ready node, another yanking action can be initiated by the yank node Start Next Shell, which indicates that next shell can start with cell traversal.

In the above, we have described the queueing network of the shelling technique without workload balancing. As discussed earlier, system performance can be further improved through dynamic workload balancing. Some modifications in the network must be made to model this run-time control behavior. We explain this in the following (refer to Figure 12.5):

- Each section is now associated with a Ray Buffer node which is normally used to fill up the pipeline of its corresponding Intersection Computation node. Only when none is left in the queue of a Ray Buffer node, a work request transaction is generated to ask for a work-item from a busy Ray Buffer node. The work-item is moved in through a Split Work node by a yanking action which is initiated by a yank node Distribute Work.
- If the requirement of work-item cannot be satisfied at the instant of requesting work, the request transaction will circulate in Distribute Work nodes with a specified rate defined in Sampling Work node. It should be noted that the size of work-item and the rate of sampling work will have influence on the effectiveness of balancing.

It should be noted that all the information needed for constructing the queueing network are acquired by running a serial program implementing the shelling technique. Based on that information, a program can map the static partitioning onto a pipelined parallel architecture which is modeled as a queueing network. Then a trace-driven simulation is invoked to estimate the performance of the network.

12.3.2 The Results

In this Subsection, the results of two scenes as depicted in Figures 12.6 and 12.7 are shown. The control parameters of the shelling technique are shown in Table 12.1. In the following, three factors which are relevant to the performance are shown:

- Workload Distribution: For the two test scenes, the workloads of ICUs after applying static partitioning are shown in Figures 12.8, 12.9, 12.10, 12.11, 12.12, and 12.13.

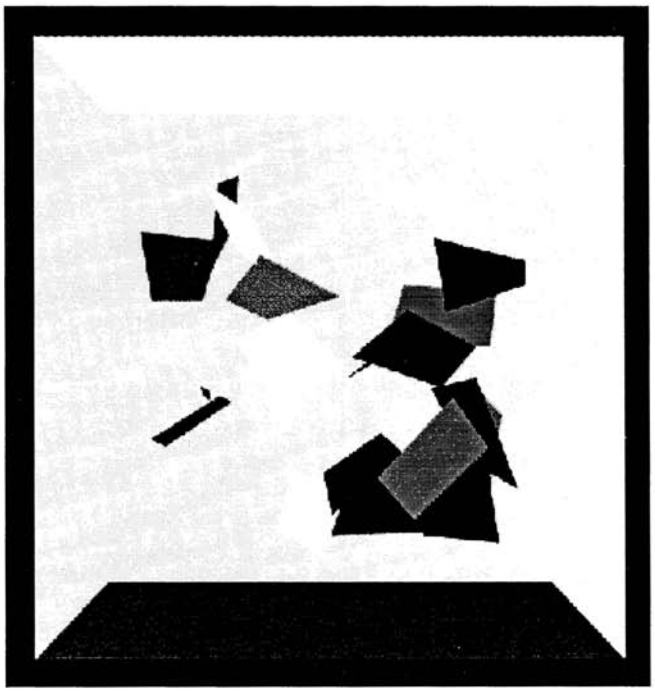

FIGURE 12.6. The *test scene 1* consists of 6 walls and 20 randomly generated patches.

TABLE 12.1. The control parameters of the shelling technique for the two test scenes.

Number of Hemisphere Rays	969
Hi_Low_Ratio	1, 10, 30
# of Sectors in Low Resolution	25
# of Sections in High Resolution	2, 4, 8
Scene Boundaries	(0, 0, 0) - (1, 1, 1)
Size of a Cell	0.05
Shell Radius	0.289

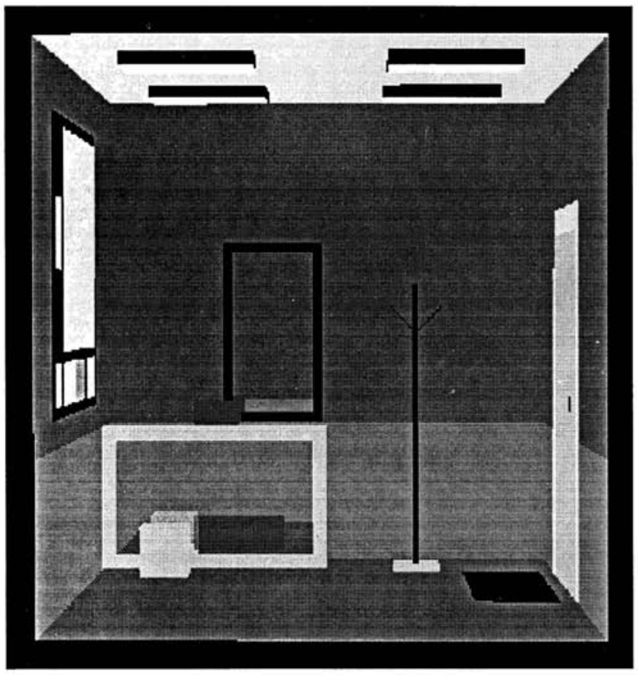

FIGURE 12.7. The *test scene 2* consists of 808 polygonal patches.

The Hi_Low_Ratio represents the proportionality of the number of rays used in both high and low resolution ray castings. Its value is dependant on the ADLC[1] of a scene. In our cases, it is favourable to choose 10 as the Hi_Low_Ratio.

- Pipeline Efficiency: The pipeline efficiency is defined as the percentage of busy time-space span over the total time-space span, which equals the sum of all busy and idle time-space spans. Without using dynamic workload balancing, the pipeline efficiency of the *test scene 2* is shown in Figure 12.15. As can be seen in Figure 12.15, 15% improvement can be achieved while exploiting dynamic workload balancing.

- Speedup: The speedup is the ratio of the overall computation time of one PE to that with p PEs. It represents, in fact, the number of PEs effectively used during the parallel processing of an algorithm. The ideal speedup is never achievable because of dependencies and communication overheads. On average, 15% improvement can be achieved by exploiting dynamic workload balancing.

[1]The Average Degree of Local Coherence of a scene represents the average number of rays hit a patch in the scene.

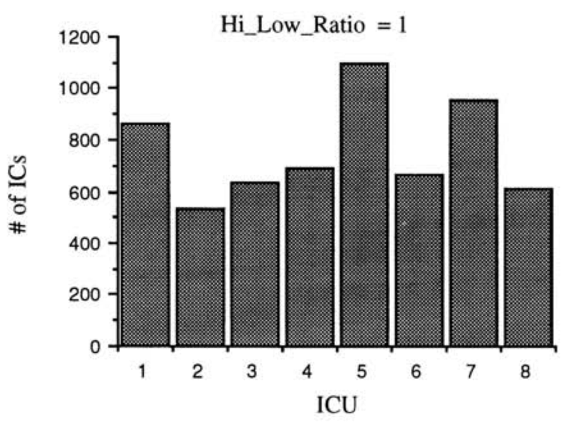

FIGURE 12.8. The distribution of workloads for the *test scene 1*.

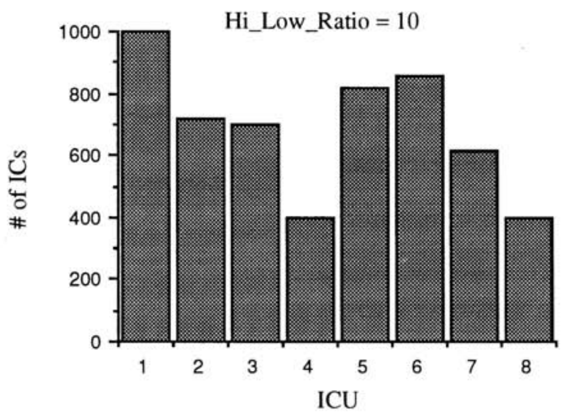

FIGURE 12.9. The distribution of workloads for the *test scene 1*.

12.4 Conclusion

In this paper we have proposed a new space partitioning technique such that the requirement of loading patches can be reduced considerably. By controlling the granularity through space partitioning and balancing workloads through run-time control, it is possible to achieve a balanced computation on a pipelined parallel architecture. Our future work will be to establish a theoretic basis for the shelling technique and to set up an

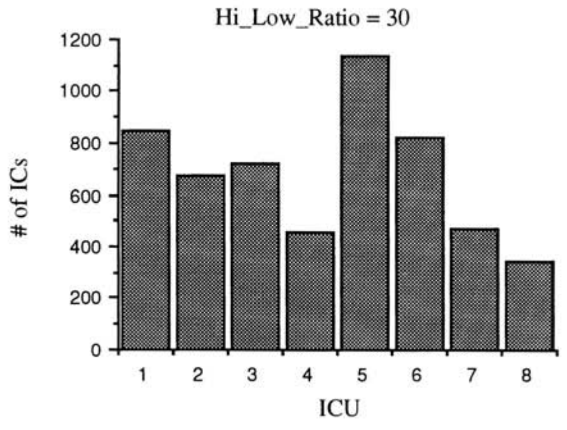

FIGURE 12.10. The distribution of workloads for the *test scene 1.*

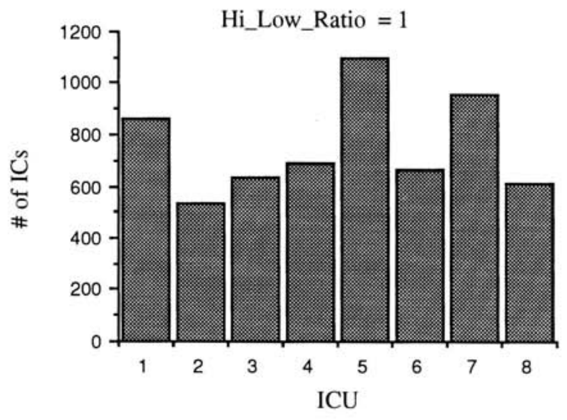

FIGURE 12.11. The distribution of workloads for the *test scene 2.*

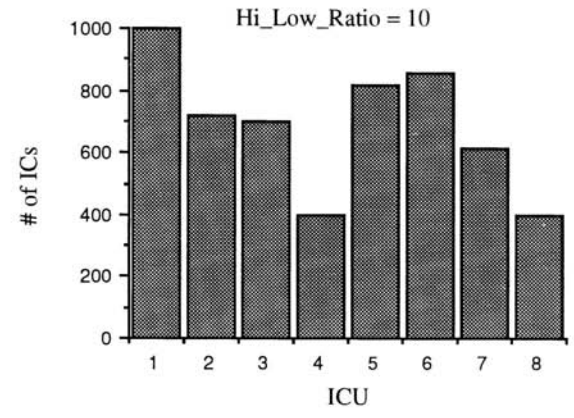

FIGURE 12.12. The distribution of workloads for the *test scene 2*.

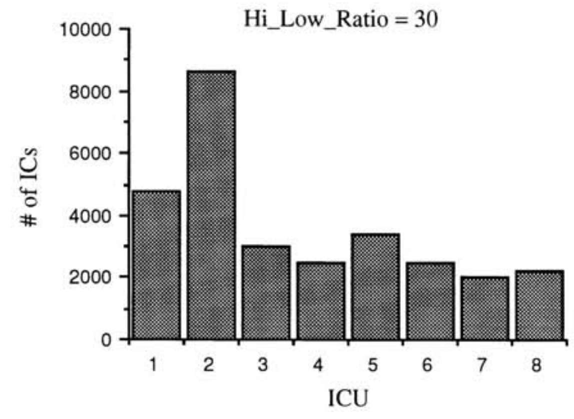

FIGURE 12.13. The distribution of workloads for the *test scene 2*.

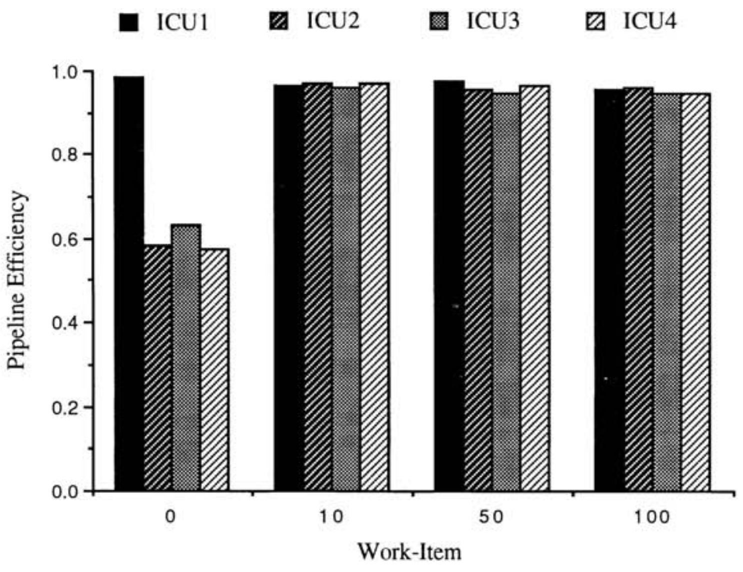

FIGURE 12.14. The pipeline efficiency vs. work-item for the *test scene 2*.

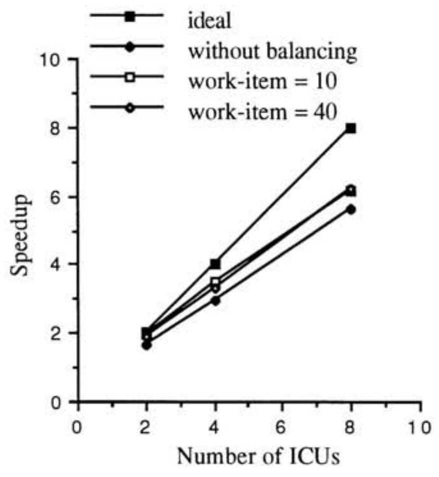

FIGURE 12.15. The speedup vs. the number of ICUs.

environment to facilitate the mapping procedure. The key issues in that work will be the computational structure of the problem, the modelling of an environment and the underlying architecture, and most of all a suitable language for describing them.

Acknowledgements:

This research has been supported in part by the commission of the EEC under the ESPRIT program, project BRA 3280 (NANA) and by the Dutch Technology Foundation under contract DEL 99.1982. The authors also like to thank Prof. F.W. Jansen and A.J.F. Kok from Informatics Department for many fruitful discussions.

12.5 References

[1] A.J.F. Kok and F. W. Jansen. Efficient Complete Radiosity Ray Tracing Using a Shadow Coherence Method. *Technical Report, Delft University of Technology*, 1991.

[2] B. Melamed. The performance analysis workstation: An interactive simulation package for queueingnetworks. *Proceedings, FJCC*, pages 729-740, 1986.

[3] L. S. Shen, Ed. F. Deprettere, and P. Dewilde. A New Space Partitioning Technique to Support a Highly Pipelined Parallel Architecture for Radiosity Method. In: R.L.Grimsdale, A. Kaufman (Eds.):*Advances in Computer Graphics Hardware V*, EurographicSeminars. Springer-Verlag, Berlin, 1992, p.153-170.

[4] P. Shirley. A Ray Tracing Method for Illumination Calculation in Diffuse-Specular Scenes. *Proceeding Computer Graphics Interface '90*, pages 205-212, 1990.