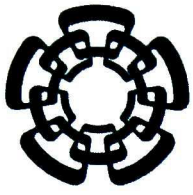
CT-831-551
DON. 2015



Centro de Investigación y de Estudios Avanzados del
Instituto Politécnico Nacional Unidad Guadalajara

Caracterización de mallas de superficie de escenas dinámicas para Ray Tracing en tiempo real

Tesis que presenta:

Ing. Ulises Olivares Pinto

Para Obtener el grado de:

Maestro en Ciencias

En la especialidad de:

Ingeniería Eléctrica

Director de Tesis.

Dr. Félix Francisco Ramos Corchado

CINVESTAV del IPN Unidad Guadalajara, Jalisco, Agosto del 2014

Caracterización de Escenas de Modelos Tridimensionales para Raytracing en Tiempo Real

Tesis de Maestría en

Ciencias de la Computación

Por:

Ulises Olivares Pinto

Ingeniero en Sistemas Computacionales

Instituto Tecnológico de Morelia

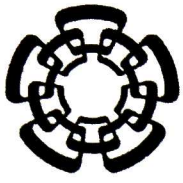
2007-2012

Becario de CONACYT, expediente No. 485947

Director de tesis.

Dr. Félix Francisco Ramos Corchado

Cinvestav del IPN Unidad Guadalajara. Agosto del 2014



Centro de Investigación y de Estudios Avanzados del
Instituto Politécnico Nacional Unidad Guadalajara

Surface Mesh Characterization of Dynamic Scenes for Ray Tracing in Real-time

A thesis presented by:

Eng. Ulises Olivares Pinto

To obtain degree of:

Master in Computer Sciences

In the subject of:

Electrical Engineering

Thesis Advisor:

Dr. Félix Francisco Ramos Corchado

Guadalajara, Jalisco. August, 2014

Surface Mesh Characterization of Dynamic Scenes for Ray Tracing in Real-time

Master in Science Thesis

In Electrical Engineering

By:

Ulises Olivares Pinto

Computer Systems Engineer

Instituto Tecnológico de Morelia

2007-2012

Scholarship granted by CONACYT, No. 485947

Thesis Advisor.

Dr. Félix Francisco Ramos Corchado

Cinvestav del IPN Unidad Guadalajara. August, 2014

To my mother, for teaching me that autonomy is conquered with consistent actions and not with empty words.

To my brother and sister for their support.

To my girlfriend for her love and support.

Acknowledgements

Foremost, I would like to express my sincere gratitude to my advisor Dr. Félix Francisco Ramos Corchado for making this research possible, for his continuous support, patience, motivation and knowledge whilst allowing me the room to work in my own way.

My sincere thanks also goes to Dr. Luis Ernesto López Mellado, Dr. Mario Angel Siller González Pico, Dr. Raul Ernesto González Torres and Dr. Andrés Mendez Vázquez, for offering me a new way to visualize problems in computer science, and for planting on me thirst of knowledge.

I would like to extend my appreciation to Dr. Arturo José García García for his support, time and advice throughout the research project.

This research project would not be possible without financial support of CONACYT. I would like to express my gratitude to this governmental institution.

Last but not least, to my friends and colleges at CINVESTAV to share knowledge, projects, foot-ball games, stressful moments, funny moments and the most important, to share with me their friendship.

Resumen

Ray tracing es una técnica de renderizado que produce imágenes realistas y de alta calidad como producto de un proceso de renderizado. Sin embargo, el costo asociado a la generación de estas imágenes es computacionalmente alto. Debido a esto, se han desarrollado diversas técnicas de optimización. Estas técnicas enfatizan el diseño eficiente de estructuras de datos que contienen información geométrica de una escena; se debe navegar a través de estas estructuras cuando se realizan pruebas de intersección. Sin embargo a pesar de estas optimizaciones, existe un balance entre el tiempo de construcción y el rendimiento de una estructura durante las pruebas de intersección. Este balance depende de la geometría de la escena de entrada, y esto no se ha estudiado a profundidad.

Esta investigación propone un método eficiente para extraer características de una escena con la finalidad de hacer una selección asertiva de una estructura de aceleración para Ray tracing. Este método decide de forma automática el mejor tipo de estructura para una escena específica, dada la información geométrica de una escena. Esta tesis describe un conjunto de características que impactan en el rendimiento de diversas estructuras de aceleración. Además, en esta tesis se analizan algunas métricas como: tiempo de construcción, frame-rate y número de intersecciones por caja y por primitiva con la finalidad de verificar si la estructura seleccionada es la apropiada para una geometría dada. Como el rendimiento de las estructuras varia drásticamente con diversas escenas, este método podrá reducir en gran medida el tiempo de computo seleccionando una estructura que maximice el rendimiento de Ray tracing.

Abstract

Ray tracing is a rendering technique that produces high quality and realistic images as a final product to a render process. Nevertheless, the computational cost required to generating these images is very high. Due this fact, several optimization techniques have been developed for Ray tracing. These optimizations focus on design efficient data-structures that can store geometry of the scene, and then these structures are traversed during intersection tests. Nonetheless, despite optimizations, there exist a trade-off between build time and ray traversal performance. This trade-off depends on the geometry of the input scene, and it has not been studied deeply.

This research proposes an efficient method to extract the characteristics of a scene to make an assertive choice of an acceleration structure for ray tracing in real-time. This method automatically decides the best structure type for a specific scene, given the geometric information. This thesis describes which geometric characteristics impact directly on performance for different acceleration structures. Also, in this thesis are analyzed some metrics as: build times, frame-rate and number of intersections per box and per primitive in order to verify if the selected structure is the most appropriate for a particular geometry. As the performance of acceleration structures varies with different scenes, this method could save much calculation resource by always selecting a structure that maximizes ray tracing performance.

Contents

List of Figures

List of Tables

Abbreviations

CPU	**C**entral **P**rocessing **U**nit
GPU	**G**raphics **P**rocessing **U**nit
GPGPU	**G**eneral **P**ropose computing on **G**raphics **P**rocessing **U**nit
SAH	**S**urface **A**rea **H**euristic
BVH	**B**ounding **V**olume **H**ierarchy
BIH	**B**ounding **I**nterval **H**ierarchy
LBVH	**L**inear **B**ounding **V**olume **H**ierarchy
HLBVH	**H**ierarchical **L**inear **B**ounding **V**olume **H**ierarchy
FPS	**F**rames **P**er **S**econd

Chapter 1

Introduction

1.1 Background

Ray tracing is a rendering technique that can generate realistic and high quality images. Ray tracing provides vivid illusion of reality; this was possible through introduction of shading [1], this concept improves the sense of solidity and depth. However, the computational cost required to render a model using ray tracing is very high. Due expensive computational cost ray tracing had been considered an offline rendering technique. Simulation of effects such as refraction and reflection [2] [3], are the hallmark of ray tracing since its introduction.

Ray tracing has long been a method of choice for off-line rendering due to its high computational cost. Moreover, hardware and computational improvements allow that real-time ray tracing finally within reach. Nevertheless, with real-time ray tracing arise new problems that did not exist with off-line ray tracing. Principally, real-time ray tracing offers the possibility to interactively ray trace moving animated scene content. This fact presents a challenge for current data structures.

Acceleration structures must be rebuilt or updated as the scene changes. Selecting the right kind of the acceleration structure that maximizes performance of ray

tracing, according to the characteristics of a scene is an important issue that has not been widely studied.

There have been several optimization techniques to decrease rendering times of three-dimensional scenes using ray tracing. One of the most appropriate strategies is the use of acceleration structures. Diverse acceleration structures have been proposed; These structures are grouped fundamentally in grids, kd-trees and bounding volume hierarchies (BVH) [4]. These structures differ in the way that they store primitives and on the split method that they employ; but all of these structures are introduced to reduce the number of intersection tests that a ray tracer must perform when visible surfaces of the scene are computed.

Dynamic scenes are scenes where movement in geometry of the scene is present from frame to frame. When geometry of the scene changes, information related to primitives is not consistent and then, no longer valid. Due this, to produce an animation of dynamic geometry, it is necessary to update or rebuild from scratch the structure on each frame. According to Wald et al. in [4], the most common approach in dynamic scenes is to rebuilt from scratch. Thus rebuild time of the acceleration structure is a major concern, because it represents a big percentage of the total amount of time required to get a rendered frame as a final product.

A challenge for real-time ray tracing applied to dynamic scenes has been to get the fastest construction times of acceleration structures. In order to get faster constructions, there have been proposed some optimizations using parallel techniques for CPU and GPU architectures. Such techniques are described deeply on Chapter 2.

In ray tracing one trade-off between build time and ray traversal performance that has not been studied deeply yet, is the geometric complexity of scenes. Complexity of scenes is an important issue to consider to choose an adequate acceleration structure that provides the fastest construction time, while the structure preserves good ray tracing performance for a particular scene. Ray traversal performance and construction time for an acceleration structure are not constant. They depend

on the complexity of the scene, number of primitives, density and concentration of primitives in a surface area[1]

This research proposes an efficient method to extract the characteristics of meshes to make an assertive choice of an acceleration structure for ray tracing in real-time. This method automatically decides the best structure type for a specific scene, given the geometric information. This thesis describes which geometric characteristics impact directly on performance for different acceleration structures. Also, in this research are analyzed some metrics as: build times, frame-rate, and intersections to verify if the selected structure is the most appropriate for a particular geometry. As the performance of acceleration structures varies with different scenes, this method could save much calculation resource by always selecting a structure that maximizes ray tracing performance.

In order to characterize a scene, it was necessary to:

1. Represent each primitive of the mesh. For this purpose, it was chosen a centroid representation due to this effectiveness of representation and for decreasing space complexity.

2. Build an auxiliary structure in order to store information related to distribution and concentration of primitives and then, map each Morton code to a Z-order curve and in this way partition the scene.

3. Use statistical methods to obtain the distribution and spatial concentration of the primitives along the surface mesh, and use information about the geometry of the scene to select an acceleration structure.

[1]The surface area is the sum of the areas for a set of given primitives.

1.2 Objectives

1.2.1 General Objective

The general objective in this thesis is to define a method to extract geometric characteristics of three-dimensional models. Then, using these characteristics as a priori information to select an adequate structure that provides the best performance in construction and ray traversal operations.

1.2.2 Particular Objectives

- Use an efficient representation of primitives.
- Use an efficient structure to store information of primitives.
- Design an algorithm to characterize in real-time a scene.
- Identify important characteristics to make an assertive characterization.
- Identify desirable characteristic of the state of the art acceleration structures.
- Use statistical methods to extract the characteristics of scenes.
- Generate an algorithm that maps the geometric complexity of the scene to an acceleration structure according observed characteristics.
- Use parallel techniques to improve performance of the proposed algorithm.

1.3 Hypotheses

Through observation of behavior for each acceleration structure with diverse scenes, it was possible to identify substantial differences between construction times and ray traversal performance. For instance, the acceleration structure that performed well with one kind of scene did not perform well with different kinds of scene. This

behavior means there is not an acceleration structure that performs well with all scenes. Based on these observations, it was proposed the following hypotheses.

1. Adequate selection of an acceleration structure depends on the geometric characteristics of a scene.
2. Minimization of construction time and maximization of ray traversal performance depends on the right kind of acceleration structure selection.

1.4 Problem Definition

Ray tracing is a rendering technique that requires compute intensive operations. In order to show how compute intensive is ray tracing, a complexity analysis is shown below.

Complexity[2] $O(N \quad M \cdot L)$ (Does not include secondary rays produced by effects)
N= Number of primitives (this number is given by the scene)
M= Number of pixels = Width Height
L= Number of light source

In order to illustrate the complexity of ray tracing with a typical scene, below are shown some numbers. These numbers are from a medium complexity scene.

Complexity $O(N \quad M \quad L)$
N= $252,000$ primitives
M= $1024x1024 = 1,048,576$ pixels
L= 2 light sources
$(252000)(1048576)(2) = 528,482,304,000$ rays.

A ray tracer must find intersection points between many rays and many primitives. The cost of testing each ray against each polygon is prohibitive.

[2]This complexity means that a Ray tracer must trace $(N \cdot M \cdot L)$ rays to render a scene

In order to solve this prohibitive cost, primitives are explicitly sorted yielding an acceleration structure such as Bounding Volume Hierarchy, KD-Tree, 3D Grid [5]. This fact optimizes performance of traditional ray tracers because intersection tests are faster. Nevertheless, for interactive applications such as games this is not enough, it is necessary a further optimization to reach interactive frame rates.

Interactive applications notwithstanding, are highly dynamic in nature [6]. Thus, the ray tracer has to either rebuild or update its internal acceleration structure every frame, and reaching interactive frame rates requires looking at traversal performance and rebuild/update performance [4].

Currently, there is no method or algorithm to characterize geometry of a three-dimensional scene. This characterization is useful to select an acceleration structure that guarantees a lower bound in build time and an upper bound on ray traversal performance.

1.5 Contributions

This research proposes a new and efficient method to characterize scenes using parallel techniques. This characterization takes primitives of the model, and groups them in clusters, those clusters are analyzed using statistical methods to extract certain patterns. Those patterns give some clues about performance of the input geometry with diverse acceleration structures, and they provide quantitative information about distribution and concentration of primitives along the scene.

Information about the geometric complexity of the scene is useful to select the right kind of state of the art acceleration structure. This selection guarantees that the construction time is the minimum while this building attains good quality that provides good ray traversal performance. Through this characterization, it is expected a significant performance improvement. Figure 1.1 shows a flow chart of the proposed solution.

Contributions are summarized and listed below:

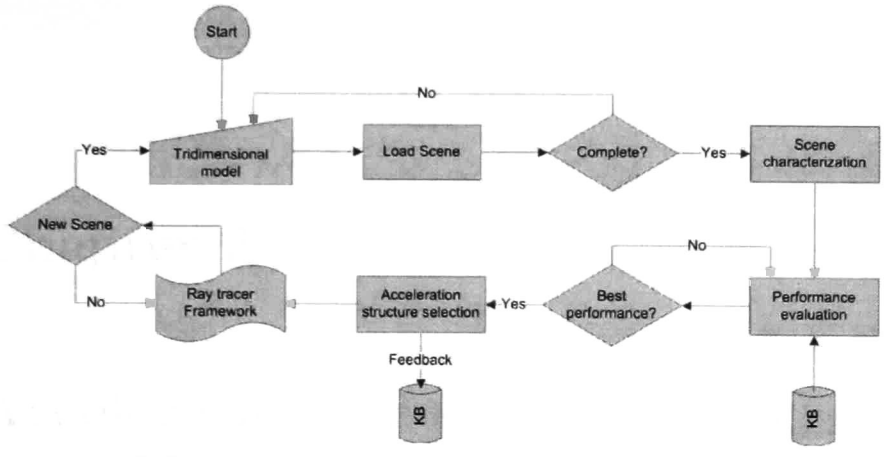


FIGURE 1.1: Flow chart of the proposed contribution.

1. An automatic selection of the best type of structure for a scene.
2. A new and efficient method to characterize three-dimensional scenes.
   (a) A new way to group primitives in clusters in linear time $O(n)$.
   (b) A new partitioning scheme of primitives based on Morton codes.
   (c) An efficient sort of primitives using a hash function $O(1)$.
   (d) A statistical analysis to detect patterns in primitives.
3. A parallel implementation in GPU of proposed method.

Chapter 2

Acceleration structure strategies

The main objective of this chapter is to describe and classify each state of the art acceleration structure. This description is useful to identify how primitives are stored in an acceleration structure. This information can be used as a reference to identify possible application advantages of one structure over the others. Finally, information about acceleration structures give clues on how a structure performs with an input geometry.

In this chapter is addressed some of the most significant improvements in acceleration structures. It includes object bounding volumes, spatial split algorithms and diverse optimization techniques for acceleration structures. These topics will be discuss in subsections 2.1, 2.2, 2.3. Subsection 2.4 focuses on optimizations related to hardware architectures.

2.1 Object Bounding Volumes

A bounding volume is a single and simple volume encapsulating one or more objects of more complex nature [7].

Directly testing an intersection of a ray with a primitive in ray tracing is often very expensive, especially when three-dimensional scenes consist of thousands of

primitives. Due this prohibitive cost, bounding volumes are usually employed for intersection tests before primitives do.

The main idea is for the simpler volumes (such as boxes and spheres) to have cheaper intersection tests than the complex objects they bound. Using bounding volumes allows fast rejection tests[1]. Intersection tests are against the complex bounded geometry (set of primitives) only when the initial intersection query for the bounding volume gives positive. For further information about rejection tests in ray tracing, see [8].

When a ray intersects a bounding volume, sometimes is necessary an additional intersection test, this can increase computation time. However, this additional intersection test depends on how well bounding volume adapts to object's shape (see Figure 2.1).

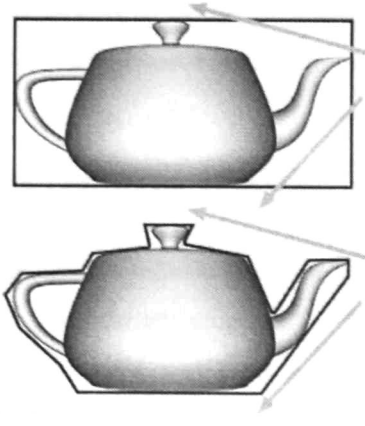


FIGURE 2.1: Object Bounding Volume [7].

Bounding volumes should also be as tight fitting as possible, resulting in a trade-off between tightness and intersection test cost. Many geometrical shapes have been suggested as bounding boxes in [7] (see Figure 2.2)

Axis-aligned bounding box (AABB) is one of the most implemented methods to contain a three-dimensional object in the most state of the art ray tracers [9–11]. It is because its inexpensive intersection cost.

[1]If a ray does not intersect a bounding volume, this means either intersect the primitives that a bounding volume contains. Rejection tests are an early exit test.

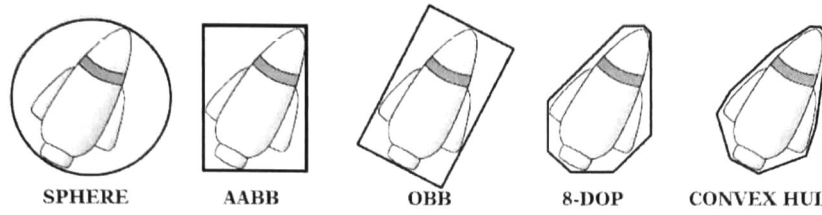


FIGURE 2.2: Type of bounding volumes: Sphere, axis-aligned bounding box (AABB), oriented bounding box (OBB), eight-direction discrete orientation polytope (8-DOP), and convex hull [7].

2.2 Split Algorithms for Ray Tracing

In order to get interactive rendering times for dynamic scenes, one important issue to consider is the kind of split algorithm that an acceleration structure employs. Construction times and ray traversal performance of a particular acceleration structure are directly related to the use of a specific split algorithm. This section analyses the most important space subdivision algorithms nowadays.

2.2.1 Surface Area Heuristic (SAH)

SAH is known as the best method to split the geometry in a scene while maximizing ray traversal performance. This heuristic proposed in [12] and, generally speaking, it partitions the scene in a recursive way until it finds the best partition. To evaluate if a partition is efficient, is used a cost function. The main objective of this function is to find the minimum value in order to maximize ray traversal performance. This heuristic is described in equation 2.1.

Given a set of N primitives contained in a sub-tree that covers a particular 3D volume V, and assuming that a sub-tree gets partitioned into two halves L and R with a number of triangles N_L and N_R and associated volumes V_L and V_R respectively.

$$C(V \rightarrow \{L, R\}) = K_T + K_I \left(\frac{SA(V_L)}{SA(V)} N_L + \frac{SA(V_R)}{SA(V)} N_R \right) \tag{2.1}$$

Where SA is the surface area, and it has associated a volume. K_T and K_I are implementation constants that represent the cost of traversal and intersection respectively.

The cost related to evaluating the cost function for every possible partition in a scene is intractable. In order to restrict the number of possible partitions, Havran suggested in [13] that only $6N$ split locations were evaluated to minimize expected function cost. However, with this construction scheme and the evaluation of only $6N$ partitions is not possible to get interactive construction times.

To get improvements in construction times Wald et al. and Shevstove et al. [6, 14] proposed a scheme using a SAH construction. In this scheme, they made some assumptions: first perfect splits are ignored and second instead of evaluating all potential split planes they only use K equidistant space planes. This improvement may come to lose the minimum cost. However, it was demonstrated through the results obtained that there still produces binned SAH constructions for kd-trees that preserve its quality respect to original SAH construction.

In [6] Wald generalizes binned SAH construction for BVHs using the same approaches used for kd-trees and rebuild from-scratch. Results showed that rates obtained for BVH are up to $10X$ higher than comparable rates of kd-trees in [14, 15]. In Figure 2.3 can be observed how SAH split partitioning maximizes partition of primitives.

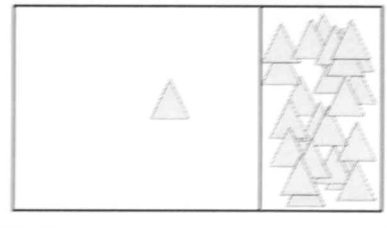


FIGURE 2.3: SAH partitioning scheme[16].

2.2.2 Space Median Heuristic

The space median heuristic is a simple method to partition primitives, it was proposed in [17]. This heuristic algorithm is based on divide primitives in a midpoint. In order to select the midpoint of primitives the minimum and maximum values are considered along an axis as the split plane position (see Figure 2.4).

Wächter et al. proposed in [18] a fast spatial median build that can decrease and beat the fastest SAH constructions for BVH and Kd-trees. This construction is based on space median build [17]. Nonetheless, this construction is not good with all kind of scenes.

This space split method is very fast for constructing acceleration structures, but in general it has bad ray tracing traversal performance. An optimization of this method is to use a cost function similar to SAH in order to minimize the expected cost of traversing (see equation 2.2).

$$f(b) = [LSA(b) \cdot L(b)] + [RSA(c) \cdot (N - L(b))] - [SA \cdot N] \qquad (2.2)$$

Where N is the number of objects in the node, $L(b)$ is the number of objects to the left of the plane at b, and $N - L(b)$ is the number of the right of the plane. The surface area of the left and right sub nodes are $LSA(b)$ and $RSA(b)$, respectively and the surface area of the node is SA.

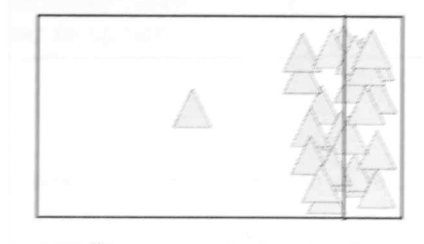


FIGURE 2.4: Space median partitioning scheme[16].

2.3 Acceleration Data Structures

In general, there are three main ray tracing data structures BVH (Bounding Volume Hierarchies), KD-Trees and Grids. These acceleration structures fall in two main categories according to Wald in [4] spatial subdivision and object hierarchy.

In spatial subdivision each point has a spatial representation, and each primitive can reference multiple cells. While object hierarchy structures reference each primitive exactly once. Kd-trees and grids are examples of spatial subdivision. As a counterpart, bounding volume hierarchies (BVH) and BIH are examples of object hierarchies.

Characteristics of each category of acceleration structures can be summarized on Table 2.1, where are analyzed structural characteristics (construction) and ray traversal characteristics.

TABLE 2.1: Characteristics to each classification of acceleration structures.

Characteristics	Spatial Subdivision	Object Hierarchy
Finer subdivision	†	
No repetitive nodes	†	
Efficient ray traversal	†	
Avoid empty cells		†
Faster to build		†
Less memory usage		†
Fewer intersections	†	
Easy to update		†

Spatial subdivision obtains the finest subdivision of primitives. On this category one primitive can be contained in more than one cell. While in object hierarchy the minimum subdivision unit is a primitive. Spatial subdivision techniques can avoid visit one node many times when make an intersection tests. Consequently fewer tests are performed using space subdivision techniques. This is translated to a higher efficiency of ray traversal performance than in object hierarchy.

Object hierarchy is faster than in space subdivision for construction. Spatial subdivision techniques have the finest subdivision, and then the number of references required to store primitives is higher than in the object hierarchy. This subdivision is translated to a less memory usage in object hierarchy. Updates are faster and easier in an object hierarchy than in spatial subdivision. When an update occurs in object hierarchy, only the affected nodes are modified, and the structure remains valid. As a counterpart, in spatial subdivision updates are complex, a little change in a split plane modifies previous partitions.

Spatial subdivision makes less and more efficient ray intersection tests, but the constructions are very expensive. In object hierarchy constructions and updates are faster and more efficient than in spatial subdivision, but ray traversal performance are not as good as in spatial subdivision. Thus, for dynamic scenes have widely used object hierarchy techniques.

2.3.1 Kd-trees

A Kd-tree is a binary tree, it was proposed by Bentley in [19]. On it, every node is a k-dimensional point. Every non-leaf node can be thought of as implicitly generating a splitting hyperplane that divides the scene in two parts. In Kd-trees, k represents the number of dimensions subdivided, which does not have to match the dimensionality of the space used. Kd-trees divide the space along one dimension at a time as is shown in Figure 2.5.

Kd-tree is a spatial subdivision acceleration structure. It has been widely used for static scenes where no change in primitives is present from frame to frame. Construction times for kd-trees are greater than other acceleration structures notwithstanding, the main advantage on using kd-trees is the highest quality of structures that this expensive construction produces. This quality translates to a great ray traversal performance.

Due the expensive construction time of this structure using it for dynamic scenes and getting interactive frame rates is unfeasible because any little change in the

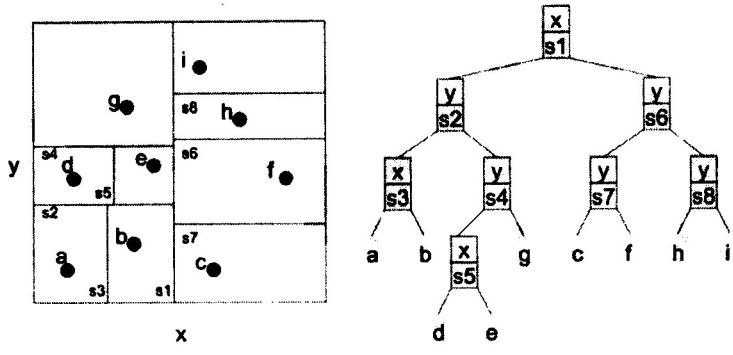


FIGURE 2.5: Kd-tree structure representation [20].

geometry of the scene invalidates the entire structure. In order to face this disadvantage, there have been developed some strategies to accelerate construction for kd-trees. Relevant strategies optimize construction by using parallel techniques [14], explode coherence of motion in animations and dynamic scenes (see Figure) [15] and use lazily rebuilds[2] [21].

There exist some optimizations for construction of kd-trees. In [22, 23] was proposed optimizations for SAH heuristic to build kd-trees. These optimizations were described in section 2.2.1. In [14] are also included optimizations for SAH algorithm and proposes a high parallel construction that builds a kd-tree from scratch each frame.

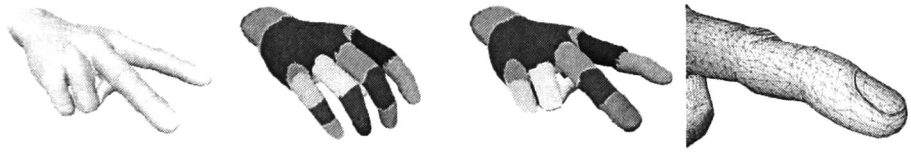


FIGURE 2.6: Motion decomposition together with fuzzy kd-trees allow for ray tracing animated models with continuous deformation by decomposing the motion of the mesh into an affine transformation plus some residual motion [15].

[2]Lazily rebuilds consist in restricting the construction only to subtrees that are traversed by rays.

2.3.2 Bounding Volume Hierarchies

BVHs are part of object hierarchy structure classification, so BVHs are built over the primitives of the scene, and they are delimited by bounding volumes. BVH have been widely implemented in dynamic scenes due their flexibility during construction and consequently in updates [24, 25]. In BVHs incremental updates are allowed [26, 27].

The original set of bounding volumes forms the leaf nodes of the tree. These nodes are grouped as small sets and enclosed within larger bounding volumes. These, are also grouped and enclosed within other larger bounding volumes in a recursive fashion, eventually resulting in a tree structure with a single bounding volume at the top of the tree (see Figure 2.7).

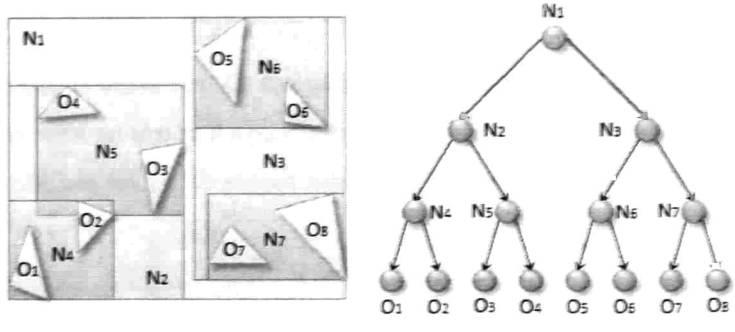


FIGURE 2.7: Tree representation of Bounding Volume Hierarchy [28].

BVH considered by many as an acceleration structure that cannot be competitive in ray tracing traversal performance respect to other acceleration data structures, as kd-trees or grids [4]. Notwithstanding, it has been shown that well-constructed BVH have competitive performance respect to other structures [21, 25–27]. The success of BVH depends on the construction of this structure because this fact determines ray traversal performance. The best known method to construct a BVH is the surface area heuristic. Optimizations for BVHs are focused on: using binned SAH construction [6], using packets for BVH traversal [29, 30] and using parallel techniques for construction of BVH [29]. Wald in [6] proposes a method where BVH can be rebuilt up to $10X$ faster than competing kd-tree implementations.

In [30] Wald et al. proposed a traversal algorithm that reduces the number of ray-box tests by an order of magnitude compared to a pure packet tracer.

2.3.2.1 Bounding Interval Hierarchy (BIH)

One of the variants of BVH is BIH [18]. This variant is one of the fastest methods to build BVH. It uses a fast spatial median split approach.

Unlike classic bounding volume hierarchies which store a full axis-aligned bounding box for each child, the idea of the bounding interval hierarchy is only to store two parallel planes perpendicular to either one of x, y and $z-axis$. Given a bounding box and the axis, the left child L results from replacing the maximum value along that axis of the first plane. In an analogue way the right child R, results from replacing the minimum value of the second plane (see Figure 2.8).

BIH-style builds works only on the centroids, not on actual primitives, but instead of building with an expensive cost function, it successively splits along the spatial median until less than a threshold number of primitives per leaf is reached.

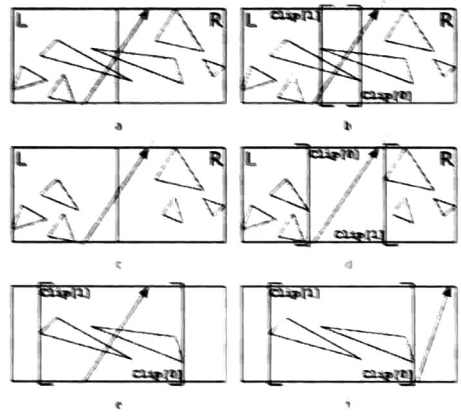


FIGURE 2.8: Bounding intervals hierarchy representation Geometric primitives overlapping the splitting plane have to be referenced in both child volume elements: a) show overlapping splitting plane, b) bounding interval hierarchy for overlapping splitting plane, c) shows assignation of an object to left L or right R child, d) shows the case where a ray traverse an empty volume e) and f) are introduced in order to show how allowing splitting planes to tighten the node volume instead of partitioning it [18].

2.3.2.2 Linear Bounding Volumes Hierarchies (LVBVH)

Lauterbach et al. in [29] proposed an efficient parallel method for constructing BVH. This method is called Linear Bounding Volume Hierarchies (LBVH). This efficient parallel algorithm uses spatial Morton codes[3] to reduce the BVH construction problem to a simple sorting problem.

This algorithm focuses on minimizing the cost of construction while it still produces BVHs of good quality. One of the strongest contributions is a breadth-first queue-based algorithm for construction of BVH. This algorithm incorporates a novel and very fast method to performs the initial splits of the hierarchy in parallel. This algorithm removes many of the existing bottlenecks during the construction. Another contribution of this algorithm is a hybrid construction which builds the highest levels of the tree with LBVH algorithm and makes the rest with SAH algorithm.

The simplest approach to parallelize a BVH construction, is to reduce it to a sorting problem. This approach was taken by LBVH construction. It begins by ordering all n input primitives along a space-filling curve, thus "linearizing" them into a single fixed sequence of length N. Given this proposed sequence the next step is to construct the tree recursively splitting interval of this sequence and creating corresponding nodes. The root corresponds to the interval $[0, n)$, its children to a partition $[0, m), [m, n)$ of the root interval and so on. Thus, every node in the BVH will correspond to a range of indices $[l_i, r_i)$ in the sequence of primitives.

The correct ordering of primitives can be computed simply by sorting them with their Morton codes as the sort keys (see Figure 2.9). The Morton codes themselves encode all the necessary information about where in the tree a particular primitive will go.

[3]Morton Code is a function which maps multidimensional data to one dimension while it preserves locality of the data points. The Morton code of a point in multidimensional space is directly calculated by interleaving the binary representations of its coordinate values. In this way can be represented multidimensional data in a single value.

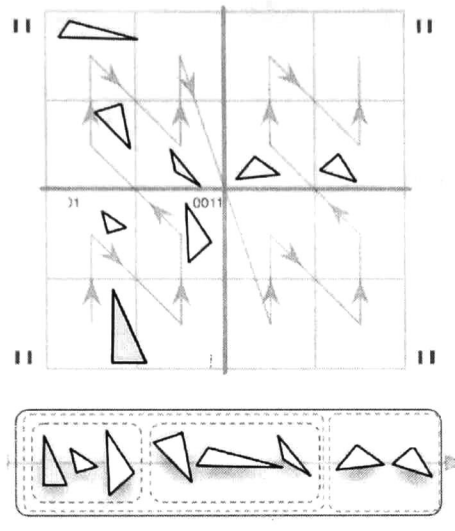


FIGURE 2.9: Morton code ordering of triangles [29].

This algorithm uses parallel techniques during construction. On it, each block i reads in its split item from the input queue and processes it and then, either writes out new splits or a null split to the output queue at positions $2i$ and $2i + 1$ (see Figure 2.10). After that, the output work queue may have several null splits. It is necessary to eliminate these null splits through a compaction kernel and write the result into the input queue. Then again, run the split algorithm on the new input queue as long as there are still active splits left.

2.3.2.3 Hierarchical LBVH Construction (HLBVH)

Based on the work of Lauterbach, Pantaleoni et al. in [31] extended this work through the introduction of a hierarchical algorithm (HLBVH) that reduces the amount of computations and the memory traffic required to build a (LBVH). Their three main contributions are:

The introduction of more efficient reformulation of the Morton curve-based primitive sorting, in order to reduce work and memory traffic. This sorting scheme is up to $8X$ more efficient than state of the art global radix sort procedure until publication of [31].

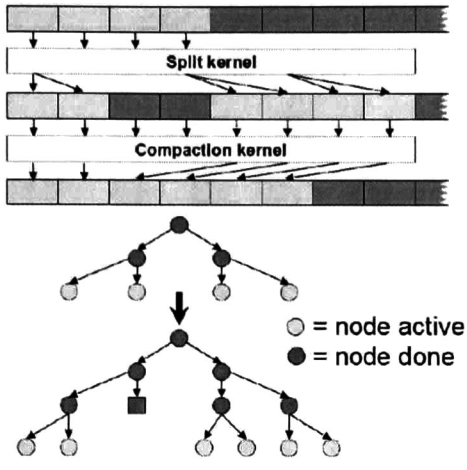


FIGURE 2.10: Work queue construction where all active splits are running in parallel [29].

The implementation of novel node hierarchy emission procedure that improves on the technique proposed by Lauterbach in [29], in order to improve computational and memory efficiency (see Figure 2.11) consults [31] for detailed information.

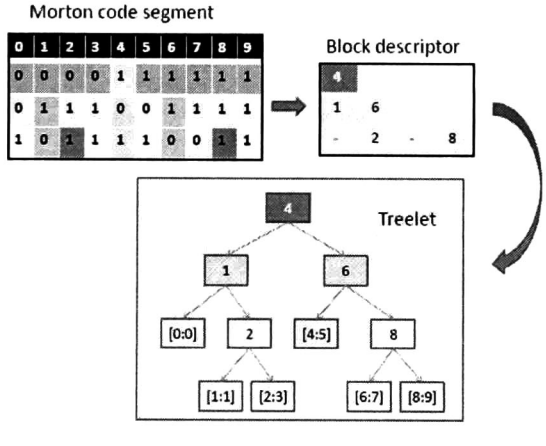


FIGURE 2.11: Treelet emission process [31].

A new proposal of a hybrid algorithm that uses the surface area heuristic to build the top BVH levels, and uses their fast Morton curve-based partitioning within cluster sub trees (see Figure 2.12).

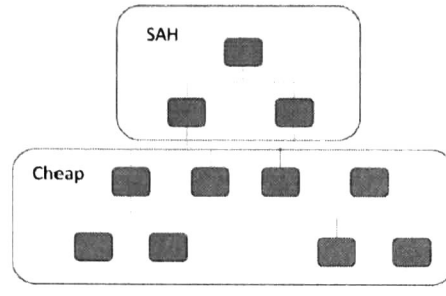


FIGURE 2.12: Hybrid construction of HLBVH [31].

Finally, Garanzha et al. in [32] made some improvements of HLBVH, proposing an algorithm that employed work queues. With this approach was possible to accelerate the overall construction speed by a factor of 5 to $10X$ respect original propose of HLBVH in [31].

The main contribution focused on replacing breadth-first tree traversal primitives used to perform the object partitioning with a single pipeline based on efficient work-queues. This optimization offers superior speeds than originally propose of HLBVH (see Figure 2.13). An additional contribution is the parallelization of the

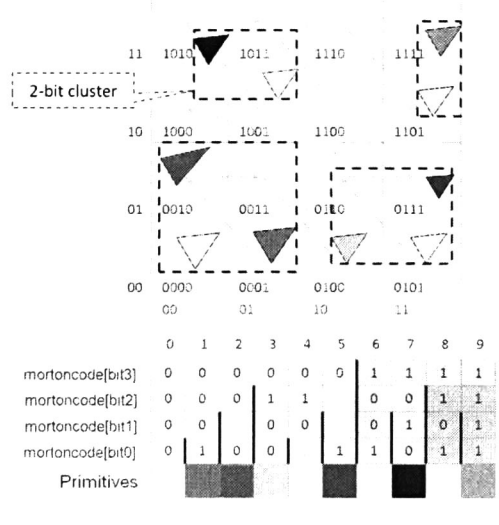


FIGURE 2.13: Assignment of the morton codes to the primitive centroids (a 2D projection and 4-bit morton codes are shown). Bottom: sorted sequence of the primitives where morton codes are used as keys [32].

Surface Area Heuristic (SAH) that combines advantages on using a task-based pipeline with the parallel efficiency of the binning scheme [6].

2.3.3 Grids

A Grid is an acceleration structure that subdivides the space in a scene uniformly (see Figure 2.14), as a consequence grids are not useful with complex scenes where concentration of primitives in a surface area is high. Moreover, it is considered that construction of grids is faster and easier than the construction of kd-trees and BVH.

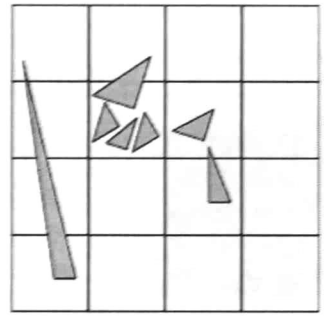


FIGURE 2.14: Assignment of the morton codes to the primitive centroids (a 2D projection and 4-bit morton codes are shown). Bottom: sorted sequence of the primitives where morton codes are used as keys [32].

Grids can be useful for static and dynamic scenes with uniform distribution of primitives. For this structure, some optimizations have been developed: To avoid duplicate intersection tests is proposed a technique called mailboxing [33]. In this technique, every projected ray has a unique ID, and every object has a mailbox that stores the rays IDs. In this way before to do an intersection test, the mailbox of the object is checked in order to verify if a previous intersection was made.

Ize et al in [34] proposed a linear construction where insertions are parallel. First, buckets are divided by the number of threads, and then each thread takes a set of buckets and writes the triangles on it (see Figure 2.15).

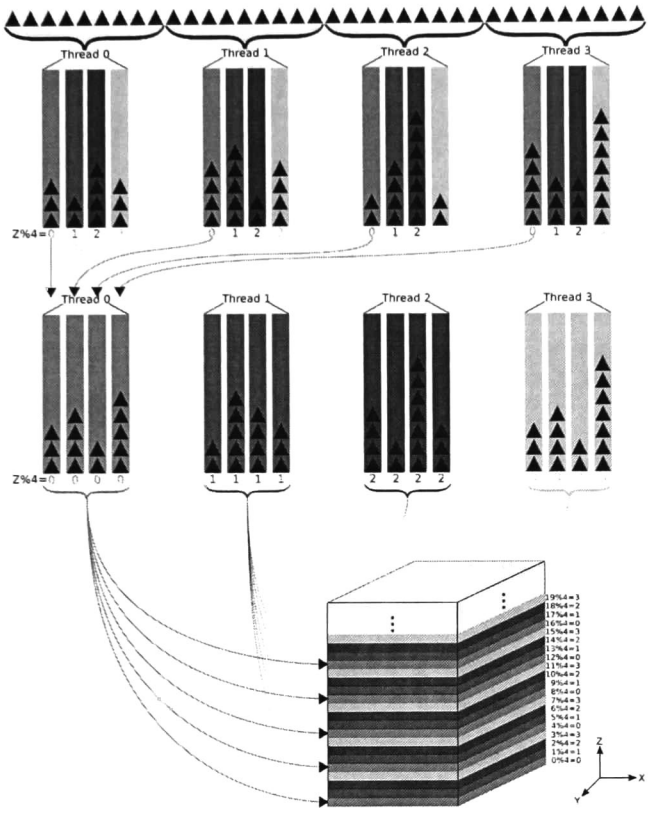


FIGURE 2.15: Given 4 processors, in the sort-middle build, the T triangles are equally divided among the 4 threads. Each thread has 4 buckets which are used to sort its $\frac{T}{4}$ triangles, based on the z cell index modulo 4. After all 4 threads have finished sorting, the threads regroup the buckets and fill in the grid [34].

A coherent grid traversal is proposed by Wald et al. in [35], where an algorithm is used to traverse a grid slice by slice instead of doing it cell by cell (see Figure 2.16). This algorithm evaluates vertical slices in order to find cells that a ray intersects. With this scheme, the grid traversal is accelerated by more than a factor of 10 and is achieved ray traversal performance competitive with fastest known kd-tree implementations.

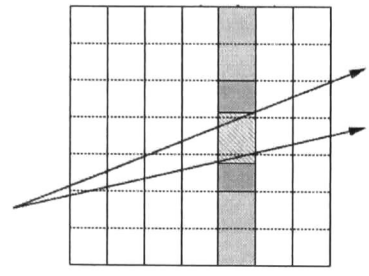


FIGURE 2.16: Coherent grid traversal. Evaluate vertical slices (blue) in order to find cell that a ray intersects [35].

2.4 Hardware Architectures

Make a direct comparison between many acceleration structures is complex, due effectiveness of each structure depends on the complexity of the scene and efficiency of current optimization techniques for a particular acceleration structure. The software optimizations for acceleration structures were discussed in previous sections. Nevertheless, in most of the cases each software optimization is oriented mostly to a specific hardware GPU or CPU architecture.

Construction of acceleration data structures is pretty compute intensive. To affront this challenge and to decrease construction times, diverse parallelization techniques have been proposed. It has been observed that these techniques scale well on parallel architectures [9, 10, 36, 37]. Some optimizations for hardware architecture will be analyzed below. These architectures developed for a specific hardware.

2.4.1 T&I Engine

Nah et al. in [36] proposed the T&I engine. It is a hardware architecture for efficient tree traversal and intersection tests. This architecture includes three concepts: The tree traversal unit with ordered depth-first layout, the three-phase intersection test unit and the ray accumulation unit for hiding memory latency. Figure 2.17 illustrates the system organization of T&I engine that includes a ray dispatcher (RD), traversal units (TRVs), list units (LISTs), the first intersection

unit(IST1s), and the second intersection unit(IST2s). In this architecture, each unit is connected by buffers. These buffers pass a ray from one unit to others.

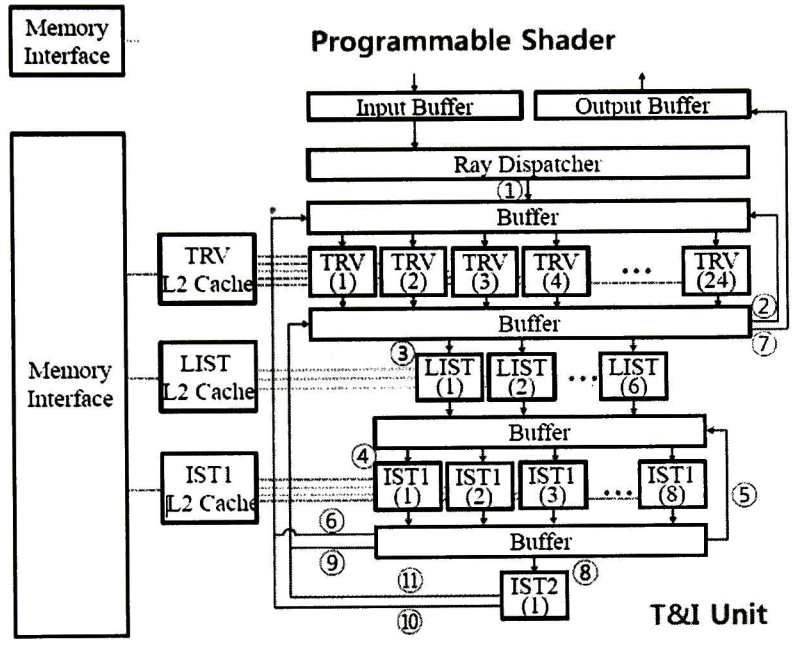


FIGURE 2.17: Overall system architecture of the T&I engine [36].

2.4.2 Intel Many Integrated Core(MIC) Architecture

Wald in [9] designed a framework for the fast construction of SAH BVH which uses an architecture designed by Intel, Many Integrated Core (MIC) Architecture to achieve performance improvements that can compete and improve best implementations nowadays in GPU and CPU architectures [6, 31, 34]. Optimizations of this hardware architecture can be generalized to other acceleration structures as: grids or kd-trees.

Figure 2.18 shows how a job is divided and dispatched to different threads of the framework. Wald uses a structure-of-array-of-structures (AoSoA) as data layout to manage storage of his framework (see Figure 2.19).

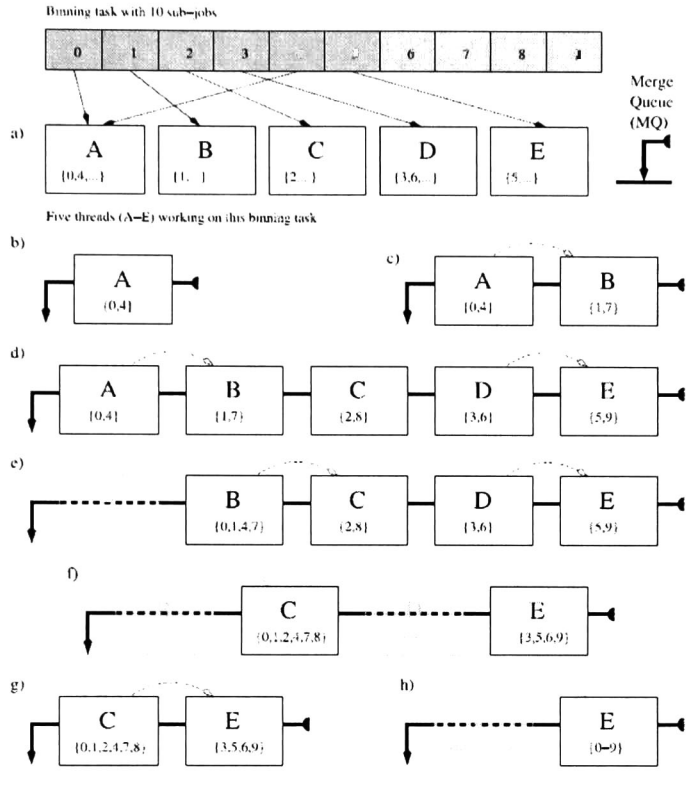


FIGURE 2.18: Horizontal merging: In this hypothetical but plausible example, five threads (A-E) are collaboratively binning a task consisting of 10 subjobs (0-9) [9].

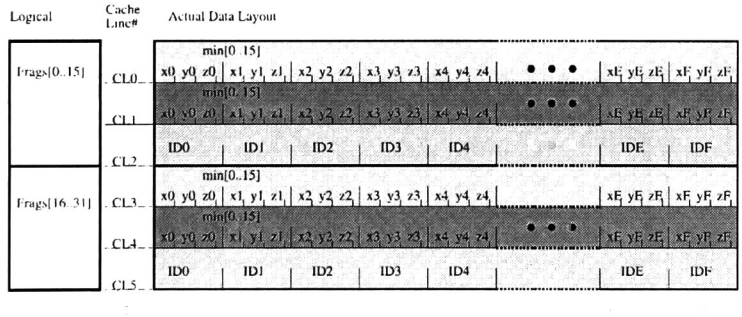


FIGURE 2.19: AoSoA data layout with 10-bit quantization per coordinate. Each chunk of 16 fragments requires only three consecutive cache lines, this representation allows very efficient SIMD processing. Each [begin . . end) region of chunks is contiguous, and can be addressed with a single scalar register [9].

2.4.3 Specialized Microarchitecture for construction of BVH

Doyle in [37] designed the first specialized micro-architecture for the construction of binned SAH BVH for ray tracing. Figure 2.20 and 2.21 show the overall system micro-architecture, and it shows the central units of the micro-architecture. DRAM interface consists of a number of RAM pairs. Each RAM pair consists of two memory channels. Primitives are divided over the RAM pairs. The *upper builder* reads and writes directly to DRAM and is responsible for constructing the upper levels of the hierarchy. The *sub-tree builders* are responsible for constructing the lower levels of the hierarchy.

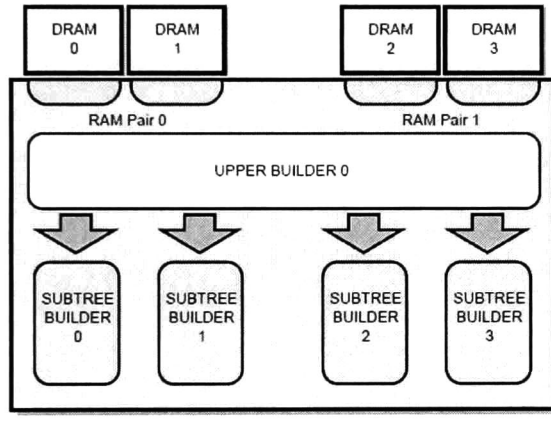


FIGURE 2.20: A top level overview of the binned SAH BVH construction hardware, showing memory interfaces, upper and subtree builders [37]

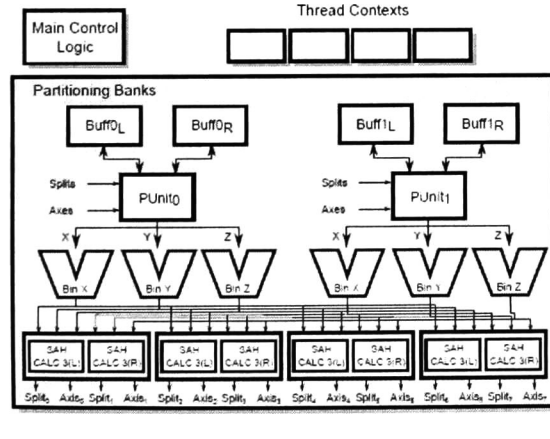


FIGURE 2.21: Detailed diagram of the subtree builder micro-architecture [37]

This micro-architecture offers: Acceleration of up to $10X$ for binned SAH BVH compared to manycore platforms, low memory bandwidth consequently a low memory footprint and minimal hardware resources consumption. This micro-architecture offers construction times lower than software implementations [6, 31, 32].

Chapter 3

Characterization of Scenes

There are several optimization techniques for acceleration structures. However, despite all these proposed optimization techniques, there exist a trade-off between build time and ray traversal performance of acceleration data structures that depends on geometric complexity of the scene. This geometric complexity is related to spatial distribution of primitives in the scene. Performance of acceleration structures varies greatly according to the complexity of scenes, and it is necessary to use knowledge about spatial distributions of primitives to use this information to increase performance of acceleration structures.

In this chapter is addressed a new method to analyze the geometric complexity of scenes. This method implements statistical methods during the analysis to get the geometric complexity. This method analyzes how the complexity of scenes is related to construction times and ray traversal performance for each acceleration structure. As a product of this analysis, it is obtained a lower bound. A lower bound guarantees that a selected structure has the fastest construction times and produces the highest frame-rate.

3.1 Contributions

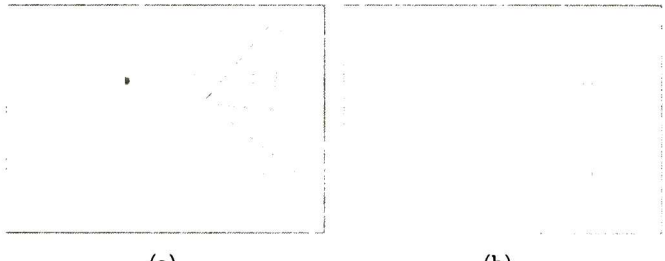
Geometric complexity of the scene is obtained by looking how primitives are distributed and spatially concentrated along the scene. In order to get these properties was necessary to: represent each primitive of the mesh, generate a structure to store information of concentration and distribution of primitives and extract relevant characteristics of the geometry.

3.1.1 Representation of primitives

In order to represent primitives, it was selected the simplest and inexpensive way to do it; a centroid representation due the effectiveness of representation and to decrease space complexity. A centroid representation can be computed by using Equation 3.1. This representation preserves physical locality of primitives while it is possible decrease space complexity reducing a primitive to a single representative point.

$$Centroid = \left(\frac{\sum_{i=1}^{n} x_i}{n}, \frac{\sum_{i=1}^{n} y_i}{n}, \frac{\sum_{i=1}^{n} z_i}{n} \right) \tag{3.1}$$

Figure 3.1 shows how a primitive can be represented by a centroid. In this figure is shown how this representation maintains the locality of primitives.



(a) (b)

FIGURE 3.1: Centroid representation of a primitive. a) Shows a primitive representation that in this case is composed by three vertices b) Shows how a primitive can be represented by compute a centroid.

Figure 3.2 shows how centroids represents a three-dimensional model. This figure shows how this representation maintains and preserves details of the model.

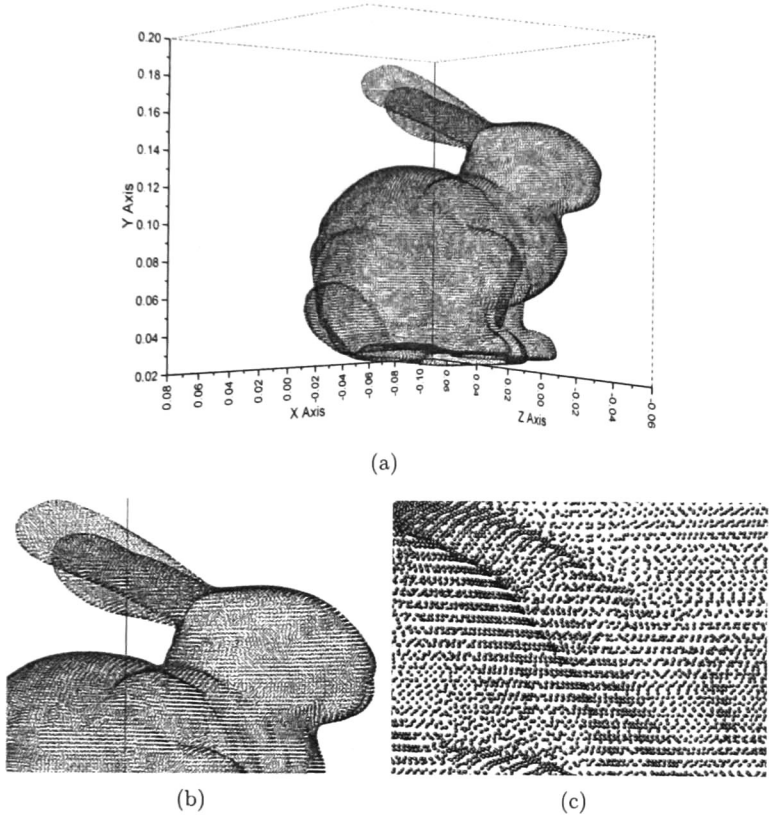


FIGURE 3.2: Centroid representation of Stanford Bunny model composed by 69,451 primitives a)Shows a complete model representation b) Shows a close-up of the model where some details can be observed c) Shows a close-up of the model where individual centroids can be observed.

Once the algorithm compute the centroids of primitives. It is necessary to obtain information about spatial location of primitives. In order to compress the representation of primitives, it was used the z-order curve see Figure 3.3. The z-order curve defines a Morton code representation [38].

For each centroid that represents a primitive, it was calculated a spatial Morton code to translate three-dimensional coordinates in one-dimensional value. This transformation is obtained taking bits of each value of the centroid's coordinate

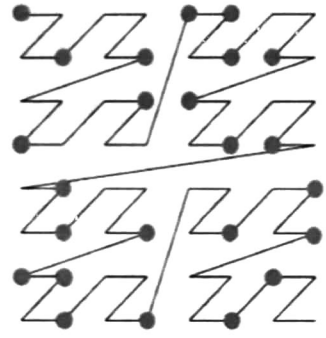


FIGURE 3.3: Implicit Z-order curve of morton codes.

axis and then interleaved it to produce a single value see Figure 3.4. Lauterbach used this way of represent primitives in [29].

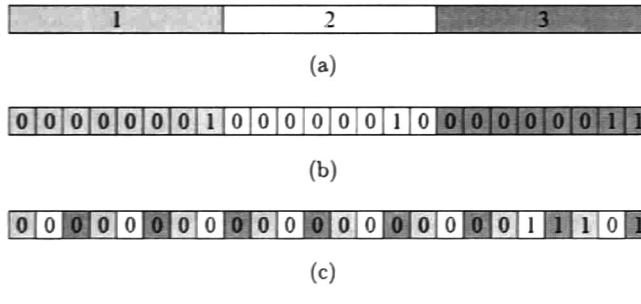


FIGURE 3.4: Morton code representation of three-dimensional coordinates. a) Shows a centroid's three-dimensional coordinates C=(1,2,3). b) Shows a binary representation of each axis. c) Shows a computed Morton code that was obtained by interleaving bits of binary representation of each axis.

3.1.2 Auxiliary Structure for Characterization

In order to observe how primitives are distributed and concentrated, it is necessary to build a structure that allows store this information. It was selected a histogram as an auxiliary structure to store spatial information related to primitives. Each bucket in the histogram represents an equivalence class.

An equivalence class is a spatial range given by the number of classes and the minimum and maximum values of Morton codes. The number of equivalence classes is

given by the formula of Doane [39], which is a modification of the Sturges' formula [40]. The Sturges' formula assumes that the input data is normally distributed, and this assumption is not applicable to primitives of all scenes. For this reason, it is better to use the formula of Doane. and additionally this formula is better than the Sturges for large samples as is the case of primitives.

$$k = 1 + log_2(N) + log_2\left(1 + \frac{\mid g_1 \mid}{\sigma_{g_1}}\right) \tag{3.2}$$

Where g_1 is the estimated third-moment-skewness of the distribution and σ is the sample standard deviation.

Once the algorithm calculates the number of classes, is necessary to define the width of the classes in order to define the membership of a primitive to a class.

$$width = \frac{maxValue - minValue}{k} \tag{3.3}$$

Finally, in order to verify membership of a primitive to an equivalence class, it must be computed the next equation.

$$\prod_{i=0}^{k} \{x \mid \quad i \cdot width \leq x \leq (i+1) \cdot width\} \tag{3.4}$$

An equivalence class denotes affinity between primitives, this affinity, indicates that primitives are physically close to each other. Thus, primitives in the same equivalence class are next to each other. Given a set of input primitives P that belong to a scene is necessary to map each Morton code that represent a primitive to an equivalence class. For this propose, there exist two approaches:

- Sorting Morton codes [41] and then, distribute them to classes (Radix Sort). Complexity $O(nlogn)$.
- Looking for a position in the histogram for each Morton code. Complexity $O(n \cdot k)$.

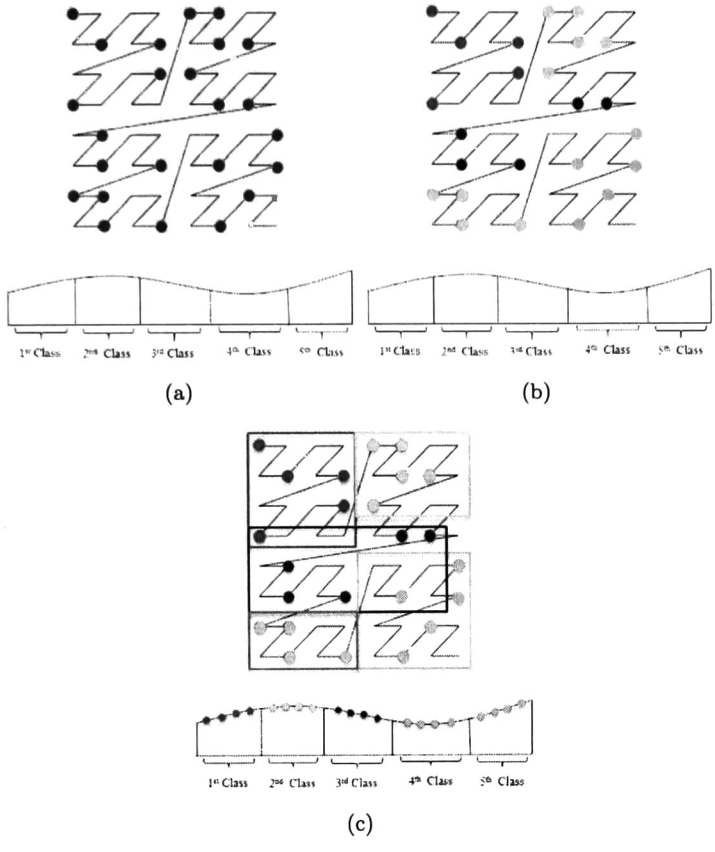


FIGURE 3.5: Partition scheme of equivalence classes. a) Shows implicit order of Morton codes. b) Shows how the method groups primitives by affinity. c) Shows the proposed method partition primitives in equivalence classes.

Both methods are quite expensive. To decrease complexity, in this thesis is proposed a method to map a Morton code to an equivalence class in constant time $O(1)$. This method uses a hash function, and this function performs an arithmetic operation and returns the position on the histogram for a given Morton code see Figure 3.5.

Finally, a scene is partitioned in equivalence classes, those equivalence classes group primitives by affinity as is shown in Figures 3.6.

Pseudocode of the algorithm 1 describes the process to construct an auxiliary structure that stores information related to input geometry for a specific scene.

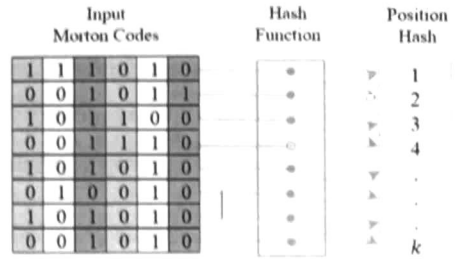


FIGURE 3.6: Hash function that maps a spatial Morton code to an equivalence class.

```
1  map<int,vector<int>>histogram;
2  main()
3  |  foreach primitive do
4  |  |  calculateCentroid(primitive.vertices);
5  |  |  calculateMortonCode(primitive.centroid);
6  |  end
7  |  calculateBounds();
8  |  generateEquivalenceClasses();
9  |  generateHistogram();
10 end
11 Function generateEquivalenceClasses()
12 |  k= 1 + log2(numPrimitives) + log2 (1 + (g1/σg1));
13 |  width=bound/k;
14 |  map<int,vector<int>>histogram(k);
15 end
16 Function generateHistogram()
17 |  foreach primitive do
18 |  |  pos=hashFunction(primitive.mortonCode);
19 |  |  histogram[pos]+=1;
20 |  end
21 end
22 Function hashFunction(value)
23 |  pos=ceil(abs(minVal)+val/width);
24 |  return pos;
25 end
```

Algorithm 1: Pseudo-code to construct a structure that contains information about concentration and distribution of primitives.

3.1.3 Analysis of scenes

Performance of acceleration structures varies considerably depending on the input geometry. The main objective of this paper is to find out which characteristics

are strictly related to performance of acceleration structures. For this reason, this paper proposes a statistical analysis of the geometry for scenes. The information that the analysis produces ought to be useful to know which structure is the best suited for a given geometry.

Generally speaking, the performance of an acceleration structure is given by its construction. Using this premise, the first step was to use this knowledge in order to search specific patterns in a scene. This patterns or characteristics give some clues about performance of a specific acceleration structure with an input geometry. The analysis presented in this paper focuses on detect those patterns in a single pass $O(n)$.

The most common patterns or characteristics detected in scenes, and they have a positive or negative influence in construction of acceleration structures are:

1. Distribution of primitives in the scene.
2. Concentration of primitives in the scene.
3. Empty classes
4. Kind of scenes.
5. Adjacent primitives.

Distribution of primitives in the scene. This pattern affects performance of structures. A well behaved uniform triangle distribution favors all structures, but particularly a grid structure is the less adaptive of all structures. Since a grid structure subdivides the space uniformly, if the geometry is not uniform ray traversal performance of grids is poor. Nevertheles, Kd-trees and BVHs, are structures more adaptive to non uniform geometry.

In order to measure quantitatively if a given geometry is uniform, the Chi-Square goodness of fit test is used.

$$\chi^2 = \sum_{i=1}^{n} \left(\frac{(O_i - E_i)^2}{E_i} \right) \tag{3.5}$$

Once the algorithm calculates the Chi-Square goodness of fit test, the next step is to verify if the current value of Chi-Square is less than the critic value. If it is less or equal, the observed distribution of the histogram fits a probability distribution, in this case an uniform distribution. To calculate the critic value is necessary to provide the degrees of freedom $(k-1)$ and probability.

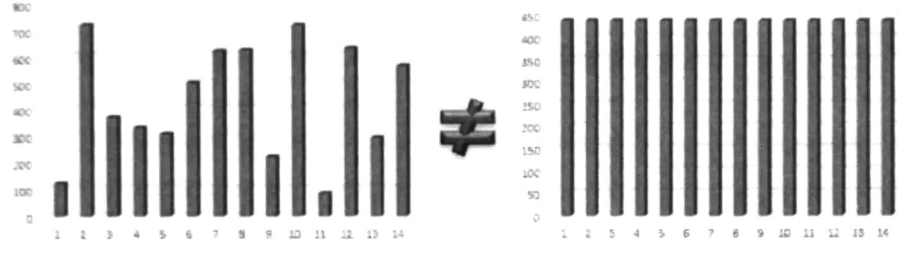


FIGURE 3.7: Chi-Square goodness of fit test measures how different are two probability distributions.

Concentration of primitives in the scene. The concentration of primitives focused in a defined area is a pattern that influences performance of scenes, this kind of concentration presents a disadvantage to some structures that employ uniform partition schemes.

To analyze this pattern is necessary to observe if there exist a significant difference in the number of primitives in an equivalence class respect the others. in other words is necessary to detect the mode or modes in the input data. To determine a class has a high concentration of primitives, It is proposed a threshold, this threshold is the arithmetic mean of all classes.

$$\overline{X} = \frac{c_1 + c_2 + c_3 + \cdots + c_k}{k} \tag{3.6}$$

A class with a higher number of primitives than the other classes denotes a high concentration of primitives in a determined region. This concentration affects some structures that cannot handle this type of geometry.

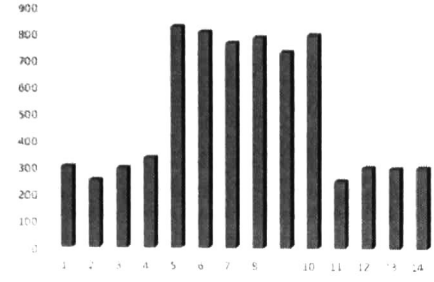


FIGURE 3.8: Figure shows a high concentration of primitives in classes 5-10.

Empty classes. When equivalence classes are empty or poorly populated, this means structures that allow empty nodes can perform unnecessary tests when rays traverse nodes of the structure. An empty class generates invalid nodes in the structure, and it also decreases ray traversal performance.

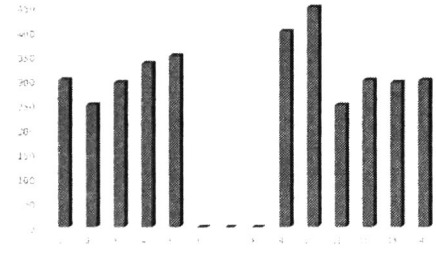


FIGURE 3.9: Figure shows empty classes in the histogram classes 6-8.

Kind of scenes. One important characteristic to consider is the kind of the input geometry for a given scene. If the geometry of the scene transforms from frame to frame, a structure must be rebuilt or regularly update. Thus, the main objective for this kind of scenes is to choose a structure that can be rebuild from scratch or update efficiently rather than select a structure that has the most efficient ray traversal performance, but has higher construction times.

For scenes where geometry does not change, the construction time is not significant. The most important aspect for this type of scenes is an efficient ray traversal performance. For this scenes, high construction times can be amortized by rendering many frames.

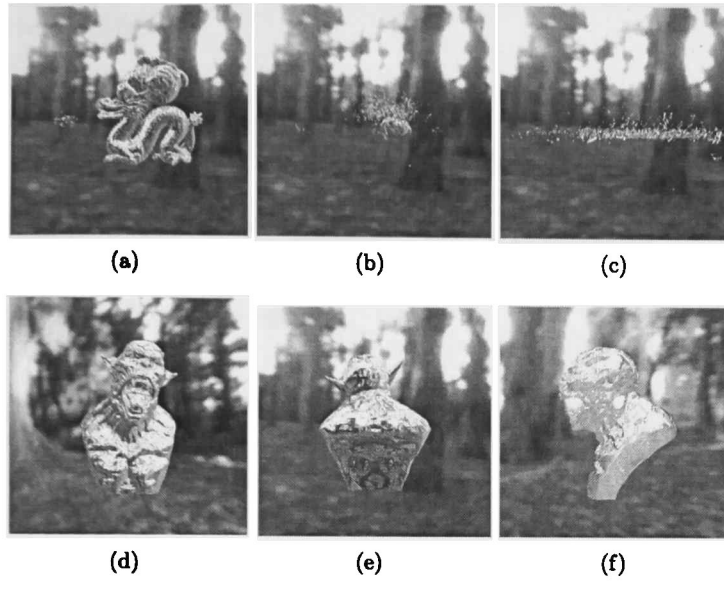


FIGURE 3.10: Figures a, b and c are three frames of a dynamic model Exploding Dragon and Bunny (252K). Figures d, e and f are three frames of a static Daemon model (935K).

Adjacent primitives. When primitives are adjacent to each other and constitute an object (rigid objects) there exist a coherence between primitives that form a particular object, and consequently there exist a coherence in rays that traverse these primitives. Is more efficient traverse primitives that have coherence than primitives that do not have it.

Pseudocode of the algorithm 2 describes the process to extract characteristics and to detect patterns in the input geometry for a specific scene.

All these cited characteristics must be taken into account to select an acceleration structure. This prior knowledge of geometric complexity of the scene can be used properly to select an acceleration structure that minimizes construction times and maximizes ray tracing traversal performance.

```
1  expFreq=numPrim/k;
2  XSquare=0;
3  main()
4  |  for obsFreq in equivalenceClasses do
5  |  |  XSquare+=pow(obsFreq-expFreq,2)/expFreq;
6  |  end
7  |  mean=calculateMean(numPrimitives);
8  |  stdDev=calculateStdDeviation(mean,numPrim);
9  |  isUniformDistributed(XSquare,mean,stdDev);
10 |  hasEmptyClasses(histogram);
11 end
12 Function calculateMean(n)
13 |  mean=0;
14 |  for m in mortonCodes do
15 |  |  mean+=m;
16 |  end
17 |  return mean/n;
18 end
19 Function calculateStdDeviation(mean, n)
20 |  stdDev=0;
21 |  for m in mortonCodes do
22 |  |  stdDev+=pow((m-mean),2);
23 |  end
24 |  stdDev=sqrt(stdDev/n);
25 |  return stdDev;
26 end
27 Function isUnifDistributed(chiSquare, stdDev)
28 |  cVal=computeCriticVal(k-1,0.005);
29 |  if XSquare>cVal and stdDev is near to 0 then
30 |  |  return true;
31 |  end
32 |  else
33 |  |  return false;
34 |  end
35 end
36 Function hasEmptyClasses(histogram)
37 |  for h in histogram do
38 |  |  if h is empty then
39 |  |  |  return true;
40 |  |  end
41 |  |  else
42 |  |  |  return false;
43 |  |  end
44 |  end
45 end
```

Algorithm 2: Pseudo-code of proposed algorithm to verify how well obtained data fits to a uniform probability distribution.

Chapter 4

Parallel Characterization

This chapter addresses the method described in chapter 3 as an implementation on a modern programmable GPU. It describes optimization techniques used to ensure maximal utilization of GPU resources. Figure 4.1 shows a general overview of the interaction between diverse components during characterization, and it also describes which operative unit do a particular task. As a result of characterization, it is selected an acceleration structure. This method amortizes its complexity preloading geometric information of the model on the GPU. This preload substantially decreases memory latency avoiding communication between CPU and GPU. See Apendix for detailed information of graphic card.

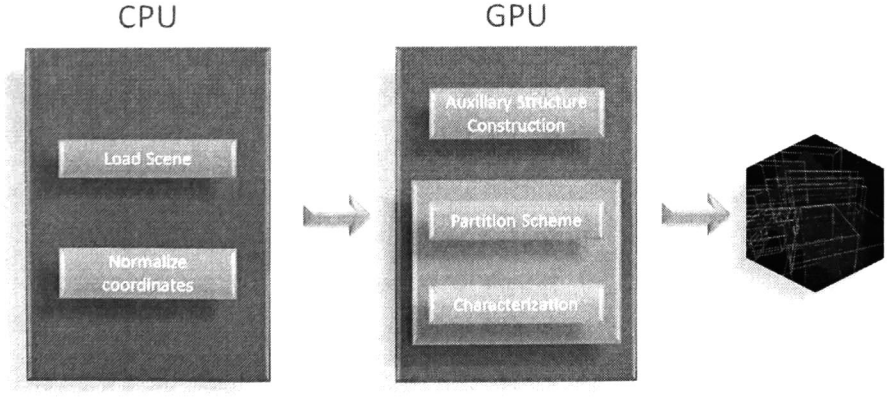


FIGURE 4.1: Architecture of parallel implementation

4.1 Parallel construction

The construction of the histogram is fully parallelized in GPU in order to decrease construction time, and guarantee a fastest construction time. This construction has different parallel phases: 1) Calculate a centroids and a Morton codes. 2) Calculate the standard deviation of the sample. 3) Map each Morton code to a histogram position.

4.1.1 Calculate centroids and Morton codes

This task is completed by launching many threads through a kernel that calculates for each primitive a centroid. A thread sums the values of each vertex by axis and then the result is divided by the number of vertices see Figure 4.2. When a thread calculates a centroid it needs to compute equation 4.1.

$$\frac{V_1.axis + V_2.axis + V_3.axis}{\mid V \mid} \tag{4.1}$$

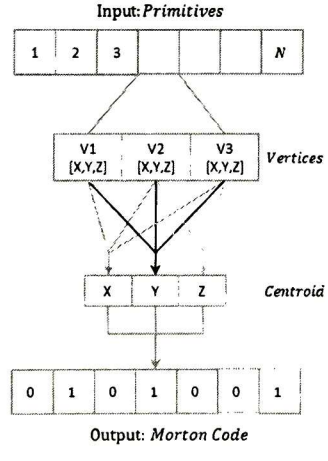


FIGURE 4.2: Parallel calculation of centroids and Morton codes.

Threads must perform the operation by each axis in a scatter pattern. For instance, threads take X-axis of each vertex and then, they sum those values in a reduction

process. The result is divided by the number of vertices. This process must be repeated in each axis.

4.1.2 Calculate the standard deviation of the sample

This statistical metric is necessary in order to determine the number of equivalence classes. To calculate the standard deviation of the sample is necessary to obtain the arithmetic mean, the mean can be obtained by a parallel reduction. In this reduction, each thread takes two numbers and then makes an addition. Then a thread uses the results of previous additions and sums them in a recursive process, as a product of the end of the process it finally obtains a single value, the sum of all input elements see Figure 4.3. Then, to get the mean the obtained value is divided by the number of input elements. The overall process of compute the arithmetic mean is given by equation 4.2.

$$\overline{X} = \frac{\sum_{i=1}^{N} m_i}{\mid m \mid} \tag{4.2}$$

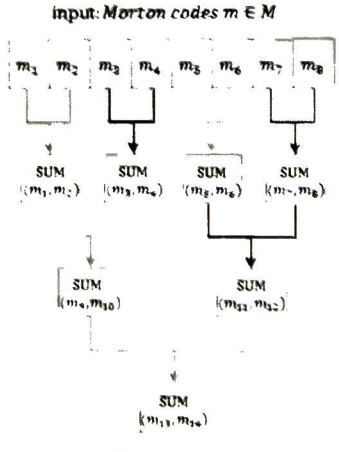


FIGURE 4.3: Parallel sum reduction.

4.1.3 Map each Morton code to a position in the histogram.

This process is completed by assign a thread for each input Morton code. Each thread executes an arithmetic operation. This hash function assigns a Morton code to a position in the histogram see Figure 4.4. It is necessary to employ atomic memory operations in order to guarantee that only a single thread at the same time can access and modify a memory location. Employing this technique the algorithm get consistent values in the histogram.

Each thread performs the arithmetic operation defined in equation 4.3 in order to assign a position to a Morton code.

$$Pos = \frac{m \in M}{width} \tag{4.3}$$

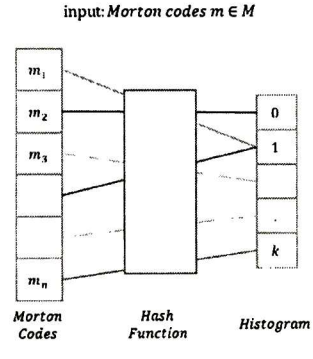


FIGURE 4.4: Parallel map a Morton code to a histogram.

4.2 Parallel Characterization

The characterization of the input geometry is fully parallelized in GPU. The most compute demanding calculation during characterization is the Chi-Square metric. In order to compute this metric, the implementation uses parallel techniques.

4.2.1 Chi-Square Metric

To compute the Chi-Square value is necessary make it in two phases. The first phase (blue rectangle) makes a subtraction between observed frequency and expected frequency, and then the result to the square is divided by the expected frequency see equation 4.4. The second phase (red rectangle) takes all values, and it makes a parallel sum reduction see equation 4.5. The parallel calculation of a Chi-Square value is illustrated on figure 4.5, in this figure each color of the lines represents a thread.

$$\frac{(O_i - E_i)^2}{E_i} \tag{4.4}$$

$$\chi^2 = \sum_{i=1}^{n} \left(\frac{(O_i - E_i)^2}{E_i} \right) \tag{4.5}$$

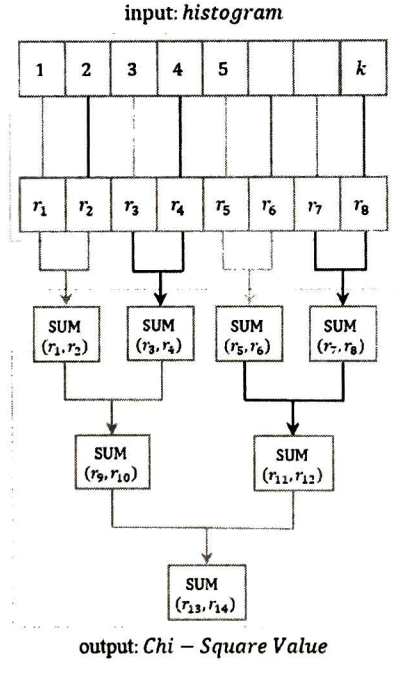


FIGURE 4.5: Parallel calculation of Chi-Square metric. This calculation is divided in two phases

4.3 Generation of Equivalence Classes

Each bucket in the histogram represents an equivalence class, and the number of elements in each bucket represents the cardinality of an equivalence class. Every primitive grouped in an equivalence has the properties described below:

- Primitives in the same equivalence class are close to each other.
- Primitives in the same equivalence class are part of the same object.
- Primitives in the same equivalence class preserve coherence during traversal.
- Primitives in the same equivalence class are stored in contiguous segments of memory, this property allows a coalesce access pattern; this pattern improves access performance.

These properties are pretty useful to know the distribution of geometry for a particular scene. Nevertheless these features also give information about coherence of primitives during traversal. This information can be used to build a structure and maximize coherence in rays during intersection tests.

The parallel construction completes the histogram. It contains references to primitives in order to minimize memory footprint. Graphically the partition method subdivides primitives in equivalence classes. Figure 4.6 shows equivalence classes of the Stanford Bunny model (69K). On it, each color represents an equivalence class. In this figure is observed how primitives are spatially located near to each other in the same equivalence class.

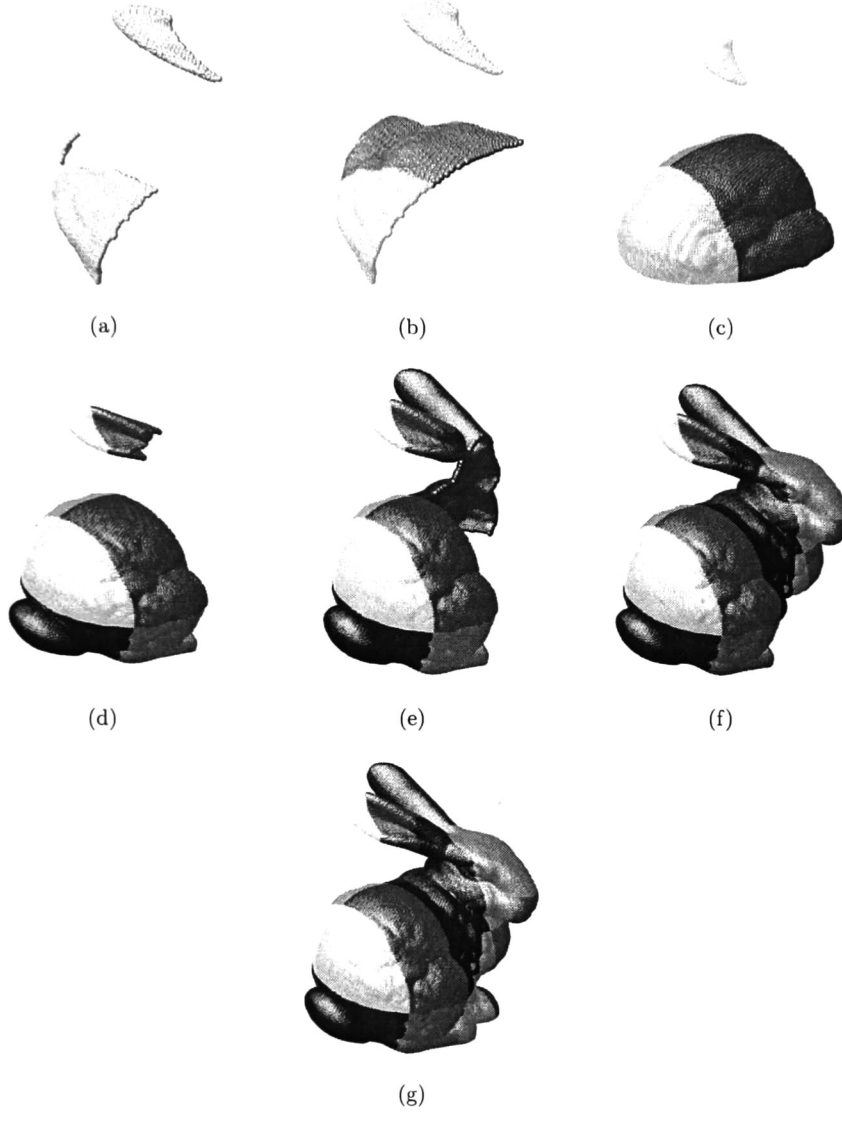


FIGURE 4.6: Proposed partitioning scheme. Equivalence classes in a Stanford Bunny model (69K). In these images each color represents an equivalence class.

Chapter 5

Results

This section exhibits the results of characterize diverse scenes. These scenes varies in the number of primitives and complexity. It also differs in the kind of motion in scenes. The models used are Stanford Bunny (69K primitives), Crytek Sponza (279K primitives), Happy Budha (1M primitives), Welsh Dragon (2M primitives) and Exploding Dragon and Bunny (252K primitives) this last scene is dynamic, and it has 16 raw sequences.

The metrics used in order analyze how efficient is an acceleration structure with the input geometry are construction time, frame rate and intersections. These metrics acts as a reference in order to verify if the characterization selects the best suited structure for a specific geometry.

The construction times and frame rates were obtained from the implementation in Compute Shader proposed by Garicía et al. [10]. In order to execute the tests of proposed method, and to take construction times, it was used a 3.30 GHz Intel core i5 2500, with 4Gb of 1333 MHz DDR3 RAM (2x 2GB) compiled as a 64-bit application, the GPU is an NVIDIA Geforce GTX590 video card. To get the frame rate the scenes are rendered ad 1024 x 1024 pixels.

The proposed method groups primitives in equivalence classes. These classes have a number of primitives given by the Z-order of Morton codes. This way of group primitives allows to do a statistical analysis to get information of geometry. In

Figure 5.1 are shown diverse scenes, where each color represents an equivalence class.

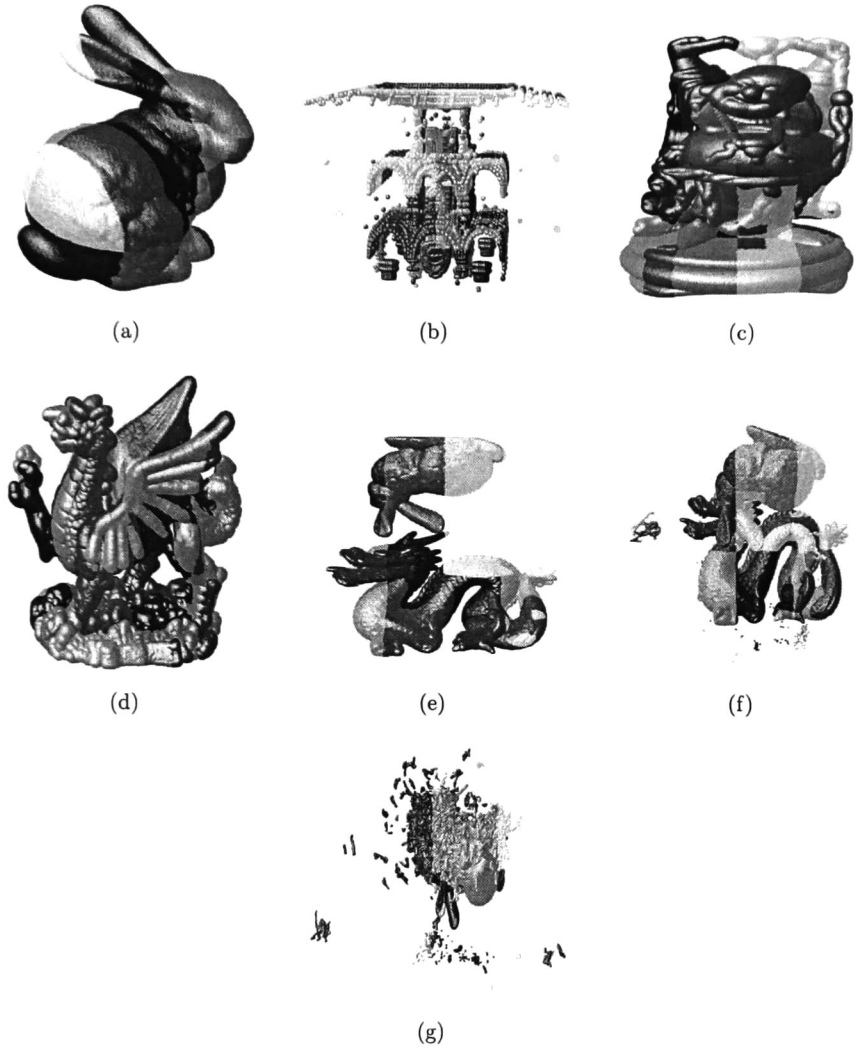


FIGURE 5.1: Primitives are grouped in clusters of equivalence classes; each equivalence class represents a color. a) Stanford Bunny Model (69K). b) Crytek Sponza Model (279K). c) Stanford Budha Model (1M). d) Welsh Dragon Model (2M). e), f) and g) Represents a Dynamic Scene Exploding Dragon and Bunny model (252K).

5.1 Static Scenes

Table 5.1 shows the product of characterization for each scene that the proposed method gives as output.

TABLE 5.1: Best acceleration structure for each scene according characterization.

Scene	Selected Structure
Stanford Bunny(69 K)	BVH
Crytek Sponza(279 K)	Kd-Tree
Happy Buddha(1 M)	BVH
Welsh Dragon(2.2 M)	BVH

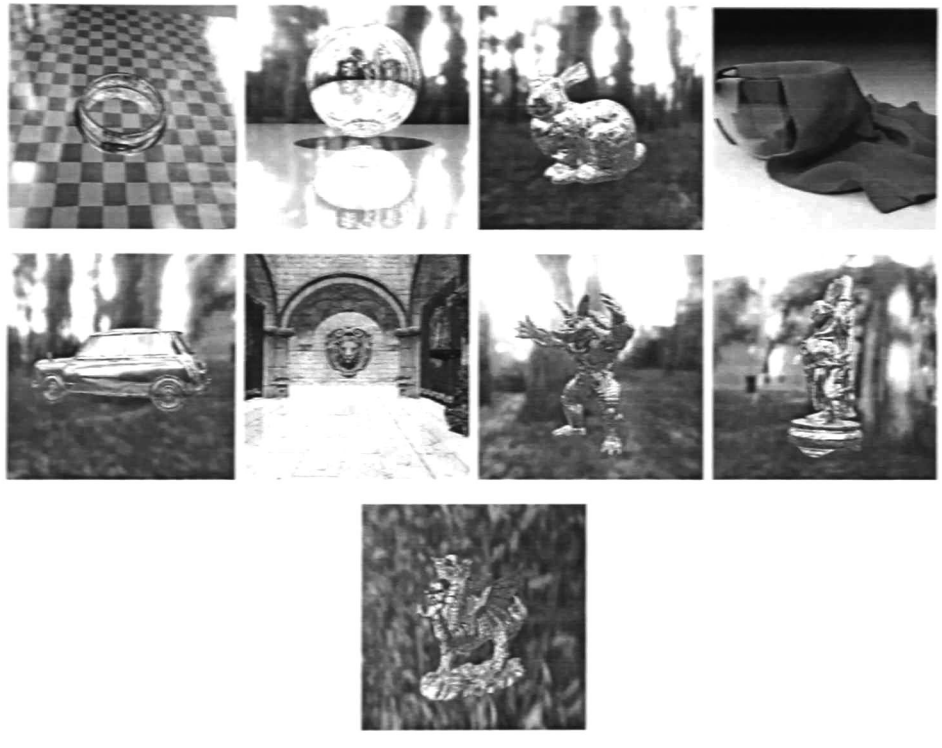


FIGURE 5.2: Rendered frames of static three-dimensional models.

In order to verify if this characterization is assertive, Figures 5.3-5.8 show some metrics to measure quantitatively advantages of one acceleration structure over the others.

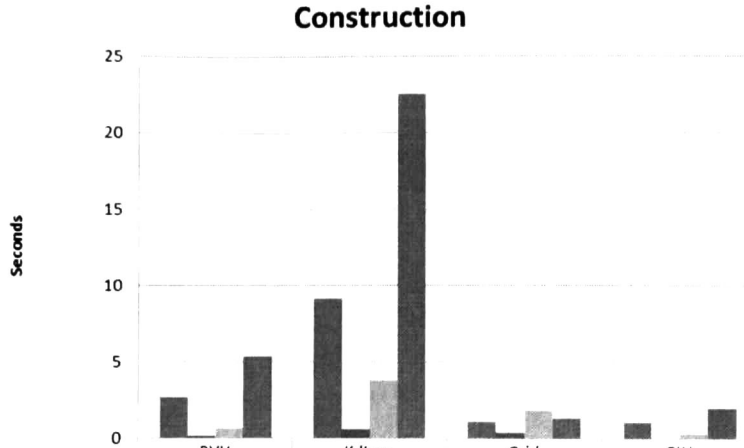


FIGURE 5.3: Construction times in seconds for diverse static scenes.

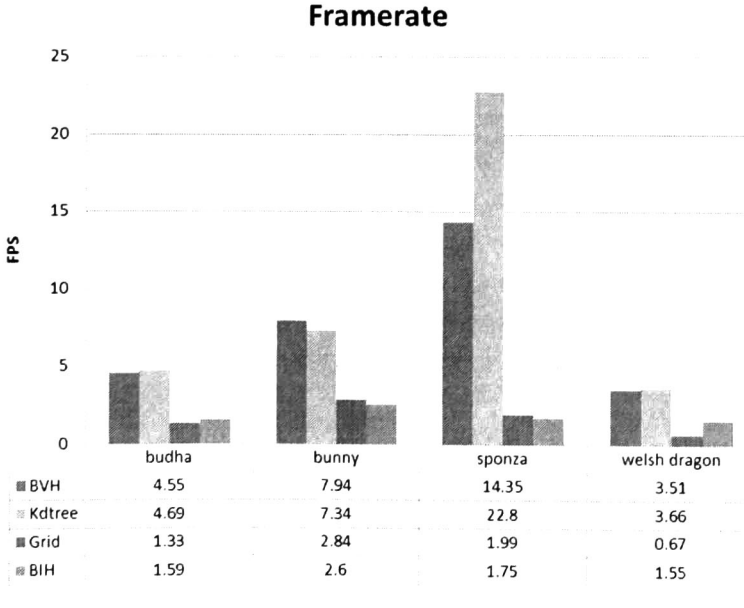


FIGURE 5.4: Frame rate in frames per second for diverse static scenes.

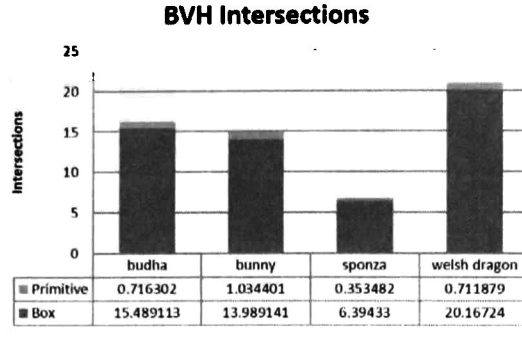


FIGURE 5.5: Intersection tests for BVH with diverse static scenes.

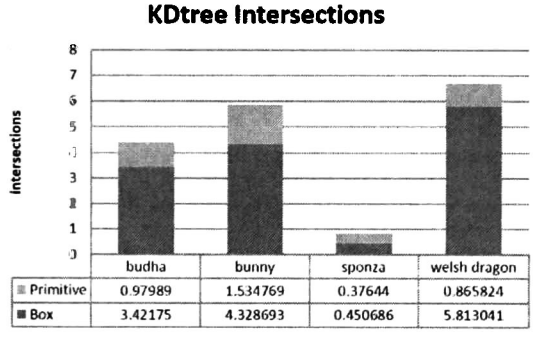


FIGURE 5.6: Intersection tests for KD-Trees with diverse static scenes.

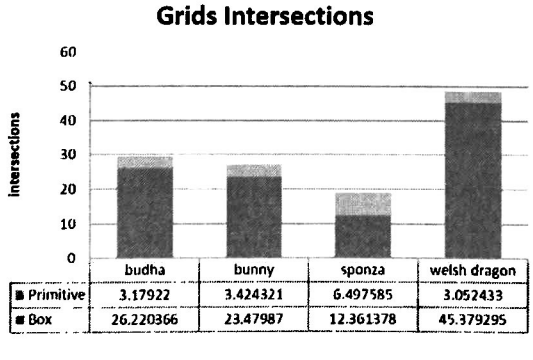


FIGURE 5.7: Intersection tests for grids with diverse static scenes.

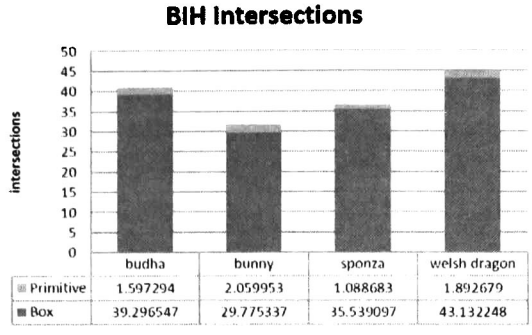


FIGURE 5.8: Intersection tests for BIH with diverse static scenes

As this scenes are static, the most important thing is the ray traversal performance, since a ray tracer builds only once the structure, expensive constructions do not impact in every frame, and the time spent in construction amortizes over the frames. Based on results of metrics, it is important to observe that characterization takes good approximations of the best suited structure for a scene by selecting the structure that in most of the cases has lower construction times and good ray traversal performance.

5.2 Dynamic Scenes

The exploding dragon and bunny scene is a dynamic scene that has 16 raw sequences. Nevertheless in order to show succinct results in this paper, Only three representative sequences of the scene were analyzed. Information of geometry of the scene changes from frame to frame and the consequent performance of acceleration structures changes as te geometry varies. Table 5.2 shows the product of characterization for each scene that the proposed method gives as output.

TABLE 5.2: Best acceleration structure for Bunny and Exploding Dragon (252K) according characterization.

Scene	Selected Structure
Frame 1(252 K)	BVH
Frame 3(252 K)	BVH
Frame 7(252 K)	BVH

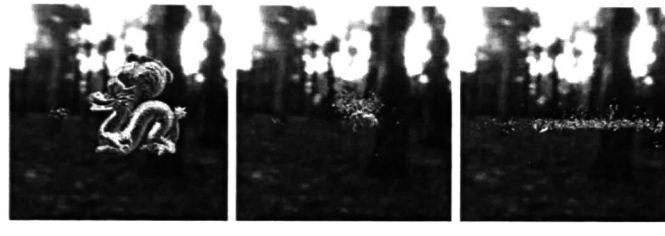


FIGURE 5.9: Rendered frames of a dynamic three-dimensional model.

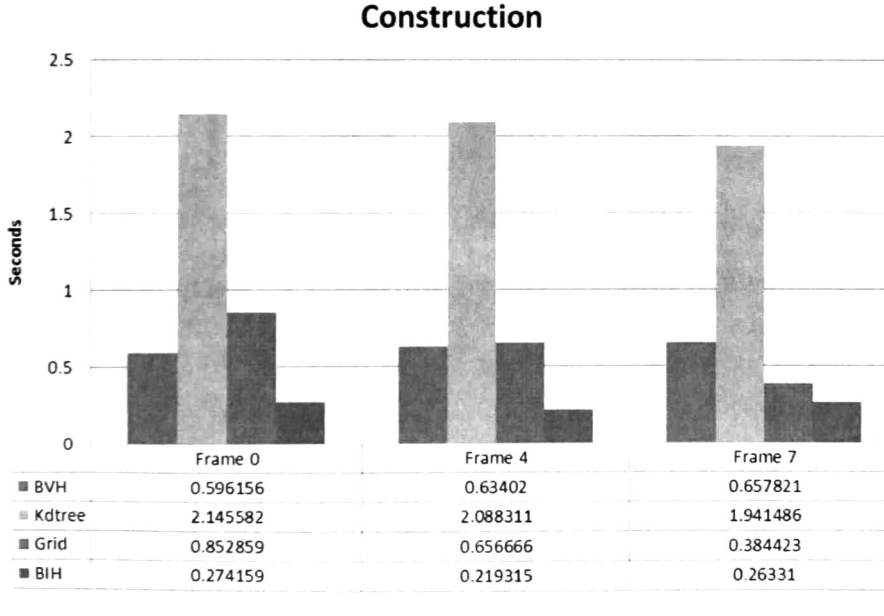


FIGURE 5.10: Construction times in seconds for diverse dynamic scenes

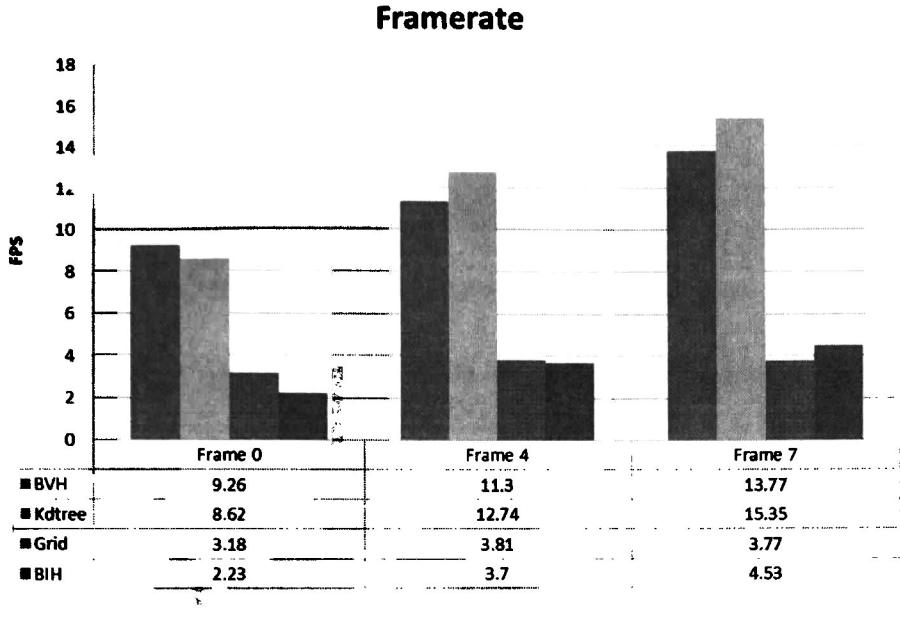


FIGURE 5.11: Frame rate in frames per second for diverse dynamic scenes

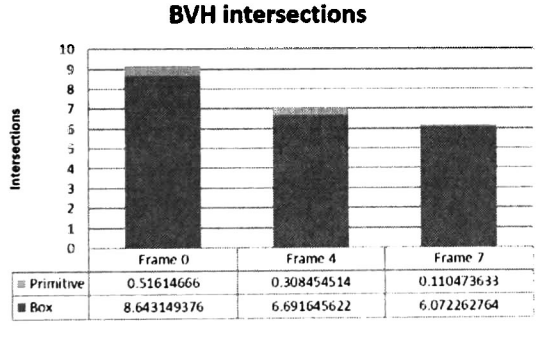
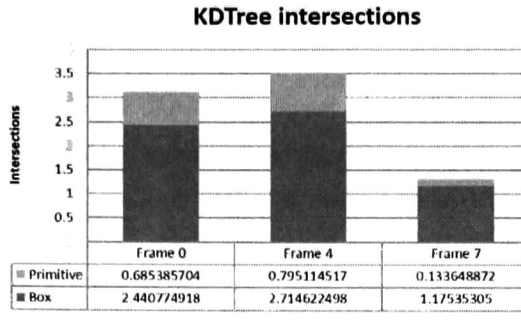
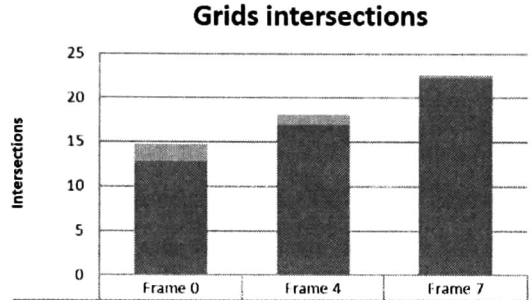


FIGURE 5.12: Intersection tests for BVH with diverse dynamic scenes.



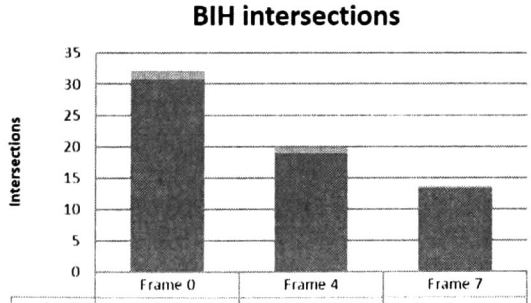
	Frame 0	Frame 4	Frame 7
Primitive	0.685385704	0.795114517	0.133648872
Box	2.440774918	2.714622498	1.17535305

FIGURE 5.13: Intersection tests for KD-Trees with diverse dynamic scenes.



	Frame 0	Frame 4	Frame 7
Primitive	1.916538239	1.222675323	0.315746307
Box	12.81358433	16.931777	22.31864643

FIGURE 5.14: Intersection tests for grids with diverse dynamic scenes.



	Frame 0	Frame 4	Frame 7
Primitive	1.343489647	0.823866844	0.250542641
Box	30.78947926	18.9542942	13.3561182

FIGURE 5.15: Intersection tests for BIH with diverse dynamic scenes

In this kind of scenes where geometry changes constantly, the most important is to build or update the structure as fast as possible for this reason, during the characterization this an important aspect to take in account. Metrics are important to realize how an acceleration structure behaves when the geometry changes. Results of the characterization are consistent with data obtained on the characterization. An in this way is demonstrate that the characterization was assertive in all the cases.

5.3 Performance of the proposed algorithm

Overall Complexity of the proposed algorithm is linear $O(n)$. Moreover, the total amount of work performed by the algorithm is fully parallelized. Table 5.3 shows execution times of the proposed algorithm, in this table can be observed a Speed-up factor from 18 to 49 X in characterization time. Figure 5.16 shows a linear behavior in execution times with scenes from 6K to 2.2M number of primitives. Finally, Figure 5.16 show a comparative between the naive method of characteriza a scene and the current proposal.

TABLE 5.3: Execution times of the proposed algorithm.

Scene	Number of Primitives	Naive Implementation	Current Proposal	Speed-Up Factor
Ring	6,146	0.037 sec	0.002 sec	18X
Diamond	20,482	0.097 sec	0.004 sec	24X
Stanford Bunny	69,451	0.446 sec	0.009 sec	49X
Crytek Sponza	279,163	1.346 sec	0.033 sec	40X
Minicooper	304,197	1.642 sec	0.035 sec	46X
Demon	935,236	4.786 sec	0.107 sec	44X
Happy Buddha	1,087,716	5.642 sec	0.126 sec	44X
Welsh Dragon	2,210,673	11.179 sec	0.256 sec	43X

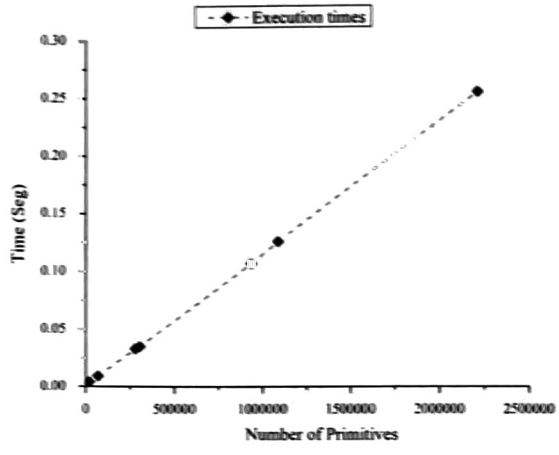


FIGURE 5.16: Linear behavior of execution times of the proposed algorithm.

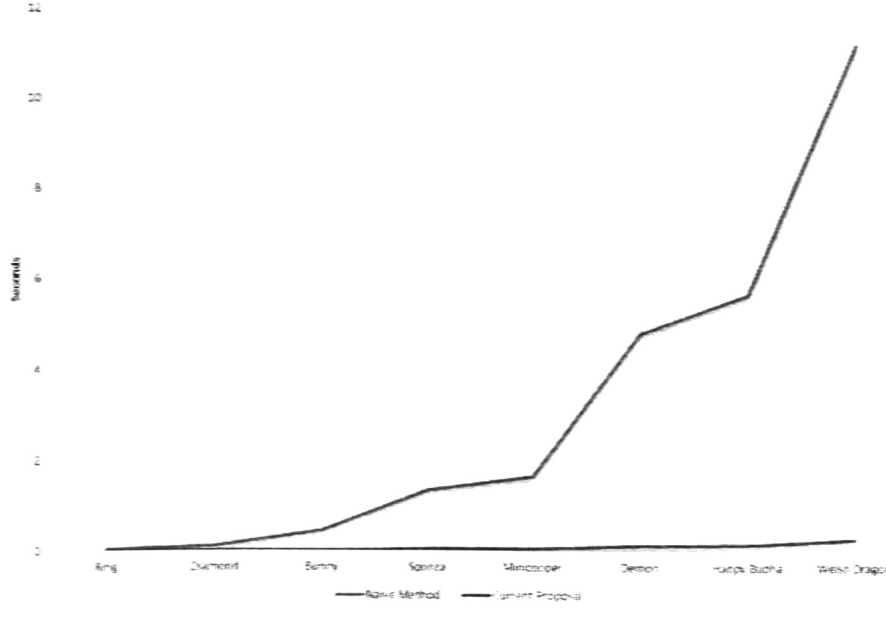


FIGURE 5.17: **Naive method vs current implementation.**

Chapter 6

Conclusions and Future Work

This chapter summarizes the thesis, discusses its findings and contributions, points out some contributions to the area with the current work, and also outlines direction for future research in the area.

The work described in the entire thesis has been concerned with the development of a new method to extract geometric complexity of scenes. This new method consisting of statistical analysis was proposed to explode a priori knowledge of geometry in scenes. A number of interesting features of the proposed algorithm have been described, and the method was shown to be efficient on dynamic scenes.

6.1 Summary of the Thesis

A general introduction of the challenges of Ray Tracing was first presented in Chapter 1. With Ray Tracing becoming one of the most implemented rendering techniques, the problem of rendering complex three-dimensional models in real-time has become important. This chapter also presented the general direction of this thesis and pointed the initial hypotheses, main objectives and contributions.

A review of common acceleration structures and architecture improvements for Ray Tracing was given in Chapter 2. This chapter also reviewed bounding volume

techniques. This kind of techniques acts as early exit tests; if a ray does not intersect a bounding volume neither intersects a primitive contained in a bounding volume. Generally, the acceleration structures can be divided into two main categories as a) space subdivision structures, and b)object hierarchy structures. The first kind of structures subdivides the scene spatially using axis while the second technique subdivides primitives of the scene. There exist three main structures types: Bounding Volume hierarchies, Kd-Trees and Grids. In this chapter were discussed the most relevant optimization for each structure and finally, this chapter review some optimizations related to hardware architectures improvements.

Chapter 3 shows a method to characterize scenes. In this chapter is divided into three parts. The first part presents an optimal representation of primitives, the second part describes the construction of an auxiliary structure and finally, the third part describes which patterns are important to select a structure and the characterization process.

A parallel implementation of the proposed method is described in Chapter 4, This chapter is divided into three parts. The first part is related to parallel construction of an auxiliary structure (histogram), this section describes parallel calculation of centroids and Morton Codes and the process to map a Morton code to a histogram position. The second part describes the parallel characterization, and finally the third part describes the generation of equivalence classes for the model.

Chapter 5 showed obtained results. In order to get these results were used two varieties of scenes, static and dynamic scenes. This chapter shows the selection of the proposed method and additionally, shows some metrics for each scene as construction times, frame-rate and intersections per box and primitive.

6.2 Thesis Conclusions

This dissertation has investigated how to characterize surface meshes of three-dimensional models using statistical methods. This information is used as a priori

knowledge of geometric complexity of the scene to evaluate which acceleration structure is the best suited for a specific scene. It can be concluded that this characterization is pretty useful in dynamic scenes where an acceleration structure must be rebuilt from scratch (in most of the cases) every time the geometry in the scene changes. Then, if for each frame is selected a lower bound, It is translated to decrease construction times and increase ray traversal performance and consequently in a higher frame-rate.

This research proposed an efficient method to extract patterns (characteristics) from three-dimensional models. It also presents an efficient parallel process to complete characterization in the minimum amount of time. It is pretty useful for ray tracing in real time, as performance of acceleration structures varies considerably with a diversity of geometry, and the proposed method can save much calculation time and resources by always selecting the most appropriate structure for a given geometry.

This method can be useful, to give information about how to deal with a particular geometry focused to acceleration structures. Also, it can be generalized to segment a scene by coherent primitives (distributed ray tracing).

As a result of this research, it was demonstrated that there exist significant characteristics in a scene that impact directly on construction times (concentration and distribution of primitives in a surface area). It is demonstrated through obtained results that is possible, take advantage of a priori knowledge about the geometric complexity of the scene to select an acceleration structure. Also, It was demonstrated that initial hypotheses were true.

Finally, complexity of this proposed method can be amortized, the auxiliary structure can be considered as a low quality structure that maintain memory coherence on its construction and values of primitives can be relocated or operated on GPU avoiding memory latency.

6.2.1 Publications

The results in this thesis have previously appeared in the following publications:

- U. Olivares, A. García and F. Ramos "Surface Mesh Characterization of Scenes for Ray Tracing in Real-time", proceeding of the Computer Graphics International, 2014 [42]. Status: Accepted
- U. Olivares, A. García and F. Ramos "Surface Mesh Characterization of Scenes on GPU for Ray Tracing in Real-time", Pacific Graphics 2014 [43]. Status:In review phase.

6.3 Future Work

Although the results presented in this thesis have demonstrated the effectiveness of our approach to characterize three-dimensional scenes, it could be further development in some ways, and there are many avenues for future work to improve the algorithms and the statistical analysis proposed in this thesis.

One direction on future work could be the prediction in ray tracing. This prediction can be possible due primitives are sorted in equivalence classes, and in each class there exist a coherence between primitives. This information about coherence could be useful during intersection tests.

One interesting application would be to investigate allowing automatic learning of the method. Another possible direction is to investigate about distributed ray tracing. As primitives are grouped in equivalence classes, and there exist coherence between primitives of the same class. It can be used a hybrid construction, this construction allows the creation of specialized acceleration structures for each object that has coherence on its primitives. This application can improve performance of complex scenes.

Finally, in terms of application the proposed method can be applicable in a lot of potential areas in computer graphics. Due information provided by the proposed method is useful to know the geometric complexity of the input scene.

Appendix A

Graphic Card Specifications

This appendix shows specifications of the NVIDIA GeForce GTX 780 3GB graphic card. This card was used to obtain the results on chapter 5. Tables A.1 A.3 summarize all specifications for this graphics card.

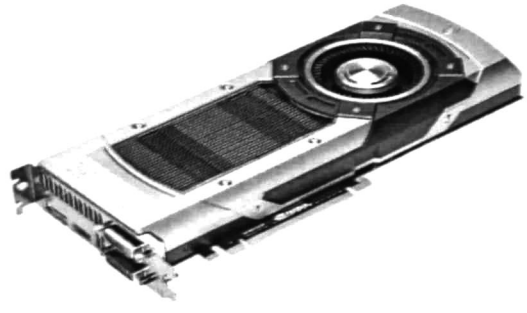


FIGURE A.1: NVIDIA GTX780 graphic card

TABLE A.1: GPU Engine Performance Specifications.

Description	Unit
CUDA CORES	2043
Base Clock(MHz)	976
Boost Clock(MHz)	976
Texture Fill Rate (MHz)	185.6 GT/s

TABLE A.2: GPU Memory Performance Specifications.

Description	Unit
Memory Speed	6.0 Gbps
Standard memory config	3072MB
Memory interface	GDDR5
Memory interface width	384-bit
Memory bandwidth	188.4

TABLE A.3: GPU Resolution and Refresh Performance Specifications.

Description	Unit
Max Refresh	240Hz
Max Analog	2048 x 1536
Max Digital	4096 x 2160

Bibliography

[1] A. Appel, "Some techniques for shading machine renderings of solids," in *Proceedings of the April 30–May 2, 1968, Spring Joint Computer Conference*, AFIPS '68 (Spring), (New York, NY, USA), pp. 37–45, ACM, 1968.

[2] T. Whitted, "An improved illumination model for shaded display," *Commun. ACM*, vol. 23, pp. 343–349, June 1980.

[3] R. L. Cook, T. Porter, and L. Carpenter, "Distributed ray tracing," *SIGGRAPH Comput. Graph.*, vol. 18, pp. 137–145, Jan. 1984.

[4] I. Wald, W. R. Mark, J. Günther, S. Boulos, T. Ize, W. Hunt, S. G. Parker, and P. Shirley, "State of the art in ray tracing animated scenes," in *STAR Proceedings of Eurographics 2007* (D. Schmalstieg and J. Bittner, eds.), pp. 89–116, The Eurographics Association, Sept. 2007.

[5] J. Arvo and D. Kirk, "An introduction to ray tracing," ch. A Survey of Ray Tracing Acceleration Techniques, pp. 201–262, London, UK, UK: Academic Press Ltd., 1989.

[6] I. Wald, "On fast construction of sah-based bounding volume hierarchies," in *Proceedings of the 2007 IEEE Symposium on Interactive Ray Tracing*, RT '07, (Washington, DC, USA), pp. 33–40, IEEE Computer Society, 2007.

[7] C. Ericson, *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3-D Technology) (The Morgan Kaufmann Series in Interactive 3D Technology)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[8] G. van den Bergen, G. Van, and D. Bergen, "Efficient collision detection of complex deformable models using aabb trees," *J. Graphics Tools*, vol. 2, 1998.

[9] I. Wald, "Fast construction of sah bvhs on the intel many integrated core (mic) architecture," *IEEE Transactions on Visualization and Computer Graphics*, vol. 18, pp. 47–57, Jan. 2012.

[10] A. García, F. Ávila, S. Murguía, and L. Reyes, *Interactive Ray Tracing Using the Compute Shader in DirectX 11*, pp. 353–376. A K Peters, 2012.

[11] W. Hunt and W. R. Mark, "Ray-specialized acceleration structures for ray tracing," in *Interactive Ray Tracing, 2008. RT 2008. IEEE Symposium on*, pp. 3–10, IEEE, 2008.

[12] D. J. MacDonald and K. S. Booth, "Heuristics for ray tracing using space subdivision,' *Vis. Comput.*, vol. 6, pp. 153–166, May 1990.

[13] V. Havran, "About the relation between spatial subdivision and object hierarchies used in ray tracing," in *In Proceedings of the Spring Conference on Computer Graphics (SCCG*, 2007.

[14] M. Shevtsov, A. Soupikov, and A. Kapustin, "Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes.,' *Comput. Graph. Forum*, vol. 26, no. 3, pp. 395–404, 2007.

[15] J. Günther, H. Friedrich, I. Wald, H.-P. Seidel, and P. Slusallek, "Ray tracing animated scenes using motion decomposition," *Computer Graphics Forum*, vol. 25, pp. 517–525, Sept. 2006. (Proceedings of Eurographics).

[16] G. Stoll, "Ray tracing performance," *Annual Conference on Computer Graphics and Interactive Techniques SIGRAPH 2005*, 2005.

[17] P. Heckbert, "Color image quantization for frame buffer display," in *Proceedings of the 9th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '82, (New York, NY, USA), pp. 297–307, ACM, 1982.

[18] C. Wächter and A. Keller, "Instant ray tracing: The bounding interval hierarchy," in *In Rendering Techniques 2006 - Proceedings of the 17th Eurographics Symposium on Rendering*, pp. 139–149, 2006.

[19] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, pp. 209–226, Sept. 1977.

[20] R. Ladner, "Data structures and algorithms, kd-trees." http://courses.cs.washington.edu/courses/cse373/02au/lectures/lecture22l.pdf, Sept. 2002.

[21] W. Hunt, W. R. Mark, and D. Fussell, "Fast and lazy build of acceleration structures from scene hierarchies," in *IEEE/EG Symposium on Interactive Ray Tracing 2007*, pp. 47–54, IEEE/EG, Sep 2007.

[22] W. Hunt, W. Mark, and G. Stoll, "Fast kd-tree construction with an adaptive error-bounded heuristic," *Symposium on Interactive Ray Tracing*, vol. 0, pp. 81–88, 2006.

[23] S. Popov, J. Günther, H.-P. Seidel, and P. Slusallek, "Experiences with streaming construction of SAH KD-trees," in *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pp. 89–94, Sept. 2006.

[24] M. Teschner, S. Kimmerle, G. Zachmann, B. Heidelberger, L. Raghupathi, A. Fuhrmann, M.-P. Cani, F. Faure, N. Magnetat-Thalmann, and W. Strasser, "Collision detection for deformable objects," 2004.

[25] T. Larsson and T. Akenine-Mö ller, "A dynamic bounding volume hierarchy for generalized collision detection,' in *Proceedings of the 2nd Workshop on Virtual Reality Interactions and Physical Simulations*, pp. 91–100, 2005.

[26] S.-E. Yoon, S. Curtis, and D. Manocha, "Ray tracing dynamic scenes using selective restructuring," in *ACM SIGGRAPH 2007 Sketches*, SIGGRAPH '07, (New York, NY, USA), ACM, 2007.

[27] J. Goldsmith and J. Salmon, "Automatic creation of object hierarchies for ray tracing," *IEEE Comput. Graph. Appl.*, vol. 7, pp. 14–20, May 1987.

[28] T. Karras, "Thinking parallel, part ii: Tree traversal on the gpu." http://devblogs.nvidia.com/parallelforall/thinking-parallel-part-ii-tree-traversal-gpu/, 2012.

[29] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha, "Fast bvh construction on gpus," in *In Proceedings of the EUROGRAPHICS '09*, 2009.

[30] I. Wald and V. Havran, "On building fast kd-trees for ray tracing, and on doing that in o(n log n)," in *In Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pp. 61–70, 2006.

[31] J. Pantaleoni and D. Luebke, "Hlbvh: Hierarchical lbvh construction for real-time ray tracing of dynamic geometry," in *Proceedings of the Conference on High Performance Graphics*, HPG '10, (Aire-la-Ville, Switzerland, Switzerland), pp. 87–95, Eurographics Association, 2010.

[32] K. Garanzha, J. Pantaleoni, and D. McAllister, "Simpler and faster hlbvh with work queues," in *Proceedings of the ACM SIGGRAPH Symposium on High Performance Graphics*, HPG '11, (New York, NY, USA), pp. 59–64, ACM, 2011.

[33] D. Kirk and J. Arvo, "Improved ray tagging for voxel-based ray tracing," in *Graphics Gems II* (J. Arvo, ed.), pp. 264–266, Academic Press, 1991.

[34] T. Ize, I. Wald, C. Robertson, and S. G. Parker, "An Evaluation of Parallel Grid Construction for Ray Tracing Dynamic Scenes," in *Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, pp. 27–55, 2006.

[35] I. Wald, T. Ize, A. Kensler, A. Knoll, and S. G. Parker, "Ray tracing animated scenes using coherent grid traversal," in *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, (New York, NY, USA), pp. 485–493, ACM, 2006.

[36] J.-H. Nah, J.-S. Park, C. Park, J.-W. Kim, Y.-H. Jung, W.-C. Park, and T.-D. Han, "T&ıengine: Traversal and intersection engine for hardware accelerated ray tracing," in *Proceedings of the 2011 SIGGRAPH Asia Conference*, SA '11, (New York, NY, USA), pp. 160:1–160:10, ACM, 2011.

[37] M. J. Doyle, C. Fowler, and M. Manzke, "A hardware unit for fast sah-optimised bvh construction," *ACM Trans. Graph.*, vol. 32, pp. 139:1–139:10, July 2013.

[38] Morton, "A computer oriented geodetic data base and a new technique in file sequencing," Tech. Rep. Ottawa, Ontario, Canada, 1966.

[39] D. P. Doane, "Aesthetic Frequency Classifications," *American Statistician*, vol. 30, no. 4, pp. 181–183, 1976.

[40] H. A. Sturges, "The choice of a class interval," *American Statistical Association*, vol. 21, pp. 65–66, 1926.

[41] W.-m. W. Hwu, *GPU Computing Gems Jade Edition*, ch. A Prodictivity–Oriented Library for CUDA, Chapter 26. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1st ed., 2011.

[42] U. Olivares, A. García, and F. Ramos, "Surface mesh characterization of scenes for ray tracing in real-time," in *Proceedings of the Conputer Graphics International Conference*, IEEE, 2014.

[43] U. Olivares, A. García, and F. Ramos, "Surface mesh characterization of scenes on gpu for ray tracing in real-time," in *Proceedings of the Pacific Graphics Conference*, IEEE, 2014.

CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL I.P.N.
UNIDAD GUADALAJARA

El Jurado designado por la Unidad Guadalajara del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó la tesis

Caracterización de mallas de superficie de escenas dinámicas para Ray Tracing en tiempo real

del (la) C.

Ulises OLIVARES PINTO

el día 08 de Agosto de 2014.

Dr. Luis Ernesto López Mellado
Investigador CINVESTAV 3C
CINVESTAV Unidad Guadalajara

Dr. Juan Manuel Ramírez Arredondo
Investigador CINVESTAV 3C
CINVESTAV Unidad Guadalajara

Dr. Félix Francisco Ramos Corchado
Investigador CINVESTAV 3B
CINVESTAV Unidad Guadalajara

Dr. Mario Angel Siller González Pico
Investigador CINVESTAV 3A
CINVESTAV Unidad Guadalajara

Dr. Marco Antonio Ramos Corchado
Profesor Investigador Nivel E
Universidad Autónoma del Estado de México