A Parallelisation of Ray Tracing with Diffuse Interreflection

Erik Reinhard

Faculty of Technical Mathematics and Informatics,
Delft University of Technology
Julianalaan 132, 2628BL Delft, The Netherlands
Email: erik@duticg.twi.tudelft.nl

Abstract

Ray tracing is a powerful technique to generate realistic images of 3D scenes. However, the rendering of complex scenes under advanced lighting circumstances may easily exceed the processing and memory capabilities of a single workstation. Distributed processing offers a solution if the algorithm can be parallelised in an efficient way. In this paper a hybrid scheduling approach is presented that combines demand driven and data parallel techniques. Which tasks to process demand driven and which data driven, is decided by the data intensity of the task and the amount of data locality (coherence) that will be present in the task. By combining demand driven and data parallel tasks, a good load balance is achieved, while at the same time spreading the communication evenly across the network. This leads to a scalable and efficient parallel implementation of the ray tracing algorithm with fairly no restriction on the size of the model data base to be rendered.

1 Introduction

Ray tracing is a powerful and widely used technique to generate highly realistic images of 3D scenes. It calculates the reflection of light in a scene by tracing the path of light backwards from the viewpoint towards the light sources. Ray tracing is ideally suited for lighting simulations.

Applications include a.i. architectural design, theatre and greenhouse lighting simulations and traffic lighting simulations (tunnels, crossings). An example of an indoor lighting simulation using the Radiance lighting simulation package (Ward 1994) is given in figure 1. Rendering such complex scenes may easily exceed the processing and memory capabilities of a single workstation, especially if diffuse reflection needs to be computed. This is the case in for example Radiance.

Distributed processing may offer a solution to excessive processing demands provided the algorithm can be parallelised in an efficient way. A parallel rendering algorithm should both minimise communication requirements and balance the workload well over the processors available. Various mechanisms exist to accomplish this.

In data parallel solutions, objects are distributed over processors and ray tasks are handled by those processors that possess the relevant data. Objects may be dynamically redistributed to provide a better load balance, but this introduces extra object communication. Furthermore, redistributing objects too quickly may introduce oscillations, so that object communication becomes a bounding factor. However, if objects are not distributed quickly enough, a sub-optimal load balance is achieved.

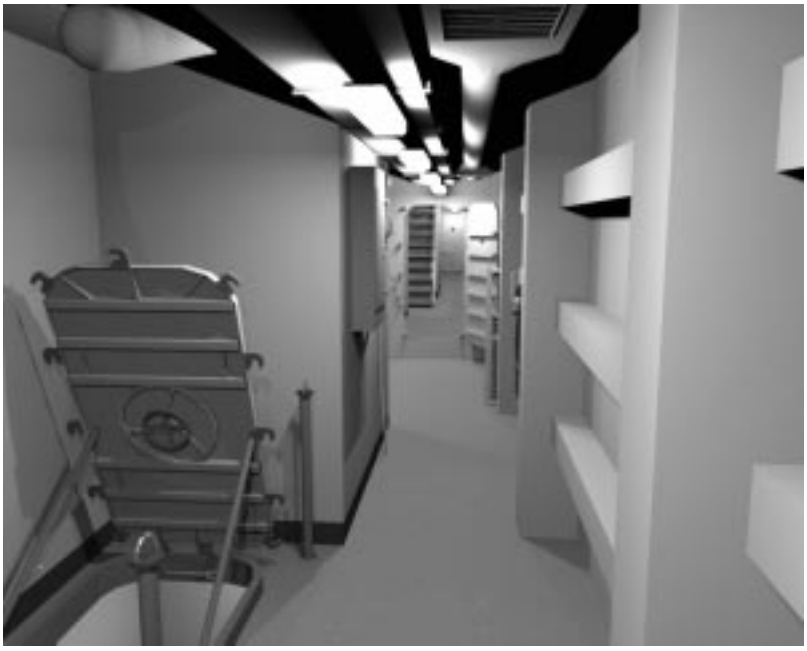

Figure 1: US Naval Cruiser (U.S.S. Generic) lower deck. The model was created by Saba Rofchaei and Greg Ward under contract from the U.S. Navy and rendered using the Radiance package.

Demand driven approaches usually split the screen into equal sized regions and assign each region (or a number of regions) to a processor. The number of objects visible in each region now more or less determines the workload of each task. A good load balance can be dynamically maintained by a master process that keeps track of when each processor finishes a task.

A well balanced workload is therefore easier obtained by employing demand driven techniques than with a data parallel approach. However, demand driven solutions work best when each processor is capable of storing the entire scene description in local memory. This is an unacceptable restriction in most applications, so objects are normally distributed over processors, with additional caching of objects to reduce communication requirements.

However, caches are only effective provided that sufficient data locality is present. For a number of coherent rays this may be guaranteed. Examples are primary rays, specular reflection rays and light rays that sample area light sources. For these rays, caching is a viable solution, so that they are best handled as demand driven tasks. Communication requirements should be low and a good load balance may be achieved through a scheduling technique that sends new tasks to processors that have finished previous tasks.

A remaining set of ray tasks will not be coherent and therefore caching will not be effective for these rays. Diffuse reflection rays in particular show unpredictable behaviour with respect to data access. As it is impossible to store the entire scene description in a single cache, a data parallel approach is a better alternative for these rays. A ray task is then sent to the processor containing the relevant data, instead of fetching a large amount of remotely stored data that the ray will not intersect with anyway.

Given the above, neither a purely demand driven, nor a data parallel implementation can handle very large models efficiently. We suggest that for the parallelisation process the hybrid demand

driven/data parallel approach be further explored. Experiences with simple ray tracing (Reinhard & Jansen 1995) show that large models can be rendered with a combination of demand driven and data parallel tasks, provided that most work is performed demand driven while any remaining rays that show no apparent coherence are executed as data parallel tasks. Additionally, this approach assures that a single processor's memory provides no upper limit on the model size that can be rendered.

2 Algorithm

In order to create a parallel version of Radiance (Ward 1994), including its diffuse reflection sampling, a similar path as that taken for Rayshade (Reinhard & Jansen 1995) may be followed (with a few notable exceptions). The parallelisation will follow the hybrid approach, where coherent subtasks will be formed and executed as demand driven tasks, and the remainder of the work, for which coherence is not obvious, is scheduled data parallel. The emphasis of the resulting system will be on both model size and efficiency.

In order to render very large models, the data should be distributed over processes, rather than duplicated. Data intensive tasks such as tracing of non-coherent rays and shading (including texture mapping) should be performed data parallel, while coherent tasks may be allocated to processes according to a scheduling algorithm which takes the load of all processes into account.

The algorithm is split into a traditional master-slave setup. The master process will steer the computation by keeping track of the workload of the slave processes and will thus determine which process will receive the next demand driven task. All slave processes (one per processor) are capable of handling all tasks that are generated in the course of the computation (i.e. a mix of demand driven and data parallel tasks). The demand driven tasks will be handled with lower priority than the data parallel tasks, so that the load is effectively balanced by the demand driven tasks.

As not all demand driven tasks will be generated by this master process, each process will be periodically informed about the workload of other processes. If a slave process generates a demand driven task, it can thus determine where to send the task. These demand driven tasks will be either self-contained, i.e. the data to complete such a task is sent along with the task itself, or the required data is fetched from a remote processor. In the latter case, caching techniques need to be employed to reduce object communication. Both variants should be implemented and evaluated.

The local illumination model consists of three components: specular reflection, diffuse reflection and light source sampling. The cost of these components depends on the accuracy of the rendering and on the number of light sources in the model. If the accuracy required is very high, the diffuse reflection component may dominate the computation. This part of the computation is also not equally spread over the data parallel processes, as visible objects will attract more computations than objects that are not visible from the view point. While coherence is high for specular reflection and light rays, this may not be the case for diffuse reflection rays. Unfortunately, the diffuse sampling constitutes the bigger part of the total work. Therefore, splitting off primary rays and light rays as demand driven tasks (and maybe even specular reflection rays) as such, will probably not be sufficient to properly balance the load. The computational load of the demand driven part of the algorithm may still not match the load imbalance of the data parallel network.

As in this scheme all tasks that can be performed demand driven, are performed demand driven, a better matching between data parallel and demand driven components may be achieved by a reduction

of the (communication) cost of the data parallel network or by a better load distribution for the data parallel network.

3 Data distribution and data structure

The unequal load distribution for the data parallel processing may be corrected by another distribution of objects over the processors. As hierarchical spatial subdivision techniques, such as an octree or a bintree, show a degree of adaptability, we may employ this to improve the object distribution over processors. In very dense areas of the model, the cells become smaller, so that the cost of intersecting a ray with a cell, is approximately the same for each cell.

In order to distribute an octree over a number of processors, the following strategy may be chosen. The total number of objects N in the spatial subdivision structure is divided by the number of processors P. The problem to be solved now, is to find exactly P (or an integer multiple of P) branches in the bintree (or octree) that have approximately $\frac{N}{P}$ objects each, see figure 2 for an example.

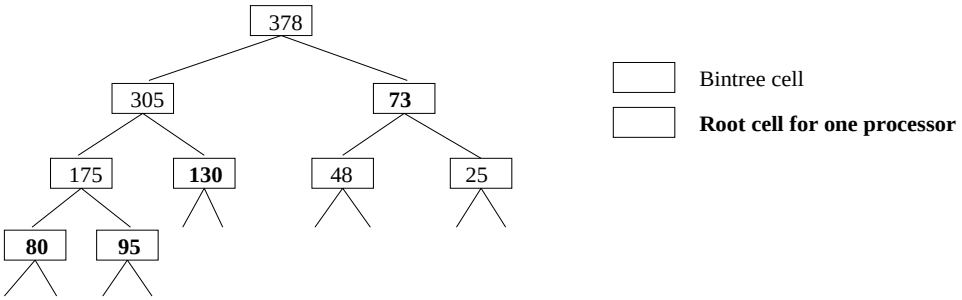

Figure 2: Example of how a bintree may be distributed over four processors.

The advantages of this data distribution are the adaptability to local scene complexity, resulting in approximately equal numbers of objects stored with each processor. It is also relatively easy to determine neighbouring processors, which is useful should ray tasks need to be communicated to these neighbours.

A consequence of regular spatial subdivisions such as the bintree and the octree, is that objects can be arbitrarily intersected by cell boundaries. This is especially true for small cells containing relatively large objects. An alternative distribution that avoids this problem may be achieved by analysing the model and explicitly indicating clusters of objects. In that case, clusters of objects will not be arbitrarily intersected by voxel boundaries, but the determination of neighbouring processes will become slightly more awkward. Groups may also spatially overlap, which may introduce additional problems.

The grouping operation should be performed as a pre-processing step, so that a model once clustered, can be rendered repeatedly. Initially, groups of objects should be specified by the user, as currently the user is the best judge of what objects belong to each other. Methods to automate the grouping process should be investigated. It should be possible to find simple criteria for deciding which objects belong in a group, such as distance and size of objects in relation to other objects.

For both the bintree/octree approach as the grouping approach, some additional duplication of voxels may reduce the impact of possible hotspots. If a large number of processors is used and the model to be rendered would fit easily in the memory available, then the parts of the scene that get referenced most often, may be duplicated a few times to distribute the load more evenly. A number

of processes then have a ghost process which may take over any excessive load. This should be most effective for rescheduling data parallel tasks, as demand driven tasks are processor independent anyhow.

Alternatively, in a fine grain approach, each voxel may be enlarged, so that overlap between voxels exists. Data parallel tasks could be rescheduled with neighbouring processors should one processor become a hotspot. The advantages are the same as with spatial subdivision duplication, with the additional advantage that data locality is better preserved. Rescheduling a task involves only local task communication.

Finally, there are opportunities to distribute objects according to some cost prediction algorithm (Reinhard, Kok & Jansen 1996). Such an algorithm may incorporate information regarding light sources, eye point and object densities. It should also include the details of the rendering algorithm which is chosen. The initial object distribution would be affected, for example, by the capability to reassign a data parallel task to another processor.

4 Conclusions

A parallel rendering algorithm that should be capable of rendering complex scenes with advanced lighting requirements such as diffuse inter-reflection, needs to be capable of splitting off both demand driven as data parallel tasks. Such a hybrid approach caters for extreme storage requirements coupled with efficient rendering. The overal idea is to execute as much work as possible as demand driven tasks and the remainder of the tasks should be handled data parallel. Ray tasks will be classified according to the amount of coherence between them. In practice this will mean that primary rays, shadow rays and specular reflection rays will constitute the set of demand driven ray tasks. A scheduling mechanism will be implemented that gives each processor an overview of the relative workload of the processors, so that tasks can be efficiently scheduled by any processor.

Diffuse reflection rays will be handled as data parallel tasks, as per intersection point, half the objects in the scene may be sampled. The contribution to the final result of these rays is too small to warant data communication of half the scene. Scheduling of tasks that did not generate a local intersection, is based on the data distribution over the processors.

For the data parallel tasks, initially a data distribution based on the octree spatial subdivision as present in sequential Radiance, will be achieved. The data requirements of the demand driven tasks will be satisfied on a flexible basis. Objects necessary for the completion of a demand driven task will be retrieved from the processors that store these objects. Caching strategies will be implemented to reduce object communication for demand driven tasks.

As a refinement to the above algorithm, the performance of the data parallel part of the algorithm may be improved by augmenting the octree spatial subdivision by a grouping approach. Grouping of primitives into objects is initially performed by the user, but may be replaced by an automated scheme that identifies groups in a pre-processing step.

References

Reinhard, E. & Jansen, F. W. (1995), Hybrid scheduling for efficient ray tracing of complex images, *in* M. Chen, P. Townsend & J. A. Vince, eds, ‘High Performance Computing for Computer

Graphics and Visualisation', University of Swansea, Springer, London, pp. 78–87.

Reinhard, E., Kok, A. J. F. & Jansen, F. W. (1996), Cost prediction in ray tracing, *in* X. Pueyo & P. Schroeder, eds, 'Rendering Techniques '96', Eurographics, Springer Wien, Porto, pp. 41–50.

Ward, G. J. (1994), The RADIANCE lighting simulation and rendering system, *in* A. Glassner, ed., 'Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)', Computer Graphics Proceedings, Annual Conference Series, ACM SIGGRAPH, ACM Press, pp. 459–472.