A Spatial Representation for Ray-Scene Intersection Test Improvement in Complex Scenes

J. Revelles, N. Aguilera, J. Aguado, M. Lastra, R. García, R. Montes

Dpt. Lenguajes y Sistemas Informáticos, E.T.S.I. Informática, University of Granada, Spain
e-mail: [jrevelle,mlastral,ruben,rosana]@ugr.es
URL: http://giig.ugr.es

Abstract
We present a spatial representation based on a hierarchical structure using the well-known spatial indexing structure called octree. There are very useful spatial representations for scenes whose objects can be distributed in clusters and here we present a new one. In order to prove its benefits, several results are shown in scenes using a photom tracing algorithm to compute the global illumination based on photon map. These results show that the hierarchy of octrees becomes a good choice to improve the performance when compare to other strategies such as octrees and hierarchy of uniform grids[3].

Keywords: Acceleration techniques, ray casting, spatial indexing methods.

1. Introduction

Most of the rendering algorithms require a long time for the ray-scene intersection test process. In order to improve the performance of this process, that is, to minimize the time necessary to compute the ray-scene intersection test, several techniques are available. These techniques are based on different spatial indexing strategies on the scene domain.

Several spatial indexing techniques have been proposed as efficient strategies such as octrees[2, 5, 4, 11], 3D grids[1, 2], and BSPtrees[9, 13, 8]. These spatial representations are useful for different scenes, i.e. objects with several sizes and with a non-homogeneous distribution. Each space index needs a process which simulates a ray traversal of this structure. Havran[7] presented a good work comparing several spatial representations and their traversal algorithm.

These spatial representations are proposed to accelerate the ray-scene intersection test in a rendering system when a ray-casting based method is used. However, when the scene is very complex (a great amount of objects) both memory usage and rendering time could be too high. Another problem for the user is that several parameters must be set not only for the rendering process but for the acceleration technique applied as well.

Due to the reasons mentioned above, a rendering system must contain several efficient techniques to accelerate the ray-scene intersection test. Moreover, the parameters required to build an specific spatial representation for a given scene must be easy to set by the user. For this reason, in most of cases a previous analysis for a given complex scene is required.

Section 2 presents the state of the art of several spatial representations which satisfy the above requirements. In section 3 a new spatial representation is proposed. In order to show the performance of the acceleration technique, several scenes have been rendered and analysed. Comparative timetable with the results are shown in section 4. Finally, the conclusions and future works are presented.

2. Previous Work

There are two main aspects when a scene is processed using a rendering system:

- Set the necessary parameters in order to get a realistic image. This step only depends on the rendering algorithm used and is outside the scope of this work.
- For a given complex scene, choose an acceleration technique and the parameters required in order to increase its performance. Some work has been done for setting the values of the parameters of the hierarchical spatial representations[6, 12].

When considering complex scenes, a relevant contribution was made by Cazals et al.[3]. In that paper the authors proposed a new strategy for managing complex scenes in a rendering system which increased the performance of the ray-scene intersection test process. They presented a spatial representation based on a regular grid called hierarchy of uniform grids (*HUG*).
The main characteristic of the strategy was the distribution of objects in groups called clusters. In a first step, a classification by size of the input objects is done. Next, a clustering step is applied to objects which are the same size, and finally a hierarchy of regular grids is built for each cluster.
The spatial representation which is explained in section 3 is based on the same philosophy of the work proposed by Cazals. The main difference between Cazals' method and ours is that an octree is built for each cluster instead of a regular grid.

3. The Hierarchy of Octrees as Spatial Representation

We introduce a spatial representation which can be seen as a hierarchy of octrees (*HOO*). Using the same advantages of the clustering process proposed by Cazals, the only difference is in the third step, where an octree is applied for each cluster.
The process for constructing a hierarchy of octrees is described as follows:

1. The clustering algorithm[3] is used in a first step.
2. For each cluster, an octree is constructed.
3. Group two neighbour clusters into a bounding box (this process continues until a balanced binary tree representing the whole scene is obtained).

An example of a scene where a spatial representation based on a *HOO* may be applied is shown in figure 1. In this scene seven clusters will be obtained. For each cluster, an octree is built allocating all scene objects in this cluster (see figure 2).

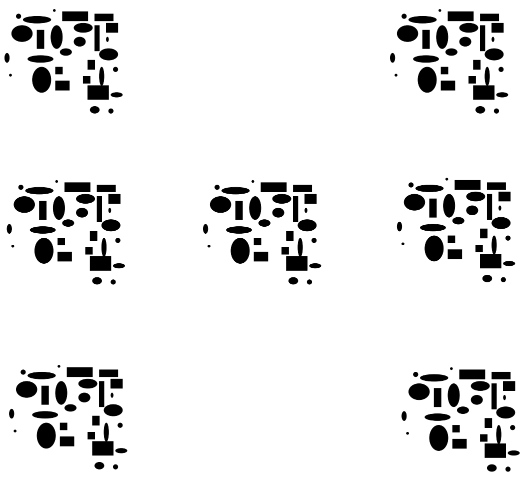
Figure 1: *Scene grouped in 7 clusters.*

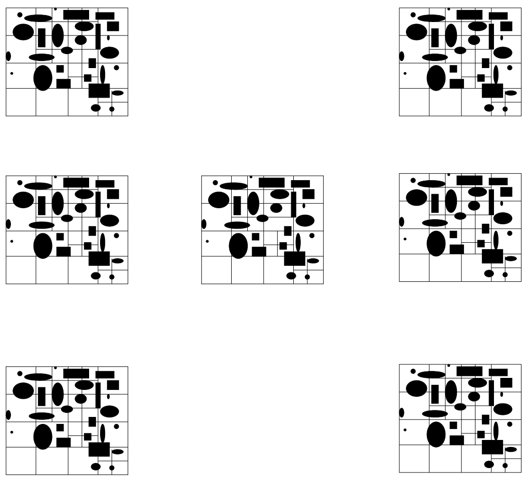
Figure 2: *Octrees applied for each cluster.*

In some cases, a scene can not be divided into clusters because the clustering algorithm does not distinguish the existing clusters (in most cases, the whole scene would be matched as an only cluster). In this situation, other considerations must be taken into account to use these spatial representations.
When the scene is feasible to be split into clusters, the following advantages are gained using a *HOO*:

– The spatial representation requires less memory than a simple spatial representation for the whole scene (i.e. using a regular grid or an octree).
– As the space between clusters increases, so does the performance difference between an octree and a *HOO*
– Finally, a *HOO* provides better results than a *HUG* as is shown in section 4.

The main disadvantage of a *HOO* and a *HUG* is the required time of the clustering process. However, the clustering algorithm has a linear increase in relation to the number of objects.

4. Results

In order to show the performance of the proposed spatial representation, two comparisons and scenes have been deployed:

– Comparisons between *HOO* and octree (scene shown in figure 3). This scene is composed of two clusters. One of them is shown in figure 4).
– Comparisons between *HOO* and *HUG*.

All the results were obtained using a Pentium 4 processor at 1.7 GHz with 1GB of RAM running Linux 2.4.20. The testbed for the different spatial representations was a photon tracing system. The resulting rays are used in a second stage by the density estimation method described in [10]. The time considered is only related to the photon tracing process. In scene 3, there are two clusters with a large empty space between them. Each cluster consists of an array of $8 \times 8 \times 8$ cubes with an extended light source over it. Each cube has 12 triangles (which is the maximum number of objects allocated in a leaf node for an octree). The scene objects are triangles. In order to build an octree for one cluster considering 12 objects per leaf node, the maximum depth level is 3. Nonetheless, when both clusters are considered to construct a unified octree, the maximum depth level would become 10 in this case. Therefore, the memory requirements for the unified octree are larger than for a *HOO* (see table 1). The speedup of *HOO* is due to the time necessary for traversing the internal nodes in the octrees until a leaf node is reached, which is reduced because of the lower depth. In this case, it is very easy to determine the maximum depth level for both spatial representations. When this value is hard to set, a good way to compare the performance is using similar memory amounts in both spatial representations. Therefore, a spatial representation based on a *HOO* provides better results than an octree. The performance is higher when the clusters are further apart.

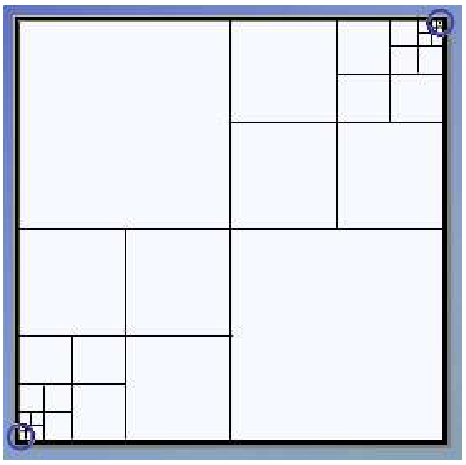
Figure 3: *Scene with two clusters and a large space between them.*

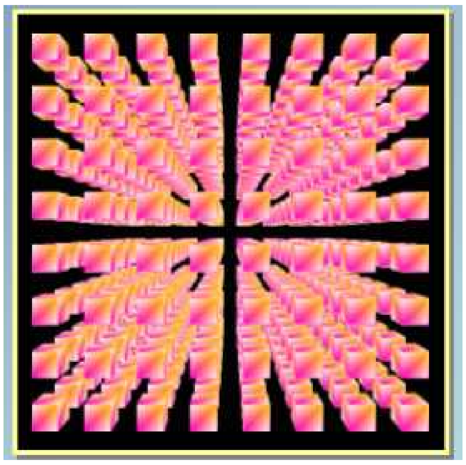
Figure 4: *Cluster of the above scene described as an array of* $8 \times 8 \times 8$ *cubes.*

Octree		***HOO***
10	*Depth level*	3
1147	*Illumination Time (in seconds)*	139
	Gain percentage	87%

Table 1: *Octree vs. Hierarchy of Octrees.*

In order to compare the *HUG* and *HOO* structures, a scene with 10 clusters is proposed (see figure 5). This scene has 50000 triangles including the floor. All the triangles have a similar size. The number of detected clusters is 10. The photon tracing process generated 1000000 primary photons. The results are shown in table 2.

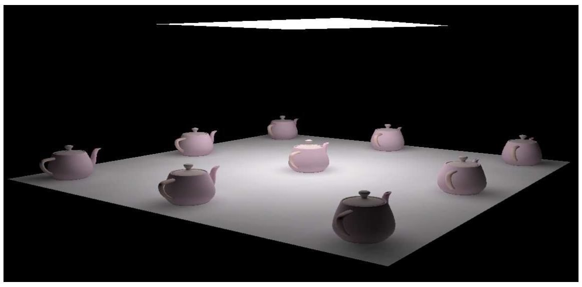
Figure 5: *Scene with 10 clusters (9 teapots and the floor).*

Octree depth level and subdivisions in the regular grid are set to get spatial representations with similar memory usage in both cases. Under these circumstances, the hierarchy based on an octrees performs better than the one based on a regular grid.

The reduction in execution time ranges between 21% and 39% according to the times obtained in table 2.

5. Conclusions and Future Effort

In this paper we have introduced a spatial representation for complex scenes. It is based on the same principles proposed in a previous work[3] but using octrees instead of regular grids. So, this work has similar advantages and disadvantages than *HUG*.

The main contribution of this spatial indexing consists of

HOO **depth level**	**Time**	**%**	**Time**	***HUG*** **Subdivisions**
4	**64.55**	21%	**81.23**	13
5	**52.19**	35%	**80.10**	21
6	**48.73**	39%	**80.26**	29
7	**48.74**	39%	**80.35**	37
8	**49.35**	38%	**79.34**	45
9	**49.97**	38%	**80.51**	53

Table 2: *Hierarchy of Octrees vs. Hierarchy of Regular Grids.*

lower memory requirements and better results in terms of execution time for the photon tracing process. Good results were obtained for cluster oriented scenes, specially when the clusters are far apart.
As future work, we will study the benefits of this spatial representation for more complex scenes and for different cluster distributions. In addition, we will study the importance of the space between clusters.

Acknowledgments
Special thanks to Carlos Ureña and Luis Miguel Vílchez for their contributions to this work. This work has been supported by a grant coded as TIC2001-2932-C03-03 (Spanish Commission for Science and Technology).

References

1. J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *EUROGRAPHICS'87*, pages 3–10, Amsterdam, 1987.
2. J. Arvo and D. Kirk. *A Survey of Acceleration Techniques*, chapter chapter 6. An Introduction to Ray Tracing, pages 201–262. Academic Press, San Diego, 1989.
3. F. Cazals, G. Drettakis, and C. Puech. Filtering, clustering and hierarchy construction: A new solution for ray-tracing complex scenes. In *EUROGRAPHICS'95*, pages 371–382, 1995.
4. R. Endl and M. Sommer. Classification of ray-generators in uniform subdivisions and octrees for ray tracing. *Computer Graphics Forum*, 13(1):3–20, 1994.
5. A.S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics & Applications*, 4(10):15–22, 1984.
6. J. Goldsmith and J. Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics & Applications*, 7(5):14–20, 1987.
7. V. Havran. *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, 2001.
8. V. Havran and J. Bittner. Rectilinear bsp trees for preferred ray sets. In *Proceedings of 14th Spring Conference on Computer Graphics, Budmerice in Slovakia*, pages 171–179, 1999.
9. V. Havran, T. Kopal, J. Bittner, and J. Zara. Fast robust bsp tree traversal algorithm for ray tracing. *Journal of Graphics Tools*, 2(4):15–24, 1997.
10. M. Lastra, C. Ureña, J. Revelles, and R. Montes. A particle-path based method for monte carlo density estimation. In *Eurographics Workshop on Rendering (short paper) 2002, Pisa (Italy).*, 2002.
11. J. Revelles, C. Ureña, and M. Lastra. An efficient parametric algorithm for octree traversal. *Journal of WSCG (Copyright UNION Agency-Science Press)*, 8(2):212–219, 2000.
12. K.R. Subramanian and D.S. Fussell. Automatic termination criteria for ray tracing hierarchies. In *Proceedings of Graphics Interface'91*, pages 93–100, June 1991.
13. K. Sung. *Ray Tracing with the BSP Tree*, pages 271–274. Academic Press, 1992.