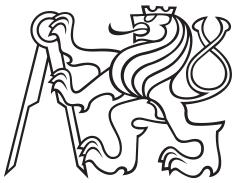
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Computer Graphics and Interaction

# Bounding Volume Hierarchies for Oblong Objects

**Emese Szabó**

**Supervisor: doc. Ing. Jiří Bittner, Ph.D.**
**Field of study: Open Informatics**
**Subfield: Computer Graphics**
**August 2021**

# ZADÁNÍ DIPLOMOVÉ PRÁCE

## I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Szabó** Jméno: **Emese** Osobní číslo: **469879**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Specializace: **Počítačová grafika**

## II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

**Hierarchie obalových těles pro podlouhlé objekty**

Název diplomové práce anglicky:

**Bounding Volume Hierarchies for Oblong Objects**

Pokyny pro vypracování:

Zmapujte metody stavby hierarchií obalových těles (BVH). Navrhněte úpravu existující metody PLOC [1] pro použití obalových tělech vhodných pro reprezentaci podlouhlých tenkých objektů jako jsou vlasy nebo paprsky. Soustřeďte se na efektivní výpočet sjednocení navržených obalových těles vhodných pro tento typ objektů. Navrženou metodu implementujte v jazyce C++ a důkladně otestujte z hlediska cenového modelu a rychlosti zobrazování pomocí metody sledování paprsku. Testy realizujte na nejméně pěti scénách různého typu (vlasy, postavy, architektura).

Seznam doporučené literatury:

[1] Meister, D., Bittner, J. 'Parallel Locally-Ordered Clustering for Bounding Volume Hierarchy Construction,' in IEEE Transactions on Visualization and Computer Graphics, vol. 24, no. 3, pp. 1345-1353, (2018).
[2] Fonseca, R., Winter, P. Bounding Volumes for Proteins: A Comparative Study Journal of Computational Biology 19(10), 1203-1213, (2012).
[3] Woop, S., Benthin, C., Wald, I., Johnson, G. S., Tabellion, E. Exploiting Local Orientation Similarity for Efficient Ray Traversal of Hair and Fur. High Performance Graphics, (2014).
[4] Sun, X., Zhou, K., Lin, S., Guo, B.. Line space gathering for single scattering in large scenes. Acm Transactions Graph Tog 29, 4 (2010).

Jméno a pracoviště vedoucí(ho) diplomové práce:

**doc. Ing. Jiří Bittner, Ph.D., Katedra počítačové grafiky a interakce**

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **11.02.2021** Termín odevzdání diplomové práce: **13.08.2021**

Platnost zadání diplomové práce: **30.09.2022**

| doc. Ing. Jiří Bittner, Ph.D.<br>podpis vedoucí(ho) práce | podpis vedoucí(ho) ústavu/katedry | prof. Mgr. Petr Páta, Ph.D.<br>podpis děkana(ky) |
|---|---|---|

## III. PŘEVZETÍ ZADÁNÍ

Diplomantka bere na vědomí, že je povinna vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. Datum převzetí zadání

Podpis studentky


© ČVUT v Praze, Design: ČVUT v Praze, VIC

# Acknowledgements

I would like to express my gratitude to my supervisor Jiří Bittner for his helpful comments, corrections and encouraging words. I would also like to thank my family, friends and my dear boyfriend, who are all very unlikely to ever read this thesis, but I am still grateful for the support, kind words, and not bothering and letting me do what I had to.


# Declaration

I declare that I have created the presented thesis independently and that I have quoted all used sources of information in accordance with the Methodical instructions about ethical principles for writing academic theses.

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Prague, August 13, 2021

# Abstract

This thesis discusses using cylinders as bounding volumes in ray tracing for specific scene types containing oblong objects. It offers a method for joining cylinder-shaped bounding volumes and shows their advantages. The presented method incorporates cylinders into a state-of-the-art bottom-up hierarchy builder, a locally-ordered clustering algorithm. The provided implementation is a proof of concept. Three types of bounding volume hierarchies were compared, the common axis-aligned bounding box (AABB) hierarchy, a cylinder version, and a hybrid. The obtained results show that cylinder volumes have the potential to reduce the overall surface area of the hierarchy for oblong objects. Moreover, the hybrid hierarchy outperforms the AABB version in rendering certain scenes while increasing the build times.



**Keywords:** bounding volumes, bounding volume hierarchies, ray tracing, oblong objects, cylinder



**Supervisor:** doc. Ing. Jiří Bittner, Ph.D.
Department of Computer Graphics and Interaction,
Faculty of Electrical Engineering,
CTU in Prague



# Abstrakt

Práce pojednává použití válců jako obalová tělesa při sledování paprsku pro různé typy scén obsahující podlouhlé objekty. Představuje způsob sjednocení obalových těles ve tvaru válce a zkoumá jejich výhody. Navrhnutá metoda používá válce v rámci konkrétního algoritmu pro stavbu hierarchie obálek. Poskytnutá implementace je ověřením konceptu. Byly porovnány tři typy hierarchií: hierarchie osově zarovnaných obálek, verze používající válce a hybridní varianta kombinující obě typy obálek. Získané výsledky ukazují, že válce mají potenciál zmenšit celkový povrch hierarchie pro podlouhlé objekty. Hybridní hierarchie překonává verzi s osově zarovnanými obálkami při syntéze obrazu v případě určitých scén, ale zároveň výrazně zvyšuje dobu postavení hierarchie.



**Klíčová slova:** obalová tělesa, hierarchie obalových těles, sledování paprsků, podlouhlé objekty, válec

**Překlad názvu:** Hierarchie obalových těles pro podlouhlé objekty

# Contents

# Figures

# Tables

# Chapter 1

# Introduction

Ray tracing is a key element of rendering algorithms that simulate realistic light propagation. For each ray, their closest intersection with the scene is computed. To achieve a convincing image, millions of rays have to be traced in the process, with billions of intersection tests carried out. To enhance ray traversal performance, state-of-the-art methods use bounding volume hierarchies (BVHs). They are tree-like data structures containing the bounding volume of the whole subtree of each node, with primitives contained in leaf nodes. These data structures have been proven to be very efficient in rendering, as they accelerate the process of finding the closest intersection with the scene.

BVHs wrap parts of the scene into tight bounding volumes. The most commonly used bounding volume type is an axis-aligned bounding box (AABB). However, this bounding box is known to perform poorly on long, thin, and especially diagonal objects, namely curves, fur, and hair. To overcome this deficiency, oriented bounding boxes (OBBs) may be used instead, which perform better on thin or diagonally oriented primitives under certain circumstances. Wald et al. [4] proposed an enhancement for hardware BVH traversal on NVIDIA's Turing architecture. They trick the hardware into performing a hardware-accelerated OBB intersection test before the user-defined intersection test with an AABB. They demonstrate a speedup of up to 5.9× compared to a standard hardware-accelerated BVH [4]. Woop et al. [1] exploit the orientation similarity between individual hairs in scenes with densely packed strands. They discuss the various bounding options for thin, curved hair strands, use AABBs in combination with OBBs, and achieve 2× better performance than with BVHs using purely axis-aligned boxes.

Fonseca and Winter [5] discuss different types of bounding volumes for protein chains, which are essentially small objects aligned along curves. They wanted to detect clashes of subchains (long, thin, slightly curved pieces) in proteins as precisely as possible, hence the need for tight bounding volumes.

## 1.1 Goals of the thesis

Solids of revolution qualify as non-standard shapes since their use in BVHs is unprecedented. The reason for avoiding these shapes is simple: they are not memory-efficient and demand more computationally expensive intersection tests. Moreover, calculating the joint bounding volume of two such shapes is not straightforward. However, these circular shapes could provide tighter bounds for thin, long objects, such as hair strands and fur. This idea builds on the conclusions of the studies mentioned above, exploiting similar orientation of primitives, wrapping diagonal objects with oriented volumes, and reducing the surface area of the whole hierarchy by reducing redundancy in a BVH introduced by overlapping AABB-nodes.

In order to make ray tracing as fast as possible, BVH construction should be relatively fast and efficient. Bittner and Meister [2] introduced a parallel locally-ordered clustering (PLOC) algorithm for BVH construction. This algorithm is highly parallel and is able to utilize many cores of current GPUs. This thesis aimed at constructing a functional variant of the PLOC algorithm using cylinders as bounding volumes. The results show that cylinders are the most advantageous when used in combination with AABBs, resulting in hybrid hierarchies. These are shallower structures and have lower surface areas than the purely AABB hierarchy, though they are only suitable for certain scene types.

## 1.2 Thesis structure

Chapter 2 offers a summary of related work. Chapter 3 elaborates on the considered bounding volume types, while chapter 4 discusses methods for computing the joint bounding volume of volumes. Chapter 5 describes the hybrid hierarchy idea. Then, chapter 6 details the implementation of the proposed algorithm, and chapter 7 discusses the obtained results. Finally, chapter 8 concludes the thesis.

# Chapter 2

# Related work

One of the earliest bounding volume hierarchies used was constructed manually by Rubin and Whitted [6]. The first BVH construction algorithm that was able to build a hierarchy in a top-down manner was published by Kay and Kajiya [7]. Various other builders and improvements followed. A well-known quality metric, the surface area heuristic (SAH) function was introduced by Goldsmith and Salmon [8]. This heuristic function is used to influence the construction decisions to obtain a higher quality BVH. Lower node overlap and a lower overall surface area are preferred.

## 2.1 BVH improvements

Aila et al. [9] showed that the final SAH cost of a hierarchy does not necessarily correlate with the ray tracing speed of the resulting BVH, and they introduced new metrics that exhibit better prediction qualities. The SAH function was further used by Stich et al. [10] and Popov et al. [11]; while the former algorithm uses the SAH as the decision criterion for choosing a good split position and as a possible terminal condition, the latter introduces a penalty for node overlap enforcing strict space division. However, along with Ernst and Greiner [12], both take advantage of spatial splitting to utilize the benefit of spatial and object hierarchies alike. Wald [13] used a SAH cost estimation based on binning, essentially discretizing the cost function. Top-down hierarchy builders start with a large bounding box of the whole scene and often use some form of the SAH to make an informed decision when

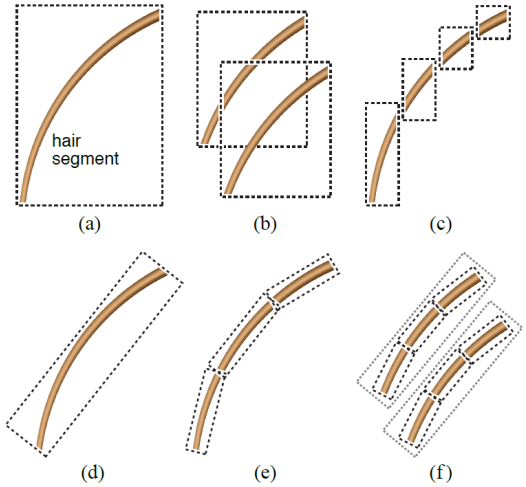



**Figure 2.1:** Illustration of the different options of bounding hairs and hair segments with rectangular shapes by Woop et al. [1]. (a) and (b) depict the situation when a diagonal strand is wrapped with an AABB. (c) the strand split into segments and wrapped with AABBs. (d) OBB of the same hair, and OBBs of hair segments (e). The OBB-wrapped segments produce less overlap (f).

splitting a node. The bounding volume surface area may be used similarly in bottom-up algorithms.

Wald et al. [4] proposed an enhancement for hardware BVH traversal on NVIDIA's Turing architecture. They trick the hardware into performing a hardware-accelerated OBB intersection test before the user-defined intersection test with an AABB. They demonstrate a speedup of up to 5.9x compared to a standard hardware-accelerated BVH [4]. Woop et al. [1] worked with scenes containing densely packed hair strands. To accurately represent hair curvature, the strands must be finely tessellated, which often yields primitives smaller than a pixel, which may cause aliasing issues. Thus, high sampling rates are required to avoid aliasing, which means a large number of traced rays [1]. For this reason, they aimed to reduce the cost of traversing a ray by exploiting the orientation similarity of neighboring hair strands. Using a small number of OBBs to wrap each hair by parts minimizes node overlap and allows for more efficient spatial data structures to be built over them.

## 2.2 Protein chains

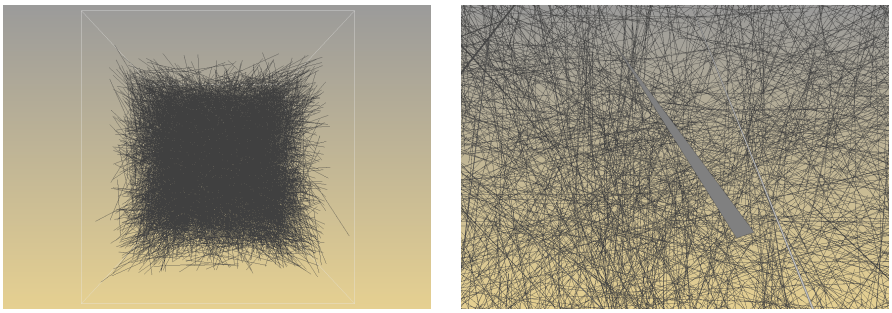
Fonseca and Winter [5] discuss the use of bounding volumes in computational molecular biology. Namely, they wanted to detect clashes of subchains in

protein chain trees. The chain tree is a balanced binary tree that stores protein chain bonds in leaves and a whole subchain in internal nodes. The internal nodes have an associated bounding volume of their subtree. A clash only occurs if the bounding volumes of two subchains intersect, requiring tight volumes with efficient intersection tests and the ability to compute a tight bounding volume of two smaller volumes.

They concluded that for their chain tree structure, OBBs performed worse than expected. Spheres outperformed OBBs and line swept spheres (capsules), which provide a sufficiently tight bound when enclosing smaller bounding volumes into one. The *efficient union* of certain bounding boxes will be discussed later in section 4.

## 2.3 Ray gathering

Long thin objects resembling hair strands may also be randomly distributed rays in a scene. Sun et al. [14] describe an algorithm that evaluates radiance along a viewing ray by searching for nearby lighting rays in the scene. They focus on rendering single scattering in large complex scenes that contain homogeneous participating media and refractive and reflective materials. They approximate lighting and viewing rays by the infinite line they lie on, and use them as 6D points and planes in parametric space by computing their Plücker coordinates [14]. The task of building a spatial structure over randomly distributed rays bears some resemblance to constructing a BVH over hair strands, as rays may be represented by long triangles with nearly zero surface area (see Figure 2.2).



**(a) :** 10k randomly generated rays **(b) :** Close-up of the thin triangles

**Figure 2.2:** Screenshot of an artificial scene with 10K randomly generated rays - each ray represented by a single long triangle with a close to zero surface area ($\sim 0.002$). The triangle length is approximately 30% of the diagonal of the scene's bounding box.

## 2.4 Parallel Locally-Ordered Clustering (PLOC)

Designed by Bittner and Meister [2], the algorithm operates by performing clustering in parallel and employing sorting of primitives by the Morton codes [15] of clusters to enhance the search for nearest neighbors.

The agglomerative clustering starts with $n$ trivial clusters, each cluster containing one triangle representing the leaves of the resulting tree. The clusters are sorted by the Morton codes of their centroids. The nearest neighbor search is a range search along the Morton curve for each cluster $i$ in the range $\langle i - r, i + r \rangle$, where $r$ is the search radius (for an illustration, see Figure 2.3). They define a distance metric according to which the nearest neighbor search operates. The distance between two clusters is defined as the surface area of the axis-aligned bounding box tightly enclosing the two clusters [2]. According to this metric, two clusters are only merged if they are the nearest neighbors of each other since there will not be found any better neighbor for either of them. Although they note that this nearest neighbor search can only find an approximate nearest neighbor, they deem this performance sufficient.

After the nearest neighbor search, they perform the merging step in parallel on all mergeable pairs. Merging constitutes of replacing one of the merged clusters by a new parent node in the tree, and discarding the other cluster. Iterations continue until there is only one cluster left containing the whole scene (see Figure 2.4). They guarantee algorithm termination by prioritizing the nearest neighbor with the lowest index in the rare case that two neighbors are at the same distance from a cluster [2].

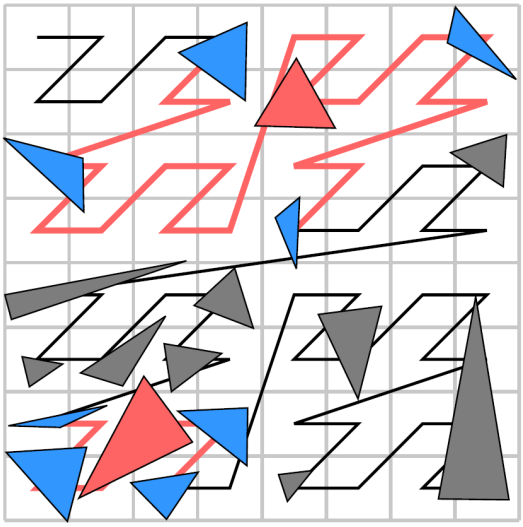


**Figure 2.3:** 2D example of primitives ordered along the Morton curve with the search radius $r = 2$ (image source: [2]). The two clusters (red triangles) search for their nearest neighbors (blue triangles). This order adapts well to the primitive density of the scene, as noted by the authors.

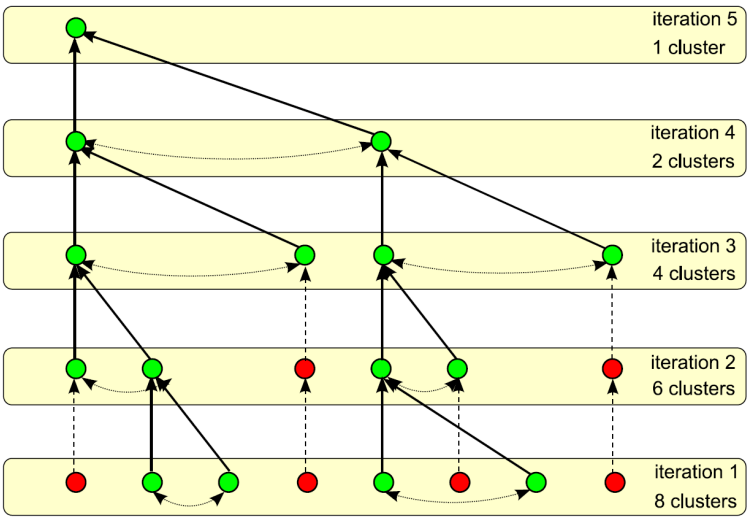



**Figure 2.4:** An illustration of clustering iterations (image source: [2]). The cluster pairs connected with two-way arrows are mutually corresponding nearest neighbors; thus. they are merged into one. Iterations continue until a single cluster remains containing the whole scene - until iteration 5 in this example.

# Chapter 3

# Bounding volumes

Selecting the most suitable bounding volume type is a crucial part of building an efficient acceleration structure. Various bounding volumes exhibit distinct traits, making them appropriate for different tasks. The most commonly used volume is an axis-aligned bounding box (AABB). An AABB is easy to implement because it only requires computing the extent of an object in the direction of the three principal axes. It is memory efficient since it requires storing only six values to determine the bounding box's position and size in the scene (minima and maxima on the principal axes). However, it does not produce a tight bound for objects that are longer in one direction, especially when they are rotated around one of the principal axes. For a rotated object, the AABB will partially overlap with other objects or enclose a large amount of extra scene space along with the object, forcing the algorithm to carry out unnecessary intersection tests.

To overcome this efficacy issue, an oriented bounding box (OBB) might be used instead. Although an OBB offers a tighter bound for an object, it is less efficient memory-wise. However, an OBB might not be tight enough for long and slightly curved shapes since the bounding volume will also enclose the space inside the curve. A possible solution may be to wrap parts of the primitive separately to achieve a tighter bound.

This work focuses on thin hair-like objects and explores the efficiency of non-standard bounding volumes, such as cylinders, truncated cones, and line swept spheres (capsules). In particular, emphasizing the potential of the volume to enclose two volumes of the same type efficiently.

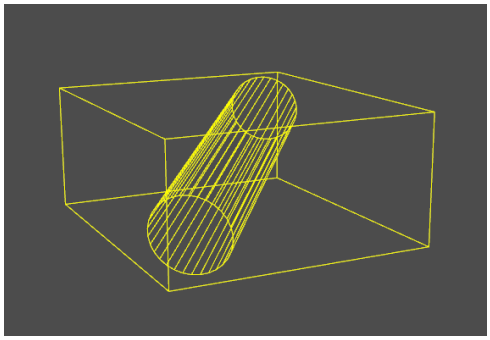


**Figure 3.1:** Axis-aligned bounding box of a cylinder.

Since hair and fur are inherently cylinder-like shapes, a cylinder-shaped bounding volume might offer the tightest bound, assuming that the object is not curved or otherwise deformed. However, considering the fact that most scenes are triangle meshes, the curved hair strands will likely be represented by triangles ordered into cylinders; thus, they may be wrapped segment by segment. In case the object has a varying radius along its length, a truncated cone might outperform a cylinder. A cylinder is a special case of a truncated cone because it has identical radii at both ends. Thus cylinders may also be assumed to be truncated cones. These bounding volumes require additional information to be stored (a point and a vector for the axis, one or two radii, and height).

Line swept spheres (LSS) are similar to cylinders, but they have a hemisphere at both of their ends instead of a flat cap. This means more straightforward intersection tests since every point on the LSS surface is exactly at the distance of the radius from the centerline, including the line's endpoints.

However, it should be easy to wrap every bounding volume around any object. For an AABB, this is straightforward, requiring only three minimum-maximum pairs. For an OBB, the bounding volume is oriented along the object axes. For a cylinder, it suffices to determine an object's central axis and then the perpendicularly farthest point from the axis (radius) and the two most distant points that lie on the axis (height). For an LSS, it might be more challenging to determine the correct volume length (specifically, the length of the centerline) because the hemisphere cap might not be a tight fit for the object.

Another essential property of any chosen bounding volume is to yield a tight bound for two smaller volumes of the same type (inside a BVH). The question is if the larger cylinder (or truncated cone) on the next level gives a tighter bound for two volumes than an AABB (or OBB). In theory, there might exist a BVH level, from which it is more efficient to switch to simpler

bounding volumes, such as AABB, to reduce computational overhead and make the intersection test faster.

## 3.1 Axis-aligned bounding box

An axis-aligned bounding box is the simplest and most widely used bounding volume. It is memory efficient as it may be represented by only six values, two on each axis. Thus, an AABB is computed by finding the minimum and maximum values covered on each axis according to the wrapped object's extent. This might be a rather complicated task for primary scene geometry, but since most scenes are triangular meshes, it poses no problem in practice. Wrapping AABBs by another AABB is very efficient, both time and memory-wise, making it the superior choice for building BVHs.

## 3.2 Oriented bounding box

An oriented bounding box (OBB) is very similar to an AABB, as it has the same form of a rectangular prism. An OBB extends along the axes of the object's coordinate system with the origin at the center of mass.

OBBs are sometimes used in ray tracing as a tighter alternative instead of an AABB or in combination with AABBs. Wald et al. [4] proposed an algorithm that forces hardware-accelerated OBB intersection tests before carrying out the user-defined test. They pointed at the problem of AABBs enclosing unnecessary empty space around diagonal objects that result in a large number of intersection tests. However, they do not build a BVH based on OBBs, but use affine transformation and instancing to carry out the hardware-accelerated tests at the object level of the BVH.

The hypothesis this project builds on is that cylinders and truncated cones could perform better in ray tracing than rectangular shapes, providing tighter bounds in certain scene types.

## 3.3 Cylinder

Intuitively, a cylinder should provide a tighter bound for cylinder-shaped objects than an axis-aligned box; moreover, it should be a better choice at the lower levels of a BVH than an oriented box. This is particularly true for scene primitives that are considerably longer along one of their axes. For such triangles, a cylinder would enclose less unnecessary space around the primitive than a rectangular volume. A cylinder can be represented by only seven values in memory - a point in space, a direction vector, the base radius, and the height of the cylinder. Memory-wise, a cylinder behaves precisely like an AABB because storing and accessing six values is generally less cache-friendly than a power of two; the AABB representation would probably be aligned to eight floating-point numbers.

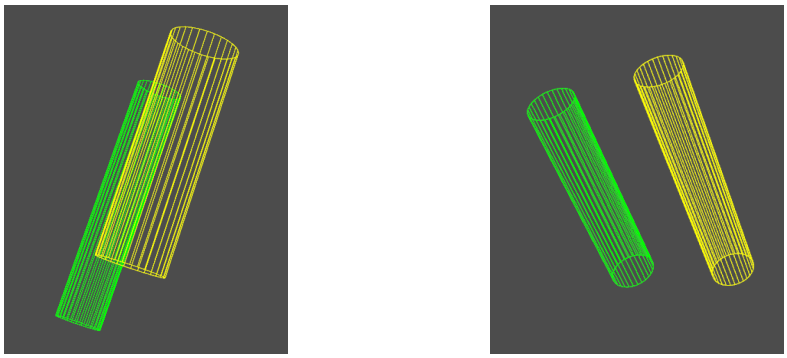


**Figure 3.2:** Similar cylinder pair examples.

## 3.4 Truncated cone

Although cylinders represent special cases of truncated cones, and they may be considered a part of the same problem, this experiment distinguishes between cones and cylinders. Truncated cones are considered due to their potential to provide a better enclosure for groups of rays or groups of hair strands or even cylinder-shaped bounding volumes. However, they could also be less efficient since they might encapsulate unnecessary space around bounded objects if the difference between the two radii is significant. They might also prove less cache-friendly because of the need to store another value, the second radius; however, that may be solved by carefully arranging the data.

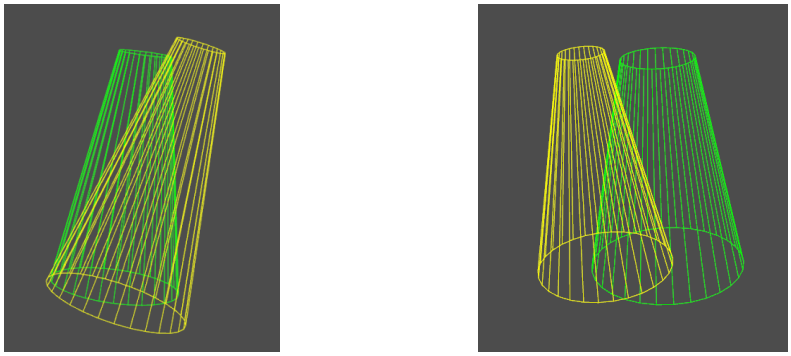


**Figure 3.3:** Similarly oriented truncated cones.

## 3.5 Line swept sphere

Line swept spheres (LSS) are capsule-like shapes that may be represented by two points in space and a radius because the surface of an LSS consists of all points exactly at the distance of the radius from the line segment defined by two points. They are more often used in collision detection ([3], [16]) than in ray tracing because they offer a fast overlap test and quick distance computation between two capsules.

Fonseca and Winter [5] discussed the usage of bounding volumes in molecular biology; they built a chain tree structure for studying changing protein conformations and used LSSs as bounding volumes to detect clashes of protein subchains. They compared LSSs to simple spheres, OBBs, and rectangular-swept spheres. They concluded that simple spheres and LSSs outperform OBBs. As protein chains are not necessarily similar to hair because of their tree-like structure, the only takeaway for this project is that LSSs might present a slight improvement over cylinders due to a faster intersection test.

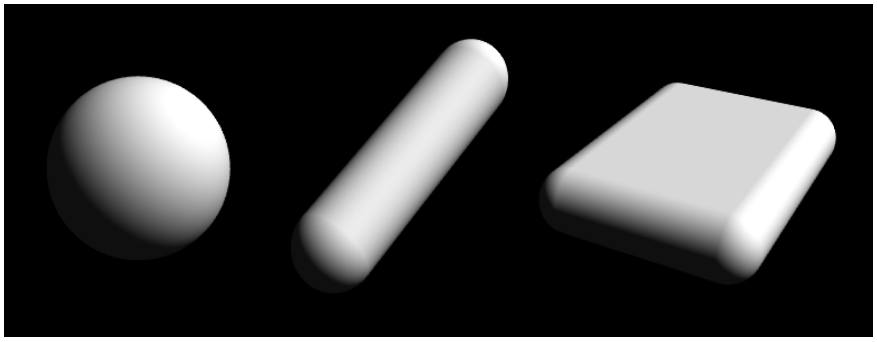
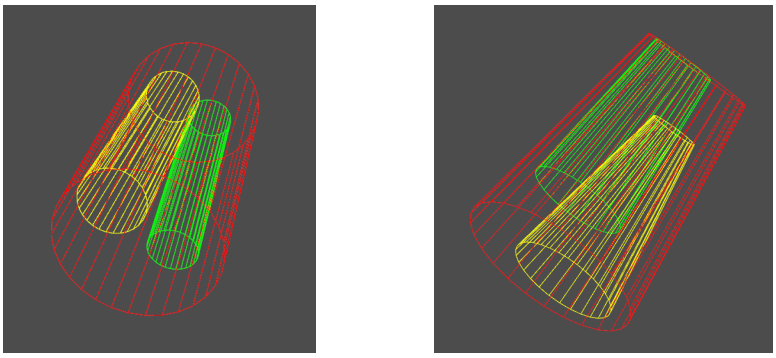


**Figure 3.4:** Renders of swept sphere shapes; from left to right: sphere, line swept sphere, rectangular swept sphere (image source: [3]).

# Chapter 4

# Bounding volume of smaller volumes

Computing the bounding volume of other volumes of the same type is straightforward for rectangular shapes. As discussed in the previous chapter, AABBs are ideal for this task; however, they produce very loose enclosures for oblong, diagonally oriented objects. The tightness of cylinder or cone-shaped bounding volumes depends on the definition of the constructed volume's axis, along with the orientation similarity of the enclosed volumes.



**(a) :** Cylinders **(b) :** Truncated cones

**Figure 4.1:** Wrapping two similar volumes (green, yellow) with a joint bounding volume of the same type (red).

## 4.1 Joining AABBs

In the context of easily computed joint bounding volumes, axis-aligned bounding boxes are far superior to any other volume. Joining two AABBs is

often realized by extending one volume by the other volume's extent, which only takes six comparisons.

## 4.2 Wrapping cylinders with a cylinder

Tightly wrapping two cylinders by a new bounding volume of the same shape is fairly complicated. An AABB has its extent precisely defined on each principal axis. However, a cylinder's extent is only known with respect to its center of base and its axis. Moreover, the arbitrary orientation of the two cylinder axes complicates the construction. Thus, the axis selection is crucial for the resulting volume to have the smallest achievable surface area. Let the cylinders be reasonably similar, with a relatively small angle and equally small distance between the two axes (as in Figure 4.2). Intuitively, to have the least amount of empty space around the wrapped objects, the new axis should be positioned somewhere *between* the two axes. A line can be defined by two points. This is true in both the context of axis construction and for the individual cylinders. Their axes are defined by their two centers of bases. The radius of the constructed cylinder will directly depend on the distance between the base centers and the selected axis (see Figure 4.3). To minimize this impact, the new axis will be defined by two points, each on one of the lines defined by a pair of centers of bases (Figure 4.2). The cylinder cap centers are paired with the closer point.

The proposed algorithm for constructing joint bounding cylinders first computes the axis of the resulting cylinder. It takes the endpoints of the axes of the two encapsulated cylinders (the centers of the cylinder bases), and it computes the middle point of the line connecting a pair of centers, yielding two points that will define the axis of the wrapping cylinder (see Figure 4.2, left). The other explored option was to find the center of mass on the line connecting the two bases using the volume of the wrapped cylinders instead of the middle point (Figure 4.2, right); the practical effect of this choice is further discussed in section 4.6.

The algorithm then establishes the height of the final cylinder by computing the perpendicular projections of the cylinder bases onto the newly formed axis, finding the projections of the farthest points of the cylinders onto this axis, and calculating the distance between the two farthest projections. The next step is to calculate the radius of the cylinder base. The cylinder should provide a tight enclosure. Thus, the radius is given as the perpendicular distance of the farthest point of the two cylinders from the new axis. Figure 4.3 illustrates the radius selection process with a given axis and one cylinder.

**Figure 4.2:** Two ways of calculating the axis of the bounding cylinder. Line defined by the middle points (left) and choosing the centers of mass (right) as if the two volumes were hanging at the ends of the connecting line (here assuming the yellow cylinder to be larger).

## 4.3 Wrapping truncated cones with a truncated cone

The algorithm essentially performs the same steps as when computing an enclosing cylinder, the main difference being finding two different radii for the upper and lower bases. It assumes two similarly oriented truncated cones, both having the smaller radius at the same end of their axis (Figure 4.1 (b)). Another critical step is to save all projected point - distance pairs while searching for the maximal radius.

The presented method generally does not provide the tightest possible enclosure for truncated cones; it slightly overestimates the radii of the bounding cone. Finding the tightest possible enclosing truncated cone is a complicated optimization problem. When the two radius candidates are found, the algorithm additionally checks the cone slope; essentially, whether the actual cone radius is larger than the distance of the projected point (the farthest point of the wrapped cone) at that height on the axis. If it is smaller, the constructed bounding volume needs to be adjusted. The nature of this adjustment has a massive impact on the tightness of the computed volume and its surface area. It is desirable to keep the surface area as low as possible, and the adjustment procedure should respect that.

As shown in Figure 4.4, there are two correction schemes involving only one radius. These methods enlarge either the top or the bottom cone radius, changing the slope angle, and as a result, changing the surface area of the volume. Another straightforward method is to equally adjust both radii by shifting the slope side away from the axis, making the volume larger while

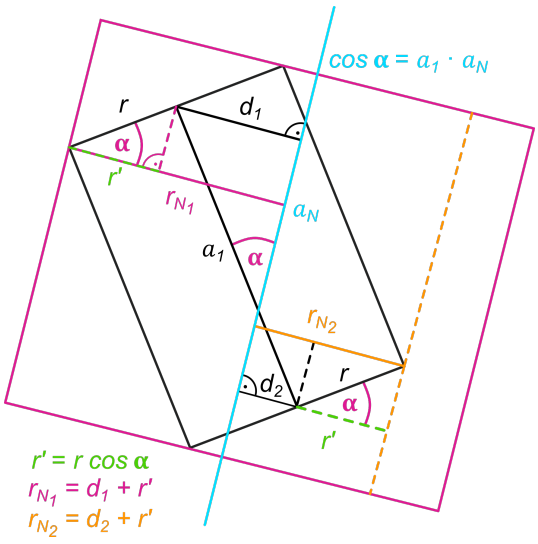


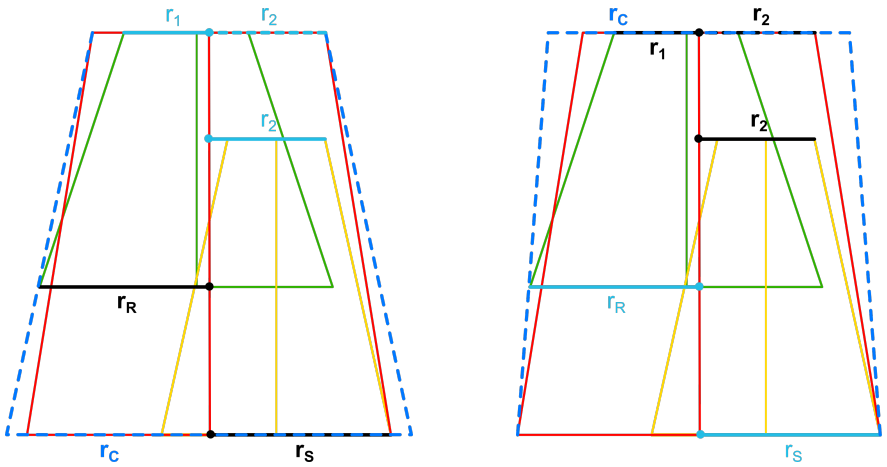

**Figure 4.3:** Calculating the new radius $r_{N_1}$ (magenta) of the bounding cylinder using the angle $\alpha$ between the cylinder axes $a_1$ and $a_N$. Distances $d_1$ and $d_2$ are the perpendicular distances of the centers of bases of the black cylinder from the axis $a_N$. $r'$ is one of the legs of the right triangle with hypotenuse $r$. The radii $r_{N_1}$ and $r_{N_2}$ (orange) are computed as the sum of the respective distance $d_i$ and $r'$. The larger radius is then selected to construct the bounding cylinder (magenta).

not affecting the side slope angle. Note that for the algorithm to yield an optimal volume, both radii would have to be adjusted, which requires solving the optimization problem of minimizing the surface area. The surface area of a truncated cone may be calculated as

$$\begin{aligned} S &= \pi r_1^2 + \pi r_2^2 + \pi(r_1 + r_2)s \\ s &= \sqrt{h^2 + (r_1 - r_2)^2} \end{aligned} \tag{4.1}$$

where $s$ is the length of the sloping side of the truncated cone, computed as the hypotenuse of the right triangle with legs $h$ along the axis and $|r_1 - r_2|$ at the bottom. By increasing a radius, the value of $S$ will also increase. However, by increasing the smaller radius, the resulting volume will be closer to a cylinder by making its sloping side steeper - this also means that in Equation 4.1 the length of $s$ will decrease.

However, practice does not support this theory. Systematically choosing to adjust the smaller radius in the current form is by no means superior to adjusting the other radius. Choosing to adjust only the smaller radius yields worse average surfaces than selecting the larger or correcting both radii. The average relative surface areas of the enclosing volume are approximately




**(a) :** Correcting the larger radius **(b) :** Adjusting the smaller radius

**Figure 4.4:** Illustrating the problematic radius selection for the red enclosing truncated cone. In both cases, the dashed blue shape represents the cone after the radius correction. (a) $r_S$ is the selected larger radius calculated from the yellow cone's farthest point from the axis, while $r_R$ is the rejected distance (it is shorter than $r_S$). The selected radius correctly represents the farthest wrapped point from the axis at that height; however, this selection does not automatically respect the distance $r_R$ at a different height along the axis. (b) Analogically, $r_2$ is selected as the radius, but the green volume is not fully enclosed by the resulting red volume. By increasing the radius and making the slope steeper the bounding volume provides a sufficiently tight bound for both truncated cones.

the same for the larger radius and the double adjustment after thousands of generated instances (see Table 4.1). For further measurements and comparison, the correction of the larger radius was chosen, as it seemed to be the most consistent improvement.

| **Comparing radius corrections** | | | | | | |
|---|---|---|---|---|---|---|
| **surface average** | very similar | | similar | | divergent | |
| **10K instances** | $\frac{S_C}{S_1+S_2}$ | $\frac{S_C}{S_{AB}}$ | $\frac{S_C}{S_1+S_2}$ | $\frac{S_C}{S_{AB}}$ | $\frac{S_C}{S_1+S_2}$ | $\frac{S_C}{S_{AB}}$ |
| smaller radius | 1.151 | 0.637 | 1.159 | 0.640 | 1.171 | 0.642 |
| larger radius | 1.022 | 0.565 | 1.042 | 0.572 | 1.074 | 0.586 |
| both | 1.033 | 0.571 | 1.048 | 0.576 | 1.060 | 0.581 |

**Table 4.1:** The measurements show that adjusting the smaller radius is probably the worst choice. Adjusting the larger radius or enlarging both perform similarly.

## 4.4 Wrapping cylinders and cones with an AABB

To compute the AABB of two cylinders or two truncated cones, the AABB of each has to be calculated first by finding their minimum and maximum extents when projected onto the principal axes. The AABB of a cylinder is calculated by first finding the bounding box of the centerline, then expanding it by including the projection of the circular bases. However, wrapping a truncated cone with an AABB is more complicated; as there is no available method for projecting a truncated cone directly onto the principal axes, the algorithm replaces the cone with two cylinders. Both replacement cylinders inherit one of the cone radii, and both of them will be half the height of the original truncated cone; although, decreasing the inherited height is only important for the cylinder with the larger radius. Calculating the joint AABB of two smaller AABBs is straightforward; one of them will be extended by the minimum and maximum values of the second one. The reason for considering the AABBs of cylinders and truncated cones is to use it as a reference when comparing the resulting surface areas of the different volumes.

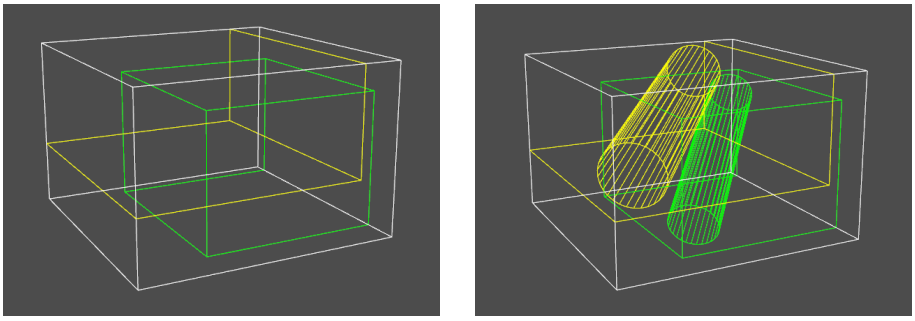


**Figure 4.5:** Joining two AABBs. The joined bounding boxes (left) and the boxes with the enclosed objects (right). The yellow and green bounding volumes are computed separately; the white AABB is the tight bounding box of the two.

## 4.5 Relative surface area test

The test results of the proposed methods for computing the unions of bounding volumes are in accordance with the expectations. They are not optimal but do not require complicated calculations. Although, constructing the joint bounding cylinder of two cylinders demands significantly more computational time than an AABB. Cylinders wrapping cylinders and truncated cones enclosing truncated cones were tested separately on a large number of randomly generated instances of similar object pairs to calculate the average surface ratios. Generally, the wrapping volume has at most a 50% larger surface area as the two separate objects together; however, its surface area is 40%

smaller on average than the corresponding AABB, depending on the object orientation.

Two options of axis construction were tested: one takes the middle points of the lines connecting the pairs of cylinder bases, the other finds the center of mass on the line with respect to the two volumes. The improvement over the middle point method is very subtle yet consistent in the case of cylinders. Truncated cones seem to perform worse with the weighted axis computation for the configuration next to each other; the surfaces are virtually the same for the after each other position.

The effect of choosing the middle point at axis construction or weighing by the object volume was tested on 10K randomly generated instances in three configurations. Configuring means limiting the ranges of randomly generated parameters, which are: *position*, *offset*, *rotation*. *Position* is given by a radius around the origin of the coordinate system, and it defines the distance of the first generated object from the origin. *Offset* is a radius around *position* where the second generated object will be translated. *Rotation* describes the range for rotation angles in degrees, limiting object rotation around a randomly generated axis in space. The larger these limit values, the larger the variance of the generated instances. The following configurations were compared:

- **Very similar.** The position and offset radii are 0.1, the angle limit is 10°.
- **Similar.** The position radius is 2, the offset is 0.8. The rotation angle limit is 25°.
- **Divergent.** The position radius is set to 3, and the offset is 2. The angle limit is 35°.

Table 4.2 shows that by increasing the range of values, the variance in similarity increases, and the average surface area ratios increase, confirming that the closer and more similar the two objects are, the better the enclosure.

## 4.6 Bounding volume tightness

The currently computed bounding volumes are not tight in general, and a possible error source is the choice of the new axis. Although the joint volume's

| **average surface 10K instances** | very similar | | similar | | divergent | |
|---|---|---|---|---|---|---|
| | $\frac{S_C}{S_1+S_2}$ | $\frac{S_C}{S_{AB}}$ | $\frac{S_C}{S_1+S_2}$ | $\frac{S_C}{S_{AB}}$ | $\frac{S_C}{S_1+S_2}$ | $\frac{S_C}{S_{AB}}$ |
| **Next to each other, axis: middle points** | | | | | | |
| cylinder with cylinder | 1.311 | 0.570 | 1.431 | 0.610 | 1.501 | 0.625 |
| cone with cone | 1.022 | 0.565 | 1.042 | 0.572 | 1.074 | 0.586 |
| **Next to each other, axis: weighted** | | | | | | |
| cylinder with cylinder | 1.280 | 0.556 | 1.401 | 0.594 | 1.469 | 0.618 |
| cone with cone | 1.018 | 0.574 | 1.117 | 0.622 | 1.458 | 0.804 |
| **After each other, axis: middle points** | | | | | | |
| cylinder with cylinder | 2.271 | 0.524 | 2.395 | 0.545 | 2.527 | 0.571 |
| cone with cone | 1.922 | 0.608 | 1.927 | 0.608 | 1.947 | 0.614 |
| **After each other, axis: weighted** | | | | | | |
| cylinder with cylinder | 2.246 | 0.516 | 2.353 | 0.539 | 2.511 | 0.566 |
| cone with cone | 1.921 | 0.606 | 1.933 | 0.610 | 1.945 | 0.613 |

**Table 4.2:** Average relative surface coefficients. $S_C$ is the surface area of the bounding cylinder or cone, $S_1$ and $S_2$ denote the surface area of the two wrapped volumes, and $S_AB$ denotes the surface of the corresponding axis-aligned box. The improvement over the middle point method is minimal.

surface area is equally dependent on the mutual position and orientation of the wrapped volumes, that is an unalterable characteristic. As noted before, axis-aligned bounding boxes do not provide tight enclosures for oblong objects with a diagonal orientation with respect to the principal axes. Cylinders proved to have approximately 40% lower surface areas, which should be a sufficient enhancement for complete scenes.

Since the radius of the bounding cylinder or cone is established by finding the farthest point from the chosen axis, if one of the objects is significantly thinner, and the two object axes are almost parallel, the smaller volume might not be touching the side of the enclosing shape. However, these loose-fitted bounding shapes still prove to be smaller than the compared AABB, especially for diagonally oriented volumes. Note that this is an indication that cylinders should not only be similarly oriented but also similar in size.

There was an effort towards further optimizing the axis selection for cylinders by an exhaustive search on the two lines connecting the base centers. However, this process requires an enormous amount of extra computation for a relatively small improvement, making it an unnecessary modification.

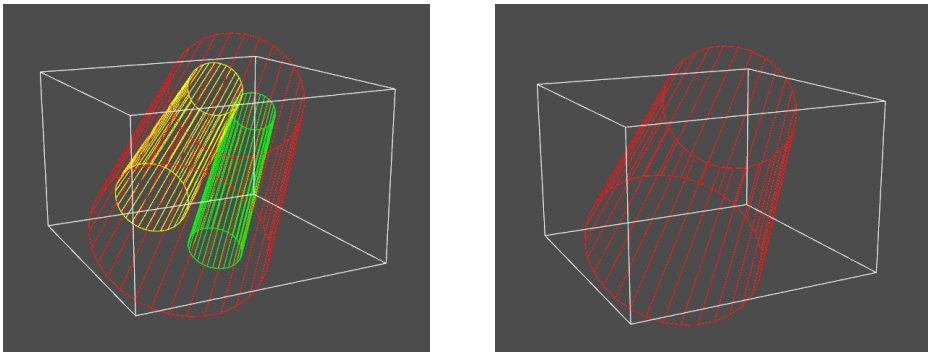


**Figure 4.6:** An example of comparing the bounding cylinder (red) and the AABB of two similar cylinders. The relative surface of the enclosing cylinder to the sum of the enclosed is 1.78. However, the relative surface of the red cylinder to the AABB is 0.60.

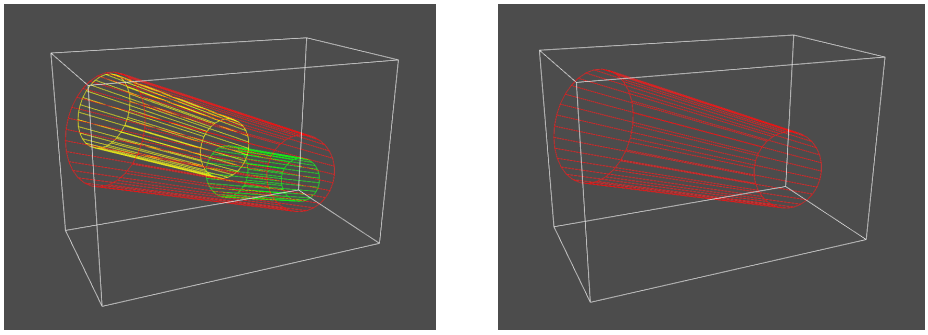


**Figure 4.7:** An example of comparing the bounding cylinder and the AABB of two similarly oriented cylinders positioned after each other. The relative surface of the red cylinder to the sum of the surfaces of the wrapped volumes is 1.59. The relative surface of the red volume to the AABB is 0.40.

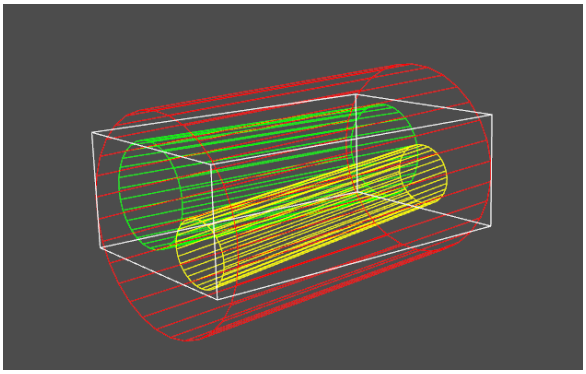


**Figure 4.8:** An example of the rare case when the AABB is in fact superior to the bounding cylinder in the context of surface areas. The relative surface area of the red bounding cylinder to the sum of the enclosed cylinders is 1.566. The red cylinder's surface to the AABB surface ratio is equal to 1.094. However, this ratio is the result of the cylinders being aligned with the coordinate axes.

# Chapter 5

# Hybrid hierarchy

The principal idea that this work aimed at exploring was building hierarchies that provide a tighter bound for oblong objects than the commonly used bounding volume hierarchies (BVH). The first step was to enable the construction of a BVH purely built on cylinders.

## 5.1 Theory and data

Theoretically, bounding cylinders should provide a more precise bound for similarly shaped, oblong scene objects. This might be true for the lowest levels of the hierarchy; however, the orientation similarity, one of the exploitable characteristics of oblong primitives, is less and less prominent in the higher levels, producing needlessly large bounding volumes. Axis-aligned bounding boxes are likely to produce highly redundant levels for scenes densely packed with thin objects, resulting in a high number of intersection tests per ray. Nevertheless, the overall surface area of the hierarchy will be much lower than for the cylinder hierarchy.

The results obtained from measuring the surface areas of the AABB and cylinder BVH versions implied that there should be a clustering iteration at which it would be beneficial to switch to AABBs from cylinders, thus creating a hybrid BVH. The primary scene used for experimenting was a fraction of 0.1% of the Hairball model ($\sim$ 2.8M triangles). Figure 5.1 shows that while boxes tend to keep the overall surface area relatively low, the cylinder

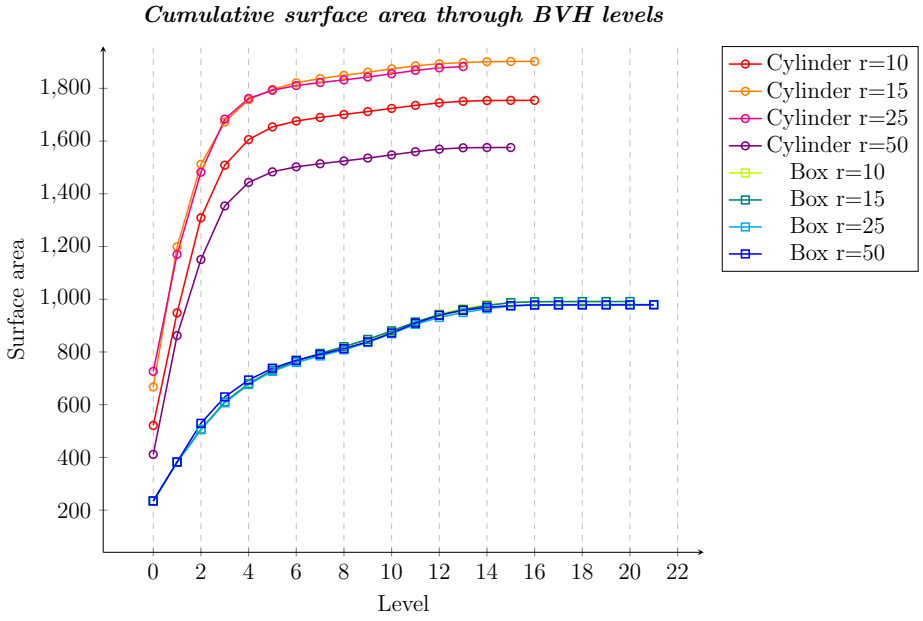



**Figure 5.1:** Comparing the cumulative BVH surface area of the cylinder and AABB hierarchies with various radii for the model Hairball 0.1%.

hierarchy surface appears to grow slower below the first 4-5 levels, implying tighter bounds. An interesting observation can be made about the effect of the radius selection on the final BVH surface.

The AABB hierarchy appears to follow the same path for a range of different radii, while the cylinder surfaces exhibit high variability. The reason for this might be that cylinders are also sorted by the Morton codes of their centroids; yet, these volumes may be oriented arbitrarily in the scene, which the centroid sorting does not take into account. While the initial clusters are sorted along the Morton curve, they are not sorted again after iterations because they tend to adhere to their original ordering, as noted in [2].

Due to the fact that the ordering only considers the centroids of volumes, different radii facilitate the detection of better neighbors for certain clusters. These neighbors might be relatively distant along the Morton curve, yet the joint bounding volume of the pair of cylinders has a lower surface area; thus, such a neighbor is *closer* and preferred. However, the fluctuation of the cumulative surface areas seen in Figure 5.1 implies no direct correlation between smaller surfaces and higher radii. One explanation might be that finding a more suitable neighbor farther away along the Morton curve in an early iteration causes disruption in later iterations. As Figure 5.2 illustrates, the clusters merged later become extremely large extremely fast because their orientation is highly disparate. This tendency is not present in the AABB

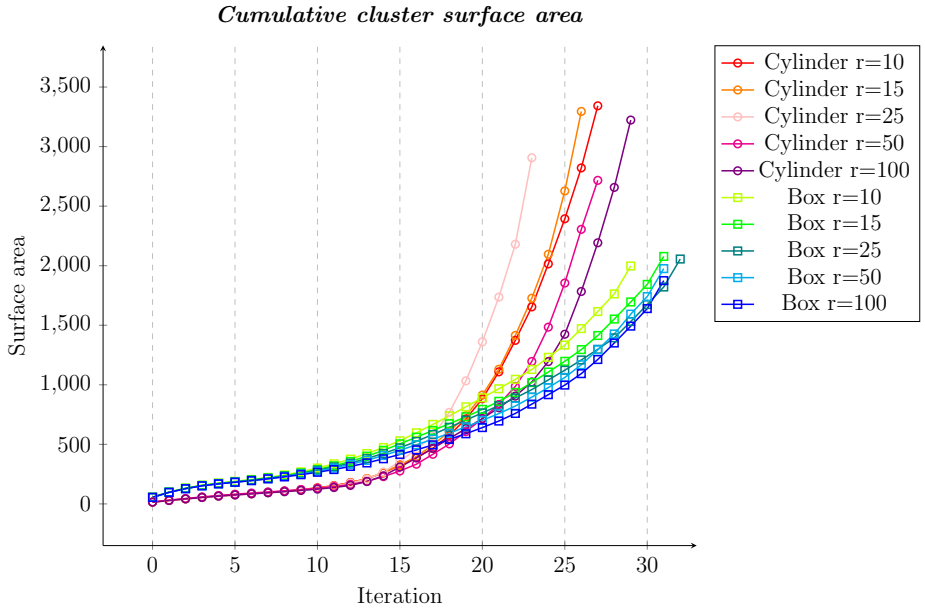



**Figure 5.2:** Comparing the cumulative cluster surface area of the cylinder and AABB hierarchies for the model Hairball 0.1%.

hierarchy, as these volumes may not be rotated, and orientation is generally undefined for them.

While the cumulative cluster surface areas of both hierarchies tend to grow exponentially, the box cluster areas grow significantly slower. As opposed to the later iterations, the sum of the cluster surface areas is significantly lower for cylinders until a specific iteration (see Figure 5.2), validating the proposal of hybrid hierarchies. The idea was that it could suffice to choose an iteration in which every cluster's bounding cylinder would be replaced by the AABB of that cylinder. Although this naturally introduces some amount of redundancy, the resulting hierarchy should have a lower surface area.

## 5.2 Realization

Ideally, a hybrid hierarchy should utilize the different types of bounding volumes by enabling the clustering process to switch from cylinders to boxes regardless of the iteration at any level of the hierarchy. The SAH function could evaluate the benefits of a switch by comparing both joint bounding volumes of two clusters at each merge step, preferring the one with a lower surface area. This solution would be relatively costly, however, it has the

potential to avoid some of the larger cylinders and keep the overall surface areas low. It also opens the question of joining two bounding volumes of differing types; however, it seems obvious to always favor the AABB in these situations. If the transition happens individually at every branch, the selected data structures have to be assembled carefully to preserve simple traversability of the tree.

Currently, the hybrid builder is only able to construct a hierarchy with a pre-defined transition iteration, from which it operates exclusively on AABBs. This choice was deemed sufficient for the purpose of this thesis. For best results, the choice of the switch iteration should be scene-dependent and probably be selected together with the search radius. This form of optimization fell out of scope for this work.

# Chapter 6

# Implementation

The PLOC [2] algorithm serves as the basis for this work. Since the goal was to incorporate the discussed non-standard bounding volumes into this algorithm, and to compare the resulting hierarchies in terms of build and render performance, the implementation of Arsène Pérard-Gayot [17] was utilized and modified accordingly. It is a header-only BVH library in C++17 containing a collection of different BVH builders, including a CPU version of the PLOC algorithm and a simple ray tracer to allow builder comparison. The library does not use any platform- or hardware-specific intrinsics, and it has no other dependencies than the C++ standard library. It employs OpenMP for parallelization.

## 6.1 PLOC implementation on a CPU

The locally-ordered clustering algorithm in [17] is implemented according to Bittner and Meister [2]. Several buffers are used, with two main buffers that store the clusters before and after an iteration, which are swapped at the end of each iteration. Child nodes (and primitives) belonging to the same node are stored at successive indices. The algorithm starts with computing the corresponding bounding volume for each scene primitive and the centroid of the volume. According to the centroid, Morton codes are generated for the clusters; encoding the centroid consists of interleaving the bits of the $x$, $y$ and $z$ values. These will be the initial clusters. The main loop consists of three phases: the *nearest neighbor search*, the *merging phase* and the *compaction phase*, repeated until a single cluster remains [2].

**Nearest neighbor search.** The search is performed as a 1D search in an array of clusters along the Morton curve, computing the distances between clusters. The distance between two clusters is defined as the surface area of the axis-aligned bounding box of the two clusters. In his implementation, Pérard-Gayot [17] heavily uses inline functions and utilizes distance caching in the nearest neighbor search to lower the computational time. Additionally, he saves a multiplication operation in each distance computation since he only considers half of the surface area of the axis-aligned bounding box. By eliminating factor 2 from the cylinder surface area equation, this operation can be saved similarly in the case of cylinders.

**Merging.** In the merge phase, every cluster is checked against its nearest neighbor. If two clusters are the nearest neighbors to each other, they are marked for merging, replacing one of the clusters with a new cluster and marking the other as invalid [2]. The new cluster becomes the parent of the two merged clusters.

**Compaction.** According to the indices computed in a prefix scan, the new clusters are inserted and the invalid clusters removed from the output array. The prefix scan is performed before the actual construction of the new parent nodes but after the nodes are marked as mergeable or not. The newly constructed clusters are then directly inserted at the indices calculated in the prefix sum pass. The clusters that did not find their nearest neighbor in the given iteration are copied to the output array unaffected.

## 6.2 Modifications

To use non-standard bounding volumes, precisely cylinders with the PLOC algorithm, it was essential to define new structures and functions handling cylinders. However, it was also essential to keep the original structure of the code for it to remain comparable. Most of the modifications came in the form of adding new overloads of certain functions, such as `build()`, `cluster()` and `traverse()`.

The `build()` function has overloads for every considered hierarchy type. The original and the cylinder version only differ in the parameter list, while the hybrid builder is further modified, see section 6.2.4. The original function consists of sorting the primitives by the Morton code of their center of mass, then creating the initial clusters by computing the bounding volume of each

primitive. It also sets `begin` and `end` indices into the cluster array. After the initialization, the function enters a simple while loop that runs until a single cluster remains, that is, until the difference of `begin` and `end` becomes one. In each iteration, the `cluster()` function is called.

The `cluster()` function also has overloads suitable for the distinct builders, but they only differ in their parameter list. After initializing the distance matrix for caching and establishing the search range, the distance cache is filled with computed distances for every cluster $i$ in the interval $\langle i+1, i+r\rangle$. Then during the nearest neighbor search, for each cluster, the previously computed distances are used for the interval $\langle i-r, i\rangle$, and new distances are calculated and cached for the interval $\langle i+1, i+r\rangle$. After the best neighbor is established for each cluster, nodes that have found their mutually corresponding nearest neighbor are marked as mergeable. Then a prefix sum is computed to find the insertion indices for the new clusters. Finally, the newly constructed nodes are inserted into the output array, and the next `begin` and `end` indices are returned for the next iteration.

The `traverse()` function is called for every pixel of the image during rendering, but the actual functionality is inside the `intersect()` call. This function also has a hybrid and a cylinder variant. The original hierarchy traversal processes encountered leaves immediately, as well as moving to the nearest child node and only pushing the farther child to the stack, if intersected. The cylinder traversal is identical to the original, but the hybrid traversal employs two stacks instead of one and needs to mind the volume type switch.

### 6.2.1 BVH

The `bvh` struct originally contained a `Node` struct holding the six values of the bounding box, one boolean bit for distinguishing leaf nodes, a bit field for the number of primitives belonging to the node, and an index to the node's first child or first primitive, for inner and leaf nodes respectively. This structure should be 32 bytes with single precision, and 64 bytes with double precision [17]. Although this is preferable, another field had to be inserted to facilitate the hybrid hierarchy construction - node origin. This means saving an index into the cylinder node array to the cluster from which the AABB was created. The `bvh` struct contains an array holding pointers to every node and another holding indices to the scene primitives. A new pointer array was added to hold the cylinder node pointers. The new node type, `CustomNode` is virtually the same as `Node`, but instead of the six values representing the bounds of the axis-aligned bounding box (AABB), it holds a point and a vector defining

the cylinder center and axis. That would be exactly six values like the AABB bounds. However, two additional scalars are required for storing the radius and the height of the volume.

Both node types contain a structure, `BoundingBoxProxy` that handles the conversion of nodes to bounding boxes. The node is also equipped with a function returning a proxy to this node. The actual bounding box or cylinder does not exist as a separate entity until the clustering algorithm converts a node into a bounding volume to call one of its member functions. All values are kept inside the node structure.

The `Scalar` type may be set as desired. It defaulted to single precision floating point numbers in the original implementation. However, cylinders required complicated calculations demanding double precision, which currently seems unavoidable.

```
struct Node {
    Scalar bounds[6];
    bool is_leaf : 1;
    IndexType primitive_count :
    sizeof(IndexType) * CHAR_BIT - 1;
    IndexType first_child_or_primitive;
    // index to cylinder in hybrid version
    IndexType origin;

    ...
}
```

**Listing 6.1:** `Node`

```
struct CustomNode {
    Vector3<Scalar> p1, axis;
    // cylinder height and radius
    Scalar h, r;
    bool is_leaf : 1;
    IndexType primitive_count :
    sizeof(IndexType)* CHAR_BIT - 1;
    IndexType first_child_or_primitive;

    ...
}
```

**Listing 6.2:** `CustomNode`

### 6.2.2 Clustering

In each `cluster()` call, the bounding volume's member function, `extend()` is called frequently. For the AABB, this means simply inflating the bounding volume by taking the common minima and maxima of the two volumes. For a cylinder, extending a volume means calculating a completely new cylinder providing a tight enclosure for both the extended and the extending volumes. To potentially save some computation time, an inclusion test is carried out before computing the new cylinder. It is a rejection test gradually checking the cylinder qualities: whether the smaller volume projected onto the larger axis falls inside the larger radius, then if it fits between the two caps of the larger cylinder. If one of the cylinder volumes completely includes the other cylinder, the larger one is returned as their joint bounding volume. If the test

fails, a new cylinder is computed.

### 6.2.3 Ray tracing

Tracing a ray consists of traversing the hierarchy while computing the intersection of each bounding volume with the ray, searching for the closest intersection to the camera along the ray. It is an elimination test - if the ray misses a node's bounding volume, it misses the whole subtree of that node. If the ray does intersect the node, traversal continues until a leaf is reached. In a leaf node, each primitive in that node is checked against the ray.

#### Ray-box intersection

The intersection test uses the slab method [7], treating the bounding box as the space between three pairs of parallel planes. To make the intersection test more efficient, rays are classified by octant inside the `NodeIntersector` struct constructor, see Listing 6.3. The intersection test itself consists of computing the entry and exit points of the ray against the bounding box. To eliminate another time-consuming step, the test does not use divisions. Instead, there is a multiplication by the inverted components of the ray direction vector. This provides safe handling of zero components.

```
struct NodeIntersector {
    std::array<int, 3> octant;

    NodeIntersector(const Ray<Scalar>& ray)
        : octant{
            ray.direction[0] < Scalar(0),
            ray.direction[1] < Scalar(0),
            ray.direction[2] < Scalar(0)
    }
    {}

    double intersect_axis(int axis,
        const Vector3<Scalar> point,
        const Ray<Scalar>& ray)
    {
        // with division:
        // (p[axis] - ray.origin[axis]) / ray.direction[axis];
        // without division:
        return (p[axis] - ray.origin[axis]) * ray.inverse_direction[axis];
    }
}
```

**Listing 6.3:** The `NodeIntersector` struct.

### Ray-cylinder intersection

Computing the intersections of a ray with a cylinder is a more challenging task. The test function is implemented according to the procedure described by Zorin [18]. First, the intersection with an infinite cylinder along the bounding cylinder axis is computed. The equation for an infinite cylinder along the line $p_a + v_a t$ with radius $r$ is

$$(q - p_a - (v_a \cdot (q - p_a))v_a)^2 - r^2 = 0 \tag{6.1}$$

Where $q = (x, y, z)$ is a point lying on the cylinder. Substitute a ray $p + vt$ for $q$, which yields

$$(p - p_a + vt - (v_a \cdot (p - p_a + vt))v_a)^2 - r^2 = 0 \tag{6.2}$$

which reduces to

$$At^2 + Bt + C = 0 \tag{6.3}$$

where

$$\begin{aligned} A &= (v - (v \cdot v_a)v_a)^2 \\ B &= 2(v - (v \cdot v_a)v_a) \cdot (\Delta p - (\Delta p \cdot v_a)v_a) \\ C &= (\Delta p - (\Delta p \cdot v_a)v_a)^2 - r^2 \end{aligned}$$

with $\Delta p = p - p_a$. If Equation 6.3 has a non-negative determinant, the intersections exist, which need to be checked if they lie between the two planes of the cylinder caps, precisely if

$$\begin{aligned} v_a \cdot (p + vt_i - p_1) > 0 \\ v_a \cdot (p + vt_i - p_2) < 0 \end{aligned} \tag{6.4}$$

for $i = 1, 2$. If they lie between the cap planes, there follows an intersection test with each of these planes. Similarly to box intersections, the slab method may be used here; thus, the inverse ray direction is also stored by the modified

intersector. As a last check, these two intersection points must lie inside the cylinder caps and be non-negative.

$$\begin{aligned} &v_a \cdot (p + vt - p_k) = 0 \\ &\text{with } t > 0 \text{ and} \\ &(p + vt - p_k)^2 < r^2 \end{aligned} \tag{6.5}$$

where $k = 1, 2$ for the top and bottom cap, yielding $t_3$ and $t_4$, respectively. In the end, there should be at most four point candidates if any intersection exists, the closest of which is returned.

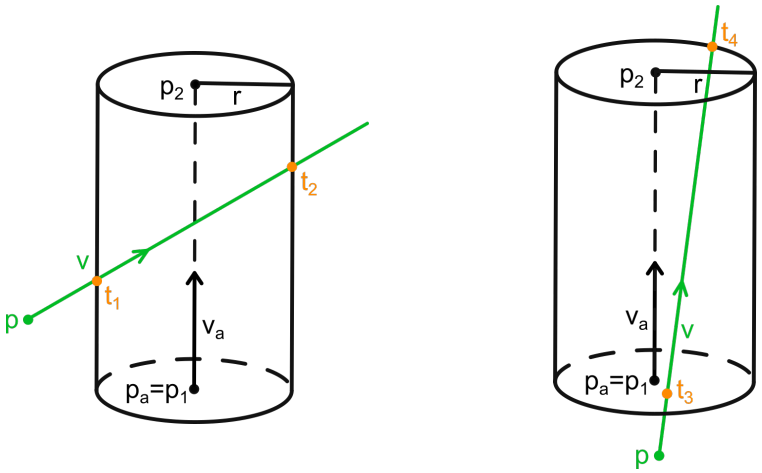
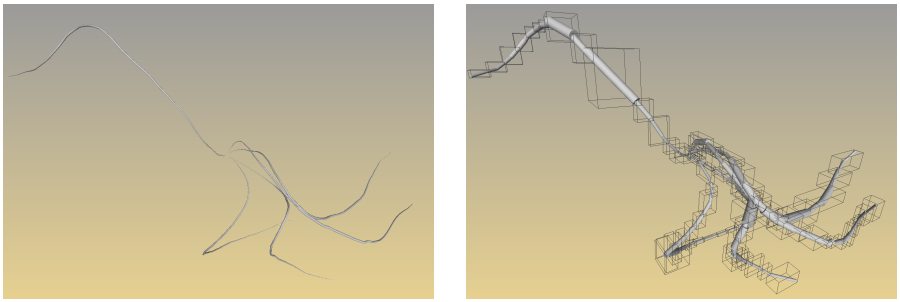



**Figure 6.1:** Ray-cylinder intersection types. $t_1$ and $t_2$ (left) are two possible intersection points of the ray with the side of the cylinder. $t_3$ and $t_4$ (right) illustrate valid cap intersections.

### 6.2.4 Building a hybrid hierarchy

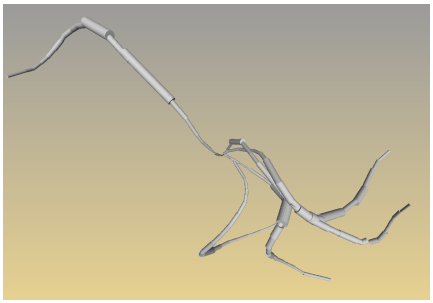
In the current implementation, the main loop of the hybrid `build` function is equipped with an iteration counter. When the desired iteration is reached, it breaks the main loop and transitions from cylinders to axis-aligned bounding boxes. The process of switching consists of creating a new node for each cluster and calculating the AABB of the cylinder volume, saving it in the newly created node. Then these new nodes copy the `first_child_or_primitive` and `primitive_count` values of the cylinder nodes. Note that all tested BVH versions have only one primitive per leaf. Each node saves an index to the cylinder node array indicating their `origin` - a newly added variable in the `Node` struct (see Listings 6.1, 6.2). The most relevant step is marking these nodes as *leaves*, even though they are most likely inner nodes of the whole



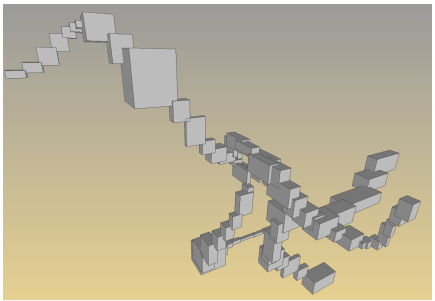
**(a) :** The initial scene



**(b) :** AABBs of the cylinders



**(c) :** Cylinders before the switch



**(d) :** AABBs after the switch

**Figure 6.2:** Showing the transition from cylinders to axis-aligned bounding boxes in the scene Hairball 0.1% at iteration 10. (a) the original scene before clustering, (b) illustrating the switch with both cylinders and boxes visible, (c) cylinders before, (d) AABBs after the transition.

hierarchy. However, these nodes are leaf nodes in the context of the AABB hierarchy, which will be built over the clusters present at the switch iteration. This indicator facilitates traversal during ray tracing since there is no other connection between the cylinder and box nodes. After the transition, the function enters a second loop calling the suitable cluster function variant for the new nodes, which will be repeated until a single cluster remains.

### 6.2.5 Hybrid traversal

The hybrid hierarchy traversal is very similar to the original function. However, it might be suboptimal because of the different node types that currently require the use of two separate stacks. The original function traverses the hierarchy and processes every leaf node immediately upon encounter. When the hybrid traverser encounters a node marked as a leaf, it means it reached a point where the box hierarchy changes to a cylinder hierarchy. The origin of this pseudo leaf node quite rarely might be an actual leaf that has to be intersected instantly. In the more likely case of an inner node, a nested

traversal loop is entered that searches through the cylinder subtree, as if it was an independent cylinder hierarchy. This nested while loop is exactly the same as the main traversal, however, it operates on the cylinder stack (see Listings 6.4, 6.5). It returns to the main loop only after processing the whole cylinder stack.

**Listing 6.4:** Pseudo code of the basic traversal loop body, responsible for finding the closest intersection for a ray. It is used in the box and cylinder hierarchy, and a nested version is utilized in the hybrid hierarchy. First, it intersects the left and the right child node. If an intersection exists and the child is a leaf, it processes the primitive intersections. Otherwise, it continues to the closer node pushing the farther to the stack, if the ray intersects both children; or to the only node that is intersected, if any.

```
/* BASIC TRAVERSAL LOOP BODY */
first_child = node.first_child_or_primitive
left = node_array[first_child]
right = node_array[first_child + 1]
d_left = intersect_node(left)
d_right = intersect_node(right)
if left exists and is leaf
    intersect_leaf(left)
end if
if right exists and is leaf
    intersect_leaf(right)
end if
if exactly one exists
    node = left exists ? left : right
else if both exist
    stack.push(farther)
    node = closer
else
    if stack is empty
        break
    end if
    node = stack.pop
end if
```

### 6.2.6 Hierarchy built on rays

As mentioned in chapter 2, it is possible to build a bounding volume hierarchy over rays, assuming each ray is an oblong triangle. The idea is similar to the line-space gathering process of Sun et al. [14]. They observe that lighting rays are generally long, and cross large sections of the scene, producing large bounding volumes, thus being unsuitable for spatial hierarchies. The proposed cylindrical bounding volumes offer a partial solution to this problem with reduced surface areas. However, the surface area reduction is insufficient due to the issues with wrapping arbitrarily oriented cylinder pairs with a cylinder.

Thus, the randomly generated rays (triangles) may hardly be improved by cylindrical bounding volumes. Evidence is provided in Chapter 7.

**Listing 6.5:** Pseudo code of the hybrid ray traversal function with nested cycles. It utilizes the *basic traversal loop* (Listing 6.4) twice with certain modifications. In the outer while cycle, it traverses the box hierarchy processing each node. In the while loop inside the first if statement, it searches through the cylinder subtree of the pseudo leaf box, until the whole branch is processed.

```
while true
    if node is leaf
        c_node = node.origin
        if c_node is leaf
            intersect(c_node)
            if stack is empty
                break
            end if
            node = stack.pop
            continue
        end if
        while true
            /* BASIC TRAVERSAL LOOP BODY */
            // with c_node and c_stack
        end while
        if stack is empty
            break
        end if
        node = stack.pop
        continue
    end if

    /* BASIC TRAVERSAL LOOP BODY */
    // but only intersects leaves if the node origin is a real leaf
end while
```

# Chapter 7

# Results

This project aimed at slightly improving bounding volume hierarchies for oblong cylinder-shaped objects such as hair. The discussed methods for constructing joint bounding volumes succeed in significantly reducing the immediate surface area of the bounding volumes. Although this effect is generally not as prevalent throughout the hierarchy as at the lowest levels, the proposed solution has the potential to improve the qualities of a BVH for certain scene types. The results presented in the following are per the expectations and support the primary hypothesis that using cylinder-shaped bounding volumes may sometimes be beneficial.

The small Hairball fraction scene was used for tuning and measuring the performance ($\sim$ 2880 triangles) while working on the project, as it comprises a low number of oblong triangles. Further tested scenes that exhibit comparable qualities are a larger portion of the Hairball scene (10%), a bust of a woman, and random rays represented by thin triangles, generated by the author. The control scenes were of a different kind, the Sibenik Cathedral and Park as architecture and Armadillo as an evenly tessellated figure.

The current version of the hierarchy builder and ray tracer is a proof of concept; it is an experimental tool lacking intensive optimization and unsuited for use outside this project. All measurements were conducted on an Intel Core i5 7300HQ 2.50 GHz CPU under Windows, without parallelization due to issues with OpenMP.

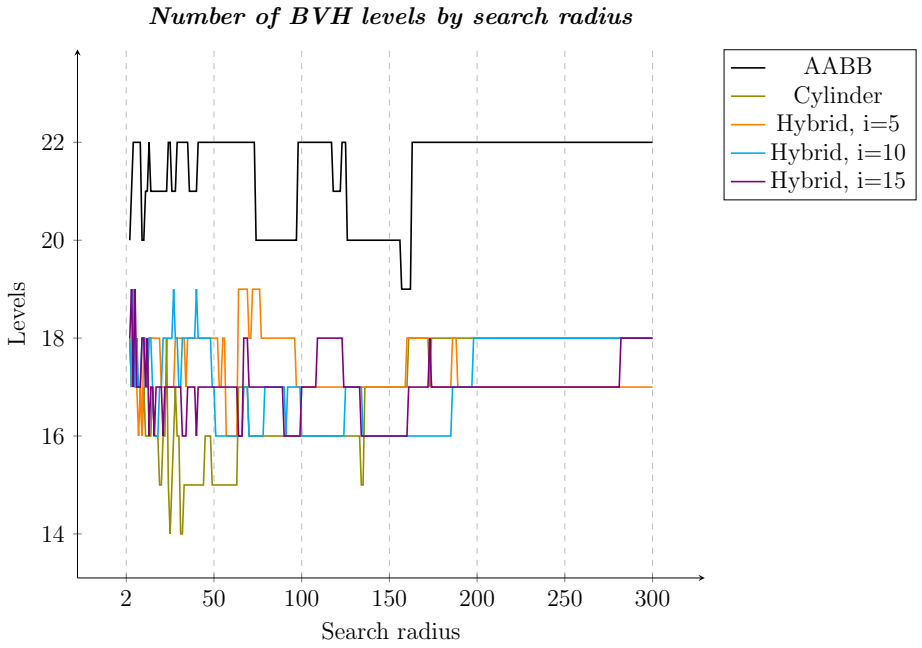



**Figure 7.1:** Comparing the depth of every type of hierarchy for a varying search radius for the scene Hairball 0.1%.

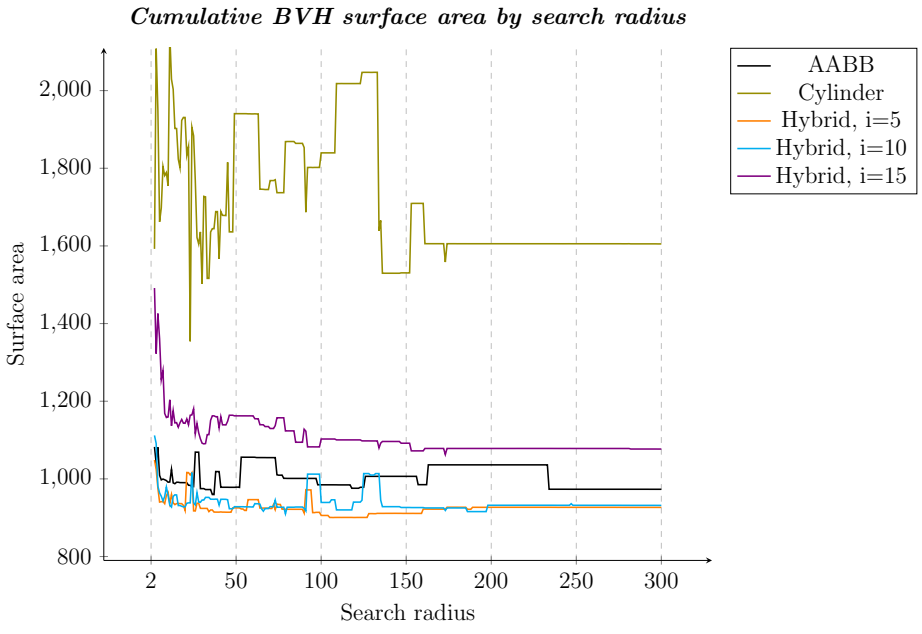



**Figure 7.2:** Showing the cumulative surface area of each BVH type for a large range of search radii. Measured for the scene Hairball 0.1%.

A notable characteristic of the cylinder and hybrid hierarchies is that they consistently produce shallower trees than an AABB hierarchy, as shown in Figure 7.1. The cylinder variant produces a record low of 14 levels (for $r = 25, 31, 32$). A shallower tree means that during clustering, a high number of clusters were merged in every iteration; thus, most of the leaf nodes are positioned at the lowest levels of the BVH, making it more balanced. Taking Figure 2.4 as an example, the amount of unmerged red clusters has to be relatively low in most iterations.

The most considerable weakness of the pure cylinder hierarchy is visible in Figure 7.2, with certain search radii approximately doubling the surface area of the AABB hierarchy with the same radius. However, the hybrid hierarchies produce surface values a lot closer to the AABB results. The lower the switch iteration, the lower the overall surface area values. For $i = 5$, it is almost always under the AABB variant, except for a few radius selections around 25. Although render times tend to change for the worse with alternative bounding volumes, the hybrid hierarchy shows a consistent improvement of the hierarchy surface area with 3-8% percent lower values (Table 7.1). The cylinder hierarchy performs exceptionally poorly, with the build taking 10-20× longer and rendering 7-8× longer than the reference BVH with AABBs.

Figures 7.1, 7.2 suggest that the most promising hybrid alternative among the tested hierarchies for the Hairball fraction is the BVH switching at iteration 5. It exhibits highly similar behavior to the AABB BVH both in the context of cumulative cluster surfaces and the surface of the final BVH. As Figure 7.2 revealed, this variant almost always outperforms the AABB hierarchy for this scene. Moreover, the larger the search radius, the lower the cumulative surface areas. This is illustrated in Figures 7.3, 7.4. Equivalent measurements for hybrid hierarchies with a different switch iteration and further results can be found in Appendix A. The superiority of the $i = 5$ hybrid is proven by the test results in Tables 7.1, 7.2.

As noted in section 4, the greatest potential of cylinders lies in the densely packed scenes with thin triangles that are similarly oriented, as the bounding cylinder loses its dominance over an AABB with increasing primitive orientation variability. To verify this in practice, various models were used and their performance compared. The supposedly most suitable scenes were those containing hair strands, such as Hairball and Bust. Although the former comprises cylinder-shaped hair strands with similarly oriented triangles, the strands display high curvature variability, bringing different triangles near each other. However, hair present in the Bust model largely constitutes similarly oriented clusters with less randomness in bending, more closely resembling average human hair.

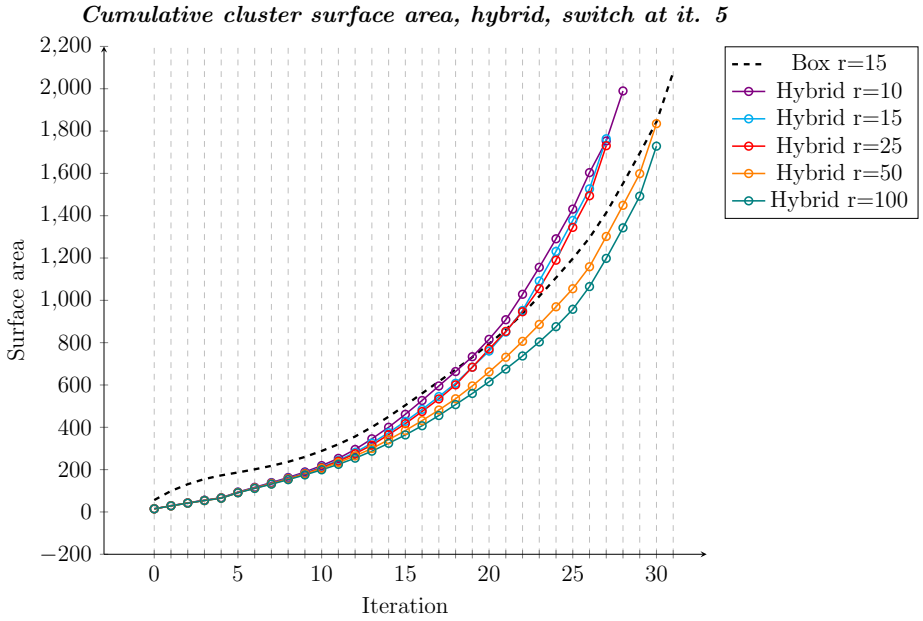



**Figure 7.3:** Comparing the cluster surface areas by iteration to the box hierarchy, switch at iteration 5. Scene: Hairball 0.1%.

The 10% Hairball scene displays too much randomness in the curvature of the strands to benefit from an alternative hierarchy. However, the measured surface areas are at least 13% lower than the AABB hierarchy when switched in iteration 5 (see Table 7.1). The later the transition occurs, the larger the resulting tree surface. At iteration 10, the hybrid variant still displays an approximately 10% improvement; the larger the radius, the more noticeable the improvement.

As mentioned previously, the Bust scene shows a different order of hair strands with less randomness and large clusters of similarly oriented hairs. The orientation similarity gets exploited in the alternative hierarchies for this scene. The results show a stunning decrease of approximately 50% in the cumulative surface area. The cylinder hierarchy even achieves a 60% decrease for $r = 10$, although for an incredibly long render. For this reason, the render times were not measured for the remaining radii with the cylinder hierarchy, as there is no potential to improve them. Although the hybrid hierarchies take significantly longer to build, they produce 23-30% percent lower render times compared to the reference BVH.

Adhering to the project requirements, the hierarchies were also tested on scenes containing different geometry, such as architecture and figures. The Armadillo model was chosen as a figure, as it is a finely and evenly tessellated mesh with a moderately high number of triangles. As expected, the cylinder

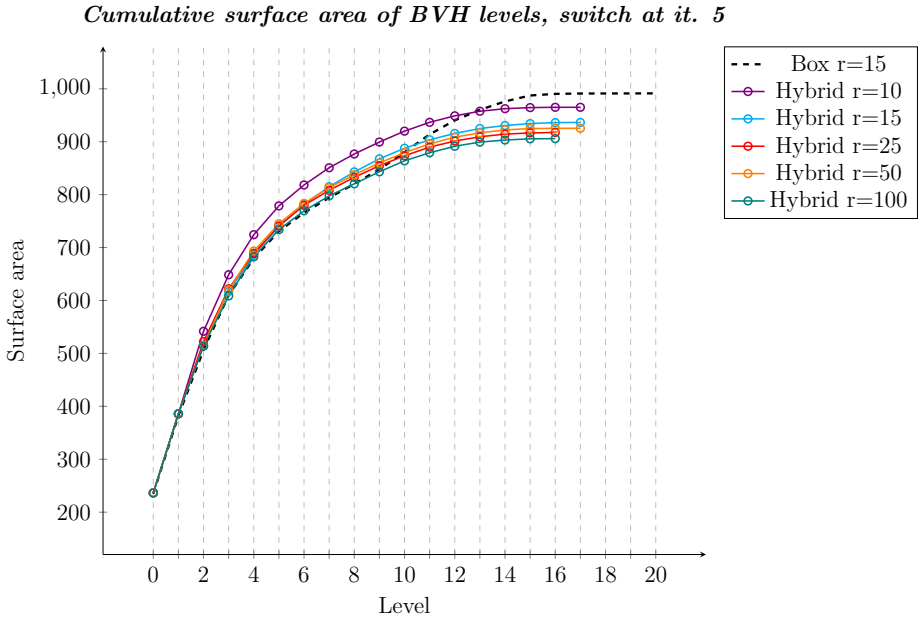



**Figure 7.4:** Comparing cumulative BVH surface areas of the hybrid hierarchy levels to the AABB hierarchy. Scene: Hairball 0.1%.

hierarchy performs appallingly, and the hybrid BVHs also fail to provide any improvement at all. The architectural scenes were the Park and the Sibenik Cathedral. The former contains less than 30K triangles and the latter roughly 80K, but both scenes are unevenly tessellated. This characteristic causes the alternative builders to approximately double the reference surface areas and render times, confirming the necessity of similar orientation. The main reason for the failure is that the large triangles in the scene, especially those along the principal planes that would have an exceptionally thin AABB, are wrapped with unnecessarily large cylinders immediately at the start of the algorithm. To illustrate this, the initial bounding volumes of the Hairball fraction and Park scenes are compared in Figure 7.5.

The Generated random rays scenes simulate randomly bouncing rays in a scene by 10K arbitrarily positioned and oriented triangles. The triangles were generated by defining the scene boundaries as a cube, generating two random points inside the cube for each triangle. Then, a third vertex was constructed at an $\varepsilon$ distance to one of the previous points. Generally, these triangles lack any similarity. The longest side of each triangle is equal to a chosen percentage of the diagonal of the scene's bounding box. For the two tested scenes, 30% and 70% lengths were chosen. In the scene with shorter rays, the alternative hierarchies all fail, the cylinder BVH nearly doubling the surface area. However, hierarchies built over the longer rays display more consistency between variants. Moreover, the cylinder hierarchy outperforms

the AABB version, and all alternatives yield lower surface areas with larger radii (see Table 7.2).

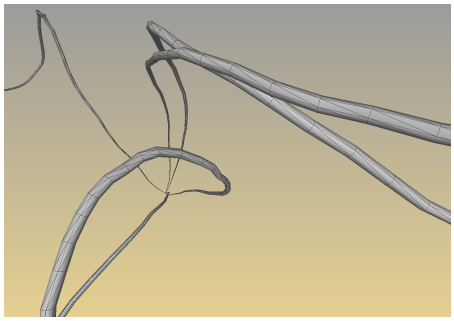
Figure 7.7 (a, b) illustrates the differences in ray tracing the 10% Hairball model with a box and a hybrid hierarchy, both having the same search radius. Traversal steps and intersection tests carried out for each ray are mapped to colors. Although the hybrid version appears to reduce the red regions and it lowers the number of intersection tests, the number of traversal steps increases, which results in longer renders (see Table 7.2). Figure 7.7 (c, d) offers a side-by-side comparison of the render performance for the Bust model between the AABB and hybrid BVHs. The hybrid variant displays a considerable improvement over the box hierarchy, decreasing the average number of intersection tests by an order of magnitude.

| **Hairball 0.1%, 2 880 triangles** | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **AABB** | | | **Cylinder** | | | **Hybrid (5)** | | | **Hybrid (10)** | | | **Hybrid (15)** | | |
| **r** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** |
| **10** | 0.12 | 5.66 | 4.21 | 2.30 | 37.89 | 7.46 | 1.60 | 8.81 | 4.10 | 1.85 | 11.12 | **4.06** | 2.09 | 14.91 | 4.93 |
| **15** | 0.17 | 6.11 | 4.21 | 2.90 | 38.31 | 8.09 | 2.42 | 8.76 | **3.98** | 2.72 | 11.36 | 4.07 | 3.91 | 17.04 | 4.86 |
| **25** | 0.27 | 5.95 | 4.16 | 4.89 | 36.61 | 8.01 | 4.27 | 8.63 | **3.90** | 4.44 | 11.10 | 3.99 | 5.03 | 16.55 | 5.02 |
| **50** | 0.54 | 6.08 | 4.16 | 9.16 | 39.21 | 8.25 | 7.72 | 8.95 | **3.93** | 8.66 | 13.77 | 3.94 | 9.45 | 15.31 | 4.94 |
| **100** | 0.94 | 6.18 | 4.18 | 19.92 | 44.04 | 7.91 | 16.85 | 9.31 | **3.85** | 17.13 | 9.96 | 3.99 | 17.53 | 14.54 | 4.68 |
| **Hairball 10%, 288 000 triangles** | | | | | | | | | | | | | | | |
| | **AABB** | | | **Cylinder** | | | **Hybrid (5)** | | | **Hybrid (10)** | | | **Hybrid (15)** | | |
| **r** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** |
| **10** | 0.20 | 1.61 | 95 | 3.75 | 41.81 | 219 | 2.68 | 1.89 | **83** | 3.49 | 3.38 | 87 | 3.65 | 6.37 | 109 |
| **15** | 0.32 | 1.73 | 95 | 5.17 | 41.58 | 186 | 4.08 | 2.11 | **83** | 4.63 | 3.64 | 87 | 6.13 | 6.78 | 106 |
| **25** | 0.54 | 1.76 | 94 | 10.04 | 39.65 | 168 | 6.63 | 2.29 | **82** | 7.53 | 3.62 | 85 | 9.79 | 6.91 | 103 |
| **50** | 0.88 | 1.50 | 94 | 16.20 | 31.01 | 469 | 13.11 | 2.03 | **82** | 15.83 | 3.07 | 84 | 17.85 | 5.56 | 99 |
| **100** | 1.86 | 1.54 | 94 | 32.83 | 28.36 | 160 | 25.51 | 1.81 | **81** | 30.00 | 3.00 | 82 | 35.24 | 5.74 | 96 |

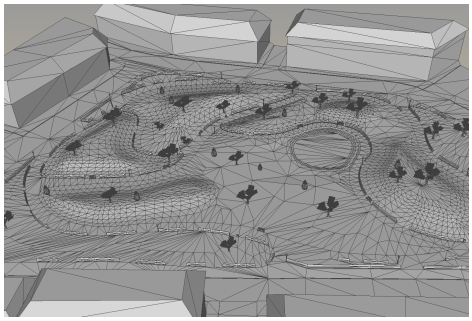
**Table 7.1:** Build and render times along with the relative surface area of hierarchy alternatives for different radii. Measured times are in seconds for Hairball 0.1% and in minutes for Hairball 10%. Surface areas (SA) are shown relative to the surface are of the scene's AABB. The lowest surface area for each radius (each row) is emphasized in bold. Every BVH contains one primitive per leaf.

| Bust, 4 271 696 triangles | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | AABB | | | Cylinder | | | Hybrid (5) | | | Hybrid (10) | | | Hybrid (15) | | |
| **r** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** |
| **10** | 5.06 | 18.34 | 1644 | 52.87 | >2h | **634** | 38.72 | 14.16 | 812 | 49.87 | 17.19 | 719 | 49.68 | 21.99 | 666 |
| **25** | 13.48 | 16.41 | 1617 | >2h | — | — | 99.96 | 13.50 | 769 | 119.62 | 15.78 | **680** | >2h | — | — |
| **100** | 60.51 | 17.01 | 1585 | >2h | — | — | >2h | 10.97 | **714** | >2h | — | — | >2h | — | — |
| **Generated random rays (30% of the diagonal), 10 000 triangles** | | | | | | | | | | | | | | | |
| | AABB | | | Cylinder | | | Hybrid (5) | | | Hybrid (10) | | | Hybrid (15) | | |
| **r** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** |
| **10** | 0.01 | 24.78 | **395** | 0.17 | >2h | 756 | 0.10 | 53.10 | 504 | 0.14 | 98.54 | 578 | 0.15 | >2h | 642 |
| **15** | 0.02 | 23.95 | **390** | 0.25 | >2h | 720 | 0.17 | 49.12 | 487 | 0.22 | 94.99 | 562 | 0.23 | >2h | 616 |
| **25** | 0.04 | 25.06 | **386** | 0.48 | >2h | 692 | 0.28 | 47.20 | 471 | 0.37 | 79.36 | 535 | 0.41 | >2h | 585 |
| **Generated random rays (70% of the diagonal), 10 000 triangles** | | | | | | | | | | | | | | | |
| | AABB | | | Cylinder | | | Hybrid (5) | | | Hybrid (10) | | | Hybrid (15) | | |
| **r** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** |
| **10** | 0.01 | >2h | **766** | 0.15 | >2h | 854 | 0.10 | >2h | 812 | 0.13 | >2h | 841 | 0.15 | >2h | 852 |
| **15** | 0.04 | >2h | 735 | 0.41 | >2h | **724** | 0.26 | >2h | 791 | 0.33 | >2h | 734 | 0.39 | >2h | 732 |
| **25** | 0.19 | >2h | 708 | 1.72 | >2h | **639** | 1.04 | >2h | 677 | 1.36 | >2h | 669 | 1.66 | >2h | 659 |
| **Armadillo, 345 944 triangles** | | | | | | | | | | | | | | | |
| | AABB | | | Cylinder | | | Hybrid (5) | | | Hybrid (10) | | | Hybrid (15) | | |
| **r** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** |
| **10** | 0.32 | 0.30 | **33** | 4.95 | 16.66 | 148 | 3.59 | 0.83 | 58 | 4.53 | 1.48 | 67 | 5.17 | 2.44 | 76 |
| **15** | 0.47 | 0.29 | **33** | 7.44 | 14.76 | 140 | 5.27 | 0.84 | 57 | 6.66 | 1.33 | 65 | 7.04 | 2.06 | 73 |
| **25** | 0.81 | 0.31 | **33** | 12.72 | 13.74 | 128 | 8.65 | 0.76 | 56 | 11.10 | 1.44 | 63 | 11.92 | 1.93 | 70 |
| **Sibenik Cathedral, 80 479 triangles** | | | | | | | | | | | | | | | |
| | AABB | | | Cylinder | | | Hybrid (5) | | | Hybrid (10) | | | Hybrid (15) | | |
| **r** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** |
| **10** | 0.08 | 2.03 | **30** | 1.35 | 77.30 | 122 | 0.85 | 4.41 | 68 | 1.13 | 6.82 | 71 | 1.29 | 11.93 | 73 |
| **15** | 0.12 | 1.87 | **29** | 2.41 | 76.11 | 114 | 1.33 | 5.39 | 67 | 1.82 | 6.60 | 70 | 1.84 | 9.16 | 71 |
| **25** | 0.21 | 1.41 | **29** | 3.62 | 62.63 | 106 | 2.24 | 4.74 | 65 | 2.87 | 5.52 | 67 | 3.36 | 9.51 | 68 |
| **Park, 29 172 triangles** | | | | | | | | | | | | | | | |
| | AABB | | | Cylinder | | | Hybrid (5) | | | Hybrid (10) | | | Hybrid (15) | | |
| **r** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** | **build** | **RT** | **SA** |
| **10** | 0.03 | 0.99 | **22** | 0.48 | 80.51 | 72 | 0.30 | 2.77 | 38 | 0.39 | 2.99 | 43 | 0.44 | 6.79 | 46 |
| **15** | 0.05 | 1.06 | **21** | 0.80 | 77.32 | 68 | 0.48 | 2.79 | 37 | 0.66 | 4.27 | 42 | 0.80 | 6.43 | 44 |
| **25** | 0.07 | 1.01 | **21** | 1.29 | 71.29 | 67 | 0.75 | 2.80 | 37 | 1.03 | 4.02 | 41 | 1.22 | 5.51 | 45 |

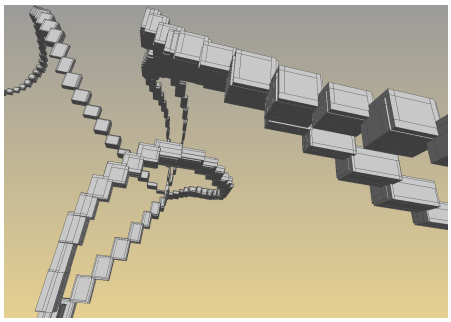
**Table 7.2:** Displaying the build and render times along with the relative surface area of hierarchy alternatives for different radii. Surface areas (SA) are shown relative to the surface are of the scene's AABB. For some of the scenes, certain radii were not considered due to their tendency to produce extremely long builds and renders, while not showing notable improvement. Each measurement had a time limit of 2 hours - some configurations were unable to produce a hierarchy in that time, others only failed to finish the rendering process. The lowest surface area for each radius (each row) is emphasized in bold. Measured times are in minutes and every BVH contains one primitive per leaf.



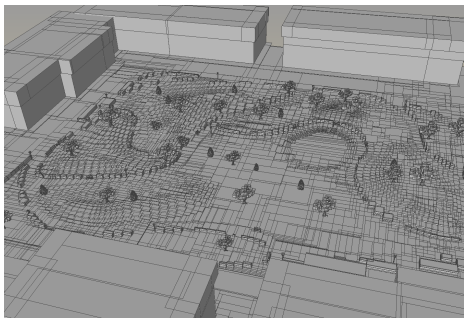
**(a) :** Hairball detail



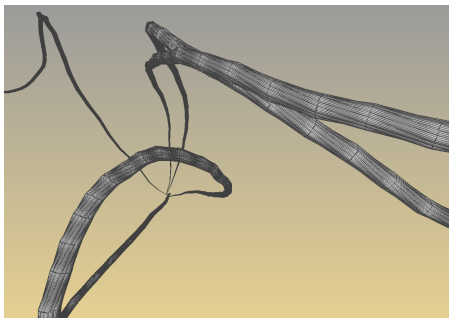
**(b) :** Park



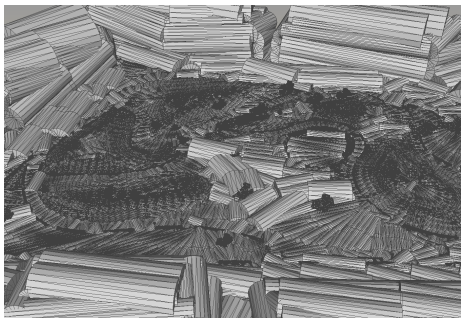
**(c) :** Hairball detail with AABBs



**(d) :** Park with AABBs



**(e) :** Hairball detail with cylinders



**(f) :** Park with cylinders

**Figure 7.5:** Illustrating the differences in bounding volume tightness for a more suitable and an unsuitable scene. Each triangle is bounded with a single bounding volume. The Hairball fraction (left) wrapped with AABBs shows volumes enclosing a large amount of extra space, while the cylinder bounding volumes are inconspicuous. However, the Park scene (right) demonstrates the exact opposite. AABBs seem to wrap primitives relatively tightly, while cylinders inflate the initial surface areas.

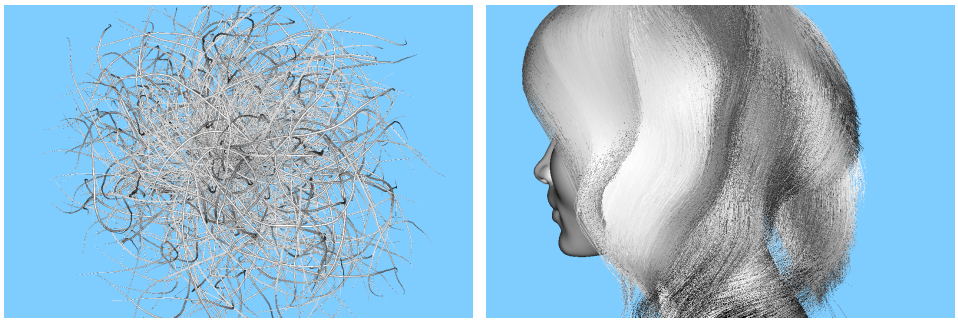
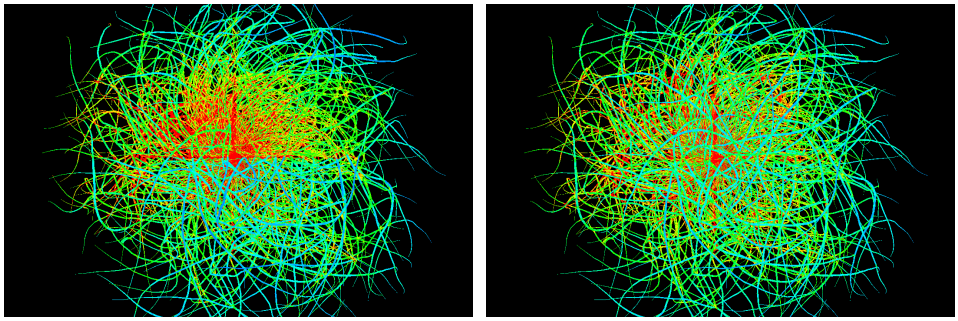
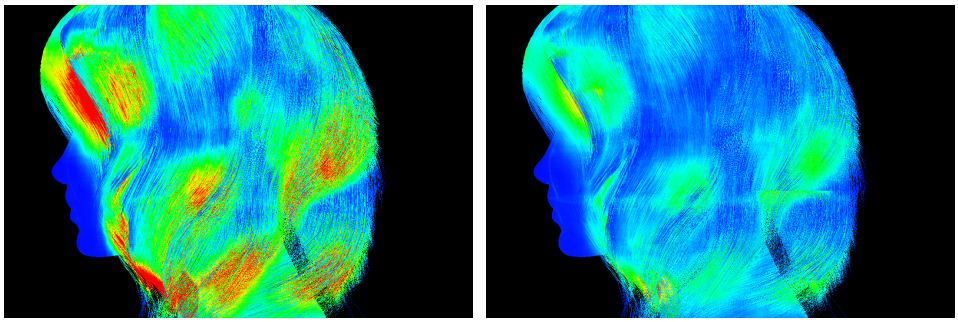


**Figure 7.6:** Rendered image of the 10% Hairball scene (left) and Bust scene (right)



**(a) :** AABB $r = 10$
avg. # steps = 60.13
avg. # intersections = 9.26

**(b) :** Hybrid $r = 10$, $i = 5$
avg. # steps = 64.14
avg. # intersections = 3.56



**(c) :** AABB $r = 25$
avg. # steps = 436.47
avg. # intersections = 307.59

**(d) :** Hybrid $r = 25$, $i = 5$
avg. # steps = 425.39
avg. # intersections = 16.20

**Figure 7.7:** Comparing the rendering statistics of the AABB (left) and hybrid (it. 5) hierarchies for the Hairball 10% and the Bust models. The sum of traversal steps and intersection tests are mapped to colors for each pixel. For the Hairball, red corresponds to $\sim 4000$ steps and intersection tests combined, with the minimum (dark blue) being $\sim 20$. For the Bust, red corresponds to $\sim 12800$, while the dark blue corresponds to $\sim 40$.

# Chapter 8

# Conclusion

The goal of this project was to improve the qualities of bounding volume hierarchies for oblong objects. Various bounding volume types were discussed, with cylinders and truncated cones selected for further examination. It has been shown that cylinders are superior to truncated cones in many aspects. Cylinder-shaped bounding volumes were then selected to be incorporated into the locally-ordered clustering algorithm of Bittner and Meister [2].

The implementation is a proof of concept lacking optimization. However, the presented results support the idea of non-standard bounding volumes being beneficial under certain circumstances. The proposed algorithm is able to construct cylinder-shaped bounding volumes for scene primitives and for bounding volumes of the same type. For volumes similar in size and orientation, the resulting bounding cylinders generally have a lower surface area than the compared axis-aligned bounding boxes (AABBs). The results confirm the importance of the orientation and size similarity of bounded objects. After exploring the cylinder hierarchy, the idea of a hybrid hierarchy was introduced with minor success. The experiments show that combining the powerful surface area reduction of cylinders in the lowest levels of a hierarchy with the simplicity of AABBs results in notable surface area reduction. The modified algorithm closely follows the original clustering implementation [17], providing alternatives for the common BVH. The alternative hierarchies show potential to significantly decrease the BVH surface area for scenes comprising oblong objects.

Generally, cylinder hierarchies perform the worst among all discussed variants, producing extremely long render times. Because the introduced

bounding volume is more complicated than an axis-aligned bounding box, improving the build times is infeasible. The alternative hierarchy proved to be unsuitable for most scene types. However, the Bust scene delivered promising results, with remarkably low surface areas and the hybrid hierarchies decreasing render times in exchange for highly increased build times.

This thesis has shown that using cylinders as bounding volumes could be a viable solution for scenes containing hair and fur. However, it would be necessary to fully optimize the bounding volume computation, the clustering and the ray traversal. Ideally, a parallel GPU version would be fast enough for practical use.

# Appendix A

## Diagrams

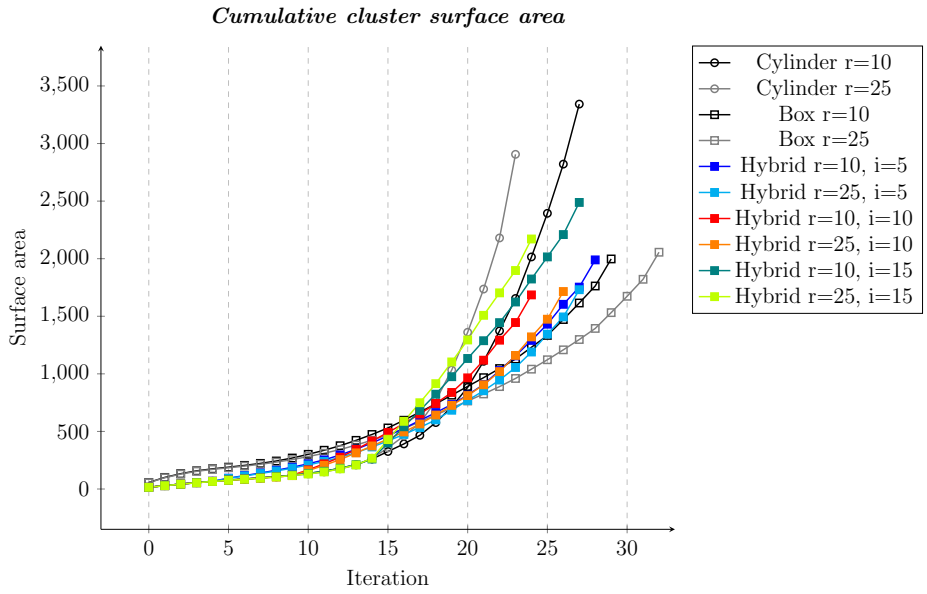



**Figure A.1:** Cumulative cluster surface areas of all three types of BVHs at selected radii. The hybrid version displays slower growth than the cylinders, yet faster than the AABB hierarchy. Scene: Hairball 0.1%.

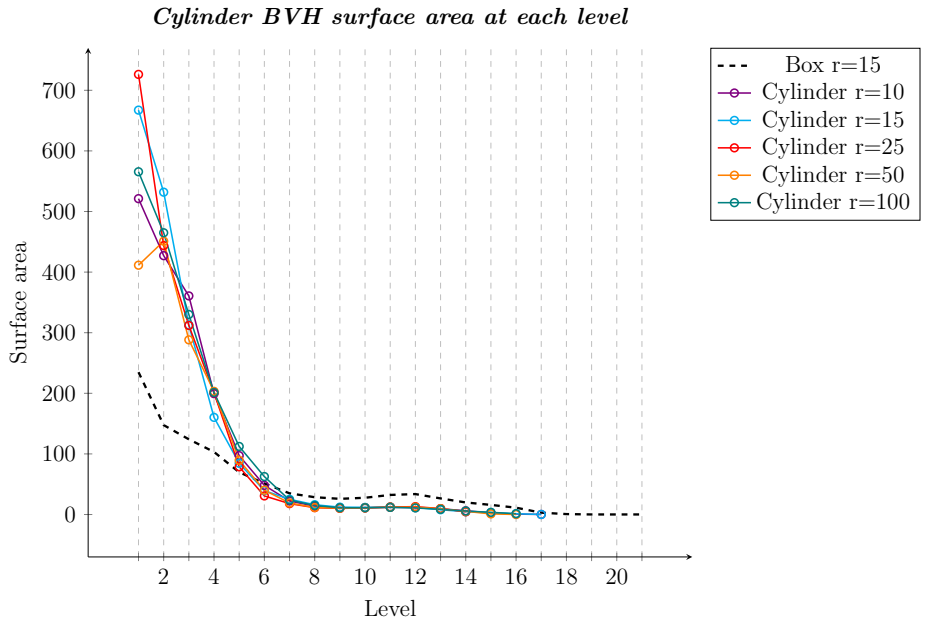



**Figure A.2:** Comparing the BVH surface area on each level of the cylinder hierarchy with the AABB version. Scene: Hairball 0.1%.

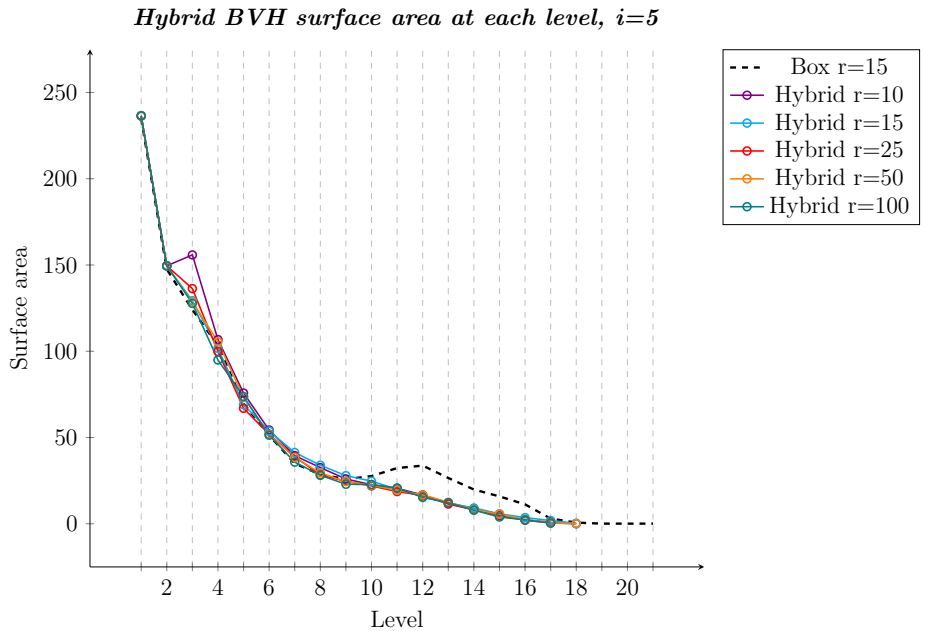



**Figure A.3:** BVH surface area by levels in hybrid hierarchies, type transition at iteration 5. Scene: Hairball 0.1%.

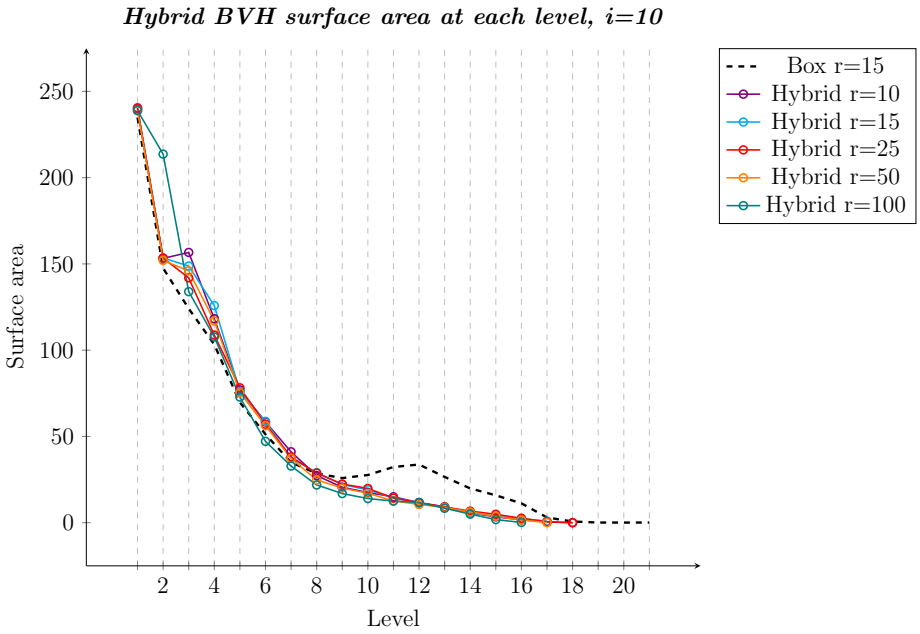



**Figure A.4:** BVH surface area by levels in hybrid hierarchies, type transition at iteration 10. Scene: Hairball 0.1%.

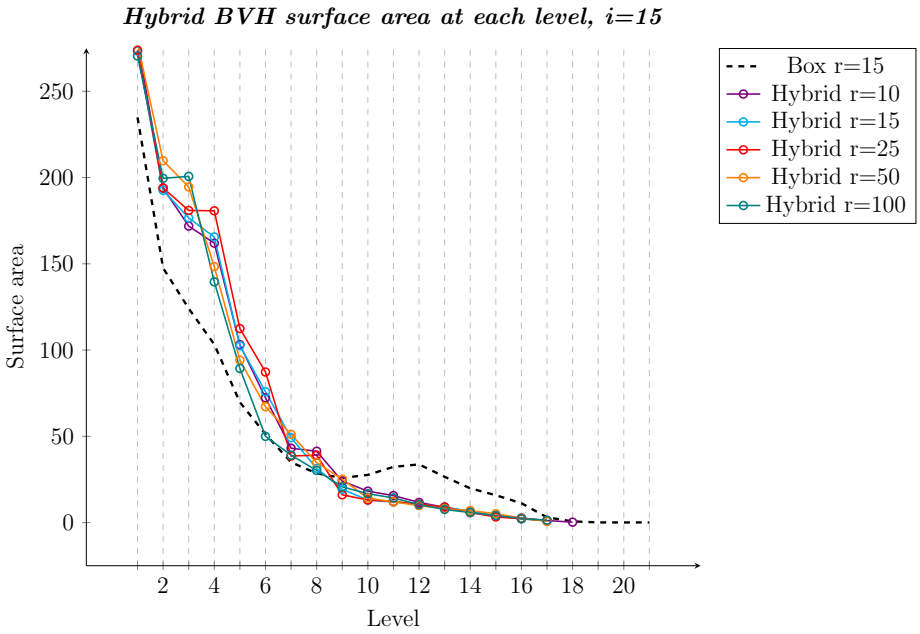



**Figure A.5:** BVH surface area by levels in hybrid hierarchies, type transition at iteration 15. Scene: Hairball 0.1%.

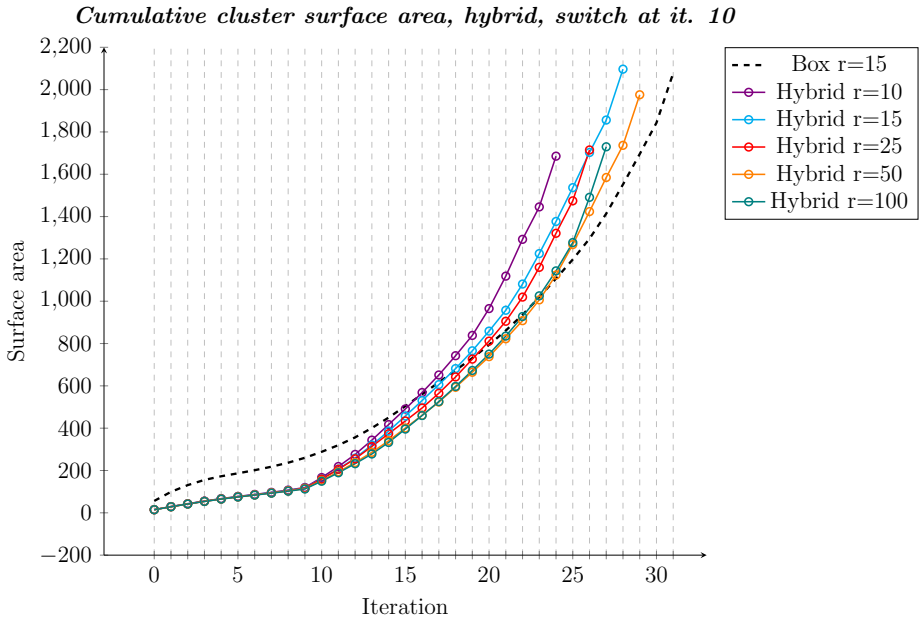



**Figure A.6:** Comparing the cluster surface areas by iteration to the box hierarchy, switch at iteration 15. Scene: Hairball 0.1%.

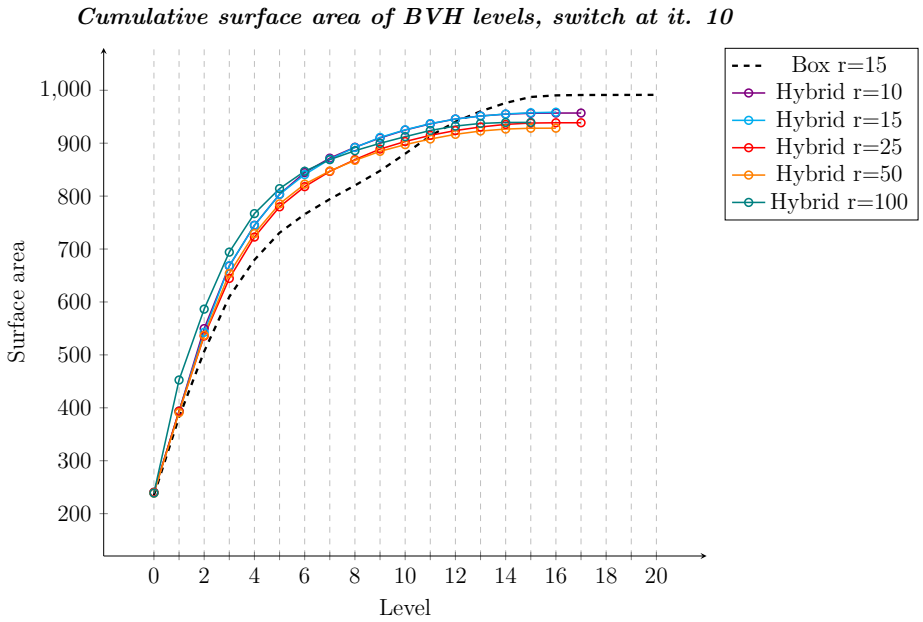



**Figure A.7:** Cumulative BVH surface area of the hybrid hierarchy levels compared to the AABB hierarchy. Scene: Hairball 0.1%.

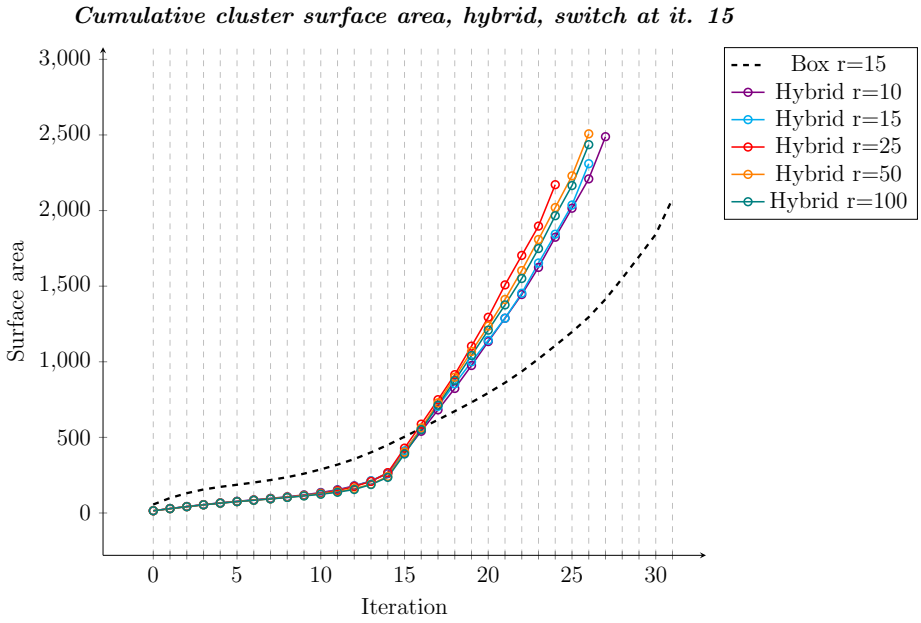



**Figure A.8:** Cluster surface areas by iteration compared to the box hierarchy, switch at iteration 15. Scene: Hairball 0.1%.

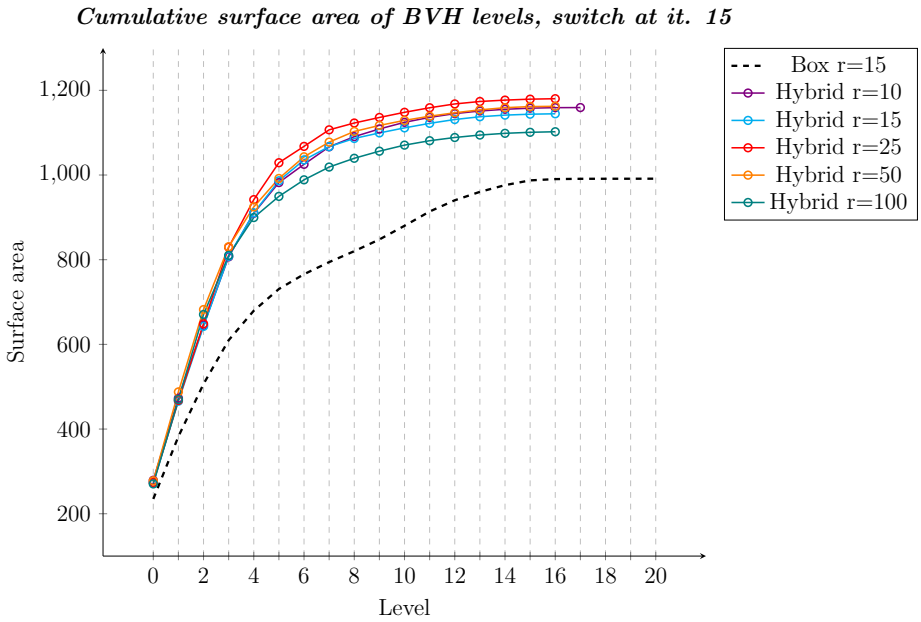



**Figure A.9:** Cumulative BVH surface area of the hybrid hierarchy levels compared to the AABB tree. Scene: Hairball 0.1%.

# Appendix B

# Bibliography


[1] Sven Woop, Carsten Benthin, Ingo Wald, Gregory Johnson, and Eric Tabellion. Exploiting local orientation similarity for efficient ray traversal of hair and fur. 06 2014.

[2] D. Meister and J. Bittner. Parallel locally-ordered clustering for bounding volume hierarchy construction. *IEEE Transactions on Visualization and Computer Graphics*, 24:1345–1353, 2018.

[3] Eric Larsen, Stefan Gottschalk, Ming Lin, and Dinesh Manocha. Fast proximity queries with swept sphere volumes. 12 2000.

[4] Ingo Wald, Nate Morrical, Stefan Zellmann, Lei Ma, Will Usher, Tiejun Huang, and Valerio Pascucci. Using hardware ray transforms to accelerate ray/primitive intersections for long, thin primitive types. *Proc. ACM Comput. Graph. Interact. Tech.*, 3(2), August 2020.

[5] R. Fonseca and P. Winter. Bounding volumes for proteins: A comparative study. *Journal of computational biology: a journal of computational molecular cell biology*, 19 10:1203–13, 2012.

[6] Steven M. Rubin and Turner Whitted. A 3-dimensional representation for fast rendering of complex scenes. *SIGGRAPH Comput. Graph.*, 14(3):110–116, July 1980.

[7] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '86, page 269–278, New York, NY, USA, 1986. Association for Computing Machinery.

[8] Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Computer Graphics and Applications*, 7(5):14–20, 1987.

[9] Timo Aila, Tero Karras, and Samuli Laine. On quality metrics of bounding volume hierarchies. In *Proceedings of the 5th High-Performance Graphics Conference*, HPG '13, page 101–107, New York, NY, USA, 2013. Association for Computing Machinery.

[10] Martin Stich, Heiko Friedrich, and Andreas Dietrich. Spatial splits in bounding volume hierarchies. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, page 7–13, New York, NY, USA, 2009. Association for Computing Machinery.

[11] Stefan Popov, Iliyan Georgiev, Rossen Dimov, and Philipp Slusallek. Object partitioning considered harmful: Space subdivision for bvhs. In *Proceedings of the Conference on High Performance Graphics 2009*, HPG '09, page 15–22, New York, NY, USA, 2009. Association for Computing Machinery.

[12] Manfred Ernst and Gunther Greiner. Early split clipping for bounding volume hierarchies. In *2007 IEEE Symposium on Interactive Ray Tracing*, pages 73–78, 2007.

[13] Ingo Wald. On fast construction of sah based bounding volume hierarchies. pages 33 – 40, 10 2007.

[14] Xin Sun, Kun Zhou, Stephen Lin, and Baining Guo. Line space gathering for single scattering in large scenes. *ACM Trans. Graph.*, 29(4), July 2010.

[15] G.M. Morton. *A Computer Oriented Geodetic Data Base and a New Technique in File Sequencing*. International Business Machines Company, 1966.

[16] H. Täubig, B. Bäuml, and U. Frese. Real-time swept volume and distance computation for self collision detection. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1585–1592, 2011.

[17] Arsène Pérard-Gayot. Header-only C++ BVH library, https://github.com/madmann91/bvh, 2020.

[18] Denis Zorin. Ray tracing II (lecture notes), 1999.