Visualizing Query Traversals Over Bounding Volume Hierarchies Using Treemaps

Abhishek Madan*
University of Toronto

Carolina Nobre†
University of Toronto

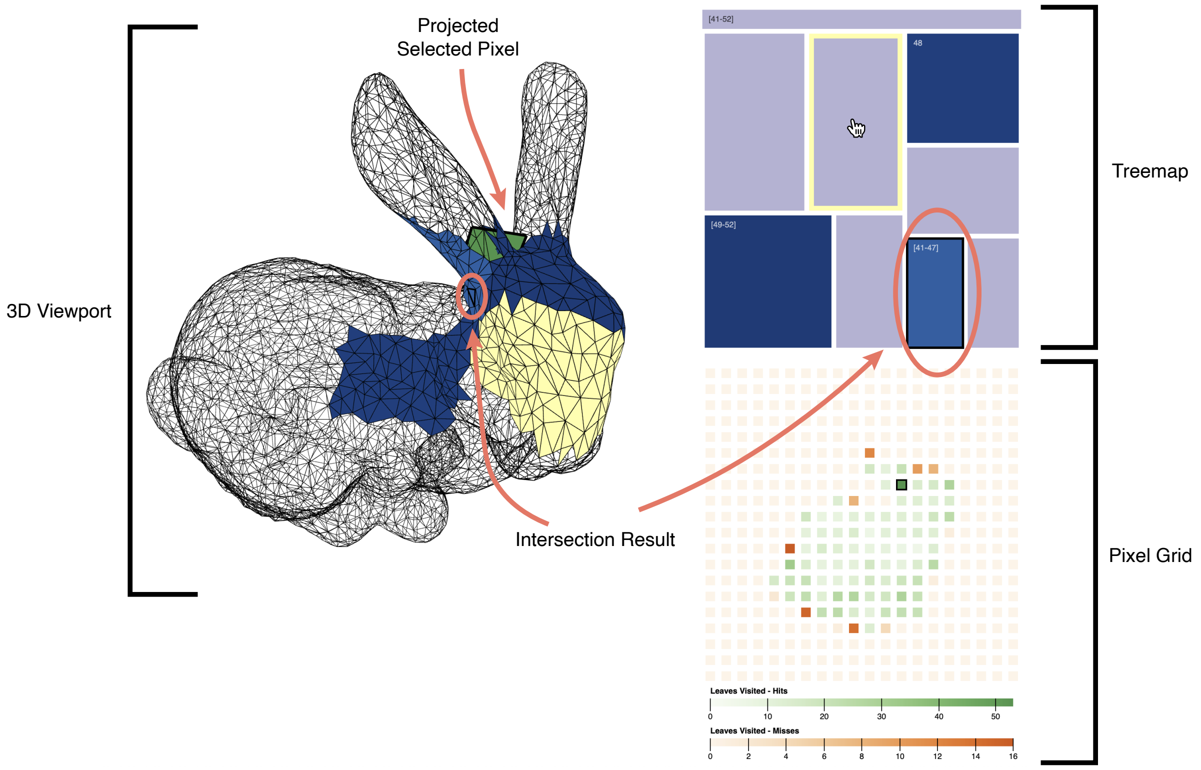

Figure 1: A bounding volume hierarchy is shown through a 3D viewport (left) coordinated with a zoomable treemap (top right) to display spatial information combined with tree structure. Nodes highlighted in the treemap (yellow outline) cause their corresponding geometry to be highlighted in the viewport (yellow triangles). Ray intersection queries can also be displayed by clicking on the pixels in a pixel grid (bottom right), and a projection of the selected pixel (outlined) is shown in the viewport as well, behind the bunny's ears in this case. The visited subtrees (i.e., groups of mesh triangles) are highlighted in both the treemap and viewport, showing their positions in space and the tree, as well as their relative traversal order via a sequential blue colormap and text labels on the treemap. The triangle representing the closest intersection with the ray, and its containing node, is outlined in both views (circled for clarity). Note that the displayed treemap is zoomed in to a subtree of the entire BVH, and the mesh is rotated from its initial position.

Abstract

Bounding volume hierarchies (BVHs) are one of the most common spatial data structures in computer graphics. Visualizing ray intersections in these data structures is challenging due to the large number of queries in typical image rendering workloads, the spatial clutter induced by superimposing the tree in a 3D viewport, and the strong tendency of these queries to visit several tree leaves, all of which add a very high dimensionality to the data being visualized. We present a new technique for visualizing ray intersection traversals on BVHs over triangle meshes. Unlike previous approaches which display aggregate traversal costs using a heatmap over the rendered image, we display detailed traversal information about individual queries, using a 3D view of the mesh, a treemap of the BVH, and synchronized highlighting between the two views, along with a pixel grid to select a ray intersection query to view. We demonstrate how this technique elucidates traversal dynamics and tree construction properties, which makes it possible to easily spot algorithmic improvements in these two categories.

Keywords: bounding volume hierarchies, treemaps

Index Terms: Human-centered computing—Visualization—Visualization techniques—Treemaps; Computing methodologies—Computer graphics—Rendering—Ray tracing

1 Introduction

One of the most popular spatial data structures in computer graphics is the bounding volume hierarchy (BVH). BVHs are trees which partition a set of geometry (e.g., triangles, tetrahedra, line segments) into a hierarchy of smaller and smaller groups based on their spatial position. This organization speeds up spatial queries like ray intersections and closest point queries, which can quickly identify and

*e-mail: amadan@cs.toronto.edu
†e-mail: cnobre@cs.toronto.edu

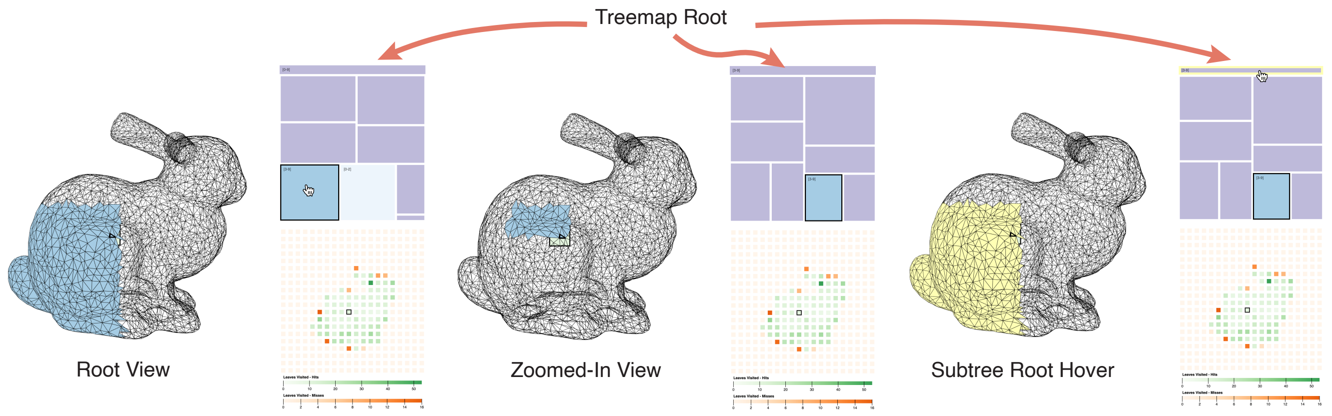

Figure 2: Demonstration of interactive features in our visual approach. Starting at the root of the BVH (left), clicking on the darker node that is traversed during the selected query zooms into that subtree (middle). The region of the mesh that is highlighted in blue is now smaller, representing the descendant of the current subtree which participated in the query. The rectangle at the top of the treemap represents the root of the subtree, as shown by its highlighting behaviour (right), and clicking on it zooms back out to the root view.

skip parts of the tree that do not contain important query candidates. BVHs are so crucial to ray tracing in particular that dedicated hardware now exists for traversing BVHs [16], and researchers are still trying to improve BVH build and query times [20, 21].

However, traversal dynamics, and the effect of tree building on traversal, are not well understood by non-experts. Researchers use their intuition and aggregate trends over a large corpus of queries to motivate and justify their methods. This process could be made much easier by visualizing individual queries, where the effects of algorithmic changes are immediately apparent. We propose a visualization technique where query traces (i.e., tree nodes visited during the traversal) are simultaneously shown in a 3D viewport displaying the geometry and a treemap representing the BVH. The separation between spatial and structural information prevents clutter, and the viewport and treemap are coordinated through color so that the correspondences between the two views are visually clear. To demonstrate the effectiveness of this visualization, we walk through two use cases for analyzing ray intersection queries using the tool. The use cases showcase how a user can identify algorithmic improvements in both tree construction and tree traversal.

2 Background and Related Work

BVHs were first proposed by Clark [5] as a way to speed up and enable a variety of applications, from level-of-detail rendering to hidden surface removal in rasterization. The basic idea is that, as opposed to *spatial subdivision* data structures like quadtrees/octrees and kd-trees, BVHs are an *object subdivison* data structure — they partition the set of geometry rather than the spatial domain. The geometry contained in each group is aggregated into a *bounding volume*, which, as the name suggests, encloses all of its associated geometry. There are many types of volumes that can be used, but the simplest one (and the one we use in this paper) is the axis-aligned bounding box, which can be completely defined by two opposite corners of the box (the "minimum" and "maximum" corners). BVHs can be constructed in many ways, based on the branching factor, leaf node size, and split criteria. In particular, modern algorithms prefer wide BVHs [21] and fairly sophisticated split criteria that attempt to minimize the estimated traversal cost [1, 14]. To keep this work simple, we start with simple BVH building and traversal algorithms (described in Sec. 4) with a naïve splitting method and a branching factor of 2 (i.e., a binary tree), but the visualization technique is agnostic to these design decisions, and we later motivate and explore more sophisticated building and traversal algorithms in Sec. 4.

Despite the extensive work on BVHs, their performance is mostly evaluated in a restricted, static manner. The most common way to report BVH performance results is through direct timing measurements or memory usage in a table [1, 6, 19, 20]. This information is very effective for conveying performance improvements but is tied to specific processors and operating systems, and on its own cannot explain performance differences. In contrast, we provide higher-level information that is both system-independent and elucidates the algorithmic behaviour that drives performance. Some papers use visualization techniques as well — for example, Ylitie et al. [21] use a heatmap of the rendered image to show per-pixel traversal performance. Our method is strictly more informative, since we show a heatmap-like view of the image as well as the associated traversal for each pixel. Liu et al. [13] use a pie chart to show a work breakdown of the average ray intersection query, such as leaf and internal node accesses that are also performed by rays from neighbouring pixels. We show internal node accesses by coloring them in zoomed-out views of the BVH and displaying their range of leaf access order indices. Our visualization also supports viewing query similarity by switching between adjacent pixel query views.

In contrast, very few prior works appear to show dynamic query visualizations. The ray tracing visualization tooklit, rtVTK [8], shows a superimposed BVH within a scene, but the main focus of the tool is on rendering algorithms, so it primarily visualizes ray paths in a 3D viewport rather than intersection queries.

Treemaps are a very popular technique for visualizing all kinds of tree data. First introduced by Johnson and Schneiderman [9], they compactly display entire trees using the size and position of tiles representing tree nodes, and are used in modern visualization systems for a wide variety of applications, from news recommenders [12] to scientific data visualization [10] and machine learning dataset exploration [2]. The layout algorithm was improved to avoid long and skinny rectangles [4], and we use this improved algorithm in our visualization. Techniques such as highlighting and distortion [18] can display search results, but prior work only seems to discuss search queries in a regime where one leaf node needs to be visited. BVH queries, on the other hand, typically visit multiple leaves during a search, and our visualization displays these visited leaves.

Mayerová [15] developed a system to view BVHs with icicle plots [11], using coordinated highlighting with a 3D viewport of the scene as well as tree and node statistics to understand the tree structure. However, their system views the entire tree at once and does not visualize queries; in contrast, our system supports both coordinated highlighting and dynamic treemap zooming to avoid overwhelming the user with information. Icicle plots create more empty space than treemaps but make the tree structure more explicit, making zoomable icicle plots a viable alternative to treemaps, though we do not explore them in this work.

3 Method

The goal of our visualization approach is to show individual BVH traversal traces, as opposed to aggregate performance over the entire query dataset. In order to do so, the entire tree structure must be clearly shown in such a way that abstract relations between nodes are clear (e.g., siblings, parents, children), as well as geometric relations between nodes (e.g., which nodes are nearby in 3D space). A simple approach is to show the entire tree in the same 3D viewport used to display the geometry, using bounding boxes with transparency so all nodes can be seen simultaneously. Unfortunately, this creates significant clutter in the viewport due to the large number of nodes which can partially overlap or even fully contain other nodes. Furthermore, it is difficult to elucidate abstract tree relationships like sibling-sibling from these bounding boxes.

A compelling alternative to a 3D view of the BVH is to simply display the tree in its abstract form using treemaps [9]. This addresses all the issues with the 3D view, because the node tiles do not overlap by definition, and the tree structure can be shown through tile positions and interactively "zooming" in and out of internal nodes. For example, in Fig. 1, there is a vertical padding line going from the root node to the bottom of the treemap that separates the descendants on each side; the nodes on the same side must be contained in the same subtree, and recursively identifying padding lines in this way allows users to deduce siblings and subtrees. However, this approach loses the spatial information derived from the viewport, making it difficult to understand why queries behave a particular way.

Neither a 3D viewport nor a treemap convey sufficient information on their own, but the information they do convey is complementary. Thus, we can obtain the best of both worlds by displaying both views simultaneously, and using coordinated highlighting to display the relationships between the two views. As shown in Fig. 1, hovering the mouse over a treemap node will highlight both that node and the corresponding viewport triangle(s) in yellow, which shows the correspondence between the two views on demand. In order to show the BVHs of large meshes, the treemap supports zooming in and out (Fig. 2): clicking on an internal node will zoom into that subtree and map it to the "root" node at the top of the treemap; clicking on that "root" node at the top of the treemap will zoom out. To display more nodes in a single view and reduce the amount of time spent zooming into leaves, we display 3 levels of the tree at a time, where the displayed nodes are obtained by traversing 3 levels down from the current root, which produces at most 8 nodes (fewer nodes are obtained if leaves are encountered less than 3 levels below the current root). The size of each treemap node is proportional to the sum of triangle areas contained in that node.

3.1 Ray Intersection Query Traces

Building upon the basic coordinated visualization established earlier, we developed a way to visualize query traces. The rest of this section will focus on ray intersection queries but it is worth noting that most of the techniques used here can also be used to visualize other queries, such as closest point queries.

Briefly, ray intersection queries work by intersecting the ray with the root BVH node's bounding box, and if it intersects, recursively intersecting with its child nodes or contained triangles if it is an internal node or leaf, respectively. This intersection method is already much faster than a brute force loop over all mesh triangles, but it can be sped up even further by carefully selecting the traversal order of sibling nodes, as we will demonstrate in Sec. 4.

A pixel grid is used to select which query to visualize (shown in the bottom right of Fig. 1), which aligns well with the most common use case of ray intersections in ray tracing, where rays start from a camera and are directed towards various points on an image plane in order to generate an image. To aid users in selecting salient queries, each pixel is colored based on whether or not its ray intersects the geometry, as well as the cost of the ray intersection query measured in terms of the number of leaves visited during the traversal. The former is encoded using hue (this implementation uses green for hits and orange for misses), and the latter is encoded using a sequential (luminance-based) colormap. These two variables remain orthogonal in the encoding, so there are two sequential colormaps used in the pixel grid. To more precisely quantify the number of leaves visited, two color bars are shown below the pixel grid, corresponding to hits and misses, and hovering over a pixel displays the exact number of leaf visits and whether the ray hit or missed the mesh.

Once a pixel (and its corresponding ray intersection query) have been selected, the query is displayed on the treemap and in the viewport, and a projection of the selected pixel is also shown in the viewport (see Fig. 1). Treemap nodes corresponding to subtrees or leaves visited during the traversal are highlighted using a blue sequential colormap based on traversal order, and triangles corresponding to those nodes are also colored the same way. For additional precision, each treemap node is also labelled with the range of leaves contained in the node that are visited by the query, indexed by traversal order (node colors are determined by the lower bound of these index ranges). The traversals we examine in this paper are depth-first, so we are guaranteed to get a contiguous range in each node. To aid in interactive zooming, treemap nodes can still be highlighted by hovering.

4 Implementation and Evaluation

This visualization was implemented in JavaScript, using D3.js for the treemap and pixel grid [3], and P5.js for viewport rendering [7].

We show how the tool highlights improvement in query performance and tree quality by comparing visualizations on a truck mesh before and after a series of algorithmic changes used in more sophisticated BVHs (Fig. 3). The truck's long and skinny triangles dramatically benefit from the algorithmic changes.

First, we implemented an optimization for the traversal algorithm. The base implementation only checks ray-bounding box intersection to determine whether or not a subtree should be traversed, but this leads to cases where the traversal visits the back of the mesh relative to the ray (e.g., the far side of the front windshield in Fig. 3, left), or where the traversal visits expensive subtrees that do not contain the intersecting triangle (the light blue group at the back of the truck, circled). We can skip many subtrees by performing an ordered "front-to-back" traversal (see, e.g., [17]), where subtrees closer to the ray origin are traversed first, and farther subtrees are skipped if we can guarantee that nothing in their bounding boxes is closer than the closest intersection recorded so far. This leads to nearly a 25% improvement in worst-case hit performance from roughly 1200 leaves visited to 900, as can be seen in the upper end of the hit color bar of Fig. 3, middle. Furthermore, in the selected query, the subtree representing the back of the truck is no longer traversed — rather than naïvely visiting it first, the traversal defers visiting that subtree until it has enough information to safely skip that region.

However, the traversal is still not as efficient as we would like, for several reasons. The selected ray intersection query is still unable to skip the group at the far side of the front windshield that we identified earlier, and the windshield subtrees are visited first despite the intersection result being on the roof of the truck. These problems all have the same underlying cause: the triangles of sibling nodes are "entangled" in such a way that their subtree boxes significantly overlap, making it impossible for even our improved traversal to safely skip many subtrees. Thus, the problem lies in the tree structure itself, so we implemented a more sophisticated tree building algorithm using a cost model called the surface area heuristic [14]. Unlike the base implementation, which constructs trees by recursively partitioning geometry based on centroid positions relative to the "midpoint" plane along the longest bounding box axis (i.e., the

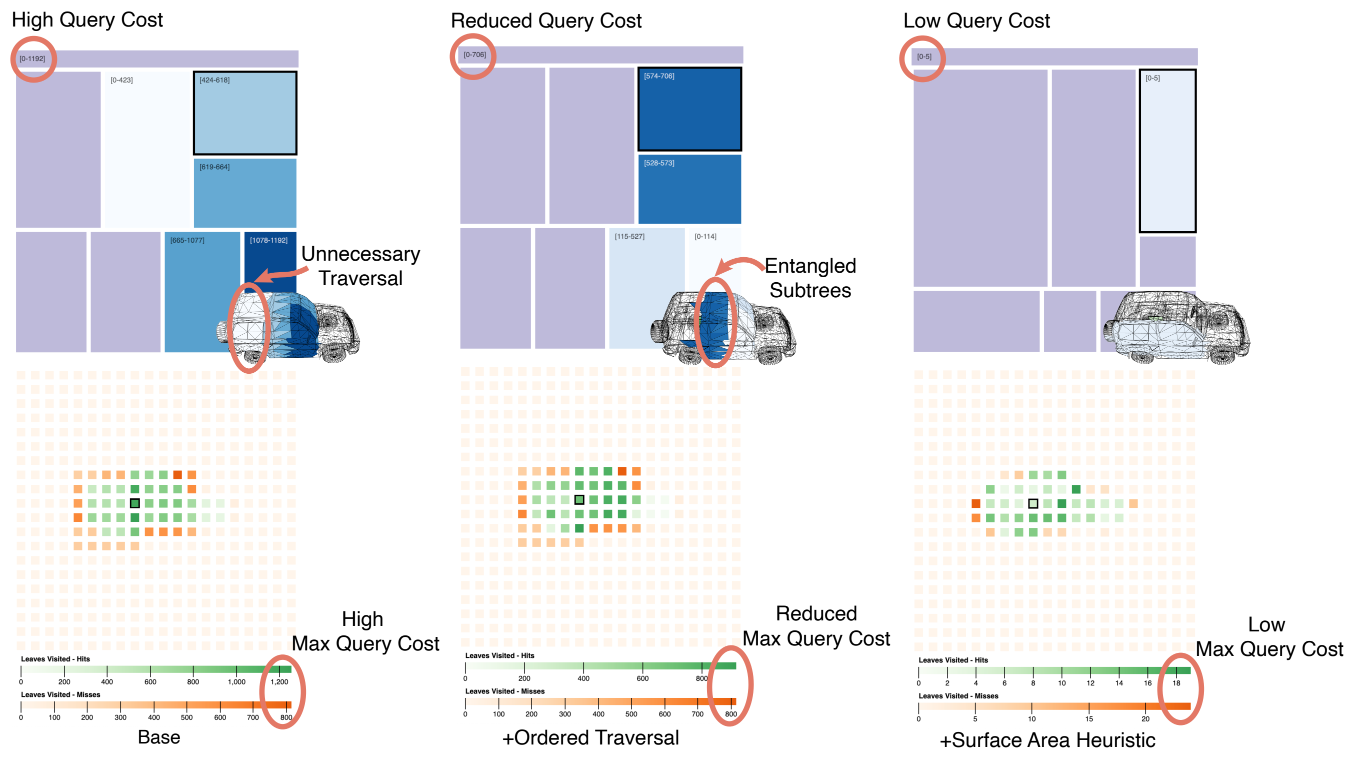

Figure 3: Two use cases showing how our visualization clearly highlights improvements from the base implementation (left), to an ordered traversal algorithm (middle), and a surface area heuristic-based tree builder (right). The same query is selected in all three versions, and the mesh view is shown as an inset for each treemap/pixel grid. The base implementation visits over one thousand leaves for the selected query; adding the ordered traversal reduces the selected query and maximum query cost, but still visits several hundred leaves; adding the surface area heuristic significantly reduces the selected query cost down to 6 leaves, as well as the maximum query cost.

plane normal to the longest axis that passes through the center of the box), we instead select an axis and (normal) splitting plane that minimizes the expected traversal cost of a random ray. The probabilistic model driving this cost function comes from the observation that, for uniformly distributed rays far away from a convex object (in this case, the mesh's bounding box), the probability of a ray intersection is proportional to the object's surface area (hence the name "surface area heuristic"). The optimization procedure is based on the one described by Pharr et al. [17], where the splitting plane can be selected from a discrete set of equally spaced positions within the box. Implementing this on top of the traversal optimization described earlier (Fig. 3, right), we see a dramatic improvement in query performance. The near side of the truck doors are one subtree, and only that subtree is traversed all the way to its leaves. As a result, the traversal only visits 6 leaves, compared to the hundreds it needed to visit with just the ordered traversal. Worst-case visited leaves improved significantly as well, going from roughly 900 to 25 (for missed rays). It is important to note that the surface area heuristic is based on rays from *any* direction, not just rays coming from a fixed camera position and a limited range of ray directions. Thus, the especially dramatic results shown here are not representative of these trees' performance on any set of ray intersection queries; some may only exhibit mild improvements or even slight degradations compared to the base tree builder. Nevertheless, analyzing performance over several viewing directions and camera positions is beyond the scope of the paper, and this example was chosen to emphasize the visualization's reflection of each change.

5 Discussion and Future Work

We presented a new technique for visualizing bounding volume hierarchies by combining a 3D geometric view with an abstract treemap, with coordinated interactions between the two, and a pixel grid to select a ray intersection query to view. Through this view, users can see the hierarchical and spatial relationships between tree nodes in separate but coordinated views, which thereby makes it easier to identify slow queries and diagnose inefficiencies in tree building and tree traversal algorithms. Through use cases for improving ray intersection performance, we can see the effect of efficient BVH algorithms used in practice.

There are also some limitations to our work in its current form, with interesting avenues for future work. It is currently not possible to easily view the distribution of work aggregated over the entire image (e.g., through a histogram); users can only do this by visually "grouping" pixels with similar colors, which is difficult since they may be far apart in the grid. It is also difficult to see the height/depth of individual tree nodes, though it can be deduced through zooming interactions. Also, the pixel grid is small, and clicking on individual pixels does not scale to high-resolution images rendered in practice; a scrubbing interface to select a small window of the full image to show in the pixel grid is one possible solution. Nevertheless, this visualization elucidates many important characteristics of BVH traversals that are typically hidden in very high-dimensional query traces or obscured by aggregated performance statistics, and we hope this work makes it easier for students, industry practitioners, and researchers to understand BVH query performance.

Acknowledgments

This research has been funded by in part by NSERC Discovery (RGPIN-2017–05524), Connaught Fund (503114), Ontario Early Researchers Award (ER19–15–034), Gifts from Adobe Research and Autodesk, and the Canada Research Chairs Program.

References

[1] T. Aila, T. Karras, and S. Laine. On quality metrics of bounding volume hierarchies. In *Proceedings of the 5th High-Performance Graphics Conference*, pp. 101–107, 2013.

[2] D. Bertucci, M. M. Hamid, Y. Anand, A. Ruangrotsakun, D. Tabatabai, M. Perez, and M. Kahng. Visual exploration of large-scale image datasets for machine learning with treemaps. *arXiv preprint arXiv:2205.06935*, 2022.

[3] M. Bostock. D3.js - data-driven documents, 2022.

[4] M. Bruls, K. Huizing, and J. J. Van Wijk. Squarified treemaps. In *Data Visualization 2000: Proceedings of the Joint EUROGRAPHICS and IEEE TCVG Symposium on Visualization in Amsterdam, The Netherlands, May 29–30, 2000*, pp. 33–42. Springer, 2000.

[5] J. H. Clark. Hierarchical geometric models for visible surface algorithms. *Commun. ACM*, 19(10):547–554, oct 1976. doi: 10.1145/360349.360354

[6] L. R. Domingues and H. Pedrini. Bounding volume hierarchy optimization through agglomerative treelet restructuring. In *Proceedings of the 7th Conference on High-Performance Graphics*, pp. 13–20, 2015.

[7] P. Foundation. P5.js, 2022.

[8] C. Gribble, J. Fisher, D. Eby, E. Quigley, and G. Ludwig. Ray tracing visualization toolkit. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, pp. 71–78, 2012.

[9] B. Johnson and B. Shneiderman. Tree-maps: A space filling approach to the visualization of hierarchical information structures. Technical report, 1998.

[10] A. Kamakshidasan and V. Natarajan. Mergemaps: Treemaps for scientific data. In *Topological Methods in Data Analysis and Visualization VI: Theory, Applications, and Software*, pp. 19–38. Springer, 2021.

[11] J. B. Kruskal and J. M. Landwehr. Icicle plots: Better displays for hierarchical clustering. *The American Statistician*, 37(2):162–168, 1983.

[12] J. Kunkel, C. Schwenger, and J. Ziegler. Newsviz: depicting and controlling preference profiles using interactive treemaps in news recommender systems. In *Proceedings of the 28th ACM conference on user modeling, adaptation and personalization*, pp. 126–135, 2020.

[13] L. Liu, W. Chang, F. Demoullin, Y. H. Chou, M. Saed, D. Pankratz, T. Nowicki, and T. M. Aamodt. Intersection prediction for accelerated gpu ray tracing. In *MICRO-54: 54th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 709–723, 2021.

[14] J. D. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer*, 6(3):153–166, 1990.

[15] E. Mayerová. Interactive visualization of bounding volume hierarchies, 2016.

[16] NVIDIA. Nvidia rtx ray tracing, 2022.

[17] M. Pharr, W. Jakob, and G. Humphreys. *Physically Based Rendering: From Theory to Implementation (3rd ed.)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 3rd ed., Oct. 2016.

[18] K. Shi, P. Irani, and P. C. Li. Facilitating visual queries in the treemap using distortion techniques. In *Symposium on Human Interface and the Management of Information*, pp. 345–353. Springer, 2007.

[19] K. Vaidyanathan, S. Woop, and C. Benthin. Wide bvh traversal with a short stack. In *Proceedings of the Conference on High-Performance Graphics*, pp. 15–19, 2019.

[20] I. Wald and S. G. Parker. Data parallel path tracing with object hierarchies. *Proc. ACM Comput. Graph. Interact. Tech.*, 5(3), jul 2022. doi: 10.1145/3543861

[21] H. Ylitie, T. Karras, and S. Laine. Efficient incoherent ray traversal on gpus through compressed wide bvhs. In *Proceedings of High Performance Graphics*, pp. 1–13. 2017.