

**BOUNDING VOLUME
HIERARCHY-TETRAHEDRALIZATION
HYBRID ACCELERATION STRUCTURE
FOR RAY TRACING**

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
COMPUTER ENGINEERING

By
Serkan Demirci
December 2020

Bounding Volume Hierarchy-Tetrahedralization Hybrid Acceleration
Structure for Ray Tracing

By Serkan Demirci

December 2020

We certify that we have read this thesis and that in our opinion it is fully adequate,
in scope and in quality, as a thesis for the degree of Master of Science.

Uğur Gdkbay(Advisor)

zcan ztrk

Yusuf Sahilliođlu

Approved for the Graduate School of Engineering and Science:

Ezhan Karařan
Director of the Graduate School

ABSTRACT

BOUNDING VOLUME
HIERARCHY-TETRAHEDRALIZATION HYBRID
ACCELERATION STRUCTURE FOR RAY TRACING

Serkan Demirci
M.S. in Computer Engineering
Advisor: Uğur Güdükbay
December 2020

The computational cost of the ray-tracing method is directly proportional to the number of ray-surface intersection tests. The naive ray-tracing algorithm requires $\mathcal{O}(N)$ computational cost for the ray-surface intersection calculations where N is the number of primitives in the scene. Ray tracing acceleration data structures like the regular grid, bounding volume hierarchy (BVH), kd-tree, constrained tetrahedralization, has been developed to reduce the number of ray-object intersection tests to speed-up ray tracing.

We propose a hybrid acceleration structure, the Bounding Volume Hierarchy-Tetrahedral mesh hybrid (BTH) acceleration structure, that can be used to speed-up ray tracing. BTH structure is composed of a BVH hierarchy where some of the leaves of the BVH hierarchy contain tetrahedralizations. We propose an algorithm for the construction of the BTH structure. We describe methods for approximating the average nearest-hit cost of a tetrahedralization, which we use for the construction of BTH. Besides, we can adapt the proposed BTH structure for dynamic scenes with hierarchical motion. We describe a two-level BVH-BTH acceleration structure for rendering animated scenes.

We test the proposed BTH structure using various scenes. For some of the experiments, the BTH structure performs better against other acceleration structures in terms of rendering times. We perform experiments for animated scenes. We show that the two-level BTH structure outperforms the two-level BVH structure for the tested dynamic scenes.

Keywords: Ray tracing, acceleration structure, tetrahedralization, Bounding Volume Hierarchy, k-d tree.

ÖZET

IŞIN İZLEME İÇİN SINIRLAYICI HACİM HİYERARŞİSİ-DÖRTYÜZLEME HİBRİT HIZLANDIRICI YAPISI

Serkan Demirci

Bilgisayar Mühendisliği, Yüksek Lisans

Tez Danışmanı: Uğur Güdükbay

Aralık 2020

Işın izleme yönteminin hesaplama maliyeti ışın-yüzey kesişim testlerinin sayısı ile doğru orantılıdır. Işın izleme için ışın-yüzey kesişim testlerinin hesaplama zamanı naif algoritma için $\mathcal{O}(N)$ olup N sahnedeki nesne sayısıdır. Düzenli ızgara, sınırlayıcı hacim hiyerarşisi (BVH), kd ağacı, kısıtlı dört yüzlüleştirme gibi ışın izleme hızlandırma veri yapıları, gerçek zamanlı ışın izleme elde etmek için ışın-nesne kesişim testlerinin sayısını azaltmak için geliştirilmiştir.

Işın izlemeyi hızlandırmak için kullanılabilecek bir hibrit hızlandırma yapısı, Sınırlayıcı Hacim Hiyerarşisi-Dörtüyzleme hibrit (BTH) hızlandırma yapısı öneriyoruz. BTH yapısı, BVH hiyerarşisinin bazı yapraklarının dörtüyzlü örgüler içerdiği bir BVH hiyerarşisinden oluşur. BTH yapısının oluşturulması için bir algoritma öneriyoruz. BTH yapısının oluşturulması için kullandığımız, dörtüyzlüleştirmenin ortalama en yakın isabet maliyetini tahmin etmek için yöntemler sunuyoruz. Ayrıca, önerilen BTH yapısını hiyerarşik hareketli dinamik sahneler için uyarlıyoruz. Dinamik sahneleri oluşturmak için iki seviyeli bir BVH-BTH hızlandırma yapısı sunuyoruz.

Önerilen BTH yapısını çeşitli sahneler kullanarak test ediyoruz. Bazı durumlarda BTH yapısı, işleme süreleri açısından diğer hızlandırma yapılarına göre daha iyi performans göstermektedir. Hareketli sahneler için deneyler yapıyoruz. İki seviyeli BTH yapısının iki seviyeli BVH yapısından daha iyi performans göstermektedir.

Anahtar sözcükler: Işın izleme, hızlandırıcı yapısı, dörtüyzleme, Sınırlayıcı Hacim Hiyerarşisi, k-d ağacı.

Acknowledgement

I would like to thank my advisor Prof. Dr. Uğur Gdkbay for his invaluable guidance throughout my research.

I would like to thank Aytk Aman providing feedback throughout all phases of my work.

I would like to thank all my friends for being helpful and supportive during my thesis study. Finally, my greatest thanks to my family. I am forever grateful.

This work is supported by the Scientific and Technological Research Council of Turkey (TBTAK) under Grant No. 117E881.

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Contributions	2
1.3	Organization of the Thesis	3
2	Background and Related Work	4
2.1	Grids	4
2.2	<i>K</i> -d Trees	6
2.3	Bounding Volume Hierarchies	7
2.4	Tetrahedralizations	9
3	Bounding Volume Hierarchy-Tetrahedralization Hybrid (BTH) Acceleration Structure	12
3.1	Construction	13
3.2	Traversal	16

4	Tetrahedralization Nearest-Hit Cost Heuristic	18
4.1	Sampling-based Cost Calculation	19
4.2	Average Depth-based Cost Calculation	20
4.3	Primitive Count-based Cost Calculation	22
5	Animated Scenes Using the BTH Structure	26
5.1	Two-Level BTH for Hierarchical Motion	27
6	Experimental Results	30
6.1	Experimental Setup	30
6.2	Selection of Tetrahedralization Cost Method	31
6.3	Results	35
6.3.1	Static Scenes	35
6.3.2	Dynamic Scenes	39
7	Conclusions and Future Research Directions	43
	Bibliography	45

List of Figures

2.1	A uniform grid structure	5
2.2	A k -d tree structure	6
2.3	A bounding volume hierarchy (BVH).	8
2.4	Nearest-hit query on tetrahedralization	10
3.1	Bounding Volume Hierarchy-Tetrahedralization hybrid (BTH) acceleration structure.	13
3.2	Finding the initial tetrahedron in a bounded tetrahedralization.	17
4.1	The average depth estimation using depth map.	21
4.2	The comparison of the estimated tetrahedron count using the average depth-based method and the sampling-based method.	22
4.3	The histogram of the relative difference of the tetrahedron count between the average depth-based and sampling-based estimation methods.	23
4.4	The comparison of the estimated tetrahedron count using the average depth-based method and the sampling-based method.	24

4.5	The histogram of the relative difference of the tetrahedron count between the primitive count-based and the sampling-based estimation methods.	25
5.1	An instancing example.	28
5.2	Two-Level BVH-BTH structure.	29
6.1	Tetrahedralizations of the BTH structure constructed using different C_{tet} values.	32
6.2	Rotating armadillo animated scene rendered using the two-level BVH-BTH structure.	40
6.3	Random motion animated scene rendered using the two-level BVH-BTH structure.	41
6.4	Falling airballs animated scene rendered using the two-level BVH-BTH structure.	42

List of Tables

6.1	BTH32 statistics for varying C_{tet}	33
6.2	BTH20 statistics for varying C_{tet}	34
6.3	The comparison of the construction and rendering times for different acceleration structures on scenes that cannot be directly tetrahedralized.	37
6.4	The comparison of the construction and rendering times for different acceleration structures on scenes that can be directly tetrahedralized.	38
6.5	The comparison of the construction and rendering times for different acceleration structures for the animated scenes.	39

Chapter 1

Introduction

1.1 Motivation

Ray tracing [1] is a rendering method that generates photo-realistic images by simulating the interaction of lightpaths between objects. One of the most fundamental operations used in ray tracing is ray-surface intersection tests. In ray tracing, a significant amount of computational time is dedicated to finding the nearest surface that a ray hits. With the introduction of path tracing [2] for global illumination, the importance of fast nearest hit tests increased. Path tracing is a form of ray tracing that uses Monte Carlo method to generate realistic images. Path tracing requires large number of lightpaths to be simulated. The quality of the generated image in path tracing directly depends on the number of simulated lightpaths.

Given a scene and a ray, a naive way to find the nearest-hit requires $\Theta(N)$ time, where N is the number of primitive surfaces in the scene. $\Theta(N)$ time is slow for most of the ray-tracing algorithms. To speed-up the ray-surface intersection tests, various spatial acceleration structures, such as regular grids, octrees, k-d trees, bounding volume hierarchies, are proposed in the literature. Such spatial acceleration structures reduce the number of ray-surface intersection tests by

eliminating some of the candidate primitives. In ray tracing, substantial gains can be achieved by using such acceleration structures.

One alternative spatial acceleration structure is tetrahedralizations. Lagae and Dutré [3] proposed the use of constrained tetrahedralizations as spatial acceleration structure. Although constrained tetrahedralizations are alternatives to well-known acceleration structures, the limitations of tetrahedralization algorithms make them limited in real-world applications.

We propose a BVH-Tetrahedral mesh hybrid acceleration structure (BTH) as a spatial acceleration structure. BTH structure combines the strength of two acceleration structures BVH and tetrahedralizations.

1.2 Contributions

The main contributions of the thesis are as follows.

- We propose a hybrid acceleration structure, BTH, which is composed of a BVH and tetrahedralizations.
- We propose a construction method for the new hybrid structure
- propose methods to find approximate nearest-hit cost for the tetrahedralization acceleration structure.
- We show that just like BVH structure, hybrid structure can be adapted to the dynamic scenes.

1.3 Organization of the Thesis

The organization of the thesis is as follows. Chapter 2 summarizes the existing acceleration structures, their strengths, and weaknesses. Chapter 3 introduces construction and nearest-hit traversal algorithms of the BTH acceleration structure. Chapter 4 describes the nearest-hit cost approximation methods for tetrahedralization acceleration structure, which is used by BTH construction algorithm. Chapter 5 shows how BTH structure can be modified to be used for dynamic scenes. Chapter 6 presents the experimental setup and results of the experiments.

Chapter 2

Background and Related Work

The visibility test is one of the most fundamental operations for ray tracing. The naive, brute-force, visibility test algorithm checks every primitive in the scene with the ray to find the nearest intersection. The time complexity of the naive visibility algorithm ($\Theta(N)$) makes it impractical for complex scenes. Acceleration structures reduce the complexity of visibility tests by decreasing the number of ray-primitive tests required. In this section, we discuss grids, k -d trees, and BVHs that are widely used in modern ray tracers. We also discuss tetrahedralizations as another acceleration structure for ray-surface intersections. We discuss the strengths and weaknesses of different acceleration structures.

2.1 Grids

One of the basic nearest hit acceleration structure is uniform grids. Uniform grids [4] subdivide the scene space into uniform-sized cells (cf. Figure 2.1). Each cell stores a list of primitives that occupy the cell. Multiple cells can contain the same primitive if such primitive spans multiple cells. The nearest-hit traversal algorithm on grids, often referred to as the three-dimensional digital differential

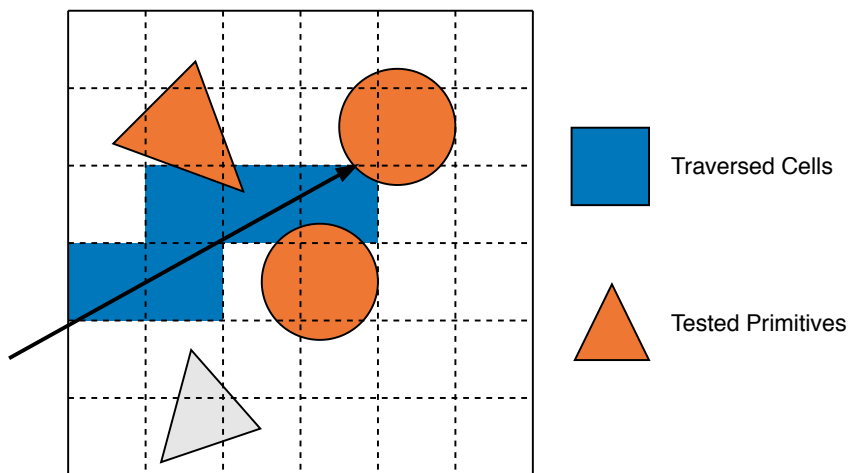


Figure 2.1: A uniform grid structure

analyzer (3D-DDA) algorithm, visits each cell along the ray. In nearest-hit traversal, as each cell is traversed, all of the primitives in the cell's list are checked for intersections. The traversal stops when a cell contains a primitive that intersects the ray. Since cells are checked in ray's order, starting from the cell that ray's origin resides in, along the ray's direction.

The nearest-hit performance of uniform grids directly depends on the size of the cells. If cells are large, the grid will be less effective in reducing the number of ray-primitive intersection tests. On the other hand, if cells are small, more cells are needed to be traversed before finding the nearest-hit.

One other factor that affects the performance of uniform grids is the distribution of primitives in the scene. Since uniform grids do not adapt to the distribution of primitives in the scene, the distribution of primitives in the scene directly affects the performance of the uniform grid. If primitives in the scene are distributed unevenly, occupancy of the cells will also be unevenly distributed. Empty cells and overpopulated cells will decrease the performance of the cells.

For the optimal grid sizes, different analyses are done in the literature ([5], [6], [7] and [8]). Clearly and Wyvill [5] showed that for scenes with N evenly distributed small, equally sized primitives, optimal case is to subdivide each axis

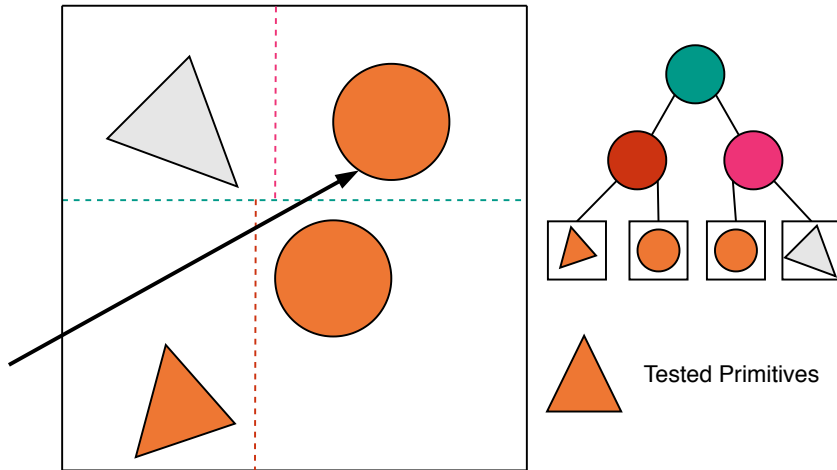


Figure 2.2: A k -d tree structure

into $\sqrt[3]{N}$ cells, with minimum average $\Theta(\sqrt[3]{N})$ time and $\mathcal{O}(N)$ space complexity.

2.2 K -d Trees

In k -d trees [9] scene is recursively divided into two half-spaces (cf. Figure 2.2). A primitive is associated with a half-space if the half-space contains a part of the primitive. If a primitive lies in both half-spaces it is included in both sub-trees.

The nearest-hit traversal in k -d trees starts from the root, in each split, half-spaces are tested for intersection in the ray's traversal order. Traversal descends through the tree until a terminal node is reached. If a terminal node is encountered, each primitive inside the terminal node is tested for intersection. Similar to grids, traversal stops when the first primitive intersection is found in a terminal node. The structure of the k -d trees and the axis and position of the splitting planes affect the nearest-hit test performance. In general, k -d trees are constructed in a top-down manner using heuristics to find the best splitting plane. Given a split, heuristics estimate the cost of nearest-hit tests. Heuristics are also used as a terminating criterion for k -d tree construction.

Surface area heuristic (SAH) [10, 11] is the most commonly used heuristic for the construction of k -d trees. SAH estimates the expected cost of the nearest-hit test for a long uniformly distributed random ray. Given a split s , the SAH cost is calculated as

$$Cost_{SAH}(s) = \frac{C_i}{SA(s)}(n_l SA(l) + n_r SA(r)) + C_t, \quad (2.1)$$

where the SA function calculates the surface area of the node’s bounding box. C_i constant is the cost of the ray-primitive intersection, C_t constant is the cost of traversing a node. n_l , and n_r are the number of primitives in the left and right half-spaces. The SAH-based k -d tree construction further improved by [12] and [13].

2.3 Bounding Volume Hierarchies

One of the popular spatial acceleration structures is bounding volume hierarchies (BVHs). BVHs are composed of a hierarchy of partitions where each partition is represented by a volume that encloses all of the primitives in that partition (cf. Figure 2.3). Unlike other acceleration structures, BVHs partition primitives instead of subdividing the scene space. Therefore, bounding volumes can intersect. Because BVHs partition primitives, primitives are not duplicated in BVHs, unlike k -d trees.

Bounding volumes in the hierarchy provide a simple method for finding the nearest-hit. If a ray does not pass through a bounding volume, primitives represented by such bounding volume are not needed to be tested. Therefore, the nearest-hit algorithm (see Algorithm 1) recursively traverses the hierarchy in depth-first order. If the algorithm encounters an inner node, the algorithm checks whether a ray passes through the bounding volume. If the ray does not pass through the bounding volume, the algorithm does not traverse its children. If the algorithm encounters a leaf node, the algorithm checks each primitive with

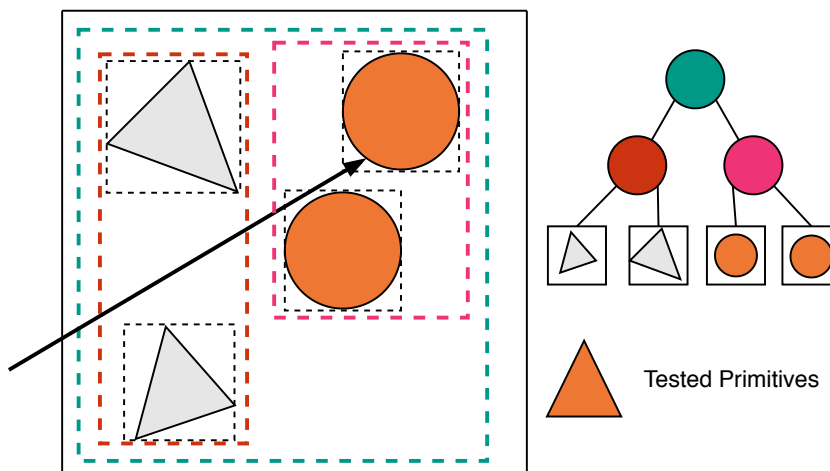


Figure 2.3: A bounding volume hierarchy (BVH).

the ray for an intersection. Since BVHs bounding volumes can intersect, unlike space subdivision acceleration structures, the algorithm does not stop when the first intersection is found. The algorithm needs to test all possible nodes, before finding the nearest-hit.

Algorithm 1 Recursive BVH Nearest-hit traversal

```

1: procedure BVHINTERSECT(node, ray)
2:   if node is a leaf then
3:     for all primitive in node do
4:       INTERSECT(primitive, ray)
5:   else
6:     for all children c of node do
7:       BVHINTERSECT(c, ray)

```

BVHs are first used by Clark [14] for visible surface determination. Later Rubin and Whitted [15] used them to accelerate ray tracing. In early BVH research, BVHs are constructed by hand. Kay and Kajiya [16] proposed an automatic top-down construction of BVHs using median splits. Goldsmith and Salmon [10] proposed SAH measure and used incremental insertions for construction. Incremental construction with SAH measure leads to poor quality BVHs. Müller and Felner [17] used SAH to build a BVH in a top-down manner. Later Wald et al. [18] used centroid-based SAH partitioning to improve top-down construction with $\Theta(N \log^2(N))$ average construction complexity. Later streamed

binning idea from k -d trees [19] are applied to BVHs by Wald [20] to further improve the runtime of the construction algorithm.

In general, modern BVH structures use Axis Aligned Bounding Boxes (AABB) as bounding volumes. The BVH tree is constructed from top to bottom using Surface Area Heuristics (SAH) to find the good primitive partitions.

2.4 Tetrahedralizations

Lagae and Dutré [3] first proposed constrained tetrahedralizations to accelerate nearest-hit tests for ray tracing. In two dimensions, the various forms of triangulation algorithms are defined as follows.

- *Triangulation* of a set of points S in a plane is a partition of the region of the convex hull of S into non-overlapping triangles, such that vertices of triangles are a superset of S .
- *Delaunay triangulation* of set of points S is a triangulation of S such that every triangle is Delaunay [21].
- *Constrained Triangulation* of points S , segments P is a triangulation that conforms points and segment constraints. A triangulation conforms to a constraint if it contains that constraint as a part of a triangle.
- *Constrained Delaunay Triangulation* of S points and constraints is a triangulation that whose triangles are constrained Delaunay and it conforms to given constraints.

Tetrahedralizations are three-dimensional equivalents of triangulations in a plane. Similarly, constrained tetrahedralization of a set of points, segments, and faces is a tetrahedralization that conforms to given constraints; points, segments, and faces. Many tetrahedralization algorithms assume that input constraints

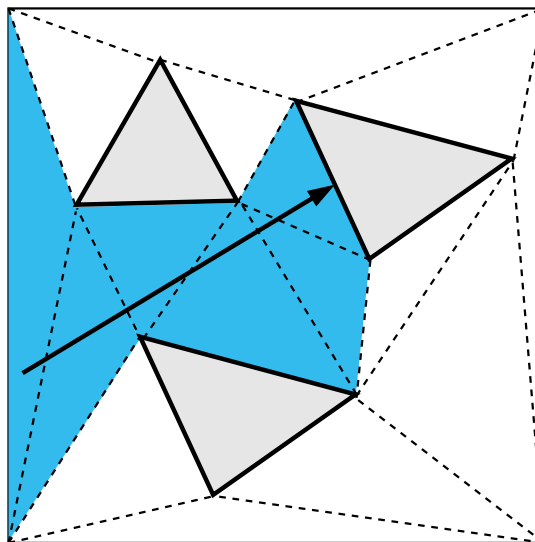


Figure 2.4: Nearest-hit query on tetrahedralization

are formalized as Piecewise Linear Complex (PLC) [22]. A PLC in the three-dimensional context consists of simplices; points, segments, and faces, such that the intersection of simplices in PLC is also in the PLC. If the two faces of a PLC intersect, its intersection (a point or line) must also be present inside the PLC. In the context of ray tracing, scenes are usually represented as a triangle soup. A triangle soup can trivially be converted into a PLC if the soup does not contain self-intersections. If a triangle soup contains self-intersections, these self-intersections must be eliminated by triangulating input triangles. TetGen [23] is a popular tool that is commonly used for constrained tetrahedralization of arbitrary meshes. Given a PLC, TetGen uses a combination of boundary constrained methods [24] and Delaunay refinement method described by Ruppert [25] and Shewchuk [26] to calculate constrained Delaunay triangulation. For robustness, TetGen uses a mixture of static filters and Shewchuk’s filtered exact geometric predicates [27]. Recently, Hu et al.’s TetWild [28] and fTetWild [29] tetrahedralization algorithms reformulated the constrained tetrahedralization problem to allow triangle soup inputs. Although their algorithm accepts any kind of geometry that can be represented by a triangle soup, their algorithm outputs approximate tetrahedralization of the input geometry.

In the context of ray tracing, constrained tetrahedralizations are used as an acceleration structure as follows (cf. Figure 2.4). Given a scene, a constrained tetrahedralization of the scene is constructed. Faces in the scene are considered as constrained faces in tetrahedralization. Nearest-hit tests in the tetrahedralization structure are similar to grids. Starting from the source tetrahedron, which is a tetrahedron that contains the origin of the ray, each tetrahedron is traversed in order using shared faces, one tetrahedron at a time. Unlike grids, instead of storing input faces inside cells, in constrained tetrahedralization, input faces are stored as some of the faces of tetrahedra. When a ray hits such a face, the traversal is stopped and the face is reported as nearest-hit. For the determination of the next tetrahedron in traversal, Lagae and Dutré [3] used the scalar triple product method. To improve traversal efficiency Maria et al. [30] proposed the use of Plücker coordinates for determination of next tetrahedra. Aman et al. [31] proposed an efficient and compact tetrahedralization structure for ray tracing. They proposed *Tet32*, *Tet20*, *Tet16* tetrahedron storage schemes, along with nearest-hit traversal algorithms for their proposed structures. Their traversal method uses projected 2D ray’s coordinates to efficiently traverse the tetrahedralization. They also reordered tetrahedralization using a space-filling curve to improve cache locality during traversal.

Chapter 3

Bounding Volume Hierarchy- Tetrahedralization Hybrid (BTH) Acceleration Structure

The Bounding Volume Hierarchy-Tetrahedralization Hybrid (BTH) Acceleration structure is composed of a BVH where some leaves of the hierarchy contain tetrahedralizations of the primitives instead of primitive lists (see Figure 3.1). An axis-aligned bounding box encloses each tetrahedralization leaf node. Traversal in BTH structure is similar to BVH traversal. We recursively traverse each node until we reach a leaf node. In the leaf node, if we have a tetrahedralization, we traverse the tetrahedralization. Otherwise, we check each primitive in the leaf for an intersection. The BTH acceleration structure combines the strengths of BVH and tetrahedralization structures. Compared to BVH, the nearest-hit test on the BTH structure is faster because we can terminate the traversal on tetrahedralization leaves early. We can construct the BTH structure for the scenes where scenes cannot be completely tetrahedralized. In the BTH structure, a part of the geometry that can be tetrahedralized can be selected and used as a leaf node. Moreover, the construction of the BTH structure is faster than regular tetrahedralizations due to the non-linear complexity of tetrahedralization algorithms.

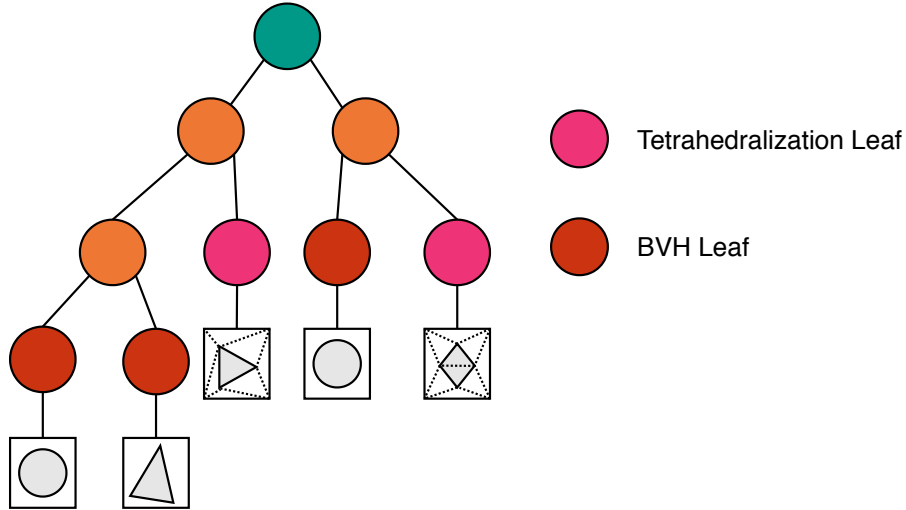


Figure 3.1: Bounding Volume Hierarchy-Tetrahedralization hybrid (BTH) acceleration structure.

In the following sections, we define construction and nearest-hit traversal methods for the BTH structure.

3.1 Construction

Similar to any hierarchical acceleration structures that are adaptive to the input geometry, the performance of the nearest-hit traversal of BTH depends on the construction quality. Top-Down BTH construction starts with a regular BVH construction. Given input geometry, we construct a BVH using the existing SAH based construction methods. Then, we modify and convert the constructed BVH structure into a BTH structure.

We first construct a complete BVH for the given geometry. Then we select some nodes of the BVH structure to be a tetrahedralization structure. We trim the selected nodes and construct tetrahedralizations for the nodes. We explain the three steps of the top-down construction algorithm (Algorithm 2) as follows:

Algorithm 2 Top-down BTH construction

```
1: procedure CONSTRUCTBTHTOPDOWN(primitives)
2:   bvh  $\leftarrow$  CONSTRUCTBVHTOPDOWN(primitives)
3:   MARKSUITABLENODES(bvh)
4:   bth  $\leftarrow$  CONSTRUCTTETRAHEDRALIZATIONS(bvh)
5:   return bth
```

First Step: In the first step, we use the binned BVH construction method [32] to construct a BVH in a top-down fashion. We use the SAH heuristic to build the BVH for the scene.

Second Step: In the second step, we mark the suitable nodes that can be tetrahedralized. Fast tetrahedralization algorithms [23] require self-intersection free geometries as input. Although we can remove self-intersections by splitting the faces and adding new primitives, removing the self-intersections slow down the construction algorithm. Instead, we choose not to tetrahedralize the geometries with self-intersecting primitives. In this step, we mark the BVH nodes that do not contain self-intersecting primitives. We call such nodes as *suitable nodes*. We mark suitable nodes by a recursive algorithm that visits each node and checks for self-intersection (see Algorithm 3). The algorithm marks the non-self-intersecting nodes starting from the leaf nodes. In leaf nodes, we check each pair of primitives for self-intersection. If a node contains self-intersecting primitives, its parent must also contain the same self-intersecting primitives. Therefore, if a node is marked as unsuitable, its parent is also marked as unsuitable. For an inner node, the algorithm marks the node suitable if the children of the node do not contain any self-intersecting primitives, and the children nodes do not intersect with each other. To detect the collisions between two children nodes, we use the existing BVH structure. The BVH collision test algorithm (see Algorithm 4) checks whether two BVH nodes intersecting with each other [33]. A descend rule is used to determine the node that should be descended first.

We use the BVH we constructed in the previous step to find the self-intersection free nodes. A recursive algorithm marks nodes containing no self-intersecting primitives, starting from the leaf nodes. If a node is a leaf node, we test each

pair of primitives in the node for self-intersection. If the node is a nonleaf node, it does not contain self-intersecting primitives if and only if both children of the node are non-self-intersecting nodes and the children of the nodes do not intersect with each other.

Algorithm 3 MARKSUITABLENODES

```

1: procedure MARKSUITABLENODES(node)
2:   if node is a leaf node then
3:     if node.primitives has no self intersection then
4:       MARKSUITABLE(node)
5:   else
6:      $l \leftarrow \textit{node.left\_child}$ 
7:      $r \leftarrow \textit{node.right\_child}$ 
8:
9:     MARKSUITABLENODES(l)
10:    MARKSUITABLENODES(r)
11:
12:    if l and r are suitable then
13:      if l.primitives and r.primitives do not intersect then
14:        MARKSUITABLE(node)

```

Algorithm 4 BVHCOLLISION

```

1: procedure BVHCOLLISION(node1, node2)
2:   if !OVERLAP(node1.bounding\_volume, node2.bounding\_volume) then
3:     return
4:   if node1 and node2 are leaf nodes then
5:     CHECKINTERSECTION(node1, node2)
6:   else
7:     if descend node1 then
8:       BVHCOLLISION(node1.left\_child, node2)
9:       BVHCOLLISION(node1.right\_child, node2)
10:    else
11:      BVHCOLLISION(node1, node2.left\_child)
12:      BVHCOLLISION(node1, node2.right\_child)

```

Third Step: In the last step, we select some of the suitable nodes as tetrahedralization leaves. Starting from the root node, a recursive algorithm (cf. Algorithm 5) traverses the BVH and selects nodes that are advantageous to use as tetrahedralization leaves. Using tetrahedralization nearest-hit cost heuristic,

Algorithm 5 CONSTRUCTTETRAHEDRALIZATIONS procedure constructs tetrahedralizations for advantageous BVH nodes.

```

1: procedure CONSTRUCTTETRAHEDRALIZATIONS(node)
2:   if node is marked then
3:     if  $Cost_{tet}(node) < Cost_{SAH}(node)$  then
4:       PRUNE(node)
5:       TETRAHEDRALIZE(node.primitives)
6:     return
7:   if node is not a leaf node then
8:     CONSTRUCTTETRAHEDRALIZATIONS(node.left_child)
9:     CONSTRUCTTETRAHEDRALIZATIONS(node.right_child)

```

we determine whether a node is advantageous to be tetrahedralized or left as a BVH node (cf. Chapter 4). Given a node, if the approximate average nearest-hit cost of traversing tetrahedralization is less than the SAH cost, we select the node as a tetrahedralization node. If a node is selected as a tetrahedralization node, we do not check its children. After the selection process, we convert selected BVH nodes into tetrahedralization leaves by pruning their children and constructing tetrahedralizations for the primitives in selected nodes. We use the axis-aligned bounding box of BVH nodes to bound each tetrahedralization. We use Tetgen [23] software to tetrahedralize the primitive groups and use compact tetrahedron representation proposed in [31] to store the tetrahedralizations.

3.2 Traversal

The nearest-hit traversal for the BTH structure (cf. Algorithm 6) is similar to the nearest-hit BVH traversal. Starting from the root, we descent the hierarchy until we reach a leaf node. If the bounding box a node does not intersect with the ray, we do not check its children for an intersection. When we encounter a leaf node, we perform different intersection tests based on the type of the leaf node. If a leaf node does not contain a tetrahedralization, we test for a ray-primitive intersection for all primitives in the node. Otherwise, we traverse the tetrahedralization using the traversal method proposed in [31]. Starting from the initial tetrahedron, we process the tetrahedralization structure, one tetrahedron at a time, until the ray

hits a primitive face, or it exits the tetrahedralization.

Algorithm 6 Recursive BTH Nearest-hit traversal

```

1: procedure BTHINTERSECT(node, ray)
2:   if node is a leaf then
3:     if node is a tetrahedralization then
4:        $\tau_0 \leftarrow$  Find first tetrahedron
5:       TETRAHEDRALIZATIONINTERSECT(ray, node,  $\tau_0$ )
6:     else
7:       for all primitive in node do
8:         INTERSECT(primitive, ray)
9:     else
10:      BTHINTERSECT(node.left)
11:      BTHINTERSECT(node.right)

```

Nearest-hit traversal on tetrahedralization requires an initial tetrahedron. If the ray's origin is inside the boundary of the tetrahedralized area, we use the point-location method explained in [3] to find the tetrahedron that contains the ray's origin as the initial tetrahedron. If the ray originates from outside the tetrahedralization, we use the faces of the bounding box of the tetrahedralization to find the boundary tetrahedron that the ray first hits. We first locate the point that the ray hits on the boundary (see Figure 3.2). Then we find the face that contains the intersection point. After we find the boundary face, we find the tetrahedron that the face belongs to by using a lookup table.

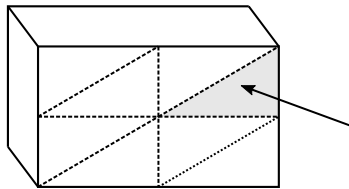


Figure 3.2: Finding the initial tetrahedron in a bounded tetrahedralization. The initial tetrahedron can be found using the bounding face that the ray hits (shaded region).

Chapter 4

Tetrahedralization Nearest-Hit Cost Heuristic

BTH construction algorithm builds tetrahedralizations for nodes that are advantageous to be tetrahedralized. Given a BVH node, the algorithm constructs tetrahedralization for the geometry if the average nearest-hit cost of tetrahedralization of the node’s primitives is less than the SAH cost of the corresponding BVH branch ($Cost_{TET}(node) < Cost_{SAH}(node)$). We calculate the approximate cost assuming

- rays are distributed uniformly throughout the space and
- rays originated from the outside of the geometry.

We propose different approximations for the average nearest-hit cost of a tetrahedralization structure. Since many nodes are considered in BTH construction, a fast cost approximation is essential for quick BTH construction. The approximate nearest-hit cost on tetrahedralization is directly dependent on the number of tetrahedra traversed. We define the approximate cost of traversing a tetrahedralization as

$$Cost_{TET}(node) = C_{tet} \times N_{avg}(\mathcal{T}_{node}), \quad (4.1)$$

where C_{tet} is the cost of traversing a single tetrahedron in tetrahedralization, and $N_{avg}(\mathcal{T}_{node})$ is the average number of tetrahedra that are traversed during nearest-hit traversal for tetrahedralization of the node. To calculate the cost, we first calculate approximation of $N_{avg}(\mathcal{T}_{node})$. Then using Equation 4.1 we calculate the $Cost_{TET}(node)$. We propose three different ways to approximate $N_{avg}(\mathcal{T}_{node})$; sampling-based cost calculation, average depth-based cost calculation, face count based cost calculation. Among the approximation methods, only the sampling-based calculation requires a tetrahedralization to be present. Others, approximate the cost without requiring a tetrahedralization to be present. For the evaluation of the approximation methods, we used tetrahedralizations of models from the Thingi10k [34] data set.

4.1 Sampling-based Cost Calculation

We calculate the cost using the following Monte Carlo approach; given a tetrahedralization, we randomly sample rays that originate at the boundary of the tetrahedralization. Then we traverse the tetrahedralization along the ray, counting the number of tetrahedra traversed. We calculate the average number of tetrahedra traversed from sampled rays as

$$N_{sampling}(\mathcal{T}_{node}) \approx \frac{1}{n} \sum_0^n N_{ray_i}(\mathcal{T}_{node}) \quad (4.2)$$

where n is the number of sampled rays and $N_{ray_i}(\mathcal{T}_{node})$ is the tetrahedra count for the randomly sampled ray ray_i . With this approach, the accuracy of the approximated average tetrahedra count gets better as the number of sampled rays increases. Although this approach can approximate the tetrahedra count well, it requires an existing tetrahedralization. Constructing a tetrahedralization for each possible node in the BVH hierarchy is slow and not feasible. We only used this method to evaluate other approximation methods that can approximate the tetrahedra count without requiring a tetrahedralization to be constructed.

4.2 Average Depth-based Cost Calculation

The average depth-based calculation estimates the average number of tetrahedra traversed during nearest-hit traversal by estimating the average depth of the rays that are sampled on the boundary of the tetrahedralization. The relation between the average ray depth and tetrahedra count is formalized using Theorem 1:

Theorem 1. *Let s be a line segment starts and ends within the given tetrahedralization \mathcal{T} . s must stab at least $\frac{\|s\|}{l_{max}^*}$ tetrahedra of \mathcal{T} , where l_{max}^* is the length of longest edge in \mathcal{T} .*

$$N_s(\mathcal{T}) \geq \frac{\|s\|}{l_{max}^*}. \quad (4.3)$$

Proof. Let τ be a tetrahedron that the line segment intersects. It can be shown that the length of the intersection cannot be greater than l_{max}^τ , where l_{max}^τ is the length of the longest edge of τ . Therefore,

$$\|s \cap \tau\| \leq l_{max}^\tau. \quad (4.4)$$

By using Equation 4.4 for each tetrahedron that intersects s , we can sum all the lengths, as in Equation 4.5:

$$\|s\| = \sum_{\tau \in \mathcal{T}} \|s \cap \tau\| \leq \sum_{\tau \in \mathcal{T}} l_{max}^\tau. \quad (4.5)$$

Since $l_{max}^\tau < l_{max}^*$ for all $\tau \in \mathcal{T}$ from Equation 4.5, we get Equation 4.6.

$$\|s\| \leq N_s(\mathcal{T}) l_{max}^*. \quad (4.6)$$

□

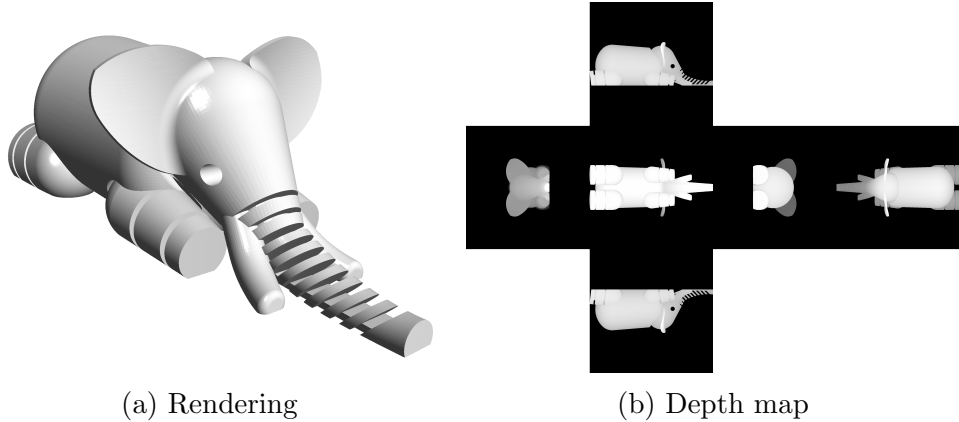


Figure 4.1: The average depth estimation using depth map. The elephant model is the courtesy of Qingnan Zhou and Alec Jacobson [34].

By using Theorem 1, we can derive an approximation for the average number of traversed tetrahedra as follows:

$$N_{depth}(\mathcal{T}_{node}) \approx \frac{d_{avg}}{l_{avg}^*}, \quad (4.7)$$

where d_{avg} is the average depth of rays in the tetrahedralization and l_{avg}^* is the average length of the edges of the constrained faces. To estimate the average depth of rays, d_{avg} , we used the z-buffer algorithm to calculate a depth map (see Figure 4.1) for each side of a bounding box of the constrained faces. The calculation of the depth map with resolution R requires $\mathcal{O}(NR^2)$ time complexity in the worst case. Using the calculated depth map, we estimated the average depth. Although the depth map only contains the depth of equally spaced axis-aligned rays, it is fast and it can estimate average depth well for most of the models.

Figures 4.2 and 4.3 compare the average depth-based and sampling-based methods. Figure 4.3 shows that the average depth-based cost calculation method underestimates the cost. To resolve this, we modified Equation 4.7 as

$$N_{depth}(\mathcal{T}_{node}) \approx \frac{d_{avg}}{l_{avg}^*} + C, \quad (4.8)$$

where C is a constant value.

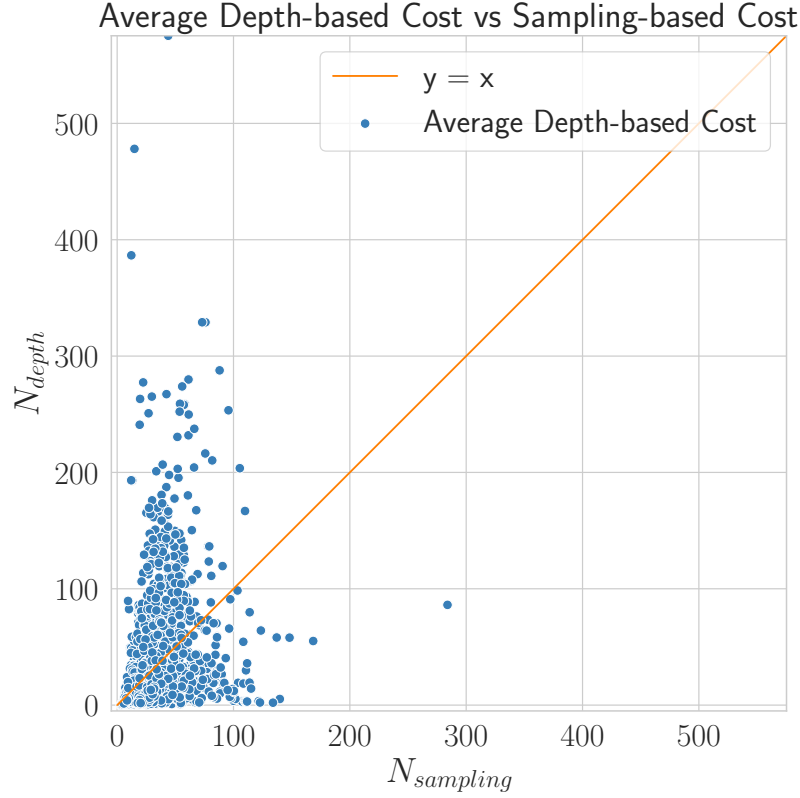


Figure 4.2: The comparison of the estimated tetrahedron count using the average depth-based method and the sampling-based method.

4.3 Primitive Count-based Cost Calculation

In our experiments, we found that the average traversed tetrahedra count is related to the cube root of primitive count. This result is similar to the runtime complexity of regular grids ($\Theta(\sqrt[3]{N})$) shown by Clearly and Wyvill [5]. Therefore, we define primitive count-based cost as

$$N_{count}(\mathcal{T}_{node}) \approx \sqrt[3]{|F|}. \quad (4.9)$$

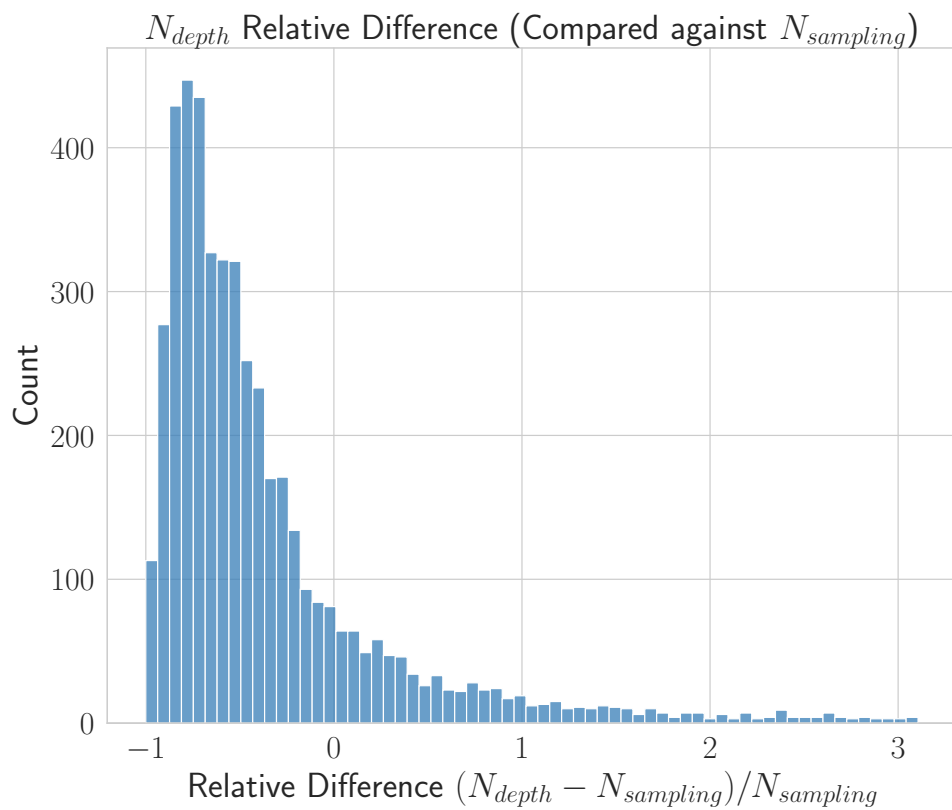


Figure 4.3: The histogram of the relative difference of the tetrahedron count between the average depth-based and sampling-based estimation methods.

where $|F|$ is the number of primitives. Figures 4.4 and 4.5 show the comparison of primitive count-based method and sampling-based method. It can be seen that the N_{count} can approximate the average tetrahedron count well.

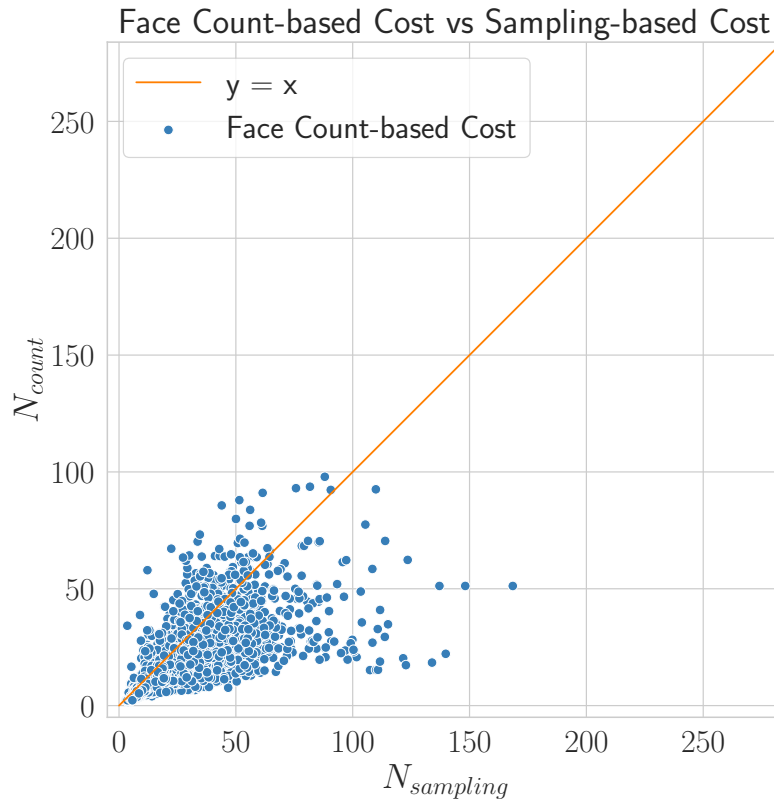


Figure 4.4: The comparison of the estimated tetrahedron count using the average depth-based method and the sampling-based method.

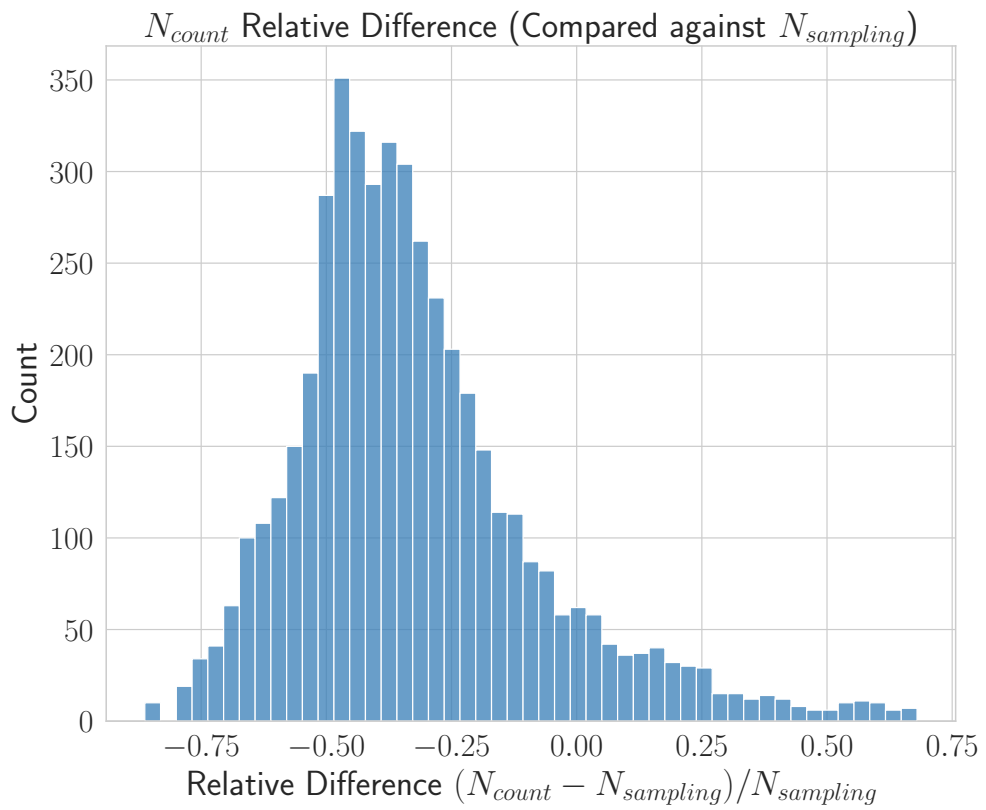


Figure 4.5: The histogram of the relative difference of the tetrahedron count between the primitive count-based and the sampling-based estimation methods.

Chapter 5

Animated Scenes Using the BTH Structure

In general, ray tracing acceleration structures are designed for static scenes. For dynamic scenes, acceleration structures need to be adapted. In early research, the effectiveness of acceleration structures only measured using rendering times; the construction costs are usually ignored. As ray tracing achieves interactive frame rates, the construction time is also an issue to be considered as a measure of effectiveness. In general, there is a tradeoff between the construction quality and construction times. For dynamic scenes, both are important because the acceleration structure needs to be updated at each frame.

In terms of dynamic, or animated scenes, there are two approaches to update the acceleration structure when the geometry changes. The first approach is to rebuild the entire acceleration structure. Although rebuilding provides a simple way to adapt to the changing geometry, it can be an expensive operation for some structures. The second approach is to update the existing acceleration structure by only updating parts of the acceleration structure that changed. Although this is more efficient than rebuilding the entire structure, updates reduce the quality of the acceleration structure, thereby reducing the nearest-hit efficiency.

We can categorize dynamic scenes according to the types of motion that the objects perform. In scenes with hierarchical motion, groups of primitives move in the same way. In incoherent motion, the primitives move independently from each other. Depending on the characterization of motion in the scene different algorithms can be better suited for adapting acceleration structures.

Because BTH structure includes a BVH, most of the dynamic BVH algorithms also work for BTH. The only problem that can arise when adapting a BVH update algorithm into a BTH algorithm is to tetrahedralization leaves. It is shown that a small number of deformations can be applied to tetrahedralizations without any update [3].

We will describe how the BTH structure can be adapted to dynamic scenes that perform hierarchical motions in the following section.

5.1 Two-Level BTH for Hierarchical Motion

If the primitives in the scene exhibit hierarchical motion, we can use two-level (multi-level) hierarchies [35] as an acceleration structure. For such a two-level hierarchy, we group the primitives in the scene into separate objects and build an acceleration structure for each group. Constructed acceleration structures are the bottom-level of the two-level acceleration structure. When building bottom-level acceleration structures, we use the local reference frames of the objects. Then, we construct a top-level acceleration structure for the bottom-level acceleration structures. In two-level hierarchies, nearest-hit traversal starts from the node at the top-level. When we reach a bottom-level node, we transform the ray into the object's reference frame and test it for intersection using the bottom-level structure. When an object in the scene exhibits a rigid body motion, only the top-level needs to be rebuilt or updated since primitives in the bottom-level acceleration structure does not change. A side effect of two-level is that two-level acceleration structures also support instancing (see Figure 5.1). When the scene contains duplicate objects, we can use the same bottom-level structure to

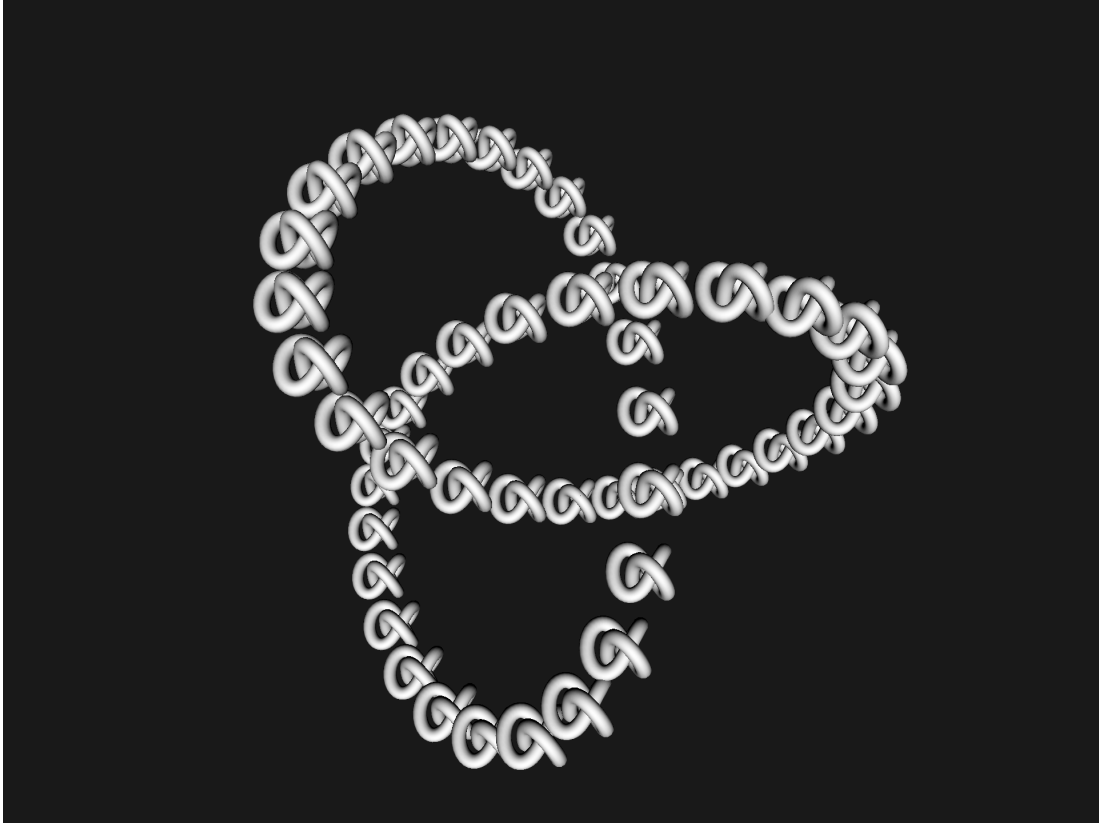


Figure 5.1: An instancing example: Torus knot made of smaller torus knots. Single bottom-level acceleration structure built for one torus knot is enough to render all of the torus knots.

represent the duplicated objects.

We use BVH as the top-level and BTH as the bottom-level acceleration structure for animated scenes (see Figure 5.2). For each object in the 3D scene, we construct a BTH structure. Then we combine these bottom-level structures into the top-level using a BVH. For the construction of BVH, we used the midpoint to partition the bottom-level nodes. In each animation frame, we re-build the top-level BVH using updated coordinates of the objects. The midpoint partitioning scheme allows us to re-build the top-level efficiently for each animation frame.

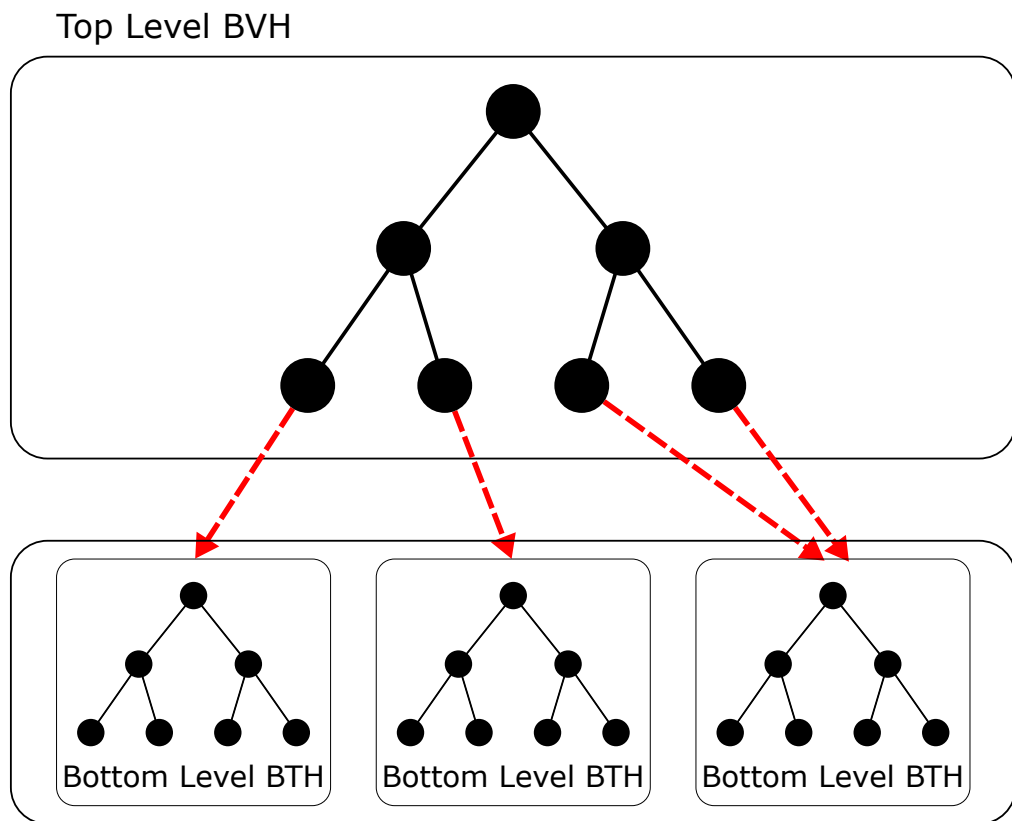


Figure 5.2: Two-Level BVH-BTH structure.

Chapter 6

Experimental Results

6.1 Experimental Setup

Experiments are conducted on a computer with six cores @3.2 GHz (Intel i7-9750H), 16 GB of main memory. We construct the acceleration structures on a single thread. We render images at 1920×1080 resolution. For ray tracing, we used a multi-threaded 16×16 tile-based rendering method. For consistency, we use a single ray per pixel and each ray passes through the center of the pixels in the image plane. We render images using only primary rays without secondary rays or shadow rays.

Our BTH implementations is based on the BVH implementation of Pbrt [36]. We used TetGen [23] tool to tetrahedralize the bounded tetrahedralizations. We used Aman et al.'s [31] *Tet32* and *Tet20* tetrahedral representation schemes and named our acceleration structures accordingly, i.e., *BTH32* and *BTH20*. Although Aman et al. [31] also propose the *Tet16* structure, which has a smaller memory requirement than *Tet32* and *Tet20* structures, we did not use it in our experiments because finding the source tetrahedron that the ray starts in tetrahedralizations stored by *Tet16* scheme requires a more complex lookup table. For comparison, we used BVH and *k*-d tree implementations of Pbrt.

In our experiments, we used models from Morgan McGuire’s Computer Graphics Archive [37]. Additionally, the Armadillo model is from Krishnamurthy and Levoy [38] and the Crown model is from Martin Lubich [39].

6.2 Selection of Tetrahedralization Cost Method

The average depth-based and the primitive count-based tetrahedralization cost methods allow us to automatically select tetrahedralizations based on their estimated cost. Both cost estimation methods rely on C_{tet} , the cost of traversing a single tetrahedron in tetrahedralization. Different values of C_{tet} leads to different BTH structures (see Figure 6.1). In general, as C_{tet} increased, the BTH construction algorithm selects and constructs fewer tetrahedralizations.

Although we defined C_{tet} as the cost of traversing a single tetrahedron, we used C_{tet} as a parameter for the construction of BTH. We performed a grid search on various models to select the best value for C_{tet} . We used the construction and rendering times in Tables 6.1 and 6.2 to select the best C_{tet} . Because BTH32 and BTH20 have different tetrahedra traversal times, we decided to select C_{tet} values for them separately. For BTH32, we decided to use 0.7 for C_{tet} . For BTH20, we decided that 0.5 is a good choice for C_{tet} . Additionally, Tables 6.1 and 6.2 show the effect of changing C_{tet} . As C_{tet} increases, more primitives at the leaves of the BVH structure are tetrahedralized; hence, the construction of the structure takes more time.

We also utilize Tables 6.1 and 6.2 to select the cost method. These tables show that tetrahedralizations built by both average-depth based cost method and primitive count-based cost method can speed-up the rendering times of the BTH structure. Although the calculation of the average depth-based cost method takes more computation time than the primitive count-based cost method, the rendering times of the average depth-based cost method are usually faster than the primitive count-based cost method. In the following experiments, we only use the average depth-based method to construct BTH structures.

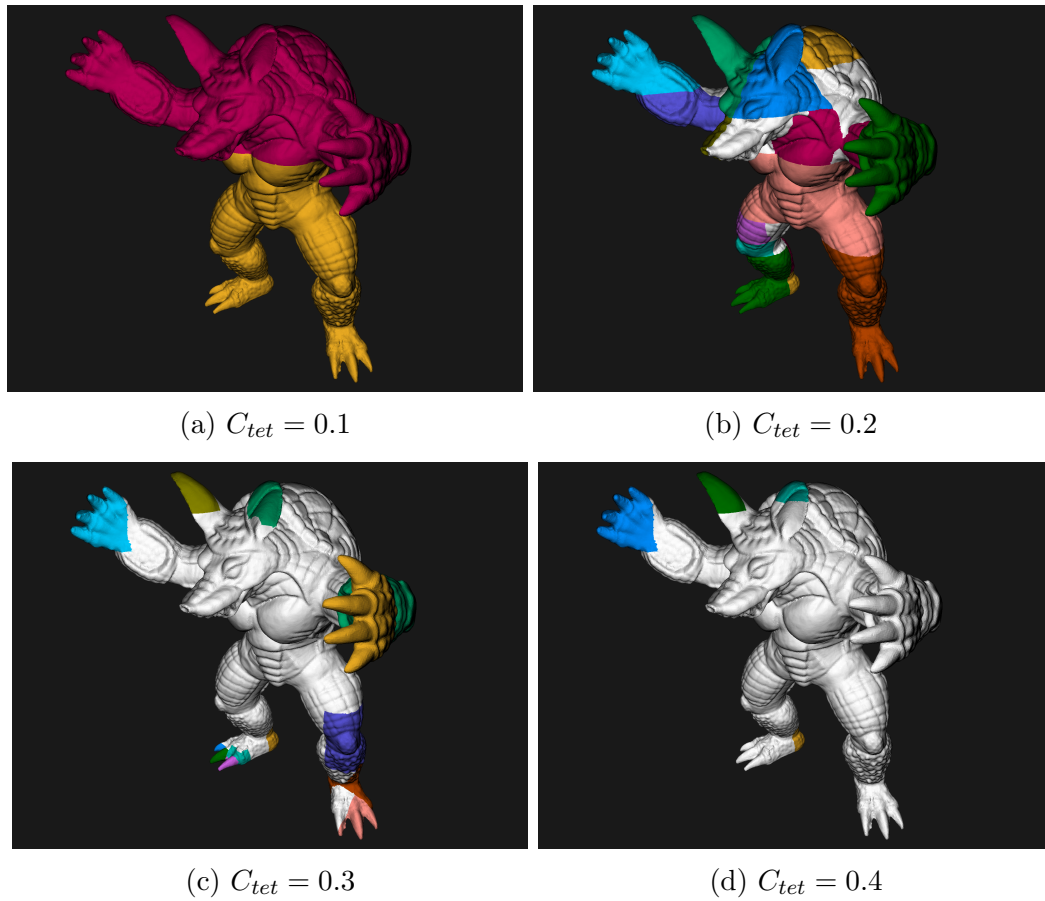


Figure 6.1: Tetrahedralizations of the BTH structure constructed using different C_{tet} values. Different tetrahedralizations are colored differently. Gray color is used for faces in the BVH structure.

Table 6.1: BTH32 statistics for varying C_{tet} . “No. tet. primitives” is the number of primitives at the leaf nodes of the BVH that are tetrahedralized. “No. BVH primitives” is the number of primitives at the leaf nodes of the BVH that are not tetrahedralized. Times are in milliseconds (ms).

Scenes	C_{tet}	Primitive count-based cost				
		0.1	0.3	0.5	0.7	1.0
Armadillo	Rendering time	134.34	136.56	86.14	80.43	82.50
	Construction time	18,592.1	16,297.3	1,298.2	481.7	479.2
	No. tet. primitives	345,938	330,809	30,980	0	0
	No. BVH primitives	0	15,129	314,958	345,938	345,938
Lumberyard	Rendering time	466.75	492.11	476.28	488.51	477.62
	Construction time	6,154.9	4,799.5	3,179.0	2,551.9	1,971.1
	No. tet. primitives	119,242	81,051	55,083	32,720	20,121
	No. BVH primitives	901,665	939,856	965,824	988,187	1,000,786
Sponza	Rendering time	412.98	410.93	408.99	413.85	403.93
	Construction time	364.0	361.3	331.2	310.0	307.6
	No. tet. primitives	4,764	4,764	2,035	0	0
	No. BVH primitives	257,503	257,503	260,232	262,267	262,267

Scenes	C_{tet}	Average depth-based cost				
		0.1	0.3	0.5	0.7	1.0
Armadillo	Rendering time	136.52	101.08	80.90	80.38	80.99
	Construction time	16,777.0	4,367.7	2,151.3	2,023.5	2,043.6
	No. tet. primitives	345,938	88,923	1,024	0	0
	No. BVH primitives	0	257,015	344,914	345,938	345,938
Lumberyard	Rendering time	481.34	484.81	481.65	474.76	479.86
	Construction time	6,321.8	4,516.2	3,419.7	2,686.3	2,348.9
	No. tet. primitives	117,371	80,258	56,919	34,636	24,717
	No. BVH primitives	903,536	940,649	963,988	986,271	996,190
Sponza	Rendering time	409.98	408.12	405.41	408.20	408.08
	Construction time	365.1	369.1	343.5	337.7	315.7
	No. tet. primitives	4,764	4,764	2,035	2,035	0
	No. BVH primitives	257,503	257,503	260,232	260,232	262,267

Table 6.2: BTH20 statistics for varying C_{tet} . “No. tet. primitives” is the number of primitives at the leaf nodes of the BVH that are tetrahedralized. “No. BVH primitives” is the number of primitives at the leaf nodes of the BVH that are not tetrahedralized. Times are in milliseconds (ms).

Scenes	C_{tet}	Primitive count-based cost				
		0.1	0.3	0.5	0.7	1.0
Armadillo	Rendering time	111.95	110.96	75.23	80.52	79.04
	Construction time	18,882.9	15,812.2	1,302.3	466.3	490.8
	No. tet. primitives	345,938	330,809	30,980	0	0
	No. BVH primitives	0	15,129	314,958	345,938	345,938
Lumberyard	Rendering time	449.00	447.07	450.31	450.71	449.52
	Construction time	6,203.8	4,371.2	3,138.6	2,397.3	1,980.8
	No. tet. primitives	119,242	81,051	55,083	32,720	20,121
	No. BVH primitives	901,665	939,856	965,824	988,187	1,000,786
Sponza	Rendering time	392.33	391.72	388.26	381.45	379.68
	Construction time	363.6	377.1	333.1	306.8	303.5
	No. tet. primitives	4,764	4,764	2,035	0	0
	No. BVH primitives	257,503	257,503	260,232	262,267	262,267

Scenes	C_{tet}	Average depth-based cost				
		0.1	0.3	0.5	0.7	1.0
Armadillo	Rendering time	112.60	84.38	77.47	77.93	76.99
	Construction time	16,750.7	4,199.9	2,030.7	2,027.9	1,984.50
	No. tet. primitives	345,938	88,923	1,024	0	0
	No. BVH primitives	0	257,015	344,914	345,938	345,938
Lumberyard	Rendering time	451.15	449.46	451.51	451.20	448.55
	Construction time	6,374.3	4,405.2	3,409.2	2,698.6	2,329.0
	No. tet. primitives	117,371	80,258	56,919	34,636	24,717
	No. BVH primitives	903,536	940,649	963,988	986,271	996,190
Sponza	Rendering time	387.37	376.36	378.60	376.36	379.17
	Construction time	375.7	365.3	331.8	336.1	310.4
	No. tet. primitives	4,764	4,764	2,035	2,035	0
	No. BVH primitives	257,503	257,503	260,232	260,232	262,267

6.3 Results

6.3.1 Static Scenes

We used a variety of scenes to compare BTH structure to other acceleration structures. Table 6.3 compare the construction and rendering times our hybrid acceleration structures (BTH32 and BTH20) against other structures (BVHs and k -d trees) for a set of scenes. The scenes in this table cannot be tetrahedralized directly. Hence, we did not compare tetrahedralization-based acceleration structures in this table.

Table 6.4 also compare the construction and rendering times our hybrid acceleration structures (BTH32 and BTH20) against other acceleration structures, including tetrahedralization-based acceleration structures, BVHs and k -d trees, for a set of scenes. The scenes in this table can be directly tetrahedralized without any preprocessing.

In most cases, the BTH structure performs better than the BVH structure. This is due to the tetrahedralization traversal cost heuristic. As long as the tetrahedralization traversal cost approximation methods are accurate, the BTH construction algorithm selects the tetrahedralizations that improve the rendering (ray tracing) cost.

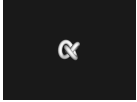



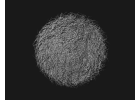
For some cases, k -d tree performs better than the BTH structure in terms of the rendering cost. In general, k -d tree performs worse than BTH when the scene contains a large amount of intersecting geometry. Many intersecting geometries cause many duplicate primitives in k -d tree structure. Therefore, self-intersections affect k -d tree negatively, for both construction and rendering times. On the other hand, self-intersections have a small effect on the BTH structure. If a scene contains a high number of self-intersections, these self-intersecting geometry is stored in BVH leaves in the BTH structure.

In terms of the construction times, compared against BVH, the BTH construction algorithm is always significantly slower than BVH. This can be attributed to the tetrahedralization algorithm. Firstly, TetGen uses algorithms with $\mathcal{O}(N^2)$ worst-case time complexity. Secondly, robust tetrahedralization with exact arithmetic slows down the tetrahedralization process. In some cases, the BTH structure outperforms the k -d tree. When a scene contains a lot of intersecting geometry, the construction of the k -d tree takes a significant amount of computation time. This can be seen in the Hairball model. Since the intersecting geometry is stored in BVH nodes of the BTH structure, BTH performs well.

Compared to tetrahedralizations, rendering using the BTH structure is faster. Besides, the BTH requires less time to construct in all cases than tetrahedralizations. Additionally, another advantage of BTH over tetrahedralizations is that we can build the BTH structure for any three-dimensional geometric models. On the other hand, we can construct tetrahedralizations for non-self-intersecting geometries, or we must perform a self-intersection removal step before tetrahedralization.

Experimental results show that the BTH20 has a faster rendering speed than the BTH32. It is because the Tet20 used in BTH20 requires less memory than the Tet32 used in BTH32. The smaller memory requirement of the tetrahedron representation of the BTH20 leads to higher cache utilization than the BTH32. Both the BTH32 and BTH20 structures require approximately similar construct times.

Table 6.3: The comparison of the construction and rendering times for different acceleration structures on scenes that cannot be directly tetrahedralized. We compare our proposed BTH32 and BTH20 with BVH [36] and k -d tree [36]. Times are in milliseconds (ms).

	Scenes				
					
	Torus Knot	Armadillo	Mix	Bmw	Hairball
No. faces	2,880	345,938	2,505,992	385,162	2,880,000
	Rendering times				
BTH32	14.94	80.09	145.74	92.18	274.73
BTH20	14.66	76.88	140.46	87.73	266.58
BVH	17.50	79.75	145.59	94.36	271.20
k -d tree	15.13	95.09	166.78	88.345	313.78
	Construction times				
BTH32	103.20	496.21	3,558.79	4,293.72	14,527.30
BTH20	101.79	476.33	3,485.51	4,197.93	14,541.70
BVH	2.43	341.56	2,640.36	359.97	2,976.25
k -d tree	24.26	1,459.58	14,127.60	2,489.23	62,846.80





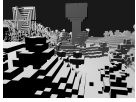



	Scenes				
					
	Lumberyard	Crown	Sponza	San Miguel	Vokselia
No. faces	1,020,907	3,540,310	262,267	9,980,699	1,875,632
	Rendering times				
BTH32	444.60	241.61	378.73	550.58	186.13
BTH20	423.91	231.10	349.98	516.49	173.60
BVH	459.61	242.66	384.20	586.89	190.15
k -d tree	419.74	247.21	213.80	315.46	194.70
	Construction times				
BTH32	2,873.49	24,365.30	1,944.00	18,569.90	4,690.44
BTH20	2,786.16	24,311.90	1,951.73	18,536.50	4,643.17
BVH	1,025.23	3,808.49	245.38	11,185.10	1,546.62
k -d tree	9,315.58	26,292.00	2,028.37	77,815.50	4,521.61




Table 6.4: The comparison of the construction and rendering times for different acceleration structures on scenes that can be directly tetrahedralized. We compare our proposed BTH32 and BTH20 with the tetrahedralization-based acceleration structures [31], BVH [36] and k -d tree [36]. Times are in milliseconds (ms).

	Scenes		
			
	Armadillo	Mix	Mix Close
No. faces	345,938	2,505,992	2,505,992
	Rendering times		
BTH32	79.54	142.37	232.23
BTH20	77.95	140.70	218.13
Tet32	150.84	312.55	363.67
Tet20	118.28	252.76	273.62
BVH	78.71	143.45	224.06
k -d tree	98.18	164.56	254.79
	Construction times		
BTH32	2,398.8	4,473.5	4,593.5
BTH20	3,367.7	4,485.8	4,468.8
Tet32	8,392.1	89,949.7	91,551.4
Tet20	8,380.4	90,291.4	90,564.1
BVH	373.1	2,683.6	2,683.3
k -d tree	1,460.6	14,830.4	14,090.6

6.3.2 Dynamic Scenes

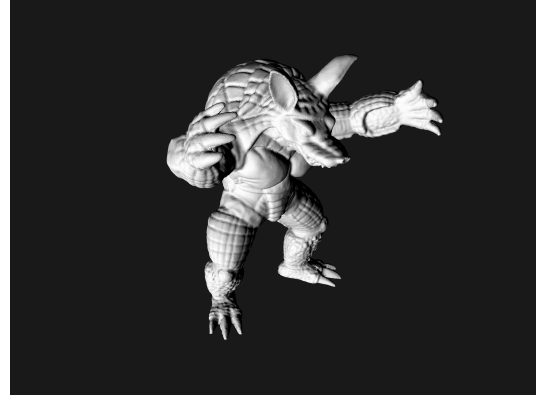
Figures 6.2, 6.3, and 6.4 show consecutive frames of the animations of the rotating armadillo, the random motion, and the falling hairballs, respectively, rendered using the proposed two-level BVH-BTH structure. We compared the two-level BVH structure against our two-level BVH-BTH structure. Table 6.5 shows the results. In general, the proposed two-level BVH-BTH structure using BTH20 at the lower level is faster than the two-level BVH structure.

Table 6.5: The comparison of the construction and rendering times of different acceleration structures for the animated scenes. We compare the proposed BTH32 and BTH20 with BVH [36]. Times are in milliseconds (ms).

	Scenes		
			
	Rotating Armadillo	Random Motion	Falling Hairballs
No. faces	345,938	1,020,907	12,860,699
No. objects	1	101	101
	Average rendering times (per frame)		
BTH32	70.88	675.51	721.06
BTH20	70.14	659.55	699.02
BVH	70.84	671.56	737.32
	Initial Construction times		
BTH32	2,403.3	8,831.5	37,920.2
BTH20	2,466.7	9,075.9	38,030.4
BVH	363.14	1,609.1	14,409.5



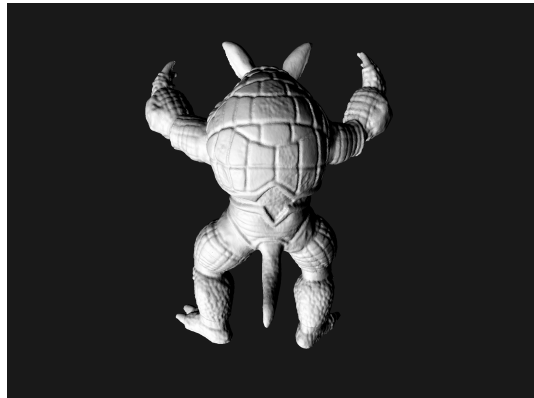
Frame 1



Frame 21



Frame 41



Frame 61



Frame 81



Frame 101

Figure 6.2: Rotating armadillo animated scene rendered using the two-level BVH-BTH structure.



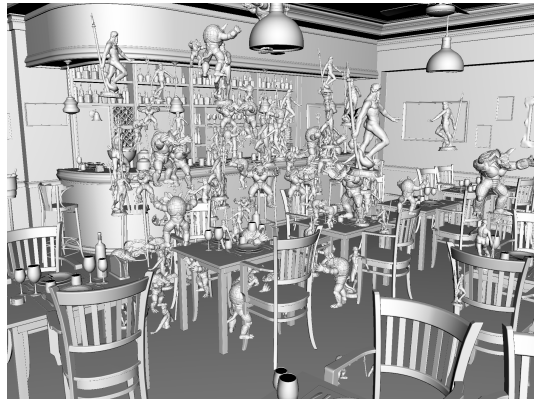
Frame 1



Frame 21



Frame 41



Frame 61



Frame 81



Frame 101

Figure 6.3: Random motion animated scene rendered using the two-level BVH-BTH structure.



Frame 1



Frame 21



Frame 40



Frame 61



Frame 81



Frame 101

Figure 6.4: Falling airballs animated scene rendered using the two-level BVH-BTH structure.

Chapter 7

Conclusions and Future Research Directions

We proposed the BVH-Tetrahedralization Hybrid structure for accelerating the nearest-hit (ray-surface intersection) tests for the ray tracing algorithm. We tested our acceleration structure with different scenes. Our experiments show that for some of the scenes, our BTH structure outperforms existing acceleration structures. We showed that we could improve the rendering time of the BVH structure by converting it into a BTH hybrid structure. Our experiments show that, in all cases, the proposed BTH20 acceleration structure outperforms the BVH structure in terms of rendering times at the cost of slower construction times. We proposed two methods for approximating average nearest-hit costs for tetrahedralization structures. We show that the proposed cost calculation methods that do not require the construction of tetrahedralizations can provide good approximations for the average nearest-hit cost on tetrahedralized scenes.

Additionally, we can use the two-level acceleration structure for dynamic scenes with hierarchical motions. Our experiments show that the two-level BVH-BTH outperforms the two-level BVH-BVH for the tested scenes where the objects perform hierarchical movements.

Some possible future work areas can be as follows:

- *Adapting the BTH structure for scenes with deforming geometries:* One advantage of using tetrahedralizations in the BTH structure is that we can modify tetrahedralizations up to a certain degree without the need for refitting. For animated scenes with deforming geometry, we could use BVH fitting methods to update the BTH structure. We could also use the BTH structure for rendering animated frames of articulated bodies with nonrigid limbs.
- *Exploiting ray connectivity for secondary rays:* In hierarchical acceleration structures, it is hard to exploit ray connectivity. The nearest-hit test starts from the root of the hierarchy for secondary rays. On the other hand, tracing secondary rays is easy on tetrahedralizations. After a primary ray hits a surface, secondary rays can continue from where the primary ray hits without the need for an initialization step, which is necessary for tree traversals. Similarly, we can easily trace secondary rays in tetrahedralizations on the leaves of a BTH hierarchy.

Bibliography

- [1] T. Whitted, “An Improved Illumination Model for Shaded Display,” *Communications of the ACM*, vol. 23, p. 343–349, June 1980.
- [2] J. T. Kajiya, “The Rendering Equation,” *ACM Computer Graphics (Proceedings of SIGGRAPH ’86)*, vol. 20, p. 143–150, Aug. 1986.
- [3] A. Lagae and P. Dutré, “Accelerating Ray Tracing using constrained Tetrahedralizations,” *Computer Graphics Forum*, vol. 27, no. 4, pp. 1303–1312, 2008.
- [4] A. Fujimoto, T. Tanaka, and K. Iwata, “ARTS: Accelerated Ray-tracing System,” in *Tutorial: Computer Graphics; Image Synthesis* (K. I. Joy, C. W. Grant, N. L. Max, and L. Hatfield, eds.), pp. 148–159, New York, NY, USA: Computer Science Press, Inc., 1988.
- [5] J. G. Cleary and G. Wyvill, “Analysis of an Algorithm for Fast Ray Tracing Using Uniform Space Subdivision,” *The Visual Computer*, vol. 4, no. 2, pp. 65–83, 1988.
- [6] A. Woo, “Ray Tracing Polygons Using Spatial Subdivision,” in *Proceedings of Graphics Interface*, vol. 92, (San Francisco, CA, USA), pp. 184–191, Morgan Kaufmann Publishers, Inc., 1992.
- [7] K. Klimaszewski and T. W. Sederberg, “Faster Ray Tracing Using Adaptive Grids,” *IEEE Computer Graphics and Applications*, vol. 17, no. 1, pp. 42–51, 1997.

- [8] T. Ize, P. Shirley, and S. Parker, “Grid Creation Strategies for Efficient Ray Tracing,” in *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, RT '07, (Washington, DC, USA), pp. 27–32, IEEE Computer Society, 2007.
- [9] J. L. Bentley, “Multidimensional Binary Search Trees Used for Associative Searching,” *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.
- [10] J. Goldsmith and J. Salmon, “Automatic Creation of Object Hierarchies for Ray Tracing,” *IEEE Computer Graphics and Applications*, vol. 7, no. 5, pp. 14–20, 1987.
- [11] D. J. MacDonald and K. S. Booth, “Heuristics for Ray Tracing Using Space Subdivision,” *The Visual Computer*, vol. 6, no. 3, pp. 153–166, 1990.
- [12] V. Havran, *Heuristic Ray Shooting Algorithms*. Ph.d. thesis, Department of Computer Science and Engineering, Faculty of Electrical Engineering, Czech Technical University in Prague, November 2000.
- [13] I. Wald and V. Havran, “On Building Fast kd-Trees for Ray Tracing, and on Doing That in $O(N \log N)$,” in *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, pp. 61–69, 2006.
- [14] J. H. Clark, “Hierarchical Geometric Models for Visible Surface Algorithms,” *Communications of the ACM*, vol. 19, no. 10, p. 547–554, 1976.
- [15] S. M. Rubin and T. Whitted, “A 3-dimensional Representation for Fast Rendering of Complex Scenes,” *ACM Computer Graphics (Proceedings of SIGGRAPH '80)*, vol. 14, no. 3, pp. 110–116, 1980.
- [16] T. L. Kay and J. T. Kajiya, “Ray Tracing Complex Scenes,” *SIGGRAPH Comput. Graph.*, vol. 20, p. 269–278, Aug. 1986.
- [17] G. Müller and D. W. Fellner, “Hybrid Scene Structuring with Application to Ray Tracing,” in *Proceedings of the International Conference on Visual Computing*, pp. 19–26, 1998.
- [18] I. Wald, S. Boulos, and P. Shirley, “Ray Tracing Deformable Scenes Using Dynamic Bounding Volume Hierarchies,” *ACM Transactions on Graphics*, vol. 26, no. 1, Article no. 6, 18 pages, 2007.

- [19] S. Popov, J. Gunther, H.-P. Seidel, and P. Slusallek, “Experiences with streaming construction of sah kd-trees,” in *2006 IEEE Symposium on Interactive Ray Tracing*, pp. 89–94, IEEE, 2006.
- [20] I. Wald, “On Fast Construction of SAH-Based Bounding Volume Hierarchies,” in *Proceedings of the IEEE Symposium on Interactive Ray Tracing*, RT ’07, pp. 33–40, IEEE, 2007.
- [21] B. N. Delaunay, “Sur la Sphère Vide,” *Izvestia Akademia Nauk SSSR, VII Seria, Otdelenie Matematicheskii i Estestvennyka Nauk*, vol. 7, pp. 793–800, 1934.
- [22] G. L. Miller, D. Talmor, S.-H. Teng, N. Walkington, and H. Wang, “Control Volume Meshes Using Sphere Packing: Generation, Refinement and Coarsening,” in *Proceedings of the 5th International Meshing Roundtable*, pp. 47–61, 1996.
- [23] H. Si, “TetGen, a Delaunay-Based Quality Tetrahedral Mesh Generator,” *ACM Transactions on Mathematical Software*, vol. 41, no. 2, Article no. 11, 36 pages, 2015.
- [24] P. L. George, F. Hecht, and É. Saltel, “Automatic Mesh Generator with Specified Boundary,” *Computer Methods in Applied Mechanics and Engineering*, vol. 92, no. 3, pp. 269–288, 1991.
- [25] J. Ruppert, “A Delaunay Refinement Algorithm for Quality 2-Dimensional Mesh Generation,” *Journal of Algorithms*, vol. 18, no. 3, pp. 548–585, 1995.
- [26] J. R. Shewchuk, “Tetrahedral Mesh Generation by Delaunay Refinement,” in *Proceedings of the Fourteenth Annual Symposium on Computational geometry*, SCG ’98, pp. 86–95, 1998.
- [27] J. R. Shewchuk, “Adaptive Precision Floating-Point Arithmetic and Fast Robust Geometric Predicates,” *Discrete and Computational Geometry*, vol. 18, pp. 305–363, 1996.

- [28] Y. Hu, Q. Zhou, X. Gao, A. Jacobson, D. Zorin, and D. Panozzo, “Tetrahe-
dral Meshing in the Wild,” *ACM Transactions on Graphics*, vol. 37, no. 4,
Article no. 60, 14 pages, 2018.
- [29] Y. Hu, T. Schneider, B. Wang, D. Zorin, and D. Panozzo, “Fast Tetrahe-
dral Meshing in the Wild,” *ACM Transactions on Graphics*, vol. 39, no. 4,
Article no. 117, 18 pages, July 2020.
- [30] M. Maria, S. Horna, and L. Aveneau, “Efficient Ray Traversal of Constrained
Delaunay Tetrahedralization,” in *Proceedings of the 12th International Joint
Conference on Computer Vision, Imaging and Computer Graphics Theory
and Applications*, vol. 1 of *VISIGRAPP '17*, pp. 236–243, 2017.
- [31] A. Aman, S. Demirci, and U. Gdkbay, “Compact Tetrahedralization-based
Acceleration Structure for Ray Tracing,” *Computers & Graphics*, Submitted.
- [32] I. Wald, “On Fast Construction of SAH-based Bounding Volume Hierar-
chies,” in *Proceedings of the IEEE Symposium on Interactive Ray Tracing*,
RT '07, (Washington, DC, USA), pp. 33–40, IEEE Computer Society, 2007.
- [33] C. Ericson, *Real-Time Collision Detection*. San Francisco, CA: CRC Press,
2004.
- [34] Q. Zhou and A. Jacobson, “Thing10K: A Dataset of 10,000 3D-Printing
Models,” *arXiv preprint arXiv:1605.04797*, 2016.
- [35] J. Lext and T. Akenine-Mller, “Towards Rapid Reconstruction for Ani-
mated Ray Tracing,” in *Proceedings of Eurographics 2001–Short Presenta-
tions*, pp. 311–318, 2001.
- [36] M. Pharr, W. Jakob, and G. Humphreys, *Physically Based Rendering: From
Theory to Implementation*. San Francisco, CA, USA: Morgan Kaufmann
Publishers, Inc., 3rd ed., 2016.
- [37] M. McGuire, “Computer Graphics Archive,” July 2017.
<https://casual-effects.com/data>.

- [38] V. Krishnamurthy and M. Levoy, “Fitting Smooth Surfaces to Dense Polygon Meshes,” in *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '96, pp. 313–324, 1996.
- [39] M. Lubich, “Blender Minutes,” 2012. <https://www.loramel.net/>.