Hybrid Scheduling for Realistic Image Synthesis

Erik Reinhard[1] Alan Chalmers[1] Frederik W. Jansen[2]

Abstract

Rendering a single high quality image may take several hours, or even days. The complexity of both the model and the lighting simulation may require excessive computer resources. In order to reduce the total rendering time and to accommodate large and complex models that exceed the size of a single processor system, a parallel renderer may provide a viable alternative to sequential computing.

In this paper, a data-parallel strategy is applied to allow large models to be distributed over the processors' memories. The resulting uneven workload is then balanced by scheduling demand driven tasks on the same set of processors. Tasks are scheduled in either demand driven or data parallel fashion, according to ray coherence. Implications of this hybrid algorithm with respect to performance, caching and memory usage are investigated.

1 Introduction

Ray tracing [8] simulates the behaviour of light by shooting rays from the eye point into the scene (figure 1). Pixels are coloured according to the object that was hit. Shading and shadowing is computed recursively by shooting rays from the intersection point of the ray and the object towards the light sources and into reflected and refracted directions.

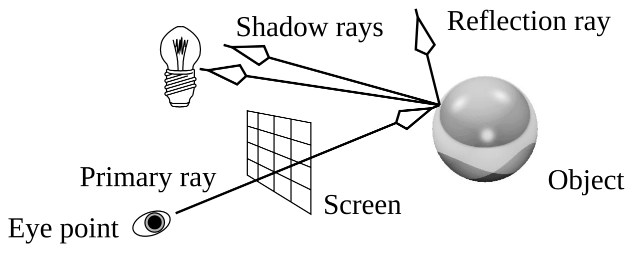


Figure 1: Ray tracing overview.

Lighting simulations are expensive algorithms, whose irregular data access patterns complicate parallel solutions. This is especially true when large scenes need to be rendered on distributed memory computers. In this paper, a ray tracing algorithm is analysed with respect to data accesses and a hybrid scheduling solution and its memory usage are evaluated.

[1]University of Bristol, UK. E-mail: reinhard|alan@cs.bris.ac.uk
[2]Delft University of Technology, The Netherlands. E-mail: f.w.jansen@its.tudelft.nl

Previous parallel ray tracing approaches usually fall into one of two classes: data parallel and demand driven scheduling. Demand driven [3], usually equivalent to image space partitioning [1], subdivides the screen into a number of regions. Whenever a processor finished a task, it requests a new one from the master until the image is complete. Data accesses to most of the scene geometry require scene replication for this method to be most effective. Otherwise caching could be used to some extent to salvage performance loss due to excessive data communication. The effectiveness of caching is discussed in the following section.

Data parallel approaches [2] partition object space by distributing the geometry over the processors. Ray tasks are then migrated to the processors that hold the relevant data. A single ray may therefore pass through multiple processors before an intersection is found. At the cost of efficiency, this method allows very large scenes to be rendered (see also next section for a discussion).

Hybrid scheduling techniques may overcome the disadvantages of both these methods. One such a scheme separates the work into ray traversal and intersection tasks [6]. The former would require only a limited amount of data in the form of the spatial subdivision structure, while for the latter the full object geometry would be required. The ray intersection tasks are scheduled data parallel and provide a basic but uneven load, while the ray traversal tasks are scheduled demand-driven to compensate for this load imbalance. However, ray traversal typically takes less than 10% of the total computation time, and is therefore not computationally intensive enough to effectively balance the load.

Alternatively, coherence could be used to subdivide tasks into data parallel and demand driven components [5]. Coherence can be found in bundles of rays that leave from one point and travel in similar directions. Coherent ray bundles, which require a relatively small data set while being sufficiently expensive, are good candidates to be scheduled as demand driven tasks, while the remainder of the tasks is handled in a data parallel fashion. Both primary and shadow rays can be grouped into coherent bundles.

This paper first shows the behaviour of object accesses during ray tracing and explains why neither demand driven nor data parallel solutions are likely to be efficient when the scene geometry is distributed over the available processors. Then our parallel algorithm is briefly described in section 3. Memory and cache management is discussed in section 4, followed by experiments and conclusions in sections 5 and 6.

2 Scene analysis

Assuming that no super-sampling is performed to reduce aliasing artifacts, the computations associated with different pixels are independent. Demand driven screen space subdivision algorithms exploit this independence by farming out different parts of the image to different processors. However, sampling an entire ray tree for a single pixel may involve a potentially large and unpredictable subset of the scene geometry. Demand driven algorithms therefore give best performance when the scene data is replicated with each processor. In the case that the processors' local memories are too small, the scene will have to be distributed. In order to reduce the number of data fetches, normally object caches are employed. These caches work under the assumption that object accesses are coherent.

On the other hand, data parallel algorithms distribute the scene over the processors' memories and ray tasks are migrated to the processors that store the relevant data. These algorithms assume that an object distribution can be found which equalises the workload associated with each processor. Unfortunately, predicting object distributions which equalise the workload as well as the memory consumption with each processor, is a non-trivial problem which is as yet unsolved.

The validity of the assumptions made for both demand driven and data parallel processing, can be assessed by collecting statistics during sequential rendering of a number of scenes. A simple count of the number of intersection tests performed for each object provides insight in the distribution of work over the scene. For these tests (and the work presented elsewhere in this paper), the Radiance package [7] was modified to collect these statistics. The test scenes are depicted in figure 2.

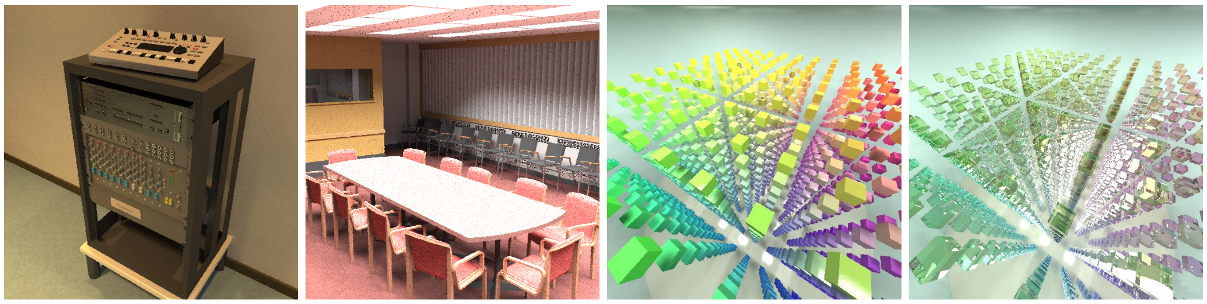


Figure 2: Left to right: studio model (5311 objects), conference room (3951 objects), colour cube and transparent colour cube (11026 objects each).

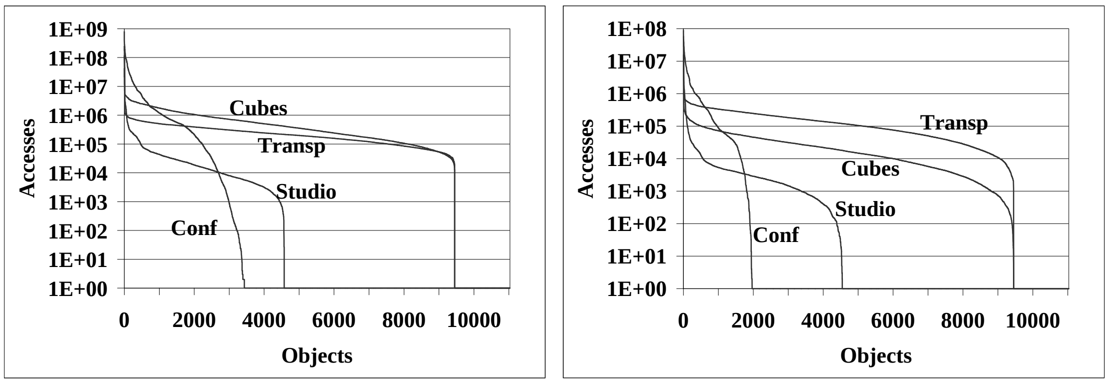


Figure 3: Object accesses sorted by frequency with (left) and without (right) diffuse inter-reflection. Note the logarithmic scale on the vertical axis.

As Radiance has the capability to sample diffuse inter-reflection, by sampling a hemisphere of rays into all directions seen from the intersection point, its effect on the distribution of work over the scene is measured as well. Results for the four test scenes are depicted in figure 3. In all cases, besides increasing the workload by an order of magnitude, the distribution of object accesses is more even when diffuse inter-reflection is sampled. For the conference model, a substantial larger set of objects is used during rendering as well.

For all scenes a small subset of the objects is intersected three orders of magnitude more often than the majority of the objects. This peak is good for demand driven algorithms, because typically these objects should be cached with all processors. However, the remainder of the objects in the scenes are still intersected quite often, and hence the efficiency of caching algorithms can still be low. For data parallel processing, the peak to the left of the figures normally causes severe load imbalances, but a large majority of the objects are accessed more or less evenly. This peak is mainly caused by the light sources, which attract a large portion of the workload, but unfortunately other objects cause load imbalances too.

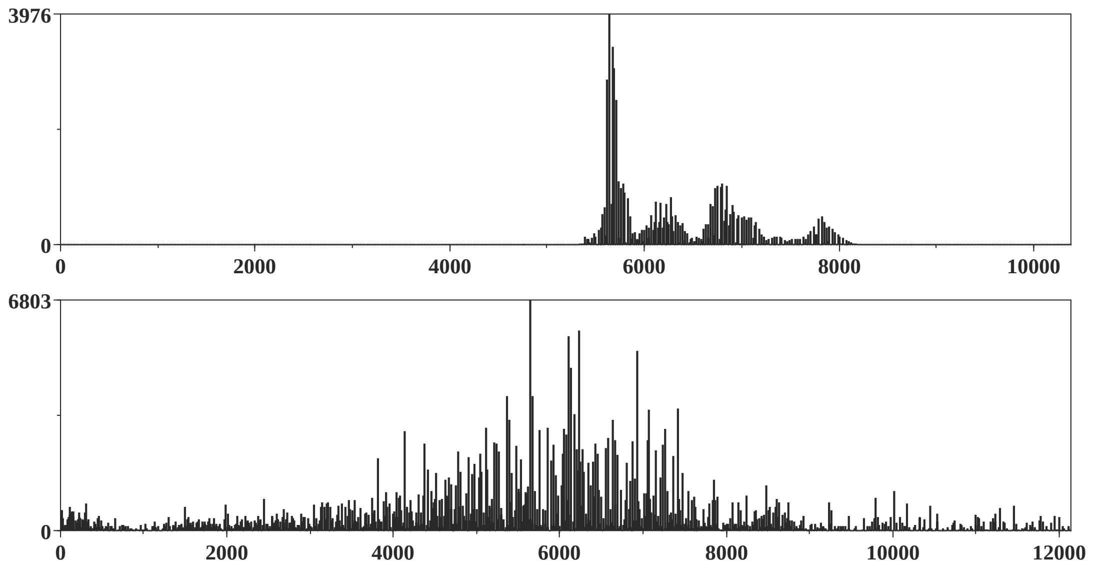


Figure 4: Temporal behaviour for a typical object of the colour cube without (top) and with (bottom) diffuse inter-reflection.

The temporal access pattern is assessed by looking at the median objects. For the colour cube model, the data accesses per second for one of its objects near the median is given in figure 4. These graphs show that adding diffuse inter-reflection to the repertoire of lighting effects, has a detrimental effect on temporal coherence. But even when no diffuse inter-reflection is sampled, the data accesses for this object are spread out over a considerable amount of time. For other scenes temporal coherence can be better without diffuse inter-reflection, but when ambient sampling is performed, temporal coherence is always lost, irrespective of which scene is rendered.

In all, the workload distribution over the objects in the scene is too even to allow the design of efficient caches in demand driven algorithms. On the other hand, the peak of object accesses for some objects leads to poor efficiency in data parallel approaches. Hence, we propose to split the problem into a demand driven component and a data parallel component (section 3). Caching will be used to serve objects for the demand driven part (sections 4 and 5).

3 Hybrid scheduling

We propose to deal with the irregular structure of ray tracing by using a hybrid scheduling algorithm. The algorithm consists of a demand driven part and a data parallel part. Each processor handles both types of tasks, but gives data parallel tasks a higher priority, thus achieving automatic load balancing.

Bundles of primary rays and shadow rays are executed as demand driven tasks, while specular reflection and refraction, as well as any diffuse inter-reflection is by necessity executed as data parallel tasks. The majority of ray tasks is therefore computed in demand driven mode, ensuring load balancing capability until the end of the computation. This was not yet achieved in previous research presented in [5].

The demand driven tasks in this system require objects to be fetched, and a caching mechanism is implemented to reduce overall communication requirements. Because the most unpredictable rays are now executed as data parallel tasks, cache consistency is expected to be better than in pure demand driven approaches. Its details and behaviour are explained in the following two sections. A more general overview of the algorithm can be found in [4].

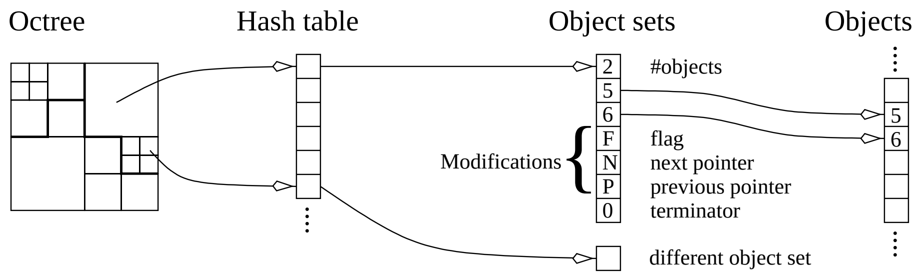


Figure 5: Radiance's data structure, including modifications required for parallel rendering.

4 Memory management

In our hybrid algorithm we would like to be able to fetch data for certain demand driven tasks and have the data parallel component benefit from these fetches without any additional overhead. For this reason, the cache is incorporated into the main spatial subdivision structure, which is based upon an octree (figure 5). All the non-empty leaf nodes in the octree point to an index in a hash table, the hash function being the sum of the objects contained within the leaf node. The hash table points to a set of pointers to objects. Hence, leaf nodes containing the same objects, will point to the same index in the hash table. In order to determine which processor stores the data in this object set, a flag field is added to each of these sets. This field stores the processor number, as well as a flag indicating whether this object set is cached or not. Whenever a voxel is not resident, it can therefore be requested from the processor which is indicated by this flag field. Currently, the octree and the object sets are replicated with each processor. The objects themselves are distributed.

During ray traversal, for each new voxel traversed, this field is checked to determine whether the data for this voxel is available or not. If not, the ray task is migrated to the processor indicated by the flag field. Otherwise, the objects are intersected with the ray.

When memory is full, some data needs to be freed, in order to make space for newly requested data. To this extent, the cached voxels all link to each other using a doubly linked list, for which the next and previous fields are used (figure 5). When a voxel is traversed, the object set is unlinked from this list and re-inserted at the front. Clearing voxels proceeds from the end of the list, effectively implementing a LRU scheme.

5 Experiments

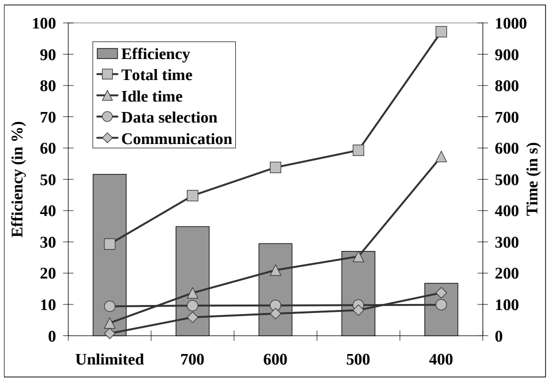
During the course of the computation, two different types of data needs to be stored: objects that can be removed at any time to create more storage space, and data that can not be freed. The resident set belongs to the latter category, as well as storage of intermediary results (intersection points that await results from other processors before shading can be performed) and input buffers. As the size of the object cache is changed, variations in performance and memory consumption can be expected. The smaller the cache, the more data fetches are required, resulting in longer idle time and reduced efficiency. At the same time, intermediary results may need to be stored longer, and therefore memory consumption may be negatively affected. If a processor becomes a bottleneck, then the number of tasks being queued for that processor may become rather large as well.

In order to test this, the studio model (figure 2) was rendered on a Parsytec PowerXplorer using our hybrid scheduling version of Radiance. This machine consists of 32 PowerPC 604 nodes running at 133 MHz with 96 MB of memory per node, connected by a network with a sustained speed of 27 MB/s.

Linear speed-ups with an efficiency of around 50-60% are obtained when the object cache is large enough to hold whatever data is required by each processor. In that case the algorithm is scalable at least up until 32 nodes. The efficiency is not higher because overhead is incurred for selecting the relevant data for each of the demand driven task. However, this overhead is scene and parameter dependent and therefore affects efficiency, but not scalability.

However, when the cache size is artificially limited to a fixed size, efficiency, scalability and memory consumption does change. Figure 6 (left) shows the efficiency for the studio model on 8 processors with an object cache size that ranges from unlimited to 400 kB along the horizontal axis. The same graph als shows the total rendering time als well as idle time, data selection time and time spent in communication routines. By reducing the object cache size, the loss of efficiency is almost entirely due to an increase in idle time, which is mainly the result of one processor being overloaded. A load balancing bottleneck arises because the smaller the object cache, the less the capability of the demand driven component to balance the workload. Hence, The importance of the demand driven component varies according to the size of its object cache and therefore, this hybrid scheduling algorithm sits in-between pure demand driven and pure data parallel approaches.

When the size of the object cache is reduced to under 400 kB for the studio model, the ren-



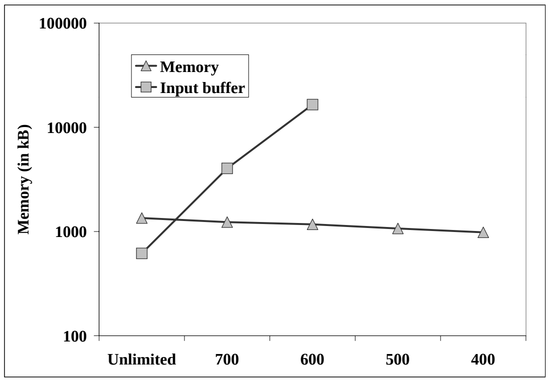


Figure 6: Efficiency and timing (and its components) for the studio model on 8 processors (left) and the associated maximum memory consumption (right).

dering time increases disproportionately. It appears that this is the point when the most important objects of the studio model, as discussed in section 2, cease to fit in the object caches.

The total memory consumption (figure 6 right), which includes both the object cache and storage of intermediary results, but excludes PVM's input buffers, shows that generally the memory overhead is acceptable and reduces when the object cache is made smaller. This pattern is well-behaved and according to expectation, although with smaller cache sizes, intermediary results may be in memory longer before shading occurs. Unfortunately, for smaller cache sizes, the data parallel component becomes more important (as argued above), and the resulting processing bottleneck is accompanied by a large input buffer of tasks awaiting execution (also depicted in figure 6 right[3]). This is a problem which could be solved by carefully monitoring the amount of work in the system at any one time. By reducing the total amount of work, the processing bottleneck at one processor is not resolved, but the input buffer should be shorter.

As it generally is cheaper for an overloaded processor to serve data to another processor than to execute ray tasks, another possible solution would be to allow data to be fetched more than once for those tasks that required data that was once cached but already replaced by new data. Currently, this memory problem is the most important remaining issue that prevents this algorithm to render very large scenes.

6 Conclusions

In this paper, we have analysed the spatial and temporal behaviour of ray tracing and it is shown that this is an irregular problem which complicates possible parallel implementations, assuming the memory associated with each processor is not sufficient to replicate the scene data. Most current parallel implementations are not suitable for efficiently rendering large scenes, i.e. scenes that do not fit into a single processor's memory. When memory permits, our hybrid scheduling algorithm resembles an efficient demand driven approach, but when larger scenes need to be

[3]Measuring the size of a PVM input buffer is inefficient and affects the performance of the algorithm. For this reason, these measurements were carried out separately on a cluster of 8 SGI O2's.

rendered, resulting in less available memory per processor for object storage, the algorithm's performance progressively deteriorates towards a data parallel solution. It is therefore more capable of exploiting the available memory than pure data parallel implementations, while also being more efficient than demand driven approaches for those cases where the scene data can not be completely replicated. A gradual trade-off between memory use and performance is therefore possible, thus optimising efficiency when large scenes are to be rendered. However, before this can be achieved within our implementation, a solution for the input buffer problem needs to be devised.

Acknowledgements

This work was sponsored by the European Commission under TMR grant number ERBFM-BICT960655 and by the Stichting Nationale Computerfaciliteiten (National Computing Facilities, NSF) for the use of supercomputer facilities, with financial support from the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Netherlands Organisation for Scientific Research, NWO).

References

[1] F. C. Crow, G. Demos, J. Hardy, J. McLaugglin, and K. Sims. 3d image synthesis on the connection machine. In *Proceedings Parallel Processing for Computer Vision and Display*, Leeds, 1988.

[2] M. A. Z. Dippé and J. Swensen. An adaptive subdivision algorithm and parallel architecture for realistic image synthesis. In H. Christiansen, editor, *Computer Graphics (SIGGRAPH '84 Proceedings)*, pages 149–158, July 1984.

[3] S. A. Green and D. J. Paddon. Exploiting coherence for multiprocessor ray tracing. *IEEE Computer Graphics and Applications*, 9(6):12–26, November 1989.

[4] E. Reinhard, A. Chalmers, and F. W. Jansen. Hybrid scheduling for parallel rendering using coherent ray tasks. In *Proceedings Parallel Visualization and Graphics Symposium*, 1999.

[5] E. Reinhard and F. W. Jansen. Rendering large scenes using parallel ray tracing. *Parallel Computing*, 23(7):873–886, July 1997. Special issue on Parallel Graphics and Visualisation.

[6] I. D. Scherson and C. Caspary. Multiprocessing for ray tracing: A hierarchical self-balancing approach. *The Visual Computer*, 4(4):188–196, 1988.

[7] G. Ward Larson and R. A. Shakespeare. *Rendering with Radiance*. Morgan Kaufmann Publishers, 1998.

[8] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, jun 1980.