

Multi-Level Hashed Grid Construction Methods

Vasco Costa
INESC-ID / IST
Lisboa, Portugal
vasc@vimmi.inesc-id.pt

João Pereira
INESC-ID / IST
Lisboa, Portugal
jap@vimmi.inesc-id.pt

Joaquim Jorge
INESC-ID / IST
Lisboa, Portugal
jaj@vimmi.inesc-id.pt

ABSTRACT

Ray tracing is an inherently parallel visualization algorithm. However to achieve good performance, at interactive frame rates, an acceleration structure to decrease the number of per ray primitive intersections is required. Grid acceleration structures have some of the fastest build times, with $O(N)$ complexity, but traditionally achieved this at a high memory cost. Recent research has reduced the memory footprint by employing compression for one-level grids. Render time performance can be improved using multi-level grids. We describe two methods for building such multi-level grids. In the first method we employ a recursive compressed grid in which grid cells are adaptively subdivided in a variable fashion. The second method uses a finely divided compressed grid, with a lower resolution macrocell overlay to speed up traversal. We analyze the performance of these new algorithms, which enable improved render times, versus existing solutions.

Keywords

Ray tracing, spatial subdivision, grid.

1. INTRODUCTION

Realtime ray tracing is an active area of research [Wal07]. Even traditionally skeptical hardware vendors have recently demonstrated, or made available, realtime ray tracing solutions [Sei08]. Ray tracing is desirable for several reasons, namely per pixel accurate shadows, reflections and refractions. It can also be used as a base for other global illumination algorithms such as path tracing, and photon mapping, to add more effects such as caustics and diffuse interreflections.

In the naive ray tracing algorithm, it is necessary to search the nearest intersected primitive for each ray. Without an acceleration structure, the complexity for such an algorithm is $O(N)$, where N is the number of primitives in the scene. Hence to enable realtime ray tracing for complex scenes, with many primitives, acceleration structures are used. These acceleration structures can theoretically reduce per ray complexity to $O(\log N)$.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Ideally an acceleration structure should be fast to build and use as little memory space as possible, while still delivering good render time performance.

This work describes our efforts to combine the desirable traits of multi-level grid [Jev89,Wal06] render time performance, with the low build time and memory consumption characteristics of row displacement compression [Lag08].

Existing related work in this area is surveyed in Section 2. Section 3 describes the proposed multi-level grid construction methods. The performance results of these methods are analyzed in Section 4. Finally conclusions are presented in Section 5.

2. RELATED WORK

Grid acceleration structures for ray tracing were first described by Fujimoto et al. [Fuj89]. These acceleration structures subdivide 3D space in near cubical cells. It was found that grids, by eliminating vertical traversal time costs present in other acceleration structures popular at the time, had increased overall render time performance. 3DDA, a 3D extension of the raster line drawing algorithm, was employed for ray grid traversal.

An improved grid traversal algorithm was later near simultaneously devised by several researchers [Woo87,Cle88]. This algorithm is still employed today. The historical grid ray tracing acceleration structures around this period are described by Havran et al. [Hav99]. Grid dimensions ($M_x \times M_y \times M_z$) are determined based on heuristics related to the number

of scene primitives, scene bounding box, and certain constant factors.

Recently Lagae and Dutré [Lag08] employed grid row displacement compression (i.e. hashing) to reduce the memory footprint of this kind of acceleration structure. It does this by compressing empty cells. By allocating all memory, before inserting primitives into the data structure, build time performance was also improved. The render time performance of this one-level grid algorithm is however inferior to non-compressed multi-level algorithms, such as the rgrid used by the Manta ray tracer [Big06], as shall be seen in Section 4.

Kim et al. [Kim09] have created compressed versions of the bounding volume hierarchy (BVH) acceleration structure, one of the acceleration structures first used in ray tracing. Kim et al. also compress the triangle mesh and page data to the disk providing increased memory savings.

BVH acceleration structures have higher construction time complexity than grids. BVH construction complexity is $O(N \log N)$ versus a grid construction complexity of $O(N)$.

More recent, faster to build, grid acceleration structures have many advantages. However further work is necessary to improve their render time performance. This work aims at filling this gap.

3. METHODS

The classification of multi-level grid construction methods employed here is based on that of Jevans and Wyvill [Jev89].

Variable construction methods recursively subdivide the grid, by employing subgrids in each cell. Subgrid dimensions are chosen using a similar heuristic to that employed for the first cell division level. Memory consumption is hard to predict, usually leading to the use of dynamic memory allocation along the construction method.

Fixed construction methods use a fixed ratio, finer subdivision than a regular one-level grid would employ. Since the total size of a grid acceleration structure can be known in advance, all memory allocation can be done before the method is employed. A fixed construction grid can be build using macrocells for the lower resolution levels.

Fixed construction methods have good performance for uniformly distributed scenes, such as laser scanned models. Variable construction methods adapt more easily to varying scene primitive distribution but at increased memory consumption and build time costs.

The following heuristic, attributed to Woo, is employed to determine grid dimensions:

$$M_i = \frac{S_i}{\max\{S_i\}} \sqrt[3]{\rho N} \quad (i \in \{x, y, z\})$$

Equation 1. Woo's heuristic. S_i is the scene bounding box size in dimension i , ρ is 4.

Via profiling we noticed some characteristics in the existing algorithms [Lag08, Big06] described at Section 2. Grid traversal dominates render time, and one-level grids spend a lot more time doing ray/triangle intersections than multi-level grids. In attempting to improve render-time performance we posed the following hypothesis: we can reduce the number of ray/triangle intersections by using smaller cells, with fewer triangles per cell. To reduce traversal time we can employ a multi-level structure to skip empty cells in larger steps.

3.1. Multi-Level Variable Hashed Grid

This subsection describes the multi-level variable hashed grid implementation. It is a recursive grid, with the top level grid and subgrids using the hashed grid [Lag08] algorithm. This grid has a maximum grid depth size of 2.

First the top level hashed grid is built using the algorithm described by Lagae et al. [Lag08] but using the heuristic from Equation 1. We selected a grid density ρ of 4 since it empirically provided good render time performance. Each cell of this top level grid is then subdivided using the same algorithm, creating a new subgrid, for each cell containing more than a certain number of primitives.

3.2. Multi-Level Fixed Hashed Grid

In this subsection a multi-level fixed hashed grid is described. It is a high resolution hashed grid [Lag08] with multi-level macrocells [Wal06] to speedup traversal.

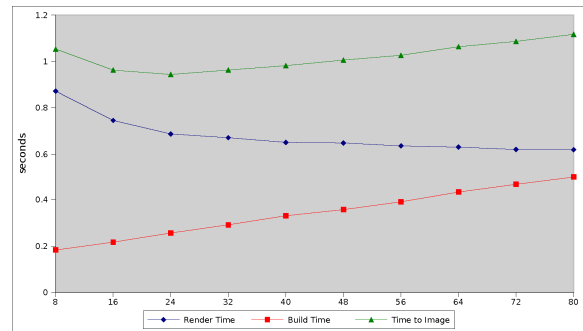


Figure 1. Timings for the Buddha scene according to grid density.

First a finely divided one-level hashed grid is built in a similar fashion to that of Lagae et al. [Lag08], but using the grid heuristic described in Equation 1 with a high grid density parameter to reduce cell size.






					
	Bunny	Dragon	Buddha	Asian Dragon	Thai Statue
Scene statistics					
# triangles	69.45K	871.41K	1.09 M	7.22 M	10 M
memory	1.2MB	15.0MB	18.7MB	123.9MB	171.7MB
Manta recursive grid [Big06]					
Primitive intersections/ray	1.58	1.58	1.56	0.91	1.17
Cell traversals/ray	4.73	5.80	4.95	6.44	7.00
Grid traversals/ray	1.38	1.28	1.17	0.72	0.82
Build Time (s)	0.47	3.46	4.50	20.44	29.59
Render Time (s)	0.30	0.52	0.34	0.36	0.58
Time to Image (s)	0.78	3.98	4.84	20.80	30.17
One-level hashed grid [Lag08]					
Primitive intersections/ray	8.35	9.87	9.53	13.15	12.67
Cell traversals/ray	14.53	35.23	26.93	93.14	100.76
Grid traversals/ray	0.00	0.00	0.00	0.00	0.00
Build Time (s)	0.02	0.22	0.26	1.48	2.07
Render Time (s)	0.58	0.89	0.78	1.60	1.80
Time to Image (s)	0.60	1.11	1.04	3.09	3.88
Multi-level variable hashed grid					
Primitive intersections/ray	3.99	3.83	3.92	1.93	2.63
Cell traversals/ray	15.12	26.05	17.21	68.31	69.38
Grid traversals/ray	0.54	0.53	0.53	0.27	0.36
Build Time (s)	0.09	0.75	0.81	4.09	6.29
Render Time (s)	0.51	0.64	0.55	1.00	1.11
Time to Image (s)	0.60	1.39	1.36	5.10	7.39
Multi-level fixed hashed grid					
Primitive intersections/ray	6.14	8.26	10.06	8.74	9.06
Cell traversals/ray	14.04	17.86	13.10	29.97	27.31
Grid traversals/ray	0.57	0.47	0.45	0.24	0.25
Build Time (s)	0.04	0.39	0.29	3.09	3.45
Render Time (s)	0.57	0.68	0.67	0.79	0.82
Time to Image (s)	0.61	1.07	0.97	3.88	4.27

Table 1. Scene triangle mesh statistics, render time profile results, timings for the studied grid acceleration structures.

We empirically chose the grid density parameter by analyzing the behavior for the Buddha scene as can be seen in Figure 1. We selected a grid density ρ of 32 since it features adequate render time without having a severe impact on time to image.

Next multi-level macrocells [Wal06], are built to skip empty cells in larger steps during traversal. Macrocells overlay a coarser grid over the finely divided grid. The macrocells for each level consist of a 3D bit array with information if a region of space is empty or not. To speed up this construction step macrocells are downscaled by a factor S of 6 on each extent. We arrived at this value by empirically analyzing algorithm behavior for the tested scenes. Wald et al. [Wal06] reached the same value with a

different heuristic and test scenes. Macrocell downscaling can be done with a quick 3D bitmap scaling operation.

4. PERFORMANCE AND RESULTS

This section evaluates the performance of the grid construction methods.

All tests were performed on a single Intel Core 2 Duo processor at 3 GHz. The machine has 4GB of RAM running the Linux operating system. The algorithms were implemented in C++ using STL and Boost without use of assembly or intrinsics.

Only a single thread was used, with one ray per pixel and diffuse shading, at 1024×1024 resolution. A

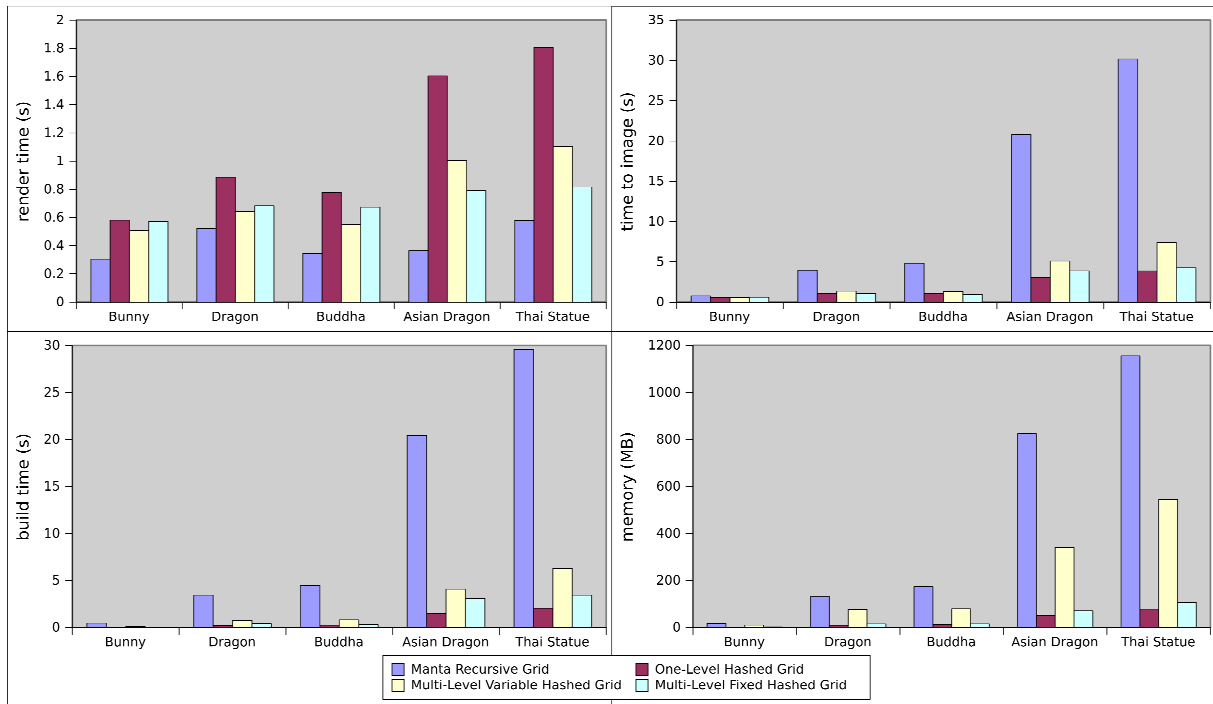


Figure 2. From bottom right clockwise: memory consumption; build time; render time; time to image acceleration structure statistics for the tested scenes.

variety of models from the Stanford 3D Scanning Repository were used for the evaluation.

The top of Table 1 shows scene statistics such as number of triangles, memory used by the triangles. These scenes were chosen because the system is expected to support visualization of laser scanned architectural models. Scene memory usage is computed by using 12 bytes per triangle to store vertex index information (three machine words for each vertex index), plus 12 bytes per vertex (three floating point numbers for each coordinate). This provides reduced memory usage in an expedient fashion. Ray/triangle intersection was done using the Möller-Trumbore [Mol05] intersection algorithm because of its low memory requirements.

For performance comparison purposes with existing published algorithms the recursive grid from the Manta interactive ray tracer [Big06] was tested. An implementation of the hashed grid algorithm by Lagae and Dutré [Lag08] was added to the system to serve as the one-level compressed grid baseline.

The multi-level hashed grid structures feature improved render time performance compared to the one-level hashed grid. This is markedly so for the larger scenes where over twice the render time performance is achieved. Of the multi-level hashed grid methods, the fixed hashed grid is better for the larger scenes, as can be seen at top left in Figure 2. Fixed grid features improved render times, versus the variable grid, due to several factors: the fixed grid has a smaller memory footprint (and increased

memory coherence); the cells of the top hierarchical level of the fixed grid have a larger volume, skipping empty regions of space faster, this is reflected in the cell traversals/ray.

The recursive grid from Manta has even better render time performance, although the performance difference varies according to the tested scene.

These performance results required a more in depth examination by profiling the acceleration structures in terms of number of primitive intersections, horizontal cell traversals and vertical grid traversals.

Profiling, seen in Table 1, shows improved Manta render time performance is due to the lower number of ray/primitive intersections and horizontal cell traversals used by the recursive grid to display the same scene.

Manta employs a deeper variable grid structure with maximum depth of 3 and has a modified heuristic. This enables improved render time performance but comes at a big build time penalty. It takes six times longer to build the acceleration structure for the Thai Statue scene for example as can be seen at the bottom left of Figure 2.

Memory usage paints a similar picture to the build time statistics. The Thai Statue scene uses around ten times more memory in the non-compressed Manta multi-level acceleration structure versus the fastest compressed multi-level acceleration structure we implemented.

The compressed multi-level grid acceleration methods of note feature much improved performance on the figures of merit. Time to first image in particular is much improved versus the times achieved by Manta using algorithms of the same class. The multi-level fixed hashed grid has a similarly low time to image compared to the one-level hashed grid. This makes it the best option among the multi-level grids for the tested scenes.

5. CONCLUSION

Multi-level compressed grid methods achieve best of class performance by combining the desirable traits from existing algorithms: low memory requirements, fast build and render times. The algorithms presented here could still use some work in the heuristics, as the multi-level heuristic from Manta has quicker render times. There is also room for expansion in improving the number of cell traversals and primitive intersections per ray. Alternative methods for speeding up traversal time by skipping empty voxels, not studied in this work, include proximity clouds [Coh94], macro-regions [Dev89], and similar directional techniques [Sem97].

We would also like to implement these algorithms on GPUs to investigate the performance characteristics of compressed structures on that hardware class.

6. ACKNOWLEDGEMENTS

It would not have been possible to make the tests in this work without the models from the Stanford 3D Scanning Repository.

This work was supported by the Portuguese Foundation for Science and Technology project VIZIR (PTDC/EIA/66655/2006).

7. REFERENCES

- [Big06] J. Bigler, A. Stephens and S. G. Parker. Design for Parallel Interactive Ray Tracing Systems Proceedings of the IEEE Symposium on Interactive Ray Tracing, 2006.
- [Coh94] D. Cohen, and Z. Sheffer. Proximity clouds - an acceleration technique for 3D grid traversal. *The Visual Computer*, 11(1): 27–38, 1994.
- [Cle88] J. Cleary and G. Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer*, 4(2):65–83, 1988.
- [Dev89] O. Devillers. The macro-regions: an efficient space subdivision structure for ray tracing. In *Eurographics '89*, pages 27–38, 1989.
- [Fuj89] A. Fujimoto, T. Tanaka, and K. Iwata. Arts: Accelerated ray-tracing system. *Computer Graphics and Applications*, IEEE, 6(4):16–26, 1986.
- [Jev89] D. Jevans and B. Wyvill. Adaptive voxel subdivision for ray tracing. In *Graphics Interface '89*, pages 164–172, June 1989.
- [Hav99] V. Havran, F. Sixta, and S. Databases. Comparison of hierarchical grids. *Ray Tracing News*, 12(1):1–4, 1999.
- [Lag08] A. Lagae and P. Dutré. Compact, fast and robust grids for ray tracing. *Computer Graphics Forum (Proceedings of the 19th Eurographics Symposium on Rendering)*, 27(8), 2008.
- [Kim09] Tae-Joon Kim, Bochang Moon, Duksu Kim, Sung-Eui Yoon. RACBVHs: Random-Accessible Compressed Bounding Volume Hierarchies. *IEEE Transactions on Visualization and Computer Graphics*, 17 Jun. 2009.
- [Mol05] T. Möller and B. Trumbore. Fast, minimum storage ray/triangle intersection. In *International Conference on Computer Graphics and Interactive Techniques*. ACM Press New York, NY, USA, 2005.
- [Sei08] L. Seiler, D. Carmean, E. Sprangle, T. Forsyth, M. Abrash, P. Dubey, S. Junkins, A. Lake, J. Sugeran, R. Cavin, R. Espasa, E. Grochowski, T. Juan, and P. Hanrahan. Larrabee: a many-core x86 architecture for visual computing. *ACM SIGGRAPH*, 2008.
- [Sem97] S.K. Semwal, and H. Kvanstrom. Directed Safe Zones and the Dual Extent Algorithms for Efficient Grid Traversal during Ray Tracing. In *Graphics Interface '97*, pages 76–87, May 1997.
- [Wal06] I. Wald, T. Ize, A. Kensler, A. Knoll, and S. Parker. Ray tracing animated scenes using coherent grid traversal. In *International Conference on Computer Graphics and Interactive Techniques*, pages 485–493. ACM Press New York, NY, USA, 2006.
- [Wal07] I. Wald, W. Mark, J. Gunther, S. Boulos, T. Ize, W. Hunt, S. Parker, P. Shirley. State of the art in ray tracing animated scenes *Eurographics 2007 State of the Art Reports*, 2007.
- [Woo87] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In *Eurographics '87*, pages 3–10, 1987.

