Grid Creation Strategies for Efficient Ray Tracing

Thiago Ize
SCI Institute, University of Utah
Intel Corp

Peter Shirley
School of Computing, University of Utah

Steven Parker
SCI Institute, University of Utah

ABSTRACT

Both theoretical analysis and practical experience have shown that when ray tracing a well-behaved model with N geometric primitives, the lowest ray tracing times using a grid acceleration structure occurs when the grid has $O(N)$ cells. This paper extends the theoretical analysis in two ways and then experimentally verifies that analysis for several geometric models. The first extension is to examine how model characteristics influence the choice of the number of cells in a grid, with models made of long thin primitives being of particular interest. For such models, the lowest trace times come when $O(N^{1.5})$ cells are used, but may not always be practical due to the super-linear memory usage. The second extension is to nested grids where a grid cell may itself contain another grid. For the case of scattered data such as exploding particles, nesting is not helpful. For the case of tesselated manifolds with compact triangles, $O(N^{0.6})$ cells at top level is optimal if only one level of nesting is allowed. For d levels of nesting, $O(N^{3/(3+2d)})$ is optimal for the top level. For long thin primitives, $O(N)$ cells at the top level is optimal when one level of nesting is allowed, but this again comes at the cost of super-linear memory usage.

1 INTRODUCTION

Acceleration structures are used in ray tracing to improve the time needed to compute ray-scene intersections at the expense of increased preprocessing time [1]. Popular methods include hierarchical spatial subdivision (e.g., octrees and k-d trees), hierarchical object subdivision (e.g. bounding volume hierarchies), and uniform spatial subdivision often called "grids". All of these methods can be mixed in the form of "meta-hierarchies" such as bounding volume hierarchies with grids or k-d trees in their leaves [9]. There is currently no agreement on what acceleration structure is best [12], and it is likely that the answer varies depending upon the details of the model.

In this paper we examine some properties of the grid acceleration structure [2, 6], as well the parameters that should be used when using nested grid structures [8, 10]. We examine different build strategies that are implied by whether a model is manifold-like, and whether a model is made mainly of long-skinny triangles. Our approach is to use theory with simplifying assumptions, and to then experimentally test whether these assumptions prevent the theoretical results from applying in practice. We take our inspiration from the fact that an analysis on uniformly random rays and infinitesimal points in 3D (Figure 1, left) yields useful grid construction guidelines despite our straightforward analysis. We extend such simple analysis to long-skinny primitives that we approximate with lines and to models whose primitives lie along surfaces (Figure 1).

The guidelines for grid construction we suggest are:

- for scattered data composed of N compact primitives, a single level grid with $O(N)$ grid cells has an asymptotically optimal tracing time of $O(N^{1/3})$;
- for scattered data composed of N long skinny primitives, a single level grid with $O(N^{3/2})$ grid cells has an asymptotically optimal trace time of $O(N^{1/2})$;
- for manifold-like models composed of N compact primitives, a two-level grid can reduce the time complexity of tracing to $O(N^{1/5})$ if $O(N^{3/5})$ grid cells are used for the top level;
- for manifold-like models composed of N compact primitives and d levels, $O(N^{3/(2d+1)})$ grid cells should be used for the top level, resulting in tracing asymptotic complexity of $O(N^{1/(2d+1)})$;
- for manifold-like models composed of long-skinny triangles, a two level grid should use $O(N)$ cells in the first level grid and results in a tracing complexity of $O(N^{1/3})$.

We believe that all but the first two observations are new. Cleary and Wyvill have a more complicated analysis for the first two observations that makes fewer simplyfing assumptions [2].

Previously, recursive grid dimensions were choosen such that each level subgrid would always create $O(N)$ cells, where N is the number of primitives in that specific subgrid [8]. This results in only $O(N^{1/3})$ time complexity, instead of the $O(N^{1/(2d+1)})$ presented here. We were not able to discover any preexisting theory that helped guide in the choosing of each parameter value, which made choosing the correct parameters challenging, especially since they could vary substantially with N and the subgrid level. For instance, a two level grid would often see an order of magnitude difference in parameter values between the two levels. Our strategy not only can give better performance, it is also much easier to use since only one parameter needs to be adjusted and this parameter does not vary with the scene.

2 CLASSIC RESULTS FOR ONE-LEVEL GRIDS

In this section we review the derivation of choosing the optimal grid resolution for simplified conditions. Details of this derivation can be found in classic works [2, 4, 8]. We then discuss how well the results of this simple analysis apply to several geometric models.

2.1 Theoretical analysis

The terms and symbols we use are summarized in Table 1. We now summarize the classic analysis for one-level grids. We begin with several simplifying assumptions:

- the N primitives are all points (infinitely small);
- all rays hit the root box from the outside, and all are equally likely;
- the root box is a cube;
- each atomic function time can be treated as a constant;
- mathematical operations (as opposed to memory effects etc.) dominate performance;
- the grid has $m^3 = M$ cells, and M can be treated as a continuous number so discrete math idiosyncrasies can be avoided.

IEEE/EG Symposium on Interactive Ray Tracing 2007
10 - 12 September, Ulm, Germany
978-1-4244-1629-5/07/$25.00 ©2007 IEEE

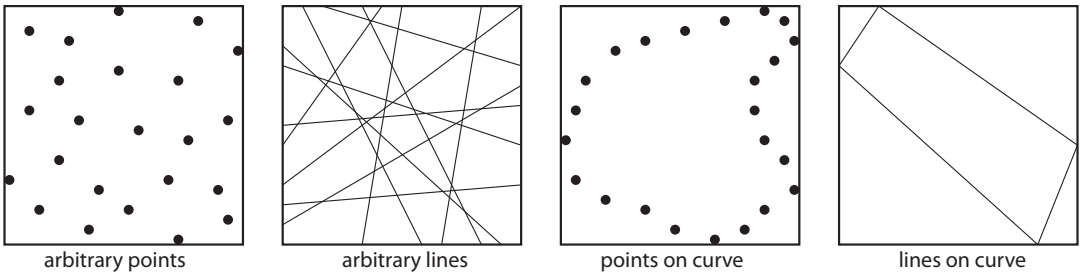

Figure 1: 2D versions of the scene types analyzed.

symbol	meaning
N	number of geometric primitives
M	number of grid cells
M_i	number of grid cells at level i ($i = 1$ is the top)
m	number of grid cells in one dimension: $m = \sqrt[3]{M}$
m_i	the cube root of M_i
k	a grid has km^2 occupied cells
T	average time to trace a single random ray
T_{setup}	time to do initial intersection with scene bounding box and setup traversal
T_{step}	time for ray to advance from cell to its neighbor
$T_{\text{intersection}}$	ray/primitive intersection time

Table 1: Symbols used in the paper

Geometric probability (or the closely related integral geometry) shows that exactly m cells on average are hit by a ray. Regardless of the distribution of the points, an average of N/m^3 points occupy a single cell, so a ray tests (and always misses) an average of N/m^2 infinitesimal points on its way through the grid. The average time T for a single ray to compute the fact that it misses all points is:

$$T = T_{\text{setup}} + mT_{\text{step}} + \frac{N}{m^2}T_{\text{intersection}} \tag{1}$$

By treating m as a continuous number we can take a derivative and minimize T:

$$\frac{dT}{dm} = T_{\text{step}} - \frac{2N}{m^3}T_{\text{intersection}} \tag{2}$$

Which when set to zero implies that T is minimized when:

$$m = \sqrt[3]{N\frac{2T_{\text{intersection}}}{T_{\text{step}}}} \tag{3}$$

2.2 Empirical test of analysis

The simplicity of these assumptions is cause for concern, but unfortunately the analysis is much more involved when these assumptions are relaxed [2, 4, 8, 13]. However, Equation 3 does seem to apply to at least some real models, and that happy fact is the inspiration for the simplified analyses we make later in this paper. We deviate from the results given by our theory by using grid cells that are as close to a cube as possible (to allow for non-cubical grid shapes), which in turn leads to the grid containing different numbers of divisions per dimension, as explained in [11]. Since m no longer represents the number of divisions, we use the computed value of M instead of m. The resulting value for M (from cubing the m in Equation 3) is:

$$M = N\frac{2T_{\text{intersection}}}{T_{\text{step}}} \tag{4}$$

The models in particular where the assumptions underlying the simple analysis are not too damaging are scanned models composed of relatively small triangles. In Figure 2 we have fit runtimes on several models to Equation 1 and have found the empirical behavior to be quite close to the theory with the caveat that the coefficients are slightly different for each model. The key ratio for optimization of runtime, $T_{\text{intersection}}/T_{\text{step}}$, is on average 4 for the models we have tested. This ratio and Equation 4 imply $M = 8N$. The relative performance of this M and the empirically determined optimal M for several models is shown in Table 2. As can be seen, for such scanned models, we only sacrifice a few percentage points in runtime by using our theoretical M for every model. Even the 282K triangle conference room that is even further from our assumptions closely follows our theory.

model	N	empirical M	theoretical M	penalty
bunny	948	29172	7584	4%
	69451	645028	555608	2%
buddha	15536	132848	124288	1%
	1087716	4873932	8701728	3%
conference	282664	1121514	2261312	5%

Table 2: The performance of $M = 8N$ for several models. Penalty is the percentage of extra time used by choosing the theoretical M versus the empirically determined optimal M.

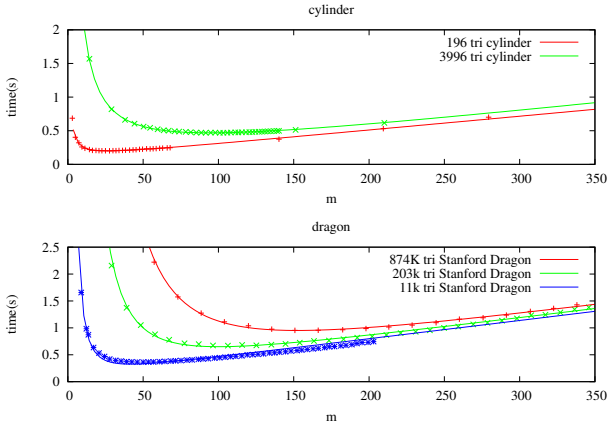

Figure 2: The fitted cost runtimes (Equations 1 and 5) match very closely with the actual runtimes.

3 Long-skinny triangles and one-level grids

If the model is made up of long skinny objects (as they would be in a tesselated cylinder) then each object will be added to more than $O(1)$ cells so the basic assumptions of the previous section do not apply. In practice, long skinny objects get cut into more and more effective objects as more grid cells are added, so the benefits of

model	N	empirical M	theoretical M	penalty
cylinder	196	16875	21952	0%
	3996	768337	2020822	3%

Table 3: The performance of $M = 8N^{1.5}$ for a tesellated cylinder. Penalty is the percentage of extra time used by choosing the theoretical M versus the emprically determined optimal M.

grids are diminished. Rather than using points to approximate small objects as in the last section, we use line segment to approximate long thin objects.

3.1 Theoretical analysis

In the same spirit of simplified analysis, we can assume each long-skinny object is actually a line segment spanning the entire model. This means as m increases, so does the total number of effective primitives in the model. Each line will intersect $O(m)$ cells. If all lines are equally likely each will intersect an average of m cells. This changes Equation 1 to have an extra factor of m in the numerator of the last term:

$$T = T_{\text{setup}} + mT_{\text{step}} + \frac{N}{m}T_{\text{intersection}} \tag{5}$$

and the resulting derivative:

$$\frac{dT}{dM} = T_{\text{step}} - \frac{N}{m^2}T_{\text{intersection}} \tag{6}$$

Which when set to zero implies that T is minimized when:

$$m = \sqrt{N\frac{T_{\text{intersection}}}{T_{\text{step}}}} \tag{7}$$

and the total number of grid cells:

$$M = \left(N\frac{T_{\text{intersection}}}{T_{\text{step}}}\right)^{1.5} \tag{8}$$

Unfortunately this implies memory use of $O(N^{1.5})$ but that suggests a tradeoff between runtime and memory utilization that could be made by the user. Again if we assume the ratio in the equation is 4, then we have $M = 8N^{1.5}$.

3.2 Empirical test of analysis

We have observed parts of models that have long skinny triangles. These are typically tesselated parts that themselves are long and skinny. To mimic such situations, we have used a tesselated cylinder. The cylinder consists of very long skinny triangles that span the entire length, with a triangle fan making up the caps. Figure 3 shows the 196 triangle version of the cylinder we used for testing.

Our theory suggests that $M = 8N^{1.5}$, and using that we get good results for all of our reasonably sized tests as shown in table 3. For the extremely large 19996 triangle cylinder, using 8 is noticeably inferior and 2 works best. However, since it is unlikely a $O(N^{1.5})$ space-complexity algorithm would be used for a model with that many skinny triangles, in practice using 8 as the constant will work very well.

Compared to using the $M = 8N$ metric that produces near-optimal single level grids for most models, using $M = 8N^{1.5}$ as suggested by our theory produces near-optimal grids that are 1.25× faster and 14× larger for the 196 triangle cylinder, and 1.7× faster and 63× larger for the 3996 triangle cylinder.

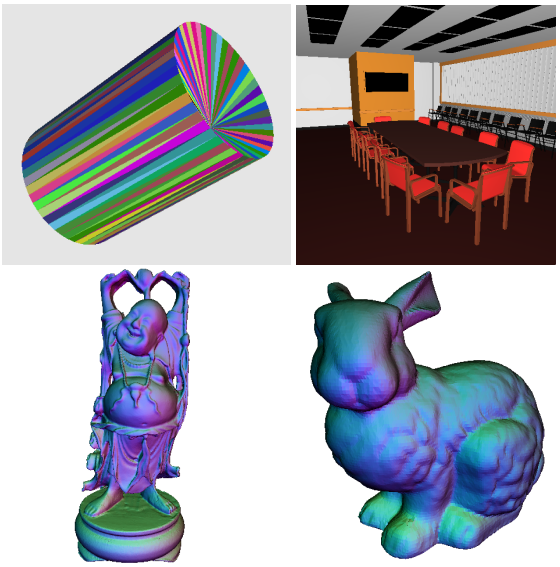

Figure 3: Four of the models we use in our tests. Depicted here are the 196 triangle cylinder used for modeling long skinny triangle scenes, the 282K triangle conference room, the 1M triangle Happy Buddha statue, and the 69K triangle Stanford Bunny.

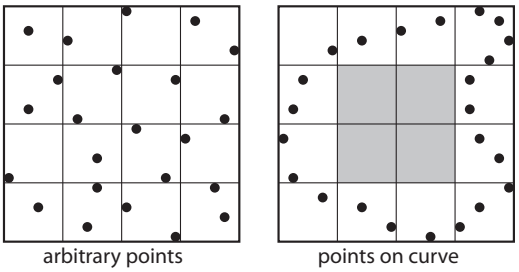

Figure 4: In scattered data most or all cells need to be refined, defeating the purpose. In a suface (curve in 2D) there are cells (shaded) that need not be refined.

4 Multi-level grids and points on surfaces

For scattered data having hierarchical grids does little good because most or all cells will be refined, while for points on surfaces there is more potential (Figure 4). There are a variety of ways to support a multi-level grid. Two simple options in 2D are shown in Figure 5. These two options trade-off geometric compactness for sharing setup costs between levels. For our analysis we adopt the method where the whole new cell is a grid as the analysis is easy due to predictable surface area of sub-grids. In implementation we adopt the geometrically compact method as in our experience it is usually faster, and it is simpler from a software engineering standpoint as black-box software can be used: a grid cell can simply point to an abstract object that happens to be another grid. In addition to the two options above there are other methods that use a single high resolution grid and use some method of skipping the empty spaces, such as proximity clouds [3] and macro-regions [5]. We do not try to analyse these alternatives.

4.1 Theoretical analysis for 2-level grids

When a surface sweeps through a grid it leaves many cells empty. For example, a single axis-aligned plane hits exactly one "sheet" of cells and thus occupies m_1^2 of the m_1^3 cells. A cube composed of six squares will occupy about $6m_1^2$ cells. We assume that for a

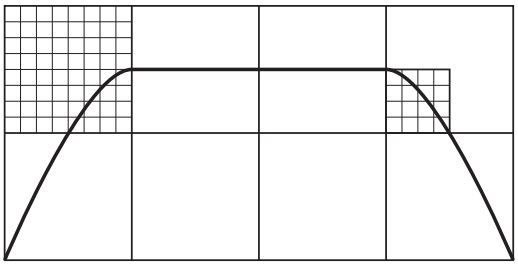

Figure 5: Two different ways to refine a cell.

given model, there will be some constant k such that the number of occupied cells is km_1^2. The rest of the m_1^3 cells are empty and would not be refined into subgrids.

A key assumption to make the analysis more tractable is that the points in occupied cells are evenly divided between those cells. Thus each of the occupied cells has $N/(km_1^2)$ points. This assumption is usually never realistic, but we hope that as in the single level grid analysis a simplified case that does not discard objects will still yield useful guidance.

Since we assume the occupied cells have the same number of points, each occupied cell is divided into m_2^3 level-2 cells. Since the probability of intersecting a point is 0, a ray will pass through the entire grid, touching on the order of m_1 level-1 cells, and since the fraction of level-1 cells with a subgrid is k/m_1, the average number of subgrids hit by a ray is k. Within a subgrid the analysis from Section 2 applies. Since each subgrid contains $N/(km_1^2)$ points, the average number of points in one of the cells in a level-2 grid is $N/(km_1^2m_2^3)$. For recursive grids, the setup time must be done each time a grid or subgrid is entered. Thus the average time is:

$$T = T_{\text{setup}} + m_1 T_{\text{step}} + k\left(T_{\text{setup}} + m_2 T_{\text{step}} + \frac{N}{km_1^2m_2^2} T_{\text{intersection}}\right) \tag{9}$$

If we take partial derivatives, we get

$$\frac{\partial T}{\partial m_1} = T_{\text{step}} - \frac{2N}{m_1^3m_2^2} T_{\text{intersection}} \tag{10}$$

and

$$\frac{\partial T}{\partial m_2} = kT_{\text{step}} - \frac{2N}{m_1^2m_2^3} T_{\text{intersection}} \tag{11}$$

Setting those both to zero and solving for m_1 yields:

$$m_1 = \sqrt[5]{N\frac{2k^2T_{\text{intersection}}}{T_{\text{step}}}} \tag{12}$$

Thus the optimal choice for the number of cells in the top level grid is:

$$M_1 = \left(N\frac{2k^2T_{\text{intersection}}}{T_{\text{step}}}\right)^{0.6} \tag{13}$$

The formula for M_2 differs only in constants, and ends up being the same answer as in Section 2 once you account for the fact that there are $N/(km_1^2)$ points total in the subgrid. This intuitively makes sense since we want to build an optimal single level grid for the points in the subgrid. Note that if you did the above build on scattered small points ($k = m_1$), you would get $O(N^{6/5})$ memory use and $O(N^{2/5})$ time, both of which are not ideal. The type of model matters.

model	N	empirical total cells	theoretical total cells	penalty
bunny	948	36016	17315	1%
	69451	1863727	747518	2%
buddha	15536	132848	215579	1%
	1087716	7308615	10350245	0%
conference	282664	5194622	2832557	1%

Table 4: The performance of $M_1 = (8N)^{3/5}$ and $M_2 = 8N_2$ for several models. Penalty is the percentage of extra time used by choosing the theoretical M_1 versus the empirically determined optimal M_1.

4.2 Empirical tests for 2-level grids

We first build a grid using the M_1 cells given from Equation 13. In each occupied cell we build an optimal single level grid with the number of cells for that subgrid determined by Equation 4, where N in this case corresponds to the number of primitives in that subgrid. We use the same values of $T_{\text{intersection}}$ and T_{step} as in Section 2. For k we found that setting it to 1 seemed to give the best results. This is likely due to our assumptions with using points, such as a ray going all the way through the grid and not hitting any objects, not carrying over completely when using primitives. This gives us $M_1 = (8N)^{.6}$ and $M_2 = 8N_2$, where N_2 is the number of triangles in the subgrid being created.

Using just the parameters already obtained for the single level grids (the ratio of $T_{\text{intersection}}$ to T_{step}) and setting $k = 1$, we are extremely successful in finding near-optimal 2-levels grids. In table 4 we see that for both tesselation extremes of the buddha and bunny models, we are within 2% of the optimal time and the number of cells used appears linear. More surprisingly, even the conference room is handled by our method extremely well.

Figure 6 shows how varying the number of cells used for each level affects render performance. Since we use $m_1 = (8N)^{1/5}$ and $m_2 = (8N_2)^{1/3}$, this corresponds to the point $(8^{1/5}, 8^{1/3}) = (1.5, 2)$ in the plots, which is usually very close to the optimal point.

Compared to an optimal single ray grid, we get close to a $2\times$ speedup in rendering time for the buddha model and a $2.4\times$ speedup for the conference room when we use the two level grid suggested by our theory.

4.3 Theoretical analysis for multi-level grids

Applying the logic as above, we define k_1 to be the occupancy constant for the first level grid, and k_2 for the second. Thus for a three level grid, the cost is:

$$T = T_{\text{setup}} + m_1 T_{\text{step}} + k_1(T_{\text{setup}} + m_2 T_{\text{step}}) + k_1k_2\left(T_{\text{setup}} + m_3 T_{\text{step}} + \frac{N}{k_1k_2m_1^2m_2^2m_3^2} T_{\text{intersection}}\right) \tag{14}$$

Optimizing this shows that $m_1 = O(\sqrt[7]{N})$, and $m_2 = O(\sqrt[5]{N/m_1^2})$. More precisely,

$$M_1 = \left(2Nk_1^4k_2^2\frac{T_{\text{intersection}}}{T_{\text{step}}}\right)^{\frac{3}{7}}. \tag{15}$$

This suggests the general rule that for L levels in a grid, the optimal subdivision for the top level is:

$$M = O\left(N^{\frac{3}{2L+1}}\right). \tag{16}$$

This formula can be repeatedly applied for the number of objects in each subgrid. For example, for two levels, we would use $M_1 = O(N^{3/5})$ and then M_2 would be linear in the number of objects in that particular subgrid.

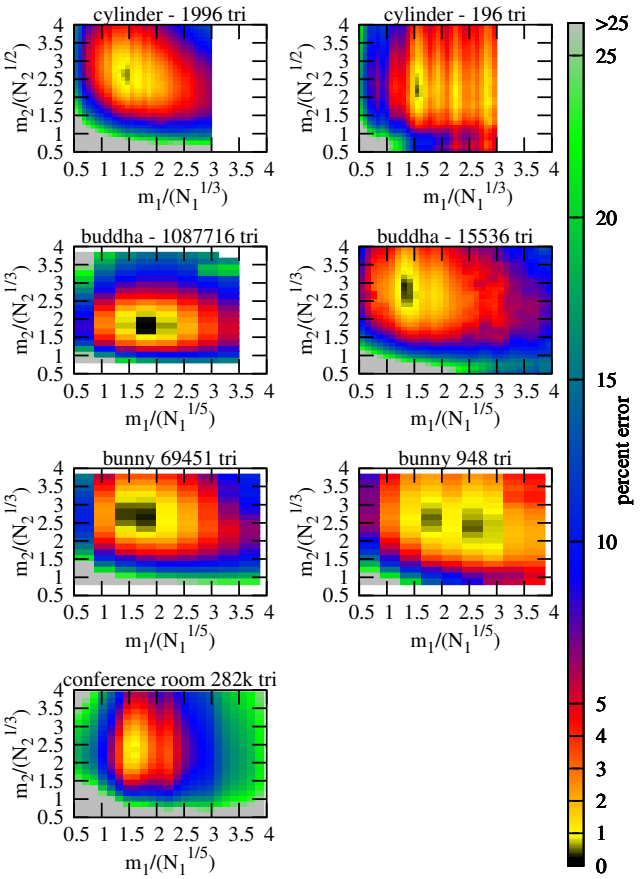

Figure 6: Percent error in rendering times found through an exhaustive search across the $2D$ parameter space for creating 2-level grids. Moving across the x-axis changes m_1 and the y-axis corresponds to m_2. Our theory predicts the point $(1.6, 2)$ for the cylinders and $(1.5, 2)$ for the other models as being optimal.

4.4 Two Level Grids and long-skinny triangles

As with one level grids, long, skinny objects deviate from the analysis above because they occupy a number of cells that depend on m. As before, if the primitives are on a surface, it is possible that two-level grids can improve runtime. The terms change slightly but the same basic techniques apply, yielding:

$$M_1 = kN\frac{T_{\text{intersection}}}{T_{\text{step}}} \tag{17}$$

and M_2, which is identical to M in Section 3. The associated space is:

$$\text{space} = O(N^{\frac{5}{3}}) \tag{18}$$

Figure 6 shows that our theory once again predicts to a very high accuracy what the optimal grid resolutions are. Compared to using the 2-level grid formula for regular triangles, we get a 1.6× and a 1.2× rendering speed increase for the 1996 and respectively, 196 triangle cylinders.

5 Conclusion and Discussion

Clearly our analysis makes many assumptions, some of them possibly unrealistic. But it does highlight two important things:

1. For well-behaved models, using nested grids can lower time complexity below $O(\sqrt[3]{N})$, but this comes at the cost of increased constant overhead, so the number of levels depends both on the model and on the setup cost of entering a grid. Models with more primitives, or multi-level grid implementations with low setup costs, such as with macro cells, should be able to take advantage of more levels.

2. Models with long skinny triangles can use grids for sub-linear time, but only if super-linear memory is used. If linear memory is used, then the intersection time is linear but with a reduced time constant.

The first observation above gives some explanation as to why grids can sometimes be competitive with tree-based techniques despite their worse time complexity. The difference between $N^{1/5}$ and $\log N$ requires fairly large N or large constants to become obvious.

The second observation illustrates how difficult long-skinny polygons are for ray tracing in general. In fact, kd-trees and axis-aligned BVHs experience even more severe difficulties with such models [12]. It also points out that simplifying tesselated models for ray tracing is problematic—in areas of high curvature along one tangent axis and low along the other, simplified models will have long skinny triangles that will effectively be re-tesselated by the grid. That all suggests oriented acceleration structures should be investigated more thoroughly.

While we don't recommend using the optimal M for large N, for small N the super-linear storage cost can be better justified. Furthermore, these results can be used in picking a middle ground where performance is close to optimal, but storage use is still reasonable. This is important because many architectural or CAD-like scenes contain parts made up of long skinny triangles. In a multi-level grid it is likely that some subgrids would consist of mainly long skinny triangles where our results could be used.

Using mailboxing with long skinny triangles will not allow us to use $O(N)$ grid cells since mailboxing only prevents redundant intersection tests from occurring; however our problem is not redundant intersections, but intersecting a ray against a very large number of triangles that exist in a single cell.

Our results for determining the optimal grid dimensions for single and multi-level grids are very good for all the models we tested, which is especially surprising when we look at the simplifications we had to make. For instance, assuming the primitives are points and so have zero probability of being hit by a ray is clearly false. This could have an impact on our results since we assume a ray will always traverse through the grid and through k subgrids, when in reality, for a manifold-like object the ray will on average hit a primitive (and thus stop traversing) after only traversing through a fraction of the grid and will usually only need to enter one or two subgrids. This would suggest that the cost to trace a ray can be, on average, significantly lower. On the other hand, since rays are hitting primitives, our assumption that the number of primitive intersections is N/m^2, which converges to 0 in the limit, is too low since at least one primitive must be hit, which raises the total cost. Finally, even just changing the camera location can affect the optimal grid resolution. In the end we expect that most of these differences are just folded into our constants, average each other out, are insignificant, or in the case of k, become 1 and disappear. Since other unknown terms are being folded into the constants we use, this means that our constants no longer specifically represent what they started as. This means that plugging in the actual time to intersect a triangle into $T_{\text{intersection}}$ would likely lead to an incorrect solution and so these parameters should be experimentally determined.

While our theory is corroborated very well by our experiments, it is not perfect. Interestingly enough, while experimenting with single level grids we empirically found that using $M = O(N^{7/9})$ for manifold-like models with compact triangles and $M = O(N^{4/3})$ for manifold-like models with long skinny triangles produced essentially perfect results for choosing M. We hypothesize that these are

actually the correct formulas to use when dealing with real primitives and not points and lines. Using these results for single level grids or the bottom level of a grid might produce even better results both in time and space costs, especially when N becomes extremely large. Proving this hypothesis would be interesting future work.

In this paper we use an unoptimized single ray grid even though a faster grid traversal scheme exists that makes use of SIMD instructions and ray coherence [11]. We do so for two reasons. First, our analysis is based on recursive grids for which more interesting results can be proven, such as the storage complexity staying linear as the number of levels increases. Wald et al.'s coherent grid traversal algorithm, on the other hand, used macro cells that do not exhibit linear storage complexity. Secondly, a single ray recursive grid is more robust than a coherent grid traverser and for scenes with incoherent rays will perform better (the coherent grid traverser would need to revert to single ray) and so the analysis still applies. Combining our results with a coherent grid traversal algorithm could produce some interesting results.

Build times have now become very important in interactive ray tracing [12]. We show that grids with compact triangles take linear space, which means building a recursive grid can be done in linear time. While not previously mentioned in our paper, our current build times are an order of magnitude slower than those reported by Ize et al. [7]. In addition to our build being unoptimized, it is also slower because they report *rebuild* time, meaning that memory has already been allocated and an initial grid built. We are confident that rebuild times of nested grids could become fast enough for handling dynamic scenes.

REFERENCES

[1] J. Arvo and D. Kirk. A survey of ray tracing acceleration techniques. In A. S. Glassner, editor, *An Introduction to Ray Tracing*. Academic Press, San Diego, CA, 1989.

[2] J. G. Cleary and G. Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer*, 4(2):65–83, 1988.

[3] D. Cohen and Z. Sheffer. Proximity clouds–an acceleration technique for 3d grid traversal. *The Visual Computer*, 11(1):27–38, 1994.

[4] O. Devillers. *Méthodes d'optimisation du tracé de rayons*. PhD thesis, Université de Paris-sud, 1988.

[5] O. Devillers. The macro-regions: an efficient space subdivision structure for ray tracing. In *Proceedings of Eurographics '89*, pages 27–38. Elsevier Science Publishers, September 1989.

[6] A. Fujimoto, T. Tanaka, and K. Iwata. ARTS: Accelerated ray tracing system. *IEEE CG&A*, 6(4):16–26, 1986.

[7] T. Ize, I. Wald, C. Robertson, and S. G. Parker. An evaluation of parallel grid construction for ray tracing dynamic scenes. In *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pages 47–55, 2006.

[8] D. Jevans and B. Wyvill. Adaptive voxel subdivision for ray tracing. *Proceedings of Graphics Interface '89*, pages 164–172, June 1989.

[9] D. Kirk and J. Arvo. The ray tracing kernel. In *Proceedings of Ausgraph*, pages 75–82, 1988.

[10] K. S. Klimaszewski and T. W. Sederberg. Faster ray tracing using adaptive grids. *IEEE CG&A*, 17(1):42–51, Jan./Feb. 1997.

[11] I. Wald, T. Ize, A. Kensler, A. Knoll, and S. G. Parker. Ray tracing animated scenes using coherent grid traversal. *ACM Transactions on Graphics*, 25(3):485–493, 2006. (Proceedings of ACM SIGGRAPH).

[12] I. Wald, W. R. Mark, J. Günther, S. Boulos, T. Ize, W. Hunt, S. G. Parker, and P. Shirley. State of the art in ray tracing animated scenes. In *State of the Art Reports, Eurographics 2007*, 2007.

[13] B. Walter and P. Shirley. Cost analysis of a monte-carlo radiosity algorithm. Technical Report PCG-95-3, Program of Computer Graphics, Cornell University, 1995.