

On the Importance of Scene Structure for Hardware-Accelerated Ray Tracing

Martin Kacerik
Czech Technical
University in Prague
Karlovo namesti 13
121 35, Prague, Czech
Republic
kacerma2@fel.cvut.cz

Jiri Bittner
Czech Technical
University in Prague
Karlovo namesti 13
121 35, Prague, Czech
Republic
bittner@fel.cvut.cz

ABSTRACT

Ray tracing is typically accelerated by organizing the scene geometry into an acceleration data structure. Hardware-accelerated ray tracing, available through modern graphics APIs, exposes an interface to the acceleration structure (AS) builder that constructs it given the input scene geometry. However, this process is opaque, with limited knowledge and control over the internal algorithm. Additional control is available through the layout of the AS builder input data, the geometry of the scene structured in a user-defined way. In this work, we evaluate the impact of a different scene structuring on the run time performance of the ray-triangle intersections in the context of hardware-accelerated ray tracing. We discuss the possible causes of significantly different outcomes (up to 1.4 times) for the same scene and identify a potential to reduce the cost by automatic input structure optimization.

Keywords

real-time ray tracing, acceleration structures, bounding volume hierarchies

1 INTRODUCTION

Graphics APIs with access to ray tracing hardware features, such as Vulkan or DirectX, utilize internally built two-level acceleration data structures. These data structures are used during the ray tracing to accelerate a ray-triangle intersection. The data layout, as well as the build algorithm, is provided by a driver vendor and is typically opaque to the user of the API (except for several open-source driver implementations).

When an AS is being constructed for an existing scene, certain organized layout of the scene data is expected as an input for the algorithm. In this paper, we refer to this input layout as a *scene structure*. Intuitive transfer of a scene organized in a scene graph to this scene structure is natural, but likely a suboptimal approach, especially for more complex and dynamic scenes. Nevertheless, preserving the information from the scene graph is beneficial, as it connects the geometric and material properties, including UV coordinates, or object instancing.

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.

This work explores multiple available options for construction of an opaque acceleration structure for hardware ray tracing. We evaluate the impact of scene structuring in conjunction with different construction algorithm hyper-parameters on the traversal performance of the final acceleration structure. Finally, we reason about the differences in performance and outline a way to derive a better input scene layout automatically.

The paper is structured as follows: Section 2 presents relevant research in the field. Section 3 gives a brief introduction to the acceleration structure API. Section 4 explores the input data layout for bounding volume hierarchy (BVH) construction. Section 5 describes the evaluation process and presents the measured data. In Section 6, we discuss the results and draw an explanation for the measurements. Finally, Section 7 concludes the paper.

2 RELATED WORK

An exhaustive study on the topic of bounding volume hierarchies for ray tracing was recently presented by Meister et al. [Mei21a].

Our work is directly related to massively parallel GPU-accelerated BVH construction algorithms. One of the fastest among these techniques is the top-down, binning-based construction algorithm *linear BVH* (LBVH) proposed by Lauterbach et al. [Lau09a]. The LBVH was then extended to the *hierarchical LBVH*

by Pantaleoni and Leubke [Pan10a], who employed a *surface area heuristic* (SAH) for the upper levels of the hierarchy. The SAH was originally introduced by Goldsmith and Salmon [Gol87a] as a metric approximating the likelihood of a ray-volume intersection and it is employed for driving the vast majority of the BVH build algorithms. A different approach was taken by Meister and Bittner [Mei17a], who proposed a GPU-accelerated bottom-up build algorithm using agglomerative clustering: parallel locally-ordered clustering (PLOC). Recently, *PLOC++* by Benthin et al. [Ben22a] has been proposed, addressing certain technical weaknesses of the original PLOC, such as the number of dispatched GPU kernels.

The refinement of an existing BVH structure of a lower or deteriorated quality back to a near-optimal state was investigated by Benthin et al. [Ben17a], who presented a process of *partial re-braiding* to reduce overlaps in the AS and improve the SAH quality of the BVH. In a similar fashion, Hendrich et al. [Hen17a] proposed a *progressive hierarchical refinement*, a method of improving the outcome of a fast but low-quality BVH builder, such as L BVH. This method could be used to perform a build directly from the scene graph hierarchy.

In past years, major chip vendors, starting with NVIDIA, later joined by AMD and Intel, introduced ray tracing acceleration to their mainstream GPU lineup. Although the internal functioning is mostly hidden, there were attempts to improve the overall ray tracing performance by reorganizing the data used for the ray tracing process, while considering the ray tracing API as a black box. In particular, Meister et al. [Mei20a] investigated the possibilities of ray reordering, while Wald et al. [Wal20a] focused on exploiting the API design to improve the AS hierarchy for a special case of long and thin geometries. Our work aims to lay a foundation for further improvements by performing scene graph restructuring prior to passing the data to the ray tracing API.

3 ACCELERATION STRUCTURE API

In this work, we consider following ray tracing APIs: DirectX Raytracing (DXR), Vulkan and NVIDIA OptiX. Although they might differ in specific capabilities and naming conventions, they all conform to similar programming model.

Acceleration structure defined in a particular API is an opaque data structure utilized in subsequent queries to accelerate ray-object intersection. In general, the layout of the AS, as well as the related algorithm to build the AS, is internal to a specific implementation. Based on the public interface for an AS manipulation, we can derive the knowledge discussed in this section.

3.1 Data layout

The AS is formed in two logical levels - a *bottom level acceleration structure* (BLAS) and a *top level acceleration structure* (TLAS). BLAS nodes consist of geometry data (multiple disjoint geometries can be easily merged into one BLAS), while TLAS nodes reference the BLAS nodes and include their respective transformation and shading data (using a relevant shader index). The TLAS enables storage of geometry in a local coordinate system and supports geometry instancing, as illustrated in Fig. 1.

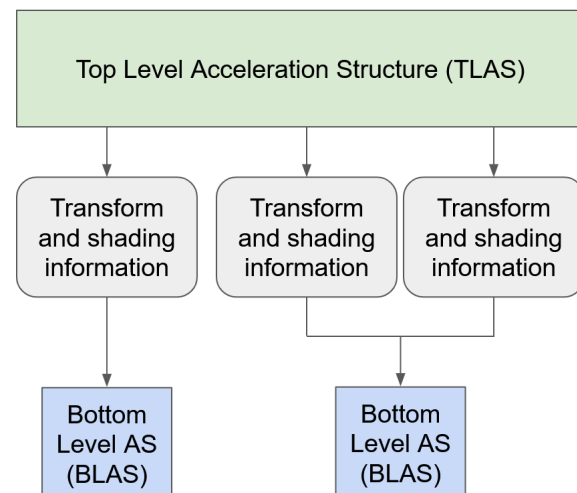


Figure 1: Logical representation of the acceleration structure design as exposed by contemporary GPU APIs.

Terms BLAS and TLAS are directly utilized in DXR and Vulkan APIs, while OptiX chooses terms *geometry acceleration structure* (GAS) and *instance acceleration structure* (IAS).

3.2 Build/update algorithms

The API distinguishes between two modes when building a new AS, either building from scratch or updating the existing AS. AS update corresponds to bounding volume refit, a technique available when specific conditions regarding the topology of the updated geometry are met. Specifically, only instance definitions, transform matrices, and vertex or axis aligned bounding box (AABB) positions are allowed to change during the update. Refit is fast but also likely to deteriorate the quality of the AS over time.

In accordance of the two-level logical hierarchy, both modes are executed in two steps. In the first step, BLASes are built. As long as there is enough memory available for auxiliary buffers required by BLAS builders, multiple BLAS builds can be scheduled to run concurrently without any additional synchronization. When all BLASes are available, the TLAS build

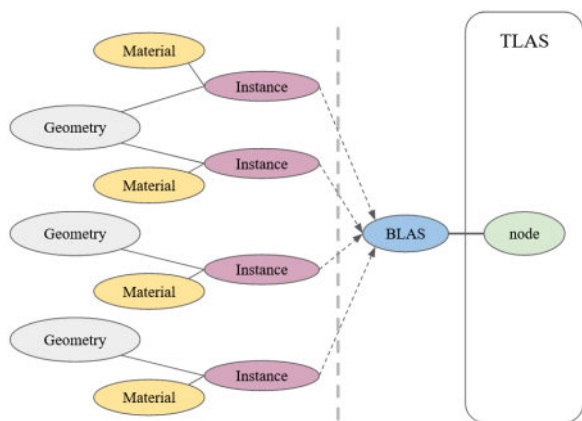


Figure 2: Input scene structure for AS construction, created by mapping a scene graph structure (grey, yellow, and pink nodes) to one joined BLAS node. Material and transformation information is not stored in the AS.

can start. Once finished, the AS is ready to be deployed for hardware-accelerated ray tracing.

Coarse control over the building process is allowed in the form of flags, hinting our preference to the builder. Notably, flags such as `PREFER_FAST_TRACE`, `PREFER_FAST_BUILD` (available in all mentioned APIs), or `LOW_MEMORY` (available in DXR and Vulkan) indicate that the underlying implementation is expected to utilize multiple different algorithms with different trade-offs in terms of the AS build speed, runtime traversal cost, or overall memory consumption.

4 MAPPING A SCENE GRAPH TO THE AS BUILD LAYOUT

3D scenes are typically stored in a scene graph hierarchy, maintaining all vital information about the relations of scene objects: i.e. instances, their geometries, materials, and transformations.

A valid method to submit such data for an AS build is to transform all geometries in place, merge them into one BLAS node, and assign it to a TLAS node with a unit transformation, see Fig. 2. Under such conditions, the BLAS builder is likely to execute the job to its full potential and build the AS of the best quality, thanks to global knowledge of the scene in one place. However, all the benefits of TLAS are given up, including any possibility of fast partial rebuilds and refits of the AS, geometry instancing, or referencing of a specialized shader for defined material.

Another approach to submit scene data to an AS building API is to map geometry nodes to BLAS nodes and object instances to TLAS nodes, see Fig. 3. However, as we show in the next section, submitting the scene graph data organized from a scene designer perspective can have significant performance consequences and will likely translate to a suboptimal acceleration structure.

5 EVALUATION

We evaluated several different ray tracing setups using eight static test scenes. The list of evaluated scenes with some of their parameters is shown in Table 1. All scenes were taken from Morgan McGuire’s online archive [McG17a]. Default scene graph hierarchy was loaded from the scene source file. The measurements were performed on a computer equipped with Intel i9-10900X CPU, 128GB RAM, and NVIDIA RTX3080Ti GPU (driver v525.89.02).

Scene	Triangles	#Instances	Overlap
Fireplace room	143 173	51	4.1
Chestnut	316 880	5	3.6
Sibenik cathedral	75 284	1087	21.1
Crytek Sponza	262 267	393	11.1
Bistro interior	1 046 609	2062	22.8
Bistro exterior	2 832 120	1591	22.5
Power plant	12 759 246	57	8.7
San Miguel	9 980 699	2135	226.3

Table 1: List of used scenes and their geometric complexity. Overlap metric represents an overlap of axis aligned bounding boxes of instances in the scene and is computed in a following way: for each pair of AABBs of instances, we compute a surface area of their overlapping region. The overlap value is then the sum of these areas divided by a surface area of the scene’s AABB.

Performance evaluation was done in a custom path tracing engine running on a Vulkan API backend. For each scene, we traced five 1920x1080 views with 2048 samples per pixel and a maximum recursion depth of eight. To focus the measurement on the AS performance, we employed simple Lambertian BRDF for surface interaction, avoiding any complex shading computations.

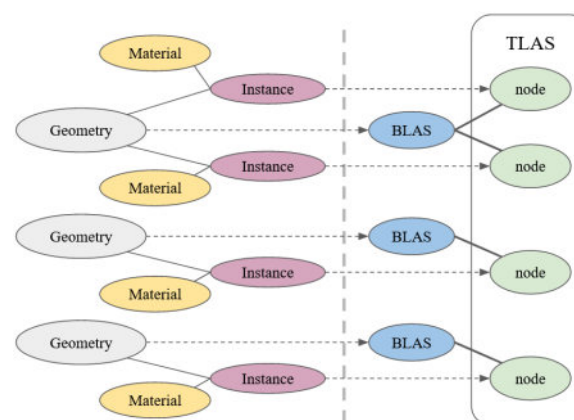


Figure 3: Input scene structure for AS construction, created by mapping a scene graph structure (grey, yellow, and pink nodes) to one BLAS node per scene geometry, with the possibility of instancing. Material and transformation information is stored in TLAS nodes.

Every sample tracks the number of traced rays internally, which is then added to an atomic counter. The measurements showed that the atomic add has an insignificant impact on the performance. Time is tracked through the timestamp query API with advertised nanosecond precision.

We measured the following configurations of the scene: input scene structures mapped as described in chapter 4: (1) one BLAS per geometry node in the scene and (2) one BLAS for the whole scene. Both configurations are then measured with three different opaque AS builder options: `PREFER_FAST_TRACE`, `PREFER_FAST_BUILD`, and `LOW_MEMORY` in the build AS mode (AS is built from scratch). The rendered output is always identical. The measured values are presented in Figures 4, 5, and 6. Reported values represent sum of costs of both AS levels, where TLAS times or memory consumption are negligible compared to BLAS values.

6 DISCUSSION

The measured results support our initial assumption that the scene structure provided to the opaque AS builder significantly impacts the final ray tracing performance. The relation between the geometric complexity and overlap of the acceleration structure nodes seems like a good final performance predictor. This correlation can also be observed in the visualizations shown in Tab. 2 and the traversal performance in Fig. 4. The superior traversal performance of one BLAS per whole scene is especially pronounced in the scenes with an otherwise high overlap of instance AABBs.

The hypothesis outlined in section 4 states, that the global scene knowledge available for the AS builder in one BLAS node will lead to superior AS quality and thus better runtime performance. We can conclude that this hypothesis was proven right in almost all cases, except the Fireplace scene with the fast trace setting on. The behavior in the Fireplace scene is unexpected and it suggests that, in some cases, a better acceleration structure can be found when the scene graph holds useful structural information that is not found by the AS builder. As an anomaly, this case is interesting, and it will be the subject of further investigation.

On average, the trace speed of the BLAS per scene is 1.3 times higher in the `PREFER_FAST_TRACE` case, 1.41 times higher for the `PREFER_FAST_BUILD` case, and 1.37 times higher for the `LOW_MEMORY` case than that of the one BLAS per geometry.

In contrast to the trace speed, the build speed of the AS builder itself, reported in Fig. 5, remains mostly in favor of multiple smaller BLAS nodes compared to one big BLAS node. This is expected due to the hardware ability to run the construction of multiple nodes concur-

rently, as well as due to the known $O(n \cdot \log(n))$ complexity of the AS construction algorithms.

We consider minimization of the node overlap as one of the key steps in the potential scene restructuring process that would optimize the trace speed while keeping the high level scene structure. The problematic nodes, causing unnecessary overlap, have to be identified and, based on the local decision, cut or joined. This decision has to be guided by the local complexity, possibly predicted by metrics like SAH.

A possible further research direction is the analysis of the influence of BLAS orientation. Axis-aligned bounding boxes, which serve as an underlying bounding volume for the AS, are not invariant to rotation of bounded geometry. Discovering the optimized initial rotation for the geometry, minimizing the SAH cost and likely the overlap of its AABB, can thus benefit the overall performance.

7 CONCLUSION

This paper discussed the problem of the dependence of the hardware accelerated ray tracing performance on the AS construction input scene structure. We showed that although the benefits of a two-level acceleration structure are numerous (instancing, local transformations, material referencing, fast refit), the performance penalty for suboptimally organized scenes can be significant.

As indicated by the Fireplace scene, we believe that it is possible to find a scene structure that keeps the TLAS benefits and also minimizes the performance impact. This paper aims to provide the groundwork necessary for the following research of a scene graph layout restructuring to achieve superior results with an opaque AS builder.

8 ACKNOWLEDGMENTS

This research was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS22/173/OHK3/3T/13 and the Research Center for Informatics No. CZ.02.1.01/0.0/0.0/16_019/0000765.

9 REFERENCES

- [Ben17a] Benthin, C., Woop, S., Wald, I., and Áfra, A. T. Improved two-level bvhs using partial re-braiding. In Proceedings of High Performance Graphics (New York, NY, USA, 2017), HPG '17, Association for Computing Machinery, pp. 1-8.
- [Ben22a] Benthin, C., Drabinski, R., Tessari, L., and Dittebrandt, A. Ploc++: Parallel locally-ordered clustering for bounding volume hierarchy construction revisited. Proc. ACM Comput. Graph. Interact. Tech. 5, 3 (Jul 2022), pp. 1-13.

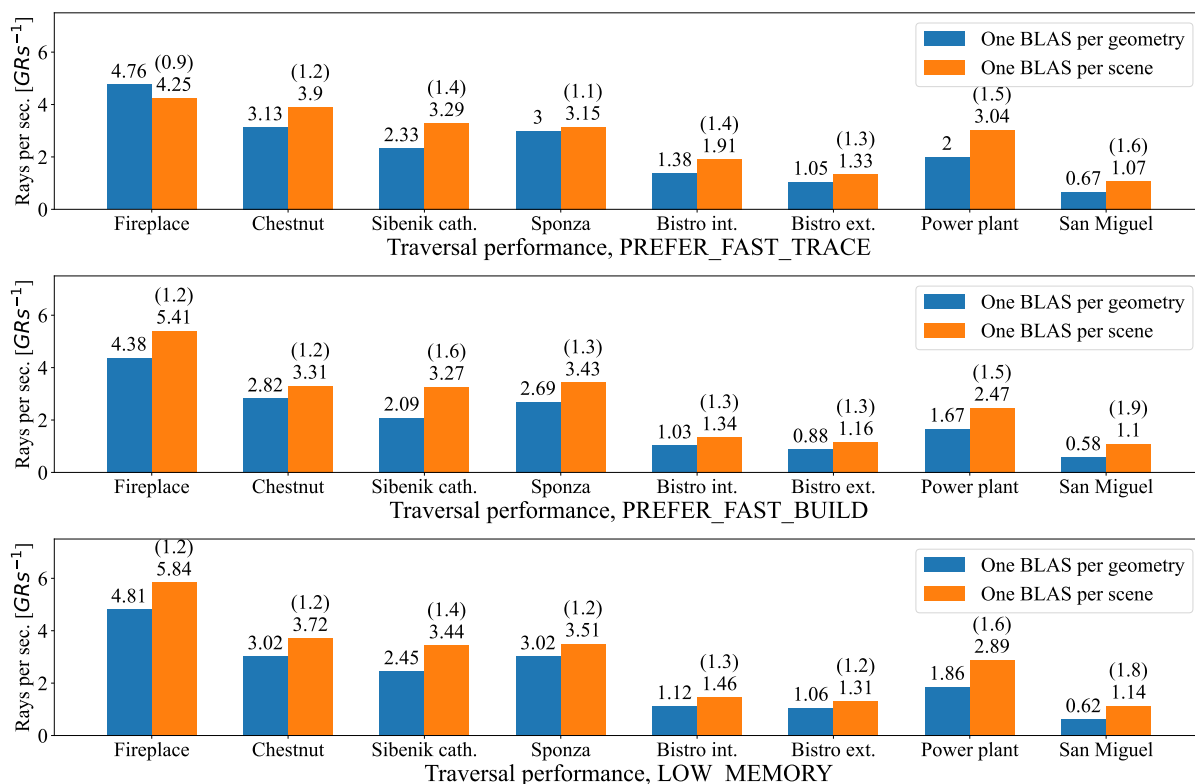


Figure 4: Ray tracing speed of acceleration structures constructed with six different configurations. The trace speed is measured in GigaRays per second (the higher the better).

[Gol87a] Goldsmith, J., and Salmon, J. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications* 7, 5 (May 1987), 14-20.

[Hen17a] Hendrich, J., Meister, D., and Bittner, J. Parallel bvh construction using progressive hierarchical refinement. *Computer Graphics Forum (Proceedings of Eurographics 2017)* 36, 2 (2017), 487-494

[Lau09a] Lauterbach, C., Garland, M., Sengupta, S., Luebke, D., and Manocha, D. Fast bvh construction on gpus. In *Computer Graphics Forum* (2009), vol. 28, Wiley Online Library, pp. 375-384.

[McG17a] McGuire, M. Computer graphics archive, July 2017. <https://casual-effects.com/data>. Accessed: 2023-03-27.

[Mei17a] Meister, D., and Bittner, J. Parallel locally-ordered clustering for bounding volume hierarchy construction. *IEEE transactions on visualization and computer graphics* 24, 3 (2017), 1345-1353.

[Mei20a] Meister, D., Boksansky, J., Guthe, M., and Bittner, J. On ray reordering techniques for faster gpu ray tracing. In *Symposium on Interactive 3D Graphics and Games (New York, NY, USA, 2020)*, I3D '20, Association for Computing Machinery, pp. 1-9.

[Mei21a] Meister, D., Ogaki, S., Benthin, C., Doyle, M. J., Guthe, M., and Bittner, J. A survey on bounding volume hierarchies for ray tracing. In *Computer Graphics Forum* (2021), vol. 40, Wiley Online Library, pp. 683-712.

[Pan10a] Pantaleoni, J., and Luebke, D. Hlbvh: hierarchical lbvh construction for real-time ray tracing of dynamic geometry. In *Proceedings of the Conference on High Performance Graphics* (2010), pp. 87-95.

[Wal20a] Wald, I., Morrical, N., Zellmann, S., Ma, L., Usher, W., Huang, T., and Pascucci, V. Using hardware ray transforms to accelerate ray/primitive intersections for long, thin primitive types. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 3, 2 (2020), 1-16.

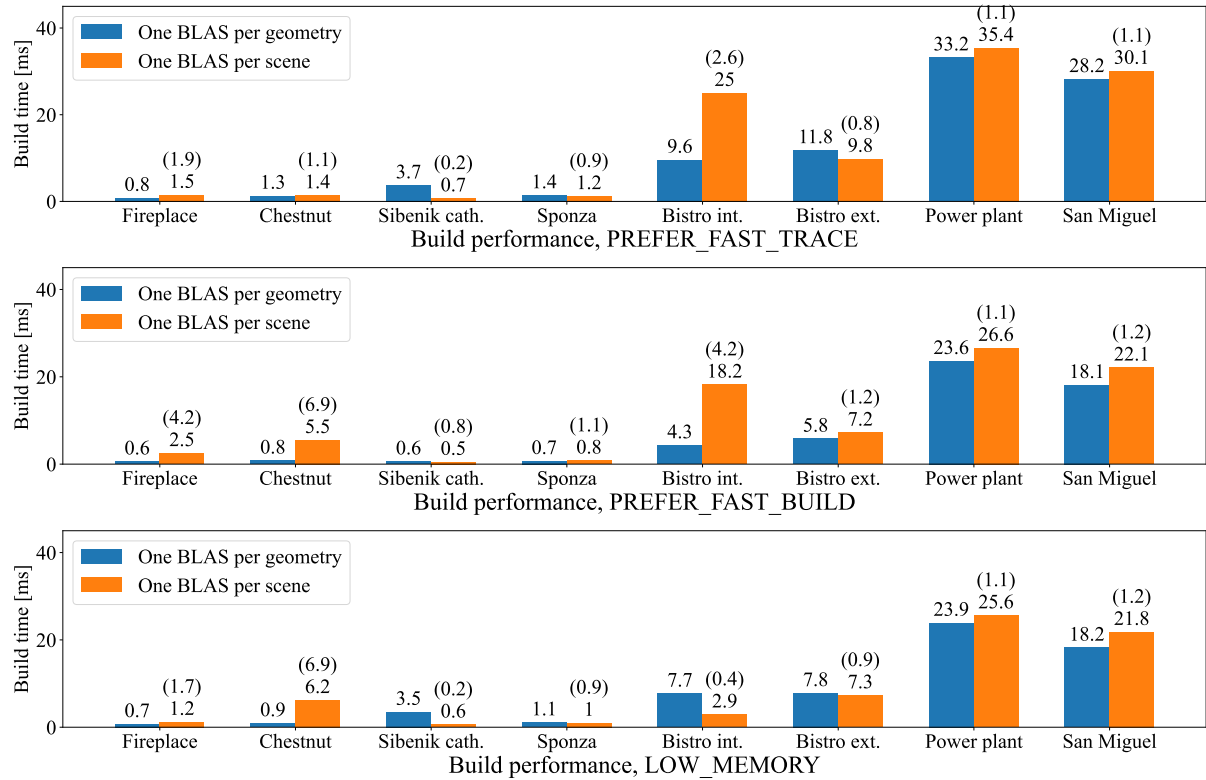


Figure 5: Build times of acceleration structures constructed with six different configurations, measured in milliseconds (the lower the better).

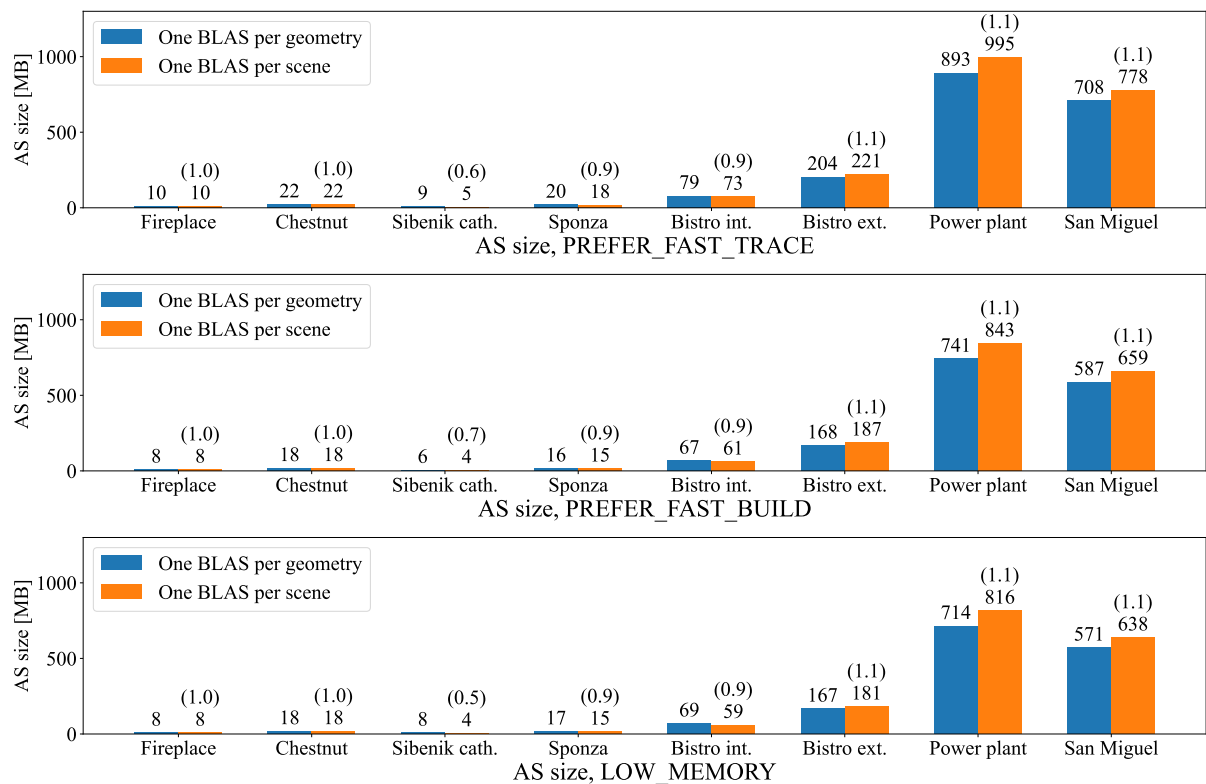


Figure 6: Memory consumption of acceleration structures constructed with six different configurations, measured in MegaBytes (the lower the better).



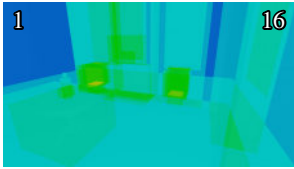


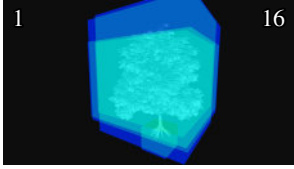

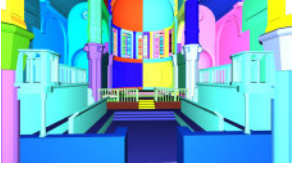
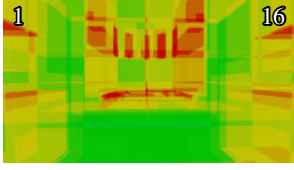


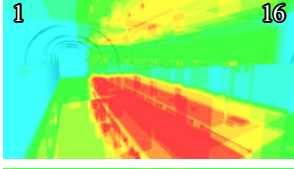


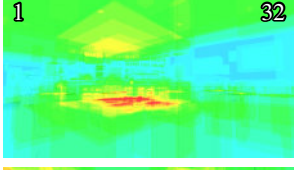


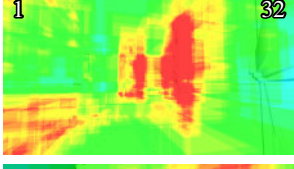

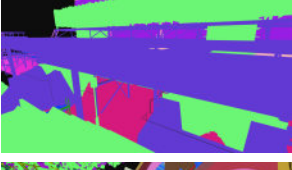
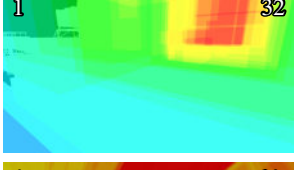

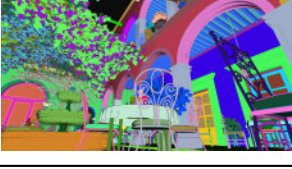
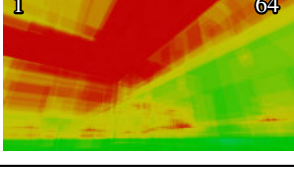
Scene	Diffuse view	Instance view	Instance AABB overlap
Fireplace room			 1 16
Chestnut			 1 16
Sibenik cathedral			 1 16
Crytek Sponza			 1 16
Bistro interior			 1 32
Bistro exterior			 1 32
Power plant			 1 32
San Miguel			 1 64

Table 2: Evaluated scenes, with visualization of complexity of the scene for "One BLAS per geometry" case. The middle column shows ID buffers with unique colors for each object instance, and the right column shows heat maps indicating the overlap of bounding boxes of instances. The lower and upper bounds of the heat map scale defined are shown in the top left and top right corners of the heat maps, respectively.