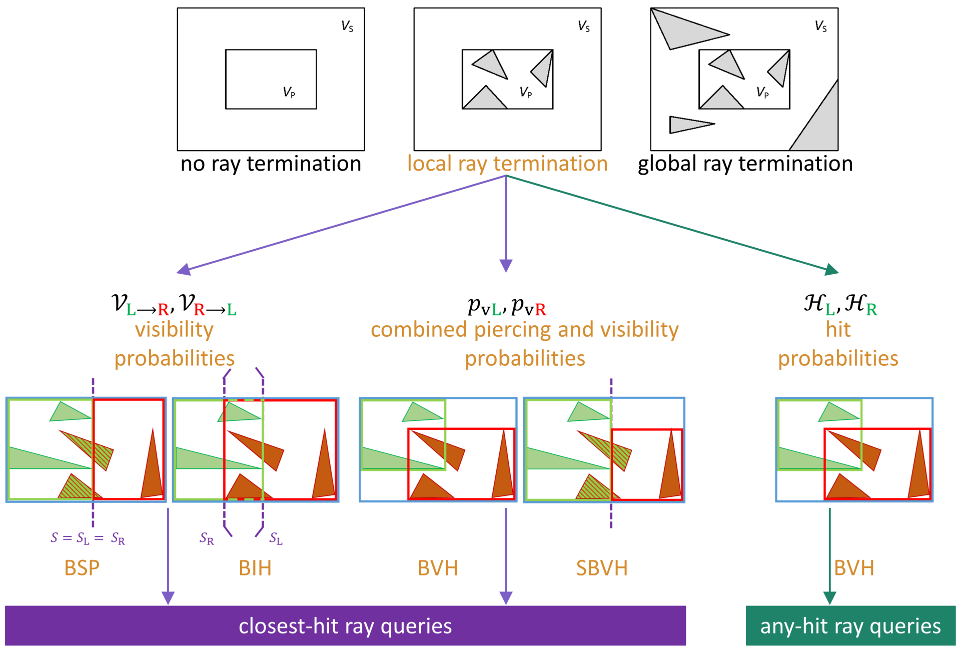
On the Use of Local Ray Termination for Efficiently Constructing Qualitative BSPs, BIHs and (S)BVHs

Design of the build heuristic framework

Matthias Moulin, Philip Dutré



Contents

A. Design

To facilitate the integration and experimentation with a wide variety of heuristics at different levels of the ADS construction, we use a very modular design and focus on extensibility and maintainability as opposed to performance. In practice, however, one typically relies on a restricted number of ADSs constructed with one of few different and heavily optimized build heuristics. Therefore, the different abstractions are better removed and collapsed in these settings once a decision regarding the build heuristic is made. In this auxiliary report, we give a broad overview of our design focussing on ADSs for accelerating closest-hit ray queries and their build heuristics. ADSs for accelerating any-hit ray queries and their build heuristics can be trivially mapped to this framework as well.

A.1. ADS build

The ADS build receives a set of geometric primitives (and additional precomputed data such as geometric normals, surface areas, AABBs, etc.) and constructs a tree ADS in $\mathcal{O}(|G_{\text{S}}| \log |G_{\text{S}}|) \left(\text{or } \mathcal{O}\Big(|G_{\text{S}}| \left(\log |G_{\text{S}}|\right)^2\Big)\right)$ time, with $|G_{\text{S}}|$ the total number of geometric primitives to partition. At least one `Partitioner` is used to obtain (i) the optimal candidate partition of a voxel, (ii) the associated cost in case of partitioning that voxel (i.e. creating an interior node) and (iii) the candidate-partition independent cost in case of not partitioning that voxel (i.e. creating a leaf node). Based on both costs and some additional criteria (to allow locally suboptimal candidate partitions) [HB02,PH10], a decision is made whether the current voxel is partitioned into child voxels or not.

We support the following ADSs: kd-trees [BF79,Kap85], BIHs[1] [OMR87,HHS06,WMS06,WK06], BVHs [RW80], GK-BVHs [PGDS09] and SBVHs [SFD09].

A.2. Partitioner

A *Partitioner* receives a voxel, V_{P}, that needs to be partitioned and returns (i) the optimal candidate partition, (ii) the associated cost in case of partitioning and (iii) the candidate-partition independent cost in case of not partitioning in $\mathcal{O}(|G_{\text{P}}|)$ (or $\mathcal{O}(|G_{\text{P}}| \log |G_{\text{P}}|)$) time.

Binned Partitioner. The `Binned Partitioner` considers a finite set of candidate splitting planes. A `Partition Strategy` is used for creating the different candidate partitions (uniquely determined by the candidate splitting planes) and a `Cost Strategy` is used for obtaining the associated costs of these partitions. `Binned Partitioners` differ in (i) the number of candidate splitting planes, (ii) the domain of candidate splitting planes (e.g., spatial extent of the voxel versus centroid extent of the geometric primitives contained in the voxel, one versus three primary axes, etc.), and (iii) the selection of candidate splitting planes on their domain (e.g., equidistant, adaptive, etc.). In our experiments, we use a constant number ($\mathcal{B} = 256$) of equidistant candidate splitting planes on the spatial extent of the voxel (kd-trees and GK-BVHs) or centroid extent of the geometric primitives contained in the voxel (BIHs and BVHs), for every axis. Furthermore, we explicitly consider a candidate partition with an empty left and a candidate partition with an empty right child voxel, for every axis, for BIHs to favour large empty space cut-offs.

Kd-trees, BIHs and BVHs can be constructed with all types of `Partitioners`, GK-BVHs and SBVHs need to be constructed with `Binned Partitioners` to ensure a $\mathcal{O}(|G_{\text{P}}|)$ time. SBVHs are constructed with a special `Binned Partitioner` which internally uses both a `Binned Partitioner` for BVHs and GK-BVHs.

Sweeping Plane Partitioner. Unlike `Binned Partitioners` which delegate the partition construction ($\mathcal{O}(|G_{\text{P}}|)$) and cost evaluation ($\mathcal{O}(|G_{\text{P}}|)$) for each candidate partition to a `Partition Strategy` and `Cost Strategy`, respectively, `Sweeping Plane Partitioners` incrementally update the candidate partition and associated cost in

[1]Unlike Wächter and Keller [WK06], we build BIHs by explicitly considering multiple (locally adapted) candidate partitions for every voxel to partition, and a cost metric such as the SAH or RTSAH to chose among these candidate partitions.

$\mathcal{O}(1)$ time while sweeping the splitting plane over $\mathcal{O}(|G_{\mathrm{P}}|)$ candidate splitting planes.[2] `Binned Partitioners` differ in the domain of candidate splitting planes (e.g., spatial extent of the voxel versus centroid extent of the geometric primitives contained in the voxel, one versus three primary axes, etc.).

Hybrid Partitioner. `Hybrid Partitioners` use both a `Binned Partitioner` and `Sweeping Plane Partitioner`. When $|G_{\mathrm{P}}| \leq 2\mathcal{B}$ in case of using the spatial extent of the voxel, V_{P}, (or $|G_{\mathrm{P}}| \leq \mathcal{B}$ in case of using the centroid extent of the geometric primitives contained in the voxel, V_{P}) the `Sweeping Plane Partitioner` is used. Otherwise, the `Binned Partitioner` is used.

Single Partitioner. `Single Partitioners` consider only one valid candidate partition for each voxel to partition (e.g., spatial median splitting [Kap85], object median splitting [Hec82]).

A.3. Partition Strategy

A *Partition Strategy* receives a voxel, V_{P}, that needs to be partitioned and a candidate splitting plane, and returns the associated candidate partition. A candidate partition for a voxel consists of all (two) child voxels (i.e. the AABB constituting the voxel and the geometric primitives contained in that voxel) in $\mathcal{O}(|G_{\mathrm{P}}|)$ time.[3]

Kd-tree Partition Strategy. Geometric primitives (i–ii) whose AABB is to the left (right) of the splitting plane belong to the left (right) child voxel, (iii) whose AABB is straddling the splitting plane belong to both child voxels. The AABBs of both child voxels can be trivially computed given the AABB of the parent voxel. The spatial union of the AABBs of both child voxels is equal to the AABB of the parent voxel (none of the six surrounding planes is tight).

All candidate partitions obtained with a `Kd-tree Partition Strategy` are valid. Child voxels are not required to contain at least one geometric primitive.

Kd-tree Partition Strategy with Split Clipping. Geometric primitives (i–ii) to the left (right) of the splitting plane belong to the left (right) child voxel, (iii) straddling the splitting plane belong to both child voxels. Geometric primitives (iv) lying outside the parent voxel belong to none of the child voxels. Note that Kd-tree `Partition Strategies with Split Clipping` clip the geometric primitives [HB02] as opposed to `Kd-tree Partition Strategies` which solely rely on the AABB of the geometric primitives. The AABBs of both child voxels can be trivially computed given the AABB of the parent voxel. The spatial union of the AABBs of both child voxels is equal to the AABB of the parent voxel (none of the six surrounding planes is tight).

All candidate partitions obtained with a `Kd-tree Partition Strategy with Split Clipping` are valid. Child voxels are not required to contain at least one geometric primitive.

BIH Partition Strategy. Geometric primitives (i–ii) whose centroid is to the left (right) of the splitting plane belong to the left (right) child voxel, and (iii) whose centroid lies on the splitting plane are added to the left child voxel. The AABBs of both child voxels are similar to those of kd-trees except that the AABB planes corresponding to the splitting planes are made tight to the geometric primitives.

Originally, candidate partitions obtained with a `BIH Partition Strategy` were only valid if both child voxels contained at least one geometric primitive. The BIHs constructed in this way, however, generally resulted in a larger number of ray-triangle intersection tests and ADS node traversal steps per ray compared to kd-trees, and offered no real advantages over kd-trees or BVHs. Therefore, we allowed one child voxel to be empty if and

[2]If multiple splitting planes coincide, the candidate partitions considered by a `Sweeping Plane Partitioner` will be different from the corresponding candidate partitions obtained after using a `Partition Strategy` for these splitting planes. The former candidate partitions will differ from one another, whereas the latter candidate partitions will all be equal to each other. To be consistent, it is possible to look ahead while sweeping the splitting plane and skip all splitting planes coinciding with the next splitting plane.

[3]SBVHs are built with a combination of a `BVH Partition Strategy` and a `GK-BVH Partition Strategy`.

only if the AABB of the other child voxel is strictly smaller than the AABB of the parent voxel. This generally results in better ray tracing performance.

BVH Partition Strategy. Geometric primitives (i–ii) whose centroid is to the left (right) of the splitting plane belong to the left (right) child voxel, and (iii) whose centroid lies on the splitting plane are added to the left child voxel. The AABBs of both child voxels are made tight to the geometric primitives. Note that the AABBs of the child voxels may not be larger than the AABB of the parent voxel, which can occur when using the `BVH Partition Strategy` for constructing SBVHs. In this case, only the overlap with the AABB of the parent voxel will be used.

Candidate partitions obtained with a `BVH Partition Strategy` are only valid if both child voxels contain at least one geometric primitive.

GK-BVH Partition Strategy. Geometric primitives (i–ii) to the left (right) of the splitting plane belong to the left (right) child voxel, (iii) straddling the splitting plane belong to both child voxels. Geometric primitives (iv) lying outside the parent voxel belong to none of the child voxels. The AABBs of the child voxels are made tight to the geometric primitives, but the plane corresponding to the splitting plane can only be moved to the left (right) for the left (right) child voxel. So the geometric primitives first need to be clipped against the splitting plane. Furthermore, only the overlap with the AABB of the parent voxel will be used when geometric primitives straddle the AABB of the parent voxel.

All candidate partitions obtained with a `GK-BVH Partition Strategy` are valid. Child voxels are not required to contain at least one geometric primitive.

A.4. Cost Strategy

A *Cost Strategy* receives a candidate partition and returns its cost according to some cost metric. Interesting cost metrics consider both the AABB constituting the child voxels and the geometric primitives contained in the child voxels in $\mathcal{O}(|G_{\mathrm{P}}|)$ time. With regard to the latter, only the number of geometric primitives contained in both child voxels (e.g., SAH [GS87, MB90], SIROH [FFD09]) or the geometric primitives themselves are required (e.g., RTSAH [IH11, MBD15]).

SAH Cost Strategy. The `SAH Cost Strategy` evaluates the SAH cost function (3.1) by first obtaining the piercing probabilities (3.2)–(3.3) ($\mathcal{O}(1)$).

RTSAH for kd-trees Cost Strategy. The `RTSAH for kd-trees Cost Strategy` evaluates the RTSAH cost function (3.4) for kd-trees by first obtaining the piercing probabilities (3.5)–(3.7) ($\mathcal{O}(1)$) and the visibility probabilities (4.3)–(4.4) ($\mathcal{O}(|G_{\mathrm{P}}|)$). The latter are delegated to a `Permeability Strategy`.

RTSAH for BIHs Cost Strategy. The `RTSAH for BIHs Cost Strategy` evaluates the RTSAH cost function (3.4) for BIHs (and kd-trees) by first obtaining the piercing probabilities (3.5)–(3.8) ($\mathcal{O}(1)$) and the visibility probabilities (4.5)–(4.10) ($\mathcal{O}(|G_{\mathrm{P}}|)$). The latter are delegated to a `Permeability Strategy`.

RTSAH Cost Strategy. The `RTSAH Cost Strategy` evaluates the RTSAH cost function (3.18) by first obtaining the combined piercing and visibility probabilities (3.19)–(3.20) ($\mathcal{O}(|G_{\mathrm{P}}|)$), which are delegated to a `Permeability Transfer strategy`.

Exact RTSAH for kd-trees Cost Strategy. The `Exact RTSAH for kd-trees Cost Strategy` evaluates the RTSAH cost function (3.4) for kd-trees by first obtaining the piercing probabilities (3.5)–(3.7) ($\mathcal{O}(1)$) and the visibility probabilities (4.3)–(4.4). The latter are evaluated using ray sampling.

Exact RTSAH for BIHs Cost Strategy. The `Exact RTSAH for BIHs Cost Strategy` evaluates the RTSAH cost function (3.4) for BIHs (and kd-trees) by first obtaining the piercing probabilities (3.5)–(3.8) ($\mathcal{O}(1)$) and the visibility probabilities (4.5)–(4.10). The latter are evaluated using ray sampling.

Exact RTSAH Cost Strategy. The `Exact RTSAH Cost Strategy` evaluates the RTSAH cost function (3.18) by directly sampling the combined piercing and visibility probabilities (3.19)–(3.20).

A.5. Permeability Strategy

A *Permeability Strategy* receives a kd-tree or BIH candidate partition and returns both (plane) visibility probabilities (4.3)–(4.8) in $\mathcal{O}(|G_{\text{P}}|)$ time by using one of our **visibility heuristics**.

A.6. Permeability Transfer Strategy

A *Permeability Transfer Strategy* receives a candidate partition and returns both combined piercing and visibility probabilities (3.19)–(3.20) in $\mathcal{O}(|G_{\text{P}}|)$ time. An irregular grid constituting the candidate partition is constructed and for each irregular grid cell a $6{\times}6$ transfer matrix (4.13) is created by using one of our **visibility heuristics** in $\mathcal{O}(|G_{\text{P}}|)$ time. Next, a parent-exterior isotropic ray distribution is transferred over the grid planes according to the transfer matrices of the grid cells until no non-terminated rays remain in the grid. During this process, the accumulated fraction of rays entering through each grid plane is tracked. Finally, the accumulated fraction of rays on each of the six AABB planes of a child voxel is used to obtain the combined piercing and visibility probability for that child voxel (3.19)–(3.20).

A.7. Visibility Heuristic

The *Visibility Heuristics* are located at the deepest level of our design. For more information, we refer to our visibility probability approximations presented in the main text.

References

[BF79] Bentley J. L., Friedman J. H.: Data Structures for Range Searching. *ACM Comput. Surv. 11*, 4 (Dec 1979), 397–409.

[FFD09] Fabianowski B., Fowler C., Dingliana J.: A Cost Metric for Scene-Interior Ray Origins. In *Eurographics 2009 - Short Papers* (2009), Alliez P., Magnor M., (Eds.), The Eurographics Association.

[GS87] Goldsmith J., Salmon J.: Automatic Creation of Object Hierarchies for Ray Tracing. *IEEE Computer Graphics and Applications 7*, 5 (May 1987), 14–20.

[HB02] Havran V., Bittner J.: On Improving Kd-Trees for Ray Shooting. In *Proceedings of Winter School of Computer Graphics 2002 Conference* (2002), pp. 209–217.

[Hec82] Heckbert P.: Color Image Quantization for Frame Buffer Display. *SIGGRAPH Comput. Graph. 16*, 3 (Jul 1982), 297–307.

[HHS06] Havran V., Herzog R., Seidel H.-P.: On the Fast Construction of Spatial Hierarchies for Ray Tracing. In *IEEE Symposium on Interactive Ray Tracing 2006* (Sept 2006), pp. 71–80.

[IH11] Ize T., Hansen C.: RTSAH Traversal Order for Occlusion Rays. *Computer Graphics Forum 30*, 2 (2011), 297–305.

[Kap85] Kaplan M. R.: The Use of Spatial Coherence in Ray Tracing. *ACM SIGGRAPH Course Notes 11* (1985).

[MB90] MacDonald D. J., Booth K. S.: Heuristics for Ray Tracing Using Space Subdivision. *Vis. Comput. 6*, 3 (May 1990), 153–166.

[MBD15] Moulin M., Billen N., Dutré P.: Efficient Visibility Heuristics for Kd-Trees Using the RTSAH. In *Eurographics Symposium on Rendering - Experimental Ideas & Implementations* (June 2015), Lehtinen J., Nowrouzezahrai D., (Eds.), The Eurographics Association, pp. 31–39.

[OMR87] Ooi B. C., McDonell K. J., Ron S.-D.: Spatial Kd-Tree: An Indexing Mechanism for Spatial Databases. In *Proceedings of the IEEE COMPSAC Conference* (1987).

[PGDS09] Popov S., Georgiev I., Dimov R., Slusallek P.: Object Partitioning Considered Harmful: Space Subdivision for BVHs. In *Proceedings of the Conference on High Performance Graphics 2009* (New York, NY, USA, 2009), HPG '09, ACM, pp. 15–22.

[PH10] Pharr M., Humphreys G.: *Physically Based Rendering, Second Edition: From Theory To Implementation*, 2nd ed. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2010.

[RW80] Rubin S. M., Whitted T.: A 3-dimensional Representation for Fast Rendering of Complex Scenes. *SIGGRAPH Comput. Graph. 14*, 3 (Jul 1980), 110–116.

[SFD09] Stich M., Friedrich H., Dietrich A.: Spatial Splits in Bounding Volume Hierarchies. In *Proceedings of the Conference on High Performance Graphics 2009* (New York, NY, USA, 2009), HPG '09, ACM, pp. 7–13.

[WK06] Wächter C., Keller A.: Instant Ray Tracing: The Bounding Interval Hierarchy. In *Proceedings of the 17th Eurographics Conference on Rendering Techniques* (Aire-la-Ville, Switzerland, Switzerland, 2006), EGSR '06, Eurographics Association, pp. 139–149.

[WMS06] Woop S., Marmitt G., Slusallek P.: B-Kd Trees for Hardware Accelerated Ray Tracing of Dynamic Scenes. In *Proceedings of the 21st ACM SIGGRAPH/EUROGRAPHICS Symposium on Graphics Hardware* (New York, NY, USA, 2006), GH '06, ACM, pp. 67–77.