Hybrid Regular Grid - KD-tree for Ray tracing acceleration.

Didier Muñoz Díaz
Instituto Tecnológico Autónomo de México

***Abstract*—The Ray-Tracing technique is fundamental in most of current rendering systems for movies and animation. This technique, along with light-transport realistic models, is capable of generating photorealistic images that accomplish in faithfully imitating real images through the rendering of 3D scenes descriptions. Recent attempts on improving rendered images quality are focusing on bringing Ray-Trancing to the interactive realm (e.g. videogames). The biggest issue associated to this technique is the high computational cost related to searching ray-scene intersections. This work aims to achieve an improvement in ray traversal time through a new algorithm that mixes the adaptability of kd-trees and the traversal efficiency of uniform grids.**

I. Introduction

Ray tracing is a computer graphics technique for generating digital images[12, p. 4]. To determine the color for each pixel in an image, this algorithm recursively trace the traversal path along the scene[1] for each light ray starting with the rays reaching the observer (See Fig. 1). In this algorithm, 3 phases can be distinguished:

- *Acceleration structure construction.* To accelerate the ray-scene intersection search, several data structures have been proposed. The two most prominent being *kd-trees* and *Bounding Volume Hierarchies*.
- *Ray-Intersection search.* Once the acceleration structure has been constructed, for each ray generated, a traversal path needs to be followed inside the acceleration structure until the desired intersection between the ray and the scene is found (a hit) or the ray leaves the scene.
- *Lighting calculations.* With the intersection found, the next step is to calculate the color based on the material properties and the scene lighting. New rays can be cast starting from this point as the result of this calculation requirements.

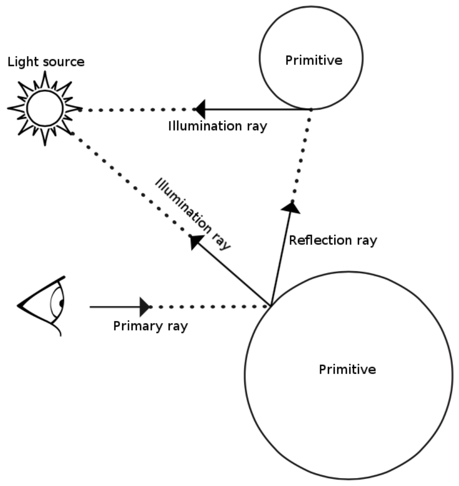

Fig. 1. Diagram of the basic ray tracing technique.

This work focuses on the first two phases as they are closely related and, depending on the scene, can account for the largest part of the rendering process. The intersection search asymptotic complexity is calculated through the number of pixels to be rendered (an image of $width = w$ columns and $height = h$ rows would have $m = w \times h$ number of pixels) and the number of primitives[2] n in the scene. In the brute force approach, each ray is tested against all the primitives in the scene. This means that if there are at least m initial rays to render the image and n primitives in the scene, the asymptotic complexity would be $O(mn)$. When an acceleration structure, like kd-trees, is involved,

[1]In the computer graphics context a *scene* is a mathematical description of virtual objects.

[2]In this context a *primitive* is a mathematical description of an object, e.g. a sphere or a triangle.

the intersection search complexity is reduced to $O(m \log n)$[13, p. 20, 12, p. 4].

Due to its high complexity, Ray tracing has been exclusively considered for static-scenes offline rendering, but new attempts are trying to bring this technique to interactive frame-rates[6]. Even though, we are still far from achieving the needed frame-rates for most interactive applications (videogames), one of the most prominent solutions is related to the most recent Nvidia *Turing* GPUs architecture: Turing GPUs include specialized hardware (reffered to as RTX) for Ray-Tracing, able to trace reflections and direct shadows with interactive frame-rates[10].

This work organization is as follows:

Section II contains the main objectives of this work. Section III is about this work's scope defining its initial constraints. Section IV contains the justification of this research. Section V introduces the background and base algorithms upon this work is built on. Section VI presents an overview of the proposed algorithm. Section VII describes the methodology. Finally, section VIII is about future work to be done in this line of research.

II. Objective

The objective of this work is to propose a new hybrid acceleration structure and traversal algorithm which reduces the ray traversal complexity and increases the general rendering performance. This is done by taking the adaptability of kd-trees and the lower traversal complexity of uniform grids.

III. Scope

For the present work, the scope is limited by:

- *The type of primitives supported*. Even though ray tracing scenes can be made-up by numerous different primitives, this work focuses on triangles[3], which are the most common ones.
- *Acceleration structure selection*. Although other hierarchical structures (like BVH) are suitable for the proposed algorithm, the present work only focuses on kd-trees.
- *Architecture* The implementation is aimed to the x86 architecture.

[3]Numerous scenes are described or approximated by meshes composed by triangles. Also, in general, the operations involved in triangle intersection search have their analogous with other primitives making it easy to incorporate them to the ray tracer.

IV. Justification

The relevance of this work lies in the fact that currently generating high quality photorealistic images is computationally expensive. The rendering times of images used in current movies has been reported to be up to 30-40 hours long per frame [21] running on rendering farms with more than a thousand of servers. This rendering time is highly expensive[4] and it is dominated by rays-scene intersection searches.

V. Background

Ray tracing by computer dates back to 1968 [2], when the computing problem was first defined and its restrictions were established. A new algorithm was introduced for the shading of linear drawings, with the objective of generating images of objects delimited by plane surfaces and evaluating the cost of generating such images. The main issue was the needed computational time.

With the implementation of the recursive visible surface algorithm in 1980 [19], ray tracing began to receive more attention. As a result, new realistic images could be created at the expense of more computational time needs, depending on the complexity of the scene[6, 12].

One of the biggest problems to solve was the ray-intersection search complexity (of $O(nm)$ as mentioned in Section I). A more efficient way to find the intersections was needed. Ray-intersection approaches can be classified into: KD-tree, Bounding Volume Hierarchies and uniform grids.

a) KD-trees: K dimensional trees (kd-trees) were introduced in 1975 [3]. This data structure hierarchically organizes the points in an euclidean k-dimensional space, by successively subdividing the space through hyperplanes perpendicular to one of the coordinate system axes. As a result, a binary tree associated to a binary space partition is created. All the nodes store a value: root and intermediate nodes store the information of the partition and leaf nodes store the primitives wanted to be hierarchically organized. Fig. 2 shows an example of a 2 dimensional kd-tree.

For each space subdivision, a point along an axis is chosen to be the place to introduce the division

[4]Current price for 1000 instances rent with 32 cores each on Google Cloud[8] is of more than $USD800,000$.

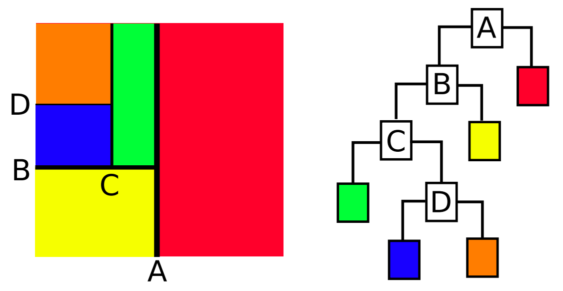

Fig. 2. A 2 dimensional kd-tree showing the binary tree associated to the hierarchical binary space subdivision.

plane. Many methods for choosing the division point have been proposed. Probably the most used one, known to produce really good partitions, is the Surface Area Heuristic (SAH) [6]. The SAH has been the standard algorithm for choosing splitting planes in the construction of kd-trees and other acceleration structures. Initially constructing a kd-tree was achieved in $O(n \log^2 n)$ or even $O(n^2)$, but in [17] an algorithm is presented that improves the complexity to $O(n \log n)$.

The Surface Area Heuristic kd-tree construction is based on the following assumptions:

- The cost of traversal is known: t_{trav}.
- The primitive intersection test is known: t_i.
- A current node C is intended to be split in two nodes A and B.

The fundamental idea is that for a convex volume A contained in a convex volume C, the conditional probability that a uniformly distributed random ray passing through C will also pass through A is the ratio of their surface areas S_A and S_C:

$$p_A = \frac{S_A}{S_C}$$

Also, the cost of testing intersections against all the k primitives in a node A is:

$$T_A = \sum_{i=1}^{k} t_i$$

With all this specified, the SAH defines the cost function of splitting $c(A, B)$ or not splitting a node $c(C)$ as:

$$c(A, B) = t_{trav} + (p_A \cdot T_A) + (p_B \cdot T_B)$$
$$c(C) = T_C$$

Using this method for constructing a tree means implementing a greedy algorithm: each time a node is to be split, $c(A, B)$ is calculated for several splitting planes and the minimum between all of them is chosen. If $c(C)$ is actually lower, then the splitting process is stopped.

In [5, 16, 9] other methods not SAH based are shown which improve on specific aspects:

- The solution proposed in [5] tries to address the problems involved with the assumptions made by the SAH: rays distribute uniformly, rays arrive from outside of the node, and rays do not get blocked by objects during traversal. For this they propose a new cost metric (to substitute SAH) based on a new *visibility voxel* concept, alongside with a 2 kd-tree scheme where each tree specializes in an especific type of primitives: static or dynamic.
- The approach taken by [16] is to approximate the triangles visibility by taking advantage of images coherence and along with it, a cost function that processes differently visible triangles and invisible triangles. This approach constructs BVH instead of kd-trees.
- On the other hand, [9] presents a kd-tree construction algorithm, which uses a new cost metric based on sampling ray distribution in the scene to avoid assuming that rays distribute uniformly.

Also, memory optimizations have been attempted which can help when the scene descriptions are big ($>$ 1000000 triangles), see [4].

With kd-trees when searching for the ray intersection a ray-tree traverse algorithm needs to be chosen. Traversing a kd-tree is usually done in a Depth-First pre-order fashion, but taking into account the order in which the ray intersects child nodes.

b) Bounding Volumes Hierarchies: Another very popular hierarchical acceleration structures are Bounding Volume Hierarchies (BVH). The core idea is to generate a tree where each node represents a bounding volume (i.e. a box), child nodes are bounding volumes completely contained in their parent bounding volume, and leaf nodes represent bounding volumes that contain one and only one primitive, see Figure 3.

When comparing to kd-trees, each one has its mayor drawback:

- KD-trees can present the situation where a

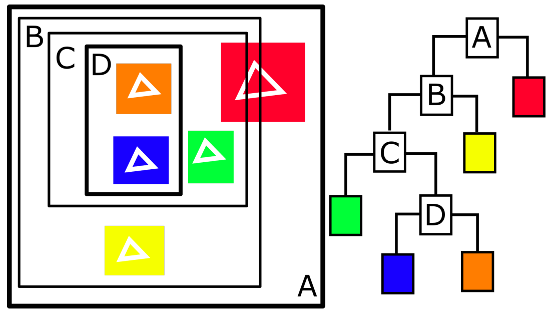

Fig. 3. A BVH showing how each bounding volume completely contains child bounding volumes.

single primitive appears on several sections of the hierarchy because it intersects them. This provokes that copies or references of the primitive must be kept in several nodes. In BVHs primitives have their own leaf bounding volume, so there is no need for duplicate primitives or several references.

- BVHs can present the situation where several bounding volumes intersect each other. This can greatly degrade the ray-traversal step if the scene has several primitives close to each other. KD-trees, by construction, does not have this problem because the space subdivision generated is in fact a partition.

One of the most relevant works to compare with, is the one in [11]. They also propose a hybrid data structure. An overview of their construction algorithm is as follows:

1) Construction of a coarse regular grid.
2) Subdivision of cells with primitives in an Oct-tree fashion.
3) Merging of the cells using SAH to achieve BVH characteristics.
4) Expanding cells to improve empty space coverage.

c) Uniform Grids: This approach was proposed in 1985 [7], as a new method with different strenghts and weaknesses. This method has the advantage of faster data structure creation and faster ray traversal than hierarchical tree-based structures, such as kd-trees and oct-trees[5].

Uniform grids are based on the space division through a simple and regular mesh. The main

[5]An *Oct-tree* is a special case of kd-trees where the space partitioning is done in the middle of the region to be subdivided.

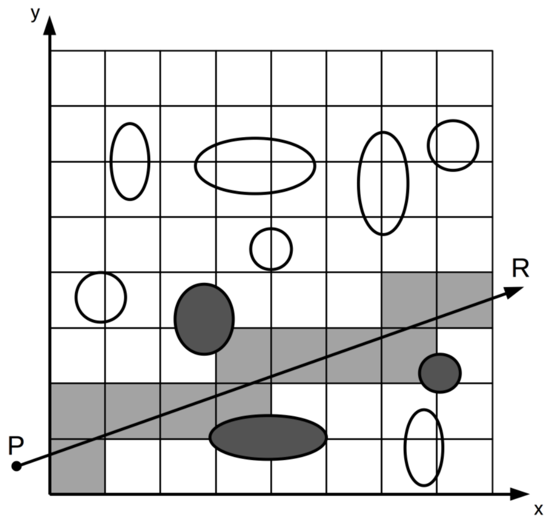

Fig. 4. A uniform grid showing the space subdivision and ray traversal.

problem is choosing the mesh size, because correctly choosing this parameter can significantly improve or decrease ray tracing performance. Fig. 4 illustrates this method, where:

- R is the ray and P is the origin point.
- The scene consists of 10 objects, represented by ellipses.
- The shaded ellipses are the only ones that are tested for intersection, thus avoiding calculations for the other seven.
- The shaded cells are visited and tested against.

The main drawback in uniform grids is the overhead caused by exhaustively testing every cell, empty or not, in the ray path. Therefore, for efficiently traversing a uniform grid, several techniques have been proposed, see [20, p. 19, 1]. The baseline approach is “based on the incremental algorithm for line drawing on 2D raster grid”. However one of the most common techniques, combines a compact storage of the scene light information with for-loop index-operations based cells traversal. Moreover, attempts to improve efficiency through detection of empty cells have been made, see [14].

VI. Hybrid Acceleration Data Structure

Some attempts to merge different accelerating structures have been made. Usually the approach taken has been to apply different accelerating structures to different groups of primitives based on a general hierarchy [18]. In this work the approach is different, we propose a new algorithm as a modified version of a kd-tree that intends on taking the best of both kd-trees and uniform grids:

- Make use of the fact that kd-trees cover the space completely and without overlappings (they form a partition).
- Use the simplicity of uniform grids ray traversal.
- Benefit from the adaptability of kd-trees.
- Take advantage on the simplifications made to the SAH method.

The proposed algorithm description is as follows:

1) First, a uniform grid is constructed. For this a basic cell size needs to be chosen. A good 'naive' guess could be the minimum cell size that can fit most of the primitives. This cell size defines the granularity to which the SAH tests will be made, so if constructing time were important (i.e. for interactive rendering), this granularity could be adjusted to imply fewer tests.
2) Using the defined basic cell size, a complete-scene bounding volume (a box) of size a multiple of the basic cell is defined as the root of the kd-tree.
3) A kd-tree is constructed in the usual way with the added restriction of having the subdivision planes on locations aligned to an integer multiple of the basic cell size.
4) The generated kd-tree can easily be translated to a 3 dimensional matrix. Each cell needs only to store its size and position.
5) For the ray traversal, only the primary rays would need to search the entire structure for the starting point. If still present, the kd-tree could be used for this purpose allowing a depth-first pre-order search.
6) Once the starting point is found, following steps are done in a loop-index fashion (similar to the uniform grids way).

VII. Methodology

The proposed steps to develop a new hybrid acceleration structure are:

- Develop a Ray Tracer implementing a minimum pipeline. This means:
    1) Loading scenes descriptions.
    2) Constructing a basic acceleration structure allowing for operations count and time measurement.
    3) Searching rays-scene intersections allowing for rays processed per second measurement.
    4) Implementing light calculations and recursive rays generation (reflections).
    5) Generating the final image.
- Implement the state of the art algorithms to which this solution will compare (if they are not provided by the authors).
- Design the hybrid kd-tree - uniform grid structure focusing on the reported and detected ray-traversal deficiencies of the related works.
- Implement the designed acceleration structure.

The validation is done by:

- Rendering standard scenes (as the ones in the Stanford 3D Repository [15].)
- Counting the number of rays traversed per second during the rendering process and comparing them with the state of the art algorithms.
- Counting the number of triangles per second processed, and comparing them to the related work versions.
- Compare the memory usage for each scene rendering against the related work versions.

VIII. Future Work

- As GPU Ray Tracing implementations are becoming more common[6] the next step is to look forward for a GPU-based implementation.
- Also, as many-Core implementations are currently the best in performance[13], depending on the specific architecture, such an implementation for the proposed solution in this work should be straightforward.

References

[1] John Amanatides and Andrew Woo. "A Fast Voxel Traversal Algorithm for Ray Tracing". In: *Proceedings of EuroGraphics*. 1987, p. 87.

[2] Arthur Appel. “Some Techniques for Shading Machine Renderings of Solids”. In: *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*. AFIPS ’68 (Spring). Atlantic City, New Jersey: ACM, 1968, pp. 37–45. DOI: 10.1145/1468075.1468082. URL: http://doi.acm.org/10.1145/1468075.1468082.

[3] Jon Louis Bentley. “Multidimensional Binary Search Trees Used for Associative Searching”. In: *Commun. ACM* 18.9 (Sept. 1975), pp. 509–517. ISSN: 0001-0782. DOI: 10.1145/361002.361007. URL: http://doi.acm.org/10.1145/361002.361007.

[4] B. Choi, B. Chang, and I. Ihm. “Improving Memory Space Efficiency of Kd-tree for Real-time Ray Tracing”. In: *Computer Graphics Forum* 32.7 (2013), pp. 335–344. DOI: 10.1111/cgf.12241. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12241. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12241.

[5] Byeongjun Choi, Byungjoon Chang, and Insung Ihm. “Construction of efficient kd-trees for static scenes using voxel-visibility heuristic”. In: *Computers And Graphics* 36.1 (2012). Cultural Heritage, pp. 38–48. ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2011.11.007. URL: http://www.sciencedirect.com/science/article/pii/S0097849311001671.

[6] Yangdong Deng et al. “Toward Real-Time Ray Tracing: A Survey on Hardware Acceleration and Microarchitecture Techniques”. In: *ACM Comput. Surv.* 50.4 (Aug. 2017), 58:1–58:41. ISSN: 0360-0300. DOI: 10.1145/3104067. URL: http://doi.acm.org/10.1145/3104067.

[7] Akira Fujimoto and Kansei Iwata. “Accelerated Ray Tracing”. In: *Computer Graphics*. Ed. by Tosiyasu L. Kunii. Tokyo: Springer Japan, 1985, pp. 41–65. ISBN: 978-4-431-68030-7.

[8] *Google Cloud Platform Pricing Calculator*. Nov. 2018. URL: https://cloud.google.com/products/calculator/.

[9] Xiao Liang et al. “Efficient kd-tree construction for ray tracing using ray distribution sampling”. In: *Multimedia Tools and Applications* 75.23 (Dec. 2016), pp. 15881–15899. ISSN: 1573-7721. DOI: 10.1007/s11042-015-2896-7. URL: https://doi.org/10.1007/s11042-015-2896-7.

[10] *Nvidia Turing*. Dec. 2018. URL: https://www.nvidia.com/en-us/geforce/turing/.

[11] Arsène Pérard-Gayot, Javor Kalojanov, and Philipp Slusallek. “GPU Ray Tracing Using Irregular Grids”. In: *Comput. Graph. Forum* 36.2 (May 2017), pp. 477–486. ISSN: 0167-7055. DOI: 10.1111/cgf.13142. URL: https://doi.org/10.1111/cgf.13142.

[12] Matt Pharr, Wenzel Jakob, and Greg Humphreys, eds. Third Edition. Boston: Morgan Kaufmann, 2017. ISBN: 978-0-12-800645-0. DOI: https://doi.org/10.1016/B978-0-12-800645-0.50019-1. URL: http://www.sciencedirect.com/science/article/pii/B9780128006450500191.

[13] Joseph Bo Spjut. “Efficient Ray Tracing Architectures”. Ph.D. Thesis. The University of Utah, 2015.

[14] Eugene M. Taranta II and Sumanta N. Pattanaik. “Macro 64-regions for Uniform Grids on GPU”. In: *Vis. Comput.* 30.6-8 (June 2014), pp. 615–624. ISSN: 0178-2789. DOI: 10.1007/s00371-014-0974-x. URL: http://dx.doi.org/10.1007/s00371-014-0974-x.

[15] *The Stanford 3D Scanning Repository*. Aug. 2014. URL: http://graphics.stanford.edu/data/3Dscanrep/.

[16] Marek Vinkler, Vlastimil Havran, and Jiří Sochor. “Visibility driven BVH build up algorithm for ray tracing”. In: *Computers & Graphics* 36.4 (2012). Applications of

Geometry Processing, pp. 283–296. ISSN: 0097-8493. DOI: https://doi.org/10.1016/j.cag.2012.02.013. URL: http://www.sciencedirect.com/science/article/pii/S0097849312000362.

[17] I. Wald and V. Havran. "On building fast kd-Trees for Ray Tracing, and on doing that in O(N log N)". In: *2006 IEEE Symposium on Interactive Ray Tracing*. Sept. 2006, pp. 61–69. DOI: 10.1109/RT.2006.280216.

[18] Yuanlong Wang, Ping Guo, and Fuqing Duan. "A fast ray tracing algorithm based on a hybrid structure". In: *Multimedia Tools and Applications* 75.4 (Feb. 2016), pp. 1883–1898. ISSN: 1573-7721. DOI: 10.1007/s11042-014-2378-3. URL: https://doi.org/10.1007/s11042-014-2378-3.

[19] Turner Whitted. "An Improved Illumination Model for Shaded Display". In: *Commun. ACM* 23.6 (June 1980), pp. 343–349. ISSN: 0001-0782. DOI: 10.1145/358876.358882. URL: http://doi.acm.org/10.1145/358876.358882.

[20] Allen Y. Chang. *A Survey of Geometric Data Structures for Ray Tracing*. 2001.

[21] Jordan Zakarin. *How 'The Jungle Book' Made Its Animals Look So Real With Groundbreaking VFX*. Apr. 2016. URL: https://www.inverse.com/article/14351-how-the-jungle-book-made-its-animals-look-so-real-with-groundbreaking-vfx/.