DISSERTATION THESIS

Heuristic Ray Shooting Algorithms

by

Vlastimil Havran

Submitted to
the Faculty of Electrical Engineering, Czech Technical University, Prague,
in partial fulfillment of the requirements for the degree of Doctor.

November 2000
Prague

Revision 1.1
8 May 2001

This minor revision follows the defense of this Ph.D. thesis on 20 April 2001 in Prague and reflects the remarks and comments of oponents and other people involved. I am indebted for comments to oponents Assoc. Prof. Eduard Gröller from Vienna University of Technology, Assoc. Prof. László Szirmay-Kalos from Technical University in Budapest, and Prof. Jaroslav Pokorný from Charles University in Prague. Other comments and remarks were kindly supported by Mateu Sbert and Piero Foscari. Spanish abstract was kindly provided by Mateu Sbert.

Revision 1.0
30 November 2000

Submitted to the Faculty of Electrical Engineering, Czech Technical University, Prague, in partial fulfillment of the requirements for the degree of Doctor.

Abstract

Global illumination research aiming at the photo-realistic image synthesis pushes forward research in computer graphics as a whole. The computation of visually plausible images is time-consuming and far from being realtime at present. A significant part of computation in global illumination algorithms involves repetitive computing of visibility queries.

In the thesis, we describe our results in ray shooting, which is a well-known problem in the field of visibility. The problem is difficult in spite of its simple definition: For a given oriented half-line and a set of objects, find out the first object intersected by the half-line if such an object exists. A naïve algorithm has the time complexity $O(N)$, where N is the number of objects. The naïve algorithm is practically inapplicable in global illumination applications for a scene with a high number of objects, due its huge time requirements. In this thesis we deal with heuristic ray shooting algorithms that use additional spatial data structures. We put stress on average-case complexity and we particularly investigate the ray shooting algorithms based on spatial hierarchies. In the thesis we deal with two major topics.

In the first part of the thesis, we introduce a ray shooting computation model and performance model. Based on these two models we develop a methodology for comparing various ray shooting algorithms for a set of experiments performed on a set of scenes. Consecutively, we compare common heuristic ray shooting algorithms based on BSP trees, *kd*-trees, octrees, bounding volume hierarchies, uniform grids, and three types of hierarchical grids using a set of 30 scenes from Standard Procedural Database. We show that for this set of scenes the ray shooting algorithms based on the *kd*-tree is the winning candidate among all tested ray shooting algorithms.

The second and major part of the thesis presents several techniques for decreasing the time and space complexity for ray shooting algorithms based on *kd*-tree. We deal with both *kd*-tree construction and ray traversal algorithms. In the context of *kd*-tree construction, we present new methods for adaptive construction of the *kd*-tree using empty spatial regions in the scene, termination criteria, general cost model for the *kd*-tree, and modified surface area heuristics for a restricted set of rays. Further, we describe a new version of the recursive ray traversal algorithm. In context of the recursive ray traversal algorithm based on the *kd*-tree, we develop the concept of the largest common traversal sequence. This reduces the number of hierarchical traversal steps in the *kd*-tree for certain ray sets. We also describe one technique closely related to computer architecture, namely mapping *kd*-tree nodes to memory to increase the cache hit ratio for processors with a large cache line. Most of the techniques proposed in the thesis can be used in combination. In practice, the average time complexity of the ray shooting algorithms based on the *kd*-tree, as presented in this thesis, is about $O(\log N)$, where the hidden multiplicative factor depends on the input data. However, at present it is not known to have been proved theoretically for scenes with general distribution of objects. For these reasons our findings are supported by a set of experiments for the above-mentioned set of 30 scenes.

Resumen

La investigación en iluminación global con objetivo la síntesis de imágenes realistas hace avanzar la investigación en informática gráfica en su conjunto. El cálculo de imágenes visualmente plausibles es costoso y está lejos por el momento de ser en tiempo real. Una parte significativa del cálculo en los algoritmos de iluminación global incluye la computación repetitiva de consultas de visibilidad.

En la tesis describimos nuestros resultados en trazado de rayos, problema bien conocido en el campo de la visibilidad. El problema es difícil a pesar de su simple definición: Para una semilínea orientada y un conjunto de objetos, hallar el primer objeto intersectado por la semilínea, suponiendo que tal objeto exista. Un algoritmo naïve tiene complejidad temporal $O(N)$, donde N es el número de objetos. El algoritmo naïve es prácticamente inaplicable en aplicaciones de iluminación global para una escena con un gran número de objetos, debido a su enorme requerimiento en tiempo. En esta tesis tratamos con algoritmos de trazado de rayos heurísticos que usan estructuras de datos espaciales. Resaltaremos el caso de complejidad media y en particular investigaremos algoritmos de trazado de rayos basados en jerarquías espaciales. En la tesis tratamos principalmente con dos tópicos.

En la primera parte de la tesis, introducimos un modelo computacional de cálculo de trazado de rayos y un modelo de rendimiento. Basándonos en estos dos modelos desarrollamos una metodología para comparar distintos algoritmos de trazado de rayos para un conjunto de experimentos realizados sobre un conjunto de escenas. Comparamos consecutivamente algoritmos comunes de trazado de rayos basados en árboles BSP, árboles kd, árboles octales, jerarquías de volúmenes englobantes, mallas uniformes y tres tipos de mallas jerárquicas usando un conjunto de 30 escenas de la Standard Procedural Database. Mostramos que para este conjunto de escenas el árbol kd es el candidato ganador entre todos los algoritmos de trazado de rayos probados.

La segunda y más extensa parte de la tesis presenta varias técnicas para disminuir la complejidad espacial y temporal en los algoritmos de trazado de rayos basados en árboles kd. Tratamos con los algoritmos de construcción del árbol kd y de recorrido del rayo. En el contexto de la construcción del árbol kd, presentamos nuevos métodos para su construcción adaptativa usando regiones espaciales vacías de la escena, criterios de terminación, coste general del modelo para el árbol kd, y heurísticas de área de la superfície modificadas para un conjunto restringido de rayos. Además, describimos una nueva versión del algoritmo recursivo de recorrido del rayo. En el contexto del algoritmo recursivo de recorrido del rayo basado en el árbol kd, desarrollamos el concepto de la sucesión más larga de recorrido común. Esto reduce el número de pasos jerárquicos de recorrido transversal en el árbol kd para ciertos conjuntos de rayos. Describimos también una técnica relacionada íntimamente con la arquitectura del computador, a saber el mapeo de los nodos kd a memoria para incrementar el hit-ratio de la cache para procesadores con una línea grande de cache. La mayoría de las técnicas propuestas en la tesis se pueden utilizar en combinación. En la práctica, la complejidad temporal media de los algoritmos de trazado de rayos basados en el árbol kd, como se presenta en esta tesis, es aproximadamente $O(\log N)$, donde la constante multiplicativa depende de los datos de entrada. Sin embargo, por el momento no se conoce que se haya probado para escenas con distribución general de objetos. Por estas razones nuestros hallazgos son sustentados por un conjunto de experimentos para el conjunto de 30 escenas mencionado más arriba.

Resumé

Výzkum v oblasti algoritmů pro fotorealistickou syntézu obrazu udává směr výzkumu v oblasti celé počítačové grafiky. Výpočet obrázků, které lze těžko rozpoznat od reality, je velmi časově náročný a v současné době nerealizovatelný v reálném čase bez speciálních a nákladných technických prostředků. Významná část výpočtů algoritmů syntézy obrazu je tvořena výpočtem dotazů na viditelnost.

V disertační práci prezentujeme naše výsledky týkajících se algoritmů *vrhání paprsku* jako velmi často řešeného problému viditelnosti. Problém vrhání paprsku je zadán takto: pro zadanou polopřímku a množinu objektů najdi první objekt, který tato polopřímka protíná, pokud takový objekt existuje. Navzdory jednoduchosti formulace tohoto problému je algoritmus pro jeho efektivní výpočet netriviální. Takzvaný *triviální* algoritmus má pro N objektů ve scéně lineární časovou složitost. Pro scény, které mají velký počet objektů je tento triviální algoritmus nevhodný vzhledem k jeho neúnosným časovým nárokům. V disertační práci se zabýváme heuristickými algoritmy vrhání paprsku, které využívají pomocných prostorových datových struktur se zaměřením na průměrnou časovou složitost. Detailně se pak zabýváme algoritmy, které využívají hierarchických datových struktur. V disertační práci zpracováváme dvě hlavní témata.

V první části disertační práce popisujeme nový model výpočtu a výkonosti pro algoritmy vrhání paprsku. S pomocí těchto dvou modelů zavádíme metodologii pro porovnávání různých algoritmů vrhání paprsku pro množinu experimentů provedenou na množině testovacích scén. Poté porovnáváme dvanáct odlišných algoritmů vrhání paprsku, které využívají datové struktury binárního strom, kd-stromu, oktalového stromu, hierarchie obálek, uniformní mřížky a třech typů hierarchických mřížek a to na množině třiceti scén ze *Standard Procedural Database.* Ukazujeme, že v průměru nejrychlejší algoritmus vrhání paprsku ze všech testovaných algoritmů je ten, který využívá kd-stromu.

Druhá a obsáhlejší část disertační práce se zabývá technikami pro zmenšení časové a paměťové náročnosti algoritmů vrhání paprsku využívajících kd-stromu. Zabýváme se jak vlastní konstrukcí kd-stromu tak i algoritmy pro jeho traverzaci pro zadaný vstupní paprsek. Z algoritmů pro konstrukci kd-stromu popisujeme nové metody s využitím volných prostor ve scéně, kritéria pro ukončení stavby kd-stromu, obecný cenový model pro stavbu kd-stromu a modifikovaný algoritmus pro stavbu kd-stromu vhodný z hlediska časové složitosti pro specifické množiny paprsků. Dále popisujeme nový rekurzivní algoritmus pro traverzaci kd-stromu. Další algoritmy týkající se kd-stromu zahrnují koncept a využití nejdelší společné sekvence listů či vnitřních uzlů kd-stromu v traverzačním algoritmu pro speciální množiny paprsků, což nám umožňuje dále snížit počet hierarchických traverzačních kroků. Rovněž popisujeme novou techniku pro mapování uzlů kd-stromu do hlavní paměti počítače s dlouhou řádkou vyrovnávací paměti, která zvýšuje datovou koherenci při procházení kd-stromu. Většinu algoritmických technik popsaných v disertační práci je možné vhodně kombinovat. Z praktického hlediska dosahuje průměrná časová složitost algoritmů vrhání paprsku s využitím kd-stromu $O(\log N)$ s tím, že multiplikativní faktor asymptotické složitosti závisí na vstupních datech. Nicméně v současné době není znám teoretický důkaz týkající se této složitosti pro scény s libovolnou distribucí objektů a proto jsou výsledky pro všechny popisované algoritmy ověřeny experimentálně na již uvedené množině třiceti testovacích scén.

Preface

This doctoral thesis presents the research conducted by the author at the Department of Computer Science and Engineering, Faculty of Electrical Engineering, the Czech Technical University in Prague during the period 1996–1999 and at the IGP company during the period 1999–2000.

My interest in speeding up visibility computations started while working on my Master Thesis, which dealt with the simulation of the optical phenomena. After graduating in February 1996 from the Czech Technical University in Prague, I became a Ph.D. student with Pavel Slavík as my supervisor.

At the beginning of my Ph.D. studies I devoted my time to parallel solutions for ray tracing, which developed into an interest in a more general problem – ray shooting. I presented my postgraduate study report *Spatial Data Structures for Visibility Computation* in June of 1997, and I continued researching in this direction, specializing in ray shooting algorithms over spatial subdivisions.

This doctoral thesis covers several new methods and improvements for ray shooting algorithms based on spatial subdivisions. A description of basic and previously developed methods is followed by a description of newly developed methods that decrease the time and space complexity for a ray shooting algorithm based on *kd*-trees.

For most of the duration of my Ph.D. studies I was responsible for designing, implementing, and maintaining the *GOLEM* rendering system [75] developed as independent software, in which the presented ray shooting algorithms were implemented and tested. Several other people also contributed to the development of the GOLEM rendering system: Ph.D. students Jiří Bittner and Tomáš Kopal, and Jan Přikryl from the Vienna University of Technology. Several undergraduate students did their Master Theses utilizing or increasing features of the GOLEM rendering system under my supervision: Filip Sixta, Michal Máša, Libor Dachs, Vladimír Nádvorník, Petr Mládek, and Jaroslav Křivánek. Many undergraduate students used GOLEM in their undergraduate term projects, particularly, in the course on *Visualization* given by Assoc. Prof. Jiří Žára.

My research during the period 1996–1999 was partially supported by the following grants:

- CTU Grant No. 30/98101/336: *Adaptive Data Structures for Visibility Computing*, 1998.
- FRVŠ Grant No. 1252/1998: *Solving Visibility in Large Scenes*, which was supervised by Assoc. Prof. Jiří Žára with assistance from Jiří Bittner, 1998.
- AKTION Grant No. 1999/17, joint Czech-Austrian scientific collaboration funding between the Vienna University of Technology and the Czech Technical University in Prague, 1999–2000.

Acknowledgements

First of all, I would like to express my gratitude to my thesis supervisor, Assoc. Prof. Pavel Slavík. He has been a constant source of encouragement during my research. He together with Assoc. Prof. Jiří Žára provided me with numerous opportunities for professional advancements. They both stimulated my research interests in computer graphics, particularly in the early stages of my Ph.D. study. Without their knowledge and help I really could not have progressed. Assoc. Prof. Pavel Tvrdík made important suggestions and remarks during the course of my work that shaped my life as a researcher.

Many other people active in computer graphics influenced my work. I wish to thank Prof. Václav Skala for his impulse for my research during EGWPR'96, held in September 1996 in Bristol. I am grateful to the computer graphics group at Vienna University of Technology for their kindness and for providing me access to research papers that were not available in libraries in the Czech Republic, and especially to Prof. Werner Purgathofer, Assoc. Prof. Eduard Gröller, and Jan Přikryl. I would like to express my appreciation to Assoc. Prof. Jiří Matoušek from Charles University in Prague, who gave me an invaluable insight into the field of computational geometry.

I am much indebted to many people from the computer graphics lab at our department for their comments on my early ideas and research activities, and for keeping life in our lab running. Namely, my thanks belong to Bedřich Beneš, Roman Berka, Pavel Diblík, Petr Felkel, Petr Hejda, Aleš Holeček, Jan Buriánek, and to Jan Vorlíček and Martin Brachtl for maintaining the UNIX systems at the graphics lab. I would like to express my special appreciation to my colleague Jiří Bittner for his encouragement, comments, common interest, and collaboration within the visibility research field.

The staff of our department has provided me a pleasant and flexible environment for my research. Especially, I would like to thank Prof. Bořivoj Melichar and Assoc. Prof. Josef Kolář, both them head of our department, for enabling me to be a Ph.D. student and providing financial support for my research. For financing the research in the last stage of my Ph.D. study, I owe thanks to the IGP company.

Concerning my style of writing, I am much indebted to Robin Healey for English proofreading of the final version of my thesis. Last, but not least, I would also like to thank all my personal friends who kept my social and cultural life enjoyable during my studies. Finally, my greatest thanks to my family, whose support was of real importance during all my studies. I would have never finished this thesis without them.

Dedication

To all people who have positively influenced my life.

Contents

Chapter 1

Introduction

This thesis deals with heuristic ray shooting algorithms, namely with algorithms based on spatial subdivisions, and particularly with ray shooting algorithms based on the *kd*-tree. In this introductory chapter we describe the problem in detail together with algorithmic solutions developed in the past.

1.1 Motivation

The principal goal of computer graphics is image synthesis of a scene simulating a real environment. The algorithms for image synthesis are based on various principles influencing the quality of their outputs. A significant research effort in image synthesis involves generation of *photo-realistic* images. By a photo-realistic image we mean an image indistinguishable from a photograph of a real world. A scene simulating reality is modeled by geometric object primitives in three-dimensional space; it is not exceptional for the number of objects in a scene to reach hundreds of thousands or more.

Historically speaking, two main classes of algorithms for photo-realistic rendering have been developed: *ray-tracing* and *radiosity* [157, 47]. These both classes are used and sometimes combined together. Recently, the importance of classical radiosity algorithms based on computing form factors between patches has diminished with the coming of Monte-Carlo methods in global illumination [20, 149]. The common property of all these rendering algorithms is their possible high time and space complexity. They spend much time repeatedly computing visibility computations, either ray shooting or visibility for a pair of points. Visibility computations performed within image synthesis correspond to the discrete sampling of n-dimensional space.

1.2 Problem Statement

The *visibility for a pair of points* problem is more formally defined as follows: two points $U = (x,y,z)$ and $V = (x',y',z')$ are mutually visible if the line segment $\overline{UV}$ with U and V as endpoints does not intersect any object located in the scene. The result of the algorithm solving the problem is of Boolean type: yes (visible) or no (invisible). The computation of the visibility for a pair of points is indispensable to determine correctly the illumination and shading of objects by light sources.

The *ray shooting* problem is a more general visibility problem along a fixed line: For an oriented half-line given by its origin and direction vector, we want to find the closest object intersected by the half-line if such an object exists. The ray shooting problem is depicted in Fig. 1.1.

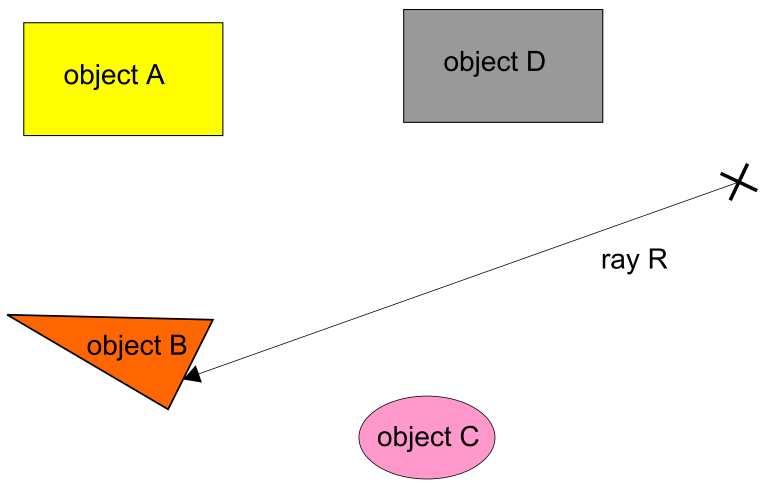


Figure 1.1: Ray shooting in $\mathbb{E}^2$ space. The answer for ray R is object B.

1.3 Basic Terminology

In this section we describe the basic terminology used within the thesis. The geometry handled in the thesis uses n-dimensional Euclidean space, further abbreviated as $\mathbb{E}^n$.

The basic geometric primitive used here is a ray. The *ray* R in $\mathbb{E}^n$ space is an oriented half-line determined by the *point of origin* O^{R} and *direction vector* $\vec{D}^{\mathrm{R}}$. Any point U lying on the ray half-line can be computed using a *parametric representation* of a ray:

$$U = O^{\mathrm{R}} + t.\vec{D}^{\mathrm{R}}, \quad O^{\mathrm{R}},\ \vec{D}^{\mathrm{R}} \in \mathbb{E}^n, \tag{1.1}$$

where parameter t is called *signed distance* ($t \in R,\ t \geq 0$). It is usually assumed that the direction vector $\vec{D}^{\mathrm{R}}$ is normalized ($|\vec{D}^{\mathrm{R}}| = 1$).

An *object* in $\mathbb{E}^n$ is a finite region of $\mathbb{E}^n$ space with continuous $(n-1)$-dimensional boundaries. There are various shapes of objects; spheres, triangles, general polygons, polyhedra, *etc*. A *surface* of the object O is its $(n-1)$-dimensional boundary ∂O, which does not cross itself. A *scene* $\mathcal{S}(N)$ is a set of N objects. We require that both the number of objects and objects themselves are finite. Then we can bound the scene with the finite convex spatial region that contains all the objects.

Given a scene $\mathcal{S}(N)$, we can formulate the pair of points visibility problem more precisely. For two points U and V it is stated as follows: points U and V are *mutually visible* if the line segment $\overline{UV}$ with U and V as endpoints does not intersect any object from $\mathcal{S}(N)$. Contrarily, the two points are *mutually invisible*, *i.e.*, there is at least one object from $\mathcal{S}(N)$ so that the intersection exists between the object and the line segment $\overline{UV}$.

Similarly, we can also more formally describe the *ray shooting* problem. Given a ray R and the scene $\mathcal{S}(N)$, we want to find out the closest object intersected by R if such an object exists. The answer of ray shooting is object O_i, the found intersection point on O_i is also required by most applications. In order to distinguish a ray shooting problem from a particular task given by a specific ray R and scene $\mathcal{S}(N)$, we refer to the latter as *ray shooting query*.

A *ray shooting algorithm* (abbreviated to *RSA* in the thesis) is an algorithm that computes the answer to any ray shooting query for a given scene. Usually, in the *preprocessing phase* an *RSA* builds up an auxiliary *data structure* for a set of scene objects. In the *execution phase* of the *RSA* the data structure is accessed during computation of the answer to a particular ray shooting query.

There is one exceptional *RSA* that does not perform any preprocessing. It is called a *naïve RSA* and it uses only a list of scene objects. When answering a ray shooting query, the naïve *RSA* tests the ray with all the objects and selects the one with the closest intersection found, if such an object exists. The time complexity of the naïve *RSA* is thus $O(N)$. In applications in computer graphics where many rays are shot, this naïve *RSA* is applicable only for scenes with a few objects. For a scene with a high number of objects it makes the application run unacceptably slowly. This property of a naïve *RSA* led to the research in *RSAs*.

Heuristic *RSAs* are based on some data structures that cover the distances between the objects in the scene. Since the data structures store the properties and geometric relationships in a space (usually in $\mathbb{E}^3$ space), they are usually called *spatial data structures*. These spatial data structures for various *RSAs* describe spatial relationships using a set of spatial regions that are called *cells*. Further we use $\mathcal{V}$ as a symbol for an arbitrary cell. The spatial region(s) covered by the cell $\mathcal{V}$ is given by the cell boundary $\partial\mathcal{V}$. If a cell $\mathcal{V}$ separates the space into two disjoint parts, we call it a *separating cell*. Practically, this means that if we denote the parts induced by the cell ρ and τ, then for two points U, V, $U \in \rho$ and $V \in \tau$ must hold that the line segment $\overline{UV}$ must cross the cell boundary $\partial\mathcal{V}$. An example of a separating cell is an infinite plane that induces two halfspaces.

A more common case is when a cell and its boundary is of finite size, in which case we call it *closed* cell. Then the interior part $int(\mathcal{V})$ of the cell $\mathcal{V}$ is finite and is completely separated from the exterior of the cell, denoted $ext(\mathcal{V}^n)$. Closed cells are commonly used in the spatial data structures that underlie a particular *RSA*. For the sake of convenience when we speak about a cell we mean the spatial region covered by the interior part of the cell, while the second meaning is the element of a data structure that represents the spatial region.

A common example of a closed cell is an *axis-aligned bounding box*, which is illustrated in Fig. 1.2. This is a parallelepiped with six faces (in $\mathbb{E}^3$), each two faces are perpendicular to the coordinate axes (the axes of the standard basis of the space). The size of the axis-aligned bounding box is given by two extreme points A and B. A point U belongs to the axis-aligned bounding box, if $U_i \geq A_i$ and $U_i \leq B_i$, assuming $A_i < B_i$ for $i \in \{x, y, z\}$. We further denote by symbol $\mathcal{AB}(X)$ the axis-aligned bounding box that tightly encloses entity X, where X stands for an object or a cell. For short, $\mathcal{AB}$ is also the abbreviation for an axis-aligned bounding box.

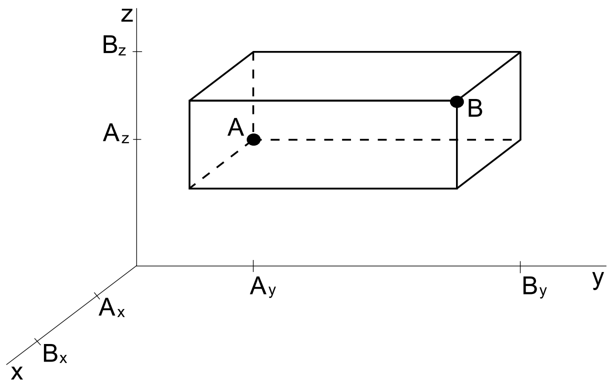


Figure 1.2: Axis-aligned bounding box in $\mathbb{E}^3$.

The cells are organized in spatial data structures. Based on the use of cells in spatial data structures we can distinguish between different types of cells. Further, there can be various relationships between any two cells. Below, we describe some terminology to cover these properties.

We call a closed cell *elementary* if its interior part does not intersect the interior part of any other elementary cell. An elementary cell is intended to contain the list of pointers to the objects that intersect the spatial region covered by the elementary cell. It can also contain other data, for example, the description of the relationship to other cells located in its neighborhood, its size, position, *etc.* If the elementary cell contains at least one pointer to the object intersecting the interior of the cell, we call it a *full elementary cell*. Otherwise, we call it an *empty elementary cell*. Such an empty elementary cell is useful in the sense that the spatial region of the elementary cell contains no object. The emptiness of the spatial region is a piece of information that can be used in a particular *RSA*, since no ray-object intersection can occur within the empty elementary cell.

If a cell is not elementary, we call it a *generic* cell. A generic cell can contain references to other generic cells, elementary cells, or to the objects or even other data required by a particular *RSA*. The cells and objects referenced in the generic cell usually intersect the spatial region covered by the cell. As we show later, generic cells are used to form spatial data structures including the hierarchy.

Having described the types of cells, we can ask about the geometrical relationship between any two cells, and their purpose. We call two cells *neighbors* if the boundary of these two cells has a non-finite intersection. The interiors of these two cells are disjoint. Another spatial relationship between two cells is that if a cell $\mathcal{V}_1$ is *completely contained* in the cell $\mathcal{V}_2$, we denote it $\mathcal{V}_1 \in \mathcal{V}_2$. Different geometric relationship between two cells is when $\mathcal{V}_1$ *partially intersects* $\mathcal{V}_2$, *i.e.*, $\mathcal{V}_1 \cap \mathcal{V}_2 \neq \emptyset$, $\mathcal{V}_1 \notin \mathcal{V}_2$, and $\mathcal{V}_2 \notin \mathcal{V}_1$. The last possible geometric relationship is when the cells are *disjoint*, *i.e.*, $\mathcal{V}_1 \cap \mathcal{V}_2 \equiv \emptyset$.

Given a scene $\mathcal{S}$ with a finite set of objects that are also of finite size, we can construct $\mathcal{AB}(\mathcal{S})$ as the smallest $\mathcal{AB}$ that contains all the objects from $\mathcal{S}$. For the cell $\mathcal{AB}(\mathcal{S})$ we can form various sets of cells inside $\mathcal{AB}(\mathcal{S})$ and then construct corresponding spatial data structures. These spatial data structures can be distinguished by the types of the cells and the properties between the cells.

A *spatial subdivision* (SSD) of a cell $\mathcal{V}_S$ is a finite ordered set S of cells, such that for each point $A \in \mathcal{V}_S$ there exists at least one cell $\mathcal{V}$, $\mathcal{V} \in S$, $A \in \mathcal{V}$.

An *elementary spatial subdivision* (ESSD) of a cell $\mathcal{V}_S$ is a SSD, which is composed of a finite ordered set S_K of closed, disjoint, separating, and elementary cells. Moreover, for each point $A \in \mathcal{V}_S$ there exists exactly one cell $\mathcal{V}$, $\mathcal{V} \in S_K$, $A \in int(\mathcal{V})$ or A lies on at least one cell boundary, $A \in \partial\mathcal{V}$.

A *hierarchical spatial subdivision* (HSSD) of a cell $\mathcal{V}_S$ is a pair of two finite sets S_E and S_H, where S_E is a set of elementary cells and S_H is a nonempty set of generic cells. The cells in S_E and S_H correspond to the nodes of a graph. Moreover, S_E is ESSD and each cell from S_E is pointed to at least in one cell from S_H.

The concept of SSD is the least restrictive and still useful set of cells that can be utilized by an *RSA*. The only one condition is that a point, possibly the point on the ray, can be located in a cell. The concept of ESSD is more restrictive in the sense that it cannot contain any type of hierarchical relationship. On the other hand, HSSD is required to contain the hierarchical relationship. All these spatial data structures can be the base of various *RSAs*, and we have described them here just to point out their differences and commonalities.

Although all the concepts above can easily be extended to lower or higher dimensional space, in this thesis we deal with *RSAs* in $\mathbb{E}^3$ only. This is the most important case for computer graphics. We do not discuss the pair of points visibility problem separately, since it can always be converted to the ray shooting problem: If U and V are the points for testing mutual visibility, we construct the ray with its origin in U and direction vector $\vec{D} = V - U$. If no intersection between the ray and scene objects is found between these two points using any *RSA*, then U and V are mutually visible. Otherwise, they are mutually invisible.

The pair of points visibility problem is less demanding than ray shooting, since it does not require us to compute exactly the point of intersection with the object. The result is of Boolean type and must be always correct – no approximative result is allowed. If the points are not mutually visible, the computation can be accelerated by *caching* the objects that are likely to be intersected on the line segment between these two points. In general, the pair of points visibility problem can thus be less computationally expensive than ray shooting using special techniques including caching of objects [164, 43]. These special techniques are not the subject of this thesis. The speedup achieved by using the special techniques instead of simply converting the pair of points visibility problem to the ray shooting is dependent on the object configuration in the scene and the algorithm used.

However, each *RSA* can be slightly modified for a pair of points visibility problem – when traversing some data structure and checking objects for the intersection with a given ray, we can stop whenever an intersected object lying at some maximum distance is found. Obviously, this modified *RSA* is more efficient for answering pairs of points visibility queries, and in the worst case it is of the same efficiency.

1.4 Related Work/Previous Results

Ray shooting has been studied by the two research communities; from the perspective of computer graphics and computational geometry. In both groups ray shooting and other visibility problems have attained great interest in the last two decades.

1.4.1 Computational Geometry

Computational geometry traditionally addresses the ray shooting problem with the aim of improving worst-case complexity using O-notation, but some attempts have also tried to handle average-case complexity. The research has been particularly active since 1989.

More specifically, in computational geometry the ray shooting problem is understood as a special instance of the range-searching problem [6]. A typical range-searching problem is defined as: Let S be a set of N entities (points, objects) in $\mathbb{E}^d$ and Υ be a family of subsets of $\mathbb{E}^d$, where the elements of Υ are called ranges. The goal of an algorithm solving the range-searching problem is to preprocess S into a data structure so that for query range $\upsilon \in \Upsilon$, the entities in $S \cap \upsilon$ can be reported or counted efficiently. Ray shooting is only one type of geometric range-searching problem.

The ray shooting problem has been solved in $\mathbb{E}^2$ and $\mathbb{E}^3$ using various approaches that reach similar query time/space/preprocessing time complexity. Computational geometry techniques mostly restrict the shape of objects to have certain properties according to a dimension of space; in $\mathbb{E}^2$ to line segments, simple polygons with N edges, N disjoint simple/convex polygons, in $\mathbb{E}^3$ to convex polytopes with N faces, N convex polytopes, terrain description by continuous surface function (height fields), triangles, spheres, and planar polygons. The corresponding *RSA* then utilizes the special properties of a given object class.

Mathematical tools used inside *RSAs* includes the parametrisation of oriented/unoriented lines (two-plane, sphere, and especially Plücker parameterization [166]). The important concepts used by computational geometry cover partition data structures including $(1/r)$-cutting [7], arrangements [8], geodesic triangulation [65], Steiner triangulation [14], *etc.*

Megiddo's *parametric search* technique [8] is often applied; the ray shooting problem is then transformed to a *segment emptiness* problem. Let us assume N objects (preferably polyhedral and convex) in the scene. A line segment in $\mathbb{E}^3$ ($\mathbb{E}^2$) is checked for intersection with objects using some segment-intersection data structure that takes $O(\log N)$ time. The detection is applied using a binary search to find out the first object intersected thus resulting in $O(\log^2 N)$ query time for ray shooting. The parametric search technique has been successfully applied to triangles and spheres in $\mathbb{E}^3$. Usually, one searching data structure is plugged into another searching data structure, and a parametric search is performed. The space and query complexity of the data structures determine the resulting properties of the *RSA*.

There are several best results from which we can trace the tradeoff between query time and space complexity. Here we discuss $\mathbb{E}^3$ space only, which is of major interest for computer graphics. Mark de Berg introduced the *RSA* [22] with $O(\log N)$ time complexity using $O(N^{4+\varepsilon})$ storage and preprocessing time for any $\varepsilon > 0$. His Ph.D. thesis includes special cases such as ray shooting with rays from a fixed point or in a fixed direction in a space of axis-parallel polyhedra, c-oriented polyhedra, arbitrary curtains, and general polyhedra. His main idea is to use Plücker representation of ray space for both the input ray and the representation of objects' edges and then to apply a point location search in this $\mathbb{E}^5$ space. Agarwal and Sharir [9] presented a method that for M possibly intersecting polyhedra with a total of N faces in $O((M.N)^{2+\varepsilon})$ storage and preprocessing time reaches $O(\log^2 N)$ query time. This is a significant improvement over the previous approach given by de Berg, if $M \ll N$. Optionally, they discussed the approach with $O(N^{1+\varepsilon})$ storage and preprocessing time that reaches $O(M^{1/4}.N^{1/2+\varepsilon})$ query time. Mohaban and Sharir [111] presented an algorithm with $O(N^{3+\varepsilon})$ storage and preprocessing time with $O(N^{\varepsilon})$ query time for a set of spheres (or other objects). More recently, Agarwal *et al.* [5] published an algorithm which, for a set of spheres or more general objects, has $O(N^{3+\varepsilon})$ storage and preprocessing time and reaches $O(\log^4 N)$ query time.

There have also been recent attempts to attack the ray shooting problem from the viewpoint of average-case complexity. This includes the concept of C-complexity for query sensitive *RSA* [110]. In this case the complexity of a ray shooting query is considered in relation to the scene complexity along the specific ray path. Under simplifying assumptions for object description using ball covering, a PM *kd*-tree [125], and bounding boxing, it is possible to achieve $O(N.\log N)$ preprocessing time with $O(N)$ storage, when the ray shooting query time corresponds to the C-complexity of this specific ray. Other approaches aimed at average-case complexity use triangulation, especially minimum weight triangulation and Steiner triangulation [14]. The idea hidden behind the techniques is straightforward; assuming arbitrary rays do not hit any object in space, then the rays cross a minimum number of boundaries of decomposition on average. Unfortunately, the complexity of the algorithms that construct these minimum size decompositions even in $\mathbb{E}^2$ belongs to the *NP*-hard class or is unknown and thus the approaches have to utilize some heuristics that can produce results that can be far from optimum.

There are several algorithmic problems closely related to ray shooting; the existence of stabbing for a given ray and set of objects, the existence of a stabbing line for a given set of objects, a moving line segment among obstacles, *etc.* The most recent survey papers on ray shooting in the computational geometry field were published by Agarwal and Erickson [6] and Pellegrini [117].

The main problem of ray shooting techniques developed in the field of computational geometry, particularly aiming at worst-case complexity, is their inapplicability in practice. First, this inapplicability is caused by rather difficult implementation of these techniques, the restriction to object classes required by these techniques, the worst-case space complexity reaching at least $O(N^3)$, and the unavailability of any practical implementations. Especially, the space complexity for the underlying data structures of *RSAs* severely limits the use of these techniques to hundred(s) of objects approximately, which is unacceptable for practical use. This restriction holds even for techniques applied to $\mathbb{E}^2$ space.

Similarly, to the best of our knowledge, those average-case techniques developed in the field of computational geometry that are potentially promising for some practical use, and that are simpler from the implementation point of view, have not been implemented.

1.4.2 Computer Graphics

The computer graphics community has developed its own *RSAs*, starting after introduction of ray tracing [160]. The first applications in computer graphics that strongly required some *RSA* were ray-casting and ray-tracing; at present most new global illumination algorithms [149, 20] aiming at photo-realistic image synthesis also use some *RSA*. The algorithms for ray shooting developed within the field of computer graphics are of a heuristic nature, and do not follow the concept of worst-case complexity, but rather average-case complexity and practical feasibility.

At present there are several known *RSAs*, a substantial part of which is listed below. In the published papers these *RSAs* are called *acceleration techniques, acceleration methods, acceleration schemes, etc.* These *RSAs* usually assume that a ray-object intersection test is available for each shape of object, which allow us to use general shapes of objects, unlike the *RSAs* developed in the field of computational geometry. It is also commonly required that an axis-aligned bounding box tightly (or bounding volume in general, see Subsection 1.6.1) enclosing an object for an arbitrary shape of the object can be computed.

The most cited survey on *RSAs* was given in Glassner's book [18] by Arvo and Kirk. *RSAs* were also surveyed by Foley [47] and by Watt and Watt [157]. A more recent and fuller survey on *RSAs* was presented in Simiakakis' Ph.D. thesis [132]. The Arvo/Kirk's survey, although very systematic, is becoming obsolete since it does not cover developments in the last decade. It also does not contain any quantitative comparison of *RSAs*. The same holds for other surveys. Similarly, the survey by Simiakakis does cover developments since 1995.

Further, we present a list of algorithmic techniques, data structures, and issues that have been dealt in the context of *RSAs*, together with the most important citations[1]:

- basic spatial data structures
  - bounding volumes [123]
  - bounding volume hierarchy [158, 96, 63, 142]
  - spatial subdivisions
    - binary space partitioning (BSP) tree [94, 148, 82, 80]
    - *kd*-tree [105, 30, 15, 144, 145, 82, 80]
    - octrees (including Octree-R) [59, 139, 126, 118, 45, 159, 54, 147], and many others, (for survey, see [74])
    - uniform grid [53, 33, 90, 45, 163]
    - non-uniform grid [57]
    - hierarchy of grids [93, 100, 90, 25, 26]
    - Voronoi diagram [106]
  - ray classification scheme [17, 133, 132, 102]
  - ray coherence theorem [116, 89]
- augmenting spatial data structures
  - macro regions [41]
  - pyramid clipping [156]
  - proximity clouds [38]
  - directed safe zones [124]
  - largest common traversal sequence [78]
- additional improvements for spatial data structures
  - ray cache (mailbox) technique [23, 12, 97]
  - ray boxing technique [136, 163]
- special techniques
  - handling CSG primitives [23, 32, 165, 56, 167, 113]
  - plane traversal [49]
  - hierarchy of 1D sorted lists [51]
  - object/ray coherence [64]
  - generalized rays
    - beam tracing [87]
    - cone tracing [11]
    - pencil tracing [131]
  - techniques for restricted sets of rays
    - fixing the origin of rays (including hidden surface removal, *i.e.*, ray casting) [55, 155, 77] and many others.
    - fixing the direction of rays [101, 77]

[1]The full list of BibTeX entries can be found at: `http://www.cgg.cvut.cz/~havran`

- specific issues
  - termination criteria and heuristics for constructing spatial data structures [146]
  - memory storage/access issues [120, 73]
  - performance prediction of *RSAs* [122]
  - acceleration techniques to generate the sequence of images [50, 19, 60, 108, 121, 92, 28, 67, 29, 114]
  - coherence [156, 95]
  - hybrid spatial data structures (meta hierarchies) [16]
  - complexity analysis and comparison of *RSAs* [128, 49, 45, 33, 69, 109, 152, 153, 150, 151, 107, 48, 74, 85, 86]

Some of the techniques listed above can be combined together to get a more efficient *RSA*. For example, the ray cache technique can be plugged into nearly all *RSAs*. Moreover, a more general concept known as *meta-hierarchies,* which combines the basic spatial data structures into one, was proposed by Arvo [16], but no construction algorithm for such a meta-hierarchy has been proposed yet.

A detailed review of all these techniques would be rather space demanding and would require a separate review publication of major extent. For this reason we recall in this chapter only the most important facts concerning *RSAs*, which are necessary for an understanding of the rest of the thesis.

1.5 Complexity of *RSA*

The theoretical complexity bounds using O-notation for ray shooting problem were addressed in a series of papers by Szirmay-Kalos and Márton (chronologically cited: [107, 48, 150, 151, 152, 153]).

Since their results are important for understanding the contribution of this thesis, we make a brief survey of their work here. The interested reader is advised to study the paper [153] for more details. Most of this section follows the main results of this paper.

Szirmay-Kalos and Márton state that worst-case optimal *RSAs* are both difficult to implement and practically infeasible due to the their prohibitive memory and preprocessing time complexity. They present the concept of a “good” *RSA*, which should run in sublinear time after sub-quadratic preprocessing with linear space complexity. They show why heuristic *RSAs* are used in global illumination and their worst-case and average-case time complexity.

They propose a *complementer plane* algorithm that solves the ray shooting problem in $O(\log N)$ complexity with $O(N^8)$ space complexity for a general class of convex objects in $\mathbb{E}^3$. The main idea is to define ray space using the *complementary plane* perpendicular to an input ray. The required complementary plane from a set of such planes for a given ray can be found using a point location search. The number of topologically different projections of objects to the plane is finite, and thus the whole search space of complementary planes is also finite. The number of objects projected to any complementary plane is again finite, which also bounds the search space. The number of intersections with objects for a single ray is also finite, which discretises this search space. As a result, all the three search spaces (complementary plane, the position of the ray projected on the complementary plane, and the position of the object along the ray path) can be searched using point location. This is the classical problem in computational geometry, solvable using balanced binary trees in $O(\log N)$ time. As a result, the proposed *RSA* reaches $O(\log N)$ time with $O(N^8)$ space complexity, which is very prohibitive for any practical use.

Their next result is the lower bound of space complexity for any worst-case *RSA* working in $O(\log N)$ time complexity. They show that this lower bound is $\Omega(N^4)$ in worst-case, which is still prohibitive for any practical use. They further show the lower bound of worst-case complexity of ray shooting as a problem itself is $\Omega(\log N)$.

They also provide an average-case analysis of heuristic *RSAs*. They propose the concept of a *provocative RSA* assuming the ray origin is fixed in the space. The basic idea of the provocative *RSA* is to sort objects according to their distance from the origin point of a ray in $O(N.\log N)$ time. A ray shooting query is solved by checking the intersection with the closest object first and the farthest object last, resulting in $O(N)$ time complexity in the worst case. They analyze the average-case complexity of this provocative *RSA* for a set of spheres of the same radius uniformly distributed in the scene. They show that the average-case time complexity of a provocative *RSA* for the scene is formally $O(1)$, and this also holds for a scene with randomly distributed spheres of varying radii [107].

They state that almost every heuristic *RSA* based on spatial data structures has a common idea in the provocative *RSA*. The underlying spatial data structures try to represent the distance between the objects, not from a single point. To save the distance from all possible positions would require infinite storage space. The spatial data structures of heuristic *RSAs* use some approximation that subdivides space into finite spatial regions from where the objects can be roughly sorted according to their distance. They support their theory by results of simulation for randomly distributed spheres for uniform grids [53], ray coherence [116], ray classification [17, 133], and the Voronoi diagram [106].

They conclude with the statement that the fundamental assumption of uniformly distributed objects in the scene can be violated in most cases in practice. We may remark that many scenes that represent the real world require a lot of free space in them. The inhabitable structures formed by humans must contain much free space just to allow movement of humans and animals and transport of things (*e.g.*, streets, corridors, rooms).

The unknown multiplicative factor hidden behind O-notation and the assumption on uniformly distributed objects in the scene disables fair comparison of the heuristic *RSAs* using common formal complexity tools – O-notation. A significant quantitative difference in performance of heuristic *RSAs* has been observed in many papers [144, 109, 49, 44], although these *RSAs* have theoretical $O(1)$ average-case complexity for scenes with randomly distributed objects [153].

The average-case analysis for practical scenes that have a distribution of objects uncatchable by simple mathematical tools typically used for this purpose remains an unsolved problem. If the distribution of objects cannot be described by several parameters, it raises the question of whether an analysis is a solvable problem from a theoretical point of view at all. For this reason, the performance of *RSAs* is practically compared on set of test scenes, using for example the scenes from *Standard Procedural Database* (abbreviated to *SPD* throughout the thesis) introduced by Haines [69].

Papers addressing heuristic *RSAs* in the past used only a subset of *SPD* scenes or a private set of scenes, containing typically from 3 to 6 scenes. Research supported by experiments on such a low number of scenes is not statistically relevant. Further, the different implementation of reference *RSAs*, and the use of hardware-dependent timing statistics rather than hardware-independent characteristics has made a fair comparison of work published in various papers rather infeasible. Additionally, it is not known whether *SPD* scenes – although being scalable – offer a good representative set of scenes suitable for testing of *RSAs*. As a result of all the factors mentioned above, the papers about heuristic *RSAs* published in the past often contain mutually contradictory statements.

1.6 Basic Techniques used in *RSAs*

In this section we present a short survey of heuristic *RSAs*, since these are further elaborated in the thesis. For a more detailed study the reader is referred to the surveys by Arvo and Kirk [18] and/or Simiakakis [132], or to the original papers cited here.

RSAs based on basic spatial data structures have in practice a lower time complexity than a naïve *RSA*. They use either spatial subdivisions [18, 157], hierarchical clustering of objects [25], or a combination of these principles [100]. Below we describe all commonly used spatial data structures.

1.6.1 Bounding Volumes

A naïve *RSA* tests every object for intersection with a given ray. The ray-object intersection test itself can be an expensive operation, particularly for some shapes of objects (NURBs and other splines surfaces, polygons with many edges, *etc.*). Therefore it is advantageous to enclose tightly the object in a *bounding volume* with a simple ray-object intersection test. Practically, the bounding volume of an object O is a cell $\mathcal{V}$ for which holds $O \cap \mathcal{V} \equiv O$. If a ray intersects the objects' bounding volume, an intersection between the ray and the object is performed. If a ray does not intersect the bounding volume, it cannot intersect the object and thus a substantial part of the computation can be avoided on average.

A simple bounding volume is a sphere, which has a particularly simple ray-object intersection test [61]. The second possibility is to use tight $\mathcal{AB}$ of the object. Another alternative is to use arbitrarily oriented rectangular parallelepipeds [18], also called slabs. The requirements posed on the properties of a bounding volume are as follows:

Tightness: If a ray intersects the bounding volume, then the probability that the ray also intersects the object is high.

Efficiency: The time complexity of the intersection test between a ray and the bounding volume is small.

Some tradeoff must be sought for these two requirements. An example of bounding volume is the above-mentioned $\mathcal{AB}$. The use of bounding volumes for the data structures of *RSAs* as described above is more general – bounding volumes can be used both inside bounding volume hierarchies and also in the spatial subdivisions described below.

1.6.2 Bounding Volume Hierarchies

A natural extension to bounding volumes is a *bounding volume hierarchy* (abbreviated to BVH further in the thesis), which takes advantage of hierarchical coherence. Given the bounding volumes of objects, an n-ary rooted tree of the bounding volumes is created with the bounding volumes of the objects at the leaves. Each interior node v of BVH corresponds to the bounding volume that completely encloses the bounding volumes of the subtree rooted at v.

The hierarchy provides naturally the method for testing a ray with the objects in the scene. If the ray does not intersect the enclosing bounding volume at the root node, it cannot intersect any object. Otherwise, the hierarchy is recursively descended again only for those nodes of BVH whose bounding volumes were intersected by the ray. The interesting property of BVH is that although a bounding volume of a node always completely includes its child bounding volumes, these child bounding volumes can mutually intersect. The method for automatic construction of BVH was first described by Goldsmith and Salmon [63]. The construction of BVH proceeds *bottom-up* and is *object-oriented,* it does not have the property of ESSD or HSSD, since the bounding volumes overlap.

1.6.3 Spatial Subdivisions

A number of basic heuristic *RSAs* developed in the past are based on spatial data structures that subdivide the spatial region of a scene into cells, mostly using a method known as *divide and conquer*. The result is approximative ordering of objects in space according to their distance among the objects. The distance is somehow encoded into the properties of the spatial data structures, which are usually called *spatial subdivisions.* The quality of the distance encoding and the ease of accessing the distance in spatial subdivisions directly effects the performance of the corresponding *RSA*, which traverses spatial data structures and finally produces a result for a given ray shooting query. The basic spatial subdivisions were developed by several authors [95, 53, 59] and are described in more detail below. The common principle of all spatial subdivision structures is to subdivide the cell $\mathcal{V}$ corresponding to the scene $\mathcal{AB}$ into a set of cells $S_{\mathcal{V}}$ by a set of boundaries $S^b{}_{\partial\mathcal{V}}$. In terms of terminology given in Section 1.2, the principle is to create ESSD or HSSD for initial cell $\mathcal{V}$, where each elementary cell contains a list of pointers to objects fully or partially contained in the cell. The geometry of ESSD/HSSD, for known spatial subdivisions excluding Voronoi diagrams [106], is induced by splitting planes perpendicular to one of the coordinate axes.

Using spatial subdivisions, the corresponding *RSA* locates the elementary cells along the ray. If any intersection exists between a given ray and an object belonging to the elementary cell, and if the corresponding intersection point lies inside the elementary cell, then the answer to the ray shooting query is found. If there are more objects found to be intersected inside the cell, the closest one is selected. If no object with an intersection has been found or if the intersection point lies outside the currently processed elementary cell, the computation proceeds to the next elementary cell along the ray. The algorithm identifying the cells along the ray path is called the *ray traversal algorithm.* An *RSA* is thus composed of an underlying spatial data structure and a corresponding ray traversal algorithm.

Some spatial subdivisions require to have elementary cells of the same shape and size. The common property of spatial subdivisions, unlike BVH, is that the elementary cells are always disjoint (non-overlapping). The constructed elementary cells are either addressed directly or from the hierarchical cells. The construction of spatial subdivisions usually proceeds in a *top-down* way. The concept of spatial subdivisions is *space-oriented.*

Since we deal with *RSAs* based on spatial subdivisions in detail in the rest of the thesis, we continue below with a description of the most commonly used spatial subdivisions.

1.6.3.1 BSP Trees and *Kd*-Trees

A *Binary Space Partitioning* (BSP) tree is a spatial subdivision that can be used to solve a variety of geometrical problems. It was initially developed as a means of solving the hidden surface problem in computer graphics [52]. It is a higher dimensional analogy to the binary search tree. The BSP tree has two major variants in computer graphics, which we call *axis-aligned* and *polygon-aligned.*

The polygon-aligned form [52, 66] chooses a plane underlying the polygon as the splitting entity that subdivides the spatial region into two parts. The scene is typically required to contain only polygons, which is too restrictive for ray shooting applications. We do not deal with the polygon-aligned form of BSP tree here. For a survey and application of the techniques based on the polygon-aligned form of the BSP tree see for example [43, 112].

In the axis-aligned form of the BSP tree the splitting entity is the plane that is always perpendicular to one of coordinate axes. The concept was first used for an *RSA* by Kaplan [94]. Since the splitting planes are perpendicular to the coordinate axes, the spatial subdivision is also called a rectilinear BSP tree or an orthogonal BSP tree. Since in most of the thesis (Chapter 4–7) we deal with the axis-aligned form of the BSP tree, we describe it in greater detail than other basic spatial data structures.

A BSP tree for a set S of objects is defined as follows: Each node ν in the BSP tree is associated with its axis-aligned bounding box $\mathcal{AB}(\nu)$, which is a cell. The cell associated with the root of the BSP tree is the axis-aligned bounding box $\mathcal{AB}$ of all objects from S. Each interior node ν of the BSP tree is assigned a splitting plane H_ν that subdivides $\mathcal{AB}(\nu)$ into two cells. Let H_ν^+ be the positive halfspace and H_ν^- the negative halfspace bounded by H_ν. The cells associated with the left and the right child of ν are $\mathcal{AB}(\nu) \cap H_\nu^+$ and $\mathcal{AB}(\nu) \cap H_\nu^-$, respectively. The left subtree of ν is a BSP tree for the set of objects $S_\nu^- = \{s \cap H_\nu^- \neq \emptyset | s \in S_\nu\}$, and the right subtree is defined similarly. Each *leaf* node ν^E may contain a list of objects S_{ν^E} that intersect the axis-aligned bounding box $\mathcal{AB}(\nu^E)$ associated with ν^E. When a leaf contains at least one object, we call it a *full leaf.* Otherwise, we call it an *empty leaf.* We call a cell $\mathcal{AB}(\nu^E)$ associated with the leaf ν^E a *leaf-cell.* For the sake of convenience, let $lchild(\nu)$ denote a left child of node ν and similarly let $rchild(\nu)$ denote its right child.

An example of a BSP tree in $\mathbb{E}^2$ is depicted in Fig. 1.3. In terms of terminology given in Section 1.2

the leaf of a BSP tree corresponds to the elementary cell of HSSD and the interior node to the generic cell of HSSD.

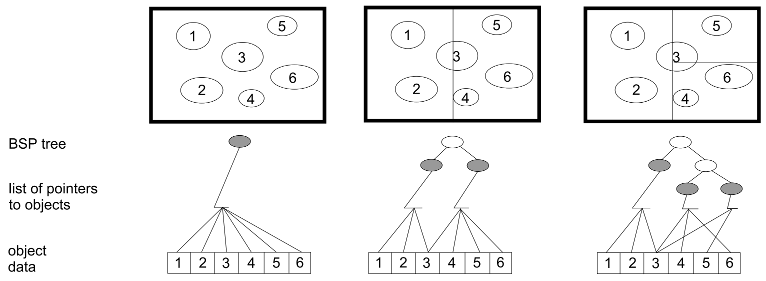


Figure 1.3: An example of the BSP tree in $\mathbb{E}^2$.

The orthogonality of splitting planes in the BSP tree significantly simplifies the intersection test between a ray and the splitting planes. The cost of computation of the signed distance corresponding to the intersection point between a ray and the axis-aligned splitting plane is roughly three times lower than for an arbitrary positioned plane (for a detailed explanation, see Section 4.2).

A BSP tree is usually constructed hierarchically in top-down fashion, as outlined in the pseudocode, Algorithm 1. At a current leaf ν a splitting plane is selected that subdivides $\mathcal{AB}(\nu)$ into two cells. The leaf then becomes an interior node with two new leaves. The objects associated with ν are distributed into its two new descendants. The process is repeated recursively until certain *termination criteria* are reached. Commonly used termination criteria are the maximum leaf depth and the number of objects associated with the leaf.

Algorithm 1 BSP tree construction, recursive version of the algorithm.

```
procedure Subdivide($CurrentNode, CurrentTreeDepth, CurrentSubdividingAxis$)
if ( ($CurrentNode$ contains too many objects)
and ($CurrentTreeDepth$ is not too high) ) then
    Children of $CurrentNode \leftarrow CurrentNode$'s Bounding Volume
    {Note that $child[0].max.DividingAxis$ and $child[1].min.DividingAxis$ are always equal.}
    if $CurrentSubdividingAxis$ = X-Axis then
        $child[1].min.x \leftarrow$ mid-point of $CurrentNode$'s X-Bound
        $child[0].max.x \leftarrow$ mid-point of $CurrentNode$'s X-Bound
        $NextSubdividingAxis \leftarrow$ Y-Axis
    else if $CurrentSubdividingAxis$ = Y-Axis then
        $child[1].min.y \leftarrow$ mid-point of $CurrentNode$'s Y-Bound
        $child[0].max.y \leftarrow$ mid-point of $CurrentNode$'s Y-Bound
        $NextSubdividingAxis \leftarrow$ Z-Axis
    else if $CurrentSubdividingAxis$ = Z-Axis then
        $child[1].min.z \leftarrow$ mid-point of $CurrentNode$'s Z-Bound
        $child[0].max.z \leftarrow$ mid-point of $CurrentNode$'s Z-Bound
        $NextSubdividingAxis \leftarrow$ X-Axis
    end if
    for all objects referenced in $CurrentNode$ do
        if the object is within children's bounding volume then
            add the object to the children's object list
        end if
    end for
    Subdivide($child[0], CurrentTreeDepth + 1, NextSubdividingAxis$)
    Subdivide($child[1], CurrentTreeDepth + 1, NextSubdividingAxis$)
end if
```

An important feature of the axis-aligned form of the BSP tree is its adaptability to the scene geometry that is induced by the possibility to position the splitting plane arbitrarily. Traditionally, the splitting plane is positioned at the mid-point of the chosen axis, and the order of axes is regularly changed on successive levels of the hierarchy [94]. Another method uses adaptive positioning of the splitting planes when the position of the splitting plane is chosen along the whole range using a surface area heuristic [105] (described in Chapter 4).

The positioning of the splitting plane is sometimes used to distinguish between the BSP tree and the kd-tree, which was introduced by Bentley [21] in 1975. Conceptually, the BSP tree and the kd-tree are equivalent. The important feature of the kd-tree is that it always has axis-aligned splitting planes unlike the BSP tree. In some literature, the axis-aligned form of the BSP tree is only a type of kd-tree where the splitting plane always lies at the mid-point of the current box resulting in two children with cells of equal size. In some other literature, the axis-aligned form of the BSP tree can have arbitrary positioning of the splitting planes and is thus equal to the kd-tree. In this thesis, a BSP tree refers to a binary space partitioning in the axis-aligned form that always uses mid-point positioning of the splitting planes. The kd-tree can have arbitrarily positioning of the splitting planes. Thus any BSP tree is a kd-tree, but not vice versa. The positioning of the splitting planes can significantly influence the performance of *RSAs* based on the kd-tree, particularly for sparsely occupied scenes[2]. This is described in detail in Chapter 4.

Algorithm 2 The BSP tree and the kd-tree ray traversal algorithm, recursive version.

```
function RayTreeIntersect(ray R, $node$, $min$, $max$): $object$
if $node$ is empty then
  RayTreeIntersect ← ”no intersection”
else
  if $node$ is a leaf then
    Intersect ray R with each object referenced in the leaf discarding those farther away than $max$.
    RayTreeIntersect ← the ”object with the closest intersection point”
  else
    $t$ ← the signed distance along ray to the splitting plane of the $node$
    $Near$ ← the child of $node$ for half-space containing the origin of R
    $far$ ← the ”other” child of $node$ – i.e. not equal to near
    if $(t > max)$ and $(t < 0)$ then
      {Whole interval is on near cell – recursion.}
      RayTreeIntersect ← RayTreeIntersect(R, $near$, $min$, $max$)
    else
      if $t < min$ then
        {Whole interval is on far cell – recursion.}
        RayTreeIntersect ← RayTreeIntersect(R, $far$, $min$, $max$)
      else
        {The ray intersects the plane – recursion.}
        hitData ← RayTreeIntersect(R, $near$, $min$, $t$) {Test near cell.}
        if (hitData indicates that there was a hit) then
          RayTreeIntersect ← [hitData]
        else
          {There was no hit in the near cell – test far cell.}
          RayTreeIntersect ← RayTreeIntersect(R, $far$, $t$, $max$)
        end if
      end if
    end if
  end if
end if
```

Given a ray and a BSP tree, the ray traversal algorithm identifies elementary cells along the ray.

[2]Sparsely occupied scene (also called unevenly occupied scene) is such a scene, where the distribution of objects in the scene is non-uniform. Most of the space in the scene is then empty.

There are a few variants of the ray traversal algorithm for the BSP tree, they are further elaborated in Chapter 5. Here, we shortly describe only the recursive ray traversal algorithm for BSP tree, which is outlined in the pseudocode, Algorithm 2. The basic idea of this algorithm is that for each node we identify one of four possible cases: to traverse only the left child, only the right child, the left child first and then the right child, or the right child first and then the left child. When we descend to the left or right child, we recurse with the same algorithm.

1.6.3.2 Octrees

An octree (Octal tree) in $\mathbb{E}^3$ is a spatial subdivision that is similar to the quadtree [127] in $\mathbb{E}^2$ space. It is also built recursively in top-down fashion, but unlike to the BSP tree the initial cell $\mathcal{V}$ is not split into two cells but into eight cubic cells (in $\mathbb{E}^n$ space into 2^n cells). The cubic cells of the octree, often called *octants*, lie at various depths from the root node and thus they vary in size. The octree itself can serve as the representation of a three-dimensional object [127]; in this case the leaves of the octree are marked either empty or full. The use of the octree for *RSA* was introduced by Glassner [59]. Cells with high object occupancy can be recursively subdivided into smaller and smaller cells, generating new cells in the octree.

The addressing of child nodes when accessing the interior octree node is provided by direct pointing or hashing. In the former case the interior node has to contain eight pointers to its descendants. In the latter case the address of the child nodes used for hashing is usually formed by postfixing or prefixing the parent address by digits from 1 to 8 corresponding to the geometrical position of the child node, as depicted in Fig. 1.4. Numbering the nodes this way (instead of from 0 to 7) loses the octal purity of the original scheme, but improves the hashing itself [59].

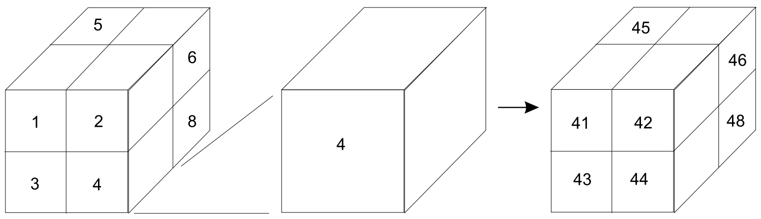


Figure 1.4: Octree space partitioning.

The ray traversal algorithm for the octree is more complicated than for the BSP tree or the uniform grid (see subsection below). Each ray intersecting an octree node can visit at most its four of eight descendants, and the computation of their order to be visited along the ray path is thus more involved than for the BSP tree. The time consumed by a ray traversal algorithm for the octree is given by the efficiency and robustness of the algorithm determining which child nodes are to be visited and in which order. Recently, a survey [74] dealing with ray traversal algorithms for the octree has been published.

The octree naturally exploits spatial coherence because the objects that are close to each other in space are referenced in leaves that are close to each other in the octree. The common termination criteria for octree construction are the same as for the BSP tree: the maximum allowed depth and the minimum number of references to objects in the leaves.

The properties of the octree that are related to the *RSA* can be compared to the properties of *kd*-tree (or the BSP tree). However, there are cases where the performance of *RSAs* based on the octree and the *RSA* based on the *kd*-tree significantly differ. We should note that the geometry induced by the octree can be simulated by the BSP tree, but the octree has a more regular structure. Depending on the ray traversal algorithm, the order of all the octants to be visited in the ray traversal algorithm is usually determined even if the ray can terminate already in the first octant. Small occupancy of the leaves for sparsely occupied scenes is the next disadvantage of the octree. The empty neighbor leaves in the octree have to be determined and traversed, whereas in the case of construction of the BSP tree it is likely they would be represented by a single leaf. The small occupancy of leaves implies relatively large memory requirements for octree representation.

There is a variant of the octree called *Octree-R* [159] that results in arbitrary positioning of the splitting planes inside the interior nodes. The difference between the octree and Octree-R is similar to that between the BSP tree and the *kd*-tree. The smart heuristic algorithm for positioning the splitting planes inside the octree interior node is applied independently for all three axes. Then the speedup between the octree-R and the octree can be from 4% up to 47%, depending on the distribution of the objects in the scene [159].

1.6.3.3 Uniform and Non-Uniform Grids

A uniform grid is another common spatial subdivision used in *RSAs*. It involves the subdivision of initial cell $\mathcal{V}$ into equally sized elementary cells formed by splitting planes that are axis-aligned regardless of the distribution of objects in a scene. In terms of introductory terminology the uniform grid is ESSD. The *n*-dimensional grid resembles the subdivision of a two-dimensional screen into pixels. A list of objects that are partially or fully contained in the cell is assigned to each parallelepiped cell (also called a *voxel* in $\mathbb{E}^3$ space). The uniform grid in the context of *RSA* was first introduced by Fujimoto [53]. The ray traversal algorithm for a uniform grid has been improved by several researchers, including Hsiung [90] and Endl [45].

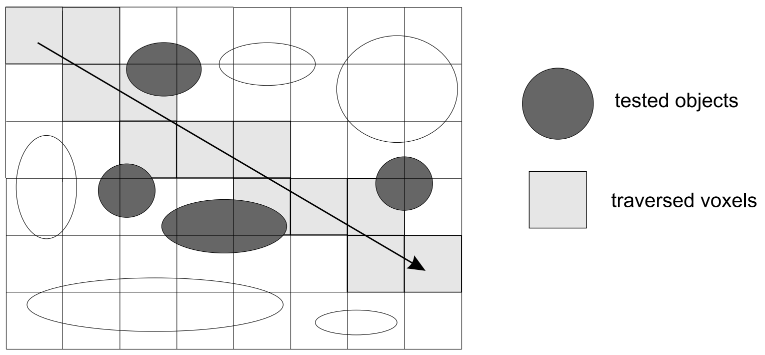


Figure 1.5: An example of the uniform grid in $\mathbb{E}^2$.

Since the uniform grid is created regardless of the occupancy of objects in the voxels, it typically forms many more voxels than the octree or the BSP tree, and therefore it demands necessary storage space. Nevertheless, the ray traversal algorithm for the uniform grid can be performed very efficiently, since the voxels are of the same size.

The ray traversal algorithm, often referred to as 3D-DDA, is analogous to the Bresenham algorithm for drawing a straight line in $\mathbb{E}^2$ raster space and thus requires a simple operation for each traversal step (addition, subtraction, and comparison).

The disadvantage of the uniform grid is that the occupancy of most voxels can be very small, particularly for sparsely occupied scenes. In this case a ray typically has to traverse many empty voxels before hitting the full voxel. Since in sparsely occupied scenes most objects are located in a relatively small number of all the voxels, the full voxels contain many references to objects. These voxels are costly to check for intersection with the ray. The uniform grid does not adapt to the distribution of the objects in the scene, and in the worst-case the improvement of an *RSA* based on the uniform grid over a naïve *RSA* need not be significant.

The uniform grid was analyzed by Clearly and Wyvill [33]; they showed that the optimal subdivision for one axis is $k \doteq d_{voxel}.\sqrt[3]{N}$ voxels and the minimum total time per ray then corresponds to

$t_{min} \doteq const.\sqrt[3]{N}$ for N objects of the same shape and size that are uniformly distributed in space. The parameter d_{voxel} denotes voxel density – the required ratio between the number of voxels and the number of objects. The required number of voxels is then $n_r = N.d_{voxel}$. We call this method for setting the resolution of the uniform grid a *homogeneous method* since it disregards the shape of the scene $\mathcal{AB}$. The shape of the voxel is a small copy of the scene $\mathcal{AB}$.

In order to get a more efficient *RSA* based on a uniform grid, other methods for setting up the uniform grid resolution have been developed (most of them in the context of hierarchical grids as described below).

In order to achieve a more cubic shape of voxels Woo [163] uses the method of setting up the uniform grid resolution that we call here *Woo's method.* Let x, y, and z be the size of the scene $\mathcal{AB}$ in all three axes. Then to get n_r voxels in the uniform grid the following formulas are used:

$$c = MAX(x,y,z), \quad n_r = N.d_{voxel}, \quad u = \sqrt[3]{d_{voxel}.n_r}$$
$$N_\mathrm{x} = MAX(1, \frac{x}{u.c}), \quad N_\mathrm{y} = MAX(1, \frac{y}{u.c}), \quad N_\mathrm{z} = MAX(1, \frac{z}{u.c}), \tag{1.2}$$

where N_x, N_y, and N_z is the number of voxels along the x, y, and z-axis.

Similarly, Klimaszewski [100] developed a method to set the resolution of the uniform grid that we call here a *heterogeneous method.* In order to get n_r voxels, the heterogeneous method uses the following formulas:

$$N_\mathrm{z} = \lceil \sqrt[3]{\frac{n_r.z^3}{x.y}} \rceil, \quad N_\mathrm{y} = \lceil \sqrt{\frac{n_r.y}{N_\mathrm{z}.x}} \rceil, \quad N_\mathrm{x} = \lceil \frac{n_r}{N_\mathrm{y}.N_\mathrm{z}} \rceil \tag{1.3}$$

The *non-uniform grid* was proposed by Gigante [57]. In the non-uniform grid the splitting planes are also axis-aligned, but along the axis they can be positioned arbitrarily. Non-uniform grids can have a better fit of voxels to the scene geometry, since for sparsely occupied scenes they put more splitting planes to the spatial regions with higher object occupancy. The positioning of the planes is performed according to the histogram of objects along the coordinate axes. The disadvantage of the non-uniform grid is the lower efficiency of its ray traversal algorithm, since the 3D-DDA algorithm cannot be used. Gigante showed that for several tested scenes the performance of *RSA* based on the non-uniform grid is lower than that of *RSA* based on the uniform grid.

In order to improve further the efficiency of ray traversal algorithms, a few techniques of coding empty space around each voxel have been developed. If a voxel is empty, then we can determine the smallest distance from some voxel to the first full voxel in the number of empty voxels in all directions. If the voxel with the encoded distance to the empty voxel is visited, a ray traversal algorithm can determine that all the empty voxels in the direction of the ray can be skipped. This technique was first introduced by Cohen and Sheffer [38]; for each voxel it determines the smallest distance to a full voxel. A similar but directional technique was presented by Semwal and Kvarnstrom [124]. It considers the six possible directions to encode the smallest distance to the first full voxel. This directionality allows us to increase the distance to the first full voxel. Obviously, both techniques suffer from large storage space and preprocessing requirements that strongly depend on the distribution of objects in the scene.

1.6.3.4 Hierarchical Grids

A hierarchical grid is another attempt to avoid the regularity of uniform grids, since this regularity is particularly inconvenient for sparsely occupied scenes. The common principle used in hierarchical grids is to insert uniform grids recursively into other uniform grids. The known methods of several hierarchical grids strongly differ in the construction phase. Hierarchical grids also require a more complicated ray traversal algorithm than uniform grids. Three *RSAs* based on hierarchical grids have been published. For a survey, more detailed description, and performance comparison, see [135], or for a less detailed survey [85]. We describe here only the basic properties of *RSAs* based on hierarchical grids.

Recursive Grids

Recursive grids (abbreviated to RG further in the thesis) simply bring the concept of recursiveness into uniform grids. The principle is as follows: construct a uniform grid over the set of objects, assign the objects to all voxels where they belong. Then recursively descend; that is, for each voxel that contains more objects than a given threshold, construct a grid again. The construction of grids is terminated when the number of objects referenced in the voxel is smaller than a threshold or some maximum depth for grids is hit. This maximum depth is usually set to two or three. The principle is thus similar to the BSP tree or the octree, but at one step a cell is subdivided into more than two or eight child cells.

Hierarchy of Uniform Grids

Cazals *et al.* [25, 26] published an *RSA* based on a *hierarchy of uniform grids* (abbreviated to HUG further in the thesis) that is similar to recursive grids. There are two main differences. First, the inserted grids need not align with the parent grid subdivision. Each grid is understood as an autonomous object possibly contained in another grid. Second, the construction of HUG is quite different. In the first step, the objects are filtered according to their size into groups, and the number of groups is usually either two or three. Let us suppose three groups are constructed by filtering. The first group contains large-size objects, the second group middle-size objects, and the last group small-size objects. Within the middle-size and small-size groups we perform clustering according to the distance between the objects in the group. Thus each group contains several clusters. For objects in the large-size group we construct an initial global grid. For all clusters from the middle-size and small-size group having a large enough number of objects we also construct a grid. These grids are inserted into the global grid recursively, so the smaller grids are contained in bigger ones. The author's statement in the introduction to the papers that the method is fully automatic is true as far as it goes; however, it does require at least two parameters to be set initially (the number of groups and delta-connectivity).

Adaptive Grids

Klimaszewski and Sedeberg described an *RSA* based on *adaptive grids* [100, 98] (further abbreviated to AG). The spatial data structure principally differs from RG and HUG within the construction. The first step of the construction algorithm is clustering of objects according to some criteria based on the distance between two candidates. A candidate can be already existing cluster of objects or a single object. The criteria take into consideration candidates' and the resulting cluster's surface areas. Moreover, the resulting cluster must be small enough in comparison with the $\mathcal{AB}$ of the whole scene. In the second phase of the algorithm, a bounding volume hierarchy (BVH) is built up over the clusters. Grids are then constructed for the clusters (leaves of the BVH). The number of children in the interior nodes of the BVH is small, so it is inconvenient to create grids for these nodes. In the last phase, so called sub-voxel grids are constructed for those voxels where the number of objects is greater than a given threshold. The ray traversal algorithm is thus similar to that for BVH.

1.6.4 Ray-Space Subdivisions

Several *RSAs* are based on the subdivision of a *ray space*. The ray space is described by 5-dimensional coordinates. The first three coordinates of the ray space are Euclidean $\mathbb{E}^3$ and correspond to the origin of the ray. The next two coordinates are spherical σ^2 and determine the direction of the ray. The ray space subdivision thus uses $\mathbb{E}^3 \times \sigma^2$ space. Let us suppose we have N objects in the cell, which are indexed from 0 to $(N-1)$. Then we can define an *assignment function* for the ray space:

Definition 1 The assignment function $f_A(x,y,z,\varphi,\rho)$ for a set of N objects is the discrete function $f : \mathbb{E}^3 \times \sigma^2 \rightarrow Z_0^+ \bigcup\{-1\}$ that solves the ray shooting problem directly and is defined as follows:

$$f_A(x,y,z,\varphi,\rho) = i \quad \begin{cases} i = k, k \in \langle 0, N-1 \rangle & i \text{ corresponds to the index of the closest object intersected} \\ i = -1 & \text{if the ray does not intersect any object in the cell} \end{cases}$$

The ray space, over which function f_A is defined, has to be discretised for any practical use. The space $\mathbb{E}^3$ for the origin of the rays is discretised into cells. Similarly, σ^2 space is also discretised so that a set of solid angles completely covers the unit sphere. Because of discretisation, one node of the data structure describing the f_A does not correspond to a single half-line but to a spatial region called a *hyper-cubic region*. The assignment function f_A should return the *candidate list* of indexes of objects instead of one index to an object that can be visible from the origin point of a ray. For a particular ray shooting query the correct candidate list is searched and all the objects in the candidate list are tested for intersection with the ray. The object with the closest intersection point is then chosen, if such an object exists.

The ray classification scheme was suggested by Arvo and Kirk [17] and further elaborated by Simiakakis [133, 132]. Ray space can be subdivided using some variant of binary space partitioning. This is performed for primary rays and also for higher order rays in ray tracing. The geometry primitive corresponding to the approximation of ray space is the polyhedral volume defined in $\mathbb{E}^3$, also called a beam. Whatever the improvements of the algorithm based on the ray space subdivision over naïve *RSA*, the method suffers from an algorithmic paradox: the construction of the candidate lists is much more computationally demanding than with spatial subdivisions. Arvo and Kirk report the high time complexity of detecting polyhedral intersections and suggest an approximation where hypercubic regions are bounded by cones [18]. Kwon *et al.* [102] presented an approach aimed at decreasing the space complexity of ray classification by removing one dimension of ray space. Nevertheless, the results presented by Simiakakis [132] show that the performance of ray classification strongly depends on the distribution of the objects in the scene. The results of Kwon *et al.* [102] support this conclusion for six scenes with uniform grids and octrees.

Several other similar *RSAs* based on the directionality of rays have been published. The best known technique, by Ohta and Maekawa [116], is based on the *ray coherence theorem*, further elaborated by Horvath *et al.* [89]. This theorem bounds the angle between the rays starting on the first object and hitting the second object. If we consider two spheres of radius r_1 and r_2 and if the distance between their centers is d_{12}, then the maximum size of the angle ϕ is:

$$\cos\phi \geq \sqrt{1.0 - \frac{r_1 + r_2}{d_{12}}}$$

RSAs based on the ray-coherence theorem for each object and several primary directions (usually six corresponding to three coordinate axes in positive and negative directions) compute the potential set of objects that can be intersected. The objects in these sets are sorted according to the distance from the base object. When a ray shooting query is to be answered, the ray is tested for intersection with all the objects within the set, starting from the closest object. This technique has at most $O(N^2)$ space complexity.

Unfortunately, all *RSAs* based on ray-space subdivision are actually an approximation of worst-case *RSAs*, hence they exhibit high storage space and preprocessing time complexity.

1.7 Contribution of the Thesis

This thesis covers two major topics. In the first part we develop an *RSA* computation model. This model allows mapping of any *RSA* to this computation model by using commonalities of underlying data structures and ray traversal algorithms of all *RSAs*. Based on the properties of the computation model we develop an *RSA* performance model. Based on these two models, we present the methodology for comparing various *RSAs* based on the results of experiments for a set of scenes. Using this methodology, we compare 12 different *RSAs* using a set of 30 scenes.

The second part of the thesis describes several new methods for improving *RSAs* based on the *kd*-tree. It covers both construction and ray traversal algorithms. Emphasis is put on the practical applicability of the achieved results in computer graphics applications. More specifically, the contribution of the thesis is:

- General issues concerning *RSAs*
  - design of a new computation model and performance model for *RSAs*. These models have enabled the development of a new methodology for comparing various *RSAs* by experiments on a set of scenes [83].
  - practical comparison of 12 *RSAs* for a set of 30 scenes, the design of testing procedures to shoot rays in a scene in this comparison [84, 79].
- Construction algorithms for *kd*-trees
  - a new construction method that uses empty spatial regions in the scene, resulting in improved performance of the *kd*-tree for *RSA*.
  - a new termination criteria algorithm for *kd*-tree construction.
  - a new principle useful in *kd*-tree construction that splits the $\mathcal{AB}$ of the object intersected by the splitting plane into two new $\mathcal{AB}$s.
  - a new general cost model for *kd*-tree construction.
  - a new construction algorithm for the *kd*-tree based on the general cost model for a preferred set of rays fixing either the origin or direction of a ray [77].
- Ray traversal algorithms for *kd*-trees
  - a new fast robust recursive ray traversal algorithm for a *kd*-tree with the minimum number of conditions to be performed [82].
  - a new concept of a longest common traversal sequence for a set of rays based on *kd*-trees, particularly suitable for hidden surface removal [78].
- Memory mapping of the nodes of the *kd*-tree to increase the cache hit ratio, and thus performance, for all traversal algorithms for computer architectures with a large cache line [72, 73].

1.8 Organization of the Thesis

The thesis includes several chapters describing the author's development together with the description of some previous work. These chapters have been arranged into an order that will be more logical for the reader than the chronological order in which the author's papers were originally published. The thesis should be read for best understanding in *linear order*, starting with the introductory chapter. The chapters include the motivation for the problem, previous work, possible definitions and terminology used, the detailed elaboration of problems, the description of new algorithms, and a summary of results.

The main content of the thesis is presented in Chapters 2–7. In Chapter 2 we deal with the *RSA* computation model and performance model. These models allow us to develop a methodology for comparing various *RSAs* based on experiments with a set of scenes. Chapter 3 presents a comparison of 12 *RSAs* for a set of 30 scenes with different numbers of objects. The performed comparison shows that for the set of scenes the statistically best *RSA* is that based on the *kd*-tree. Chapter 4 is devoted to *kd*-tree construction algorithms in detail. It describes the adaptive positioning of the splitting plane using a cost model, utilization of empty spatial regions in the scene, termination criteria, and the construction for preferred ray sets fixing either the origin or direction of a ray. In Chapter 5 we describe several ray traversal algorithms for the *kd*-tree in detail. It includes a sequential ray traversal algorithm, a recursive ray traversal algorithm, and two variants of a neighbor-links based ray traversal algorithm. Chapter 6 deals with the improving the ray traversal algorithm for the *kd*-tree to decrease the number of traversal steps for restricted ray sets. Chapter 7 deals with a more hardware specific issue; how to map the nodes of the *kd*-tree to the physical memory to improve the data locality during execution of a traversal algorithm working over the *kd*-tree. Finally, Chapter 8 concludes the thesis with a short summary of all the results and several possible topics for further research.

Chapter 2

Comparison Methodology

In this chapter we develop an *RSA* computation model that allows us to map any particular *RSA* to the computation model. Further, we develop an *RSA* performance model that establishes the correspondence between the computation model and the running time of the *RSA* for a sequence of ray shooting queries. Based on the *RSA* computation and performance models, we propose a set of parameters describing the use of *RSA* in applications that allow us to make a fair comparison of various *RSAs* for the same set of input data. This chapter follows the paper [83].

2.1 Motivation

One of the main problems in research on *RSAs* is how compare them qualitatively and quantitatively. This should be done on a technically sound basis, it should define time and memory complexity, suitability for various type of scenes, and particular features for an *RSA*. In spite of two decades of research on *RSA* in the computer graphics community, it is not yet clear if some particular *RSA* is more convenient and/or more efficient than any other *RSA*. Some contradictory statements about *RSAs* have appeared with the introduction of new types of *RSA* in the published papers.

Only a few papers devoted to the comparison of various *RSAs* [109, 45, 49] have been published until now. Moreover, each paper that introduces a new *RSA* must, or at least should, compare the proposed algorithm with some reference algorithm. There is no common choice for the reference algorithm, but in most cases a uniform grid (see Subsubsection 1.6.3.3) was used. The quantitative comparison between a reference *RSA* and newly proposed *RSA* always depends on the software implementation and on a particular hardware platform. For this reason any cross comparison of the results presented in the different papers has been rather problematic or even impossible.

In this chapter we will try to decrease the gap in understanding of the functionality of various *RSAs* by finding out their commonalities. The commonalities found in all *RSAs* allow us to describe the *RSA* computation model in a general way that allows us to map a particular *RSA* to this computation model. Further, we will define the performance model that establishes the connection between the running time of the application and the different algorithmic operations that are the subject of the computation model. Then we will develop an “ideal *RSA*” that allow us to compute the answers to ray shooting queries in constant time. The “ideal *RSA*” results in the smallest possible time that can ever be achieved using a certain hardware and a set of ray-object intersection routines. We use the time consumed by the “ideal *RSA*” as a reference time value for all the parts of the computation in a specific *RSA*. These parts cover the time needed to traverse the data structure on which *RSA* is based, ray-object intersection tests, and the remaining time consumed by the application. The design of the computation model and performance model allow us to define the set of thirteen parameters referred to as the minimum testing output that should be reported for one experiment, given a scene, a particular *RSA*, and a sequence of ray shooting queries. Further, we describe how to get the minimum testing output. The definition of the two models and the minimum testing output is the basis for a methodology for making a fair comparison of various *RSAs*.

The concepts proposed below form a comparison methodology that allows us to compare various *RSAs* independently of the hardware and the implementation used. The initial concept of the comparison methodology based on experiments was described in Havran and Žára [86] for the *kd*-tree, and the technique presented here is a generalised and extended version for any *RSA*.

2.2 *RSA* Computation Model

In this section we introduce the *RSA computation model*. We will show that any *RSA* currently known or developed in the future can be mapped to this computation model. The computation model is based on the definition of algorithmic operations in an *RSA*. These algorithmic operations must always be performed due to the nature of the ray shooting problem. We then use the computation model to describe the set of parameters to be reported, when an *RSA* is tested experimentally.

The ray shooting problem can be understood as an instance of geometric range-searching [6], which implies that some data structure is built to answer the specific query. The definition of the ray shooting implies that every *RSA* contains somewhere pointers to objects that are to be tested for intersection against a given ray. This means that each *RSA* is separated into two parts (like all algorithms, see [10]): the data structure (further abbreviated to *DS*) containing at least pointers to the scene objects and the ray traversal algorithm working over *DS*. The lifetime of an *RSA* is composed of two phases, the first one is called *preprocessing phase* and it involves the construction of the initial *DS*. The second phase of an *RSA* is called the *execution phase*. Within the execution phase the *RSA* answers given ray shooting queries.

More theoretically, an *RSA* can be described as a special case of a general *RAM* model [10, 154], where any memory cell can be accessed in constant time or through a series of pointers. A *DS* is composed of some data entries, here referred to as *nodes*, which contain some data. It usually involves the pointers to the scene objects, the pointers to the other nodes of *DS*, the description of cells, *etc.* Nodes of a *DS* can be divided into two groups: *elementary nodes* are intended to contain only pointers to objects (and, if *RSA* requires it, some other data), whereas *generic nodes* are all other nodes, which point to other generic and/or elementary nodes. A special case of elementary nodes are *empty elementary nodes* that do not contain any pointers to objects and act as "free space containers" within the *DS*.

When answering a ray shooting query in a particular *RSA*, the computation proceeds as follows. Given a ray R, a ray traversal algorithm begins at a special starting node of a *DS* and performs a sequence of the following operations:

TRAVERSAL STEP:	visit a new node of the *DS*,
NEW NODE:	create a new node of the *DS*,
DELETE NODE:	delete a node from the *DS* and unlink all pointers to the node from the remaining nodes of the *DS*,
TEST OBJECTS:	when accessing an elementary node of the *DS*, test objects pointed to in this node for the intersection with the ray R,

finally finding the closest intersected object if such an object exists. There are two cases: a *DS* is or is not changed by a ray traversal algorithm. If a *DS* underlying the *RSA* is not changed by the ray traversal algorithm, then the operations "NEW NODE" and "DELETE NODE" are not performed during the execution phase. Such an *RSA* is referred to as the *RSA* based on a *static data structure*.

There are several *RSAs* that modify the underlying *DS* on the fly within the execution phase, for example, ray space subdivision techniques [17]. The operations "NEW NODE" and "DELETE NODE" can be used within the preprocessing phase to build up some initial *DS*, however, this *DS* is modified during the execution phase. Such an *RSA* is referred to as *RSA* based on a *dynamic data structure*.

Since every *RSA* can be mapped to this general *RSA* computation model, this enables us to define a common set of parameters to be reported when any *RSA* is performed on an input scene $\mathcal{S}$ containing N objects over an input sequence of ray shooting queries. The sequence of ray shooting queries induced by the application for an input scene $\mathcal{S}$ is associated with a *testing procedure*. The symbol for the testing procedure is *TP*. The testing procedure is an algorithm in the application that generates a sequence of ray shooting queries to be answered by a particular *RSA*. A particular testing procedure *TP* can be the result of a global illumination algorithm such as ray tracing, *etc.*, or just an artificial algorithm shooting rays to obtain some required distribution of rays in space [84].

We propose to organize the set of parameters resulting from the use of a particular *RSA* on the input scene and given a *TP* into three subsets, the first two of them hardware/implementation/compiler independent:

- *RSA* parameters related to static properties of data structure *DS*:
  - If an *RSA* is based on a static data structure, the parameters depend on the scene $\mathcal{S}$ only, and they are evaluated at the end of the preprocessing phase.
  - If an *RSA* is based on a dynamic data structure, the parameters depend on the scene $\mathcal{S}$ and the testing procedure *TP*, and they are evaluated during the execution phase as maximum values reached.

$N_G[-]$	–	maximum number of generic nodes in *DS*,
$N_E[-]$	–	maximum number of elementary nodes in *DS*,
$N_{EE}[-]$	–	maximum number of empty elementary nodes in *DS* ($N_{EE} < N_E$),
$N_{ER}[-]$	–	maximum number of the pointers to objects in all the elementary nodes of *DS* ($N_{ER} \geq N$).

- *RSA* parameters related to dynamic properties of data structure *DS*, *i.e.*, the use of *DS* within an execution phase of an *RSA*. The parameters depend on the scene $\mathcal{S}$ and the testing procedure *TP*, they are evaluated at the end of the execution phase:

$r_{ITM}[-]$	–	ratio of ray-object intersection tests performed to minimum number of intersection tests ($r_{ITM} \geq 1.0$, assuming at least one object is intersected given $\mathcal{S}$ and *TP*),
$\tilde{N}_{TS}[-]$	–	average number of all *DS* nodes accessed per ray ($\tilde{N}_{TS} \geq 1.0$),
$\tilde{N}_{ETS}[-]$	–	average number of elementary *DS* nodes accessed per ray ($\tilde{N}_{ETS} \leq \tilde{N}_{TS}$),
$\tilde{N}_{EETS}[-]$	–	average number of empty elementary *DS* nodes accessed per ray ($\tilde{N}_{EETS} \leq \tilde{N}_{ETS}$).

- *RSA* hardware/implementation/compiler dependent parameters. Obviously, these parameters also depend on the scene $\mathcal{S}$ and testing procedure *TP*:

$T_B[s]$	–	time to *build* initial *DS* for the *RSA* in the preprocessing phase,
$T_R[s]$	–	*running* time of the application that uses the *RSA*. It involves the execution phase of the *RSA* and possibly other computations. (Obviously, T_B is not included in T_R.)

We consider the parameters in the first two subsets as the minimum hardware/implementation/compiler independent parameters to be reported. It is certainly possible to extend the set of parameters by others (for example, the variance of number of objects in leaves), but we want to keep this set of the smallest possible size that still characterizes an *RSA* via the computation model.

The parameters T_B and T_R depend not only on the hardware used, but also on the quality of implementation (and programming language), the compiler used and its version, the optimization switches used for compilation, *etc*. For this reason, all these experimental conditions should be described in detail. The treatment of these parameters related to the implementation makes the problem of comparing various *RSAs* rather difficult; we describe our solution to the problem in more detail below.

2.3 *RSA* Performance Model

The *RSA* computation model enables us to count the number of basic algorithmic operations performed on average in an *RSA*. The *RSA* computation model does not define any *cost*[1] of these operations in terms of running time, it only covers the description of T_B and T_R.

In order to establish the relationship between hardware/implementation dependent and independent parameters, we further develop an *RSA performance model*, which separates the cost of a ray traversal algorithm and the cost of ray-object intersection tests. The concept of the performance model for an *RSA* was first introduced by Cleary and Wyvill [33] in the context of uniform grid analysis. We present here a more general performance model for any *RSA* that is derived from the *RSA* computation model described above. The *RSA* performance model is based on the decomposition of the running time T_R of the application that uses an *RSA* into three parts:

- computing ray-object intersection tests,
- traversing the *DS* of the *RSA*, and
- the remaining computation effort required by the application.

We bind the time-dependent and independent characteristics by means of cost consumed by specific algorithmic operations. Then we can express T_R as:

$$T_R = (r_{ITM}.r_{SI}.\tilde{C}_{IT} + \tilde{N}_{TS}.\tilde{C}_{TS}).N_{rays} + T_{app}, \qquad (2.1)$$

where r_{SI} is the ratio of the number of rays intersecting objects to the number of all rays ($r_{SI} < 1.0$), thus the average number of ray-object intersection tests per ray is $\tilde{N}_{IT} = r_{ITM}.r_{SI}$. Further, $\tilde{C}_{IT}[s]$ is the average cost of a ray-object intersection test, $\tilde{C}_{TS}[s]$ is the average cost of the traversal step of a ray traversal algorithm among the nodes of *DS*, N_{rays} is the total number of rays induced by a testing procedure *TP*, and $T_{app}[s]$ is the remaining time of the application. The time T_{app} covers another computation effort performed in the application, for example, in a rendering application T_{app} might cover the time consumed to compute the ray reflection, lighting, texturing, and other material calculations. Thus T_{app} is always constant for a particular scene $\mathcal{S}$ and testing procedure *TP*, provided the same implementation and hardware is used.

We can refine the performance model if we consider the ratio of successful ray-object intersection tests to all intersection tests:

$$T_R = [(\tilde{N}_{IT}^{succ}.\tilde{C}_{IT}^{succ} + \tilde{N}_{IT}^{fail}.\tilde{C}_{IT}^{fail}).r_{SI} + \tilde{N}_{TS}.\tilde{C}_{TS}].\tilde{N}_{rays} + T_{app}, \qquad (2.2)$$

where $\tilde{N}_{IT}^{succ}$ is the average number of successful ray-object intersection tests per ray, $\tilde{C}_{IT}^{succ}[s]$ is the average cost of successful ray-object intersection tests, $\tilde{N}_{IT}^{fail}$ is the average number of failed ray-object intersection tests per ray, and $\tilde{C}_{IT}^{fail}[s]$ is the average cost of failed ray-object intersection tests.

2.4 Ideal *RSA*

Having described the refined performance model, we can now introduce the *"ideal RSA"* as an *RSA* that has the best possible performance. The concept of the "ideal *RSA*" serves us as the ultimate but in

[1]For the sake of convenience, we further use the term *cost*[*s*] as the running time to perform some particular algorithmic operation.

practice unachievable goal. However, it is important since the running time of the “ideal *RSA*” is used as the reference time value for comparing various *RSAs*.

Definition 2 An “*ideal RSA* ” is an *RSA* that for a given ray shooting query computes the answer in $O(1)$ time independently of whether an intersected object exists or not. The multiplicative factor hidden behind the O-notation is very small.

Since Szirmay-Kalos and Márton [153] proved that any *RSA* works at least at time $\Omega(logN)$ in the worst-case, we can ask whether the definition of an “ideal *RSA*” makes sense. Inspired by the idea of *Parametrized Ray Tracing* [129], we can construct the “ideal *RSA*” provided the same testing procedure *TP* is repeatedly performed for the same scene $\mathcal{S}$. Further, it is required that the application code is deterministic in the sense that the testing procedure *TP* in the application always generates *the same sequence of ray shooting queries* for a given scene. This can require the setting of initial seeds in pseudo-random generators to the same value in the application, *etc.*

Further, we describe the two procedures that form the “ideal *RSA*”. The first assumption that enables us to execute the “ideal *RSA*” is that the application is run at least twice using the same *TP* and $\mathcal{S}$. Each object is assigned the identification tag ID (integer) in the range $\langle 0, N-1\rangle$. Then we construct the array A_T where objects are addressed directly using IDs of objects in $O(1)$ time.

In the first application run we use some conventional *RSA* to compute the answers to given ray shooting queries. The results obtained by the conventional *RSA* for the sequence of input ray shooting queries generated by *TP* are saved linearly to a temporary array A_S using the objects’ IDs. When no object is intersected, the corresponding array entry is set to a special ID value (ID$_{spec} = -1$). Since the number of ray shooting queries can be high, it may be necessary to save the results of the conventional *RSA* to external memory. The procedure that must be used in the first application run and the interface between the application and the conventional *RSA* is outlined in the pseudocode, Algorithm 3.

Algorithm 3 The first run of an “ideal *RSA*” that saves the results of ray shooting queries.

```
{Preprocessing phase}
Assign each object a unique ID in the range $\langle 0, N-1\rangle$.
Allocate the array $A_S$ to store IDs of objects, the number of entries in $A_S$ must be greater than or equal
to the number of all ray shooting queries generated by TP.
{The index into the array – order of ray shooting query}
$i \leftarrow 0$
{Execution phase}
function ShootRay(ray R): object
{Compute the result of the i-th ray shooting query by some other specific RSA.}
Compute the result for R using some conventional RSA.
Object $O \leftarrow$ the result of a conventional RSA for R
if object $O$ was found then
   $A_S[i] \leftarrow$ ID of object $O$
else
   $A_S[i] \leftarrow$ ID$_{spec}$
end if
$i \leftarrow i+1$
ShootRay $\leftarrow$ object $O$
{Postprocessing phase}
Possibly save $A_S$ to external memory.
```

In the second (repetitive) application run, instead of calling a specific *RSA*, we read the correct answer to the ray shooting query from the array A_S provided that repetitive run(s) of the application results in the same testing procedure *TP* and uses the same scene $\mathcal{S}$. If we get the object's valid ID, we get the address of the object through the array A_T and compute the ray-object intersection point exactly by at most one ray-object intersection test. This computation is required to get the correct signed distance for the current ray shooting query. If the object's ID has the value ID_{spec}, then the answer to the ray shooting query is "no object", and no ray-object intersection test is computed. Since the ray-object intersection test is computed at most once for each ray shooting query, the "ideal *RSA*" runs in $O(1)$ time. The "ideal *RSA*" performed in a repetitive run of the application is outlined in the pseudocode, Algorithm 4.

Algorithm 4 The second run of an "ideal *RSA*" reading the results of ray shooting queries.

```
{Preprocessing phase}
Assign each object its unique ID in the range $\langle 0, N-1 \rangle$.
{These IDs correspond to the first run of "ideal RSA".}
Allocate the array $A_S$ to store IDs of objects.
Possibly read $A_S$ from the external memory.
Allocate the array $A_T$ to store the pointers to objects, size of $A_T$ is the number of objects.
for each object $O$ is specified by its ID. do
    $A_T[\text{ID}] \leftarrow$ address of the object $O$
end for
{The index into the array – order of ray shooting query}
$i \leftarrow 0$
{Execution phase}
function ShootRay(ray R): object
ID of object $\leftarrow A_S[i]$
$i \leftarrow i+1$
if ID $\neq \text{ID}_{spec}$ then
    Object $O \leftarrow A_T[\text{ID}]$
    Compute the signed distance for the ray R and the object $O$.
else
    Object $O \leftarrow$ "no object"
end if
ShootRay $\leftarrow$ object $O$
```

For the repetitive run(s) of the "ideal *RSA*" the time T_R becomes the minimum application running time T_R^{MIN}:

$$T_R^{MIN}[s] = T_{RSA}^{MIN} + T_{app}, \tag{2.3}$$

where T_{RSA}^{MIN} is the minimum time devoted to ray shooting only, further called the *ideal ray shooting time*:

$$T_{RSA}^{MIN}[s] = \tilde{C}_{IT}^{succ}.N_{rays}.r_{SI} \tag{2.4}$$

If external memory is used to save the array A_S, we should avoid the time consumed to transfer the data from this external memory to internal memory to minimize the repetitive running time of the application T_R. Practically[2], array A_S is read from a file by blocks to internal memory, and the time for reading the blocks is not included in T_R^{MIN}.

[2]From the implementation point of view, the "ideal *RSA*" is fairly easy to implement in the application.

2.5 Minimum Testing Output

The results of experiments published in the papers introducing new *RSAs* were often restricted only to times T_B and T_R and some other parameters. Based on these hardware dependent parameters, we could not fairly compare newly introduced *RSAs* with those published in the past. It follows from the description of the computation and performance model that experiments allowing us to fairly compare various *RSAs* must be performed for the same scene $\mathcal{S}$ and testing procedure *TP*. For this purpose Haines introduced a *Standard Procedural Database* [69]. This database enables us to procedurally generate a set of scenes with various numbers of objects. It also defines some standard sizes of the scenes that should preferably be used for testing *RSAs*. However, the use of *SPD* scenes for testing *RSAs* has also been violated, and research papers often show results for testing performed on private scenes, or on only a small subset of *SPD* scenes. Such a researcher's behavior is a direct violation of research etiquette, since the nature of science is that every research paper should describe new techniques and experiments that will be *reproducible* and *verifiable* by all following researchers [37]. Therefore, whenever possible, qualitative properties of algorithms should always be tested on non-private input data.

Let us discuss why the comparison of various *RSAs* based only on time T_R consumed by the whole application is rather incorrect. The first reason is that T_R also includes T_{app}, which is constant. If we want to compare the ratio of performances of various *RSAs* on the same hardware and with the same implementation, instead of comparing T_R^1 for RSA^1 and T_R^2 for RSA^2 it is more correct to compare $(T_R^1 - T_{app})$ with $(T_R^2 - T_{app})$, since then we consider only the time consumed by the *RSA*. Obtaining the value of T_{app} can be difficult, as it usually requires profiling of the application by some software tool. We propose a way avoiding the use of a profiler in the section below. The value of T_R can be used correctly only for ranking of *RSAs*, but it cannot be used to express how much an *RSA* is faster than another *RSA*.

The *SPD* package [69] also recommends that some time-independent characteristics should be reported: N_{rays}, $\tilde{N}_{IT}^{succ}.N_{rays}$, $\tilde{N}_{TS}.N_{rays}$. We follow this approach by extending this set of hardware/implementation independent characteristics.

In order to avoid mutually contradictory statements in further papers concerning *RSAs*, we define a set of parameters to be reported from the experiments. We call the set of parameters the *minimum testing output*. This consists of three subsets as already presented: *RSA* parameters that relate to static properties of *DS*, *RSA* parameters that relate to dynamic use of *DS*, and *RSA* parameters dependent on hardware/implementation. The hardware/implementation dependent characteristics T_B and T_R are supplied by three other parameters. We normalize the time portions devoted to the particular phases to the ideal ray shooting time T_{RSA}^{MIN} to allow us to make a fair comparison among different implementations and different hardware used for testing. Our main goal is that the parameters in the minimum testing output should allow us to compare the performance of various *RSAs* independently of hardware and implementation issues. We define the minimum testing output for an *RSA* as:

subset Σ of parameters describing the *static* properties of a *DS* within the *RSA*:

$$\Sigma = \{N_G, N_E, N_{EE}, N_{ER}\},$$

subset Δ of parameters describing the *dynamic* use of the data structure *DS* within the *RSA*, which also depend on the input scene $\mathcal{S}$ and testing procedure *TP*:

$$\Delta = \{r_{ITM}, \tilde{N}_{TS}, \tilde{N}_{ETS}, \tilde{N}_{EETS}\},$$

and subset Θ of hardware/implementation/compiler dependent parameters of the *RSA* that concern running time, which also depend on the input scene $\mathcal{S}$ and testing procedure *TP*:

$$\begin{aligned}\Theta &= \{T_B, T_R, \Theta_{APP}, \Theta_{rat}, \Theta_{RUN}\} = \\ &\{T_B, T_R, \\ &\frac{T_{app}}{T_{RSA}^{MIN}}, \\ &\frac{N_{rays}.(\tilde{N}_{IT}^{succ}.\tilde{C}_{IT}^{succ} + \tilde{N}_{IT}^{fail}.\tilde{C}_{IT}^{fail})}{T_R - T_{app}}, \\ &\frac{T_R - T_{APP}}{T_{RSA}^{MIN}}\}.\end{aligned} \tag{2.5}$$

The parameter Θ_{APP} expresses the ratio of the remaining application time to the ideal ray shooting time T_{RSA}^{MIN}, which is not necessary for the comparison of *RSAs*, however, suitable for other reasons as we will show later. The parameter Θ_{rat} ($\Theta_{rat} \in \langle 0, 1 \rangle$) is the ratio of time required for computing the ray-object intersection tests to time consumed only by an *RSA*. The parameter Θ_{RUN} gives the ratio of time consumed by an *RSA* to T_{RSA}^{MIN}.

The time portions related to the ideal ray shooting time T_{RSA}^{MIN} can be difficult to measure. In the next section we deal further with this problem. The value of T_{RSA}^{MIN} enables us to compare different hardware/implementation/compiler dependent characteristics. Subset Θ contains the value of the ideal ray shooting time T_{RSA}^{MIN} only indirectly, since it can be computed as:

$$T_{RSA}^{MIN} = \frac{T_R}{\Theta_{APP} + \Theta_{RUN}} = \frac{T_R}{\frac{T_{app}}{T_{RSA}^{MIN}} + \frac{N_{rays}.\tilde{N}_{TS}.\tilde{C}_{TS}}{T_{RSA}^{MIN}} + \frac{N_{rays}.(\tilde{N}_{IT}^{succ}.\tilde{C}_{IT}^{succ} + \tilde{N}_{IT}^{fail}.\tilde{C}_{IT}^{fail})}{T_{RSA}^{MIN}}} \tag{2.6}$$

2.6 Measuring the Minimum Testing Output

The minimum testing output allow us to make a fair comparison of various *RSAs*. We pay for it by additional effort needed to get this set of thirteen parameters for one experiment. The counters to get the subsets Σ and Δ must be coded inside the *RSA* in its preprocessing and execution phase, which is fairly easy to implement. It is advantageous to check these counters for verification purposes as well, since they can indicate to us an implementation error of a particular *RSA*. In order to have a correct implementation of a particular *RSA* given some testing procedures *TP* and scene $\mathcal{S}$, the parameters N_{rays} and r_{SI} must have correct values when the application run is over. Although N_{rays} can be considered as an independent input quantity, it is often the case that the number of rays generated is connected with the use of an *RSA* and thus N_{rays} is dependent on the correctness of the *RSA*. For example, this is the case for higher order rays in various global illumination algorithms. Reference values of N_{rays} and r_{SI} can be obtained by running another *RSA* that is known to be correct. The simplest way is to implement naïve *RSA*, although the naïve *RSA* is inefficient.

To obtain subset Θ we need the running time T_R to be decomposed into the three portions: the time for the ray traversal algorithm performed within the *RSA*, the time of the ray-object intersection tests performed within the *RSA*, and the remaining application time T_{app}. There are two ways to obtain subset Θ; two profiling methods are described below.

2.6.1 Software Tool Profiling

One way to get subset Θ is to use a software *profiler tool.* This is a common method for solving performance issues in software applications. It enables us to distinguish the times consumed within particular software functional units, such as functions, procedures, or even the lines of a source code. Then we can sum the time devoted to ray-object intersection test routines, the time consumed by traversing the nodes of a *DS*, and the remaining application time.

We also require to profile the run of the application with "ideal *RSA*". In this case we need to sum only the time consumed by ray-object intersection tests, which gives us T_{RSA}^{MIN}.

Software tool profiling should be preferred for getting Θ, since it provides or at least should provide precise values. However, under certain conditions this is not possible, for one of the following reasons: the profiler is not available, the profiler does not work correctly, the profiler cannot determine the time portions of the required *RSA* parts within a given implementation, the profiler needs some compiler switches to be used, which influences T_R (a debugging switch is usually required, and this can increase the application time considerably) and the different time portions of T_R. Therefore we propose an alternative to obtain subset Θ without using a software profiler tool below.

2.6.2 Multiple Run Profiling

This profiling method involves running the application several times and computing the unknown variables in Eq. 2.2 from linear equations. Eq. 2.2 contains four unknown variables that express the costs of distinct algorithmic operations in some *RSA* application: $\tilde{C}_{IT}^{succ}$, $\tilde{C}_{IT}^{fail}$, $\tilde{C}_{TS}$, and T_{app},

In order to obtain the four unknown variables we need four different application runs. These have to use the same sequence of ray shooting queries, but they have to result in different total running times. For this purpose we utilize the concept of the "ideal *RSA*", and modify the ray-object intersection tests to be performed *K*-times. The first used equation comes from the common application run, described by Eq. 2.2. The second used equation is for the application run, when the ray-object intersection test is performed *K*-times, resulting in the running time:

$$T_R(K)[s] = (K.(\tilde{N}_{IT}^{succ}.\tilde{C}_{IT}^{succ} + \tilde{N}_{IT}^{fail}.\tilde{C}_{IT}^{fail}) + \tilde{N}_{TS}.\tilde{C}_{TS}).N_{rays} + T_{app}, \tag{2.7}$$

The third used equation is the time of the "ideal *RSA*", Eq. 2.3. The fourth used equation is for the case when the ray-object intersection test in the "ideal *RSA*" is performed *K*-times, resulting in the running time:

$$T_R^{MIN}(K)[s] = K.\tilde{C}_{IT}^{succ}.\tilde{N}_{rays}.r_{SI} + T_{app}, \tag{2.8}$$

The portions of time consumed for these four runs of *RSA* application is visualized in Fig. 2.1.

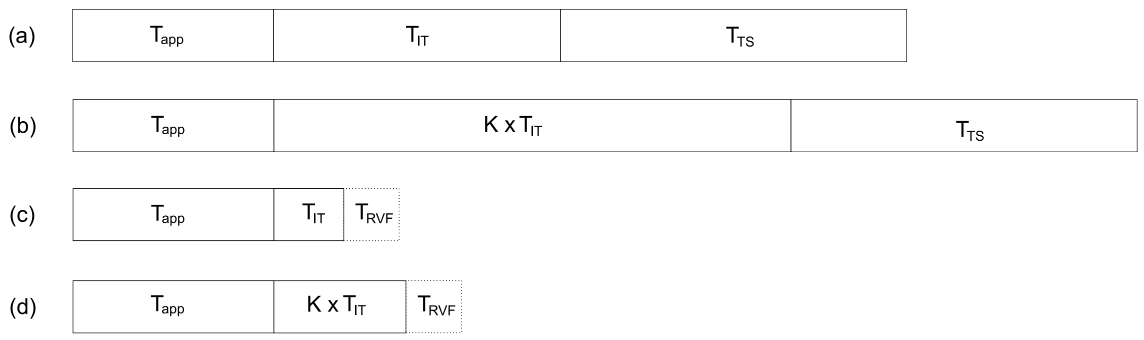


Figure 2.1: Visualisation of the running times for four runs of an *RSA* application. (a) Normal application run (Eq. 2.2). (b) Normal application run, ray-object intersection test *K*-times (Eq. 2.7). (d) "Ideal *RSA*" run (Eq. 2.3). (d) "Ideal *RSA*" run, ray-object intersection test *K*-times (Eq. 2.8). Notation: T_{IT} – time devoted to ray-object intersection tests, T_{TS} – time devoted to traversing, T_{app} – remaining application time, T_{RVF} – time required to read the visibility data from a file.

Assuming $\tilde{C}_{IT}^{succ}$, $\tilde{C}_{IT}^{fail}$, $\tilde{C}_{TS}$, and T_{app} are of the same value in these four application runs, we can compute these unknown variables by solving system of linear equations. From Eqs. 2.3 and 2.8, we get

$\tilde{C}_{IT}^{succ}$ and T_{app} as follows:

$$\tilde{C}_{IT}^{succ} = \frac{T_R^{MIN}(K) - T_R^{MIN}}{N_{rays}.(K-1).r_{SI}} \tag{2.9}$$

$$T_{app} = T_R^{MIN} - N_{rays}.r_{SI}.\tilde{C}_{IT}^{succ} \tag{2.10}$$

From Eqs. 2.2 and 2.7 we derive the $\tilde{C}_{IT}^{fail}$ and $\tilde{C}_{TS}$:

$$\tilde{C}_{IT}^{fail} = \left(\frac{T_R(K) - T_R}{N_{rays}.(K-1)} - \tilde{C}_{IT}^{succ}.\tilde{N}_{IT}^{succ}\right).\frac{1}{\tilde{N}_{IT}^{succ}} \tag{2.11}$$

$$\tilde{C}_{TS} = \left(\frac{T_R(K) - T_R}{N_{rays}} - \tilde{C}_{IT}^{succ}.\tilde{N}_{IT}^{succ} - \tilde{C}_{IT}^{fail}.\tilde{N}_{IT}^{fail}\right).\frac{1}{\tilde{N}_{TS}} \tag{2.12}$$

We call the profiling method based on the four equations *multiple run profiling.* It is oriented only to the software applications that use *RSA*.

2.6.2.1 Properties

Multiple run profiling has one big advantage, it does not require any software profiler tool. However, it suffers from several disadvantages. First, it requires the multiple ray-object intersection test to be implemented in routines for all the object types in the application. Second, the time of the multiple ray-object intersection test is affected by the cache behavior of the processor used during experiments. Even if the ray-object intersection test is performed K-times, instead of being K-times slower it is only K' times slower, where $0 < K' < K$. Further, we have found out during the testing of an "ideal *RSA*" that caching and branch prediction within the processor also influences the time of ray-object intersection tests within the "ideal *RSA*". Since in this case ray-object intersection test is always positive (Eqs. 2.3 and 2.8), the branches are always well predicted, resulting in lower cost $\tilde{C}_{IT}^{succ}$. Our experiments had best matching with the software tool profiling using $K = 2$. Third, the application must be run at least five times over the same input scene. In addition to the four application runs described above, one run is required to save the results of *RSA* into the visibility array A_S for the "ideal *RSA*" not to influence T_R in the application run corresponding to Eq. 2.2.

The second way is to get directly some other estimates of $\tilde{C}_{TS}$, $\tilde{C}_{IT}^{succ}$, $\tilde{C}_{IT}^{fail}$, and T_{APP}. Some of these may be known or well estimated for given hardware, implementation, and compiler independently of *TP* and $\mathcal{S}$ for some previous runs of the application. Provided $T_{APP} \gg T_{RSA}^{MIN}$, we can also assume $T_{APP} \doteq T_R^{MIN}$. Third, we can obtain the estimate for $\tilde{C}_{IT}^{succ}$ and $\tilde{C}_{IT}^{fail}$ when we use the same *RSA* with a different setting used for the construction of data structure underlying the *RSA*. For example, if a *kd*-tree is used, we can set the maximum depth allowed to various constants and then we get a set of equations of type 2.2, which allows us to compute $\tilde{C}_{TS}$.

Although multiple run profiling has several disadvantages, it remains the only known way when software tool profiling is not possible for some reason. Below, we improve this method using some correction parameters.

2.6.2.2 Corrected Measuring Subset Θ

To obtain more precise profiling results we can modify the equations of application runs to model the behavior of caching and branch prediction to some extent. We propose to use three correction parameters in this modified multiple run profiling. They express the time between the operation that is expected to be cached and the time of uncached operation: r_{corr}^{rep}, r_{corr}^{hit}, r_{corr}^{fail}, all of them in the range $(0.0, 1.0)$. First, we correct $\tilde{C}_{IT}$ provided that the ray-object intersection test always succeeds:

$$\tilde{C}_{IT}' = \tilde{C}_{IT}.r_{corr}^{rep} \tag{2.13}$$

Second, we correct $\tilde{C}_{IT}$ of the repetitive successful ray-object intersection test:

$$\tilde{C}'^{succ}_{IT}(K) = \tilde{C}_{IT}.(1+(K-1).r^{hit}_{corr}) \tag{2.14}$$

Third, we correct $\tilde{C}_{IT}$ of the repetitive failed ray-object intersection test:

$$\tilde{C}'^{fail}_{IT}(K) = \tilde{C}_{IT}.(1+(K-1).r^{fail}_{corr}) \tag{2.15}$$

Then we can express the corrected three equations as follows:

$$T^{MIN}_{R} = \tilde{C}^{succ}_{IT}.N_{rays}.r_{SI}.r^{rep}_{corr} + T_{app}, \tag{2.16}$$

$$\begin{aligned} T_R(K)[s] \quad = \quad & ((\tilde{N}^{succ}_{IT}.\tilde{C}^{succ}_{IT}.(1+(K-1).r^{hit}_{corr}))+ \\ & (\tilde{N}^{fail}_{IT}.\tilde{C}^{fail}_{IT}.(1+(K-1).r^{fail}_{corr}))+\tilde{N}_{TS}.\tilde{C}_{TS}).N_{rays}+T_{app}, \end{aligned} \tag{2.17}$$

and

$$T^{MIN}_{R}(K)[s] = \tilde{C}^{succ}_{IT}.r^{rep}_{corr}.(1+(K-1).r^{hit}_{corr}).N_{rays}.\tilde{n}_{IT} + T_{app}. \tag{2.18}$$

Similarly to Eqs. 2.9–2.12 we can derive the formulas to obtain $\tilde{C}_{TS}$, $\tilde{C}^{succ}_{IT}$, $\tilde{C}^{fail}_{IT}$, and T_{APP}. Corrected measuring is more precise, but it requires us to set the correction parameters. These can be estimated by using a software profiler when compiling without using optimization switches, or for a different setting of K. Another way is to use various ray traversal algorithms, since the correct setting of the correction parameters Θ_{IT} remains the same, because the number of ray-objects does not differ and Θ_{TS} depends on the ray traversal algorithm used.

2.7 Comparison Methodology

The establishment of the subsets Σ, Δ, and Θ of the minimum testing output enables us to compare different features of *RSAs*. For the use of an *RSA* in an application, when we are concerned in space and time complexity of the *RSA*, there are several different features for us to distinguish:

$\mathcal{S}$: the complexity of input scene $\mathcal{S}$ is important, for example, some *RSA* can be efficient for scenes with a small number of objects, although slow for scenes with a higher number of objects[3]. The scene influences all parameters in Σ, Δ, and Θ.

RSA: the idea behind *RSA* has a major impact on performance. *RSA* influences Σ, Δ, and Θ.

TP: the testing procedure is specific to the application used, and the use of *RSA* can vary greatly. It only influences Δ and Θ for an *RSA* based on static data structure, otherwise, it also influences Σ.

HW: type of hardware used – this influences all parameters in subset Θ, particularly T_B and T_R.

COMP: the compiler, its version, and the switches used can influence T_B and T_R significantly, and thus all parameters in subset Θ. (For example, setting optimization switch “-O2” of the C++ compiler in the UNIX operating system can decrease the running time by half compared with “-O0”.)

IMPL: implementation – the actual coding of the algorithm also has a great impact on performance, depending on the programmer’s experience, *etc.* Various implementations of the same ideas can exhibit significant differences in performance. It influences only subset Θ. When the *RSA* is (re)implemented correctly, the parameters in subsets Σ and Δ are not influenced.

[3]The notion of scene complexity is also a specific issue, dealt with in Section 3.3.

We note that *HW*, *COMP*, and *IMPL* can be intertwined to some extent, since a certain implementation can better fit to a certain hardware, *etc*. It is obvious that so many dimensions of freedom make the comparison of various *RSAs* rather difficult in general, especially for subset Θ. For example, if we want to compare two different *RSAs*, we have to fix as many other possible dimensions as possible, in this case $\mathcal{S}$, *TP*, *HW*, *COMP*, and *IMPL*. As the minimum requirement, we can require the same scene and the testing procedure within the application to be used. The existence of dimensions *HW*, *COMP*, and *IMPL* disable the direct use of T_B and T_R for comparing various *RSAs*. Some parameters in subset Σ, Δ, and Θ allow us to compare even such cases, due to the generality of the underlying *RSA* computation and performance model.

In general, we can perform the following comparisons for one experiment using the same *TP* and scene $\mathcal{S}$ for two ray shooting algorithms RSA^1 and RSA^2, where values for RSA^1 are denoted by superscript ^{1}, for RSA^2 by superscript ^{2}:

- memory complexity, we compare $(N_G^1 + N_E^1)$ with $(N_G^2 + N_E^2)$. To a constant factor given by an implementation of a particular *RSA*, it expresses the different memory requirements.
- use of hierarchy, we compare N_G^1/N_E^1 and N_G^2/N_E^2.
- use of empty space, we compare N_{EE}^1/N_E^1 and N_{EE}^2/N_E^2.
- time complexity, we have several choices depending on the conditions for comparison:
  - $T_R^1 - T_{app}^1$ with $T_R^2 - T_{app}^2$ for performance ratio, T_R^1 with T_R^2 for ranking only – time can be used directly for comparison, when *HW*, *COMP*, and *IMPL* attributes of *RSAs* to be compared are the same. For all experiments the *HW*/*COMP*/*IMPL* attributes must always be stated explicitly.
  - Θ_{RUN}^1 with Θ_{RUN}^2 – concerns the time required for ray shooting in the application related to ideal ray shooting time. Assuming that the implementation of ray-object intersection tests is practically the same, this enables a really fair comparison independent of *HW*, *COMP*, and *IMPL* attributes. The parameter Θ_{RUN} defines how far the tested *RSA* is from the "ideal *RSA*", and thus the maximum portion of the time that could possibly be reduced by some *RSA* with higher performance. It can be considered as the main index of the performance. Unlike comparing $T_R^1 - T_{app}^1$ with $T_R^2 - T_{app}^2$, it enables us to compare various different *RSAs* virtually independently of *HW*, *IMPL*, and *COMP*.
  - Θ_{rat}^1 with Θ_{rat}^2 – we can compare how much of the time for an *RSA* is devoted to computing ray-object intersection tests.
  - $\Theta_{RUN}^1.\Theta_{rat}^1$ with $\Theta_{RUN}^2.\Theta_{rat}^2$ – concerns the portion of time for ray-object intersection tests. It can be used virtually independently of *HW*, *COMP*, and *IMPL*.
  - $\Theta_{RUN}^1.(1 - \Theta_{rat}^1)$ with $\Theta_{RUN}^2.(1 - \Theta_{rat}^2)$ – concerns the portion of time for traversing and manipulating with data structures. It can be used virtually independently of *HW*, *COMP*, and *IMPL*.
  - r_{ITM}^1 with r_{ITM}^2 – an efficient *RSA* should have a ratio of ray-object intersection tests performed to the minimum number of intersection tests, as close to 1.0 as possible.
  - $\tilde{N}_{TS}^1$ with $\tilde{N}_{TS}^2$ – an efficient *RSA* has the number of traversal steps per ray as small as possible.
  - $\tilde{N}_{EETS}^1/\tilde{N}_{ETS}^1$ or $\tilde{N}_{EETS}^2/\tilde{N}_{ETS}^2$ – this shows us the utilization of empty space within the execution phase. It can have a great impact on *RSA* performance.

Based on these developments, we can formulate a *comparison methodology* for two or more *RSAs*. First, we map each tested *RSA* to the *RSA* computation model described in Section 2.2. When performing a set of experiments for a set of scenes we have to measure the minimum testing output for all the experiments. (The scenes should be publicly available, *SPD* scenes are suitable.) The testing procedure used within the application has to be the same for one scene and any *RSA* and must be well described. (Four testing procedures are described in the next chapter.) This guarantees the same sequence of ray shooting queries and thus the correctness and reproducibility of experiments. Then, we can compare various features of tested *RSAs* as described above, for each scene used and also as a whole set of scenes using basic statistics tools (*e.g.*, minimum, maximum, average, variance). It is necessary to report fully the minimum testing output for each scene and each experiment in the research work, for example, when introducing a new *RSA*.

2.8 Discussion

The proposed minimum testing output organized into three subsets has a total of thirteen parameters, which can be considered a high number. Nonetheless, we consider this as the minimum set of parameters that shows different features of an *RSA*, since it is based on the general computation model that fits any *RSA*. The minimum testing output contains both hardware/implementation independent and hardware/implementation dependent characteristics that allow us to make mutual comparisons of various *RSAs* under certain conditions. The disadvantage of this comparison methodology is the underlying assumption that the costs of the ray-object intersection tests are of the equal efficiency for various shapes of objects on different implementations. Fortunately, the ray-object intersection tests for objects' shapes in the *SPD* package are more or less standardized [3, 18, 88]. There is a set of standard scenes, and a well-defined testing procedure, namely ray tracing in the *SPD* package. However, we show that at least the same scene $\mathcal{S}$ and the same testing procedure *TP* must be used to validate the comparison.

Let us now discuss if it is possible to manipulate the minimum testing output by changing the quality of the implementation. It is not possible to influence the parameters in subsets Σ and Δ, assuming that the implementation of statistics counters and *RSA* itself is correct. If less efficient or more efficient ray-object intersection tests are applied, then practically all the parameters in Θ are influenced, excluding T_B. However, it is virtually impossible to influence the value of $\Theta_{RUN}.\Theta_{rat}$.

2.9 Conclusion

In this chapter we have shown the concept that is common for all *RSAs*, *i.e.*, an *RSA* computation model and performance model. We have described the "ideal *RSA*" that provides us with the reference value for comparing two *RSAs*. Further, we have presented a methodology for comparing various *RSAs*. However, after the analysis we performed it is clear that an experimental comparison of various *RSAs* still remains a difficult problem in general. The comparison methodology presented here enables us to compare various *RSAs*, assuming that each application uses the same sequence of ray shooting queries for the same set of objects. The construction of an "ideal *RSA*", which gives us the minimum time devoted to ray shooting only, also shows us the time in the best possible and ideal case given a hardware and implementation. The minimum application running time expresses the minimum time of a particular application that uses an "ideal *RSA*" for a given scene and testing procedure.

A by-product of this development is that we can measure how far we are from the minimum application running time ever achievable in dependence on the *RSA* used, given the application implementation that computes the set of ray shooting queries on the tested hardware. For example, it can then be shown whether or not it is possible to compute a particular global illumination task such as ray tracing in real time, given a certain hardware and a certain software implementation.

Chapter 3

Best Efficiency Ray Shooting Algorithm

In this chapter we present an experimental efficiency study of several *RSAs*. For this purpose we propose four testing procedures that induce various sets of ray shooting queries simulating the use of an *RSA* in global illumination algorithms. We use the testing procedures to produce statistics for comparison based on the developments put forward in the previous chapter. We compare various *RSAs* that have been reimplemented following the published literature. For reasons of the space we report only the main results of 1440 experiments for 30 *SPD* scenes and 12 *RSAs*.

3.1 Motivation

Looking at the literature addressing *RSAs* we find that many recent results are either limited to a subset of *SPD* scenes, or have been measured on private scene sets. Additionally, it is not known if *SPD* scenes – although they are scalable – is a good representative set of scenes suitable for testing various *RSAs*. The time complexity of most developed heuristic *RSAs* is unknown, or it is formally $O(1)$, which can be used for no valuable quantitative or qualitative comparison, since the timings for performing experiments vary greatly. Further, the same *RSA* on different hardware can have better or worse implementations. As a result of the factors mentioned above, many papers published about *RSAs* contain mutually contradictory statements.

Researchers involved in *RSAs* have tried long to find the *best efficiency RSA*, *i.e.*, an *RSA* that outperforms all other known *RSAs* for any set of ray shooting queries and any input scene. As might be supposed, no such *RSA* has been found to be generally the best until now. We propose an alternative in the search for the globally best-performing *RSA*: based on statistics provided by number of different *RSAs* tests we will try to find the *statistically best RSA*. The work presented in this chapter is part of an ongoing long-term undertaking called the BES (*Best Efficiency Scheme*) project, announced on the *globillum mailing list* [58] in October 1999 [79]. Owing to the long-term nature of the project we report in this thesis only the first results of experiments based on 30 *SPD* scenes of different complexity.

This chapter is further structured as follows: Section 3.2 gives more details on the goals of the BES project. Section 3.3 recalls the known scene complexity measures. In Section 3.4 we describe the design of the testing procedures used. Section 3.5 presents the results from experiments on 30 *SPD* scenes of different complexities and a possible interpretation of the obtained results with regard to scene complexity measures. Finally, Section 3.6 makes some conclusions, and proposes an outline for future work on BES.

3.2 Project Goals

The basic idea of the BES project is to collect a reasonable set of test scenes of varying complexity, and to use these scenes to measure the hardware/implementation/compiler independent characteristics of various *RSAs*. The collected scenes will not exhibit self-similar behavior exposed by the *SPD* scenes that are generated algorithmically. The scenes and measured results will be made available to the computer graphics community in a suitable form. The results of the project for collected scenes will enable us to evaluate the properties of *SPD* scenes for testing.

The BES project will help in clarifying the following points:

- *BES Existence.* The question whether a best efficiency *RSA* does or does not exist will be answered using a statistically relevant set of scenes. We suppose that the answer will be negative for currently known *RSAs*, but this still has to be verified.
- *Alternative BES Formulation.* A proposal for an alternative definition of a best efficiency scheme, based on hardware/implementation independent statistics for a relevant number of scenes, will be given.
- *BES Testing Procedures.* Simple testing procedures with moderate time requirements will be defined. These procedures can be used by other researchers to present the properties of a new *RSA*. These procedures will not be directly global-illumination algorithms requiring other computation than ray shooting – they will just induce different sets of ray shooting queries.
- *BES Comparison.* A concise summary and comparison of currently used *RSAs* will be provided. In order to minimise discrepancies, all tested *RSAs* are implemented within a uniform framework (GOLEM rendering system [75]).
- *BES Repository.* A collection of freely available test scenes of varying complexity will be made available for the scientific community. This can make future research in global illumination and the visibility field easier and more verifiable, as the lack of commonly used scenes makes it impossible to verify the results when reimplementing previously published methods for reference purposes.
- *BES Prediction.* We will check how the proposed definitions of scene complexity can help us to predict which *RSA* should be optimally selected in advance for a given scene. If such a prediction does not exist, the way to define such predictors will be opened.
- *Hybrid Methods.* We will try to find out if it pays off to construct hybrid spatial data structures for some spatial regions in a scene. The concept of *meta-hierarchies* [16] was defined ten years ago, but we are not aware of present-day implementations.

3.3 Scene Complexity

As we would like to be able to predict which *RSA* is the most appropriate for a given scene, we need a means characterising the scene with respect to running times for different *RSAs*. This can be accomplished by measuring different scene complexity characteristics and examining the correlation between the complexity characteristics and running times for different *RSAs* for a set of scenes. Then we can try to formulate an *RSA* selection algorithm which suggests the use of a particular *RSA* for a given scene using a certain complexity measure. The *RSA* selection algorithm is expected to select such a *RSA* that is likely to perform best of all *RSAs*.

There are several ways to estimate scene complexity. We can take into account, for example, the number of objects, scene sparseness, sparseness variance and standard deviation, non-uniformity, and so on. We have made use of several methods of evaluating scene characteristics that were introduced for this purpose [98, 27, 46].

3.3.1 Count Approach

The prevalent way to characterize scene complexity is to take the number of objects N in the scene, often referred to as *scene size*. Although this is a very simplistic definition of scene complexity – N

objects –, it raises the question, whether the use of this complexity measure is not exaggerated. To the best of our knowledge no answer to this problem has been given.

3.3.2 Voxelisation Approach

A method of scene characterization based on the presence of objects in voxels of a uniform grid has been proposed by Klimaszewski [99, chapter 4].

The resolution of the grid is selected according to the $O(\sqrt[3]{N})$ rule, using homogeneous method:

$$resolution_{\mathrm{x}} = resolution_{\mathrm{y}} = resolution_{\mathrm{z}} = \lfloor \sqrt[3]{d_{voxel} \cdot N} + 0.5 \rfloor, \tag{3.1}$$

where N is the number of objects and d_{voxel} is the voxel density. (It is usually assumed $d_{voxel} = 1.0$.) Then the number of voxels in the uniform grid is $N_V = resolution_{\mathrm{x}}.resolution_{\mathrm{y}}.resolution_{\mathrm{z}}$. The ratio between the number of empty voxels N_E to all voxels expresses the coefficient known as sparseness Ψ:

$$\Psi = \frac{N_E}{N_V} \tag{3.2}$$

The mean $\tilde{n}$ gives the average number of objects in a voxel:

$$\tilde{n} = \frac{1}{N_V} \sum_{i=1}^{N_V} n_i, \tag{3.3}$$

where n_i is the number of objects in the i-th voxel of a grid comprising N_V voxels. Variance v, the variability of the data around the mean, and standard deviation σ are given as:

$$v = \frac{1}{N_V - 1} \sum_{i=1}^{N_V} (n_i - \tilde{n})^2, \tag{3.4}$$

$$\sigma = \sqrt{v}. \tag{3.5}$$

To measure the unevenness of a distribution, the nonuniformity coefficient λ is used. The larger it is, the larger the disparities of voxel occupancy. λ is defined as:

$$\lambda = \sigma / \tilde{n} \tag{3.6}$$

Additionally, higher order moments reported are known as *skewness*:

$$s = 1/N_V . \sum_{i=1}^{N_V} (\frac{n_i - \tilde{n}}{\sigma})^3, \tag{3.7}$$

and *kurtosis*:

$$k = [1/N_V . \sum_{i=1}^{N_V} (\frac{n_i - \tilde{n}}{\sigma})^4] - 3, \tag{3.8}$$

The objects should be assigned to the voxels using the intersection of the object surface with the voxel, not the intersection of the object's $\mathcal{AB}$ with the voxel.

3.3.3 Integral Geometry Approach

Cazals and Sbert [27] investigated several integral geometry tools that characterize average case scene properties. Their strategy consisted in probing the scene with random entities (lines and planes) paying special attention to those statistics that may reveal the spatial distribution of scene objects. A *global line* in this context is a ray with its origin outside the scene. When shooting a global line, the goal is to find out not only the closest intersected object, but all objects intersected along the ray path.

We selected all the random line-based tests for our experiments, which allowed us to determine the following characteristics: average number of intersection points for a global line crossing the scene n_{int}^G, probability of not intersecting any object in the scene p_0, and relative average length of a line span that lies in free space s_{len}.

3.3.3.1 Average Number of Intersection Points

When casting a global line through the whole scene, an average number of intersections with scene objects n^G_{int} may be determined a priori as:

$$n^G_{int} = \frac{2.SA_{total}}{SA(\mathcal{AB}(\mathcal{S}))}, \quad SA_{total} = \sum_{i=1}^{N} SA(O_i), \tag{3.9}$$

where $SA(\mathcal{AB}(\mathcal{S}))$ is the surface area of a tight scene axis-aligned bounding box and $SA(O_i)$ is the surface area of the i-th object in the scene. A posteriori, we can compute this characteristics by casting n global lines to the scene:

$$en^G_{int} = \frac{1}{n}\sum_{i=1}^{n} int_i, \tag{3.10}$$

where int_i is the number of intersections with objects for i-th global line. Standard deviation σen^G_{int} of this characteristics is also reported in [27] for casting the set of global lines.

3.3.3.2 Probability of Zero Intersections

The probability of a ray not intersecting any object in the scene, denoted p_0, is quite hard to determine analytically. As we have to cast global lines anyway in order to compute other complexity characteristics, we compute the probability as:

$$p_0 = \frac{n_0}{n_{total}}, \tag{3.11}$$

where n_{total} is the total number of global lines cast and n_0 is the number of global lines that did not intersect any object in the scene.

3.3.3.3 Free Path Statistics

While tracing a global line through a scene, every intersection adds an additional “span” to the traced line. The average length of spans may give us an insight into the spatial density of the scene under investigation. For every global line n_i cast we sum up the span lengths l_k and also identify the maximum span length l_{max}. For the total of n_{spans} spans the free path statistic is then given as:

$$s_{len} = \frac{1}{n_{spans}\cdot l_{max}}\sum_{i=1}^{n_{spans}} l_i. \tag{3.12}$$

Similarly to en^G_{int}, standard deviation for free path statistics σs_{len} is in [27] also reported from the experiments.

3.3.4 Information Theory Approach

Feixas *et al.* [46] describe scene complexity as a task of determining the mutual information transfer. In their paper they present a number of complexity measures from information theory quantifying how difficult it is to compute visibility in the scene accurately. While working with a scene discretised into patches, the paper also contains a definition of scene continuous mutual information, which is mutual information independent of any discretisation of the scene. We have used continuous mutual information I^C_S to characterize our scenes.

The scene continuous mutual visibility information can easily be determined using Monte-Carlo integration with global lines. The whole quite involved formula boils down to:

$$I^C_S \approx \frac{1}{n_{PP}}\sum_{i=1}^{n_{PP}} \log\left(\frac{SA_{total}\cos\theta_{\mathrm{U}}\cos\theta_{\mathrm{V}}}{\pi d(U,V)^2}\right), \tag{3.13}$$

where n_{PP} stands for the total number of point pairs observed, SA_{total} is the total scene surface area, (U,V) is the point pair under investigation, $d(U,V)$ is the distance of those two points and θ_U, θ_V are the angles between the direction vector and the corresponding normal vector to the objects' surface at U and V. As objects in the *SPD* scenes used for tests here do not overlap, for the purposes of testing in this chapter we have $SA_{total} = \sum_{i=1}^{N} SA(O_i)$, where $SA(O_i)$ is the surface area of *i*-th object in the scene. In the case of more complex geometry (CSG objects), the stochastic area estimation method proposed by Wilkie *et al.* [161] can be used.

3.4 Testing Procedures

The design goal of the *RSA testing procedures* is to emulate the use of ray shooting in rendering algorithms. For the design we prefer to take only surface geometry into account and do not perform any lighting or material calculations. This way most of the computation time in the tests is really devoted to ray shooting, which is the subject of our main concern.

We propose four different testing procedures. The first three tests are general methods simulating the use of ray shooting in global illumination rendering algorithms. The last testing procedure is ray tracing an image as defined in *SPD* as an example of simple and real rendering task. We do not use ray tracing just to get a visually appealing image; it also allows us to find errors when implementing a new *RSA*, and it also allows subjective evaluation of scene properties.

When a test scene has quite a varying object distribution, performing ray shooting tests for a single origin of a ray located somewhere in the scene may reveal only local properties of the tested *RSA*. In order to test *RSA* behavior over the whole scene, the use of uniformly distributed global rays is more appropriate. In order to obtain equal ray distributions for all tested *RSAs* the same initial seeds for random number generators have to be used in each testing procedure.

The usual way of generating global lines is to generate two uniformly distributed points on the scene bounding sphere and to shoot the ray between these two points. This method would however be unfair to those *RSAs* that use a ray space subdivision method, assuming the successive ray queries are somehow similar [17, 132], see Section 1.6.4. Algorithm 5 shows an alternative uniform global ray generation scheme that preserves the ray coherency of subsequent rays at the same time. The basic idea of the algorithm is to split a sphere into the B bands of the same width (and thus the same surface area) and assign the same number of points to each band. This preserves the same surface area is also assigned to each point. Then casting the global lines among all pairs of points has uniform line density.

3.4.1 Definition of Testing Procedures

The first of four testing procedures, denoted TP_A, test the properties of *RSA* for rays distributed in the whole scene. The second testing procedure TP_B test the properties of *RSA* in the space, where the most objects are present. The third testing procedure TP_C simulates a random walk in the space were most of the scene objects are present. The last testing procedure TP_D is the already mentioned ray-tracing according to *SPD*. The testing procedures are designed as follows:

Testing procedure TP_A: Shoots only primary rays that are generated using Algorithm 5 on a sphere circumscribed to the $\mathcal{AB}$ of the scene. The $\mathcal{AB}$ of the scene is computed as a union of the $\mathcal{AB}$*s* of all objects in the scene.

Testing procedure TP_B: Assigns a tight $\mathcal{AB}(O_i)$ to each object O_i in the scene. For each $\mathcal{AB}(O_i)$ computes its center point Q_i and computes the center of the sphere as $Q = 1/N \cdot \sum_{i=1}^{N} Q_i$. Using a binary search finds the minimum radius of a sphere with center Q containing 90% of all Q_i. Shoots only primary rays generated on this sphere using Algorithm 5.

Testing procedure TP_C: The same as TP_B, but the rays are randomly reflected using uniform distribution over the hemisphere given by the surface normal at the hit point. The bounces continue until

Algorithm 5 Systematically casting $n \cdot (n-1)$ rays uniformly distributed on sphere.

```
{Compute n points uniformly distributed on the unit sphere.}
B ← number of bands
Subdivide sphere by $(B-1)$ planes $z = const$ $\{z_i = 1-2(i+1)/B, i \in \{0, B-2\}\}$.
{Create B bands having the same surface area on the sphere.}
$b \leftarrow 0$ {current band index}
$SA_{one} \leftarrow 4/3 \cdot \pi/n$ {required surface area of one region}
$\alpha \leftarrow 0$ {the end angle of the current region}
for all $i \in n$ points do
  $SA_{curr} \leftarrow 0$ {Current region has zero area. Always generate point at the end of a new region.}
  Integrate bands by increasing $\alpha$ or/and $b$ until $SA_{curr} = S_{one}$.
  $\vec{P}_i.z = 0.5 \cdot (1-(1-2i)/n)$
  $R = \sqrt{1.0 - (\vec{P}_i.z)^2}$, $\vec{P}_i.x = R\cos(\alpha_i)$, $\vec{P}_i.y = R\sin(\alpha_i)$
end for
Transform $n$ points to world space given the sphere center $\vec{C}$ and radius $R$.
for all $i \in n$ points do
  for all $j \in n$ points do
    if $i \neq j$ then
      {For $TP_A$, $TP_B$, and $TP_C$.}
      Shoot (primary) ray between points $i$ and $j$ on the sphere in the world space.
      {Only for $TP_C$ spawn higher order rays.}
    end if
  end for
end for
```

the maximum depth of recursion $d_{rec} = 4$ is reached (primary rays have $d_{rec} = 0$) or until the ray leaves the scene.

Testing procedure TP_D: Recursive ray tracing exactly as defined in *SPD*. This task requires a camera to be set and surface materials to be defined. Depth of recursion is the same as for TP_C, and the number of primary rays cast is 513×513. All other details can be found in the `Readme.txt` file in the *SPD* distribution [69].

Obviously, it is always possible to construct an artificial scene where our testing procedures will not fulfill the proposed goal. However, scenes that are used for practical purposes will not pose difficulties to our tests.

3.4.2 Invariants

Given a testing procedure *TP* and a scene $\mathcal{S}$ we can determine a set of parameters for every tested scene that will remain constant regardless of the *RSA* used. These *invariants* can be used to verify the results of *RSA* implementation. However, we will be aware that this verification does not impose a globally correct *RSA* implementation, it merely proves that the results are correct for the particular scene. The invariants are:

$N^{hit}_{\mathcal{AB}}$:	number of primary rays hitting the scene axis-aligned bounding box (applies for TP_A-TP_D),
N^{hit}_{prim}:	number of primary rays hitting any object (applies for TP_A-TP_D),
N_{sec}:	number of secondary rays (reflected rays for TP_C, reflected and refracted rays for TP_D),
N^{hit}_{sec}:	number of secondary rays hitting any object (reflected rays for TP_C, reflected and refracted rays for TP_D),
N_{shad}:	number of shadow rays (applies for TP_D only), and
N^{hit}_{shad}:	number of shadow rays hitting opaque objects (TP_D only).

In practice we observed that the invariants are equal or that they differ just very slightly due to numerical precision problems. The relative error for shooting 10^6 rays using single-precision floating point arithmetic was always below $\varepsilon = 10^{-4}$ in our experiments, which is acceptable under the assumption that no differences between images obtained as a result of TP_D for different *RSAs* are visible. Given the finite precision arithmetic, we should consider that tuning an *RSA* implementation to get exactly the same results can even be impossible to achieve, since the *RSA* need not be robust to numerical imprecision.

3.5 Results and Discussion

In this section we describe the scenes used for the experiments, their scene complexity measures, and the results for all testing procedures.

3.5.1 Test Scenes

The process of collecting and preparing test scenes started in October 1999 and now in November 2000 is still under progress. We decided to group the collected scenes according to the number of objects, creating 7 groups: G^X, $X \in 0 \ldots 5$, each G^X containing 15 scenes with $\langle 10^X + 1,\, 10^{X+1} \rangle$ objects, and G^6, 10 scenes with more than 10^6 objects, which is 100 scenes in total.

There are many WWW sites offering 3D models usable for our purposes, but we have encountered two problems: First, models are usually available in proprietary formats and conversion into open formats (VRML'97 [2] in our case) does not usually work very well. This results in scenes having corrupted faces, invalid normals, missing textures, and so on. Second, one can never estimate the scene size before actually downloading the model. We observed that most of the 215 scenes downloaded until now typically contain $5 \cdot 10^3$–$5 \cdot 10^4$ objects. We did not found any suitable scenes having less than 100 or more than $5 \cdot 10^5$ objects on WWW. While small scenes can be modeled, composing meaningful large scenes is quite a demanding task. As a result, groups supposed to contain scenes with higher numbers of objects are still incomplete. Therefore, the project cannot progress at present.

In the experiments presented here we have used three groups of *SPD* scenes with different object counts. Since *SPD* scenes are scalable, we decided to generate individual scenes with object counts as close as possible to maximum counts required for scene size: group G^3_{SPD} (10^3 objects), G^4_{SPD} (10^4 objects)[1], and G^5_{SPD} (10^5 objects). The scene size in generated *SPD* scenes depends on an optional size factor S_F, where the ratio of scene sizes for S_F and $(S_F + 1)$ varies from 1.2 for "teapot" to 8.0 for "jacks". Table 3.1 lists one scene size in the *SPD* scenes for size factor $S_F = 1 \ldots 6$. The bold typeset numbers of objects denote scenes selected into groups: G^3_{SPD}, G^4_{SPD}, and G^5_{SPD}. A *SPD* scene further in the thesis is referred to as "sceneX", where X is the value of the size factor. We decided not to use the scene entitled "shells" for our experiments, as this is the only *SPD* scene containing densely overlapped objects, which should not be the case for a correctly modeled scene. Naturally, such a scene causes problems for any *RSA* that we tested.

As soon as the *RSA* characteristics measured on the downloaded scenes are available, they can be compared to data measured on these *SPD* groups. This way we can verify the suitability of *SPD* scenes for testing *RSAs*.

Table 3.2 shows selected scene complexity measures for the 30 *SPD* scenes. Comparing the computed complexities with the testing results presented below, it unfortunately seems that there is no direct correlation between existing complexity measures and *RSA* performance.

3.5.2 Results

We implemented the following *RSAs*, which are all based on the static data structures:

[1]When *SPD* scene are used for testing an *RSA*, G^4_{SPD} corresponds to the standard set of scenes that is used for testing.

	Scene									
S_F	balls	gears	jacks	lattice	mount	rings	sombrero	teapot	tetra	tree
1	11	147	9	20	12	61	**1922**	57	4	7
2	92	**1169**	81	81	36	301	**7938**	244	16	15
3	**821**	3943	**657**	208	132	**841**	32258	561	64	31
4	**7382**	**9345**	**5265**	425	**516**	1801	**130050**	**1008**	256	63
5	**66341**	18251	**42129**	756	2052	3301	522242	1585	**1024**	127
6	597876	31537	337041	**1255**	**8196**	5461	–	2292	**4096**	255

Table 3.1: Number of objects of *SPD* scenes according to the size factor S_F.

		Complexity														
SceneX	Group	N	SA_{total}	Ψ	$\tilde{n}$	σ	λ	s	k	n_{int}^G	en_{int}^G	σen_{int}^G	p_0	s_{len}	σs_{len}	I_S^C
balls3	G_{SPD}^3	821	588.57	0.49	1.72	12.23	7.12	10.01	99.91	0.92	0.92	0.43	0.995	0.040	1.16	10.194
balls4	G_{SPD}^4	7382	591.71	0.97	1.28	19.20	15.00	17.68	344.85	0.92	0.92	0.47	0.995	0.036	1.12	11.167
balls5	G_{SPD}^5	66431	594.85	0.99	1.01	23.20	22.91	26.68	780.77	0.93	0.93	0.51	0.994	0.033	1.09	11.518
gears2	G_{SPD}^3	1169	32.61	0.81	1.58	5.01	3.18	3.89	15.47	1.36	1.36	1.65	0.787	0.110	0.40	4.833
gears4	G_{SPD}^4	9345	59.61	0.78	2.27	6.08	2.68	2.88	7.49	2.48	2.48	3.65	0.723	0.060	0.23	9.003
gears9	G_{SPD}^5	106435	126.16	0.76	2.82	6.70	2.37	2.40	4.62	5.25	5.25	8.46	0.691	0.025	0.14	12.728
jacks3	G_{SPD}^3	657	26.72	0.47	2.84	3.37	1.19	0.80	-0.76	1.32	0.95	1.72	0.749	0.092	0.31	4.512
jacks4	G_{SPD}^4	5265	57.26	0.58	2.63	3.61	1.37	0.96	-0.61	2.50	1.78	2.89	0.661	0.078	0.27	6.843
jacks5	G_{SPD}^5	42129	118.33	0.63	2.58	3.76	1.45	1.01	-0.58	4.87	3.34	5.05	0.604	0.064	0.19	9.437
lattice6	G_{SPD}^3	1225	14.72	0.00	2.33	1.35	0.58	0.96	-0.20	4.17	3.13	2.63	0.299	0.086	0.18	5.471
lattice12	G_{SPD}^4	8281	24.47	0.00	3.69	1.68	0.45	0.03	-1.12	7.49	5.48	4.07	0.172	0.069	0.13	7.581
lattice29	G_{SPD}^5	105300	52.73	0.02	3.59	1.68	0.47	0.10	-1.01	16.92	11.42	7.65	0.087	0.046	0.07	10.819
mount4	G_{SPD}^3	516	9.25	0.65	3.06	5.29	1.73	1.68	1.73	0.68	0.68	1.09	0.865	0.144	0.38	5.636
mount6	G_{SPD}^4	8196	10.16	0.84	2.36	6.89	2.91	3.24	10.34	0.74	0.74	1.23	0.849	0.119	0.36	6.193
mount8	G_{SPD}^5	131076	10.52	0.93	1.88	9.38	4.99	5.76	35.20	0.79	0.79	1.51	0.847	0.095	0.32	7.649
rings3	G_{SPD}^3	841	362.58	0.69	2.40	5.41	2.25	2.44	5.23	1.16	0.82	1.38	0.867	0.119	1.70	3.763
rings7	G_{SPD}^4	8401	2817.90	0.72	2.69	5.46	2.03	1.94	2.45	2.46	1.53	2.62	0.748	0.071	2.12	6.718
rings17	G_{SPD}^5	107101	32717.00	0.74	2.43	4.93	2.03	2.00	2.99	5.95	3.47	5.57	0.642	0.043	2.56	10.412
sombrero1	G_{SPD}^3	1922	72.86	0.70	2.78	4.60	1.65	1.31	0.35	0.81	0.81	0.78	0.966	0.104	0.43	5.245
sombrero2	G_{SPD}^4	7938	73.51	0.80	2.44	5.32	2.19	1.93	2.07	0.82	0.82	0.80	0.963	0.114	0.42	4.858
sombrero4	G_{SPD}^5	130050	73.70	0.92	1.91	6.96	3.64	3.60	11.40	0.82	0.82	0.79	0.962	0.117	0.42	4.622
teapot4	G_{SPD}^3	1008	115.56	0.64	3.05	7.85	2.58	4.85	36.36	1.01	1.01	1.11	0.795	0.232	1.19	7.101
teapot12	G_{SPD}^4	9264	116.73	0.85	2.20	9.74	4.43	9.21	125.22	1.02	1.02	1.13	0.791	0.255	1.20	7.408
teapot40	G_{SPD}^5	103680	116.87	0.92	1.78	13.19	7.42	19.40	625.33	1.02	1.02	1.13	0.791	0.255	1.20	7.382
tetra5	G_{SPD}^3	1024	13.86	0.63	3.27	5.11	1.56	1.44	1.03	1.15	1.15	1.75	0.610	0.059	0.25	5.006
tetra6	G_{SPD}^3	4096	13.86	0.76	2.74	5.70	2.08	2.15	3.47	1.15	1.15	1.89	0.638	0.043	0.24	6.205
tetra8	G_{SPD}^5	65536	13.86	0.90	1.99	7.02	3.52	4.12	18.44	1.15	1.15	2.14	0.681	0.030	0.22	8.747
tree8	G_{SPD}^3	1023	10006.00	0.50	1.53	23.04	15.02	22.54	509.36	0.94	0.94	0.23	1.000	0.087	7.05	7.220
tree11	G_{SPD}^4	8191	10007.00	0.67	1.37	42.97	31.41	45.86	2188.41	0.94	0.94	0.24	1.000	0.091	8.04	7.548
tree15	G_{SPD}^5	131071	10008.00	0.80	1.60	100.10	62.59	87.38	8379.39	0.94	0.94	0.24	1.000	0.086	7.74	7.587

Table 3.2: Scene complexity according to Section 3.3 for G_{SPD}^3, G_{SPD}^4, and G_{SPD}^5 scenes.

BSP: The BSP tree [94] using an efficient recursive ray traversal algorithm [82]. The splitting plane always creates equally-sized children. The maximum allowed depth was 16, maximally 2 objects were allowed in a node (see Subsubsection 1.6.3.1),

KD: The *kd*-tree, similar to the BSP tree, but the splitting plane is put according to the surface area heuristic [105]. The construction of the *kd*-tree is discussed in the following chapter. The same termination criteria as for the BSP tree were used,

UG: The uniform grid [53], with resolution according to [85] (Woo's method) with voxel density $d_{voxel} = 3.0$ (see Subsubsection 1.6.3.3),

BVH: The bounding volume hierarchy built with cost function [63] (see Subsection 1.6.2),

AG: The adaptive grid, BVH over uniform grids [100] (see Subsubsection 1.6.3.4),

RG: The recursive grid, a grid recursively put in the parent grid voxels again [93] (see Subsubsection 1.6.3.4), *HUG:* The hierarchy of uniform grids [26] (see Subsubsection 1.6.3.4).

O84: The octree with a sequential ray traversal algorithm built using midpoint subdivision [59] (see Subsubsection 1.6.3.2),

O84A: The octree-R using surface area heuristic [159] with the sequential ray traversal algorithm [59] (see Subsubsection 1.6.3.2),

O89: The octree using neighbor finding for the ray traversal algorithm [126] using midpoint subdivision (see Subsubsection 1.6.3.2),

O93: The octree using the recursive ray traversal algorithm [54] (see Subsubsection 1.6.3.2),

O93A: The octree-R using surface area heuristic [159] with the recursive ray traversal algorithm [54] (see Subsubsection 1.6.3.2).

The detailed description of parameter settings to build up underlying data structures of all *RSAs* is beyond the scope of the thesis, since it supposes detailed knowledge of all *RSAs*. We have consistently used the best settings that we found during previous experiments [85, 74], with one exception – if we ran out of memory, we allowed three iterations of modifying *RSA* construction parameter settings to require less memory and then tested again. This occurred for RG, AG, and HUG. Failures of this iterative parameter settings are reported in Table 3.4. There are two cases when results are not reported: either the testing procedure did not finish in 10 hours, or the computer memory was still exhausted even after three iterations of setting parameters for construction. It should be clear that manual tuning of these parameters to construct failure-proof data structures for all 12 given *RSAs* and 30 scenes is quite impractical.

All the tests presented in this chapter were conducted on PCs running Linux, kernel version 2.2.12-20, processor Intel Pentium II, 350 MHz, 128 MB RAM. Test program in the GOLEM rendering system was compiled using egcs-1.1.2 with "-O2" optimization switch. The total number of experiments was 1440 (12 *RSAs* by 30 scenes by 4 testing procedures). With 10 reportable parameters for every experiment (see Section 2.5) we measured 11520 hardware/implementation independent and 2880 hardware/implementation dependent *RSA* parameters. We omitted to find out the values Θ_A, Θ_{IT}, and Θ_{TS} due to the time requirements that are necessary to get these values, since we are satisfied with ranking of tested *RSAs*.

Due to space limitations it is not possible to report all the results from experiments here. We have therefore selected the main characteristics from all the experiments and we present here a short summary, the results for TP_D are given in Appendix E, lines 48–58. All measured statistics are available on the WWW site of the BES project [79].

Table 3.3 reports for each tested scene the time T_B needed to build the underlying data structure for the fastest *RSA* and the minimum time T_R over all *RSAs* needed to run all testing procedures. Table 3.4 reports the average running time $\tilde{T}_R$ for a given *RSA* and testing procedure for all the scenes and summary times for columns and rows. The parameter m is the number of tasks where experiment failed due to memory limits, f denotes the number of cases when tests were not finished within the time limit. The *RSAs* are sorted into $\tilde{T}_R$ for the total sum including all testing procedures. We can see that the winner in the tests on *SPD* scenes is the *RSA* based on the *kd*-tree, while the *RSA* based on the BVH has the worst average running time, being in some tests even more than two orders of magnitude slower than the former.

The total running time of the whole experiment was about 400 hours on a single processor. Tests TP_A–TP_C used 10^6 primary rays (exactly $1009.1008 = 1017072$ rays), and the number of bands in Algorithm 5 was 8161. The graphs in Fig. 3.1 and Fig. 3.2 show a summary of the hardware/independent parameters. Parameters for each *RSA* are summed over all tested scenes and testing procedures, and are normalized to the highest value for all *RSAs*. The graph in Fig. 3.3 shows for each *RSA* its total running

SceneX	Testing Procedure											
	TP_A			TP_B			TP_C			TP_D		
	RSA	$T_B[s]$	$T_R[s]$	*RSA*	$T_B[s]$	$T_R[s]$	*RSA*	$T_B[s]$	$T_R[s]$	*RSA*	$T_B[s]$	$T_R[s]$
balls3	KD	0.30	3.72	KD	0.30	13.63	KD	0.30	50.25	KD	0.30	21.4
balls4	KD	1.75	3.51	KD	1.75	17.64	KD	1.75	62.87	KD	1.75	27.06
balls5	KD	16.2	3.82	KD	16.2	31.41	KD	16.2	102.2	KD	16.2	42.0
gears2	KD	0.6	5.01	KD	0.6	10.1	KD	0.6	34.24	KD	0.6	38.96
gears4	KD	2.45	5.71	KD	2.45	12.94	KD	2.45	57.49	KD	2.45	36.24
gears9	KD	22.79	7.91	UG	7.24	18.36	KD	21.89	91.01	KD	22.97	40.59
jacks3	UG	0.04	12.59	UG	0.04	30.51	UG	0.04	74.05	UG	0.04	10.46
jacks4	UG	0.3	16.38	UG	0.3	38.87	UG	0.3	118.8	UG	0.3	19.82
jacks5	KD	12.68	21.76	UG	2.9	46.45	UG	2.9	167.4	UG	2.9	30.69
lattice6	UG	0.1	9.47	UG	0.1	13.36	UG	0.1	48.56	UG	0.1	33.24
lattice12	UG	0.6	12.02	UG	0.6	16.88	UG	0.6	72.96	UG	0.6	39.95
lattice29	UG	8.6	14.53	UG	8.6	19.35	UG	8.6	93.45	UG	8.6	43.65
mount4	KD	0.09	6.83	KD	0.09	11.18	KD	0.09	26.65	KD	0.09	18.88
mount6	KD	1.49	9.11	KD	1.49	15.71	KD	1.49	39.3	KD	1.49	21.06
mount8	KD	25.9	18.45	KD	25.9	37.07	KD	26.9	114.7	KD	25.82	25.14
rings3	KD	0.47	8.14	KD	0.47	30.01	KD	0.47	80.93	KD	0.47	40.39
rings7	KD	2.36	12.61	KD	2.36	38.09	KD	2.36	139.5	KD	2.36	64.76
rings17	KD	23.55	22.23	KD	23.55	61.13	KD	23.55	260.4	KD	23.55	106.6
sombrero1	KD	0.32	6.18	KD	0.3	8.00	KD	0.31	18.21	KD	0.33	3.82
sombrero2	KD	1.37	6.82	KD	1.37	9.16	KD	1.37	20.99	KD	1.37	4.0
sombrero4	KD	26.39	11.83	KD	26.39	16.88	KD	26.39	40.61	KD	26.39	6.9
teapot4	KD	0.38	5.65	KD	0.38	11.56	KD	0.38	35.25	KD	0.38	13.94
teapot12	KD	2.18	6.65	KD	2.18	13.89	KD	2.18	43.12	KD	2.18	15.66
teapot40	KD	22.4	9.54	KD	22.4	23.46	KD	22.4	74.58	KD	22.39	23.85
tetra5	KD	0.1	6.6	KD	0.1	9.33	KD	0.1	19.6	KD	0.1	2.48
tetra6	KD	0.49	7.74	KD	0.49	10.9	KD	0.47	21.96	KD	0.47	2.66
tetra8	KD	10.9	13.69	KD	10.9	19.47	KD	10.9	36.65	KD	10.9	3.57
tree8	RG	0.13	3.72	KD	0.34	20.66	KD	0.34	34.23	KD	0.34	18.39
tree11	AG	1.75	3.82	AG	1.8	29.96	KD	2.04	47.28	AG	1.8	20.61
tree15	RG	358.5	3.72	HUG	10.4	76.37	AG	380.0	68.71	AG	380.0	43.38

Table 3.3: The *RSAs* with minimum $T_R[s]$ for $TP_{[A,B,C,D]}$, hardware/implementation dependent characteristics T_B and T_R.

time T_R and the build time T_B and the sum of both. These characteristics are summed over all testing procedures and scenes. (120 experiments if all the tests were completed successfully.)

3.5.3 Discussion

Below, we comment on the characteristics of each *RSA* tested. We sorted all *RSAs* according to their time $\tilde{T}_R$ summed over all tests, we start our discussion with the slowest *RSA* and finish with the fastest one. We use $\tilde{T}_R$ just for ranking according to Section 2.7. Comparing the results presented in Tables 3.3 and 3.4, and Figures 3.1, 3.2, and 3.3 together with all the extra data [79], we can comment on the tested *RSAs*:

BVH: Has rather poor results for all testing procedures compared to other *RSAs*. We see the main problem in the nature of the construction of BVH. It does not keep track of spatial coherency – when inserting a new object into the existing hierarchy there is no global spatial information

RSA	Testing Procedure												Total		
	TP_A			TP_B			TP_C			TP_D			$\Sigma_{TP=A,B,C,D}$		
	$\tilde{T}_R$	m/f	W	$\tilde{T}_R$	m/f	W	$\tilde{T}_R$	m/f	W	$\tilde{T}_R$	m/f	W	$\tilde{T}_R$	m/f	W
KD	9.98	0/0	22	26.29	0/0	21	76.27	0/0	23	29.42	0/0	22	142.0	0/0	88
O93A	15.01	0/0	0	40.91	0/0	0	106.8	0/0	0	38.59	0/0	0	201.3	0/0	0
O84A	15.41	0/0	0	41.72	0/0	0	109.0	0/0	0	39.08	0/0	0	205.2	0/0	0
RG	35.09	3/0	2	64.44	3/3	0	134.4	0/2	0	47.84	0/0	0	281.8	6/5	2
HUG	39.63	0/0	0	99.52	0/0	1	268.2	0/0	0	77.38	0/0	0	484.7	0/0	1
AG	63.31	3/0	1	108.6	3/0	1	278.9	3/1	1	136.5	0/0	2	587.3	9/1	5
UG	11.74	0/0	5	372.7	0/0	7	525.5	0/0	6	145.4	0/0	6	1055	0/0	24
O93	16.86	0/0	0	1114	0/0	0	257.5	0/0	0	392.8	0/0	0	1781	0/0	0
BSP	28.63	0/0	0	1291	0/0	0	325.9	0/0	0	560.3	0/1	0	2206	0/1	0
O89	14.49	0/0	0	1127	0/0	0	1421	0/0	0	381.7	0/0	0	2944	0/0	0
O84	17.44	0/0	0	1132	0/0	0	1437	0/0	0	400.2	0/0	0	2987	0/0	0
BVH	1903	0/0	0	4569	0/2	0	5111	0/6	0	3376	0/0	0	14960	0/8	0
$\Sigma_{all RSA}$	2170	6/0	30	9986	6/5	30	10050	3/9	30	5625	0/1	30	27832	15/15	120

Table 3.4: Average running time $\tilde{T}_R$, memory (m) and time limit (f) failures, and number of "wins" W for all tested acceleration algorithms and testing procedures.

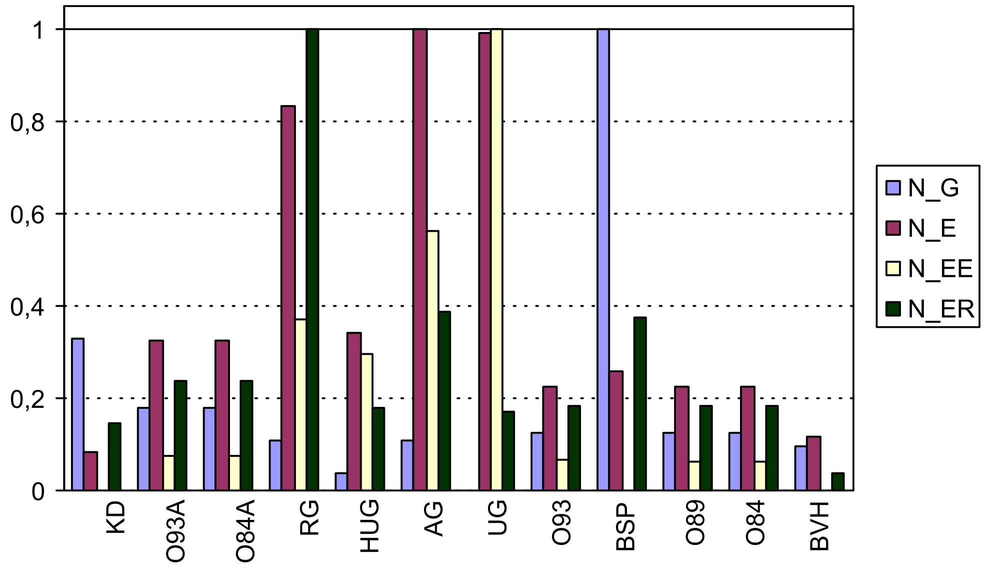


Figure 3.1: Parameters N_G, N_E, N_{EE}, and N_{ER} summed for all scenes and each *RSA*, normalized to the worst *RSA*. A description of the parameters is in Section 2.2.

about other still uninserted objects.

O84: The octree with a sequential ray traversal algorithm requires many traversal steps from the root node. Subdividing at midpoints does not work particularly well for sparse scenes ("treeX").

O89: Has a slightly better traversal algorithm that outperforms O84, especially for scenes with higher numbers of objects.

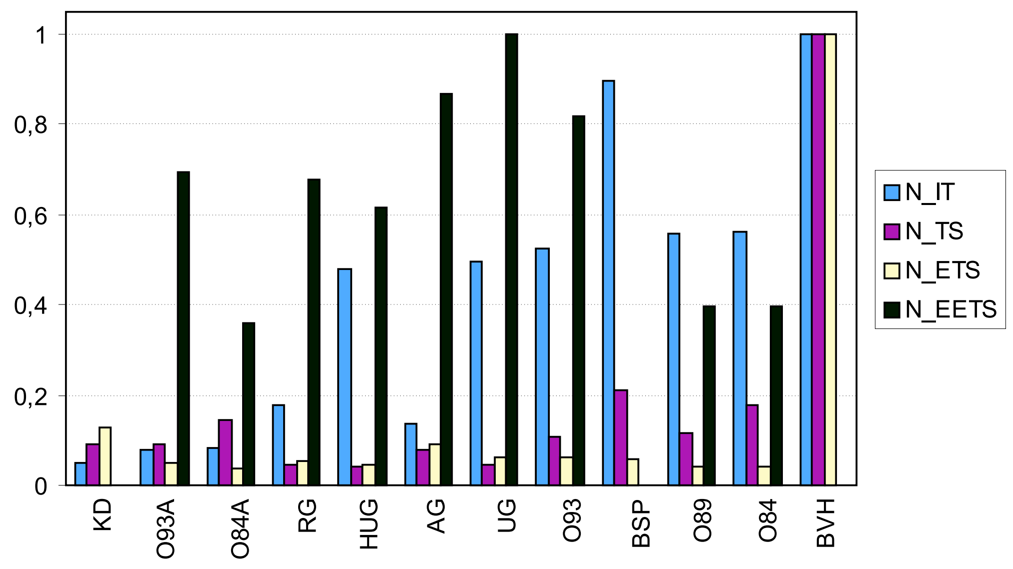


Figure 3.2: Parameters N_{IT}, N_{TS}, N_{ETS}, and N_{EETS} summed for all scenes for each *RSA* and normalized to the worst *RSA*. A description of the parameters is in Section 2.2.

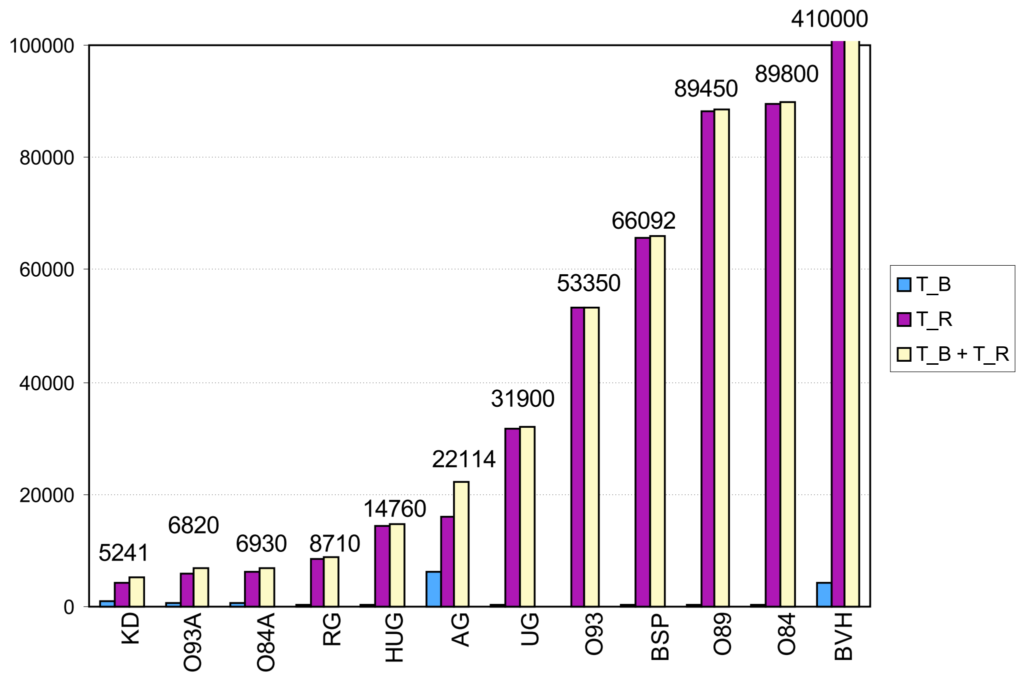


Figure 3.3: Total times $T_B[s]$, $T_R[s]$, and $T_B + T_R[s]$ for each *RSA*. The number gives the total build+running time ($T_B + T_R$) for all tests for a particular *RSA*.

BSP: Although conceptually the same structure as the *kd*-tree, subdividing at midpoints again results in poor performance for sparse scenes. For densely occupied scenes the BSP tree performs comparably to the *kd*-tree.

O93: Due to the most efficient ray traversal algorithm, this outperforms O84 and O89 even if constructed using the midpoint subdivision. We can see that for all midpoint subdivision octrees, shooting rays inside the octree (TP_B, TP_C, and TP_D) is very time demanding. This is the price we pay for traversal down to the leaf when the intersected object is inside or very close to this node.

UG: Classical *RSA*. The employed smart algorithm for heterogeneous grid resolution setting results in the best performance of UG for several scenes. These scenes are densely occupied with mostly regular structure ("jacksX", "latticeX", and "mountX"). In this kind of scene the down traversal phase for hierarchical spatial data structures is expensive, since the ray intersects the object very close to the origin of a ray. For sparsely occupied scenes the UG has rather poor performance as it lacks a sense of hierarchy.

AG: As a combination of the BVH with UG this has average performance, but the prediction of memory needed to construct the underlying data structure is difficult for G^5_{SPD} scenes. For one scene the computation failed the time limit due to swapping. Tweaking of the construction parameters was necessary to get some G^5_{SPD} scenes to work on available memory.

HUG: Consists of UGs arbitrarily positioned in other UGs. It has not only a slightly better performance than AG, but also smaller and more predictable memory usage.

RG: UG inserted in voxels of UG recursively shows negligible performance improvement especially for TP_C. Its performance varies, but five tests were failed on the time limit. Tuning the construction parameters to keep the test within the memory limit was difficult as memory consumption was rather unpredictable.

O84A: We can see that although a simple ray traversal algorithm was used, a more appropriate subdivision process improved the total performance by one order of magnitude compared with O84. The improvement is particularly apparent on sparse scenes in G^5_{SPD}. Unfortunately, the build time T_B rises rapidly for G^5_{SPD} as well.

O93A: The same improvement as between O84 and O93 can be observed due to the more efficient ray traversal algorithm. Again, the build time T_B rises rapidly with the number of objects in the scene.

KD: Although *kd*-tree is in principle the BSP tree, the positioning of the splitting plane using the surface area heuristic and fast ray traversal algorithm makes this hierarchical spatial data structure into a winner, even if the improvements from O93A are not significant. The *kd*-tree is beaten in several cases: for regular artificial scenes ("jacksX", "latticeX", and "mountX") by UG, for G^5_{SPD} sparse scene ("treeX") by AG and HUG. However, the differences in performance in all these cases are small. The only disadvantage is that the build time T_B for G^5_{SPD} scenes, which is comparable to the build times of O93A and O84A, can be rather high in comparison with the build time of UG. Therefore, if the number of ray shooting queries is low, using the *kd*-tree probably does not pay off.

In general, we observe that using surface area heuristic [105] pays off for both the octree and BSP tree to get the Octree-R and *kd*-tree. Also, *RSAs* based on hierarchical data structures win over the *RSAs* based on non-hierarchical ones in most cases, especially for sparse scenes. Whether these results and the ranking of algorithms according to $\sum_{TP=A,B,C,D} T_R$ will be maintained for the planned collection of 100 downloaded scenes is not currently known.

3.5.4 Preliminary *RSA* Selection Algorithm

Examining the results of Tables 3.2 and 3.3, we would like to present a preliminary proposal of an algorithm for selecting an *RSA* to be used given a scene: First, construct a uniform grid over the $\mathcal{AB}s$ of objects using the heterogeneous resolution setting with voxel density $d_{voxel} = 1.0$. Then compute the sparseness parameters δ, s, and k. When δ, s, and k are low (see Table 3.2), then use an *RSA* based on

UG. If these three parameters are in the middle range, use an *RSA* based on *kd*-tree. If these parameters are very high, then consider either an *RSA* based on *kd*-tree or RG/AG for even better performance, but be aware of the possible high or even unrealizable memory requirements for RG/AG. If none of the conditions above are applicable, then use the *RSA* based on *kd*-tree. Here, we should stress that this preliminary *RSA* selection algorithm is derived from the results of experiments for 30 *SPD* scenes with a fractal nature. Its validity has to be verified on larger a set of scenes. Knowledge of whether most rays will be shot inside the scene will also be helpful for selecting the *RSA*.

3.6 Conclusion and Future Work

In this chapter we outlined the goals of the BES project and its status in late November 2000. We have proposed four testing procedures for testing *RSAs*, the *RSA* invariants for particular testing procedures, and an algorithm for systematically shooting uniformly distributed rays. Based on the measured data we have also outlined a heuristic to select a suitable *RSA* given a statistics based on scene sparseness. Even if the heuristic predicts reasonably well for the tested *SPD* scenes, we do not know if the algorithm in its present form will be applicable to general scenes, and its success ratio. It is only clear that for a small number of rays to be shot no construction of an underlying data structure for a particular *RSA* pays off at all.

After testing 12 *RSAs* over 30 *SPD* scenes of different complexities and 4 testing procedures, we can conclude that using *RSAs* based on hierarchical spatial data structures, particularly on the *kd*-tree, definitely pays off – except for densely occupied scenes. This observation supports our opinion that it is very unlikely that there will be a single optimal *RSA* for general use.

The BES project has not yet come to an end. In order to provide a sound basis that will help us to avoid further speculations about the design and use of different *RSAs*, several tasks have to be completed: First, we have to make our collection of 100 practical scenes complete and run all the tests again over this set. This will provide us with a vast amount of statistical data that will have to be analyzed together with the results of experiments already presented here. When the results are ready, we will be able to conclude whether the results for the *SPD* scenes presented in this chapter correlate with the results obtained for the practical scenes. This will also reveal how well the distribution of objects in the scenes from *SPD* actually simulates the distribution that occurs in the practical scenes.

Chapter 4

Construction of *Kd*-Trees

In this chapter we describe several new methods for constructing the *kd*-tree for *RSAs* that are based on this spatial data structure. The construction algorithms use the cost model that estimates the average cost of traversing an arbitrary ray through the *kd*-tree, combining the cost of the traversal step and the cost of a ray-object intersection test. The estimated cost is then used to govern the position and orientation of the splitting plane during *kd*-tree construction, which proceeds in top-down fashion. The structure of this chapter is as follows. We describe our motivation for selecting an *RSA* based on the *kd*-tree for detailed research, previous work, and several contributions of ours in separate sections.

4.1 Motivation

Starting with this chapter, the rest of the thesis is devoted primarily to the *kd*-tree. The first phase of the BES project presented in the previous chapter showed us that the *kd*-tree is statistically the best from common heuristic *RSAs*, at least for tested 30 *SPD* scenes. There are also other reasons for selecting the *kd*-tree as the winning candidate of the first phase of the BES project for detailed research:

- The *kd*-tree does not suffer from exhaustive memory complexity requirements; We can observe from the first phase of the BES project that the number of elementary and generic cells is roughly linear with the number of objects. Moreover, the limitation on maximum memory usage given by hardware available for the *kd*-tree can be encoded in the termination criteria for its construction.
- The *kd*-tree has also been studied within the field of computational geometry, with promising results for $\mathbb{E}^2$ and higher dimensions. The research in computational geometry is connected with term partition strongly related to the *kd*-tree. A *partition* is a binary space partitioning tree with a linear splitting entity (line in $\mathbb{E}^2$, plane in $\mathbb{E}^3$) in general position such that each leaf of the constructed partition contains at most one object. d'Amore and Franciosa [36] show that is possible to construct a partition for a set of disjoint isothetic rectangles in $\mathbb{E}^2$. Mark de Berg *et al.* [39] show that for $\mathbb{E}^2$ it is possible to construct a partition of linear size for a set of objects under certain conditions for input data. Agarwal *et al.* [4] discuss practical techniques for constructing of the *kd*-tree for orthogonal rectangles in $\mathbb{E}^3$. These approaches with $O(N.\log N)$ preprocessing and with linear storage achieve $O(\log N)$ time complexity for point-location queries. However, for general shapes of objects, no algorithm for constructing a partition with linear storage and $O(N.\log N)$ preprocessing has been found, even for $\mathbb{E}^2$. These results promote the use of the *kd*-tree for *RSAs*, disregarding the worst-case complexity measures.
- The *kd*-tree allows flexible positioning of the splitting planes, which results in various sizes of the elementary cells. The cells adapt well to the geometry of the scenes. According to the results of the first phase of the BES project, this feature is particularly important for sparsely occupied scenes, and we can quantify the significant difference in performance between an *RSA* based on

the BSP tree and an *RSA* based on the *kd*-tree. Although the BSP tree is in principle the same spatial data structure as the *kd*-tree, fixing the position of the splitting planes in the center of the cell caused the performance of an *RSA* based on the BSP tree sometimes to be worse by order(s) of magnitude than that of an *RSA* based on the *kd*-tree. We observed this for experiments performed within the first phase of the BES project, when the performance of an *RSA* based on the *kd*-tree was always better than or equal to an *RSA* based on the BSP tree.

- The *kd*-tree is principally scalable to $\mathbb{E}^n$ space for arbitrary n. This holds for the *kd*-tree construction and ray traversal algorithms described in this thesis.
- The *kd*-tree can be used to model topologically many other spatial subdivisions. This means that the *kd*-tree can be constructed in such a way that it corresponds to the spatial topology of elementary cells for the uniform grid, non-uniform grid, both elementary and generic nodes of an octree, Octree-R, and BSP tree. On the other hand, the spatial topology of the *kd*-tree can be mapped to a more general spatial data structure: the bounding volume hierarchy (BVH). Unfortunately, the ray traversal algorithm for the BVH is much less efficient than for the *kd*-tree, due to the general nature of BVH, since the cells of the child nodes of BVH referenced in one node can overlap. Further, this overlapping of cells within a hierarchical data structures is not suitable for efficiency reasons, see the point below and the results presented in the previous chapter.
- The *kd*-tree contains no overlapped elementary cells, and thus no two descendants referenced in one interior node of the *kd*-tree. This overlapping of cells occurs in the hierarchy of uniform grids (HUG) by Cazals [26], adaptive grids (AG) [100], and BVH [63]. The overlapping of cells, which is the overlapping of spatial regions, always induces an elementary/generic cell $\mathcal{V}$ is to be referenced in more than one other generic cell. First, such multiple referencing leads to repetitive testing of elementary cell $\mathcal{V}$ for the intersection with a ray. It is usually necessary to solve the ray query for the whole cell $\mathcal{V}$ as if it were the scene itself. Repetitive computation for $\mathcal{V}$ can be reduced by a ray-cache (also called a mailbox [23, 12]), where the result of intersection between the ray and $\mathcal{V}$ is cached. Nonetheless, the time needed to access the ray-cache cannot be completely eliminated. Second, since an elementary cell $\mathcal{V}$ can also be overlapped with another elementary cell $\mathcal{V}'$ (or several such cells), then both of these must be checked for intersection within the overlapping spatial region, since the object with the closest ray-object intersection must be chosen. Therefore it may occur that a part of the spatial region covered by $\mathcal{V}$ checked for intersection with the ray is not used at all, that is, the computation within part of cell $\mathcal{V}$ was useless. Whether the computation is useless for a particular cell $\mathcal{V}$ cannot be determined until the second cell $\mathcal{V}'$ is tested. It would be theoretically possible to traverse more than one cell in parallel within a ray traversal algorithm, but this would be rather complicated. We are not aware that any such ray traversal algorithm has been published. The results of the first phase of the BES project support our view on cell overlapping: it should be avoided, in order not to decrease the quality of encoding the distance among the objects in spatial data structures.
- The *kd*-tree is a hierarchical spatial data structure, *i.e.*, it enables us to deal with objects of various sizes using the level of detail concept. It is particularly required for scenes with unevenly distributed objects. The use of a hierarchy in data structures for *RSAs* was considered by some researchers in the past as a disadvantage [53] in comparison with *RSAs* based on non-hierarchical data structures such as the uniform grid. Unfortunately, these *RSAs* based on non-hierarchical spatial data structures are very inefficient for scenes with unevenly distributed objects. As we have seen in the previous chapter, they can slightly outperform *RSAs* based on hierarchical data structures for densely occupied scenes with a regular structure, but in general, they suffer from performance problems.
- There are several efficient ray traversal algorithms for the *kd*-tree. The efficiency of the ray traversal algorithm for the *kd*-tree is given by the simple representation of information in the

kd-tree node – since the interior node contains the splitting plane and thus two child nodes, we have to decide between four cases: traverse only left child, only right child, the left child first and then the right one, or the right child first and then the left one. Efficient ray traversal algorithms for the *kd*-tree have been developed in the context of the BSP tree, and are further dealt with in Chapter 5.

The rest of this chapter is structured as follows. First, we discuss why an *RSA* based on the *kd*-tree is more efficient than an *RSA* based on the BSP tree with arbitrary oriented splitting planes. The problem of top-down construction of *kd*-trees is in fact simply formulated in two issues. First, we have to decide if to declare the current node containing the references to the objects as a leaf. When the answer to this question is negative, we have to put the splitting plane, and the second issue is: where to position the splitting plane. We discuss methods for positioning the splitting plane, and we describe in detail the concept of a cost model, and its use. Further, we show how the empty spatial regions in the scene can be utilized in *kd*-tree construction to increase the performance of an *RSA* based on the *kd*-tree. Then, we deal with the termination criteria – whether or not the current node of the *kd*-tree should be already declared as a leaf. Further, we generalize the cost model and show several possible *kd*-tree efficiency improvements, and also their limitations. We conclude the chapter by a summary of results from experiments that we have performed.

4.2 Previous Work

In this section we describe the work concerning *kd*-tree construction performed in the past. First, we recall several basic facts. The recursive construction of *kd*-trees in top-down fashion is similar to that for the BSP tree described in Subsubsection 1.6.3.1, namely Algorithm 1. We are not aware of any method published that uses a bottom-up approach, even if this might be possible using some clustering method and redistributing the objects straddling the splitting plane back down to the children. Obviously, the top-down construction is more straightforward. The recursiveness of the top-down approach is encoded so that in each step of the construction the set of objects is taken as the whole scene and the splitting plane position and orientation is determined.

4.2.1 Orientation of the Splitting Plane in the *Kd*-Tree

Here, we show why we use for *RSA* the *kd*-tree, which has axis-aligned splitting planes, instead of the BSP tree with arbitrarily oriented splitting planes. The main reason is imposed by the ray traversal algorithms for these two spatial data structures, which require that we compute the signed distance from the origin of a ray to a splitting plane. For example, a plane Π_{a} perpendicular to the a-axis ($a \in \mathrm{x,y,z}$) is described by the formula: $\Pi_{\mathrm{a}} : a = a_{\mathrm{p}} = const$. A ray R is described by its origin O^{R} and the direction vector $\vec{D}^{\mathrm{R}}$. The computation of the signed distance t for the intersection of an arbitrary ray with the plane is:

$$t = \frac{a_p - O^{\mathrm{R}}_a}{D^{\mathrm{R}}_a}, \tag{4.1}$$

assuming $D^{\mathrm{R}}_a \neq 0$. Note that inverse of D^{R}_a can be precomputed that is practically important for speed of computation. For an arbitrarily oriented plane given by the equation $\Pi : a.x + b.y + c.z + d = 0$ the computation of the signed distance with the ray is more computationally demanding:

$$t = \frac{a.O^{\mathrm{R}}_{\mathrm{x}} + b.O^{\mathrm{R}}_{\mathrm{y}} + c.O^{\mathrm{R}}_{\mathrm{z}} + d}{a.D^{\mathrm{R}}_{\mathrm{x}} + b.D^{\mathrm{R}}_{\mathrm{y}} + c.D^{\mathrm{R}}_{\mathrm{z}}} \tag{4.2}$$

For the arbitrarily oriented plane, the number of elementary arithmetic operations is about 3 times higher than for the axis-aligned plane. Note that unlike Eq. 4.1 the denominator in Eq. 4.2 cannot be precomputed. The computation of the signed distance forms only a portion of the cost of a traversal step in the ray traversal algorithm, but it is significant enough from the viewpoint of the total running time of the ray traversal algorithm. To justify the use of a BSP tree with arbitrarily oriented splitting planes for *RSA* we could require that we decrease the total running time significantly (about 2-3 times) by reducing the number of traversal steps or ray-object intersection tests. To the best of our knowledge no such method has been published until now. Whether or not the objects belong to one halfspace induced by the generally oriented splitting plane is also much more computationally demanding. Moreover, the number of possible positions of the splitting plane would increase from $O(N)$ to $O(N^3)$. As a result, the build time of the *kd*-tree increases considerably, which is rather unacceptable. For these reasons we do not deal with generally oriented splitting planes in the rest of this thesis, and thus we stay with the *kd*-tree.

There are several ways to select the orientation of the splitting plane provided that it is perpendicular to one of the coordinate axes. In the axis-aligned form of the BSP tree (Section 1.6.3.1) the orientation of the splitting plane is changed in cyclic order $(x, y, z, x, \dots)$ with the increasing depth of the node in the *kd*-tree. The starting axis for the splitting is not specified, but it is usually the x-axis [94, 148] regardless of the shape of scene $\mathcal{AB}$. There are also several other ways to orientate the splitting plane, but since the used method is intertwined with the positioning of the splitting plane, we describe it below.

4.2.2 Positioning of the Splitting Plane

During construction of a *kd*-tree in top-down fashion we have to solve algorithmically the problem of splitting the $\mathcal{AB}$ of the current node into two new child nodes. This also covers the assigning of objects into the child nodes. One of our assumptions is that each object O_i has a finite size and thus it also has finite $\mathcal{AB}(O_i)$. Since the splitting planes in the *kd*-tree are axis-aligned and we have to decide whether the objects lie on the left side or the right side of the splitting plane, it is also advantageous to use the $\mathcal{AB}s$ of the objects for this decision. There are several known methods for positioning and orientating the splitting plane in the *kd*-tree:

Spatial Median: In BSP tree construction the splitting plane always splits the $\mathcal{AB}$ associated with the current node into two halves. Since the result of the splitting is two $\mathcal{AB}s$ of the same size, then these $\mathcal{AB}s$ have to be smaller with the increasing depth of the node in the *kd*-tree. This approach balances the *space* on the sides of the splitting plane.

Object Median: Another way is to position a splitting plane so that the number of objects lying on its left side and right side is equal. Some objects can be assigned to both children, since their $\mathcal{AB}s$ straddle the splitting plane. This method balances the number of *objects* on the sides of the splitting plane. This may seem natural, since it resembles the way of constructing a balanced binary search tree over a set of numbers in $\mathbb{E}^1$. One can then suppose that the construction of such a balanced binary search tree could be advantageous for an *RSA*. Unfortunately, this is a rather incorrect intuition, and the method has severe performance deficiencies compared with other methods, as we will show by the results of our experiments. We remark that the *kd*-tree constructed for *RSA* is not the binary search tree used for a common range search within a one-dimensional interval. The main difference is that in search structures in $\mathbb{E}^1$ such as range trees [40], the search is finished in $O(\log N)$ time. Within a ray traversal algorithm for the *kd*-tree after we descend to the first leaf of the *kd*-tree, this does not mean that the search is finished; when a ray does not intersect any object referenced in the first leaf, the ray traversal algorithm continues finding the leaves along the ray path.

Cost Model: This method is based on a further refinement of the computation model introduced in Section 2.2 for the *kd*-tree. This approach is based on a *cost model*, which estimates the average cost of traversing an arbitrary ray through the *kd*-tree during the construction. The method outperforms both spatial median and object median methods for all tested scenes that we have tested until now, as implemented within the GOLEM rendering system. Historically speaking, the method was first introduced by MacDonald and Booth during the *Graphics Interface* conference [104] in June 1989, and was further revised and published in [105]. This cost model was also researched by Subramanian and Fussel [144, 145], who also published the an experimental comparison of the theoretical cost and measured cost of the hierarchy based on its termination criteria [146]. We deal with this construction method in deep detail below.

The construction of the *kd*-tree for the case of the spatial and object median needs no further discussion. This is not however the case for the cost model, which we recall and discuss below in detail, following MacDonald and Booth [105].

4.2.3 Cost Model for *Kd*-Tree Construction

The cost model is a theoretical model that estimates the cost of a ray passing through a *kd*-tree under several assumptions. Such an estimate includes both the cost for visiting the interior nodes and leaves of the *kd*-tree and the cost of computing ray-object intersection tests. In the following text, let $\hat{X}$ denote an *estimate of quantity* X, that is, we cannot determine a value of quantity X exactly in advance, but we can only somehow estimate its value. The development of the cost model is enabled by several simplifying assumptions. One of these assumptions uses geometric probability.

4.2.3.1 Geometric Probability

The development of the cost model is connected with the following observation, known from geometric probability theory. For more mathematical details, see [138] or the survey by Cazals and Sbert [27], which is more oriented to applications in global illumination algorithms. Geometric probability tools for the construction of underlying data structures for *RSAs* were first used for BVH by Goldsmith and Salmon [63]. Their approach is also outlined in the survey by Arvo and Kirk [18].

Let X and Y be spatial regions of convex shape such that X contains Y, *i.e.*, $X \cap Y = Y$. We want to express the conditional probability $p_{Y|X}$ that an *arbitrary ray* intersects the spatial region Y assuming it intersects the spatial region X. The arbitrary ray is a ray that has uniform distribution of the origin and direction of the ray, so the line density induced by rays in the spatial region X and hence Y is constant. Further, it is supposed that the arbitrary ray has a direction outside the spatial region X. The situation is depicted in Fig. 4.1.

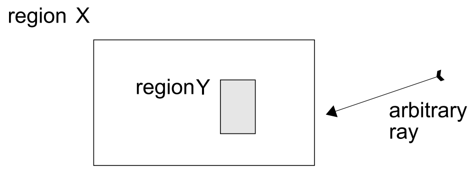


Figure 4.1: Computing the conditional probability that an arbitrary ray hits spatial region Y once it passes through spatial region X.

Stone in [140] (cited according to [63]) showed that this probability $p_{Y|X}$ is proportional to the surface area of the convex spatial region Y divided by the surface area of the convex spatial region X:

$$p_{Y|X} = \frac{SA(Y)}{SA(X)} \tag{4.3}$$

If we restrict our observation to X and Y to be $\mathcal{AB}$s, then we can express $p_{Y|X}$ as follows:

$$p_{Y|X} = \frac{(Y_w.Y_h + Y_w.Y_d + Y_h.Y_d)}{(X_w.X_h + X_w.X_d + X_h.X_d)}, \tag{4.4}$$

where subscripts w, d, and h denote the width, depth, and height of $\mathcal{AB}$.

4.2.3.2 Basic Cost Model Development

The development of the cost model here follows the paper by MacDonald and Booth [105]. The cost model is based on several rather unrealistic assumptions for ray shooting in order to apply the formula 4.4 in the estimated quantities:

- all rays intersect the $\mathcal{AB}$ associated with the root node of the *kd*-tree,
- the distribution of rays is uniform,
- all rays do not intersect any object.

The last assumption is very unrealistic, since it contradicts the purpose of any *RSA*. Nevertheless, under these assumptions we can estimate the following quantities of a *kd*-tree as follows:

Number of interior nodes of the *kd*-tree traversed per ray:

$$\hat{N}_{TI} = \sum_{i=1}^{N_i} \frac{SA(i)}{SA(root)}, \tag{4.5}$$

number of leaves of the *kd*-tree traversed per ray:

$$\hat{N}_{TL} = \sum_{l=1}^{N_l} \frac{SA(l)}{SA(root)}, \tag{4.6}$$

number of ray-object intersection tests per ray:

$$\hat{N}_{IT} = \sum_{l=1}^{N_l} \frac{SA(l).N(l)}{SA(root)}, \tag{4.7}$$

where the other quantities are:

N_i	–	number of interior nodes of the *kd*-tree,
N_l	–	number of leaves of the *kd*-tree, ($N_l = N_i + 1$),
$N(l)$	–	number of objects stored in leaf l of the *kd*-tree,
$SA(i)$	–	surface area of interior node i,
$SA(l)$	–	surface area of leaf node l,
$SA(root)$	–	surface area of the $\mathcal{AB}$ of the whole scene.

The estimate of the number of operations above performed during the ray traversal algorithm can be used to estimate the average total cost of ray shooting under the assumptions above, if we know the costs of specific operations of the ray traversal algorithm. For further development we assume a recursive ray traversal algorithm for the *kd*-tree as outlined in Subsubsection 1.6.3.1. The cost of these operations is connected with a given implementation and can be obtained experimentally. Then from a general definition of the performance model (Eq. 2.2) we can estimate the total cost $\hat{C}_T$ for shooting an arbitrary ray:

$$\hat{C}_T[s] = \hat{C}_{TI}.\hat{N}_{TI} + \hat{C}_{TL}.\hat{N}_{TL} + \hat{C}_{IT}.\hat{N}_{IT} \tag{4.8}$$

$$= \frac{1}{SA(root)}.[\hat{C}_{TI}.\sum_{i=1}^{N_i} SA(i) + \hat{C}_{TL}.\sum_{l=1}^{N_l} SA(l) + \hat{C}_{IT}.\sum_{l=1}^{N_l} SA(l).N(l)], \tag{4.9}$$

where	$\hat{C}_{TI}[s]$	–	the estimated cost of traversing an interior node of the *kd*-tree,
	$\hat{C}_{TL}[s]$	–	the estimated cost of traversing a leaf node of the *kd*-tree,
	$\hat{C}_{IT}[s]$	–	the estimated cost ray-object intersection test.

The estimate of the total cost is rather the upper bound, since it assumes that all arbitrary rays do not intersect any object, but at the same time it assumes that ray-object intersection tests are computed. MacDonald and Booth also discuss the variant with a ray-cache that avoids ray-object intersection computation for a ray more than once for the same object. The ray traversal algorithm tests a ray only once against a particular object, and the result of the ray-object intersection test is stored in the ray-cache. For the next ray-object intersection test with the same object the result is retrieved from the cache. Several research papers report that use of the ray-cache does not always increase the performance of *RSAs* in a ray tracing algorithm [147]. The use of the ray-cache also increases the ray-object intersection test cost $\hat{C}_{IT}$ of all objects. Even if the ray-cache helps for several scenes, the impact on performance can be negligible and takes a few percent at maximum of the computation time consumed by an *RSA*. Since our observation on the use of the ray-cache in the *RSAs* in our previous experiment [80] confirmed that the impact on performance is rather questionable, we will not further discuss this extension here.

MacDonald and Booth [105] claim the validity of the presented estimates for $\hat{N}_{TI}$, $\hat{N}_{TL}$, and $\hat{N}_{IT}$ by a simulation performed on arbitrary scenes with arbitrarily built *kd*-trees for arbitrary rays. Assuming that the estimate of the total cost is accurate enough, we can use it to govern the construction of the *kd*-tree. This means that we choose the positions and orientations of the splitting planes in the *kd*-tree so as to minimize its total estimated cost (Eq. 4.9). MacDonald and Booth call any algorithm that tries to minimize the estimated cost of a *kd*-tree for an *RSA* a *surface area heuristic.* We should remark here that surface area heuristic does not necessarily find the global minimum of the estimated cost, but it rather tries to decrease the estimated cost.

4.2.3.3 Position of the Splitting Plane

In top-down construction of the *kd*-tree we always consider some set of objects pointed to in an $\mathcal{AB}$. The *kd*-tree construction algorithm is then reformulated to find the position and orientation of the splitting plane for the $\mathcal{AB}$ associated with the currently processed interior node. MacDonald and Booth [105], in order to minimize Eq.4.9, proceed as follows, we quote:

> We assume that only major planes[1] are used as splitting planes and we ignore the possibility of an object straddling a splitting plane (a case of practical importance, but one we ignore nevertheless). We have to choose a parameter b to position the splitting plane, where $b = 0$ corresponds to the lower limit of the splitting plane and $b = 1$ is the upper limit. Choosing $b = 0.5$ is equivalent to selecting the spatial median.
>
> Let us look at the cost as a function of this parameter b. We observe that the internal node and leaf node components of this cost savings function are constant with respect to b. For the purposes of minimizing cost[2], we can minimize the function:
>
> $$f(b) = LSA(b).N_L(b) + RSA(b).(N - N_L(b)) - SA.N, \tag{4.10}$$
>
> where N is the number of objects in the node, $N_L(b)$ is the number of objects to the left of the plane at b, and $(N - N_L(b))$ is the number to the right of the plane because of our assumption that no objects straddle the plane. The surface area of the left and right subnodes are $LSA(b)$ and $RSA(b)$, respectively, and the surface area of the node itself is SA. The first term

[1] By major plane they mean the axis-aligned plane.
[2] This cost corresponds to the estimated cost in Eq. 4.9.

represents the probability that a ray intersects the left subnode multiplied by the number of intersection tests performed in the left subnode. The second term is a similar quantity for the right subnode. The *SA.N* term is the amount of work required if the node were not subdivided and thus is an amount of work saved by changing the original node from a leaf to an internal node, hence the minus sign. This last quantity is a constant with respect to *b*, so it may be removed from the function, resulting in the following function to be minimized:

$$f'(b) = LSA(b).N_L(b) + RSA(b).(N - N_L(b)), \tag{4.11}$$

End of quotation.

We consider that the description of the surface area heuristic is rather simplified and unclear, so we will elaborate it in more detail below. From now on we present another view of the positioning of the splitting plane. Let us assume the situation before splitting the $\mathcal{AB}$ associated with the interior node by a splitting plane, *e.g.*, at the root node of the *kd*-tree. The $\mathcal{AB}$ associated with the node intersects N objects. If the node is not subdivided, it is actually a leaf of the *kd*-tree – then all its objects have to be pointed to in this leaf and they have to be tested for intersection with a ray. Let $C_{IT}(i)$ be the cost of the ray-object intersection test for the i-th object. The cost of such an unsubdivided node ν^E for a ray shooting query is then:

$$\hat{C}_{\nu^E} = \sum_{i=1}^{N} C_{IT}(i) \tag{4.12}$$

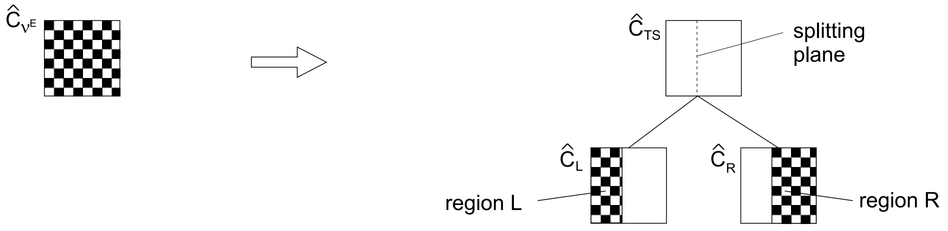


Figure 4.2: One subdivision step in a *kd*-tree

If the $\mathcal{AB}$ is subdivided as depicted in Fig. 4.2, then it is replaced by a new tree structure – the interior node ν^G with two leaves ν^E_L and ν^E_R. The estimated cost of the new tree structure $\hat{C}_{\nu^G}$ is given as the sum of three terms – $\hat{C}_{TS}$, $\hat{C}_L$, and $\hat{C}_R$. The term $\hat{C}_{TS}$ is the estimated cost of traversing the interior node of the *kd*-tree, either the leaf or the interior node. It does not incorporate any ray-object intersection tests, but the decision whether to visit either the left child or the right child or both children. Moreover, $\hat{C}_{TS}$ involves the pointing to the child nodes within a ray traversal algorithm. The costs of visiting the left and right children should contain a factor with the conditional probability that a ray hits the $\mathcal{AB}$s of the leaves ν^E_L or ν^E_R once it visits the parent node ν^G. The estimated cost $\hat{C}_{\nu^G}$ of the interior node ν^G is then expressed as follows:

$$\hat{C}_{\nu^G} = \hat{C}_{TS} + p_L.\hat{C}_L + p_R.\hat{C}_R \tag{4.13}$$

where	$\hat{C}_{TS}$	–	estimated traversal cost of interior node ν^G,
	p_L, p_R	–	probability of a ray intersecting the left or right child node, respectively,
	$\hat{C}_L, \hat{C}_R$	–	estimated cost of the left and right subtree, respectively.

Under the assumption of uniformly distributed rays we can compute the probability of a ray hitting the $\mathcal{AB}$ of the left and right child node using Eq. 4.1, *i.e.*, $p_L = SA(\mathcal{AB}(lchild(\nu^G)))/SA(\mathcal{AB}(\nu^G))$, $p_R(b) = SA(\mathcal{AB}(rchild(\nu^G)))/SA(\mathcal{AB}(\nu^G))$. Further, we can estimate the cost of the left and right subtree, supposing they will be constructed. The simplest input for the estimate is to consider the number of objects contained in the spatial regions corresponding to the subtrees to be constructed $\hat{C}_L = f_C(N_L)$ and $\hat{C}_R = f_C(N_R)$, where N_L and N_R are the number of objects in the left and the right child, respectively. If we further choose the estimate to be linear function, *e.g.*, $f_C(k) = k$, and suppose the objects do not straddle the splitting plane, we can rewrite the estimated cost in Eq. 4.13 to be Eq. 4.11. For the sake of convenience, we give the name *cost function* to any formula that estimates the cost of a *kd*-tree, *e.g.*, Eq. 4.13.

4.2.3.4 Position of a Splitting Plane with Minimum Cost

As described above, surface area heuristic corresponds to the computation of cost function for all possible positions of the splitting plane for all its three possible orientations, and to the selection of the position that has the lowest estimated cost. MacDonald and Booth [105] claim that the position of the splitting plane with the lowest estimated cost has to lie between the spatial and object median, assuming that no objects are intersected by a splitting plane at the same time. In this case they prove that the cost value $f'(b_{OM}) = f'(b_{SM})$, where b_{OM} and b_{SM} are the positions of the object and spatial median. For further discussion, let the *median interval* be an interval between the object and spatial median, and let the *minimum cost splitting plane* be a splitting plane that minimizes the value of some cost function.

The proof of $f'(b_{OM}) = f'(b_{SM})$ is simple. The value of $f'(b_{OM}) = f'(0.5) = N.LSA(0.5)$ because $LSA(0.5) = RSA(0.5)$. Since the value of $LSA(b) + RSA(b) = 2.LSA(0.5)$ is a constant independent of b, the value at the spatial median is $f'(b_{OM}) = [LSA(b) + RSA(b)].N/2 = N.LSA(0.5)$. It can also be proved that the value of the cost function in the median interval is lower than the value of cost function outside the median interval, so the minimum is inside the median interval.

The assumption that objects do not overlap in the projection to the coordinate axis is unrealistic for a general scene. If we want to minimize the cost function (Eq. 4.11) correctly for objects that possibly overlap in projection, the full range of the $\mathcal{AB}$ along the axis should be searched. Although the range is continuous, certain discrete points can be used to simplify the search for the minimum cost. Without loss of generality, let us consider only one possible orientation of the splitting plane, for instance perpendicular to the x-axis. Objects' $\mathcal{AB}$s can also be used instead of real objects. Fig. 4.3 shows an example of a scene with four objects and the corresponding graph of the estimated cost.

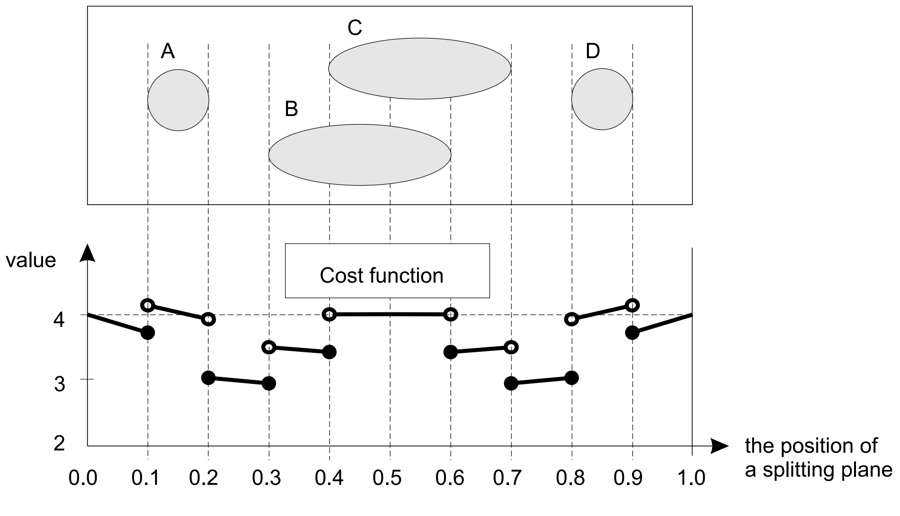


Figure 4.3: The value of the cost function in $\mathbb{E}^2$ for four objects along the axis. Objects' boundary positions play the key role in selecting the position of the splitting plane with minimum cost.

Assuming the number of objects straddling the splitting plane is N_{SP} and a linear cost estimate is used, we can formulate another variant of the cost function for surface area heuristic as:

$$C_{\nu^G} = \frac{1}{SA(\mathcal{AB}(\nu^G))}[SA(\mathcal{AB}(lchild(\nu^G))).(N_L + N_{SP}) + SA(\mathcal{AB}(rchild(\nu^G))).(N_R + N_{SP})], \quad (4.14)$$

The cost function that uses a linear estimate of the cost for child subtrees is a piece-wise continuous linear function. The discontinuity points are given by the objects' boundaries along the axis. The number of objects between two adjacent object boundary positions remains constant, and the cost function depends on the projected surface area only, which is a linear function in respect to the position of the splitting plane, which is b. This implies that the minimum value of the cost function can be found just at positions corresponding to object boundaries, *i.e.*, using a finite number of splitting plane positions. Since the cost function has discontinuities of the first order, its value has to be evaluated to be minimum at the discontinuity point that corresponds to taking the minimum number of objects in $\mathcal{AB}s$ associated with the left or the right child nodes.

We call the algorithm for positioning a splitting plane that minimizes the cost function given by Eq. 4.14 an *ordinary surface area heuristic* (abbreviated to OSAH). The algorithm computes the value of the cost function Eq. 4.14 for each position of all object boundaries within the $\mathcal{AB}$ to be split in all three axes. As a result, it selects the position of the splitting plane with the minimum value of the cost function.

4.2.4 Termination Criteria

The cost model gives us a recipe for positioning the splitting plane in this interior node, but it does not tell us whether we should proceed to subdividing the node further, or should declare the node as a leaf. Any node ν in the *kd*-tree has some basic characteristics: its depth $d(\nu)$ from the root node, $\mathcal{AB}(\nu)$ associated with ν, and the number of objects N intersecting $\mathcal{AB}(\nu)$. The construction of the *kd*-tree implies that the number of objects intersecting the node's $\mathcal{AB}$ decreases with increasing depth of the node. The positioning of the splitting plane on the object median implies that the number of objects will be one half of it, but this construction method, as we have remarked, is not suitable for an *RSA*. The number of objects also need not decrease significantly in each subdivision step, since some objects can straddle the splitting plane. We now face the question: *to subdivide or not to subdivide*? This issue in the context of hierarchical spatial subdivisions is usually called *termination criteria.*

We can view the termination criteria in terms of the *kd*-tree cost model (Eq. 4.9). We want to get an average height of the *kd*-tree that results in some "pseudo-minimum" total cost of the *kd*-tree. When the average height of the *kd*-tree is increased, the average number of traversal steps is increased and the number of ray-object intersection tests is expected to decrease – it may be possible to reach some pseudo-minimum cost point depending on the maximum allowed depth of the *kd*-tree. Below we discuss some common termination criteria.

4.2.4.1 Ad Hoc Termination Criteria

Ad hoc termination criteria were developed with the introduction of *RSAs* based on the BSP tree [94] and octree [59]. They are easily formulated, as already mentioned: the current node ν becomes a leaf when the number of objects intersecting the $\mathcal{AB}(\nu)$ is lower than or equal to a fixed constant N_{max}, or its depth $d(\nu)$ in the *kd*-tree reaches another fixed constant d_{max}. These two constants are specified by a user.

The values of these two constants N_{max} and d_{max} are left to the user's experience and practice in rendering systems based on ray shooting (Mental Ray [137]). N_{max} is usually one [94], we are not aware of any recommendations for maximum leaf depth d_{max}. In software packages some default values are provided, Mental Ray [137] has the default values $d_{max} = 24$ and $N_{max} = 4$ regardless of the number of objects in the scene and other scene complexity characteristics (see Section 3.3).

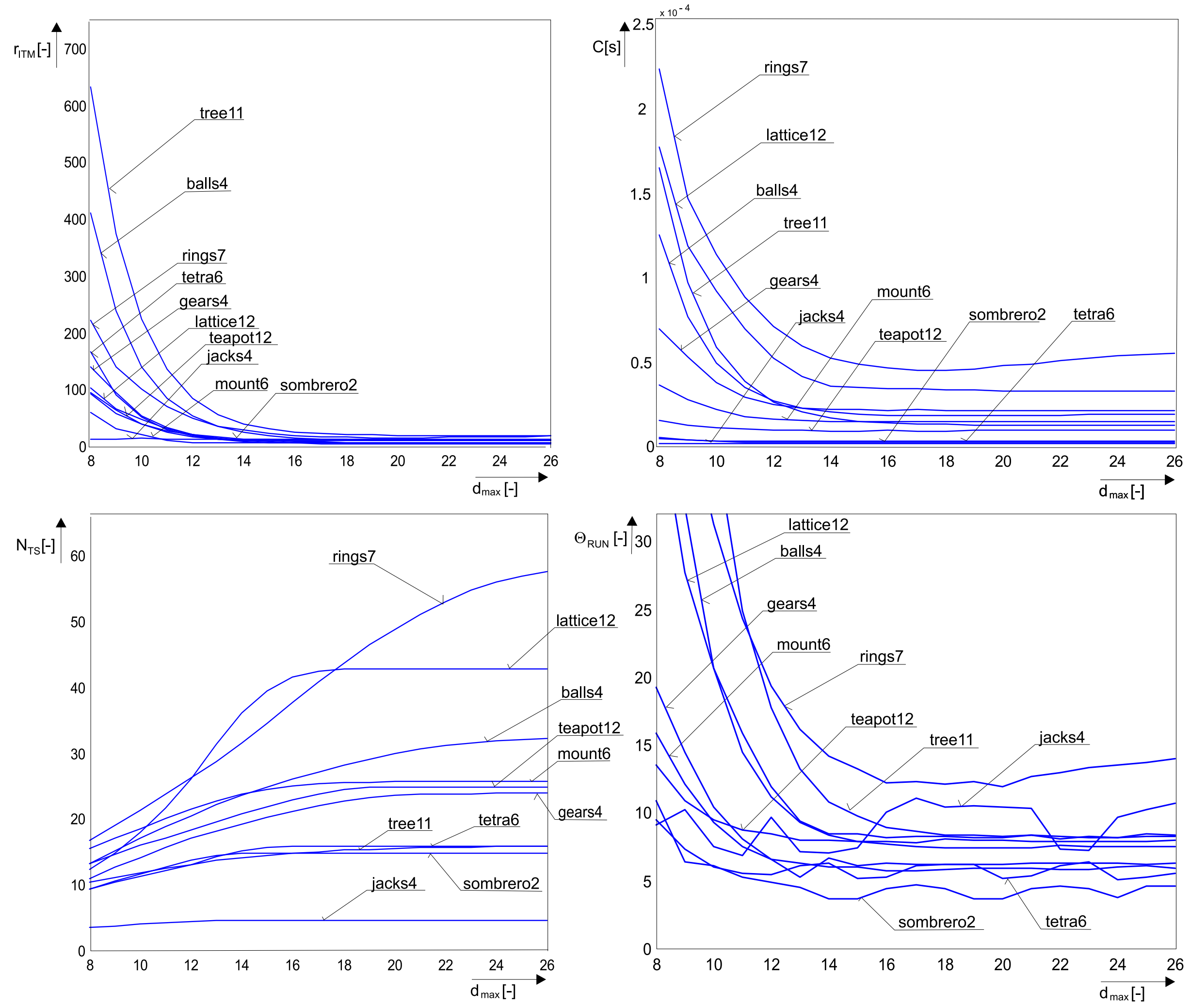


Figure 4.4: The cost in dependence on d_{max}, measured values for TP_D and G^4_{SPD}. Left top shows ratio of ray-object intersection tests performed to minimum number of intersection tests. Left bottom shows the number of traversal steps per ray. Right top shows the average running time per ray, and right bottom shows the cost normalized to the ideal ray shooting time. (See Chapter 2 for details.)

Setting the maximum leaf depth d_{max} limits the memory used by the *kd*-tree, since it restricts the number of *kd*-tree nodes to a constant, more precisely to $2^{d_{max}}$. When d_{max} is too low, the number of objects in the leaves remains high even if further subdivision steps could bring performance improvement. It has been shown by Subramanian and Fussel [146] that the estimated cost of a *kd*-tree for a particular scene has some critical point with regard to d_{max}. The dependence of the measured cost on d_{max} is shown in Fig. 4.4 for *SPD* scenes for G^4_{SPD}.

We can see from the graphs that increasing the maximum leaf depth d_{max} behind the critical point ($d_{max} \approx 16 \pm 2$), which is specific for each scene, does not bring any significant improvement in the total cost. It can even happen that the total cost increases, as it clearly does for the scene “rings”. Subramanian and Fussel [146, 143] do not discuss any method how for detecting the critical point or any algorithm for termination criteria utilizing this property of the *kd*-tree. We deal with this issue below.

4.2.4.2 Automatic Termination Criteria

Subramanian and Fussel [146, 143] coined the term *automatic termination criteria*, which should not require any user specific constants, but they did not propose any particular algorithm. Motivated by the idea of automatic termination criteria, we have designed an automatic termination criteria algorithm based on the cost model, and we will describe it in Section 4.5.

4.3 Analysis of the Cost Model

In this section we perform the initial analysis of the cost model that is required for a better understanding of the following sections. The analysis concerns the pure geometry view of the splitting and minimization of the cost of the whole *kd*-tree.

4.3.1 Splitting Geometry of an Axis-Aligned Bounding Box

The cost function of surface area heuristic can be investigated from the purely geometric viewpoint of splitting the $\mathcal{AB}(\nu)$ associated with the interior node ν into two halves. Until now we assumed that rays are uniformly distributed in space, which enables us to use Eq. 4.4 for the geometry of the split $\mathcal{AB}(\nu)$, as depicted in Fig. 4.5. Let the size of $\mathcal{AB}(\nu)$ be described by width w, height h, and depth d. The splitting plane is positioned to be perpendicular to the coordinate axis that corresponds to the width, so the $\mathcal{AB}$ associated with the left child has its right boundary at a distance w_L from the left side of $\mathcal{AB}(\nu)$.

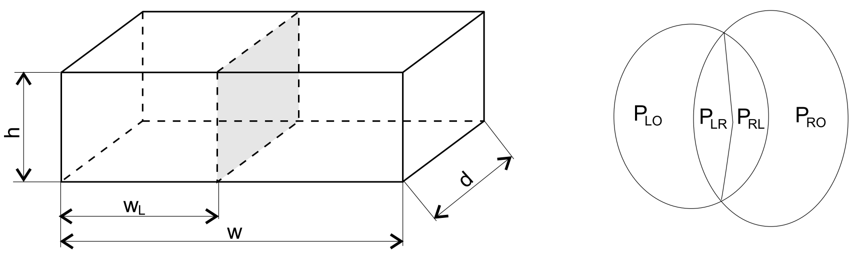


Figure 4.5: Geometry of split node's $\mathcal{AB}$.

Let us denote the surface areas of all $\mathcal{AB}s$ that are induced by the geometry of the split $\mathcal{AB}(\nu)$: $SA(\mathcal{AB}(\nu)) = 2.(w.h + h.d + d.w)$ be the surface area of the node $\mathcal{AB}(\nu)$, $SA(\mathcal{AB}(lchild(\nu))) = 2.(w.h + h.w_L + w_L.d)$ the surface area of $\mathcal{AB}(lchild(\nu))$ associated with the left child of ν, $SA(\mathcal{AB}(rchild(\nu))) = 2.(w.h + h.(w - w_L) + d.(w - w_L))$ the surface area of the $\mathcal{AB}(rchild(\nu))$ associated with the right child of ν, and $SA(SP) = 2.d.h$ the surface area of the splitting plane restricted to $\mathcal{AB}(\nu)$. Obviously, it holds for the notation above that $b = w_l/w$. We can compute the probabilities of all four possible traversal cases that occur in the recursive ray traversal algorithm:

$$p_{LO} = (SA(\mathcal{AB}(lchild(\nu))) - \frac{1}{2}.SA(SP))/SA(\mathcal{AB}(\nu)) \tag{4.15}$$

$$p_{RO} = (SA(\mathcal{AB}(rchild(\nu))) - \frac{1}{2}.SA(SP))/SA(\mathcal{AB}(\nu)) \tag{4.16}$$

$$p_{LR} = p_{RL} = \frac{1}{2}.SA(SP)/SA(\mathcal{AB}(\nu)), \tag{4.17}$$

where	p_{LO}	–	probability of a ray hitting the left child node only,
	p_{LR}	–	probability of a ray hitting the left child first and the right child afterwards,
	p_{RO}	–	probability of a ray hitting the right child node only,
	p_{RL}	–	probability of a ray hitting the right child first and the left child afterwards.

Obviously, $p_L + p_R + p_{LR} + p_{RL} = const > 1.0$. The size of $\mathcal{AB}(\nu)$ can be arbitrary, provided $h > 0$, $d > 0$, and $w > 0$. Assuming objects associated with $\mathcal{AB}(\nu)$ do not overlap in the projection to the coordinate axes and the objects are uniformly distributed in space, we can investigate the value of the cost function with regard to the choice of orientation of the splitting plane. Without loss of generality we further assume $h < d < w$.

If $\mathcal{AB}(\nu)$ is split on the largest side of size w, then the average number of child nodes hit by a ray N^w_{TV} is as follows:

$$\begin{aligned} N^w_{TV} &= (SA(\mathcal{AB}(lchild(\nu)))(w) + SA(\mathcal{AB}(rchild(\nu)))(w))/SA(\mathcal{AB}(\nu)) \\ &= 2.h.d/SA(\mathcal{AB}(\nu)) + 1 \end{aligned} \tag{4.18}$$

Similarly, if the splitting plane is oriented at $\mathcal{AB}(\nu)$ so that it splits the side of size d and h, we get the average number of child nodes hit by a ray as:

$$\begin{aligned} N^d_{TV} &= (SA(\mathcal{AB}(lchild(\nu)))(d) + SA(\mathcal{AB}(rchild(\nu)))(d))/SA(\mathcal{AB}(\nu)) \\ &= 2.w.h/SA(\mathcal{AB}(\nu)) + 1 \end{aligned} \tag{4.19}$$

$$\begin{aligned} N^h_{TV} &= (SA(\mathcal{AB}(lchild(\nu)))(h) + SA(\mathcal{AB}(rchild(\nu)))(h))/SA(\mathcal{AB}(\nu)) \\ &= 2.w.d/SA(\mathcal{AB}(\nu)) + 1 \end{aligned} \tag{4.20}$$

Since we assumed $h < d < w$, we can easily prove that $N^w_{TV} < N^d_{TV} < N^h_{TV}$. For example, for the size of the $\mathcal{AB}$ $d = 2.h, w = 3.h$ we get $N^w_{TV} = 4/22 + 1 \doteq 1.18$, $N^d_{TV} = 6/22 + 1 \doteq 1.27$, and $N^h_{TV} = 12/22 + 1 \doteq 1.55$. We can then suppose that it should be advantageous to split the $\mathcal{AB}$ in the axis corresponding to the largest side of the $\mathcal{AB}$. Since the distribution of objects in the projection to the chosen axis is not generally uniform, and objects may overlap, we cannot rely on this assumption. However, we see that the estimated cost is biased with the geometry given by $\mathcal{AB}(\nu)$. The orientation of the splitting plane that results in the minimum estimated cost cannot be chosen in advance; the cost function must be evaluated in all three axes for the boundaries of the all objects, thus possibly at $3.2.N = 6.N$ positions for N objects. At least we have shown that the selection of an axis of the splitting plane is crucial for minimizing the estimated total cost of the *kd*-tree, and thus this orientation cannot be chosen in advance. The orientation of the splitting plane changed in cyclic order [94, 148] in BSP tree construction, starting with the x-axis, is thus only one possible way, which may be rather inconvenient especially for the oblong shape of the scene $\mathcal{AB}$.

4.3.2 The *Kd*-Tree with Minimum Total Cost

Even if the cost model is based on some unrealistic assumptions, we can ask whether it is possible to construct a *kd*-tree that according to Eq. 4.9 has the minimum estimated cost from all *kd*-trees. Obviously, this task is not solved by selecting the minimum cost splitting plane using Eq. 4.14 in the currently processed interior node, since it uses a linear estimate for the cost of the subtrees that have not yet been constructed. The linear estimate is valid only for the case when the subtree is not further subdivided, *i.e.*, for a leaf. Therefore, it is possible to compute the estimated cost of the node ν correctly according to Eq. 4.9 after the process of the *kd*-tree construction is finished for the whole subtree rooted at ν.

In order to get the global minimum estimated cost for the whole *kd*-tree according to Eq. 4.9, it is necessary to take into account each possible position of the splitting plane for all nodes within the construction (at most $6.N$ positions for N objects pointed to in the current node). For all these positions we have to construct left and right subtrees, both of them again with the minimum estimated cost. Then we can combine the obtained costs using Eq. 4.13 to compute the estimated cost of the current node. As the final step, we must select a splitting plane that corresponds to the minimum total estimated cost. This algorithm is of a recursive nature, and this causes the problem of obtaining the *kd*-tree with minimum total cost according Eq. 4.9 to be *NP*-hard. Proving this statement formally correctly would be rather difficult and lengthy, it requires polynomial time reduction from thhe kd-tree construction algorithm to a problem whose *NP*-hardness has already been proven.

4.4 Construction of *Kd*-Trees with Utilization of Empty Spatial Regions

The assumption that splitting planes do not intersect objects limits the position of the splitting plane with minimum cost to the median interval. It can be supposed that both sides of the splitting plane contain some object(s), which obviously holds for the root node of the *kd*-tree, because we use the tight $\mathcal{AB}s$ of all objects. It is not the case for non-root interior nodes of the *kd*-tree; the splitting plane need not have any objects on one of its sides. If a splitting plane is positioned so that one child node contains no objects, the child is not further subdivided and it is declared a leaf. We call such a positioning of the splitting plane *cutting off empty space*. The complementary view to cutting off empty space is that the splitting plane is the face of a bounding volume enclosing the objects in the node.

Subramanian and Fussel [144, 145] first noticed the utilization of empty space in the *kd*-tree, but their view of the use of empty space within the hierarchy is inconsistent. First, they present the following opinion, quoted from their paper [145]:

> To sum things up, the creation of empty voxels is itself a source of inefficiency in the first place since no ray-object intersection can be found in such regions, but the fact that they are not subdivided any further is a desirable factor since it helps the structure adapt itself to the locations and densities of the primitives in the input scene.
>
> .... The void space represents a measure of three dimensional space that contains no useful information.

End of quotation.

They also provide a "void area measure" in the *kd*-tree that we consider rather incorrect. They propose to construct tight $\mathcal{AB}s$ for all the objects pointed to in the left and right child, and to compute the "void area measure" as the difference of volume of the $\mathcal{AB}$ of the original node minus the volumes of the tight $\mathcal{AB}s$ of the child nodes. Since the child nodes are further subdivided, some empty space within the hierarchy can be included in the total sum for the whole *kd*-tree several times (at most six times, since $\mathcal{AB}$ has six faces in $\mathbb{E}^3$). They also claim that their "void area measure" is a useful measure of performance of the *kd*-tree when used for ray shooting. On the other hand, they apply tight $\mathcal{AB}s$ included into the interior nodes, we again quote the paper [145]:

> ..... By surrounding collections of objects completely with bounding volumes, large sections of object space can be pruned away, drastically reducing the ray search space. This is because if a ray misses this bounding volume, its contents need not be examined. Bounding volumes are more effective than space partitioning structures in this respect, since, in general, they provide tighter enclosures around object collections.
>
> .... Space partitioning structures are adaptive, concentrating the partitioning in the vicinity of objects. Since all the partitioning planes are axis-aligned, they have the potential to create large void spaces which are a source of inefficiency. Bounding volume hierarchies, on the other hand, optimize ray tracing by culling away large sections of object space and provide a compact representation for the objects at every node of the hierarchy.
>
> ... Thus bounding volumes should be used only where they result in culling a large amount of void space.

End of quotation.

Their two views to a use of empty space are contradictory. First, they want to use empty space using $\mathcal{AB}s$ included into the interior nodes, which can be relatively costly to check for intersection with a ray. Second, they want to avoid the creation of empty leaves, even if this can be understood as another variant of the $\mathcal{AB}s$ in the interior nodes.

4.4.1 Theoretical Remarks

Before we describe improved versions of *kd*-tree construction using cutting off empty space, we present our motivation for this concept. The recursive ray traversal algorithm on the *kd*-tree constructed with "good" utilization of empty space should behave for any ray as follows: after descending not very deep from the root node to the first leaf containing the origin of a ray, only empty leaves of possibly large size should be traversed until the first full leaf with one object is hit and the ray intersects this object. In this case the ray need not be tested for intersection with any other objects. With increasing size of the empty leaves, the ray must traverse only a few leaves to get to the first full leaf; so the cost of the *kd*-tree for ray shooting is decreased. The empty leaves contain useful information exactly by virtue of their emptiness; since they are empty, the spatial region that they cover can be skipped within a ray traversal algorithm. No objects can be intersected in empty leaves, and this is the source of potential efficiency improvement. The basic underlying idea in this context of visibility is as follows: when an object from a viewpoint is visible (along a half-line), there must be empty space in front of this object.

Let us describe how *kd*-tree construction with utilization of empty spatial regions could proceed. After localizing empty space in *kd*-tree we can organize empty space to stay in the leaves in upper levels of the *kd*-tree hierarchy and then split the leaves, where objects are located and the probability that a ray will intersect the object is high. One can then propose a general algorithm for constructing the *kd*-tree with "good" utilization of empty spatial regions:

1. For a given scene $\mathcal{S}$ of N objects, identify all "empty space" within the scene $\mathcal{AB}$. Represent this empty space in the set S_{ER} of non-overlapping $\mathcal{AB}s$. For such a set S_{ER} holds $\sum_{i \in S_{ER}} Vol(\mathcal{AB}_i) = const$, where $Vol(\mathcal{AB}_i)$ is the volume of $\mathcal{AB}_i$. The optimization criteria to search the optimum set S_{ER} can be derived from Eq. 4.4, since for an arbitrary ray we want to get the minimum number of empty spatial regions to be intersected:

$$S_{ER}^{opt} = \{\forall S_{ER}; \sum_{i \in S_{ER}^{opt}} SA(\mathcal{AB}_i) \leq \sum_{j \in S_{ER}} SA(\mathcal{AB}_j)\}$$

2. select the subset $sel(S_{ER}^{opt})$ of empty $\mathcal{AB}s$ from S_{ER}^{opt}, which represents most of the empty space in the hierarchy.

3. construct the *kd*-tree over scene objects and empty $\mathcal{AB}s$ from $sel(S_{ER}^{opt})$, which uses some variant of surface area heuristic. Try not to split both the objects and the empty $\mathcal{AB}s$. One way could be to consider the $\mathcal{AB}s$ of $sel(S_{ER}^{opt})$ as objects.

When the objects were non-overlapping $\mathcal{AB}s$, an *RSA* using the corresponding *kd*-tree could be efficient. After locating the first full leaf a ray traversal algorithm terminates, since the ray-object intersection must occur. If $\mathcal{AB}s$ only approximate objects – intersection occurs with a certain probability – it is difficult to predict the advantage of surface area heuristic that use empty spatial regions over OSAH. This is also the case when the $\mathcal{AB}s$ associated with the objects overlap.

The outline of the *kd*-tree construction method that uses the empty space gives us several other subproblems. One of them is how many empty spatial regions exist for S_{ER}^{opt}, given a scene $\mathcal{S}(N)$

containing N objects. Let us suppose that the scene consists of non-overlapping objects having the shape of $\mathcal{AB}$. Since the number of faces for objects' $\mathcal{AB}s$ is finite, the boundary between the empty spatial regions and object spatial regions contains a finite number of corners and edges, and is thus $O(N)$. The number of possible empty spatial regions is then also $O(N)$.

A difficult problem is the algorithm for constructing the set S_{ER}^{opt}. The problem was studied in $\mathbb{E}^2$ by Lingas *et al.* [103] (cited according to [42]), and is called the *Minimum Edge Length Rectangular Partition Problem*. They formulate it as follows: given a rectilinear figure, partition it into rectangles with the minimum total length of new boundaries. Lingas *et al.* [103] show that this problem is *NP*-hard. Although we do not prove it here formally, we cannot expect that in $\mathbb{E}^3$ space the problem will not be *NP*-hard. Moreover, we have to describe the empty space boundary as the complementary space of the $\mathcal{AB}s$ associated with the objects in the scene. We can use some heuristic algorithm to get an approximative solution for S_{ER}^{opt}, but for $\mathbb{E}^3$ space we are not aware that any such algorithm has been published. For $\mathbb{E}^2$ some heuristics are presented by Du and Zhang [42].

Although the use of empty space as described above has its potential, the time complexity of *kd*-tree construction would become unacceptable for practical applications. Instead of using a general concept for cutting off empty space, we describe below three simpler methods based on the cost model.

4.4.2 Early Cutting Off Empty Space

Cutting off empty space can occur when the estimated cost computed according to Eq. 4.13 is the minimum found. For this reason computation of the cost function should be performed on the whole range of the $\mathcal{AB}$ for all three axes to find out the minimum estimated cost. We call this case *early cutting off empty space*. It occurs when the geometry term of the space to be cut off outweighs the other terms in the cost function, resulting in a reduction in the total cost. Assuming the left child node is empty, Eq. 4.13 simplifies to:

$$\hat{C}_{\nu^G} = \hat{C}_{TS} + p_R.\hat{C}_R \tag{4.21}$$

Early cutting off empty space creates new empty leaves possibly in upper levels of the *kd*-tree hierarchy: large empty spatial regions can be skipped during the ray traversal algorithm as described above.

4.4.3 Late Cutting Off Empty Space

Mostly, the termination criteria for *kd*-tree construction as already mentioned are a fixed number of objects pointed to in the leaf and the maximum depth of the leaf in the hierarchy. It seems to be uninteresting to split a leaf node containing one object only. Below we show that this case should also be investigated.

Let node ν of *kd*-tree is associated wih $\mathcal{AB}(\nu)$ representing the whole scene that consists of one object only. The cost function (Eq. 4.12) is then expressed as:

$$\hat{C}^1_\nu = \sum_{i=1}^{1} \hat{C}_{IT}(i). \tag{4.22}$$

Let us suppose the $\mathcal{AB}(\nu)$ is split by a plane in such a way that the object remains in the right child node only, whereas the left child node is empty. Then using the cost function Eq. 4.21 we get the following new cost function:

$$\hat{C}^2_\nu = \hat{C}_{TS} + p_R.(\sum_{k=1}^{1} \hat{C}_{IT}(1)) = \hat{C}_{TS} + p_R.\hat{C}_{IT}. \tag{4.23}$$

The position of the splitting plane influences only the last term in the cost function, Eq. 4.23. Since the traversal cost $\hat{C}_{TS}$ is included, the cost $\hat{C}^2_\nu$ could also be higher than the original cost $\hat{C}^1_\nu$. The selection of the minimum cost using either Eq. 4.23 or Eq. 4.22 is thus sensitive to ratio between $\hat{C}_{TS}$ and $\hat{C}_{IT}$ unlike the early cutting off empty space. Obviously, the cutting off empty space method discussed here should be applied only if $\hat{C}^2_\nu < \hat{C}^1_\nu$. Since this occurs in the leaves of the *kd*-tree, in the late phase of the *kd*-tree construction, we call the method *late cutting off empty space*.

The real values of C_{TS} and C_{IT} depend on the implementation of a ray traversal algorithm and ray-object intersection tests. We deal with efficient ray traversal algorithms further in Chapter 5. The time of the ray-object intersection test C_{IT} depends on a particular algorithm and implementation, and is related to the object's shape. Simple geometrical objects like spheres and triangles can be tested for intersection at a cost comparable to C_{TS}, however, C_{IT} for these simple objects is still higher than C_{TS} for an efficient recursive ray traversal algorithm. Complex objects like NURBS have C_{IT} by order(s) of magnitude higher than simple objects, therefore late cutting off empty space typically pays off for them. Since *SPD* scenes consist of simple geometrical objects only, cutting off empty space need not decrease the estimated cost considerably.

In Fig. 4.6 we can see the difference between late and early cutting off empty space methods. Whenever possible, early cutting off empty space should be preferred, since it creates the *kd*-tree with fewer nodes, and thus the ray traversal algorithm is in an average case more efficient under the assumption of uniformly distributed rays.

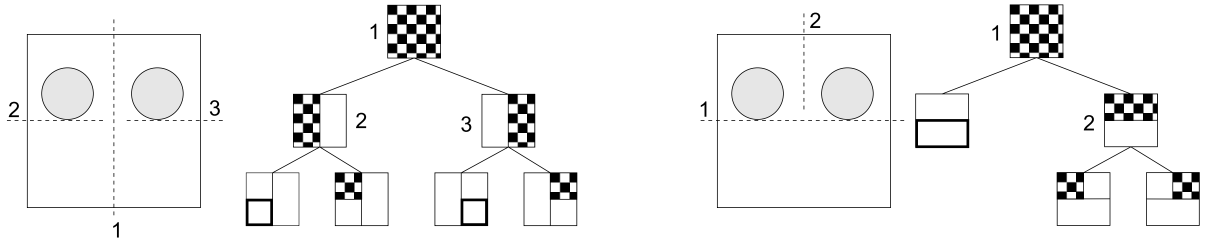


Figure 4.6: A spatial region with empty space and a corresponding *kd*-tree. Late cutting off empty space (left) is less effective than early cutting off empty space (right).

We can compare late cutting off empty space for a single object with the use of bounding volumes in the interior nodes [145] of the *kd*-tree. Usually, as the bounding volume of a *single object* is taken its $\mathcal{AB}$. This bounding volume is checked for intersection with a ray before the ray-object intersection test is performed. For the *kd*-tree, the use of $\mathcal{AB}$*s* is particularly suitable since the shape of an $\mathcal{AB}$ fits the geometry of the *kd*-tree. Even if efficient algorithms for computing an intersection between a ray and $\mathcal{AB}$ are known (for a survey, see [112]), the cost of the intersection test is relatively high compared with the cost of traversal step C_{TS}. It requires at least from two to six intersection tests with the planes of an $\mathcal{AB}$ that are perpendicular to the coordinate axes, and other computational effort, even if the $\mathcal{AB}$ is not intersected. Unlike the $\mathcal{AB}$ used as a bounding volume and put in the interior node of the *kd*-tree, late cutting off empty space puts splitting planes only for the objects' sides (at most six planes in $\mathbb{E}^3$), where this is advantageous from the viewpoint of the cost model. In the worst case, all six bounding planes can be put around the object, which is unlikely to occur, since in this case the use of $\mathcal{AB}$ as a bounding volume is less costly.

We can also compare the use of bounding volumes for a *set of objects* either in the leaf or in the interior node. Further, we again assume that we use as the bounding volume an $\mathcal{AB}$ pointed to in the node ν. When the recursive ray traversal algorithm reaches the node ν, it is required to have a computed entry and exit signed distance from the origin of a ray with the $\mathcal{AB}$. This is necessary since the $\mathcal{AB}$ used as the bounding volume is then taken as a root of the subtree of the *kd*-tree that is further built. As for the single object, the use of the $\mathcal{AB}$ as the bounding volume is insensitive to the size of the empty space saved on each side of the $\mathcal{AB}$ used as the bounding volume in respect to the size of the $\mathcal{AB}(\nu)$

associated with the node ν.

4.4.4 Two-Plane Cutting Off Empty Space

In addition to early and late cutting off empty space methods, we can ask about the existence of an algorithm as some cheap alternative to the algorithm proposed in Section 4.4.1. This algorithm should find large empty spatial regions and cut them off from the rest of the *kd*-tree hierarchy. The problem with the *kd*-tree is that whenever a node is subdivided, empty spatial regions on both sides of the splitting plane cannot ever be connected again to one leaf node, see Fig. 4.6 (left). If possible, we want to detect these situations, and try to promote the splitting that results in the creation of empty leaves, but we want to use the cost model for our decision.

Therefore, we further propose and analyze a simple method that searches for an interval of empty space on one axis only. The technique is an extension of early cutting off empty space, and further promotes the creation of empty leaves in the upper levels of the *kd*-tree hierarchy. The underlying geometry is depicted in Fig. 4.7.

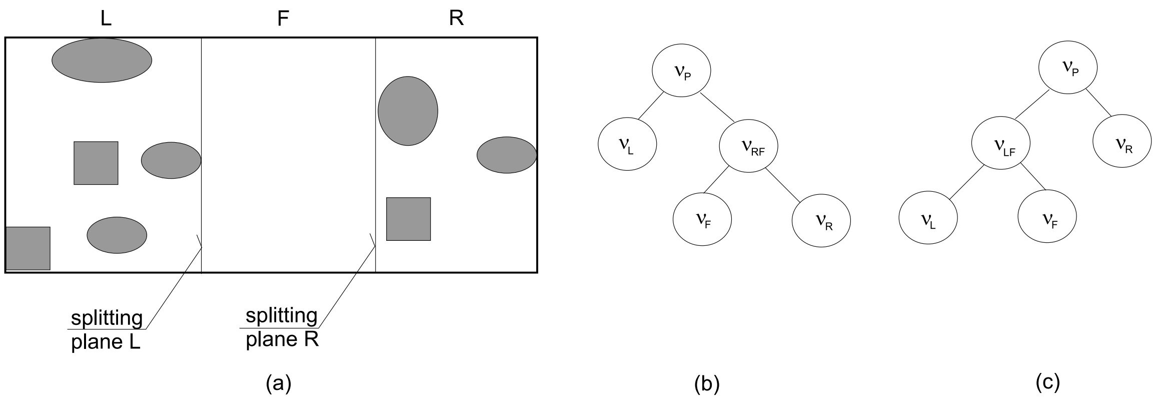


Figure 4.7: Two-plane cutting off empty space. (a) Spatial region with empty space inside the interval along the axis. (b) RF cutting. (c) LF cutting.

When a *kd*-tree is constructed as depicted in Fig. 4.7 (b) or (c), we can refine the cost model accordingly. Further, we assume that two subdivision steps are performed as depicted in Fig. 4.7 (b). In this case, the cost for parent node ν_P corresponds to Eq. 4.14. Then the empty space is cut off in the next step, so the cost of the node is:

$$\hat{C}^{RF} = \hat{C}_{TS} + SA(\mathcal{AB}(\nu_R))/SA(\mathcal{AB}(\nu_{RF})).\hat{C}_R$$

Then the total estimated cost for the parent node becomes:

$$\begin{aligned} \hat{C}^{RF}_{\nu_P} = \quad & SA(\mathcal{AB}(\nu_L))/SA(\mathcal{AB}(\nu_P)).\hat{C}_L + SA(\mathcal{AB}(\nu_R))/SA(\mathcal{AB}(\nu_P)).\hat{C}_R + \\ & (SA(\mathcal{AB}(\nu_{RF}))/SA(\mathcal{AB}(\nu_P)) + 1).\hat{C}_{TS} \end{aligned} \tag{4.24}$$

Similarly, when the *kd*-tree is constructed as shown in Fig. 4.7 (c), we can estimate the cost for this way of splitting as:

$$\begin{aligned} \hat{C}^{LF}_{\nu_P} = \quad & SA(\mathcal{AB}(\nu_L))/SA(\mathcal{AB}(\nu_P)).\hat{C}_L + SA(\mathcal{AB}(\nu_R))/SA(\mathcal{AB}(\nu_P)).\hat{C}_R + \\ & (SA(\mathcal{AB}(\nu_{LF}))/SA(\mathcal{AB}(\nu_P)) + 1).\hat{C}_{TS} \end{aligned} \tag{4.25}$$

We compute the cost function using Eq. 4.25 and Eq. 4.26 only when the empty space interval is identified within the search for the minimum cost evaluating Eq. 4.14. We select the case LF or RF that uses two predetermined splitting plane positions only if it has the cost lower than the cost according to Eq. 4.14 for the boundaries of all objects' along the axis. We can see that this method is also sensitive to the ratio between the value of C_{TS} and C_{IT}, like late empty space cutting off.

It is obvious that two-plane cutting off empty space can also be the case of two successive subdivision steps in OSAH. In this case it requires both splitting planes to have a minimum estimated cost independently. The combination of two subdivision steps into one equation further promotes the creation of empty leaves; it reveals the geometry with the empty space one step in advance.

4.5 Automatic Termination Criteria

As we described the term in Subsection 4.2.4.2, automatic termination criteria are termination criteria that do not require any user setting. In this section we deal with the design of automatic termination criteria. Looking at the graphs in Fig. 4.4 we could require that they result in a *kd*-tree with at least the critical performance point as depicted on the graphs.

A simple automatic termination criteria algorithm can follow the curves on these graphs. We can construct a *kd*-tree for some maximum leaf depth d_{max} and compute the estimated cost of the *kd*-tree $C(d_{max})$ using Eq. 4.9. Then we subdivide all the leaves of the *kd*-tree one step further (if possible) to the maximum leaf depth $(d_{max}+1)$, and we compute the new estimated cost $C(d_{max}+1)$ of this deepened *kd*-tree. When $C(d_{max}+1)$ is sufficiently lower than $C(d_{max})$, the just performed subdivision step improves the estimated cost and the *kd*-tree can be further deepened to depth $(d_{max}+2)$. Otherwise, we should not continue subdividing.

However, incrementally increasing maximum leaf depth d_{max} for the whole *kd*-tree construction is not the only way to get the pseudo-minimum of the estimated cost. Actually each node ν to be potentially subdivided has its own history of splitting that is associated with all the nodes on the path between the root node and the node ν. Setting only one maximum leaf depth d_{max} for the whole *kd*-tree is not very appropriate to the problem. Automatic termination criteria should keep track of the history of splitting the nodes of the *kd*-tree.

We should remark that one subdivision step need not necessarily bring an improvement in cost immediately. It is possible that the estimated cost increases due to a current subdivision step, even if the splitting plane with minimum cost is selected. This can be caused by two possible reasons. First, the splitting plane could not separate enough objects. In this case, most objects are intersected by the splitting plane, thus a recursive ray traversal algorithm can require one more traversal step. Second, the linear cost estimate of the unsubdivided child node is just an estimate. If a subdivision step is unsuccessful, it does not mean that further subdividing of the corresponding child nodes cannot decrease the total estimated cost of the *kd*-tree considerably. If many unsuccessful subdivision steps occur on the path between the root node and the current node, this would indicate that objects in these spatial regions were difficult to separate. Avoiding further subdivisions steps in this case could decrease the cost of the *kd*-tree.

We observed during experiments with the termination criteria algorithm that using some maximum leaf depth d_{max} is a useful feature of the termination criteria algorithm. First, the use of d_{max} bounds the maximum memory requirements for the *kd*-tree representation by limiting the number of *kd*-tree nodes to a constant. Second, since $\mathcal{AB}$s associated with objects can overlap, they need not be separable by a splitting plane at all. In this case the number of objects cannot become lower than or equal to a constant N_{max}. If the objects with the shape of $\mathcal{AB}$ do not overlap in the projection to all coordinate axes and the *kd*-tree with one object in every leaf is required, the maximum depth of a leaf node for the balanced *kd*-tree is $d_{max} = \log N$. When we assume arbitrarily distributed objects in the scene with possible overlapping of objects, then the maximum leaf depth d_{max} to achieve the critical cost point could be higher. To bound this quantity we propose to express the maximum leaf depth as:

$$d_{max} = k_1 . \log N + k_2 \tag{4.26}$$

The values of constants k_1 and k_2 can be chosen according to experiments on some set of scenes to achieve the critical performance point. We found by averaging for experiments on 30 *SPD* scenes (group G^3_{SPD}, G^4_{SPD}, and G^5_{SPD}) that one possible setting of these constants in the GOLEM rendering system is $k_1 = 1.2, k_2 = 2.0$.

Further, we discuss the setting of the constant N_{max}. Our observation is based on the case when we access a leaf with N_{max} objects during a ray traversal algorithm. Assuming that the cost of traversal step C_{TS} is sufficiently lower than the cost of ray-object intersection test C_{IT}, it should be advantageous have $N_{max} = 1$. Then in the ideal case in the constructed *kd*-tree the leaves contain either one or no object. First, the empty leaves are traversed quickly. Second, full leaves with one object are also advantageous for efficiency reasons. If a leaf with one object is checked for ray-object intersection and the intersection exists, no further ray-object intersections need to be performed. Placing one object to leaves is not always possible, since objects or/and $\mathcal{AB}$*s* associated with the objects can overlap. Therefore, during the construction we have to detect these branches of the *kd*-tree where further deepening does not bring any improvement of the whole cost of the *kd*-tree. In order to detect these cases we also use the cost model.

Let us describe the use of the cost model in automatic termination criteria algorithm. When the node ν with N objects is subdivided, its resulting estimated cost $\hat{C}_{\nu^G}$ is computed according to Eq. 4.13. When the node ν is declared a leaf, its cost $\hat{C}_{\nu^E}$ is expressed by the Eq. 4.12. Then we can express the ratio of these two costs, which shows the *quality* of the subdivision step:

$$\begin{aligned} r_q &= \frac{\hat{C}_{\nu^G}}{\hat{C}_{\nu^E}} \\ &= \frac{\hat{C}_{TS} + \hat{C}_{IT}.(SA(\mathcal{AB}(lchild(\nu^G))).N_L + SA(\mathcal{AB}(rchild(\nu^G))).N_R)/SA(\mathcal{AB}(\nu^G))}{\hat{C}_{IT}.N} \end{aligned} \tag{4.27}$$

The higher the quality of the subdivision step the lower r_q: for a successful subdivision step we assume $r_q < 1.0$. For example, if a node with $\mathcal{AB}$ of cubic shape that contains uniformly distributed objects is subdivided in its spatial median, then for $C_{TS} = 0$ we get the quality of the subdivision $r_q = 2/3$. If r_q is a constant greater than constant r_q^{min} (we can set r_q^{min} to one or some constant slightly lower than one), we can consider the subdivision step unsuccessful. However, the case that $r_q > r_q^{min}$ can occur only transiently, and the quality of subdivision step r_q can improve again in the subsequent subdivision steps. Therefore, we should keep track of the number of unsuccessful subdivision steps on the path from the root node to the current node. If the number of these unsuccessful subdivision steps is higher than the allowed fixed constant F_{max}, we declare the node as a leaf, since further subdivision steps are unlikely to decrease the total estimated cost of the *kd*-tree. We propose to compute F_{max} from the maximum leaf depth d_{max}, since for scenes with higher number of objects the number of allowed unsuccessful steps could be higher, and d_{max} is derived from the number of objects. We propose to compute F_{max} as follows:

$$F_{max} = K^1_{fail} + K^2_{fail}.d_{max} \tag{4.28}$$

We found one possible setting for the constants empirically in the GOLEM rendering system: $K^1_{fail} = 1.0$, $K^2_{fail} = 0.2$ and $r_q^{min} = 0.75$.

The automatic termination criteria algorithm described here is an empirical algorithm based on experiments using test scenes G^3_{SPD}, G^4_{SPD}, and G^5_{SPD}. We do not claim its optimality or the best setting of constants given above. Such setting of constants is implementation dependent, and particularly, it depends on the ratio C_{TS}/C_{IT}. The advantage of the proposed algorithm for automatic termination criteria is that it does not require any user intervention, it limits maximum memory usage, and it results in a *kd*-tree with lower or comparable cost with the *kd*-tree built with ad hoc termination criteria. We verified experimentally the performance of *kd*-trees built with automatic termination criteria (Section 4.10.2).

4.6 Further Results and Problems

The development of the cost model and the surface area heuristic has been conditioned by several rather unrealistic simplifications. The first is the estimate of the cost of the child subtree to be subdivided; the estimate is linear with a number of objects, which corresponds to the worst-case complexity measure. We have shown the validity of this linear cost estimate only for the case when the node is not further subdivided, however, the goal is that the time complexity of the *RSA* based on the *kd*-tree should be much less than linear.

The second simplification is that we approximate objects by their $\mathcal{AB}$*s*, regardless of their tightness to the objects. Then the objects need not intersect the $\mathcal{AB}$*s* of the leaves where they are referenced, and this can also influence the construction of the *kd*-tree.

The third simplification is the assumption that an arbitrary ray does not intersect any object, which again corresponds to the worst-case complexity measure. This is obviously not true in general, since the case of the existence of an intersection between a ray and an object is a common result of an *RSA*.

The fourth simplification is the assumption of the uniformity of ray distribution, which can be strongly violated in general. Ray distribution depends greatly on the application.

Further, we present our study in which we try to avoid these simplifying assumptions. The third and fourth simplification mentioned above are elaborated in separate sections below, since the scope of the text is rather large.

4.6.1 Cost Estimate

Until now we have supposed that the worst case in the linear cost estimate is valid – when a node containing objects is not further subdivided, which corresponds to declaring the node as a leaf. For nodes of the *kd*-tree with many objects, such an estimate seems unrealistic. Since these nodes are further subdivided, their cost is decreased.

Estimating the cost of node ν, which is to be subdivided, is however difficult for general scenes. We can solve the problem in several ways. The first solution is precise and costly; we can really construct a subtree for node ν, then evaluate its estimated cost using Eq. 4.9. This leads to a combinatorial explosion of the construction algorithm, as we stated in Section 4.3.2, and this is not usable for a practical algorithm.

The second solution is that we can use for an estimate the results measured on the *SPD* scenes. Since these scenes can be generated with various numbers of objects by specifying the size factor S_F, we can get the function $C = f(N)$, where C is the cost of the *kd*-tree constructed using Eq. 4.14, and N is the number of objects. These functions for all the *SPD* scenes are plotted in Fig. 4.8. For constructing *kd*-trees in the experiments we applied automatic termination criteria.

Unfortunately, all the measured scenes have completely different functions $f(N)$, which cannot be expressed by a single function of one variable N. We also failed with the following approach: we computed the estimated cost $\hat{f}_{\mathcal{S}}(N)$ for a particular scene $\mathcal{S}$ as the function of one variable N. We applied the estimate during the construction of the *kd*-tree for the scene $\mathcal{S}$, however, the resulting cost of the *kd*-tree was higher than for the linear estimate. This is probably due to the fractal nature of the generated *SPD* scenes – increasing the number of objects need not imply a similar distribution of objects in the scene.

In our third attempt to solve the problem, we can try to estimate the cost from Eq. 4.9 under the following assumptions: a BSP tree with a spatial median method is built over a set of uniformly distributed objects. The input for the estimate of node ν is the number of objects N and the surface area $SA(\mathcal{AB}(\nu))$ of the $\mathcal{AB}$ associated with ν. We can estimate the average depth for the constructed BSP tree as $\tilde{d} = \log N + k_1$, $k_1 > 0$, since some objects can reside in more than one leaf at the same time. For simplicity we assume that $\mathcal{AB}(\nu)$ is of cubic shape with edge size w. Then the surface area $SA(\mathcal{AB}(\nu))$

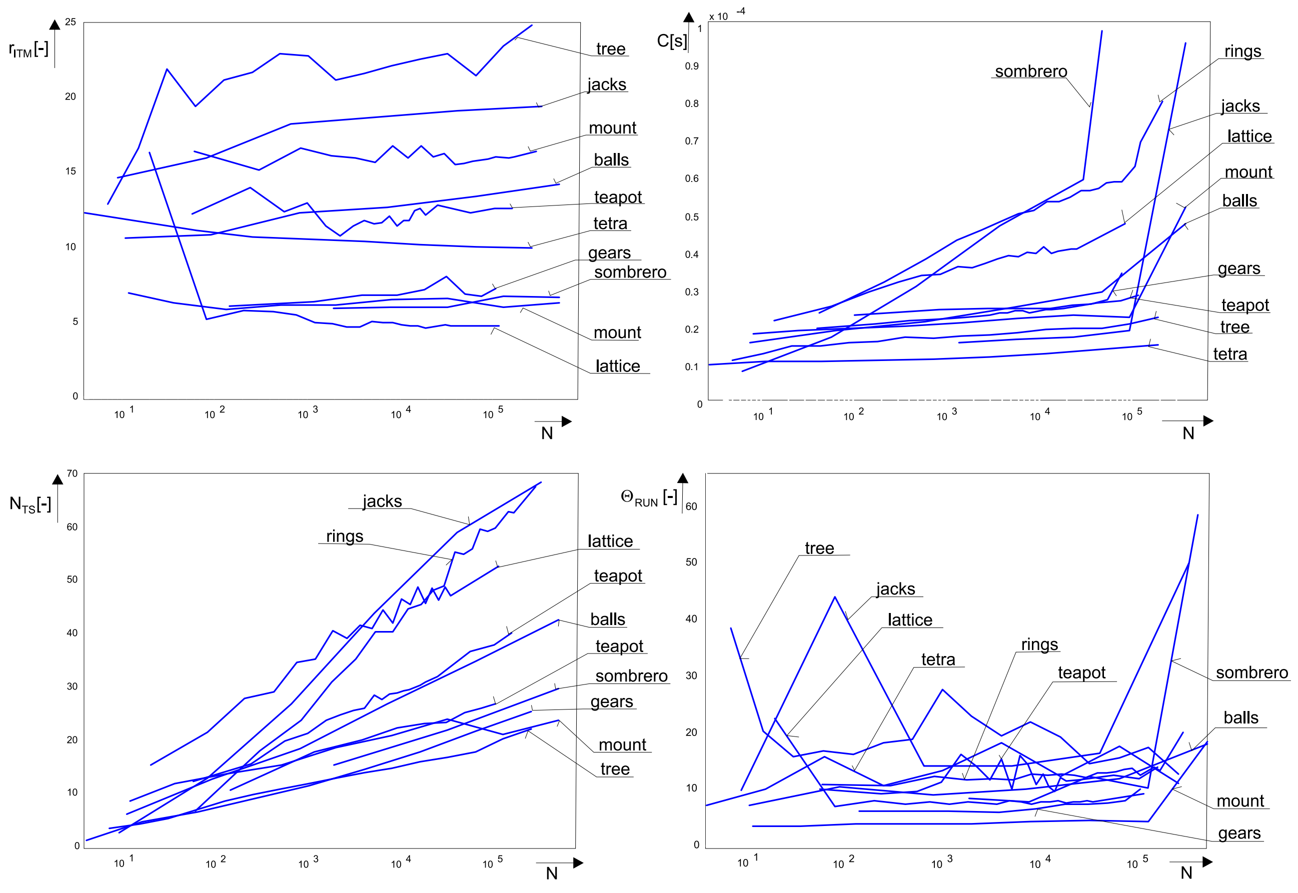


Figure 4.8: The cost in dependence on the number of objects for *SPD* scenes for TP_D for different size factors. The graphs $r_{ITM} = f_1(N)$, $\tilde{N}_{TS} = f_2(N)$, $T_R/N_{rays} = f_3(N)$, and $\Theta_{RUN} = f_4(N)$.

is $6.w^2$. The estimated cost $\hat{C}(\nu)$ of node ν is the sum of three estimated costs: the cost of traversing interior nodes $\hat{C}_{TI}(\nu)$, the cost of traversing leaves $\hat{C}_{TL}(\nu)$, and the cost of ray-object intersection tests $\hat{C}_{IT}(\nu)$.

The number of leaves in the BSP tree at the average depth $\tilde{d}$ is $N_l \approx 2^{\tilde{d}}$, and the number of interior nodes is $N_i \approx 2^{\tilde{d}} - 1$. We can compute the estimated cost $\hat{C}_{TI}(\nu)$ considering the cost of the nodes separately at different depths of the *kd*-tree. A node ν at one particular depth d in the *kd*-tree has the same shape and thus its surface area $\mathcal{AB}(\nu)$ is constant, regardless the position of ν in the *kd*-tree because we assume a spatial median method. The shape of $\mathcal{AB}$ associated with a node at depth d for which holds $d = 3.i$, $i \in \{0,1,2,\ldots\}$ is cubic again. Table 4.1 shows that the surface area of $\mathcal{AB}$ associated with the one node and all the nodes as a function of the depth in the *kd*-tree.

Then using the sum for a geometric sequence we can derive $\hat{C}_{TI}(\nu)$:

$$\hat{C}_{TI}(\nu) = \frac{1}{SA}.\hat{C}_{TS}.\sum_{i=1}^{N_i} SA(i) = \frac{1}{SA}.\hat{C}_{TS}.\sum_{i=0}^{\tilde{d}/3} 8^i.[SA*(36/6)^i] = \hat{C}_{TI}.\sum_{i=0}^{(\log N+k_1)/3} 48^i \equiv \hat{C}_{TS}.O(N+k_1) \tag{4.29}$$

Similarly, we can derive that $\hat{C}_{TL}(\nu) = \frac{1}{SA}.\hat{C}_{TS}.\sum_{i=1}^{N_l} SA(l) \equiv \hat{C}_{TS}.O(N+k_1)$. Since in a leaf the average number of objects is constant, provided that the distribution of objects in the scene space is uniform and no object is intersected by a ray, it holds that $\hat{C}_{IT}(\nu) = \frac{1}{SA}.\hat{C}_{IT}.\sum_{i=1}^{N_i} SA(i).N(l) \equiv \hat{C}_{IT}.O(N+k_1)$. To conclude, the linear estimate of cost $\hat{C}(\nu) \equiv O(N+k_1)$ is thus valid and k_1 is unknown under the conditions stated above. However, the multiplicative factor hidden behind O-notation is unknown.

d	N_i	$SA(\mathcal{AB}(\nu))$	$N_i.SA(\mathcal{AB}(\nu))$	$\sum_{j=0}^{d} N_i.SA(\mathcal{AB}(\nu))$
0	1	$6.w^2$	$6.w^2$	$6.w^2$
1	2	$4.w^2$	$8.w^2$	$14.w^2$
2	4	$2.5.w^2$	$10.w^2$	$24.w^2$
3	8	$1.5.w^2$	$12.w^2$	$36.w^2$
4	16	$1.w^2$	$16.w^2$	$52.w^2$

Table 4.1: The surface area of a node according to the depth d, assuming the input of cubic shape (size of side w) and spatial median method.

4.6.2 Reducing Objects' Axis-Aligned Bounding Boxes

The notion of $\mathcal{AB}s$ for scene objects significantly simplifies the *kd*-tree construction. Since objects can straddle a splitting plane, it can occur after several subdivision steps that some object's surface need not intersect the current node $\mathcal{AB}$ of the *kd*-tree hierarchy, although the $\mathcal{AB}s$ of some objects intersect the $\mathcal{AB}$ associated with the current node of the *kd*-tree. This is depicted in Fig. 4.9 (a).

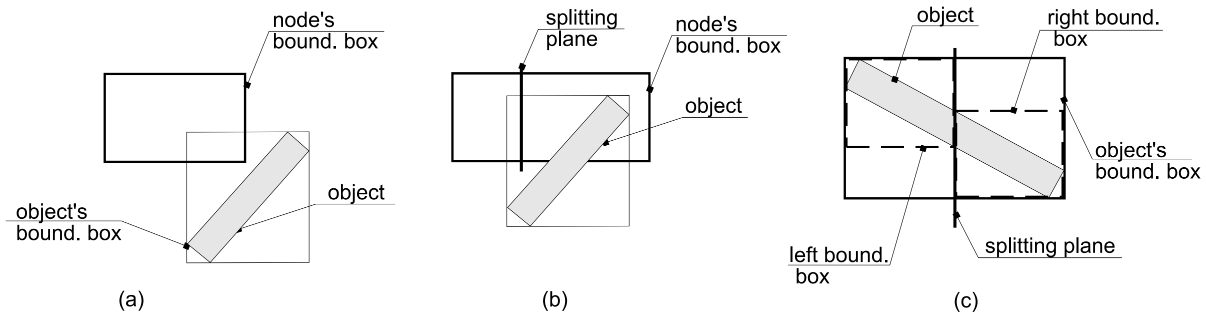


Figure 4.9: (a) Splitting during the *kd*-tree construction can result in references to objects that have no intersection with the leaves. (b) When the $\mathcal{AB}$ associated with an object straddles the slitting plane, it does not guarantee that the object also straddles the splitting plane. (c) Split clipping – reducing the $\mathcal{AB}s$ of the object, one on the left and one on the right of the splitting plane by clipping.

Although $\mathcal{AB}s$ of objects fit well for *kd*-tree construction, there are several disadvantages arising from the simple use of $\mathcal{AB}s$ instead of objects' surfaces. First, objects that cannot have an intersection with a ray are also tested in some leaves. Second, objects only virtually present in the interior node ν influence the estimated cost of ν and thus the whole process of *kd*-tree construction. We have found three possible ways to deal with this problem based on more complex intersection routines between the surface of an object and $\mathcal{AB}$ associated with the currently processed node of a *kd*-tree.

First, we can postprocess all leaves of the *kd*-tree and check using an intersection test between an $\mathcal{AB}$ associated with a leaf and an object's surface. For each leaf of the *kd*-tree we can thus remove the redundant references to objects. We call this method *leaf pruning.* In this case we do not avoid the problem of influencing the *kd*-tree construction by objects that only virtually intersect the $\mathcal{AB}s$ of the interior nodes. In the postprocessed *kd*-tree we can also get subtrees with empty leaves only. These branches of the *kd*-tree would be consolidated to empty leaves using another postprocessing step.

Second, we can apply the intersection test between the $\mathcal{AB}$ of the *kd*-tree node and surface of an object not only to leaves but also to interior nodes. In each interior node ν, excluding the root node, before evaluating the cost function we check whether all the objects belong to $\mathcal{AB}(\nu)$. We do this intersection also for leaves, which corresponds to leaf pruning. In this way we can guarantee that a minimum set of objects is considered for the currently processed node when evaluating the cost function. However, this method as presented also suffers from another problem. An object is considered to straddle the splitting plane, when the $\mathcal{AB}$ associated with the objects straddle the splitting plane. It need not hold as shown in Fig. 4.9 (b). This method can be further improved by testing the objects' assignment to left and right subtree of the objects straddling the splitting plane for each splitting plane position tested. This way is apparently costly and redundant since the intersection tests lack any computation coherency for adjacent positions of the tested splitting plane.

Third, we can solve the problem in one step advance. If the position of a splitting plane is known, we also know the objects straddling the splitting plane. For these objects we can reduce the $\mathcal{AB}s$ on both sides of the splitting plane whenever possible, as depicted in Fig. 4.9 (c). This requires a special intersection method for all shapes of objects, since, for a given object O, its current $\mathcal{AB}$, and the position and orientation of a splitting plane, we want to construct two tight $\mathcal{AB}s$ associated with the object's spatial region taken on the left and on the right of the splitting plane. These reduced $\mathcal{AB}s$ for the left and right side of the splitting plane must be passed with the references to the objects for the left and right child nodes. We call this method that clips the $\mathcal{AB}s$ of objects with regard to a splitting plane *split clipping.* Principally, split clipping works best of all the algorithms described here.

The description of the algorithms for leaf pruning and split clipping, which handles the intersection tests between an object and a plane, for particular shapes of objects is beyond the scope of this thesis. However, we have designed and implemented these algorithms [80], the results from the experiments are surveyed in Subsection 4.10.4.

4.7 General Cost Model

In the development of the surface area heuristic we assumed that a ray does not intersect any objects. Although such an assumption is rather unrealistic, the developed OSAH significantly improves the performance over the spatial median method for sparsely occupied scenes. In this section we extend the cost model to include the possibility that a ray traversal algorithm terminates in its child node because the ray intersects an object. Thus instead of using the upper bound of an estimate in the worst case, we use the average case estimate based on more realistic assumptions.

We call the proposed extended cost model the *general cost model* (abbreviated to GCM further in the text). Instead of formulating the total cost of the whole *kd*-tree we deal directly with the positioning of a splitting plane inside the interior node, *i.e.*, with the cost of a subdivided node (Eq. 4.14). During the recursive ray traversal algorithm (Chapter 5) there are exactly four ways, in which an arbitrary ray can traverse an interior node of a *kd*-tree; the left child only (subscript *LO*), the right child only (subscript *RO*), the left child first and the right child afterwards (subscript *LR*), and the right child first and the left child afterwards (subscript *RL*). Then we can express the new estimated cost as the sum of the four estimated costs for the distinct traversal cases: $\hat{C}_{LO}$, $\hat{C}_{LR}$, $\hat{C}_{RL}$, and $\hat{C}_{RO}$. Further, we assume that a ray can intersect an object in one of its child nodes with a probability p^T. For the sake of convenience, a ray that in $\mathcal{AB}(\nu)$, associated with a node ν, intersects an object will be called the *hit ray* with respect to ν. Similarly, we call a ray that does not intersect any object in $\mathcal{AB}(\nu)$ the *miss ray* with respect to ν. Let us denote the case for hit rays by superscript ^{T} and the case for miss rays by superscript ^{N}. If we have a set of rays intersecting $\mathcal{AB}(\nu)$, then p^T for the node $\mathcal{AB}(\nu)$ can be estimated as the number of hit rays divided by the number of all rays shot inside $\mathcal{AB}(\nu)$.

Then we can estimate the total cost $\hat{C}^{GCM}_{new}$ of an interior node ν to be subdivided as follows:

$$\begin{aligned}
\hat{C}^{GCM}_{new}(\nu) &= \hat{C}_{LO} + \hat{C}_{LR} + \hat{C}_{RO} + \hat{C}_{RL} && (4.30)\\
\hat{C}_{LO} &= p_{LO}.(p^T_L.\hat{C}^T_L + (1-p^T_L).\hat{C}^N_L) && (4.31)\\
\hat{C}_{LR} &= p_{LR}.(p^T_L.\hat{C}^T_L + (1-p^T_L).(\hat{C}^N_L + p^T_R.\hat{C}^T_R + (1-p^T_R).\hat{C}^N_R)) && (4.32)\\
\hat{C}_{RO} &= p_{RO}.(p^T_R.\hat{C}^T_R + (1-p^T_R).\hat{C}^N_R) && (4.33)\\
\hat{C}_{RL} &= p_{RL}.(p^T_R.\hat{C}^T_R + (1-p^T_R).(\hat{C}^N_R + p^T_L.\hat{C}^T_L + (1-p^T_L).\hat{C}^N_L)), && (4.34)
\end{aligned}$$

where	p_{LO}	–	(p_{LR}, p_{RO}, and p_{RL}) is described in Section 4.3.1,
	p_L^T	–	probability of a ray hitting an object in the left child node,
	p_R^T	–	probability of a ray hitting an object in the right child node,
	$\hat{C}_L^T$	–	estimated cost of subtree in left child only for hit rays,
	$\hat{C}_L^N$	–	estimated cost of subtree in left child only for miss rays,
	$\hat{C}_R^T$	–	estimated cost of subtree in right child only for hit rays,
	$\hat{C}_R^N$	–	estimated cost of subtree in right child only for miss rays.

The general cost model can be even more detailed, when we distinguish the estimated costs between hit and miss rays for all the traversal cases *LO*, *RO*, *LR*, and *RL*. Conversely, if we do not distinguish between the estimated cost of hit and miss rays ($\hat{C}_R^N = \hat{C}_R^T, \hat{C}_L^N = \hat{C}_L^T$), the cost model simplifies to the formulas:

$$\begin{aligned}
\hat{C}_{LO} &= p_L.\hat{C}_L && (4.35)\\
\hat{C}_{LR} &= p_{LR}.[p_L^T.\hat{C}_L + (1-p_L^T).(\hat{C}_L + \hat{C}_R)] = p_{LR}.[\hat{C}_L + (1-p_L^T).\hat{C}_R] && (4.36)\\
\hat{C}_{RO} &= p_R.\hat{C}_R && (4.37)\\
\hat{C}_{RL} &= p_{RL}.[p_R^T.\hat{C}_R + (1-p_R^T).(\hat{C}_L + \hat{C}_R)] = p_{RL}.[\hat{C}_R + (1-p_R^T).\hat{C}_L], && (4.38)
\end{aligned}$$

where $\hat{C}_L$ is the estimated cost of the left node and $\hat{C}_R$ is the estimated cost of the right node.

Since the general cost model more realistically describe the use of a *kd*-tree in an *RSA* (for average case and assuming recursive ray traversal algorithm), it could give us a more efficient *RSA* based on the *kd*-tree. Obviously, we shall pay for this by increased computational cost during the *kd*-tree construction.

The formulation of the general cost model brings us at least two subproblems. These are the estimation of the cost of a node ν distinctly for hit and miss rays, and the estimation of the probability p^T that a ray intersects an object inside the node. We discuss these issues below.

4.7.1 Estimating Blocking Factor

The first subproblem arising with the general cost model is to estimate the probability that a ray intersects an object in the node ν, *i.e.*, inside $\mathcal{AB}(\nu)$. This probability is required to be evaluated for the left and right child of ν in Eqs. 4.31–4.34 for all tested positions of the splitting plane when evaluating the cost function. The problem was discussed by Reinhard *et al.* [122] for the leaves of spatial hierarchies. The probability p^T that a ray hits an object according to that paper is referred to as the *blocking factor*. The input data given a node ν is the $\mathcal{AB}(\nu)$ associated with ν and the set of N objects pointed to in ν. The objects need not lie in $\mathcal{AB}(\nu)$ completely, but only partially. The opposite case is also possible; the tight $\mathcal{AB}$ of all objects can be smaller than $\mathcal{AB}(\nu)$ since the objects were separated by splitting planes in previous subdivision step(s). In the latter case the probability that a ray intersects an object is lower.

First, we recall the method used by Reinhard *et al.* [122] for the leaves of the spatial hierarchy. They take the $\mathcal{AB}s$ of objects and compute the blocking factor from the projected surface areas of objects $\mathcal{AB}$ inside the leaf $\mathcal{AB}$ for all three axes. The geometry of the problem is depicted in Fig. 4.10.

The blocking factor p^T is then computed as:

$$p^T = \sum_{j=1}^{N} \frac{SA^x(\mathcal{AB}(O_j)) + SA^y(\mathcal{AB}(O_j)) + SA^z(\mathcal{AB}(O_j))}{SA^x(\mathcal{AB}(\nu^E)) + SA^y(\mathcal{AB}(\nu^E)) + SA^z(\mathcal{AB}(\nu^E))}, \tag{4.39}$$

where $SA^a(\mathcal{AB}(O_j))$ is the projection area of the j-th object $\mathcal{AB}$ to the plane perpendicular to the a-axis, $a \in \{x, y, z\}$. Similarly, $SA^a(\mathcal{AB}(\nu^E))$ is the surface area of the leaf-cell in projection to the plane perpendicular to the a-axis. In order to consider the overlapping of objects in the projection they

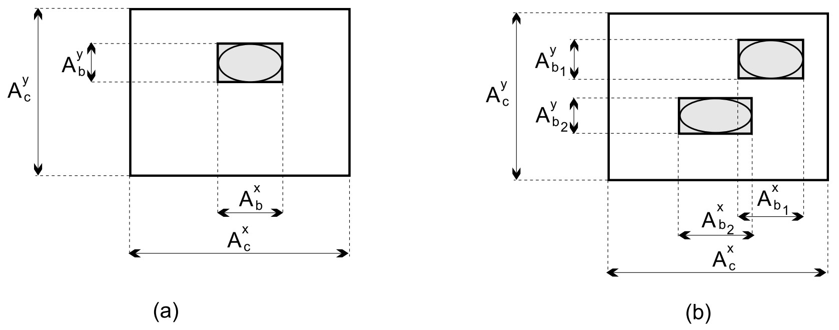


Figure 4.10: A cell in projection to the z-axis. (a) One object in a cell. (b) Two objects in a cell that overlap in projection to the plane perpendicular to the y-axis.

compute the projected surface area of the intersection for each pair of objects and they add or subtract it from the total surface area covered by the objects. The method is thus limited to the small number of objects in the leaf, since computation of overlapping N objects suffers from combinatorial explosion. We can remark that the $\mathcal{AB}$ associated with an object serves only as an approximation of the object, and the tightness of object to its $\mathcal{AB}$ would also be involved in the computation of the blocking factor.

To estimate p^T in general for many objects in the node remains a problem. One simple way is to sample the space of the node with N_{rays} rays and record the terminating position of the rays and the entry and exit points for given $\mathcal{AB}$. The number of sample rays N_{rays} must be high enough, assuming the rays are uniformly distributed. If we sort successful ray intersection points along the axis to be subdivided, we can estimate for each position of the splitting plane the blocking factor p^T for the left and right child. This algorithm for blocking factor can be costly since many ray-object intersection test can be computed. Efficient sampling assumes another *kd*-tree is constructed in advance and used for the sampling rays. In order to achieve some precision for the blocking factor estimate, the number of sampling rays should be sufficiently high; we assume at least of the order $N_{rays} = 10^2$. If the constructed *kd*-tree has N_{IN} interior nodes, the total cost of the estimate algorithm could be unacceptably high. We implemented this algorithm, and the results of applying it to the general cost model are presented in Subsection 4.10.5.

4.7.2 Cost Estimate for Hit and Miss Rays

The second subproblem induced by the general cost model is the cost estimate for hit and miss rays. Let us present arguments for the value of the ratio between costs of hit rays and miss rays. First, the cost of hit rays could be higher than for miss rays, since in the former case it must include at least one ray-object intersection test. Second, the cost of hit rays could be lower than for miss rays, since in the latter case a ray traversal algorithm has to perform many traversal steps and possibly several unsuccessful ray-object intersection tests. The question is whether it is possible to predict simply the ratio between the cost of hit and miss rays. We performed a set of experiments on *SPD* scenes (G^3_{SPD}, G^4_{SPD}, and G^5_{SPD}). The results are surveyed in Table 4.2, and we report the subset Δ of minimum testing output. The superscript *miss* denotes the case for miss rays, the superscript *hit* denotes the case for hit rays.

The results in Table 4.2 show that for the tested scenes the average number of ray-object intersection tests per ray $\tilde{N}_{IT}$ is on average lower for miss rays than for hit rays. The same holds for the average number of traversal steps per ray $\tilde{N}_{TS}$. Unfortunately, the value of the ratio between the cost for miss rays and hit rays is not constant and thus cannot be determined in advance. For example, for the scene “lattice” the cost for hit rays is lower than for miss rays, since the miss rays must traverse more nodes to get out of the scene $\mathcal{AB}$. In general, the ratio between the cost of hit rays and miss rays is difficult and/or almost impossible to predict.

Scene	Group	N	$\tilde{N}_{IT}^{hit}$	$\tilde{N}_{IT}^{miss}$	$\tilde{N}_{TS}^{hit}$	$\tilde{N}_{TS}^{miss}$	$\tilde{N}_{ETS}^{hit}$	$\tilde{N}_{ETS}^{miss}$	$\tilde{N}_{EETS}^{hit}$	$\tilde{N}_{EETS}^{miss}$
balls3	G^3_{SPD}	821	6.52	-26%	21.4	-24%	4.11	-10%	0.956	+0%
balls4	G^4_{SPD}	7382	7.37	-30%	30.9	-25%	5.61	-9%	1.54	+8%
balls5	G^5_{SPD}	66341	8.25	-35%	40.7	-27%	7.10	-14%	2.12	+2%
gears2	G^3_{SPD}	1169	3.71	-9%	19.8	-18%	3.58	-26%	0.733	-51%
gears4	G^4_{SPD}	9345	4.11	-34%	26.6	-30%	4.28	-40%	1.09	-68%
gears9	G^5_{SPD}	106435	4.44	-38%	32.5	-33%	4.67	-45%	1.09	-70%
jacks3	G^3_{SPD}	657	9.40	-55%	33.4	-28%	5.87	-13%	1.48	+85%
jacks4	G^4_{SPD}	5265	12.4	-55%	50.8	-25%	8.73	-25%	2.18	+89%
jacks5	G^5_{SPD}	42129	14.7	-60%	67.2	-27%	11.2	-18%	2.80	+92%
lattice6	G^3_{SPD}	1255	3.59	+13%	33.8	+19%	5.61	+34%	2.17	+65%
lattice12	G^4_{SPD}	8281	4.12	+4%	41.9	+5%	6.60	+14%	2.78	+26%
lattice29	G^5_{SPD}	105307	4.33	-10%	56.0	-13%	7.74	-8%	3.55	-6%
mount4	G^3_{SPD}	516	4.08	-48%	15.4	+2%	2.81	+12%	0.786	+87%
mount6	G^4_{SPD}	8196	4.36	-39%	21.6	-5%	3.66	+5%	1.36	+52%
mount8	G^5_{SPD}	131076	4.03	-43%	24.9	-30%	4.23	-20%	1.60	+4%
rings3	G^3_{SPD}	841	9.63	-29%	26.0	-16%	4.82	-4%	1.29	+57%
rings7	G^4_{SPD}	8401	11.2	-31%	43.9	-18%	7.55	-9%	2.76	+30%
rings17	G^5_{SPD}	107101	11.2	-33%	63.9	-15%	10.6	-8%	4.39	+26%
sombrero1	G^3_{SPD}	1922	3.04	-56%	24.7	-50%	4.80	-53%	2.92	-51%
sombrero2	G^4_{SPD}	7938	3.05	-56%	29.9	-52%	5.53	-53%	3.68	-50%
sombrero4	G^5_{SPD}	130050	3.43	-56%	41.4	-44%	7.18	-54%	5.00	-52%
teapot4	G^3_{SPD}	1008	5.20	-31%	23.3	-13%	4.71	-7%	2.00	+26%
teapot12	G^4_{SPD}	9264	4.57	-36%	33.8	-19%	6.19	-15%	3.23	+5%
teapot40	G^5_{SPD}	103680	5.03	-37%	45.6	-23%	7.82	-19%	4.61	-50%
tetra5	G^3_{SPD}	1024	5.53	-79%	23.4	-61%	4.50	-60%	3.08	-50%
tetra6	G^4_{SPD}	4096	5.68	-81%	31.3	-64%	5.75	-63%	4.28	-54%
tetra8	G^5_{SPD}	65536	5.91	-96%	51.1	-68%	8.95	-67%	7.39	-63%
tree8	G^3_{SPD}	1023	4.70	-31%	17.0	-35%	4.28	-30%	0.465	+11%
tree11	G^4_{SPD}	8191	4.86	-34%	20.7	-33%	4.82	-27%	0.842	+10%
tree15	G^5_{SPD}	131071	5.75	-60%	27.9	-31%	6.27	-29%	1.41	+6%
Average	–	–	–	-40.3%	–	-28.7%	–	-22%	–	4%

Table 4.2: The Δ subset of minimum testing output measured for miss and hit rays for testing procedure TP_D for G^3_{SPD}, G^4_{SPD}, and G^5_{SPD} scenes. Values for miss rays are taken relatively in % related to the values for hit rays.

Since the cost estimate for hit rays and miss rays is not easily predictable, we are forced to use one of the following ways:

- disregard the concept of distinguishing between the costs for hit and miss rays, and thus use a simpler form of GCM, *i.e.*, Eqs. 4.35–4.38. In the case we use for example a linear cost estimate, but the blocking factor is estimated via sampling.
- similarly to the estimation of p^T, we can evaluate the costs experimentally assuming a sampling *kd*-tree is built in advance. The sampling *kd*-tree is not the *kd*-tree we build up now, but it can be taken as a rough approximation of it with similar characteristics, since it was constructed with a similar algorithm for the same input data. Then the number of traversal steps and the number of ray-object intersection tests can be estimated well enough. The error of such an estimate remains unknown.

A simple way of estimating the cost for hit and miss rays given a node with the objects remains a difficult problem. We thus implemented the sampling strategy for estimating the costs. The results of applying these estimates to the general cost model are presented in Subsection 4.10.5.

4.8 Preferred Ray Sets

The basic cost model (Eq. 4.9) and thus the surface area heuristic algorithms developed until now have been based on the assumption that rays are distributed uniformly in ray space. This enables us to compute the probability that an arbitrary ray hits the $\mathcal{AB}(\nu)$ associated with the node ν using Eq. 4.4. If an *RSA* is applied for example in a global illumination rendering algorithm, the algorithm can also imply for a given scene and viewpoint some particular ray distribution. The question is if it is possible to use this knowledge about the expected ray distribution to build up a *kd*-tree so that an *RSA* based on the *kd*-tree and this ray distribution has improved performance. Further, we break the assumption of uniformity of ray distribution and change the above presented equations to consider only a certain ray set with a particular ray distribution.

We focus on three types of ray sets that correspond to parallel, perspective, and spherical projections. For these ray distributions we show how Eq. 4.4 is changed for a particular ray distribution. The idea of the *kd*-tree construction for preferred ray sets was introduced by Havran and Bittner [76, 77].

4.8.1 Parallel Projection

Probably the most intuitive set of rays is formed by fixing their direction. Rays with the same direction are involved in the parallel projection of the scene [157].

In this case the rays are perpendicular to the *projection plane* Π_P. Let us suppose that the ray distribution on Π_P is uniform. Additionally, we can restrict the preferred set of rays to a certain *viewport*. The probability that a ray hits the $\mathcal{AB}(\nu)$ associated with a node ν of the *kd*-tree can then be expressed using a surface area of the projection of the $\mathcal{AB}(\nu)$ to Π_P clipped to the viewport. The corresponding geometry is depicted in Fig. 4.11.

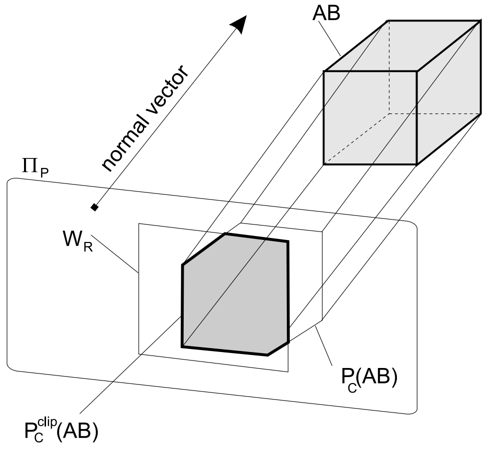


Figure 4.11: Parallel projection of an $\mathcal{AB}$ to the plane Π_P.

Let viewport W_R be a rectangular window on the projection plane Π_P. Let $S\mathrm{R}^{PAR}_{\Pi_P}(W_R)$ be a set of rays perpendicular to Π_P and intersecting W_R. The silhouette edges of an $\mathcal{AB}(\nu)$ projected onto Π_P form a convex polygon $P_C(\mathcal{AB}(\nu))$. Optionally, we can define $P_C(\mathcal{AB}(\nu))$ as a convex hull of all the vertices of $\mathcal{AB}(\nu)$ projected to Π_P. In order to determine the polygon $P_C^{clip}(\mathcal{AB}(\nu)) = W_R \cap P_C(\mathcal{AB}(\nu))$ lying on Π_P, a clipping algorithm must be applied [157]. Let $SA^{PAR}(X)$ be the surface area of the entity X on the plane Π_P and let $\mathcal{AB}(\mathcal{S})$ be the axis-aligned bounding box containing the whole scene $\mathcal{S}$. Similarly to Eq. 4.4 we can express the probability that a ray from $S\mathrm{R}^{PAR}_{\Pi_P}(W_R)$ hits the $\mathcal{AB}$ once it passes through $\mathcal{AB}(\mathcal{S})$ as follows:

$$p(\mathcal{AB}) = \frac{SA^{PAR}(P_C^{clip}(\mathcal{AB}(\nu)))}{SA^{PAR}(P_C(\mathcal{AB}(\mathcal{S})))} \tag{4.40}$$

Eq. 4.40 can be used to replace directly the probabilities in Eq. 4.9 for both left and right child nodes. We call the surface area heuristic modified for parallel projection a *parallel surface area heuristic* (abbreviated to PARSAH).

4.8.2 Perspective Projection

We form another preferred ray set by fixing the origin of the rays. Similarly as for parallel projection, we focus only on rays passing through a certain viewport. The origin of rays (viewpoint) O^{R} and the viewport define a *viewing frustum*, see Fig. 4.12. Assuming that the rays are uniformly distributed on the viewport, the corresponding ray set $S_{\mathrm{R}\Pi_P}^{PER}(W_R)$ contains rays involved in the commonly used perspective projection that intersects the viewport W_R.

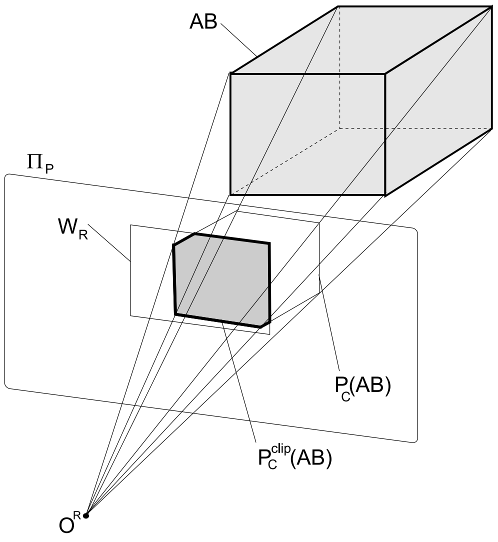


Figure 4.12: Perspective projection of $\mathcal{AB}$ to the plane Π_P.

Let $P_C(X)$ be the polygon obtained by projecting an axis-aligned bounding box X perspectively to the plane Π_P and $SA^{PER}(P_C(X))$ the surface area of the polygon $P_C(X)$ on the projection plane. In order to compute the projected surface area of $\mathcal{AB}(\nu)$, which is associated with the node ν, a clipping to the viewing frustum must be applied, $P_C^{clip}(\mathcal{AB}(\nu)) = W_R \cap P_C(\mathcal{AB}(\nu))$, similarly to the parallel projection. The conditional probability that a ray from $S_{\mathrm{R}\Pi_P}^{PER}(W_R)$ hits the $\mathcal{AB}$ once it passes through the axis-aligned bounding box of the whole scene $\mathcal{AB}(\mathcal{S})$ can be expressed as:

$$p(\mathcal{AB}) = \frac{SA^{PER}(P_C^{clip}(\mathcal{AB}(\nu)))}{SA^{PER}(P_C(\mathcal{AB}(\mathcal{S})))} \tag{4.41}$$

We call the surface area heuristic modified for perspective projection a *perspective surface area heuristic* (abbreviated to PERSAH).

The clipping of $\mathcal{AB}$ to the viewport must be always applied, hence the speed of the clipping is crucial for the build time T_B of the *kd*-tree. In the following text, we outline the algorithms of clipping and computing the surface area of the projected $\mathcal{AB}$.

Viewing Frustum Clipping

A general algorithm determining the surface area of the projection of an $\mathcal{AB}$ to projection plane Π_P can be as follows: Project all the vertices of the $\mathcal{AB}$ to Π_P and construct a convex hull of the projected vertices. This convex hull corresponds to a certain polygon. Then clip the polygon with respect to the viewport and compute its surface area.

Obviously, the projection of all eight vertices and construction of the convex hull of the $\mathcal{AB}$ for a specific viewpoint is both costly and computationally redundant. We need to construct only the projection of the *silhouette* of the $\mathcal{AB}$, since then the maximum number of projected points is six and the minimum is four. According to the mutual position of the viewpoint and the $\mathcal{AB}$ we can identify twenty-seven regions for which the projected $\mathcal{AB}$ has the same sequence of silhouette edges. These regions are induced by six planes that determine the $\mathcal{AB}$. Given a viewpoint O^{R} and an $\mathcal{AB}$ the corresponding sequence of silhouette edges can be determined by a table lookup and easily projected to Π_P.

The sequence of silhouette edges is clipped in object space by the Sutherland-Hodgman algorithm [157] using the four planes that bound the viewing frustum. The result of clipping is a sequence of connected edges with at most ten vertices. The vertices are projected on to the projection plane, forming a convex polygon. Finally, the surface area of the polygon is computed.

A special case occurs when the viewpoint lies inside the $\mathcal{AB}$. Obviously, every ray originating at such a viewpoint must always intersect the $\mathcal{AB}$ and hence the conditional probability used in Eq. 4.41 is equal to one. When this case occurs, it could degenerate the result of the cost function used in PERSAH and we must use another method to position the splitting plane, either spatial median method or OSAH.

4.8.3 Spherical Projection

Another set of rays is induced by those involved in *spherical projection.* As with perspective projection, the origin of the rays is fixed. The rays induced by spherical projection are not uniformly distributed on a projection plane, but on a sphere with its center at the origin of a ray. Let us denote the set of rays of spherical projection $S_{\mathrm{R}}{}^{\Pi_P}_{SPH}$, note that no viewport need be defined in this case, thus the clipping phase present in perspective projection can be omitted. A *kd*-tree built according to such a set of rays could be used to accelerate ray queries for point light sources [157].

Since the ray distribution in $S_{\mathrm{R}}{}^{\Pi_P}_{SPH}$ is uniform for the sphere, the task is to compute a solid angle Ω induced by the frustum enclosing a given $\mathcal{AB}$ with respect to the center of projection. This task is similar to the computation of the point-to-polygon form factor [62]. Nevertheless, in our case it involves determining a region bounded by Jordan's curve [47] on a unit sphere. The solid angle induced by a given $\mathcal{AB}$ can be computed as the sum of all solid angles for visible faces from a given viewpoint. The solid angle Ω of a rectangle with height h and width w positioned with one corner at the origin and perpendicular to the z-axis can be computed in the following way (see Fig. 4.13):

$$l = \sqrt{x^2+y^2+c^2}.\cos\beta = \frac{c}{l}.dA = dx.dy.d\Omega = \frac{dA.\cos\beta}{l^2}.d\Omega = \frac{c.dx.dy}{\sqrt{(x^2+y^2+c^2)^3}} \tag{4.42}$$

The solid angle is then computed by integration:

$$\Omega[sr] = \int_{x=0}^{w}\int_{y=0}^{h}\frac{c.dx.dy}{\sqrt{(x^2+y^2+c^2)^3}} \tag{4.43}$$

Solving this integral with the substitution $u = \frac{x}{c}$ and $v = \frac{y}{c}$ results in:

$$\Omega[sr] = \arctan\frac{w.h}{c.\sqrt{c^2+h^2+w^2}} \tag{4.44}$$

The computation of the solid angle for a rectangle with one corner not positioned in the origin is performed by more general solution of the equation above. Then we can compute the solid angle taken by all three faces of the $\mathcal{AB}$ that can be visible from a viewpoint. The conditional probability with respect to the $\mathcal{AB}(\mathcal{S})$ of the whole scene $\mathcal{S}$ is then:

$$p(\mathcal{AB}) = \frac{\Omega(\mathcal{AB})}{\Omega(\mathcal{AB}(\mathcal{S}))} \tag{4.45}$$

We call the surface area heuristic modified for spherical projection (Eq. 4.45) a *spherical surface area heuristic* (abbreviated to SPHSAH).

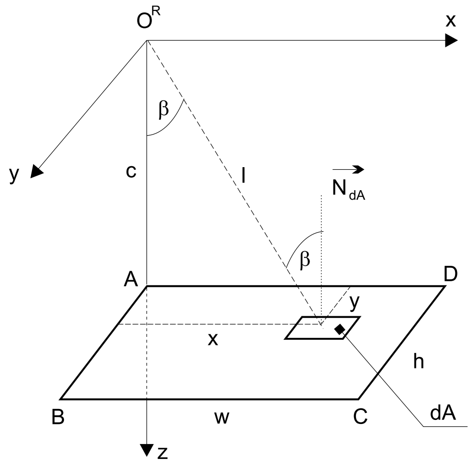


Figure 4.13: Computation of spherical projection for rectangle.

It is obvious that for a viewpoint located inside a given $\mathcal{AB}$ the corresponding solid angle is $\Omega = 4.\pi$, thus the probability $p(\mathcal{AB})$ is equal to one. In this case, similarly PERSAH, it is necessary to use another way for positioning the splitting plane.

4.8.4 Discussion

Below we discuss the properties of surface area heuristic for preferred ray sets.

First, we discuss parallel surface area heuristic for the normal $\vec{N}^{\Pi_P} = (x, y, z)$ of the projection plane Π_P. We can easily prove that the *kd*-tree constructed with PARSAH with a normal where $|x| = |y| = |z|$ corresponds to the *kd*-tree constructed with OSAH. (Eq. 4.14). There are exactly eight vectors on a unit sphere with these properties. Hence, the cost estimate of the *kd*-tree constructed for PARSAH with other vectors than these eight vectors can be potentially decreased, and thus an *RSA* based on the *kd*-tree assuming the required ray distribution with other vectors than these eight vectors would be more efficient.

One problem can arise with the clipping of the projected $\mathcal{AB}$, when the constructed *kd*-tree is used for rays outside the viewing frustum. Since the probability using Eq. 4.40 is zero, the constructed *kd*-tree can degenerate, and this can negatively influence the cost for a ray traversing this part of the *kd*-tree. We have found two solutions to this problem. The first detects the situation and then for these cases uses OSAH for any node of the *kd*-tree that is partially or fully outside the viewing frustum. Then even the nodes on the boundary of the viewing frustum will be constructed using OSAH. The second solution is simpler; we can approximate the perspective projection by spherical projection for all cases where the solid angle taken by the viewing frustum is small. For so called *paraxial rays* (maximum angle between any two rays up to 5^o) the approximation is sufficiently precise. Then the ray distribution for spherical and perspective projection is similar and no clipping is performed.

4.9 Time Complexity Analysis

Until now we have not analyzed the time complexity of *kd*-tree construction. We show below that for N objects in the scene the time complexity for construction algorithms above is $O(N.\log N)$, but with various multiplicative factors hidden behind O-notation. For analysis we assume that the constructed *kd*-tree has a number of leaves linear with N. This corresponds to the use of the automatic termination criteria algorithm developed in Section 4.5, where the computation of d_{max} is derived from N. Since each object is referenced at $O(N_{ER}/N)$ leaves on average, the number of leaves is at most $O(N_{ER})$. For ad hoc termination criteria the number of leaves is due to the constant value of d_{max} formally $O(1)$, which is not usable for any analysis at all.

For the *kd*-tree construction based on the spatial median method (see Subsection 4.2.2) given a node ν with N objects we put the splitting plane to a known position. If we do not sort objects according to their mutual position before the construction, we must test all the objects to find if they belong to the right or left node, thus in $O(N)$ time. The total number of objects associated with the interior nodes for one depth of the *kd*-tree also linear with N, and the depth of *kd*-tree is $O(\log N)$, thus resulting in $O(N.\log N)$ for the whole *kd*-tree. When we sort the objects before splitting the root node, sorting takes $O(N.\log N)$ time, which results again in $O(N.\log N)$ for *kd*-tree construction.

For the *kd*-tree construction based on the surface area heuristic algorithms we first sort the boundaries of $\mathcal{AB}s$ associated with the objects in the scene along all three axes. Sorting itself takes $O(N.\log N)$ time. In $\mathbb{E}^3$ for each subdivision step, we have three lists of sorted boundaries and we evaluate the cost function for each possible position of the splitting plane. When we select the splitting plane Π to be used, we subdivide the lists into two halves, duplicating the $\mathcal{AB}s$ of objects split by Π. Both testing and splitting of lists takes $O(N)$ time. The depth of the *kd*-tree constructed is also $O(\log N)$, which again results in $O(N.\log N)$ time complexity for *kd*-tree construction.

The running time consumed to perform particular operations of surface area heuristic algorithms within the *kd*-tree construction can vary greatly. It includes the computing of the estimated cost, estimating blocking factor and the costs of subtrees, projecting and clipping within the general cost model, *etc.* The time needed to construct the *kd*-tree is expected to be saved within the execution phase of an *RSA* based on the *kd*-tree, which is justified by the number of ray shooting queries. We discuss these experimental results in the next section.

4.10 Summary of Results and Discussion

In this section we survey the experimental results concerning *kd*-tree construction. The experiments were performed on the above-mentioned G^3_{SPD}, G^4_{SPD}, and G^5_{SPD} test scenes from *SPD* package, see Section 3.5 for more characteristics of the scenes. Preferably, we report the experimental results in the form of minimum testing outputs (Section 2.5) – for each experiment a set of 13 parameters. To decrease the space taken by tables we decided to use only testing procedure TP_D whenever applicable. The results of the first phase of the BES project (see Chapter 3) indicate that the results of using TP_A, TP_B, and TP_C are sufficiently similar. The Tables in Appendix E show the minimum testing output for each scene and tested *RSAs* based on the *kd*-tree with a particular construction algorithm. The results in this section just summarize the results for a particular construction algorithm[3].

Some of the presented methods for *kd*-tree construction can be combined together, and this is also sometimes necessary. Obviously, late cutting off empty space and two-plane cutting off empty space must be combined with ordinary surface area heuristic (OSAH). In *kd*-tree construction with the use of a general cost model it would be possible to use the ray distribution for some preferred ray set, but we show the results only for the testing procedure TP_D (see Subsection 3.4.1). The termination criteria are just another part of the *kd*-tree construction algorithm, split clipping of objects' $\mathcal{AB}s$ has to be applied

[3]The line 0 in the Tables in Appendix Eshows the result for a "naïve *RSA*".

on the fly, and leaf pruning in postprocessing. We thus do not report all the possible combinations of the methods described in this chapter, but only the reference algorithm and a new method that allows us easy comparison. If not stated otherwise, we compare Θ_{RUN} of the presented *RSA* with reference one to compare the time devoted to ray shooting only. For the all testing presented in this chapter we used the recursive ray traversal algorithm TA^{B}_{rec} (described in Section 5.4).

4.10.1 Positioning of the Splitting Plane

Let us summarize the results for positioning of the splitting plane, as stated in Section 4.2.2. The results are fully given in Appendix E, let us describe the settings for the experiments. The setting is always described for a line X, which corresponds to denotation in Appendix E:

Line 1: *kd*-tree constructed with a spatial median using ad hoc termination criteria ($d_{max} = 16$, $N_{max} = 2$) and change of the splitting plane orientation in cyclic order (x,y,z,x...).

Line 2: *kd*-tree is constructed with an object median using ad hoc termination criteria ($d_{max} = 16$, $N_{max} = 2$) and change of the splitting plane orientation in cyclic order (x,y,z,x...).

Line 3: *kd*-tree constructed with an object median using ad hoc termination criteria ($d_{max} = 16$, $N_{max} = 2$). The orientation of the splitting plane is taken so as to split the smallest possible number of objects.

Line 4: *kd*-tree constructed with OSAH using ad hoc termination criteria ($d_{max} = 16$, $N_{max} = 2$).

Line 5: *kd*-tree constructed with OSAH using ad hoc termination criteria ($d_{max} = 16$, $N_{max} = 2$). The search of the minimum cost splitting plane is restricted to the median interval.

Line 6: *kd*-tree constructed with OSAH using ad hoc termination criteria ($d_{max} = 16$, $N_{max} = 2$) and change of the splitting plane orientation in cyclic order (x,y,z,x...).

For the following discussion on *kd*-tree construction techniques, we select as a reference the constru-ction algorithm given by line 4 that has achieved the best results on average. The results are summarized in Table 4.3.

Line	Avg	CountBetter	Best	BestName	Worst	WorstName
4	reference algorithm					
1	+1804%	0	+25%	jacks5	+36604%	tree15
2	+354%	2	-7%	lattice29	+1755%	teapot40
3	+577%	1	-3%	lattice29	+2764%	teapot40
5	-3%	24	-12%	tetra5	+5%	gears9
6	+7%	11	-8%	mount4	+88%	sombrero4

Table 4.3: Summary table for positioning of the splitting plane.

The spatial median (line 1) always achieved worse performance than the reference algorithm. On average for 30 *SPD* scenes was 1804% slower than the reference (line 4), in the best case 25% slower ("jacks5"), and in the worst case 36604% slower ("tree15"). The lack of performance is particularly remarkable for large scenes and for sparsely occupied scenes ("treeX"). For highly occupied scenes ("gearsX") the decrease in performance is only small. The maximum leaf depth set to 16 does not allow the BSP tree to adapt well for highly occupied spatial regions.

The object median with change of the splitting plane orientation in cyclic order (line 2) was on average of higher performance than the spatial median. However, it was on average 354% slower than the reference, in the best case 7% faster ("lattice29"), and in the worst case 1755% slower ("teapot40"). With increasing number of objects the method reaches worse performance.

The object median with arbitrary change of the splitting plane orientation (line 3) shows similar characteristics as above (line 2), but has even worse performance. It was on average 577% slower than the reference, in the best case 3% faster ("lattice29") and in the worst case 2764% slower ("teapot40"). Moreover, the time required for construction is also higher than the time for line 2, since it requires that we evaluate the possible plane position in the object median for all three axes.

OSAH restricted to median interval (line 5) achieved practically the same results as the reference solution (line 4). The constructed BSP trees have practically the same number of leaves, and the timings differ only slightly. On average, it was 3% faster than the reference, in the best case 12% faster ("tetra5"), and in the worst case 5% slower ("gears9").

OSAH with change of the splitting plane orientation in cyclic order (line 6) was on average 7% slower than the reference, in the best case it was 8% faster ("mount4" and "teapot4"), and in the worst case 88% slower ("sombrero4"). The interesting property of this construction algorithm is the reduction of the build time, which is particularly noticeable for G^5_{SPD} scenes with high number of objects.

4.10.2 Termination Criteria

Here we summarize the results for the termination criteria. We tested the termination criteria for this setting:

Line 7: *kd*-tree constructed with OSAH using ad hoc termination criteria ($d_{max} = 8$, $N_{max} = 1$).

Line 8: *kd*-tree constructed with OSAH using ad hoc termination criteria ($d_{max} = 8$, $N_{max} = 2$).

Line 9: *kd*-tree constructed with OSAH using ad hoc termination criteria ($d_{max} = 16$, $N_{max} = 1$).

Line 10: *kd*-tree constructed with OSAH using ad hoc termination criteria ($d_{max} = 16$, $N_{max} = 2$).

Line 11: *kd*-tree constructed with OSAH using ad hoc termination criteria ($d_{max} = 24$, $N_{max} = 1$).

Line 12: *kd*-tree constructed with OSAH using ad hoc termination criteria ($d_{max} = 24$, $N_{max} = 2$).

Line 13: *kd*-tree constructed with OSAH using automatic termination criteria ($k_1 = 1.2$, $k_2 = 2.0$, $K^1_{fail} = 1.0$, $K^2_{fail} = 0.2$, and $r^{min}_q = 0.75$).

As a reference algorithm for the following discussing we take the algorithm given by line 13, the automatic termination criteria, since in practice it achieved the best performance on average. The results are summarized in Table 4.4.

Line	Avg	CountBetter	Best	BestName	Worst	WorstName
13	reference algorithm					
7	+2977%	0	+19%	mount4	+30321%	tree15
8	+2979%	0	+18%	mount4	+30259%	tree15
9	+19%	6	-12%	rings3	+225%	tree15
10	+22%	3	-12%	rings3	+228%	tree15
11	+7%	6	-12%	tree8	+44%	gears9
12	+5%	7	-10%	tree8	+20%	jacks4

Table 4.4: Summary table for the termination criteria.

The results for the *kd*-tree specified by line 7 and 8 are practically similar, and the setting $N_{max} = 1$ produces a higher number of leaves for the *kd*-tree specified by line 7. Comparing the results of these ad hoc termination criteria with reference (automatic termination criteria), setting specified by line 7 and 8 was on average 2977% slower, in the best case 19% slower ("mount4"), and at the worst case 30321% slower ("tree15"). Setting d_{max} to 8 is thus insufficient even for a small number of objects in the scene (G^3_{SPD}), particularly, it decreases the performance for G^4_{SPD} and G^5_{SPD} scenes.

The results for the *kd*-tree specified by lines 9 and 10 are also quite comparable, and the setting $N_{max} = 1$ again results in a higher number of leaves. Comparing with the reference, line 9 is on average 19% slower than the reference, in the best case 12% faster ("rings3" and "tree8", faster than reference for 6 scenes), and at the worst case 225% slower ("tree15"). The *kd*-tree specified by line 10 achieved slightly worse results, it was on average 22% slower, in the best case 10% faster ("rings3", faster than the reference for 3 scenes) and in the worst case 228% slower ("tree15"). For large scenes, the setting of the maximum leaf depth to $N_{max} = 16$ was insufficient, particularly for sparsely occupied scenes.

The results for *kd*-tree specified by line 11 and 12 show that setting $N_{max} = 24$ achieves the best results of all ad hoc termination criteria tested. The *kd*-tree specified by line 11 is on average 7% slower than the reference, in the best case 12% faster ("tree8", faster than the reference for 6 scenes), and in the worst case 44% slower ("gears9"). This setting of ad hoc termination criteria tries to continue the subdivision even if this need not improve the total performance, as we can see for the scene "gears9". The *kd*-tree specified by line 12 is on average 5% slower than the reference, in the best case 10% faster ("tree8", faster than reference for 7 scenes), and in the worst case 20% slower ("jacks4"). Unlike for $d_{max} = 8, 16$, for $d_{max} = 24$ the setting N_{max} to 2 (line 12) instead of setting N_{max} to 1 (line 11) improves the performance on average.

We can also compare automatic and ad hoc termination criteria generally. The ad hoc termination criteria result in *kd*-trees with higher performance than automatic termination criteria for 22 out of 180 experiments. However, in the best case the improvement is only 12% and in the worst case ad hoc termination criteria are 30321% slower. If we require to use ad hoc termination criteria, we can on average recommend the use of the best results achieved for $N_{max} = 2$ and $d_{max} = 24$, however, we should be aware of the increased average build time T_B and memory consumed by the *kd*-tree compared with the automatic termination criteria.

4.10.3 Cutting Off Empty Space

Here we summarize the results for cutting off empty space methods. We used the following setting for the experiments:

Line 14: *kd*-tree constructed with OSAH and automatic termination criteria, but slightly modified according to the paper by Subramanian and Fussel [145]. The probability of intersecting the left and right child nodes is taken two tights $\mathcal{AB}s$ enclosing objects on the left and right side of the splitting plane. We have implemented this other variant of surface area heuristic only to verify the results presented by Subramanian and Fussel.

Line 15: *kd*-tree constructed with OSAH using automatic termination criteria. In addition, late cutting off empty space is applied, with the setting: $\hat{C}_{IT} = 0.7$ and $\hat{C}_{TS} = 0.3$. The limit for the maximum number of planes to be put within late cutting off empty space was set to 3.

Line 16: *kd*-tree constructed with OSAH using automatic termination criteria. In addition, two-plane cutting off empty space is applied, with the setting: $\hat{C}_{IT} = 0.7$ and $\hat{C}_{TS} = 0.3$.

Line 17: *kd*-tree constructed with OSAH using automatic termination criteria. In addition, both late and two-planes cutting off empty space is applied, with the setting: $\hat{C}_{IT} = 0.7$ and $\hat{C}_{TS} = 0.3$. The limit for the maximum number of planes to be put within late cutting off empty space was set to 3.

As a reference algorithm for the following discussion we take the *RSA* specified by line 13 – it is the standard algorithm for the *kd*-tree construction without the additional improvements. The results are summarized in Table 4.5.

Line	Avg	CountBetter	Best	BestName	Worst	WorstName
13	reference algorithm					
14	+9%	4	-4%	tree8	+26%	teapot4
15	+2%	5	-2%	tre11	+13%	rings3
16	+0%	9	-14%	gears4	+13%	rings3
17	-4%	22	-18%	sombrero1	+25%	rings3

Table 4.5: Summary table for cutting off empty space for automatic termination criteria.

The *kd*-tree as described by line 14 is on average 9% slower than the reference, in the best case 4% faster ("tree8", faster than reference for 4 out of 30 scenes), in the worst case 26% slower ("teapot4"). In addition, the build time is increased on average by 20%. The use of this surface area heuristic is questionable.

The *kd*-tree as described by line 15 is for given setting of $\hat{C}_{IT}$ and $\hat{C}_{TS}$ on average slower than the reference by 2%. For 5 out of 30 scenes it performs faster, in the best case is 2% faster than the reference ("tree11"), in the worst case is 13% slower ("rings3").

The *kd*-tree as described by line 16 is on average of the same performance as the reference solution. In the best case it is 14% faster ("gears4"), and in the worst case is 10% slower ("'rings3'). For the given setting of the constants it is faster for 9 out of 30 scenes tested.

The *kd*-tree as described by line 17 – a combination of split clipping and late cutting – is on average 4% faster than the reference. In the best case it is 18% faster ("sombrero1"), in the worst case it is 19% slower ("rings3"). For the given setting of the constants it is faster for 22 scenes out of 30 tested.

The results for *kd*-trees as described by lines 15, 16, and 17 do not correspond with the results that were achieved in the previous experiments [134]. However, these previous experiments were tested for the ad hoc termination criteria setting: $d_{max} = 16$ and $N_{max} = 1$. Therefore we have performed the experiments for the following setting:

Line 18: *kd*-tree constructed with OSAH, and late empty space cutting off is applied, with the setting: $\hat{C}_{IT} = 0.7$ and $\hat{C}_{TS} = 0.3$. The limit for maximum number of planes to be put within late cutting off empty space was set to 3. Ad hoc termination criteria were used: $d_{max} = 16$ and $N_{max} = 1$.

Line 19: *kd*-tree constructed with OSAH, and two-plane cutting off empty space is applied, with the setting: $\hat{C}_{IT} = 0.7$ and $\hat{C}_{TS} = 0.3$. Ad hoc termination criteria were used: $d_{max} = 16$ and $N_{max} = 1$.

Line 20: *kd*-tree constructed with OSAH, and both two-plane and late cutting off empty space is applied, with the setting: $\hat{C}_{IT} = 0.7$ and $\hat{C}_{TS} = 0.3$. Ad hoc termination criteria were used: $d_{max} = 16$ and $N_{max} = 1$.

To evaluate these experiments we used as the reference algorithm the algorithm specified by line 9, which uses the same termination criteria. The results are summarized in Table 4.6.

For the *kd*-tree specified by line 18 we got results that correspond to the previous finding by Sixta [134]. On average, the performance was improved by 9% compared with the reference algorithm. In the best case it was 20% faster ("sombrero2"), and in the worst case 2% faster ("tetra6" and "rings17"). So for all 30 tested scenes late cutting off empty space improved the performance.

Line	Avg	CountBetter	Best	BestName	Worst	WorstName
9	reference algorithm					
18	-9%	30	-20%	sombrero2	-2%	tetra6
19	-9%	29	-18%	sombrero2	+0.5%	lattice29
20	-9%	30	-20%	sombrero2	-2%	tetra6

Table 4.6: Summary table for cutting off empty space for ad hoc termination criteria.

Similarly, for the *kd*-tree specified by line 19 we also got improved results. On average, the performance was improved by 9%, in the best case it was 18% faster ("sombrero2"), and in the worst case 0.5% slower ("lattice29"). For 29 out of 30 scenes the performance was improved.

For the combined solution (line 20) the performance improvement corresponds to that presented for line 18. Combining of the techniques (line 20) did not bring any improvement.

To conclude, both cutting off empty space methods does improve the performance of the *RSA*, when this is possible, *i.e.*, for the *kd*-tree built with ad hoc termination criteria. On the other hand, the results obtained for automatic termination criteria show that it is not likely to improve the performance of an *RSA* based on the *kd*-tree with these additional cutting off empty space methods.

4.10.4 Reducing Objects' Axis-Aligned Bounding Boxes

Here we summarize the results of applying the techniques introduced in Subsection 4.6.2. We used the following settings:

Line 21: *kd*-tree constructed with OSAH using automatic termination criteria (as for line 13). In addition, leaf pruning was used as a postprocessing.

Line 22: *kd*-tree constructed with OSAH using automatic termination criteria (as for line 13). In addition, split clipping was applied.

As a reference algorithm for the following discussion we again take the algorithm specified by line 13, since all the improvements try to increase the performance of this algorithm. The results are summarized in Table 4.7.

Line	Avg	CountBetter	Best	BestName	Worst	WorstName
13	reference algorithm					
18	-6%	18	-41%	teapot12	+19%	gears9
19	-8%	29	-35%	jacks5	+11%	gears9

Table 4.7: Summary table for reducing objects' $\mathcal{AB}$s.

The *kd*-tree specified by line 21 is on average 6% faster than the reference. In the best case it is 41% faster ("teapot12"), and in the worst case 19% slower ("gears9"). For 18 out of 30 tested scenes it is faster than the reference algorithm, but the build time is also slightly increased.

The *kd*-tree specified by line 22 is on average 8% faster than the reference algorithm. In the best case it is 35% faster ("jacks5"), and in the worst case 11% slower ("gears9"). For 22 out of 30 tested scenes it is faster than the reference algorithm.

It might be expected that Θ_{RUN} and thus the whole running time T_R should always be decreased by leaf pruning or split clipping. We have shown above that this is not the case. It always holds that the number of intersection tests per ray is decreased when the ray traversal algorithm for the ray shooting problem is used for all ray shooting queries within the testing procedure TP_D. To test the visibility for a pair of points we have used modified ray traversal algorithm that stops when *any* ray-object intersection is found (see Section 1.3) and the corresponding point of intersection lies at most at some signed distance from the origin of the ray. This enables that the number of traversal steps in the *kd*-tree to be increased, however, the number of ray-object intersection tests can also be increased. We see that modifying the ray traversal algorithm for visibility for a pair of points influences the results for leaf pruning and split clipping.

4.10.5 General Cost Model

Here we describe the results for the general cost model (GCM). We implemented and tested the algorithms with the following settings:

Line 23: *kd*-tree constructed for a variant of GCM that estimates only the blocking factor using the sampling *kd*-tree – a resulting *kd*-tree is constructed using Eqs. 4.35–4.38. The number of sampling rays N_{rays} was determined as $N_{rays} = 0.3 \times N$ with the conditions $N_{rays} \geq 100$ and $N_{rays} \leq 1000$, where N is the number of objects in the node. Automatic termination criteria were used. The estimated cost of hit ray $\hat{C}_{hit}$ was computed as $\hat{C}_{hit} = 1.5 \times \hat{C}$, where $\hat{C}$ is the cost of miss rays computed as a linear estimate taking into account the number of objects.

Line 24: like line 23, but the cost was also estimated in the sampling *kd*-tree, disregarding the difference between the cost of hit and miss rays. The cost of the subtree for one ray was thus computed as $\hat{C} = 1/N_{rays}.(\hat{C}_{IT}.N_{IT} + \hat{C}_{TS}.N_{TS})$ for N_{rays} sampling rays in the sampling *kd*-tree, where N_{IT} was the total number of ray-object intersection tests and N_{TS} was the total number of traversal steps for all sampling rays.

Line 25: like line 24, but distinguishing between the costs for hit and miss rays.

As a reference for the following discussion we again take the *kd*-tree specified by line 13, since the general cost model is intended to increase the performance over the *kd*-tree constructed with a linear cost estimate. The results are summarized in Table 4.8.

Line	Avg	CountBetter	Best	BestName	Worst	WorstName
13	reference algorithm					
23	-1%	17	-12%	tree11	+30%	gears9
24	+17%	3	-5%	tetra6	+123%	rings17
25	+131%	1	-1%	sombrero1	+2637%	rings17

Table 4.8: Summary table for general cost model.

The *kd*-tree specified by line 23 has approximately the same performance as the reference. It is on average 1% faster than the reference, in the best case it is 12% faster than the reference (“tree11”), and in the worst case 30% slower (“gears9”). For 17 out of 30 scenes the general cost model improved the performance, but the build time for the *kd*-tree includes the sampling and thus it is considerably increased.

The *kd*-tree specified by line 24, which tries to estimate the cost via sampling, is on average 17% slower than the reference solution. In the best case it is 5% faster (“tetra6”), and in the worst case 123% slower (“rings17”). We can remark that cost estimate algorithm based on the sampling as described above does not perform well for the scenes with objects whose $\mathcal{AB}s$ overlap.

The *kd*-tree specified by line 25 has even lower performance than the *kd*-tree specified on line 24. On average it is 131% slower, in the best case it is 1% faster (“sombrero1”), and in the worst case 2637% slower (“rings17”).

The summary results of the implemented estimates on the sampling *kd*-tree tree show that the general cost model for most *SPD* scenes has approximately the same performance, but for several scenes the *kd*-tree constructed with GCM has significantly lower performance. With increasing model precision and estimation of the cost via sampling the resulting performance is even decreased.

To conclude, the experiments performed here on a general cost model validate the linear cost estimate. To the best of our knowledge, linear cost estimate achieves the best results for *RSA* based on the *kd*-tree. The algorithm proposed here for general cost model, which estimates the cost of the *kd*-tree to be constructed for a given set of objects using the sampling on another *kd*-tree, in several cases completely degrades the performance.

4.10.6 Preferred Ray Sets

To test the *kd*-tree built for the preferred ray sets we used the automatic termination criteria algorithm for the constructed *kd*-trees, since it guarantees approximately the same number of leaves and thus the performance of the *RSA* from this viewpoint. The use of testing procedures TP_A–TP_D (see Section 3.4.1) has no sense, since we require some particular set of rays with a certain ray distribution. Therefore, tests were performed as for the testing procedure TP_D, but with the depth of recursion set to 1, thus shooting only so called primary rays to the scene (number of rays $N_{rays} = 513 \times 513$), which corresponds to the preferred ray sets only. This holds for perspective, parallel, and spherical projection. The spherical projection uses a viewport size of perspective one, and for parallel projection we constructed a viewport to cover roughly the same portion of the scene in the projected image as for the perspective projection. For all three projections we show the results between the *kd*-tree constructed using OSAH as the reference and the *kd*-tree constructed for the special surface area heuristic intrinsic to the projection.

First, we compared PARSAH with OSAH for parallel projection. We used the following settings:

Line 26: *kd*-tree constructed for OSAH with automatic termination criteria, the rays induced by parallel projection were used.

Line 27: *kd*-tree constructed for PARSAH with automatic termination criteria, the rays induced by parallel projection were used.

To compare line 27 with line 26, we can conclude that the *kd*-tree built using PARSAH was on average 6% faster than OSAH. In the best case it is 46% faster (“jacks3”), and in the worst case it is 71% slower (“rings17”). The *kd*-tree constructed with PARSAH results in lower performance particularly for scenes with objects whose $\mathcal{AB}s$ overlap. The performance improvement also depends on the vector that specifies the projection.

Second, we compared the *kd*-trees constructed for PERSAH with OSAH for rays induced by perspective projection. We used the following settings:

Line 28: *kd*-tree constructed for OSAH with automatic termination criteria. Rays induced by perspective projection were used.

Line 29: *kd*-tree constructed for PERSAH with automatic termination criteria. Rays induced by perspective projection were used.

Line 30: *kd*-tree constructed for SPHSAH with automatic termination criteria. Rays induced by perspective projection were used.

The *kd*-tree constructed with PERSAH (line 29) was faster for 11 scenes than the reference algorithm (line 28). For scenes “lattice6”, ”rings3”, ”lattice12”, ”rings7”, ”lattice29”, and “rings17”, the use of PERSAH obviously results in some degenerated *kd*-trees. For the scene “latticeX” this is understandable, since the viewpoint lies inside the scene $\mathcal{AB}$. If we consider all the scenes, then the *kd*-tree constructed with PERSAH is on average 621% slower than the reference. In the best case it is 27% faster(“tree15”), in the worst case 15330% slower (“lattice29”). If we remove the degenerated *kd*-trees for the scenes “latticeX” and “ringsX”, it is on average 4% faster, in the best case 27% faster (“tree15”), and in the worst case 18% slower (“tetra5”). The advantages of PERSAH over OSAH are questionable for automatic termination criteria, as the time required for constructing the *kd*-tree is significantly increased.

The *kd*-tree constructed as specified by line 30 but for rays induced by perspective projection was faster only than the *kd*-tree constructed for OSAH for three scenes (“mount4”, “balls4”, “mount8”). On average, it is 5293% slower. In the best case it is 11% faster (“mount4”), and in the worst case 153829% slower (“tree15”)). Carefully examining the results we again observe that for several scenes the *kd*-trees constructed with SPHSAH again degenerate – the scenes “sombrero1”, “lattice12”, “sombrero2”, “tree11”, “jacks5”, “rings17”, and particularly the scene “tree15”’.

Third, we examined the results for the rays induces by spherical projection. We used the following settings:

Line 31: *kd*-tree constructed for OSAH with automatic termination criteria. Rays were induced by spherical projection.

Line 32: *kd*-tree constructed for SPHSAH with automatic termination criteria. Rays were induced by spherical projection.

In general, the results achieved for spherical projection are very similar to those obtained by perspective projection, since the distributions of ray sets induced by these two projections are similar.

The *kd*-tree constructed as specified by line 32 is on average 5217% slower than the *kd*-tree constructed for OSAH. In the best case it is 12% faster (“lattice29”), and in the worst case by 153422% slower (“tree15”). It again includes some degenerated cases as for PERSAH, for the scenes ‘’sombrero1”, “lattice12”, “sombrero2”, “tree11”, and particularly “tree15”.

The preprocessing times for OSAH and PARSAH without clipping are quite comparable, since in this case the parallel projection corresponds to multiplying the surface area of the $\mathcal{AB}$ faces by elements of the projection vector defining the preferred set of rays.

The preprocessing time to construct a *kd*-tree with OSAH and PERSAH/SPSAH differ significantly. Even if the clipping algorithm for PERSAH is performed in an incremental way and it thus utilizes the coherence of clipping for subsequent cutting planes along the tested axis, the build time differs by one or two orders of magnitude compared with OSAH. To decrease the build time required by *kd*-tree construction with SPHSAH we used the approximation [24] for arctan in Eq. 4.44.

Parallel Projection in Close-Up

The most promising results for the use of *kd*-trees for preferred ray sets were obtained for parallel projection (PARSAH). Therefore we conducted more experiments with PARSAH, namely as regards the sensitivity of the construction of the *kd*-tree to the direction of ray shooting queries. We used the scene “tree11” with the following initial projection settings: $viewpoint = (4.5, 0, 1.5)$, $lookat = (0, 0, 1.5)$, $upvector = (0, 0, 1)$. This corresponds to the normal of the projection plane $\vec{N} = (1, 0, 0)$ and $azimuth = 0$. In the experiments the observer moved around the tree. That is, $lookat$ and $upvector$ remained constant and both $viewpoint$ and $\vec{N}$ changed according to the $azimuth$ in the range $\langle 0, \pi/2 \rangle$. Setting $azimuth$ to $\pi/2$ corresponds to $viewpoint = (0, 4.5, 1.5)$ and $\vec{N} = (0, 1, 0)$.

First, we compared the *kd*-trees built using OSAH and PARSAH. The *kd*-tree constructed with PARSAH correspond to the azimuth ($\vec{N}$). Fig. 4.14 (a) shows the running time T_R which also includes other computation (shading, the rendered images correspond to ray casting). Fig. 4.15 (a) depicts the average number of ray object intersection tests. Fig. 4.16 (a) shows the average number of traversal steps per ray. The curves for PARSAH are referenced as PARSAH-A.

Second, we tested the sensitivity of the *kd*-tree constructed for a fixed $azimuth = \pi/360$ for shooting the primary rays for other directions than this for which *kd*-tree was constructed. Fig. 4.14 (b) shows the running time for views specified by the azimuth, Fig. 4.15 (b) depicts the average number of ray object intersection tests, and Fig. 4.16 (b) shows the average number of traversal steps per ray. The curve for PARSAH is referenced as PARSAH-B. We can observe that the performance improvement of the *RSA* based on the *kd*-tree constructed with PARSAH is restricted to angular divergence of $\pi.2/3$ from the used for construction.

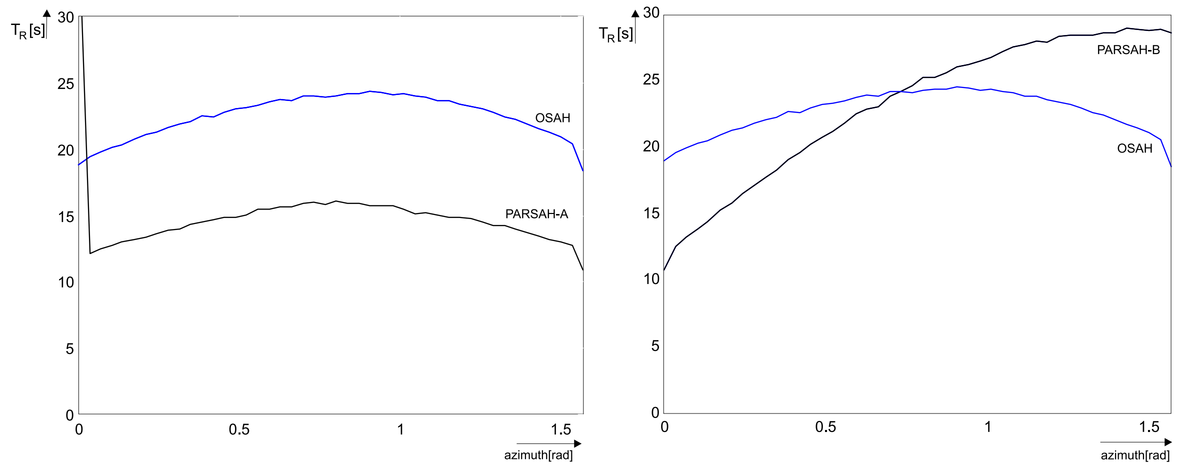
The performance improvements are rather significant. Both the number of intersection tests and the number of traversal steps are reduced by more than one half for the tested scene. The *kd*-tree constructed with PARSAH for $\vec{N} = (1,0,0)$ degenerates, because one principal axis is not taken into account at all. For this reason we selected as the reference for Fig. 4.15 (b) the *kd*-tree constructed for $azimuth = \pi/360$. The difference between the *kd*-tree constructed using OSAH and PARSAH is visualised in Fig. 4.17.

4.11 Conclusion and Future Work

In this chapter we have shown in detail the construction of the *kd*-tree based on a top-down approach. The algorithms presented for *kd*-tree construction are heuristics, solving a problem with a simple and intuitive recursive character: when and where to put the splitting plane for an $\mathcal{AB}$ containing a set of objects. The solution to the problem is not trivial: we show how to decrease the estimated cost of the *kd*-tree to be built up for the worst and average case complexity, under certain simplifying assumptions. However, we cannot claim any optimality of our methods.

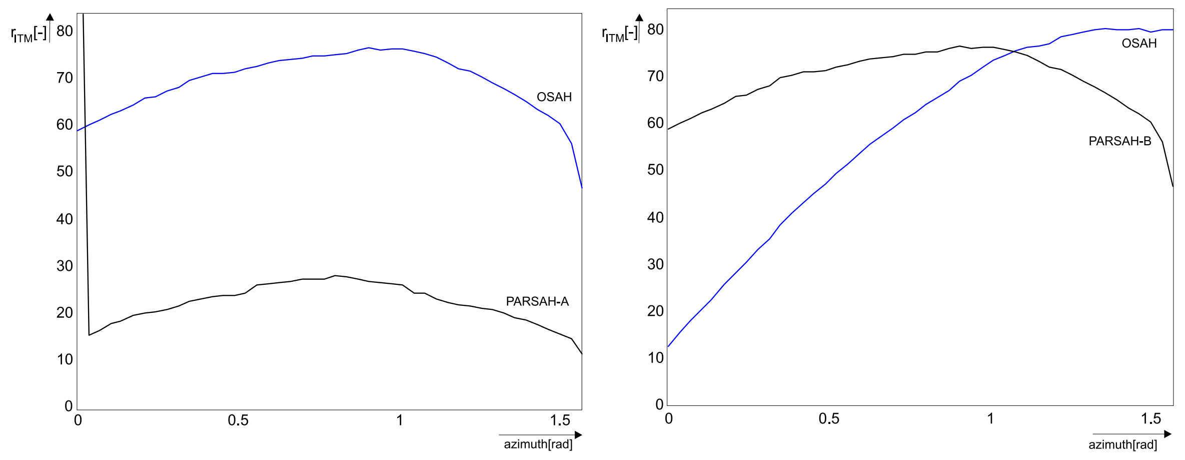
The presented techniques applied in *kd*-tree construction can be combined together in some ways. A technique of preferred ray sets requires some knowledge about the ray distribution of ray shooting queries in advance. The time consumed by *kd*-tree construction is a tradeoff with the performance of answering ray shooting queries. The longer the build time the more efficient would be the *kd*-tree. The upper bound of expected average-case complexity of an *RSA* based on the *kd*-tree for ray shooting can be determined by using a cost model when the *kd*-tree is built.

Future research work concerning *kd*-tree construction for *RSAs* could include a study of more efficient algorithms for estimating the cost, the blocking factor, and the computation of the cost itself. A more intelligent method for searching and utilizing empty spatial regions is a real challenge that should also be studied, and the automatic termination criteria as described should also be further elaborated or change using Russian roulette or other mathematical tools dealing with probability. Another possible research topic could deal with building up a *kd*-tree with some clustering method. Here we have always discussed the *kd*-tree construction using a top-down approach. For scenes with very varying size of objects it may be possible to find sets of objects with roughly the same size that occupy particular spatial regions, to build up the *kd*-trees for such objects sets, and then to take these *kd*-trees as objects within the *kd*-tree construction of a “global” *kd*-tree. Due to the formulation of the cost function in surface area heuristic algorithms, the sets of objects seem to be initially disjoint from the rest of the scene, and the *kd*-tree is then built more or less for clusters of these objects. Clustering could further improve the performance of an *RSA* based on the *kd*-tree in these sparsely occupied scenes, but this needs to be verified experimentally. A study of the degenerated *kd*-tree for preferred ray sets is another interesting topic for research, which could combine ordinary and preferred ray set surface area heuristics. The important issue of further research in the *kd*-tree construction is to what extent and at which cost it is possible to improve the performance of *RSAs* based on the *kd*-tree.



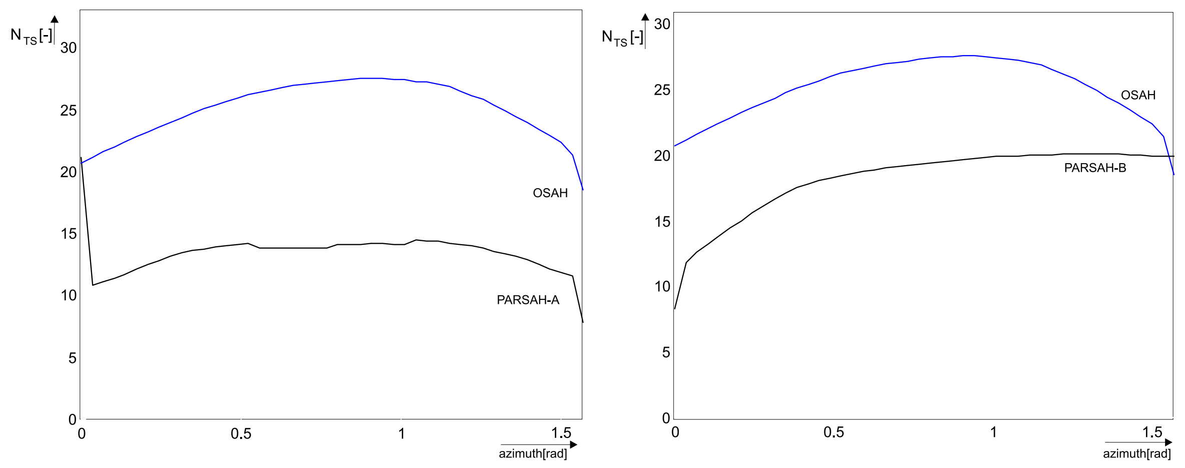
(a) (b)

Figure 4.14: The running time T_R that includes shading in dependence on angle; OSAH and PARSAH-A/B. (a) PARSAH-A and OSAH. (b) PARSAH-B and OSAH.



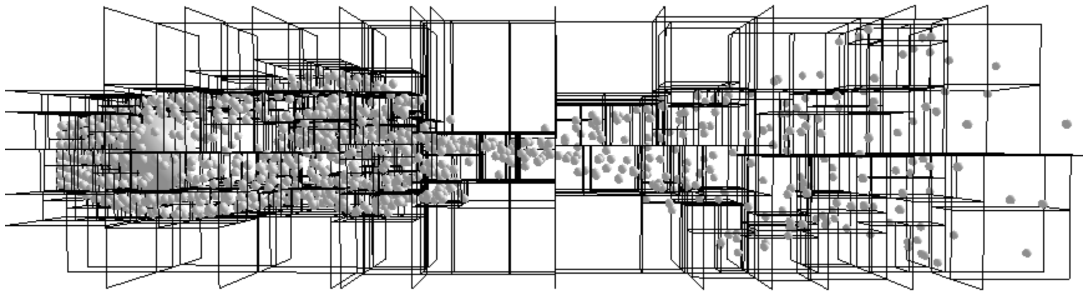
(a) (b)

Figure 4.15: ratio of ray-object intersection tests performed to minimum number of intersection tests r_{ITM} in dependence on angle for OSAH and PARSAH-A/B. (a) PARSAH-A and OSAH. (b) PARSAH-B and OSAH.

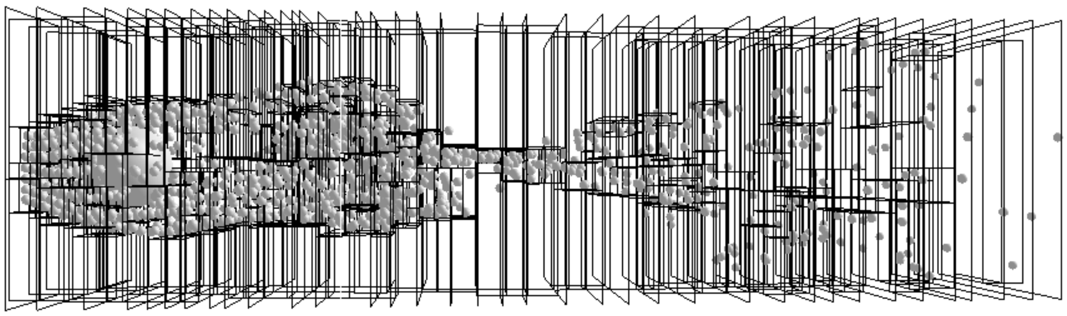


(a) (b)

Figure 4.16: Number of traversal steps per ray N_{TS} in dependence on angle; OSAH and PARSAH-A/B. (a) PARSAH-A and OSAH. (b) PARSAH-B and OSAH.



(a)



(b)

Figure 4.17: Visualization of the *kd*-tree built for a preferred ray set for scene “fluid”. (a) depicts a *kd*-tree built using OSAH. (b) shows a *kd*-tree constructed for PARSAH. For the sake of visual clarity maximum leaf depth d_{max} was set to 10.

Chapter 5

Ray Traversal Algorithms for *Kd*-Trees

In this chapter we deal with ray traversal algorithms for *RSAs* based on the *kd*-tree. First, we describe three types of ray traversal algorithms: sequential, recursive, and those with neighbor-links. Then we analyze the recursive ray traversal algorithm and develop a new more robust version of it. Finally, we present a summary of the results of our experimental comparison of the ray traversal algorithms for G^3_{SPD}, G^4_{SPD}, and G^5_{SPD} scenes.

5.1 Motivation

Given a *kd*-tree, which approximately represents the distance between objects in the scene, we need an algorithm that for a given ray R identifies the sequence of the *kd*-tree leaves intersected by R. We call this algorithm the *ray traversal algorithm.* It should be efficient, robust, and as simple as possible to implement.

The first ray traversal algorithm for the *kd*-tree was developed by Kaplan [94], together with the first use of the BSP tree to accelerate ray tracing. The algorithm based on repetitive computation of a point-location search along the ray path within the *kd*-tree was later called the *sequential ray traversal algorithm.* Further, Jansen [91] introduced a *recursive ray traversal algorithm* that significantly differs from the sequential algorithm in the way it identifies the nodes of the *kd*-tree to be visited. This algorithm recursively descends the branches of the *kd*-tree along the ray path, starting at the root node, and visits each node at most once per ray. The efficiency and robustness of the recursive ray traversal algorithm was further improved by Havran *et al.* [82]. MacDonald and Booth [105] described a ray traversal algorithm that uses neighbor-links starting on the faces of leaves of the *kd*-tree, further elaborated by Havran and Bittner [80]. Here, we call it the *ray traversal algorithm with neighbor-links.*

5.2 Basic Terminology

In this section we describe some terminology used in this chapter. The input of the ray traversal algorithm is a ray R and a *kd*-tree that is constructed for a particular scene $\mathcal{S}$. These two input entities give two basic input configurations according to the mutual position of the origin of a ray and the axis-aligned bounding box of the scene $\mathcal{AB}(\mathcal{S})$. The first configuration is when the origin of a ray is located outside $\mathcal{AB}(\mathcal{S})$. We call this a *ray with external origin.* When the origin of a ray is located inside or on the boundary of $\mathcal{AB}(\mathcal{S})$, we call it a *ray with internal origin.*

Given a ray R and an $\mathcal{AB}$ intersected by R, we can distinguish two important points along the ray path and thus two signed distances. The point where R enters the $\mathcal{AB}$ is called *entry point A*; this corresponds to the *entry signed distance.* The point where R leaves the $\mathcal{AB}$ is called *exit point B*; this corresponds to the *exit signed distance.* A ray traversal algorithm can use knowledge of whether it processes a ray with internal or external origin and possibly some additional data structures.

Each ray traversal algorithm has its own pros and cons, resulting in various average costs of one traversal step $\tilde{C}_{TS}$ and average number of traversal steps per ray (parameters $\tilde{N}_{TS}$, $\tilde{N}_{ETS}$, and $\tilde{N}_{EETS}$ of subset Δ of the minimum testing output, see Chapter 2). A correctly implemented and robust ray traversal algorithm should visit each leaf along the ray path exactly once for an arbitrary ray. In this case only $\tilde{N}_{TS}$ from subset Δ, and T_R, Θ_{rat}, and Θ_{RUN} from subset Θ of the minimum testing output can be influenced by the properties of the ray traversal algorithm. When a ray traversal algorithm requires some additional data structures used only for traversal purposes, subsets Σ and Δ can also be influenced. The C-pseudocodes of all ray traversal algorithms described in this chapter are given in the Appendices.

5.3 Previous Work

In this section we describe in detail all three types of ray traversal algorithms for *kd*-trees developed in the past.

5.3.1 Sequential Ray Traversal Algorithm TA$_{seq}$

The sequential ray traversal algorithm designed by Kaplan [94] is simply the repetitive application of a point-location search in the *kd*-tree along the ray path. We denote the sequential ray traversal algorithm by TA$_{seq}$.

Let us describe TA$_{seq}$. We denote by U the point along the ray. The point U is used to locate currently visited leaf of the *kd*-tree. First, TA$_{seq}$ determines an initial point U along the ray, which serves to search the first leaf to be visited. If a ray has an external origin, the point U is equal to the entry point A of $\mathcal{AB}(\mathcal{S})$. If a ray has an internal origin, then U is equal to the origin of the ray. The point-location search is applied to get a leaf ν^E, where for the point U holds: $U \in \mathcal{AB}(\nu^E)$. If the leaf ν^E is not empty, the objects pointed to in ν^E are tested against the ray for intersection. If any intersected objects are found, the object with the closest intersection point is selected, and it is checked whether the intersection point lies in the $\mathcal{AB}(\nu^E)$. If this is the case, the TA$_{seq}$ is finished. If the leaf ν^E is empty or no object is found to be intersected or the intersection point lies outside $\mathcal{AB}(\nu^E)$, the exit point B for $\mathcal{AB}(\nu^E)$ along the ray is determined. Point B is slightly moved forward along the ray path to ensure the next point-location search finds the next leaf. Then the ray traversal algorithm recurses, point location is applied again, etc. TA$_{seq}$ continues until the closest object along the ray path is found, or exit point B gets outside the $\mathcal{AB}(\mathcal{S})$ – thus no object is intersected. The C-pseudocode of TA$_{seq}$ is given in Appendix A.

TA$_{seq}$ requires us to know the $\mathcal{AB}$*s* associated with all leaves visited along the ray path, even when the leaves are empty. This is necessary for two reasons: to determine the exit point for $\mathcal{AB}$ along the ray path, and to test if the point of intersection between the ray and an object lies in the $\mathcal{AB}$ associated with a leaf. The first way is to store the $\mathcal{AB}$*s* directly in the leaves, which requires six floating-point variables in each leaf in the *kd*-tree. The second way is to compute the size of the $\mathcal{AB}$ associated with a leaf during a point-location search, progressively restricting the size of the $\mathcal{AB}(\mathcal{S})$ of the scene $\mathcal{S}$, however, this increases the average cost of ray traversal step $\tilde{C}_{TS}$.

The obvious disadvantage of TA$_{seq}$ is that it performs many traversal steps within a point-location search. Even if it is very likely that two successive point-location searches traverse a common subsequence of interior nodes starting at the root node, TA$_{seq}$ disregards this fact and always performs the point-location search again, always starting at the root node. When a ray traverses N_l leaves before the object intersected is found, then it visits the root node N_l-times. Although the cost of one traversal step $\tilde{C}_{TS}$ is small, many interior nodes are traversed redundantly for the ray, so the information about the distance between objects encoded in the *kd*-tree is not utilized well.

5.3.2 Recursive Ray Traversal Algorithm TA^{A}_{rec}

A recursive ray traversal algorithm tries to avoid the basic disadvantage of the sequential ray traversal algorithm – each interior node and leaf of the *kd*-tree along the ray path is visited exactly once. Obviously, the average cost of traversal step $\tilde{C}_{TS}$ for the recursive ray traversal algorithm must be higher than that for the sequential algorithm. The cost model for *kd*-tree construction described in the previous chapter was based on the assumption that the recursive ray traversal algorithm is used.

Here, we describe a variant of the recursive traversal algorithm that was first introduced by Jansen [91], which was discussed in more detail by Arvo [15], and was also republished by Sung [148]. We further denote this variant of the recursive ray traversal algorithm by TA^{A}_{rec}.

The pseudocode of the recursive ray traversal algorithm TA^{A}_{rec} was outlined in Chapter 1 as Algorithm 2. When a ray enters the interior node of the *kd*-tree, which has two child nodes, then it decides if both of them are to be traversed and in which order. It classifies the child nodes of the current interior node to be traversed according to the position of the origin of the ray with regard to the splitting plane as “near” and “far” child nodes. When the ray traverses only the “near” child node, then it descends to this node and TA^{A}_{rec} recurses. When the ray has to visit both child nodes, then TA^{A}_{rec} saves the information about the “far ” child node, descends to the “near” child node and then recurses. When no object is found to be intersected inside the “near” child node, the “far” child node is retrieved and TA^{A}_{rec} recurses, starting at the “far” child node.

The detailed C-pseudocode of the efficient implementation of TA^{A}_{rec} is given in Appendix B. The efficient implementation uses the traversal stack to avoid recursion. The stack is used to save the “far” child node if both child nodes are to be visited.

TA^{A}_{rec} uniformly decides which child nodes are to be traversed and in which order. It always computes the signed distance t to the splitting plane in the currently visited interior node. The entry and exit signed distances a and b for the current node are known from previous traversal steps, since they correspond to the signed distances with the splitting planes that were computed in previous traversal steps. For the root node, the entry and exit signed distances (it always holds $a < b$, a is the entry signed distance and b is the exit signed distance) are computed by an explicit algorithm for the intersection between the ray and the $\mathcal{AB}$ [162]. Based on the relation of the signed distances along the ray path t, a, and b it is possible to determine whether to traverse only the “near” child (the case: $(t < a)$ or $(t > b)$), or the “near” child first and then the “far” child (the case: $a < t < b$).

5.3.3 Traversal Algorithms with Neighbor-Links

Here, we describe two variants of a ray traversal algorithm with neighbor-links. Historically, the idea of using the neighbor-links for a ray traversal algorithm in the *kd*-tree was introduced in [105]. However, no clear results of experiments or some theoretical analysis were presented.

The ray traversal algorithm with neighbor-links is based on additional data structures, *i.e.*, neighbor-links starting on the faces of $\mathcal{AB}$*s* associated with the *kd*-tree leaves. We describe the construction of these additional data structures and the corresponding ray traversal algorithm.

5.3.3.1 Motivation

It is supposed that a leaf ν^E of the *kd*-tree can contain at least the list of objects that intersect the $\mathcal{AB}(\nu^E)$ associated with ν^E. As we have seen for the case of the sequential ray traversal algorithm, a ray traversal algorithms can require additional data stored in the *kd*-tree leaves – for example data describing $\mathcal{AB}(\nu^E)$ explicitly. Otherwise, the cost of the traversal step is increased, since the $\mathcal{AB}$*s* of leaves visited would had to be computed on the fly during the point-location search. A ray traversal algorithm that uses neighbor-links must always store these additional data structures explicitly inside the nodes of the *kd*-tree.

The ray traversal algorithm with neighbor-links is another method for eliminating repetitive visiting of the interior nodes for the leaves along the ray path, as occurs in the sequential ray traversal algorithm. Unlike the recursive ray traversal algorithm it requires additional data structures in the leaves of the *kd*-tree. The ray traversal algorithm with neighbor-links can also avoid the down traversal phase needed to locate the first leaf if the origin of a ray is located inside the $\mathcal{AB}(\mathcal{S})$. This down traversal phase occurs in both the sequential and the recursive ray traversal algorithm. In applications with an *RSA* (for example, in global illumination algorithms for higher order rays) it occurs quite frequently that the origin of ray O^{R} is located on the surface of an object O_i in a known leaf ν^E, since O^{R} and O_i were the answer to a previous ray shooting query. It is often the case that this higher-order ray intersects an object close to its origin and thus the down-traversal phase from the root node can form a large portion of the whole time consumed by ray traversal algorithm.

The disadvantage of ray traversal algorithms with neighbor-links is the increased cost of one traversal step $\tilde{C}_{TS}$ compared with the sequential and the recursive ray traversal algorithm. If many leaves are likely to be visited along the ray path, particularly when no object is intersected, then $\tilde{C}_{TS}$ can outweigh the number of traversal steps performed in the total cost of an *RSA* based on the *kd*-tree. For these ray shooting queries the ray traversal algorithm with neighbor-links is thus slower than a recursive ray traversal algorithm. Below, we describe two variants of traversal neighbor-links.

5.3.3.2 Ray Traversal Algorithm with Single Neighbor-Links TA$_{SNL}$

Let us describe a simpler version of a ray traversal algorithm with neighbor-links, which we call the *ray traversal algorithm with single neighbor-links* (denoted by TA$_{SNL}$).

In a *kd*-tree each leaf ν^E is associated with its axis-aligned bounding box, $\mathcal{AB}(\nu^E)$. Each $\mathcal{AB}(\nu^E)$ has six faces of a rectangular shape that we call *leaf-faces.* Let $F(\nu^E)$ denote a leaf-face of the $\mathcal{AB}(\nu)$ associated with the leaf ν^E. For the sake of convenience we here call an $\mathcal{AB}$ associated with a leaf the *leaf-cell.* If two parallel leaf-faces associated with the leaf-cells of two leaves have some intersection of rectangular shape, then the leaves are called *neighbor leaves.*

Given a leaf-face $F(\nu^E)$ and its neighbor leaves there are two mutual geometric relations. Firstly, there is only one neighbor leaf ν corresponding to the leaf-face $F(\nu^E)$ – a leaf-face $F(\nu^E)$ is completely contained in the face of the neighbor leaf-cell $F(\nu)$: $F(\nu^E) \cap F(\nu) = F(\nu^E)$. Secondly, the leaf-face $F(\nu^E)$ can have intersection with the faces of several neighbor leaves.

For a given leaf-face $F(\nu^E)$ we call a *neighbor node* the node ν of the *kd*-tree with the smallest $\mathcal{AB}(\nu)$ for which one face of $\mathcal{AB}(\nu)$ contains $F(\nu^E)$ completely. The neighbor node is either a leaf or an interior node of the *kd*-tree.

A *single neighbor-link* is a link from a leaf-face to its neighbor node. Using these links TA$_{SNL}$ can avoid many traversal steps of interior nodes that would have been performed in the sequential ray traversal algorithm. We have to construct these links and store them for all the faces of all leaves in the *kd*-tree. Fig. 5.1 (b) shows an example of single neighbor-links for a scene in $\mathbb{E}^2$.

Construction Algorithm

The construction of single neighbor-links is straightforward. For each face of each leaf in the *kd*-tree a single neighbor-link is set up. This requires that we store six pointers in each leaf of the *kd*-tree, regardless of whether or not the leaf is empty. For a given leaf-face $F(\nu^E)$ the *kd*-tree is searched, starting from the root node of the *kd*-tree. In each step the search continues in the subtree that corresponds to a cell intersecting the face $F(\nu^E)$. If the face $F(\nu^E)$ is split by the plane referred in the currently reached node (*i.e.*, it intersects both subtrees), the search is terminated. A single neighbor-link to the neighbor node obtained by the search is stored within the leaf-face the algorithm was applied. The C-pseudocode of the algorithm for constructing single neighbor-links is given in Appendix D.

Assume that a kd-tree has n leaves and its average depth is $O(\log n)$. For each face the down traversal takes $O(\log n)$ steps on average. It is applied on $6.n$ faces. Hence the complexity of the algorithm is $O(n.\log n)$.

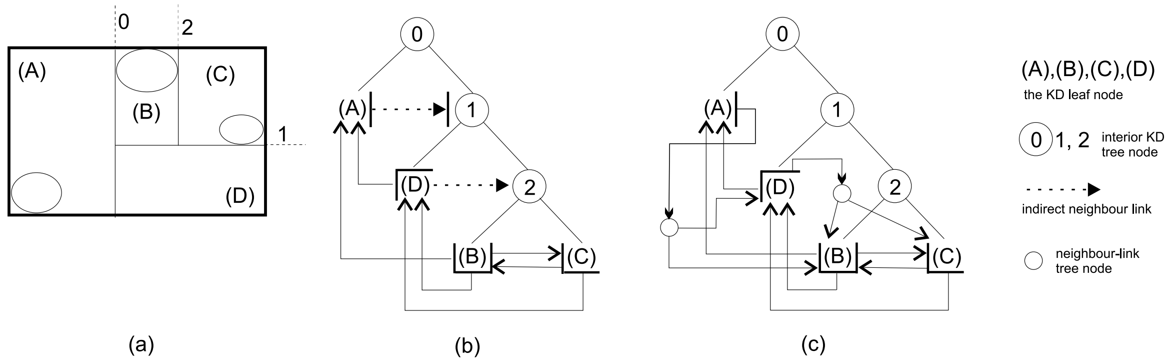


Figure 5.1: (a) Scene in $\mathbb{E}^2$. (b) A kd-tree with single neighbor-links. (c) A kd-tree with neighbor-links trees.

5.3.3.3 Ray Traversal Algorithm with Neighbor-Links Trees TA$_{NLT}$

The ray traversal algorithm TA$_{SNL}$ uses neighbor-links to locate a neighbor leaf for a ray leaving the leaf-face of the current leaf. Single neighbor-links point either to leaves or to interior nodes of the kd-tree. We denote a single neighbor-link that points to an interior node of the kd-tree as an *indirect neighbor-link*. After an indirect neighbor-link is used, the search down to the next leaf on the ray path continues as in the sequential ray traversal algorithm. Each indirect neighbor-link in the kd-tree can be replaced by a *neighbor-links tree*, which solves the search problem more efficiently. The corresponding ray traversal algorithm is more complicated and involves the construction of additional data structures. We call it the *ray traversal algorithm with neighbor-links trees* (denoted by TA$_{NLT}$).

Let us discuss the motivation for replacing indirect neighbor-links with neighbor-links trees. Although a leaf-face is two-dimensional, a search with an indirect neighbor-link is performed in three dimensions. This is the potential inefficiency spot of TA$_{SNL}$. An auxiliary spatial data structure called a neighbor-links tree enables us to replace the three-dimensional search by a two-dimensional search. Further, we discuss the construction of neighbor-links trees in detail.

A neighbor-links tree is a two-dimensional kd-tree that is formed by pruning the splitting planes in the neighbor node to a given leaf-face. The pruned splitting planes are either parallel to the plane supporting the leaf-face or they do not intersect the leaf-face. The first way, pruning the splitting planes parallel to the plane supporting the leaf-face, eliminates one dimension for a point-location search. The second way is based on the projection of the neighbor leaf-faces to the plane supporting a given leaf-face that decreases the number of nodes in a neighbor-links tree and thus the number of traversal steps. Only splitting planes (projected as lines to the plane) intersecting the leaf-face are used in the neighbor-links tree. This pruning corresponds to clipping the two-dimensional kd-tree against a rectangle formed by the leaf-face.

The algorithm constructing the neighbor-links tree replaces an indirect neighbor-link for a leaf-face $F(\nu^E)$ by the corresponding neighbor-links tree. Starting from the node that the indirect neighbor-link points to, a constrained *depth-first-search* (DFS) on the kd-tree is performed. Only subtrees corresponding to the cells that intersect the leaf-face $F(\nu^E)$ are visited during the DFS. Only the nodes where the faces intersect the leaf-face $F(\nu^E)$ are added to the neighbor-links tree. An example of a kd-tree with neighbor-links trees for a scene in $\mathbb{E}^2$ is depicted in Fig. 5.1 (c).

Two-dimensional clipping of a neighbor-links tree is depicted in Fig. 5.2. On the left side of this

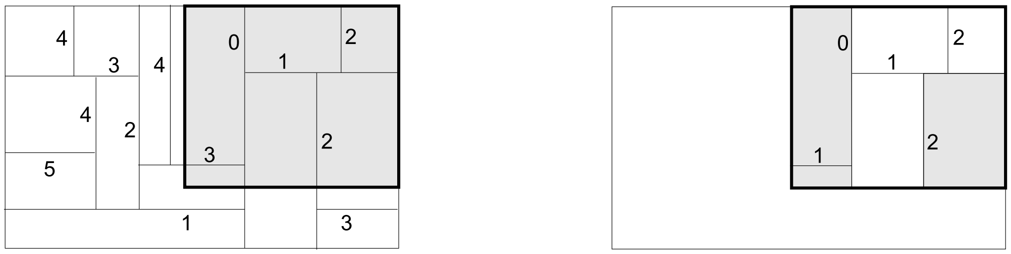


Figure 5.2: Building of neighbor-links tree: left - unclipped, right - clipped.

figure a leaf-face is shown (smaller rectangle with thicker edges) for which the neighbor-links tree is built. The large rectangle is a projection of the corresponding neighbor node subtree onto the plane supporting a given leaf-face. The numbers marking the splitting planes denote the depth of the node in the subtree (the root node of the subtree is at depth zero). Note that the first splitting plane (the root node of the subtree) always intersects the leaf-face which the neighbor-links tree is built for. On the right side of Fig. 5.2 a neighbor-links tree is constructed by clipping the subtree against the leaf-face, which results in the minimum number of splitting planes in the constructed neighbor-links tree. Three grey rectangles corresponding to neighbor leaves depict parts of the leaf-face where the traversal is accelerated owing to the clipping.

5.3.3.4 Ray Traversal Algorithm for Neighbor-Links

The ray traversal algorithm for the *kd*-tree using neighbor-links follows the sequential ray traversal algorithm. It replaces the redundant point-location search that always starts from the root of the *kd*-tree by that one that starts in the node pointed to by a neighbor-link. It also requires to explicitly store the $\mathcal{AB}$ of leaves directly in leaf node structure. The ray traversal algorithm consists of two components: the *exit-face determination* and the *point location.*

Assume the ray traversal algorithm starts in a certain leaf ν^E. When a ray does not intersect any object pointed to in ν^E, we require to locate the next leaf ν^E_{next} so that $\mathcal{AB}(\nu^E_{next})$ is pierced by the ray. First, we determine the *exit-face* $F(\nu^E)$ that is intersected by the ray in its positive direction. Knowing the exit-face $F(\nu^E)$, the *exit-point* B lying along the ray path and on the exit-face $F(\nu^E)$ is computed. Then we follow the neighbor-link corresponding to the exit-face $F(\nu^E)$, thus obtaining a node ν_{next}. When ν_{next} points to a leaf, we follow this leaf directly ($\nu^E_{next} \equiv \nu_{next}$). Otherwise, there are two cases according to use of either TA_{SNL} or TA_{NLT}.

TA_{SNL}: The node ν_{next} corresponds to an indirect neighbor-link and we perform a point location search for exit point B in the kd-tree, but starting at ν_{next}.

TA_{NLT}: The node ν_{next} corresponds to the root node of a neighbor-links tree. We perform a point-location search for exit point B to determine the next leaf ν^E_{next}, but starting at the root node of the neighbor-links tree (ν_{next}), comparing the coordinates of the exit-point B to the splitting planes of the interior nodes of the neighbor-links tree.

If the ray-traversal is terminated in a leaf ν^E_T by an intersecting object at a point I, then the leaf ν^E_T can be used to start the ray-traversal for all the rays spawned from the point I (higher order rays in global illumination algorithms, *etc.*). In this case the initial down-traversal phase to the first leaf is avoided. Most of the traversal steps are eliminated when the ray traversal algorithm visits only a few leaves along the ray.

If the origin of a ray lies outside $\mathcal{AB}(\mathcal{S})$, the entry point A of the ray with respect to $\mathcal{AB}(\mathcal{S})$ must be computed. The entry point A is used to locate the first leaf whose $\mathcal{AB}$ is pierced by the ray. The leaf is found using the point-location search in the *kd*-tree starting at the root node.

5.3.3.5 Algorithm Analysis

The ray traversal algorithm with neighbor-links trees requires some memory to store the two-dimensional *kd*-trees, and thus additional preprocessing. On the other hand, it further decreases the average number of traversal steps per ray, which can be particularly useful when computing the result of ray shooting queries for higher order rays in global illumination algorithms. The starting leaf to initiate the traversal is also known when the viewpoint for primary rays in a global illumination algorithm lies inside the $\mathcal{AB}$ of the scene.

MacDonald and Booth [105] have shown that the average number of neighbor leaf-cells per leaf-face is always lower than two for an octree. It seems difficult to analyze the case of a *kd*-tree with arbitrarily positioned splitting planes. Nevertheless, we have found [80] experimentally that this number is on average bound by a small constant (smaller than four) independently of the depth of the *kd*-tree. This observation is important, since it bounds the average size of neighbor-links trees as well as the average number of point-location search traversal steps (traversal steps within the neighbor-links tree). Assuming that the number of leaves in *kd*-tree is n (when using automatic termination criteria, $n = O(N)$ for N objects), the construction of neighbor-links trees in TA_{NLT} also takes $O(n)$ time, requiring $O(n)$ memory.

5.4 New Recursive Ray Traversal Algorithm

In this section we develop a new recursive ray traversal algorithm TA^{B}_{rec} that is robust and more efficient than TA^{A}_{rec}, *i.e.*, it has smaller expected cost of one traversal step. This algorithm was introduced by Havran *et al.* [82].

Before we start with a description of the new recursive ray traversal algorithm we classify all the possible configurations between a ray and the *kd*-tree interior node geometry that can occur when a ray enters the interior node. This classification allows us to analyze the problems arising with TA^{A}_{rec} and subsequently to design a new more robust ray traversal algorithm.

5.4.1 Traversal Classification

When visiting an interior node ν of the *kd*-tree, we must decide which child node(s) of ν are to be visited and in which order. The recursive ray traversal algorithm decides among some traversal cases induced by the geometry of the problem. The input of a the algorithm is a ray R, the axis-aligned bounding box $\mathcal{AB}(\nu)$, the position and orientation of a splitting plane Π that splits $\mathcal{AB}(\nu)$ into two new $\mathcal{AB}s$ associated with the child nodes. From this input data we can compute the intersection point I between R and Π, the entry point A and the exit point B between R and $\mathcal{AB}(\nu)$. The relationships between I, A, and B then specify which child nodes are to be visited and in which order.

Let us classify all possible traversal cases. The classification is depicted in Fig. 5.3 for one orientation of the splitting plane. In the left column the origins of rays are located below the splitting plane (*negative*, case N), in the middle column above the splitting plane (*positive*, case P), and in the right column the origins of rays are embedded in the splitting plane (*zero*, case Z). A local coordinate system referenced to the splitting plane is taken for classification. We call the child node below the splitting plane *left* and above the splitting plane *right.*

The classification depicts the rays with external origin with a thick cross, and possible internal origin is denoted by a thin cross. A ray can have an internal origin ($a < 0$) in all the cases depicted in Fig. 5.3, excluding cases N5 and P5, because these would become equivalent to cases P1 and N1, respectively.

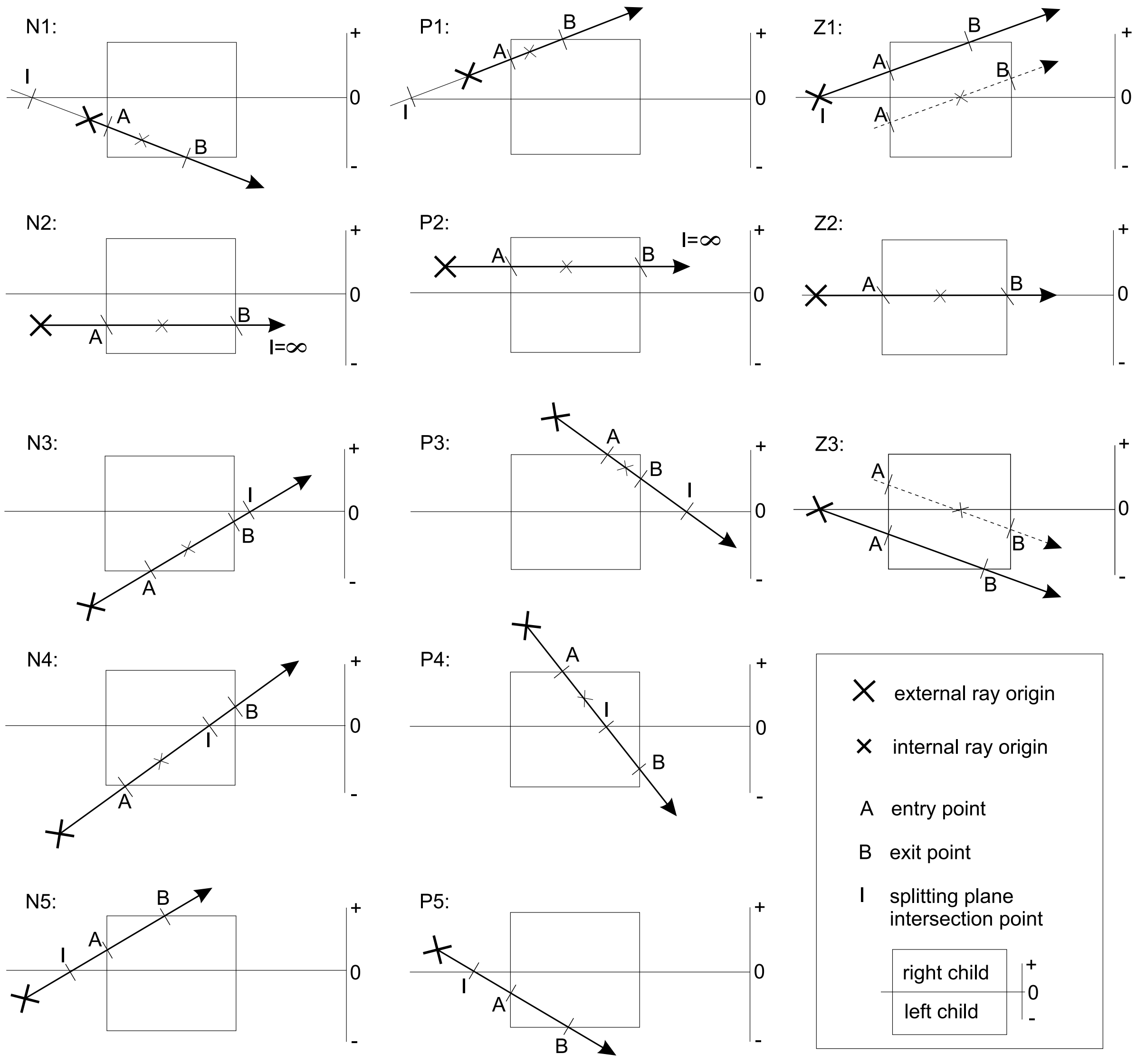


Figure 5.3: Classification of mutual positions between a ray and a *kd*-tree interior node.

5.4.2 Analysis of TA^A_{rec}

Here we analyze TA^A_{rec} and show our motivation for designing a new recursive ray traversal algorithm. The traversal step based on comparing the signed distances in TA^A_{rec} suffers from the robustness problem if the origin of a ray is embedded in the splitting plane (traversal cases Z1 and Z3 in Fig. 5.3). In this case TA^A_{rec} cannot correctly determine whether to traverse either the left or the right child. This can lead to an incorrect selection of the child node to be traversed in the next step. The traversal cases N2 and P2 are always solved correctly, because the signed distance is computed as an overflow (positive or negative) and the “near” child is selected. For case Z2 any of the two child nodes can be selected to obtain a correct result.

When an incorrect child to visit is selected for cases Z1 and Z3, the result of the *RSA* is incorrect. The impact on the visual quality of an image in a global illumination algorithm that uses TA^A_{rec} can be significant. For example, in ray casting when the viewer position and thus the origin of the primary rays is embedded in the splitting plane of the root node of the *kd*-tree, then the part of the image is incorrect (in the worst-case it is completely missing). We have found two ways to deal with the robustness problem of TA^{A}_{rec}. First, we can add two more conditions to TA^{A}_{rec} to recognize the occurrence of case Z1 and Z3 (if $-\varepsilon < t < \varepsilon$), where ε is a small positive constant) and then one more condition to distinguish between Z1 and Z3. This apparently further increases the average cost of traversal step $\tilde{C}_{TS}$. Second, we can look at the problem of a ray traversal algorithm from a different perspective, and design a new recursive traversal algorithm. We follow this second way below.

5.4.3 Design of a Recursive Ray Traversal Algorithm TA^{B}_{rec}

Uniformity of the algorithm determining between traversal cases, and lack of robustness, can been seen as the main disadvantages of TA^{A}_{rec}. Therefore, we analyze the problem of traversing a ray in the *kd*-tree more thoroughly, and design a new recursive ray traversal algorithm, further denoted by TA^{B}_{rec}. It has lower expected average cost for one traversal step $\tilde{C}_{TS}$ and, in addition, it always solves all the traversal cases depicted in Fig. 5.3 correctly.

5.4.3.1 Theoretical Considerations

The idea behind improving the efficiency in TA^{B}_{rec} is statistical optimization, and making use of the perpendicularity of the splitting planes to the coordinate axes. Traversal cases occurring less frequently can be performed with a higher cost, and traversal cases occurring more frequently will be performed with a lower cost. The most time-consuming traversal cases of TA^{A}_{rec} are N4 and P4; the "far" child is pushed onto the stack. The probability of these traversal cases is important for the efficiency of the ray traversal algorithm.

We should thus estimate the probability of various traversal cases. We estimate them using the geometric probability tools already described in Subsubsection 4.2.3.1. We assume that $\mathcal{AB}(\nu)$ associated with an interior node ν is split in the spatial median (see Section 4.2.2), subdividing the $\mathcal{AB}(\nu)$ into halves – this corresponds to the BSP tree construction as depicted in Fig. 5.4. Let the size of $\mathcal{AB}(\nu)$ in $\mathbb{E}^3$ be $w \times d \times h$. For the sake of simplicity of the analysis, we further assume that the $\mathcal{AB}(\nu)$ is cubic in shape ($w = d = h$). In $\mathbb{E}^3$ we need three subdivision steps to get the leaves associated with the $\mathcal{AB}$s cubic in shape again.

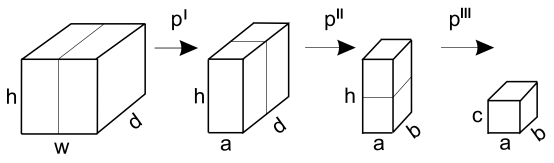


Figure 5.4: Three-step subdivision of $\mathcal{AB}$ using a spatial median.

We can determine the probabilities that a ray pierces the left node (p_{LO}) only, the right node (p_{RO}) only, and both of them ($p_{\mathrm{LR}}, p_{\mathrm{RL}}$). We demonstrate the use of geometric probability to compute $p_{\mathrm{LR}} + p_{\mathrm{RL}}$, which corresponds to the traversal cases N4 and P4. Further, we denote $p_{LR+RL} = p_{LR} + p_{RL}$, as shown in Subsection 4.3.1, $p_{LR} = p_{RL}$, since this corresponds to the case when a ray intersects the splitting plane inside the $\mathcal{AB}$. The probability of intersecting the splitting plane put in the first subdivision step p^{I}_{LR+RL} ($w = d = h$, $a = w/2$) is computed as:

$$p^{I}_{LR+RL} = \frac{d.h}{w.h + w.d + d.h} = \frac{1}{3} \tag{5.1}$$

The probability p^{II}_{LR+RL} of intersecting the splitting plane put in the second subdivision step ($h = d$, $a = b = h/2$) is:

$$p^{II}_{LR+RL} = \frac{a.h}{a.d + d.h + a.h} = \frac{1}{4} \tag{5.2}$$

Similarly, the probability p^{III}_{LR+RL} of intersecting the splitting plane put in the third subdivision step ($a = b = c = h/2$) is:

$$p^{III}_{LR+RL} = \frac{a.b}{b.h + a.h + a.b} = \frac{1}{5} \tag{5.3}$$

Since the probabilities are rational linear functions, it can be shown that they have no local extreme for any of w, d, h, a, b, and c. Since the subdivision steps are mutually independent starting with the $\mathcal{AB}$ of cubic shape, we can compute the average probability p_{LR+RL} as follows:

$$p_{LR+RL} = \frac{1}{3}.(p^{I}_{LR+RL} + p^{II}_{LR+RL} + p^{III}_{LR+RL}) = \frac{47}{180} \doteq 0.261 \tag{5.4}$$

When a ray traverses within the recursive ray traversal algorithm the whole *kd*-tree without intersecting an object, it can be shown that, for rays with uniform distribution probability, p_{LR+RL} depends on the sum of the surface areas of $\mathcal{AB}$ faces in the *kd*-tree formed by the splitting planes. Let us remark that the statistically worst case occurs for the spatial median subdivision of the cubic cell: $w = d = h$, $a = b = c = w/2$. In this case the faces corresponding to the splitting planes are also uniformly distributed in the *kd*-tree, so the probability of intersecting the splitting planes by uniformly distributed rays is the highest possible.

5.4.3.2 Experimental Statistics

We support our theoretical considerations by the results of experiments for testing procedure TP_D, which were performed on several scenes from G^4_{SPD} (see Section 3.5 for more information about the scenes). Table 5.1 shows the statistics of the traversal cases, and the underlying *kd*-trees were built with ordinary surface area heuristic.

	Scene				
Probability	balls4	tetra6	mount6	gears4	Average
$p_{\mathrm{N1+P1}}$	0.296	0.18	0.434	0.439	0.337
$p_{\mathrm{N2+P2}}$	0.0004	0.0	0.0002	0.0004	0.0003
$p_{\mathrm{N3+P3}}$	0.328	0.39	0.292	0.345	0.339
$p_{\mathrm{N4+P4}}$	0.244	0.208	0.205	0.158	0.204
$p_{\mathrm{N5+P5}}$	0.133	0.221	0.07	0.057	0.12
$p_{\mathrm{Z1+Z2+Z3}}$	0.0	0.0	0.0	0.0	0.0

Table 5.1: Traversal case statistics for *SPD* scenes for TP_D.

Pop operations from stack are not included in the table to get the probabilities in correspondence with the theoretical analysis. These pop operations are performed with a lower probability than $p_{\mathrm{N4+P4}}$, since when a ray hits an object the remaining nodes stored in the stack are not used.

We can see that $p_{\mathrm{N4+P4}} < p_{LR+RL}$, which enables us to design a new ray traversal algorithm with improved performance if cases other than N4 and P4 are solved efficiently in the new algorithm.

5.4.3.3 New Recursive Ray Traversal Algorithm TA^{B}_{rec}

The use of the signed distances for classifying the traversal cases in TA^{A}_{rec} leads to the robustness problem. For this reason algorithm TA^{B}_{rec} discards the concept of “near” and “far” child nodes and simplifies the decision phase in the traversal step to the following traversal cases:

- visit the left child only,
- visit the left child first and the right child afterwards,
- visit the right child first and the left child afterwards,
- visit the right child only.

Theoretically, these four traversal cases could be distinguished by performing at least $\lceil \log_2 4 \rceil = 2$ comparisons, which takes a lower number of conditions than the number of conditions used in TA^{A}_{rec}.

The algorithm TA^{B}_{rec} uses for the decision step a decomposition of the traversal cases based on the mutual position of the entry/exit point and the splitting plane position given the splitting plane orientation. Without loss of generality, we further assume that the splitting plane is perpendicular to the x-axis. We show the difference between TA^{A}_{rec} and TA^{B}_{rec} on the traversal case N1, see Fig. 5.5.

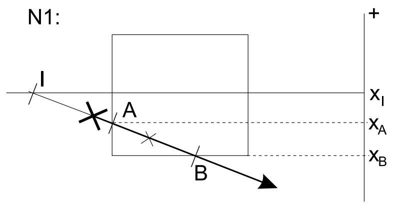


Figure 5.5: Traversal case N1.

Algorithm TA^{A}_{rec}: First, the signed distance t to the splitting plane is computed. Then the “near” and “far” children are determined; in Fig. 5.5 the “near” child is below the splitting plane and the “far” child is above the splitting plane. If the signed distance is smaller than zero ($t < 0.0$), the “near” child is selected.

Algorithm TA^{B}_{rec}: If the projection of entry point A to the current axis (x-axis, which corresponds to the orientation of the splitting plane) is less than or equal to the position of the splitting plane ($x_A \leq x_I$) and the projection of exit point B to the current axis is less than or equal to the position of the splitting plane ($x_B \leq x_I$), the child below the splitting plane (left) is selected.

The algorithm TA^{B}_{rec} does not require the signed distance to the splitting plane to distinguish between traversal cases. This is enabled by the orthogonality of the splitting planes in the axis-aligned form of the *kd*-tree. TA^{B}_{rec} then uses a comparison between the coordinates of the entry/exit point and the position of the splitting plane.

The classification of traversal cases is simplified to a comparison between real numbers in TA^{B}_{rec}. The penalty paid for this low cost traversal step is the need to compute all coordinates of the new exit point lying on the splitting plane for the traversal cases N4 and P4. This is the most time consuming part of TA^{B}_{rec}. As we have shown above, the case N4 and P4 occurs with sufficiently low probability ($p \doteq 0.26$) so that the total efficiency of TA^{B}_{rec} can be improved in comparison with TA^{A}_{rec}. The C-pseudocode of the efficient version of TA^{B}_{rec} is given in Appendix C.

5.4.3.4 Handling Singular Traversal Cases

The algorithm TA^{B}_{rec} deals with the singular cases (Z1, Z2, and Z3) correctly, because the classification is based on a comparison of the splitting plane position and the entry/exit point coordinates.

The issue of calculation imprecision is important for computing the signed distance to the splitting plane and thus new exit point coordinates. Naturally, during the descending phase to the first leaf the signed distance of a new exit point is smaller than for the current exit point. An incorrect result would occur due to the finite representation of real numbers, and therefore we should analyze this case. Let u denote the difference between one of the coordinates of the origin of a ray and the splitting plane, and let D^{R}_{a} denote the component of the ray direction for the a-axis. For the normalized vector of the ray direction always holds $D_a^{\mathrm{R}} \leq 1.0$. The signed distance to the splitting plane is computed as: $t = u/D_a^{\mathrm{R}}$. From knowledge about the representation of real numbers [1] it can be shown that imprecision can arise for $|D_a^{\mathrm{R}}| \ll 1$ and $|u| \gg |D_a^{\mathrm{R}}|$. We can show by contradiction that this problem with calculation imprecision does not arise in TA_{rec}^{B}, since the range for u and v as stated above cannot occur in the traversal cases N4 and P4.

5.4.4 Comparison between TA_{rec}^{A} and TA_{rec}^{B}

The time complexity of traversal cases in TA_{rec}^{A} and TA_{rec}^{B} can be expressed in terms of the arithmetical operations required. Table 5.2 shows the traversal complexity for all traversal cases in the algorithm TA_{rec}^{A}.

	Arithmetical operation					
Traversal case	$=$	$<$	$\pm$	$\times$	$/$	$\frac{Inc}{Dec}$
N1(P1)	5	4	1	0	1	0
N2(P2)	5	3	1	0	1	0
N3(P3)	5	3	1	0	1	0
N4(P4)	9	5	1	0	1	1
N5(P5)	5	5	1	0	1	0
Z2	5	3	1	0	1	0
pop	3	1	0	0	0	1

Table 5.2: Traversal complexity for the algorithm TA_{rec}^{A}in terms of arithmetic operations.

Table 5.3 gives the traversal complexity in terms of arithmetical operations for all traversal cases in the algorithm TA_{rec}^{B}.

	Arithmetical operation					
Traversal case	$=$	$<$	$\pm$	$\times$	$/$	$\frac{Inc}{Dec}$
N1(P1)	2	3	0	0	0	0
N2(P2)	2	3	0	0	0	0
N3(P3)	2	3	0	0	0	0
N4	11	5	3	2	1	1.3
P4	11	4	3	2	1	1.3
N5(P5)	2	3	0	0	0	0
Z1	2	4	0	0	0	0
Z2(P3)	2	3	0	0	0	0
pop	3	1	0	0	0	0

Table 5.3: Traversal complexity for the algorithm TA_{rec}^{B} in terms of arithmetic operations.

Below, we compare TA_{rec}^{A} and TA_{rec}^{B} based on the average number of arithmetical operations required for one traversal step. The probabilities for different traversal cases are obtained by averaging the results of experiments for several scenes.

Since the recursive ray traversal algorithm also includes the pop operation, this must be included in the statistics unlike for Table 5.1. That is why the probability values presented in Table 5.1 differ from those presented in Table 5.4. Since objects can be hit during the traversal, the probability for pop operation p_{pop} is smaller than probability $p_{\mathrm{N4+P4}}$. The probabilities in Table 5.4 are the most unfavorable for algorithm TA_{rec}^{B}, which includes the highest $p_{\mathrm{N4+P4}}$.

Probability	Without pop	With pop
$p_{\text{N1+P1}}$	0.308	0.262
$p_{\text{N2+P2}}$	0.0002	0.0002
$p_{\text{N3+P3}}$	0.294	0.25
$p_{\text{N4+P4}}$	0.261	0.2236
$p_{\text{N5+P5}}$	0.134	0.114
$p_{\text{Z1,2,3}}$	0.0	0.0
p_{pop}	–	0.15
$\sum p$	1.0	1.0

Table 5.4: Probabilities used for comparison of TA^{A}_{rec} and TA^{B}_{rec}.

Table 5.5 shows a comparison of TA^{A}_{rec} and TA^{B}_{rec} based on the number of arithmetical operations to be computed. The number of these operations was computed as a weighted sum of the counts of the arithmetical operations shown in Table 5.2 and Table 5.3 using the weights given in Table 5.4. In the statistically worst case, the theoretical speedup of the TA^{B}_{rec} over TA^{A}_{rec} achieves a value of about 1.2.

	Arithmetical operation					
Algorithm	$=$	$<$	$\pm$	$\times$	$/$	$\frac{Inc}{Dec}$
TA^{A}_{rec} [-]	5.60	3.07	0.85	0.0	0.85	0.37
TA^{B}_{rec} [-]	4.01	3.03	0.67	0.45	0.22	0.29
$\frac{\text{TA}^{A}_{rec}}{\text{TA}^{B}_{rec}}$ [-]	1.40	1.01	1.27	–	3.79	1.28

Table 5.5: Comparison between TA^{A}_{rec} and TA^{B}_{rec} based on number of arithmetical operations.

We also compared TA^{A}_{rec} and TA^{B}_{rec} experimentally. The results of experiments are included for all G^{3}_{SPD}, G^{4}_{SPD}, and G^{5}_{SPD} *SPD* scenes in Appendix E, and are summarized in the next section. Obviously, the parameters T_R, Θ_{rat}, and Θ_{RUN} of the minimum testing output (see Section 2.5) differ. In summary, the cost of one traversal step of TA^{B}_{rec} was decreased by 22% on average compared with TA^{A}_{rec}. The results [82] of us from experiments that were performed on different computer architecture (MIPS R4400, 200MHz running Irix 6.2) showed us that the cost of the traversal step of TA^{B}_{rec} can be decreased even more, by 47% lower on average than for TA^{A}_{rec}, but it reached from 53% to 35% in dependence on the input scene. The results of the previous experiments are presented in Table 5.6.

We do not want to speculate why the results of experiments performed on various computed architectures differ, since this is a problem inherent to the points *HW* and *COMP*, see Section 2.7. A possible reason is that MIPS architecture can execute better the part of TA^{B}_{rec}, which computes traversal cases N4 and P4, since most of the computation can be performed using more than one floating point unit.

The ray traversal algorithm TA^{B}_{rec} handles all the traversal cases correctly and always performs better than TA^{A}_{rec}. An interesting property of TA^{B}_{rec} is that the entry and exit points, which in TA^{B}_{rec}, unlike TA^{A}_{rec}, are always computed, can be used directly for ray-bounding box culling [136].

5.5 Summary of Results

In this section we summarize the results of experiments on all ray traversal algorithms described above. Since the testing procedure TP_D is the same, the input set of query rays, the parameters in minimum testing output influenced by the ray traversal algorithm are only the following:

	Scene			
Algorithm	balls4	tetra6	mount6	gears4
$T_R(\mathrm{TA}^A_{rec})$ [s]	94.21	7.86	122.4	307.7
$T_{TS}(\mathrm{TA}^A_{rec})$ [s]	45.58	4.34	52.97	58.52
$T_{TS}(\mathrm{TA}^B_{rec})$ [s]	21.71	2.81	24.83	32.22
$r_{TS}(\frac{\mathrm{TA}^B_{rec}}{\mathrm{TA}^A_{rec}})$ [-]	0.48	0.65	0.47	0.55

Table 5.6: Experimental comparison between TA^A_{rec} and TA^B_{rec} for MIPS R4400, 200MHz running operating system Irix 6.2. Parameter T_R refers to the total running time of ray tracing (TP_D), T_{TS} to the time devoted to traversing the *kd*-tree ($T_{TS} = (1 - \Theta_{rat}).T_R.\frac{\Theta_{RUN}}{\Theta_{APP}+\Theta_{RUN}}$), and r_{TS} shows the ratio between the times devoted to traversing the *kd*-tree for TA^A_{rec} and TA^B_{rec}.

- Subset Σ: N_G – for TA_{NLT} the number of generic nodes is increased by the nodes of neighbor-links trees,
- Subset Δ: $\tilde{N}_{TS}$ – the average number of traversal steps per ray,
- Subset Θ: T_R, Θ_{rat}, and Θ_{RUN} – the time portion of *RSA* devoted to traversing the *kd*-tree, since the average cost of traversal steps $\tilde{C}_{TS}$ and $\tilde{N}_{TS}$ is specific to ray traversal algorithms. The time devoted to traversing the *kd*-tree is $T_{TS} = (1 - \Theta_{rat}).T_R.\frac{\Theta_{RUN}}{\Theta_{APP}+\Theta_{RUN}}$.

The results of experiments are fully given in Appendix E, let us describe the settings used. The setting is always described for a line X, which corresponds to denotation in Appendix E. The following ray traversal algorithms were tested:

Line 33, 38, and 43: – TA_{seq} – sequential ray traversal algorithm (Section 5.3.1),

Line 39, 44, and 34: – TA^A_{rec} – recursive traversal algorithm (Section 5.3.2),

Line 40, 45, and 35: – TA^B_{rec} – recursive traversal algorithm (Section 5.4.3),

Line 41, 46, and 36: – TA_{SNL} – traversal algorithm with single neighbor-links (Section 5.3.3.2),

Line 42, 47, and 37: – TA_{NLT} – traversal algorithm with single neighbor-links trees (Section 5.3.3.3).

We decided to verify the time required by ray traversal algorithms on *kd*-trees built with different number of leaves. We used the following three different settings to build up the *kd*-trees on these lines in Appendix E:

Line 33, 34, 35, 36, and 37: – setting *CT1* – OSAH + ad hoc termination criteria with this setting: maximum leaf depth $d_{max} = 16$ and the number of objects in leaves $N_{max} = 2$ (Subsubsection 4.2.4.1),

Line 38, 39, 40, 41, and 42: – setting *CT2* – OSAH + ad hoc termination criteria with this setting: maximum leaf depth $d_{max} = 18$ and the number of objects in leaves $N_{max} = 2$,

Line 43, 44, 45, 46, and 47: – setting *CT3* – OSAH + automatic termination criteria with this setting: $k_1 = 1.2$, $k_2 = 2.0$, $r_q^{min} = 0.75$, $K^1_{fail} = 1.0$, and $K^2_{fail} = 0.2$ (Section 4.5).

The experiments were performed using the testing procedure TP_D, in total for one ray traversal algorithm we performed $30 \times 3 = 90$ experiments. The ray sets induced by TP_D contain rays with both external and internal origin. The known origins of higher order rays were used to initiate the traversal when testing the ray traversal algorithms with neighbor-links (TA$_{SNL}$ and TA$_{NLT}$).

Below, we compare the ray traversal algorithms quantitatively. As a reference for comparison we take the recursive ray traversal algorithm TA$^B_{rec}$, since in most cases it achieved the best results. We can summarize the results of the experiments as follows:

- TA$_{seq}$ – the slowest ray traversal algorithm for all experiments performed. It was in all cases slower than the reference algorithm TA$^B_{rec}$. The time for traversing the *kd*-tree (T_{TS}) was 60% higher on average than the time for the reference algorithm, in the best case 12% ("gears9" with *CT3* setting), and in the worst case 153% ("lattice29" with *CT1* setting). The time T_R was increased by average by 37%, at least by 8%, and at most by 100% in comparison with TA$^B_{rec}$. The average number of traversal steps per ray was considerably higher than for the reference algorithm, on average by 135%, at least by 69% ("mount4" with *CT1* setting) and at most by 215% ("jacks5" with *CT1* setting).
- TA$^A_{REC}$ – the traversal algorithm with reasonable performance. It was also slower than reference ray traversal algorithm TA$^B_{rec}$ for all 90 experiments. The time for traversing the *kd*-tree (T_{TS}) was 16% higher on average than the reference algorithm, in the best case 5% ("tree15" with *CT2* setting), and in the worst case 36% ("tetra8" with *CT1* setting). The time T_R was increased on average by 10%, at least by 4% and at most by 20%. The average number of traversal steps per ray was practically equal to the reference algorithm for all the experiments, so no visual artefacts due to the lack of robustness of the algorithm were found.
- TA$_{SNL}$ – the ray traversal algorithm with single neighbor-links performed better than the reference algorithm for 20 out of 90 experiments. The improved cases cover the experiments for scenes with many secondary rays, where the algorithm TA$_{SNL}$ eliminates the down-traversal phase required by the recursive ray traversal algorithm. However, the performance of TA$_{SNL}$ for other scenes was lower than for the reference algorithm. The time for traversing the *kd*-tree T_{TS} was 8% higher on average than the reference algorithm. In the best case, T_{TS} of TA$_{SNL}$ was decreased by 18% compared with the reference algorithm ("mount8" with *CT1* setting), and in the worst case it was increased by 75% ("lattice29" with *CT1* setting). The time T_R was increased on average by 6% compared with the reference algorithm, in the best case it was 10% lower, and in the worst case it was 62% higher. The average number of traversal steps per ray $\tilde{N}_{TS}$ was on average decreased by 16% ("jacks3" with *CT1* setting), at least by 1% and at most by 49% ('lattice29" with *CT2* setting).
- TA$_{NLT}$ – for 18 out of 90 experiments the ray traversal algorithm with neighbor-links trees achieved better performance than the reference algorithm. For one case there was not enough memory to construct the neighbor-links trees for one experiment (the scene "lattice29" with *CT3* setting). The time for traversing the *kd*-tree (T_{TS}) was 17% higher on average than the reference algorithm. In the best case T_{TS} was decreased by 16% ("mount4", "mount6", and "mount8" with *CT3* setting and "mount4" with *CT1* setting), and in the worst case it was increased by 307% ("teapot40" with *CT1* setting). The time T_R was increased on average by 12% than the reference algorithm, in the best case it was 9% lower, and in the worst case by 198% higher. These results contradict with the results of TA$_{SNL}$, since the running times for TA$_{NLT}$ should be better than for TA$_{SNL}$. It was probably caused by swapping virtual memory to the disk and lack of robustness and precision in measuring the user time of the process in the UNIX environment, since the construction of neighbor-links trees required some additional memory. The number of traversal steps per ray $\tilde{N}_{TS}$ was on average decreased by 24% when compared with the reference algorithm, at least by 8% ("tree8" with *CT1* setting) and at most by 50% (scene "lattice29" with *CT2* setting);

  It is also interesting to compare TA$_{NLT}$ directly with TA$_{SNL}$. Concerning subset Δ of the minimum testing output, the time T_R of TA$_{NLT}$ was increased on average probably due to the above-mentioned swapping of memory that was allocated for neighbor-links trees. The average number of nodes allocated for neighbor-links trees was about six nodes per leaf of the *kd*-tree, *i.e.*, one node for each leaf-face.

For the sake of clarity, all the average, minimum, and maximum values are summarized in Table 5.7, where TA^B_{REC} is taken as the reference algorithm.

Algorithm	Parameter								
	$\tilde{T}_{TS}$	$m(T_{TS})$	$M(T_{TS})$	$\tilde{T}_R$	$m(T_R)$	$M(T_R)$	$\tilde{N}_{TS}$	$m(\tilde{N}_{TS})$	$M(\tilde{N}_{TS})$
TA_{SEQ}	60%	12%	153%	37%	8%	100%	135%	69%	215 %
TA^A_{REC}	16%	5%	36%	10%	4%	20%	0%	0%	0%
TA^B_{REC}	-	-	-	-	-	-	-	-	-
TA_{SNL}	8%	-18%	75%	6%	-10%	62%	-16%	-1%	-49%
TA_{NLT}	16%	-16%	307%	12%	-9%	198%	-24%	-8%	-50%

Table 5.7: A summary comparison of ray traversal algorithms, where TA^B_{REC} is taken as the reference – average, minimum and maximum values for T_{TS}, T_R, and $\tilde{N}_{TS}$ for all 90 experiments for one ray traversal algorithm. Testing procedure TP_D was used. Symbol M denotes the maximum, m the minimum achieved.

The difference of performance for all ray traversal algorithms when testing on the *kd*-trees built for the different termination criteria (*CT1*, *CT2*, and *CT3*) was not significant. We only verified that the higher the number of nodes in the *kd*-tree, the lower the performance of TA_{SEQ} compared with TA^B_{REC}.

5.6 Conclusion and Future Work

In this chapter we described three types of ray traversal algorithms for the *kd*-tree, one variant of the sequential ray traversal algorithm, two variants of the recursive ray traversal algorithm, and two variants of the ray traversal algorithm with neighbor-links. All these algorithms for a given ray and a scene identify the same sequence of leaves, but they differ in number of visited interior nodes and in the way in which the leaves are determined.

For the tested scenes, the ray traversal algorithm with the best performance was the recursive ray traversal algorithm TA^B_{REC}. For scenes with the largest number of higher order rays it was outperformed by ray traversal algorithms with neighbor-links. The recursive ray traversal algorithm TA^A_{REC}, which suffers from a lack of robustness, and sequential ray traversal algorithm TA_{SEQ} were always slower than TA^B_{REC}.

Future research work could include the development of other ray traversal algorithms with a lower average cost of one traversal step and the smallest possible number of traversal steps. For individual rays, the number of traversal steps for the case of the recursive ray traversal algorithm and the algorithm with neighbor-links trees cannot be decreased. The limitation is that we have to visit each leaf along the ray path, and we have already reached the minimum number of traversal steps. It possibly remains to decrease further the cost of one traversal step, which need not only be an implementation issue, as we have shown by the design of the recursive traversal algorithm TA^B_{rec}.

A more promising topic for research is an algorithm that selects one from all implemented ray traversal algorithms for a particular ray shooting query to achieve the best possible performance. The selection algorithm for a particular ray shooting query has the input known or estimated properties of the input ray with respect to the scene configuration: is it probable that the ray intersect an object? What is the probable distance of an intersection point? Is the ray with internal or external origin? When we have a ray with internal origin, is it known which leaf of the *kd*-tree contains the origin of the ray? The research effort in this direction is intertwined with the applications for which the *RSA* based on the *kd*-tree is used.

Chapter 6

Longest Common Traversal Sequences for *Kd*-Trees

In this chapter we describe two methods utilizing spatial coherence for a particular set of ray shooting queries that decrease the average number of traversal steps per ray and thus the total running time consumed by the *RSA*. These methods use the concept of the longest common traversal sequence for the *kd*-tree introduced by Havran and Bittner [78].

6.1 Motivation

We described various ray traversal algorithms in great detail in the previous chapter. Until now we have assumed that solving each single ray shooting query is an individual task independent of solving other ray shooting queries. In global illumination algorithms it is often the case that set of rays have similar directions and origins. This occurs for primary and higher order rays spawned with the same point of origin, rays between two patches that are used to compute a form factor, *etc.* This raises the question whether it is possible to use such knowledge about the similarity of rays to further improve the efficiency of an *RSA*. The basic idea of such an improvement for *RSAs* based on a spatial subdivision is illustrated in Fig. 6.1. Rays lying within a certain convex shaft must pierce the same sequence of generic and elementary cells of a spatial subdivision. We call this phenomenon *traversal coherence*.

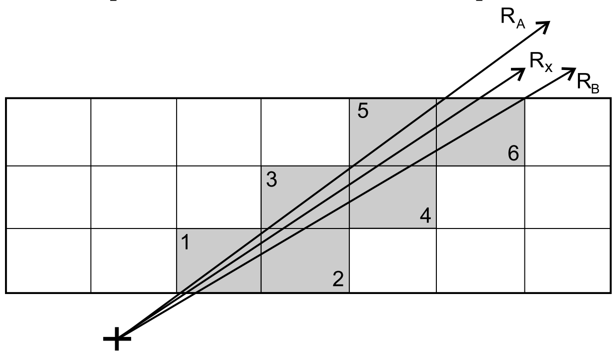


Figure 6.1: The concept of traversal coherence in two dimensions. An arbitrary ray R_X lying between rays R_A and R_B pierces the same sequence of elementary cells.

We describe two techniques utilizing the concept of traversal coherence. Our first technique determines a *longest common traversal sequence* for the *kd*-tree (further abbreviated to $LCTS$) for a given convex region (shaft) $\mathcal{CS}$ consisting solely of leaves of the *kd*-tree. We call the resulting $LCTS$ the *simple* $LCTS$ ($SLCTS$). The $SLCTS$ (if it exists) can be used for all rays contained within $\mathcal{CS}$. As will be shown later, if no intersected object is found using the current $SLCTS$, the ray traversal algorithm uses some conventional ray traversal algorithm, such as the one with neighbor-links presented in Subsubsection 5.3.3.3.

The second technique uses a more elaborate treatment of the information gained during traversing of the spatial hierarchy. It determines a *hierarchical LCTS* ($HLCTS$), which corresponds to a sequence $\mathcal{G}$ of nodes of the spatial hierarchy. These nodes form a cut of the hierarchy at the level where the order of traversed nodes can no longer be predetermined for all rays located within $\mathcal{CS}$. For any ray located in $\mathcal{CS}$ we can avoid traversal steps from the root of the hierarchy to the nodes in $\mathcal{G}$.

As we show later, the $LCTS$ concept can be further improved. One extension prunes adjacent empty elementary nodes of a spatial subdivision. Another extension determines a termination object (if it exists) that is hit by all rays located in $\mathcal{CS}$.

The rest of the chapter is organized as follows: In Section 6.2 we describe the previous work related to the approach presented here. In Section 6.3 we present the $LCTS$ construction algorithm in detail. Section 6.4 describes utilizing $LCTS$ for ray shooting between two patches and hidden surface removal based on ray casting. Section 6.5 presents results based on a practical implementation. Finally, Section 6.6 concludes the chapter with several possible directions for future research.

6.2 Previous Work

Several papers related to the $LCTS$ concept have been published. Concepts of generalized rays have been introduced; cone tracing [11], beam tracing [87], and pencil tracing [131]. Arvo and Kirk [17] presented a ray classification method which subdivides the five dimensional ray space. For each cell of this subdivision, a sorted list of objects is constructed and a ray is tested for intersection only with objects corresponding to the elementary cell that is intersected by the ray. Simiakakis and Day presented a technique that improves the space complexity of ray classification by adaptively subdividing the ray space [133]. The memory complexity of this approach was improved by Kwon *et al.* [102] by reducing the ray space from five to four dimensions. The ray coherence theorem [116] is a generalization of the light buffer [70] approach. It uses directionality of rays and a binary search. Haines and Wallace [71] utilized the concept of a *shaft*; the ray-object intersection tests are restricted to objects that intersect a shaft connecting two patches. Teller and Alex [155] subdivide the viewing frustum by combining Warnock's visibility algorithm and beam tracing. Similar use of coherence was presented by Gonzáles and Gisbert for an octree [64]. Pyramid clipping of a spatial subdivision aimed at a parallel implementation for a ray traversal algorithm was presented by van der Zwaan *et al.* [156]. The technique of directed safe zones utilizing free adjacent elementary cells within a uniform grid was published by Semwal [124]. Recently, Genetti *et al.* [55] presented an approach for adaptive supersampling in object space, using pyramidal rays. All the methods mentioned utilize some *coherence* concept, which was surveyed by Gröller [68].

The algorithm presented in this chapter is a combination of directional techniques with a hierarchical spatial subdivision, *i.e.*, with the *kd*-tree. The proposed combination takes advantages of both recursive and neighbor-links ray traversal algorithms of the *kd*-tree for a certain set of rays.

6.3 *LCTS* Construction

The $LCTS$ is constructed for a convex *shaft* defined by a set of *boundary rays*. Typically, these rays form the edges of a *frustum* (if they share the origin) or the edges of a *tunnel* (if the rays are parallel). For each of the boundary rays a *traversal history* is stored. This information is used to construct the $LCTS$ that is common to all rays belonging to the shaft. We distinguish between two types of $LCTS$. The first type – a simple $LCTS$ ($SLCTS$) exploits traversal coherence using only leaf nodes of the hierarchy. The second one – a hierarchical *LCTS* (*HLCTS*) also uses traversal coherence in the hierarchical nodes, but requires more computational effort to construct the traversal history.

6.3.1 *SLCTS*

The concept of *SLCTS* is depicted in Fig. 6.1 for a uniform grid. Similarly for *kd*-tree, we assume a convex shaft defined by several rays that traverse the same sequence $\mathcal{G}$ of elementary cells of a *kd*-tree. Then an arbitrary ray lying within the shaft traverses sequence $\mathcal{G}$ as well. The origin of the ray has to be positioned in the shaft. There are some potential problems to be solved for *SLCTS* as depicted in Fig. 6.2:

Case 1: No common sequence of leaf nodes exist (Fig. 6.2, case 1).

Case 2: Having some initial sequence of elementary cells $\mathcal{G}$ for the rays defining an *LCTS*, so the last common cell for them is known. If a ray does not hit any object in $\mathcal{G}$, which cells have to be traversed then? (Fig. 6.2, case 2).

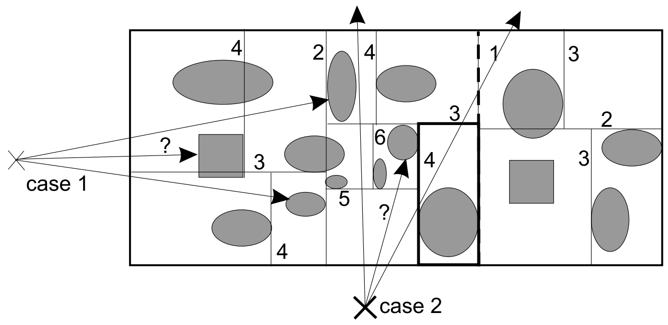


Figure 6.2: Two potential problems of *SLCTS* to be solved. The numbers mark the depth of the cutting planes in the *kd*-tree hierarchy.

We use a simple solution to the first problem; we apply any ray traversal algorithm for the *kd*-tree described in Chapter 5. The second problem can be solved for the *kd*-tree by using of a ray traversal algorithm with neighbor-links (see Subsection 5.3.3). When no object is intersected using sequence $\mathcal{G}$, then the ray traversal algorithm continues to the next leaf along the ray using either single neighbor-links or neighbor-links trees. The ray traversal algorithm with neighbor-links starts at the last cell of $\mathcal{G}$.

It is worth mentioning that the concept of *SLCTS* is applicable not only to the *kd*-tree, but to any spatial subdivision. Then it is advantageous if the ray traversal algorithm for the spatial subdivision can continue from some elementary cell (last cell of *SLCTS*) to the next cell along the ray without the down-traversal phase (*e.g.*, uniform grids).

6.3.2 *HLCTS*

The second proposed method uses the *HLCTS* and also exploits the traversal coherence of interior nodes of the *kd*-tree hierarchy. We describe the details of *HLCTS* construction and the corresponding ray traversal algorithm below.

6.3.2.1 Traversal Trees

A traversal history for a given ray can be stored by means of a *traversal tree*. The traversal tree is a binary tree, where each node τ of the traversal tree corresponds to a node ν in the *kd*-tree that was visited in the scope of the traversal. Additionally, the node contains information about traversing the node that reaches one of the following *traversal states*: LEFT, RIGHT, LEFT/RIGHT, RIGHT/LEFT, and TERMINATION. The traversal state TERMINATION corresponds either to pierced leaf-nodes of the *kd*-tree or interior nodes that were pushed on the traversal stack, but as the ray has been terminated, these nodes were not used for further traversal. Other traversal states express the order of the traversal of the *kd*-tree “below” node ν. Additionally, a node τ of the traversal tree contains a pointer to an *exit-plane*, which is a plane bounding the $\mathcal{AB}(\nu)$ associated with node ν along the ray path (see Fig. 6.3 (a)). The use of a pointer to the exit-plane will be described further.

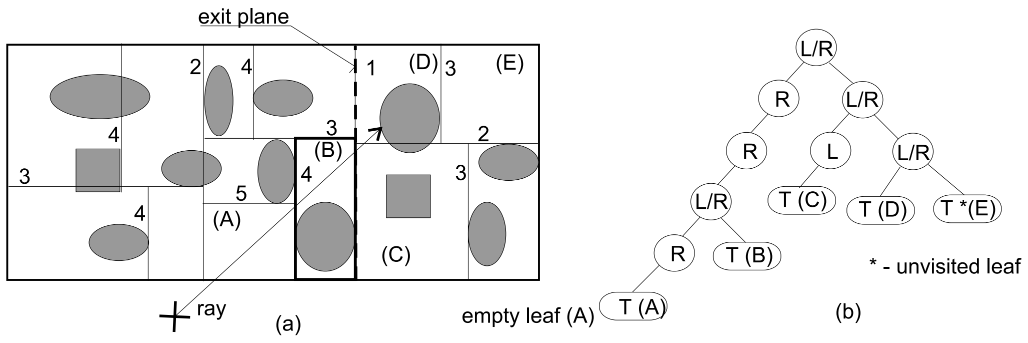


Figure 6.3: (a) Exit plane – the ray leaves the marked leaf node at the face in the exit plane that is formed by the cutting plane of the root node. (b) Traversal tree corresponding to the ray. Notation for nodes of traversal tree: L – LEFT, LR – LEFT/RIGHT, R – RIGHT, RL – RIGHT/LEFT, T – TERMINATION.

If node τ is not a leaf, then its left child contains a pointer to the *kd*-tree node ν_1 that was visited first during traversing. The right child of τ (if any) contains a pointer to the *kd*-tree node ν_2 pushed on the traversal stack and thus visited later or unvisited (if the ray has been terminated before reaching this node). See Fig. 6.3 (b), which depicts an example of a traversal tree structure.

6.3.2.2 Constructing Initial *HLCTS*

The *initial HLCTS* is constructed using n traversal trees ($n > 1$) determined for n boundary rays of a given convex shaft. Using convexity, it can be proved by contradiction that the traversal states for some nodes of the *kd*-tree are the same for all rays within the shaft, if the corresponding traversal states for all boundary rays are equal. Traversing of these nodes can be avoided by descending the hierarchy and constructing an ordered sequence of nodes where the traversal states for the boundary rays are no longer equal. The *HLCTS* can be seen as a cut on the *kd*-tree at the level where the traversal state can no longer be precomputed from the traversal histories of the boundary rays. Fig. 6.4 (a) depicts the boundary rays of a frustum.

The *HLCTS* construction algorithm performs a constrained depth-first-search (DFS) in parallel on all n traversal trees. If the traversal states associated with all n currently reached interior nodes are equal, the algorithm is applied recursively first on the left child and then on the right child (if any). If the reached nodes are leaves of the traversal trees (state=TERMINATION) or the traversal states are not equal, the *HLCTS* is enlarged using the *kd*-tree node associated with the reached nodes. Additionally, each *HLCTS* entry contains n pointers to the associated nodes of the traversal trees (see Fig. 6.4 (b)). Their use will be explained further in the text.

Once the initial *HLCTS* has been constructed, it can be used by the ray traversal algorithm to initiate the traversal for all rays within the corresponding shaft. The traversal stack can be filled using all nodes of the *HLCTS*. Note that ray traversal algorithms usually assume that the entry and exit points are known for the current node. Using an *HLCTS* these must be computed explicitly for each visited

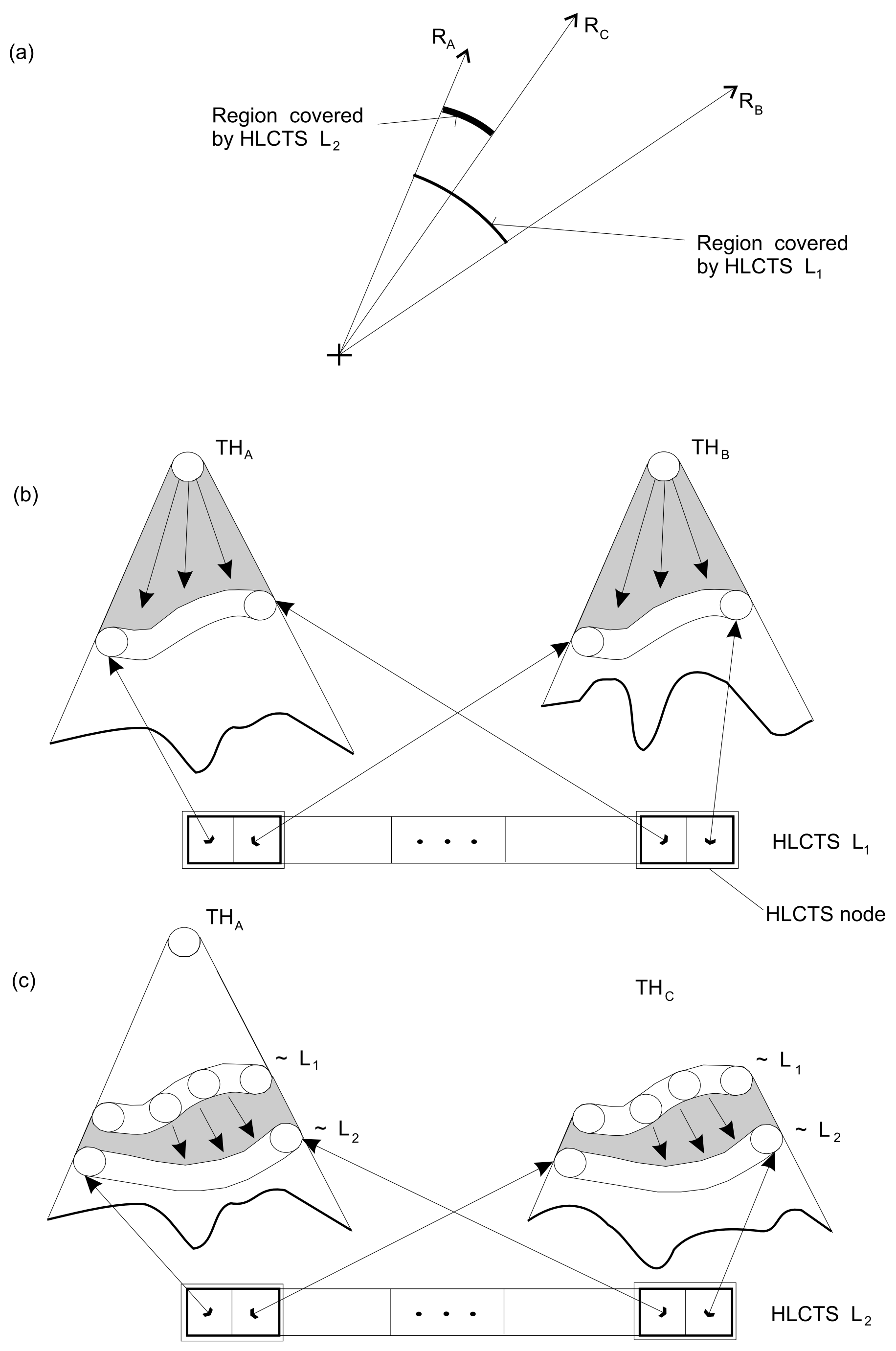


Figure 6.4: *HLCTS* construction and use: (a) Underlying geometry, two boundary rays R_A and R_B, the ray R_C between boundary rays, and the regions defined by the rays. (b) *HLCTS* L_1 generated from two traversal histories TH_A and TH_B corresponding to boundary rays R_A and R_B. (c) *HLCTS* L_2 generated from traversal histories of different number of roots, using TH_A and TH_C traversal histories and *HLCTS* L_1. TH_A is accessed using L_1, TH_C contains four root nodes generated from L_1. Notation: $\sim X$ – cut of the traversal history corresponding to *HLCTS* X. TH – traversal history; TH_A, TH_B for boundary rays, TH_C for a ray between boundary rays.

node of the *HLCTS*, since they have not been determined recursively as in the common ray traversal algorithm. The pointers to exit plane in nodes of traversal trees are used to solve this problem.

6.3.2.3 Constructing General *HLCTS*

The *HLCTS* for a given shaft can be refined further by constructing *HLCTS* for its sub-shafts. Nevertheless, if the ray traversal algorithm does not start at the root node of the *kd*-tree, the traversal history no longer corresponds to a single tree. Instead, the traversal history is stored as a sequence of traversal trees with their roots corresponding to the nodes of the *HLCTS* that were used to as initial nodes for traversing. We consider constructing *HLCTS* for the n traversal histories with the following properties:

Property 1: All traversal histories have been generated from the same *HLCTS* and thus they correspond to sequences of traversal trees of the same length.

Property 2: Only e traversal histories have been generated from the same *HLCTS*, say L. Other $n-e$ histories have been generated before, but they have been used to establish L.

The first case can be solved simply, by applying the previously mentioned algorithm on all n-tuples of the root nodes. If the algorithm is implemented using a stack, these n-tuples can be initially pushed on the stack in reverse order.

In the second case the information stored within L is used. Although $(n-e)$ "old" traversal histories correspond to sequences of traversal trees of length $l \le |\mathrm{L}|$, each entry of L contains pointers to the traversal tree nodes that correspond to this *HLCTS* entry. Using these pointers the "old" traversal histories can be accessed directly at the "level" corresponding to L. Thus the algorithm is applied on n-tuples of traversal history nodes, where e entries correspond to roots of the traversal trees generated using L. Other $(n-e)$ entries are determined using appropriate pointers stored within entries of L. The problem and its solution are illustrated in Fig. 6.4 (c).

6.3.3 Further Improvements

Below we present several improvements that can be used in the scope of the construction of both *SLCTS* and *HLCTS*.

6.3.3.1 Unification of Empty Leaves

Although the *kd*-tree is built adaptively with respect to the scene geometry, it can happen that the *LCTS* contains a subsequence $\mathcal{G}'$ of entries corresponding to empty leaves of the scene *kd*-tree. This is depicted in Fig. 6.5. The *LCTS* construction algorithm can be modified to detect this situation and to replace $\mathcal{G}'$ with a single entry. This approach is analogous to directed safe zones [124] that would be constructed on the fly.

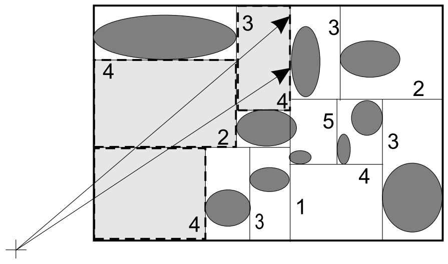


Figure 6.5: Unification of empty leaves. For two rays with common origin three empty leaves can be found.

The modification of the *LCTS* construction algorithm is straightforward. If a new *LCTS* entry X that corresponds to an empty leaf node is to be added, it is first checked whether the last entry Y of the constructed *LCTS* corresponds to an empty leaf of the *kd*-tree. If this is the case, the *LCTS* is not enlarged by X. Instead the "exit-plane-node" of X is used to replace the exit-plane-node of Y. In this way the spatial extent corresponding to Y is enlarged properly for all rays within the shaft. This is necessary for the correct behavior of the ray traversal algorithm.

6.3.3.2 Termination Object

A remarkable reduction of traversal steps for rays in a given convex shaft $\mathcal{CS}$ can be obtained by determining a single convex *termination object* that is hit by all rays in $\mathcal{CS}$. This is possible only if the origin of the rays is somehow restricted. This holds for example for a pyramidal shaft (frustum) where all rays originate at the same point.

If there is a termination object, traversing a *kd*-tree can be eliminated completely. Using *LCTS* we can perform a simple test that optionally determines the termination object for the *LCTS*. The presented approach is *conservative*, since it does not always determine the termination object even if it exists, but it never gives a wrong answer if no termination object exists for a given shaft. The termination object O exists if all of the following conditions hold:

Condition 1: All boundary rays of the shaft hit the same object O and are terminated in the same cell $\mathcal{V}$.

Condition 2: Object O is convex and it is the only object intersecting the cell $\mathcal{V}$.

Condition 3: The cells visited before reaching the cell $\mathcal{V}$ are empty.

If the unification of empty leaves described above is applied, the last condition reduces to one of the following cases:

Condition 3a: Cell $\mathcal{V}$ corresponds to the first entry of the *LCTS*.

Condition 3b: Cell $\mathcal{V}$ corresponds to the second entry and the first entry corresponds to an empty leaf (unification of empty leaves).

6.3.3.3 Initial Leaf Sequence for *HLCTS*

This improvement is applicable to *HLCTS* only. If the *HLCTS* corresponds to a pyramidal shaft (frustum), it can be expected that the first entries of the *HLCTS* correspond to leaves of the *kd*-tree. In such a case this *initial leaf sequence* of *HLCTS* forms an *SLCTS*.

With each *HLCTS* we keep a single value n that denotes the number of leaves at the beginning of the *HLCTS*. If $n = |HLCTS|$, the sequence corresponds to a sequence of leaf nodes of the *kd*-tree. In this case it forms an *SLCTS* and cannot be refined any longer.

The *HLCTS* construction algorithm can be modified to copy the first n leaf nodes from the "parental" sequence without performing any matching. The previously mentioned *HLCTS* construction algorithm is applied starting at the $(n+1)$-th node of the *HLCTS*. Similarly, index n can be exploited in the ray traversal algorithm where the first n nodes can be visited without using any traversal stack. If the traversal is terminated before reaching the $(n+1)$-th node, stack initialization is completely avoided.

6.4 Application of *LCTS*

Ray shooting with the *LCTS* concept can be used in many global illumination techniques (for a survey, see [149, 157]) based on discrete sampling of space via rays, *e.g.*, ray tracing, photon tracing, Monte Carlo methods, shadow determination, form factor computation. We discuss here two techniques that can be used in a more general way, namely patch-to-patch visibility and hidden surface removal.

6.4.1 Patch-to-patch Visibility

For the purposes of computing patch-to-patch visibility factors, the $HLCTS$ is more suitable, since it can occur that there is no $SLCTS$ for some two patches.

The task is to determine mutual visibility using ray shooting for a given two patches in the scene. We create convex hulls of both patches, then we determine a set of rays that form boundary rays of a convex shaft between these convex hulls. We construct the traversal tree for each ray in the set and then the corresponding $HLCTS$, which is subsequently used for *any ray* between the patches.

This application of $HLCTS$ is similar to the concept of shaft culling [71], but there are major differences in the way of obtaining the desired set of cells intersecting the shaft. In shaft culling, intersection tests between shafts and cells are performed, which can be costly. The $HLCTS$ technique uses only ray shooting and then only non-geometric computations for $HLCTS$ construction, which is less expensive. The results of applying these two techniques are not the same. Generally, $HLCTS$ determines a superset of cells determined by classical shaft culling, but in much more efficient way.

6.4.2 Hidden Surface Removal

Hidden surface removal that uses ray shooting is usually called *ray casting.* For this purpose, both $HLCTS$ and $SLCTS$ are suitable; a significant reduction of traversal steps can be achieved by using a termination object as mentioned in the previous chapter.

The common origin of rays (viewpoint) induces that the initial sequence of common nodes in $LCTS$ is likely to be a sequence of leaves ($SLCTS$). The more paraxial rays, the longer the initial $SLCTS$. Assuming the cells are farther from the viewpoint, the rays are less likely to generate the same sequence of *kd*-tree leaves. Thus the scene geometry, the *kd*-tree properties, and the image resolution influence the level of utilization of traversal coherence.

There are several ways how to exploit the $LCTS$ concept for hidden surface removal. We can deal with an image as a one-dimensional array of one-dimensional arrays (scanline approach, using traversal coherence in one dimension) or as a two-dimensional array (using traversal coherence in two dimensions). Since both $SLCTS$ and $HLCTS$ techniques can be applied, we have exactly four cases:

SLCTS-1D: Scanline with $SLCTS$ – this approach is basically undersampling on a scanline, creating $SLCTS$ for two adjacent samples and using this $SLCTS$ to compute samples between them. This scheme is depicted in Fig. 6.6 (a).

HLCTS-1D: Scanline with $HLCTS$ – this approach can be implemented as above, but a better utilization of traversal coherence combines $HLCTS$ with bisection. The initial $HLCTS$ is incrementally refined. The scheme is depicted in Fig. 6.6 (b).

SLCTS-2D: Two-dimensions with $SLCTS$ – the sampling can be performed as undersampling $n_x \times n_y$ pixels. Having the sequence of traversed leaves for the four corners of a rectangle, the $SLCTS$ is created and used for all the rays inside the rectangle. The $SLCTS$-1D can be seen as special case for $n_y = 1$. The scheme is depicted in Fig. 6.6 (c).

HLCTS-2D: Two-dimensions with $HLCTS$ – the bisection is applied in two dimensions; the axis for splitting is regularly changed. At the beginning four rays corresponding to the pixels in the image corners are cast. In one bisection step, four rays are cast again and two new rectangles are created. See Fig. 6.6 (d).

We should point out that all these sampling schemes can be applied more successfully when the image resolution is high with respect to the space subdivision projected to the image plane. In this case, many areas in the image have a common traversal sequence, which often form $SLCTS$. An example of such a projection is depicted in Fig. 4.17.

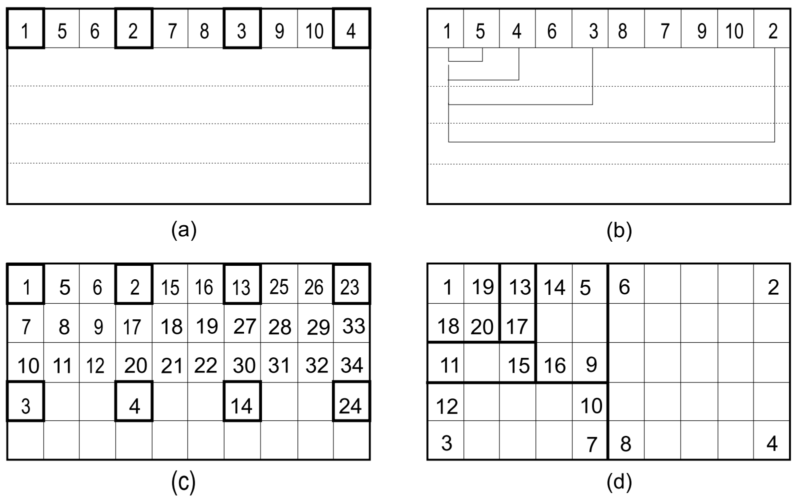


Figure 6.6: Hidden surface removal sampling patterns for *LCTS*: (a) *SLCTS*-1D (b) *HLCTS*-1D (c) *SLCTS*-2D (d) *HLCTS*-2D. The numbers mark the order in which the rays are cast.

Note that *SLCTS* can also be used with bisection, but since the rays corresponding to corner pixels are far from being paraxial, they do not generate any *SLCTS*. We have verified experimentally that the undersampling method as described here is more efficient for the *SLCTS* concept.

6.5 Results

We implemented all the sampling techniques described above for hidden surface removal and the *HLCTS* approach for ray shooting between two patches. For testing, we again used the *SPD* scenes [69], but we report here only a subset of G^4_{SPD} scenes (8 out of 10 scenes) due to a lack of space. The *kd*-trees for all G^4_{SPD} scenes were built using the ordinary surface area heuristic with late cutting off empty space (see Chapter 4) for ad hoc termination criteria with this setting: $d_{max} = 16$ and $N_{max} = 2$.

The basic scene properties and the number of leaves of the constructed *kd*-trees are listed in Table 6.1. More data on the scenes, describing their scene complexities, were presented in Subsection 3.5.1.

	Scene							
	balls4	gears4	lattice12	mount6	rings7	teapot12	tetra6	tree11
objects	7382	9345	8281	8196	8392	9264	4096	8191
spheres	7381	9345	2197	4	4195	-	-	4095
polygons	1	-	-	8192	1	9264	4096	1
cones	-	-	-	-	1	-	-	4095
cylinders	-	-	6084	-	4195	-	-	-
kd-tree leaves	5253	13830	25689	8554	12924	2387	2972	3426

Table 6.1: Properties of testing scenes and *kd*-trees built for experiments.

6.5.1 Patch-to-patch Visibility

We have observed that it is difficult to predict whether *HLCTS* construction pays off for patch-to-patch visibility. If only a low number of rays between the patches is cast, then the time required for *HLCTS* construction is not recovered later. For tens and hundreds of rays between the patches, *HLCTS* construction can be worthwhile, particularly when the patches are incrementally refined as occurs in hierarchical radiosity algorithms. The reduction in the number of traversal steps depends on the shape and the positioning of the constructed shaft in the scene and the configuration of the objects in the scene. The more elongated the shaft, and the deeper the *kd*-tree, the better improvement of efficiency can be achieved.

We do not provide here any quantitative results for two reasons, since we found that the results achieved experimentally vary greatly, depending on the above-mentioned conditions. First, the *SPD* scenes are not composed of planar primitives – so they are inconvenient for testing patch-to-patch visibility. Second, we are not aware of any standard and well described algorithm that provides a set of pairs of patches for a given scene in a similar way as hidden surface removal provides a set of rays, as described below.

6.5.2 Hidden Surface Removal

We tested hidden surface removal using ray shooting with the recursive ray traversal algorithm TA^{B}_{rec}, the ray traversal algorithm with neighbor-links trees TA_{NLT}, and *LCTS* traversal algorithms *SLCTS*-1D, *SLCTS*-2D, *HLCTS*-1D, and *HLCTS*-2D. Since applying any *LCTS* technique influences only the number of traversal steps, the average cost of one traversal step and thus the whole running time (*i.e.*, $\tilde{N}_{TS}$, $\tilde{N}_{ETS}$, $\tilde{N}_{EETS}$ of subset Δ and T_R, Θ_{rat}, and Θ_{RUN} of subset Θ of minimum testing output, see Chapter 2), we present here only a subset of the parameters, which nevertheless enables us to evaluate the results given by *LCTS* techniques.

The ratio of the number of intersection tests to minimum intersection tests r_{ITM} and the average number of traversal steps $\tilde{N}_{TS}$ for primary rays for 1024×1024 image resolution, and for all the traversal methods, is shown in Table 6.2. The recursive ray traversal algorithm TA^{B}_{rec} was used as a reference for comparison. The running times in the two tables include the construction of *LCTSs*, which are built on the fly, so no additional preprocessing is performed. Table 6.3 shows the sensitivity of the different ray traversal algorithms to the resolution of the image for the scene “teapot12”.

Note that for these tests on hidden surface removal the ray traversal algorithm with neighbor-links trees TA_{NLT} is not efficient compared to a recursive ray traversal algorithm. The first reason is that the cost of one traversal step for TA_{NLT} is slightly higher than for the recursive ray traversal algorithm. The second reason is that for hidden surface removal for the tested scenes, rays are cast from outside the scene, so many empty leaves have to be traversed before hitting an object.

Fig. 6.7 visualizes the traversal coherence for the scene “mount6” and the method *SLCTS*-2D. It is obvious that most pixels in the projection have at least one common initial leaf node.

All the experiments were conducted on the SGI O^2, MIPS R10000, 180 MHz, 256 MBytes RAM, running the *Irix 6.3* operating system. All the ray traversal algorithms tested were implemented within the GOLEM rendering system [75].

6.5.3 Discussion

Successful use of the *LCTS* concept for patch-to-patch visibility is conditioned by the number of rays shot between the patches. Similarly, the application of *LCTS* for hidden surface removal via ray casting depends on image resolution.

Let us discuss in detail the properties of hidden surface removal using ray shooting with *LCTS*. The performance improvement is scene dependent, but this is the case for all heuristic *RSAs*. It follows from the results that hierarchical traversal steps to the first leaf are successfully avoided, and the total number of traversal steps is decreased typically by more than 60% (for scene “lattice12” as much as 78%). This corresponds to an improvement in performance by 20% on average, since most of the computation is then devoted to ray-object intersections. (The total running time T_R does not include the remaining application time, the ratio of the time of *RSA* to the time of the whole ray tracing is scene dependent, and reaches from 40% to 75% for the used test scenes, see [86] and the results in Appendix E.)

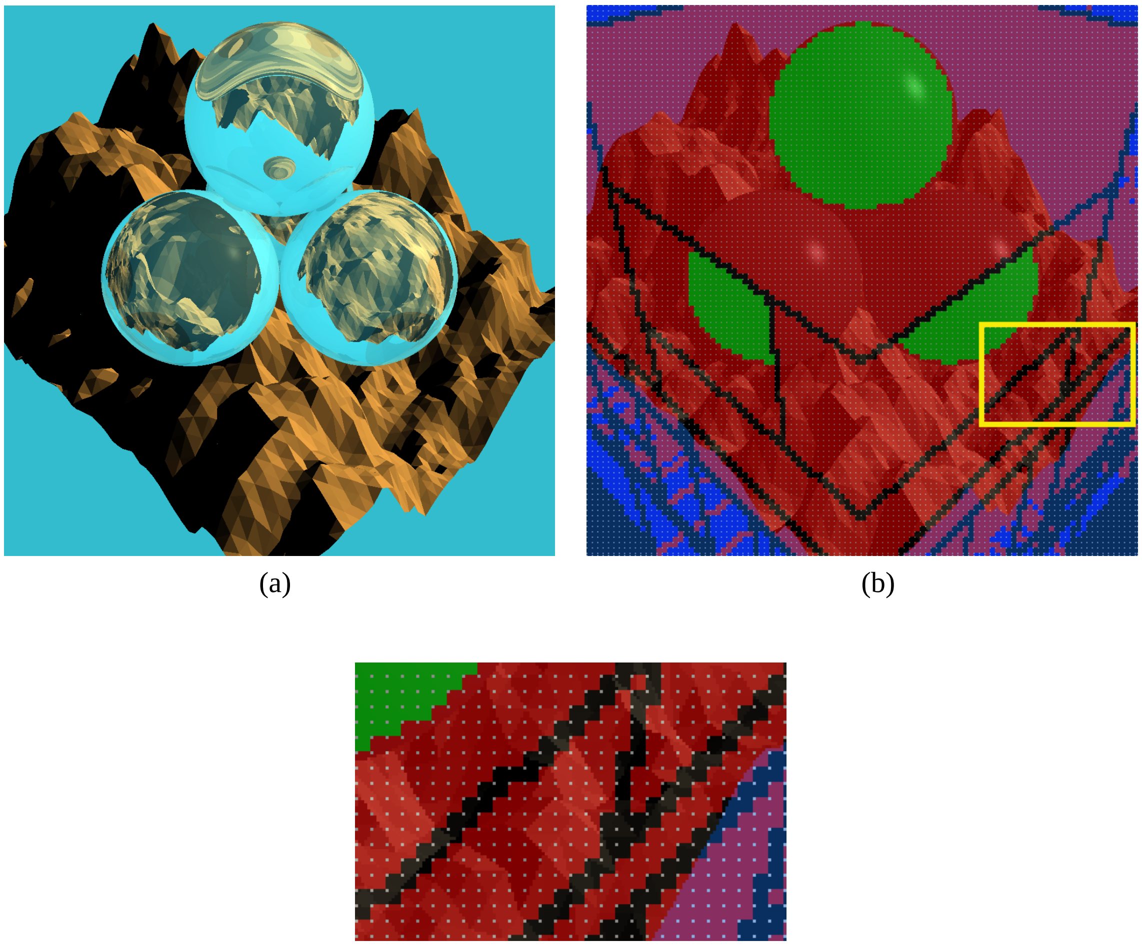

(a) (b) (c)

Figure 6.7: Visualization of the traversal coherence of *SLCTS*-2D for the scene “mount6” (a) Normal ray tracing. (b) Blended with the color for pixels: *white* – sampling pixels, *red* – pixels for which there exists a common traversal sequence with at least one leaf, *green* – pixels for which the terminating object was found, *blue* – pixels for which it is known that no object can be hit, *black* – pixels for which no *LCTS* was found. (c) A zoomed in part of image (b), where the sampling pattern is better visible.

Note that the recursive ray traversal algorithm T^B_{rec} (see Section 5.4) for an arbitrary ray used as a reference is highly optimized for SGI architecture. This means that the improving the performance of the recursive ray traversal algorithm by another 20% on average using the *LCTS* concept is significant.

Special attention should be given to the setting of the undersampling resolution for *SLCTS* approaches. We can observe a tradeoff between undersampling resolution and the possible existence of *SLCTS* or/and its properties. If we take fewer samples for whole *SLCTS*, there is a lower probability of reduction of traversal steps for the constructed *SLCTS*. The number of samples needed to construct one *SLCTS* was always constant, either two (*SLCTS*-1D) or four (*SLCTS*-2D). Let us take pixels on a scanline and construct *SLCTS*-1D for two pixels. Let n_x be the distance between the two pixels. We can then use the constructed *SLCTS* for $n' = n_x - 2$ pixels if such an *SLCTS* exists. The undersampling resolution $n_x = 5$ pixels is a reasonable compromise. The same holds for *SLCTS*-2D, when we set the undersampling resolution to 5×5 pixels. In general, *SLCTS*-2D enables us to use *SLCTS* (if *SLCTS* exists) for $n'' = n_x.n_y - 4 - (n_y - 2) - (n_x - 2) = n_x.n_y - n_x - n_y$ pixels. For high resolution images, n_x (and n_y for SLCTS-2D) can be set to an even higher value.

We can see that two-dimensional *LCTS* methods are better able to exploit coherence properties than one-dimensional *LCTS* methods. The *HLCTS* performs more efficiently than *SLCTS* with sampling 5×5 only for high resolution images, mainly because the cost of *HLCTS* construction is higher than for *SLCTS* construction, and thus it is recovered only for larger regions of the image, where *HLCTS* forms *SLCTS*.

6.6 Conclusion and Future Work

In this chapter we have studied a new way of exploiting coherence in ray shooting algorithms based on the *kd*-tree for certain ray sets that induce some similarity between rays. We tried to avoid as many hierarchical traversal steps within the spatial hierarchy as possible, and at the same time to preserve all the advantages of using the hierarchy. We introduced two concepts of longest common traversal sequence; *SLCTS* and *HLCTS*. The presented techniques decrease the number of traversal steps for hidden surface removal based on ray casting, typically by more than 60%. For high resolution images the reduction of traversal steps is even more remarkable.

There are several possible topics for future research based on the *LCTS* concept. The application of *LCTS* could be studied for higher order rays, in context of particular global illumination algorithms. A *LCTS* can be applied if the rays of the first order (primary rays) have the same termination object and create a shaft that is then completely reflected or refracted. In addition, image space sampling patterns suitable for *LCTS* application other than those studied here should be investigated. The automatic setting of the *SLCTS* undersampling resolution based on scene properties and the use of *LCTS* in rendering animation sequences are also possible topics for future research. A further research topic could be an algorithm determining whether it is efficient to construct an *LCTS* given two patches in the scene for some application such as form factor computation.

	Scene							
Parameter	balls4	gears4	lattice12	mount6	rings7	teapot12	tetra6	tree11
r_{ITM}	7.40	4.37	8.81	5.29	10.9	4.82	7.59	10.1
	Ray traversal algorithm TA^B_{rec}							
$\tilde{N}^{1024}_{TS}[-]$	30.4	16.6	49.9	30.7	41.3	22.2	13.7	23.2
$(T_R - T_{app})^{1024}[s]$	15.4	48.1	25.6	12.9	28.6	11.5	7.46	16.1
$\alpha^{1024}[-]$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
$\eta^{1024}[-]$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
$\eta^{4096}[-]$	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
	Ray traversal algorithm TA_{NLT}							
$\tilde{N}^{1024}_{TS}[-]$	26.2	15.6	19.1	26.8	34.2	19.0	11.6	16.9
$(T_R - T_{app})^{1024}[s]$	16.0	48.2	23.5	13.4	31.9	12.2	8.65	16.7
$\alpha^{1024}[-]$	0.862	0.940	0.383	0.873	0.828	0.856	0.847	0.728
$\eta^{1024}[-]$	1.04	1.00	0.918	1.03	1.12	1.06	1.16	1.03
$\eta^{4096}[-]$	1.02	1.00	0.913	1.02	1.01	1.06	1.15	1.02
	SLCTS-1D ray traversal algorithm, window 5×1 pixels							
$\tilde{N}^{1024}_{TS}[-]$	13.2	7.13	11.6	11.9	15.7	8.46	5.70	9.83
$(T_R - T_{app})^{1024}[s]$	14.7	47.7	20.8	11.0	28.6	10.6	8.32	15.9
$\alpha^{1024}[-]$	0.434	0.430	0.234	0.388	0.380	0.381	0.416	0.424
$\eta^{1024}[-]$	0.958	0.990	0.812	0.853	0.999	0.925	1.16	0.988
$\eta^{4096}[-]$	0.846	0.977	0.785	0.787	0.947	0.871	0.987	0.947
$\gamma^{1024}[-]$	0.964	0.871	0.999	0.938	0.981	0.833	0.560	0.999
	HLCTS-1D ray traversal algorithm							
$\tilde{N}^{1024}_{TS}[-]$	10.1	5.58	13.8	8.52	11.0	6.35	4.65	7.66
$(T_R - T_{app})^{1024}[s]$	14.7	48.1	22.3	11.8	27.5	10.6	7.39	15.6
$\alpha^{1024}[-]$	0.332	0.336	0.277	0.278	0.266	0.286	0.339	0.33
$\eta^{1024}[-]$	0.958	0.998	0.872	0.913	0.960	0.929	0.990	0.970
$\eta^{4096}[-]$	0.820	0.972	0.775	0.752	0.868	0.816	0.851	0.895
	SLCTS-2D ray traversal algorithm, window 5×5 pixels							
$\tilde{N}^{1024}_{TS}[-]$	12.4	6.46	10.8	10.7	13.8	7.73	5.45	9.13
$(T_R - T_{app})^{1024}[s]$	12.6	46.7	19.7	9.99	27.2	9.69	7.24	15.1
$\alpha^{1024}[-]$	0.408	0.389	0.216	0.349	0.334	0.348	0.398	0.394
η^{1024}[-]	0.819	0.971	0.768	0.772	0.949	0.846	0.97	0.939
$\eta^{4096}[-]$	0.749	0.956	0.730	0.691	0.890	0.796	0.892	0.896
$\gamma^{1024}[-]$	0.919	0.780	0.999	0.916	0.971	0.798	0.525	0.990
	HLCTS-2D ray traversal algorithm							
$\tilde{N}^{1024}_{TS}[-]$	11.3	6.37	14.8	9.35	12.7	7.31	5.32	8.61
$(T_R - T_{app})^{1024}[s]$	13.6	47.5	21.7	10.4	26.4	9.60	6.31	14.6
$\alpha^{1024}[-]$	0.372	0.384	0.297	0.305	0.308	0.329	0.388	0.371
η^{1024}[-]	0.887	0.987	0.846	0.805	0.921	0.838	0.846	0.903
η^{4096}[-]	0.800	0.964	0.782	0.691	0.857	0.751	0.719	0.855

Table 6.2: Comparison of ray traversal algorithms for hidden surface removal based on ray casting. r_{ITM} – number of ray-object intersection tests to minimum number of ray-object intersection tests. $\tilde{N}^{1024}_{TS}$ – number of traversal steps per ray on average for a specific ray traversal algorithm and resolution 1024×1024. $(T_R - T_{app})^{1024}$ – the running time for ray shooting only ($= const.\Theta_{RUN}$) for a specific traversal algorithm for resolution 1024×1024. α^{1024} – the ratio between the number of traversal steps for a specific ray traversal algorithm and number of traversal steps of TA^B_{rec} for resolution 1024×1024. η^{1024} – the ratio between the running time of the specific traversal algorithm and the running time of TA^B_{rec} for resolution 1024×1024. η^{4096} – the ratio between the running time of a specific ray traversal algorithm and the running time of TA^B_{rec} for resolution 4096×4096. γ^{1024} – the ratio of the number of *SLCTS* sequences that contain at least one leaf to the number of all possible sequences for resolution 1024×1024.

	Resolution				
Parameter	256×256	512×512	1024×1024	2048×2048	4096×4096
	ray traversal algorithm TA^B_{rec}				
$\tilde{N}_{TS}[-]$	22.1	22.2	22.2	22.2	22.2
$(T_R - T_{app})'[s]$	0.734	2.89	11.5	45.6	182
$\alpha[-]$	1.00	1.00	1.00	1.00	1.00
$\eta[-]$	1.00	1.00	1.00	1.00	1.00
	ray traversal algorithm TA_{NLT}				
$\tilde{N}_{TS}[-]$	18.9	19.0	19.0	19.0	19.0
$(T_R - T_{app})'[s]$	0.792	3.08	12.2	48.4	193
$\alpha[-]$	0.855	0.856	0.856	0.856	0.856
$\eta[-]$	1.080	1.066	1.064	1.060	1.057
	SLCTS-1D traversal algorithm, window 5×1 pixels				
$\tilde{N}_{TS}[-]$	11.4	9.87	8.46	7.31	6.41
$(T_R - T_{app})'[s]$	0.741	2.77	10.6	40.6	159
$\alpha[-]$	0.516	0.445	0.381	0.329	0.289
$\eta[-]$	1.010	0.959	0.924	0.889	0.871
	HLCTS-1D traversal algorithm				
$\tilde{N}_{TS}[-]$	10.7	8.30	6.35	4.91	3.94
$(T_R - T_{app})'[s]$	0.782	2.87	10.6	39.6	149
$\alpha[-]$	0.484	0.374	0.286	0.221	0.177
$\eta[-]$	1.065	0.990	0.929	0.868	0.815
	SLCTS-2D traversal algorithm, window 5×5 pixels				
$\tilde{N}_{TS}[-]$	11.6	9.65	7.73	6.09	4.75
$(T_R - T_{app})'[s]$	0.683	2.55	9.69	36.9	145
$\alpha[-]$	0.525	0.435	0.348	0.274	0.213
$\eta[-]$	0.930	0.883	0.846	0.809	0.796
	HLCTS-2D traversal algorithm				
$\tilde{N}_{TS}[-]$	12.4	9.70	7.31	5.43	4.09
$(T_R - T_{app})'[s]$	0.711	2.60	9.60	35.8	137
$\alpha[-]$	0.561	0.437	0.329	0.244	0.184
$\eta[-]$	0.969	0.90	0.838	0.784	0.751

Table 6.3: Comparison of ray traversal algorithms for primary rays for the scene “teapot12” at different resolutions. $\tilde{N}_{TS}$ – number of traversal steps per ray for a particular ray traversal algorithm. $(T_R - T_{app})'[s]$ – the running time for ray shooting only ($= const.\Theta_{RUN}$) for a specific ray traversal algorithm. α – the ratio between the number of traversal steps for a specific traversal algorithm and the number of traversal steps of TA^B_{rec}. η – the ratio between the running time of the specific ray traversal algorithm and the running time of TA^B_{rec}.

Chapter 7

Memory Mapping of *Kd*-Trees

In this chapter we deal with *kd*-tree representation in the main memory of a computer. We show that for computer architectures with large cache line size the way of mapping nodes of the *kd*-tree in the main memory influences the time needed to transfer data from the main memory to a processor, hence the cost of one traversal step of ray traversal algorithms. We describe and analyze a few ways of mapping *kd*-tree nodes to memory, and also provide the results obtained from experiments.

7.1 Motivation

The most important operation carried out for a *kd*-tree in any application is exhaustive traversal of interior nodes and leaves; for example, the traversal in *depth-first-search* (DFS) order or the ray traversal algorithm described in detail in Chapter 5.

Input/output efficient algorithms and data structures for memory hierarchies have acquired noticeable research interest [31, 119, 120] in last decade. The design of these data structures is driven by the properties of external/internal memory hierarchy. For example, Nyberg *et al.* [115] described an algorithm for sorting which takes into account the memory hierarchy.

Unlike the special algorithms for external memory data structures, in this chapter we deal with the internal memory hierarchy between the processor and the main memory, including either on-chip cache or second-level cache. The main difference between this hierarchy and that for external memory is its size and the access time to one data block. The values for the external memory hierarchy are much larger than those between the processor cache and the main memory.

Moreover, techniques for external memory data structures were developed mostly for one-dimensional search problems. For example, the well known *B*-tree [34] cannot be used to decrease the time complexity of the *kd*-tree traversal for $n > 1$, since the *B*-tree cannot represent the n-dimensional data that are the subject of the *kd*-tree. We analyze novel methods to increase the spatial locality of data in the cache and thus to decrease the running time of any algorithm that works over the *kd*-tree. For the sake of simplicity of theoretical analysis we will assume that we traverse the *kd*-tree in DFS order from the root node to a leaf.

7.2 Preliminaries

In this section we recall a few technical facts necessary to understand the concept of mapping *kd*-tree nodes to memory. This includes memory allocation techniques and the structure of the memory hierarchy in a computer.

7.2.1 Memory Allocation

The basic topic of this chapter is mapping *kd*-tree nodes to addresses in the main memory. The main memory of a computer is principally a one-dimensional array of memory cells where data are stored. The allocation and deallocation of dynamic variables in the main memory is always provided by procedures of software library that is usually called the *memory allocator*. Let us suppose the contiguous block of the unoccupied memory is assigned to the memory allocator at the beginning of some program that builds up a *kd*-tree. This memory block is used to assign the addresses within the block to the variables allocated, so the variables do not overlap. We call this memory block a *memory pool*. Since mapping of variables into memory by a memory allocator is crucial for an understanding of this chapter, we discuss it here in detail.

A common solution for allocating variables is to use a *general memory allocator*. Each node of the *kd*-tree is then represented as a specially allocated variable. Let M_I denote the size of memory to store information in a node. This is the position and the orientation of the splitting plane. Let M_P be the size of a pointer. Then the size required to represent one interior node of the *kd*-tree is $M_{IN} = M_I + 2.M_P$. Using the general memory allocator requires that we store with each allocated variable two additional pointers that are required later to free the variable from the memory pool.

We can also use another strategy to allocate memory for a node of the *kd*-tree. We use a *fixed-size memory allocator*, described for example in [141], to allocate variables of the same type and thus of the same and fixed size M_V. We can then dedicate a special memory pool to allocate the interior nodes of the *kd*-tree, since these nodes are of the same fixed size. During *kd*-tree construction the nodes are allocated from the memory pool as from an array in linear order.

7.2.2 Memory Hierarchy

The time complexity of a ray traversal algorithm performed on the *kd*-tree is connected with the hardware used. The cost of the ray traversal step C_{TS} includes the time needed to transfer the data from a main memory to a processor. Let us recall the organization and the properties of the memory hierarchy. For analysis we suppose *Harvard* architecture with separate caches for instructions and data. Let T_{MM} denote the *latency* of the main memory (time to read/write one data block).

The larger the memory and the smaller the access time, the higher the cost of the memory. The instruction/data latency of processors is significantly smaller than T_{MM}. That is why a *cache* is placed between the memory and the processor. The cache is a memory of relatively small size with respect to the size of the main memory. The *cache latency* T_C is smaller than T_{MM}. This solution is economically advantageous; it uses the *temporal* and *spatial locality* of data exposed by a typical program, and the average access time to the data in the main memory can then be significantly reduced. Data between the cache and the main memory are transferred in blocks that correspond to the architecture of a cache memory, and each time only one block is transferred. The size of the block is referred to as the *cache line size* M_{CL}. A typical memory hierarchy is depicted in Fig. 7.1.

We use for analysis here only one cache placed between the processor and the main memory. To quantify the running time we denote the time consumed by operations in terms of processor *cycles*. Let T_W denote the average processing time on a node of the *kd*-tree to decide whether to follow its left or right descendant.

Typical values for superscalar processors and for a typical application using *kd*-trees are $T_{MM} = 55, T_C = 4, T_W = 5, M_{CL} = 128$ Bytes for MIPS R8000 [130]. These values are used again in the numerical examples for formulas further in the text. Note that for a typical searching algorithm on the *kd*-tree holds $T_W \ll T_{MM}$.

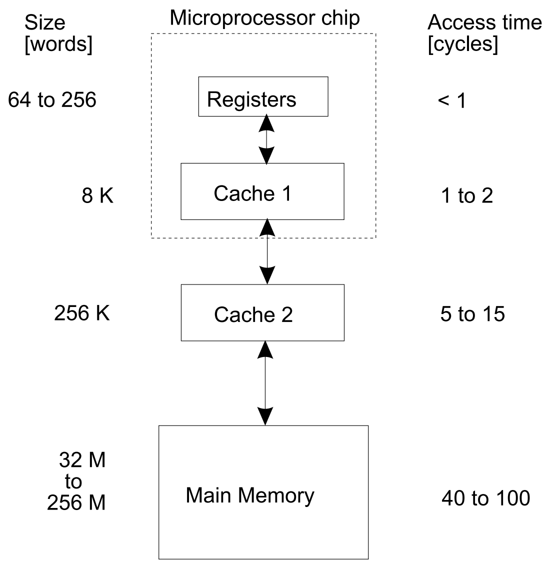


Figure 7.1: Typical memory hierarchy.

7.3 Representations of the *kd*-tree

As we have already stated, the *kd*-tree is actually represented by a binary tree. In general, a binary tree with an arbitrary specified splitting plane inside the interior nodes does not represent a valid instance of the *kd*-tree, since each splitting plane has to intersect the $\mathcal{AB}$ associated with the corresponding node. This is one reason why the decomposition induced by the *kd*-tree cannot be simply replaced by *B*-tree or some hashing scheme commonly used for one-dimensional search problems. The information stored in the interior node of the *kd*-tree is the orientation and the position of the splitting plane. The leaf contains just the pointer to the list of objects required by all ray traversal algorithms, and additional data for some ray traversal algorithms (see Chapter 5 for details). Here we suppose the $\mathcal{AB}$ is known explicitly for a root node only, also that the $\mathcal{AB}s$ associated with the interior and leaf nodes are not stored explicitly in these nodes. This *kd*-tree can be used in the recursive ray traversal algorithm.

Under the assumptions stated above, the analysis performed below disregards the *n*-dimensionality of the *kd*-tree, so we consider a *kd*-tree as a binary tree. Let us recall some terminology concerning binary trees needed for the analysis performed here. We call a binary tree *complete* if all its leaves are positioned at the same depth d from the root node and thus the number of leaves is 2^d. We call a binary tree *incomplete* when the binary tree is not complete. Let h_C define the *complete height* of a binary tree X as the maximum depth for which the binary tree constructed by the nodes of X is complete.

Below we describe in detail four representations of the *kd*-tree in memory. This includes a common method for representing *kd*-tree nodes using a general memory allocator. We call this *random* representation. A less known and less used method is *DFS order* representation. (The use of DFS order representation can be unintentional.) Finally, we describe two forms of a *subtree* representation that decrease further the average cost of one traversal step of the *kd*-tree.

7.3.1 Random Representation

A common way to store an arbitrary *kd*-tree in the main memory is to represent each node as a special variable using a general memory allocator. The representation is depicted in Fig. 7.2 (a).

This representation requires additional memory for the pointers used by the general memory allocator for each allocated variable, but it is the simplest technique to implement. The addresses of the nodes in the main memory have no connection with the location of the nodes in the *kd*-tree. Assume that two

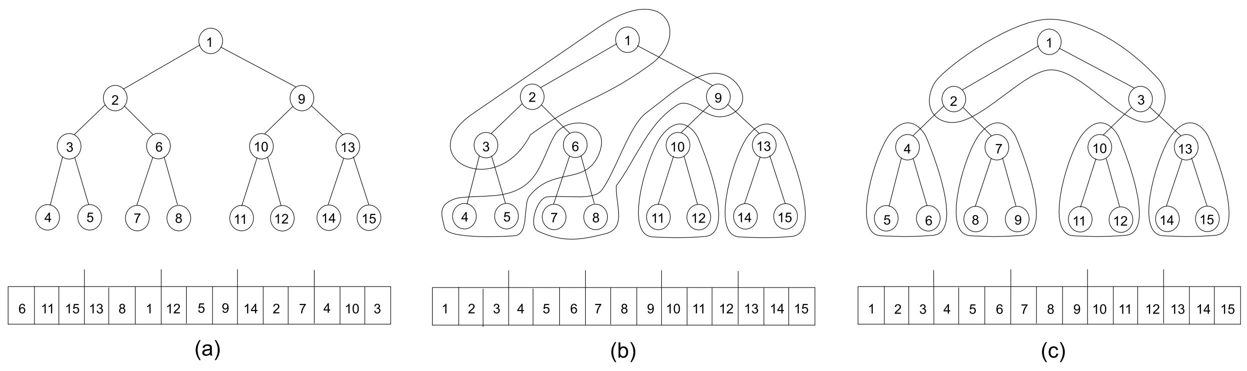


Figure 7.2: *Kd*-tree representations (cache line size $M_{CL} = 3.sizeof(kd{-}tree\ node)$ (a) Random (b) DFS (c) Subtree.

additional pointers are needed to allocate the variable, since the general memory allocator requires the pointers to deallocate the variable. Then the memory size M_S consumed by random representation to store n_{NO} nodes of *kd*-tree is:

$$M_S^{random} = N_{NO}.(4.M_P + M_I) \tag{7.1}$$

7.3.2 Depth-First-Search (DFS) Representation

A DFS representation uses a fixed-size memory allocator, which was described above. In this representation the nodes are put subsequently in the memory pool in linear order, when a *kd*-tree is built up in the DFS order, see Fig. 7.2 (b). The size of the memory consumed to represent n_{NO} nodes of the *kd*-tree is:

$$M_S^{DFS} = N_{NO}.(2.M_P + M_I) \tag{7.2}$$

Then $2.N_{NO}.M_P$ of memory is saved compared with random representation, since the two pointers would have been required by the general memory allocator.

7.3.3 Subtree Representation

Here we describe a new type of mapping *kd*-tree nodes to memory, originally introduced by Havran [72] and further elaborated in [73]. This *kd*-tree representation reduces the cost of a traversal step in ray traversal algorithms performed on the *kd*-tree. Let us describe the representation in detail.

For subtree representation we also use a fixed-size memory allocator, similarly to DFS representation, but the size of one allocated variable is equal to cache line size M_{SL}. The variable of size M_{SL} is by an explicit algorithm subsequently occupied by the nodes of the *kd*-tree organized into subtrees. All nodes of the *kd*-tree are then organized to these subtrees, see Fig. 7.2 (c). Once the subtree is read to the cache from the main memory within traversing, the access time to the nodes of the subtree is equal to cache latency T_C. The subtree need not be complete. We distinguish between two subtree representations, see Fig. 7.3.

An *ordinary subtree* has all nodes of the same size, with two pointers to its descendants, regardless of whether the descendant lies in the subtree.

A *compact subtree* has no pointers among the nodes inside the subtree, because their addressing is provided explicitly by a traversal algorithm. The pointers are needed only to point between the subtrees. Then the leaves of an incomplete subtree are marked in a special variable stored in each subtree (one bit for each node in the subtree).

The size of the memory taken by the two subtree representations is given in the next section.

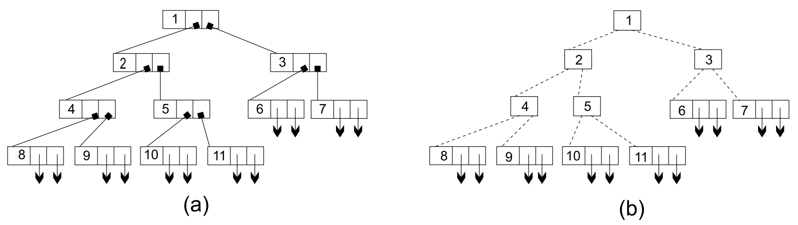


Figure 7.3: Subtree representation: (a) Ordinary (b) Compact.

7.4 Time Complexity and Cache Hit Ratio Analysis

Here we analyze the time complexity of a DFS order traversal for all *kd*-tree representations. The theoretical analysis assumes that the *kd*-tree nodes data stored in the main memory are not loaded into the cache, *i.e.*, the cache hit ratio $r_{CH} = 0.0$. Further, we suppose that the *kd*-tree is complete and its height is h_l. An incomplete *kd*-tree requires that we compute its average height $\overline{h_l}$ and substitute it for h_l.

These simplifications enable us to express the average traversal time T_A on the *kd*-tree in DFS order from its root to a leaf. We compute the T_A for an example of a *kd*-tree of height $h_l = 23$. Further, we suppose random visiting of the nodes and the probability $p_L = 0.5$ that we turn left in a node. The way of traversing nodes corresponds to the sequential ray traversal algorithm that performs the point-location search.

If some data are already located in the cache ($r_{CH} > 0.0$), an analysis using known mathematical tools can be very difficult or even infeasible [13]. Therefore we investigated the case by means of simulation.

7.4.1 Random Representation

We assume a cache hit ratio $r_{CH} = 0.0$ during the whole traversal, *i.e.*, the processing time of each *kd*-tree node is $T_{MM} + T_W$. As we know that the number of nodes along the traversal path from the root to the leaf is $h_l + 1$, we can express the average traversal time T_A as follows:

$$T_A = (h_l + 1).(T_{MM} + T_W) \tag{7.3}$$

For values given above ($T_{MM} = 55$, $T_W = 5$, $h_l = 23$) we obtain $T_A = 1392.0$ cycles.

7.4.2 DFS Representation

DFS representation increases the cache hit ratio by involuntarily reading the descendant nodes for the next traversal step(s) if DFS traversal continues to the left descendant(s) of the current node. Assuming that the size of the *kd*-tree node is $M_{IN} = M_I + 2.M_P$, we derive the average traversal time T_A as follows:

$$T_A \quad = \quad (h_l + 1).[p_L.T_{MM}.\frac{M_{IN}}{M_{CL}} + T_W + T_C.(1 - \frac{M_{IN}}{M_{CL}}) + (1 - p_L).T_{MM}] \tag{7.4}$$

For $M_{IN} = 4 + 2.4 = 12$, $M_{CL} = 128$, and $p_L = 0.5$ we obtain $T_A = 859.1$ cycles.

7.4.3 Ordinary Subtree Representation

Assume that M_{CL} and M_{IN} are given. Let M_{ST} be the size of the memory needed for each subtree used to represent subtree type identification. We express the size of the memory taken by a complete ordinary subtree of height h:

$$\begin{aligned} M(h) &= (2^{h+1}-1).M_{IN}+M_{ST} \\ M(h) &\leq M_{CL} \end{aligned} \quad (7.5)$$

From Eq. 7.5 we derive the complete height of the ordinary subtree h_C:

$$h_C = \lfloor -1+\log(\frac{M_{CL}-M_{ST}}{M_{IN}}+1) \rfloor \quad (7.6)$$

The number of nodes in the incomplete ordinary subtree at the depth $d = h_C + 1$ is then:

$$N_{ODK} = \lfloor \frac{M_{CL}-(2^{h_C+1}-1).M_{IN}-M_{ST}}{M_{IN}} \rfloor \quad (7.7)$$

The average height of the subtree $h_A \geq h_C$ for $n_{ODK} > 0$ is computed as follows:

$$h_A = -1+\log(2^{h_C+1}+N_{ODK}) \quad (7.8)$$

Finally, the average traversal time for the whole *kd*-tree of height h_l is:

$$T_A = (h_l+1).(T_W+\frac{T_{MM}+T_C.h_A}{h_A+1}) \quad (7.9)$$

The subtrees are placed in the main memory, so they are aligned with the cache lines when read to the cache. Each subtree is thus stored in one cache line. The size of the unused memory in the cache line is then:

$$M^{OSR}_{unused} = M_{CL}-(2^{h_C+1}-1+N_{ODK}).M_{IN}-M_{ST} \quad (7.10)$$

For $M_{IN} = 12, M_{ST} = 4$, we get $h_C = 2$, $n_{ODK} = 3$, $h_A = 2.46$, $M^{OSR}_{unused} = 4$, and the average traversal time $T_A = 555.9$ cycles.

7.4.4 Compact Subtree Representation

Let M_I be the size of the memory to represent the information in the *kd*-tree node, M_P the memory taken by one pointer. The size of the memory consumed by a complete subtree of height h is expressed as follows:

$$\begin{aligned} M(h) &= (2^{h+1}-1).M_I+2^{h+1}.M_P+M_{ST} \\ M(h) &\leq M_{CL} \end{aligned} \quad (7.11)$$

The complete height h_C of the subtree is derived from Eq. 7.11 similarly to Eq. 7.6 as follows:

$$h_C = -1+\lfloor \frac{M_{CL}+M_I-M_{ST}}{M_I+M_P} \rfloor \quad (7.12)$$

In the same way as for ordinary subtree representation, we derive the number of nodes n_{ODK} located at the depth $d = h_C + 1$ in the subtree:

$$N_{ODK} = \lfloor \frac{M_{CL}-2^{h_C+1}.(M_I+M_P)+M_I-M_{ST}}{M_I+M_P} \rfloor \quad (7.13)$$

The unused memory for one subtree in the cache line can be derived similarly as for an ordinary subtree:

$$M^{CSR}_{unused} = M_{CL}-(2^{h_C+1}-1+N_{ODK}).M_N-2.M_P.(N_{ODK}+2^{h_C}-N_{ODK}/2)-M_{ST} \quad (7.14)$$

The average height of subtree h_A and the average traversal time T_A are computed using Eq. 7.8 and Eq. 7.9. For $M_P = 4$, $M_I = 4$, and $M_{ST} = 4$ we compute $h_C = 3$, $n_{ODK} = 0$, $h_A = 3.0$, $M^{CSR}_{unused} = 0$, and $T_A = 510.0$ cycles.

The h_C, n_{ODK}, and h_A as the function of the cache line size for ordinary and compact subtree representations, and T_A for all *kd*-tree representations, are depicted in Fig 7.4.

7.5 Simulation Results

We implemented a special program to simulate the data transfer in a typical memory hierarchy of a computer that runs DFS traversal on a complete *kd*-tree. The simulation was carried out for the same memory hierarchy and *kd*-tree properties as in the previous section: $T_{MM} = 53$, $T_C = 4$, $T_W = 5$, $h_l = 23$, $M_P = 4$ Bytes, $M_I = 4$ Bytes, $M_{ST} = 4$ Bytes, and a four-way set associative cache with cache line size $M_{CL} = 2^7 = 128$ Bytes; the size of the cache was 2^{20} Bytes. The cache placement algorithm and its structure correspond to those found in superscalar processors, we simulated MIPS R8000/R10000 [130].

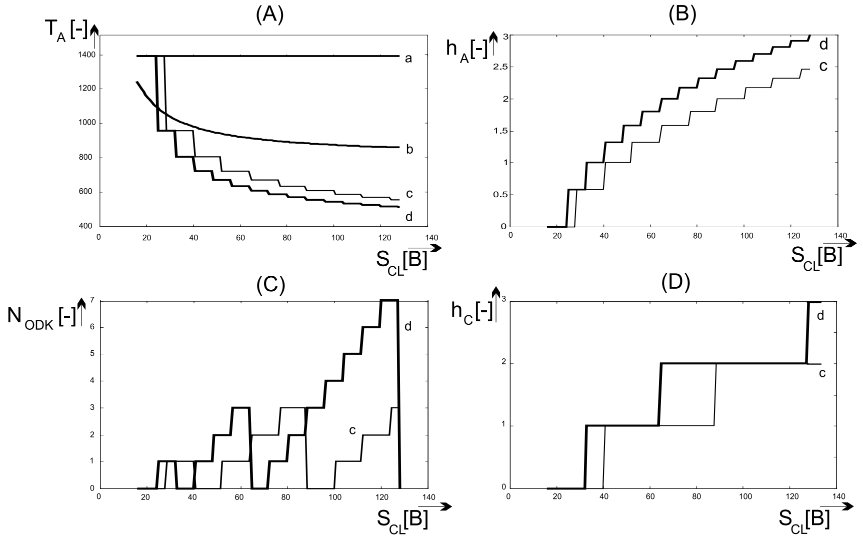


Figure 7.4: The analysis: (A) Average traversal time $T_A(M_{CL})$ for all *kd*-tree representations, (B) $h_A(M_{CL})$, (C) $n_{ODK}(CL)$, (D) $h_C(CL)$ for subtree representations; Representations: (a) Random (b) DFS (c) Ordinary subtree (d) Compact subtree.

	Representation			
Parameter	random	DFS	ordinary subtree	compact subtree
$T_A[cycles]$ (theoretical)	1392.0	859.1	555.9	510.0
$T'_A[cycles]$ (simulated)	987.1	629.4	445.6	379.3
$r = T_A/T'_A[-]$	1.41	1.36	1.24	1.34
$\tilde{r}_{CH}[\%]$	35.8	69.8	83.5	90.3

Table 7.1: The average traversal time computed theoretically and obtained by the simulation, ratio of traversal times, and cache hit ratio from the simulation for DFS traversal.

The theoretical and simulated times, and their ratio, are summarized in Table 7.1. The parameter $\tilde{r}_{CH}$ is the average cache hit ratio to access a *kd*-tree node in the cache during DFS traversal. The average cache hit ratio for the node as the function of its depth in the *kd*-tree is shown in Table 7.2.

Parameter	Depth 0	1	2	3	4	5	6	7	8	9	10	11
$\tilde{r}_{CH}$ (random)	100	100	100	100	97	91	62	52	39	25	21	18
$\tilde{r}_{CH}$ (DFS)	100	100	100	100	100	93	79	84	58	56	63	51
$\tilde{r}_{CH}$ (ordinary subtree)	100	100	100	100	100	100	97	73	90	85	53	79
$\tilde{r}_{CH}$ (compact subtree)	100	100	100	100	100	100	100	100	69	100	100	100

Parameter	Depth 12	13	14	15	16	17	18	19	20	21	22	23
$\tilde{r}_{CH}$ (random)	21	19	19	0	0	0	0	0	0	0	0	0
$\tilde{r}_{CH}$ (DFS)	57	59	47	59	54	48	51	49	47	54	43	54
$\tilde{r}_{CH}$ (ordinary subtree)	80	64	66	79	66	70	72	74	61	75	74	62
$\tilde{r}_{CH}$ (compact subtree)	7	100	100	100	1	100	100	100	0	100	100	100

Table 7.2: The average cache hit ratio $\tilde{r}_{CH}[\%]$ as the function of node depth in the *kd*-tree.

Note that for $M_{CL} = 128$ the compact subtree is complete, so the cache hit ratio for all the nodes at the same depth in the *kd*-tree is equal. This is why values of $\tilde{r}_{CH}$ for depth 12, 16, and 20 are quite different than values of $\tilde{r}_{CH}$ for neighbor depths, since the *kd*-tree nodes in the specified depths are often read first from the main memory. The probability that these nodes are already loaded in the cache is smaller with increasing depth in the *kd*-tree.

The average traversal times obtained by simulation correlate well with those computed theoretically. It is obvious that the times obtained by the simulation are smaller than those derived theoretically, since the theoretical analysis supposes in each step an initial value of cache hit ratio $r_{CH} = 0.0$.

7.6 Results of Experiments

We tested the influence of memory mapping experimentally for the recursive ray traversal algorithm TA^{A}_{rec} for random, DFS, and ordinary subtree representation (in [72]). We decided not to implement the compact subtree representation since accessing leaf nodes in subtrees without pointers is more complex and also requires a special traversal algorithm. Theoretical analysis above also shows that the results will not bring a significant improvement of performance compared with ordinary subtree representation. The tests were performed on a subset of G^{4}_{SPD} scenes from *SPD* for the testing procedure TP_D (ray tracing). The *kd*-tree representation influences only the average time of one traversal step, which has impact on T_R, Θ_{rat}, and Θ_{RUN} of the minimum testing output (see Section 2.5). The results of experiments for TP_D are summarized in Table 7.3.

The experiments were conducted on SGI O^2 with processor MIPS R8000, 180MHz, 128 MBytes RAM ($S_{CL} = 128$), running *Irix 6.1* operating system using the GOLEM rendering system.

7.7 Discussion

The performance of the recursive ray traversal algorithm is not improved as significantly as could have been expected from the simulation described in Section 7.5. The first reason is that the simulation was performing DFS with 50% probability to turn left in each node of the complete binary tree. This is no longer true for testing procedure TP_D when the subsequent primary rays are generated in scanline order. Then these similar rays are likely to hit the same sequence of nodes of the *kd*-tree (traversal coherence, see the Chapter 6). The second reason is the smaller number of nodes in the *kd*-trees built for the test scenes compared with the number of *kd*-tree nodes used for the simulation. The third possible reason

	Scene						
Parameter	balls4	gears4	mount6	rings7	tetra6	tree11	average
N	7382	9345	8196	8401	4096	8191	–
$N_G + N_E$	8469	20541	13369	23455	6011	5675	–
$\tilde{N}_{TS}$ $[\times 10^6]$	53.8	77.4	68.1	47.4	5.6	37.2	–
T_R $(random)$ $[s]$	87.0	314.6	53.0	102.4	7.41	92.6	–
T_{TS} $(random)$ $[s]$	42.1	59.8	22.9	21.5	4.09	18.6	–
T_{TS} (DFS) $[s]$	35.6	51.8	17.4	14.6	3.76	14.6	–
T_{TS} $(ordinary\ subtree)$ $[s]$	32.5	46.9	15.1	13.1	3.75	12.3	–
$\frac{\tilde{C}_{TS}(random)}{\tilde{C}_{TS}(DFS)}$ [-]	1.18	1.15	1.31	1.47	1.09	1.27	1.25
$\frac{\tilde{C}_{TS}(random)}{\tilde{C}_{TS}(ordinary\ subtree)}$ [-]	1.30	1.28	1.52	1.64	1.09	1.51	1.39

Table 7.3: The result for testing procedure TP_D on a subset of G^4_{SPD} scenes. Parameter T_{TS} refers to the time to devoted to traversing only ($T_{TS} = \tilde{C}_{TS}.N_{rays}.r_{SI}.r_{ITM} = (1 - \Theta_{rat}).T_R.\frac{\Theta_{RUN}}{\Theta_{APP}+\Theta_{RUN}}$). The *kd*-trees were built with OSAH and ad hoc termination criteria: $d_{max} = 16$, $N_{max} = 2$.

is that the model describing the transfer of data between the processor and the main memory is still too simplified to model the real behavior of a memory hierarchy system based on the MIPS R8000 processor.

We have shown that the properties of the *kd*-tree representation in the memory for such a traversal algorithm as those in *RSAs* stay in the range given by theoretical analysis and also the results of the simulation given in the previous section. The impact of *kd*-tree representation on *RSA* performance is not as high as might have been expected from the theoretical analysis and simulation, since the order of the nodes visited in the recursive ray traversal algorithm when it is applied in the testing procedure TP_D differs from that induced by DFS.

7.8 Conclusion and Future Work

In this chapter we analyzed the time complexity and cache hit ratio of different *kd*-tree representations in computer architectures with a large cache line for DFS order traversal in detail. We have shown that the time complexity of traversing a *kd*-tree is reduced by organizing its inner representation so that it matches the memory hierarchy better.

We showed experimentally that DFS and ordinary subtree representations can decrease the traversal time for *RSAs* based on the *kd*-tree, using TP_D testing procedure. Theoretically, the subtree representation decreases the traversal time for DFS order traversal by 62% and increases the cache hit ratio from 35% to 90% for a given example of a common memory hierarchy. In addition, proposed subtree representation of the *kd*-tree decreases the size of memory to store the nodes of a *kd*-tree by 57% compared with the random representation.

Future research work on the technique presented here could cover efficient memory mapping for other hierarchical data structures, namely other variants of multi-dimensional binary trees and hierarchical data structures in general. Dynamization of these data structures with regard to cache sensitive representation is also an interesting topic for further study.

Chapter 8

Conclusion and Future Work

In the thesis we dealt with ray shooting algorithms. Ray shooting itself is known to have no algorithm aiming at worst-case complexity feasible in practical implementation, since it has already been proved that such an algorithm for N objects run at least in $O(\log N)$ requiring $\Omega(N^4)$ storage and preprocessing time in the worst case. For this reason we dealt with heuristic algorithms for ray shooting aimed at average-case complexity. Since these algorithms for generally specified input – given N objects – are particularly difficult or even impossible to be successfully analyzed theoretically at present, the findings in the thesis are not expressed using the worst-case complexity measure and commonly used O-notation. Instead, for the presented algorithms, we report the results of experiments carried out for a set of scenes. For this purpose we used thirty scenes from the Standard Procedural Database, which are publicly available; the experimental results concerning algorithms described in the thesis are thus reproducible and verifiable by any subsequent researchers. Since average-case techniques have been used and the findings are supported by the results of experiments, we cannot claim time optimality of any ray shooting algorithm described in this thesis.

All the algorithms that form the subject of the thesis were implemented and tested. Since each implementation can be subject to bugs and implementation errors, in order to minimize such risks we visualized the spatial data structures underlying the tested ray shooting algorithms [81, 35] and verified the invariants of all experiments performed. A considerable implementation effort was needed to implement all the algorithms; it includes more than one hundred thousand lines of source code in C++, excluding third-party sources.

The conclusions and possible future research topics are presented at the end of each chapter, and here we provide only a *short summary* of all results achieved, and some suggestions for possible future research. Since the summary given here is concise, an interested reader should follow Sections *Conclusion and Future Work* of Chapters 2–7 for more details.

8.1 Summary of Results

In the first part of the thesis (Chapter 2 and 3), we dealt with general issues of heuristic ray shooting algorithms.

In Chapter 2 we developed computation model and performance model for ray shooting algorithms so any ray shooting algorithm can be mapped to these models. Based on these two models we developed the methodology for comparing ray shooting algorithms, which is based on reporting the minimum testing output (a set of thirteen parameters) for each experiment performed. Under certain conditions this allows us to compare experimentally various ray shooting algorithms almost independently of implementation issues. An interesting by-product of comparison methodology development is the discovery that ray tracing as defined for scenes in Standard Procedural Database (513×513 primary rays, depth of recursion 4) is impossible to run in real time (at least 25 frames per second) on present-day commonly used uniprocessor hardware because of time demands of any ray shooting algorithm. (We suppose Intel Pentium II, 466 MHz, we do not assume the use of a special graphics hardware, see results in Appendix E for details.)

In Chapter 3 we presented a comparison of twelve commonly used ray shooting algorithms for a set of thirty test scenes, the concept of statistically best ray shooting algorithm, and a preliminary version of the algorithm for selecting an efficient ray shooting algorithm given scene characteristics. The comparison is a part of the ongoing long term BES project devoted to ray shooting algorithms that is in progress at present. Our finding was that ray shooting algorithm based on the *kd*-tree achieved statistically the best results in comparison with other ray shooting algorithms tested.

The second part of the thesis (Chapter 2–7) is devoted to various issues of ray shooting algorithms based on the *kd*-tree. We selected it for detailed research in accordance with the results of the first phase of the BES project.

In Chapter 4 we addressed the problem of *kd*-tree construction when the *kd*-tree is used as the underlying data structure for ray shooting algorithms. We dealt with top-down method for *kd*-tree construction, *i.e.*, with the positioning of a splitting plane in a node of the *kd*-tree and termination criteria. This construction algorithm estimates the cost of a constructed *kd*-tree that is used to govern the position of the splitting plane. The precision of the estimates influences the resulting efficiency of the *kd*-tree for the ray shooting algorithm. The basic cost model used for the estimate is based on several unrealistic assumptions, but allows us to improve the efficiency of the ray shooting algorithm based on the *kd*-tree by order(s) of magnitude for sparsely occupied scenes. We proposed the general cost model which describes more accurately the time complexity of ray shooting algorithms based on the *kd*-tree. The general cost model validates the use of the basic cost model, since it shows that the difference in performance achievable using these two cost models are rather limited. Other developments concerning *kd*-tree construction dealt with the use of empty space inside the scene, automatic termination criteria, the properties of approximating objects by their axis-aligned bounding boxes, and utilizing knowledge of the distribution of rays to be queried to further decrease the running time of ray shooting algorithms based on the *kd*-tree.

Chapter 5 described five ray traversal algorithms, which are used for traversing a ray through a *kd*-tree. These include the sequential ray traversal algorithm, the basic and robust version of the recursive ray traversal algorithm, and two versions of the ray traversal algorithm with neighbor-links that use additional data structures. An efficient ray traversal algorithm tries to decrease both the number of traversal steps per ray and the average cost of a traversal step. The design of a ray traversal algorithm always searches some tradeoff between these two quantities to get the time devoted to traversing the *kd*-tree as small as possible. There are two extremes, ray traversal algorithm either visits a minimum number of *kd*-tree nodes with a somewhat higher cost of a traversal step, or the cost of one traversal step is small, but more traversal steps are required. Our contribution concerning ray traversal algorithms is that we developed a new robust recursive ray traversal algorithm for the *kd*-tree. Further, we compared all the known ray traversal algorithms experimentally.

Chapter 6 dealt with the problem of a ray traversal algorithm for a set of rays that have similar directions and origins. When some rays are shot and the rays exhibit some sense of similarity, this knowledge can be used to decrease further the time consumed by traversing the *kd*-tree. Such ray sets are often induced by the application, when rays are restricted to the convex shaft – they can have the same or a similar point of origin and direction. Under these conditions, we described the construction of the longest common traversal sequence, which gives us the sequence *kd*-tree nodes to be visited. The use of the longest common traversal sequence decreases the number of traversal steps and thus the time for traversing the *kd*-tree. Obviously, it does not change the number of ray-object intersection tests to be performed. We described two variants of the longest common traversal sequence – simple and hierarchical. Then we went on to show how the concept can be utilized within the application, and the results of experiments for hidden surface removal.

Chapter 7 described a more hardware-oriented topic – the mapping of *kd*-tree nodes to the memory of a computer. We showed that the mapping of *kd*-tree nodes on computer architecture with a large cache line has impact on the total running time of ray shooting algorithms, *i.e.*, the cost of one traversal step. We described four mapping methods and analyzed them both theoretically and experimentally. Our results show that it is possible to decrease the cost of one traversal step and the size of memory for representing a *kd*-tree on this type of computer architecture.

8.2 Suggestions for Further Research

Some of the algorithmic techniques described in the thesis can be extended and further researched in various contexts. Concerning the first part of the thesis (Chapter 2), we consider the comparison methodology as presented to be more or less complete. The BES project (Chapter 3) should be completed, and it will provide the results of experiments for one hundred scenes. These results will allow us to confirm or disprove the results performed on thirty scenes from the Standard Procedure Database, and then to show the relationship between these different sets of scenes. The algorithm to select an efficient ray shooting algorithm for a given scene based on the scene characteristics will be either validated or improved.

Although the second part of the thesis describing ray shooting algorithms based on the *kd*-tree would seem exhaustive, it still offers some possible topics for further research. The crucial issue of any further research in this direction is to what extent and at what cost it is possible to improve the performance of new algorithms for ray shooting based on the *kd*-tree compared with the algorithms presented in this thesis. Possible topics of further research include an improved algorithm that will estimate the cost of a *kd*-tree to be built given a set of objects, an algorithm that estimates blocking factor, an improved version of automatic termination criteria algorithm, an efficient algorithm for computing the cost in the general cost model, and an algorithm for the *kd*-tree construction with clustering of objects. For ray traversal algorithms an interesting topic of research is an algorithm that selects the ray traversal algorithm to be used to achieve the best possible performance given a ray shooting query. The concept of the longest common traversal sequence can be researched from the application point of view; how the longest common traversal sequence can be applied to minimize the running time in hidden surface removal determining the resolution of the underlying sampling pattern in image space for a given *kd*-tree. Further, the application of the longest common traversal sequence in particular global illumination algorithms can be researched. The memory mapping concept for the *kd*-tree allows us to raise the question of whether a similar approach can also be successfully applied in the representation of other spatial data structures. Another interesting and promising research issue is how to apply *kd*-trees for ray shooting algorithm in the case of moving, deforming, and animated objects without completely rebuilding the *kd*-tree for each frame of an image sequence.

Bibliography

[1] ANSI754. ANSI/IEEE std. 754-1985. An American National Standard. *IEEE Standard for Binary Floating-Point Arithmetic.* New York, IEEE 1985.

[2] ISO/IEC 14772-1:1997. VRML'97: The virtual reality modelling language, 1997.

[3] 3D Object Intersection Home Page. Maintained by T. Möller, E. Haines and P. Foscari, 2000. `http://www.realtimerendering.com/int/`.

[4] P. Agarwal, T. Murali, and J. Vitter. Practical techniques for constructing binary space partitions for orthogonal rectangles. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 382–384, 1997.

[5] P. K. Agarwal, B. Aronov, and M. Sharir. Computing envelopes in four dimensions with applications. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 348–358, 1994.

[6] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. Tech. Report CS-1997-11, Department of Computer Science, Duke University, 1997.

[7] P. K. Agarwal and J. Matoušek. On range searching with semialgebraic sets. In *Proc. 17th Internat. Sympos. Math. Found. Comput. Sci.*, volume 629 of *Lecture Notes Comput. Sci.*, Springer-Verlag, pages 1–13, 1992.

[8] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. In *Proc. 24th Annu. ACM Sympos. Theory Comput.*, pages 517–526, 1992.

[9] P. K. Agarwal and M. Sharir. Ray shooting amidst convex polytopes in three dimensions. In *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, pages 260–270, 1993.

[10] A. Aho, J. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Reading, Mass., 1974.

[11] J. Amanatides. Ray tracing with cones. In *Computer Graphics (SIGGRAPH '84 Proceedings)*, volume 18, pages 129–135, July 1984.

[12] J. Amanatides and A. Woo. A fast voxel traversal algorithm for ray tracing. In G. Marechal, editor, *Proc. Eurographics '87*, pages 3–10, Aug. 1987.

[13] O. Arnold. *Probability, statistics, and queuing theory with computer science applications.* Academic Press, San Diego, 1990.

[14] B. Aronov and S. Fortune. Average-case ray shooting and minimum weight triangulations. In *Proc. 13th Annu. ACM Sympos. Comput. Geom.*, pages 203–212, 1997.

[15] J. Arvo. Linear-time voxel walking for octrees. *Ray Tracing News. Available from `htpp://www.acm.org/tog/resources/RTNews/html/rtnews2d.html`*, 1(5), 1988.

[16] J. Arvo. Ray tracing with meta-hierarchies. In *SIGGRAPH '90 Advanced Topics in Ray Tracing course notes*. ACM Press, Aug. 1990.

[17] J. Arvo and D. Kirk. Fast ray tracing by ray classification. In M. C. Stone, editor, *(SIGGRAPH '87 Proceedings)*, volume 21, pages 55–64, July 1987.

[18] J. Arvo and D. Kirk. *A survey of ray tracing acceleration techniques,* In A. S. Glassner editor, *An introduction to ray tracing,* Academic Press, pages 201–262, 1989.

[19] J. S. Badt. Two algorithms for taking advantage of temporal coherence in ray tracing. *The Visual Computer*, 4(3):123–132, Sept. 1988.

[20] P. Bekaert. *Hierarchical and Stochastic Algorithms for Radiosity*. Ph.D. thesis, Department of Computer Science, Katholieke Universiteit Leuven, 1999.

[21] J. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18:509–517, 1975.

[22] M. D. Berg, D. Halperin, M. Overmars, J. Snoeyink, and M. V. Kreveld. Efficient ray shooting and hidden surface removal. *Algorithmica: An International Journal in Computer Science*, 12(1):30–53, 1994.

[23] K. Bouatouch, M. O. Madani, T. Priol, and B. Arnaldi. A new algorithm of space tracing using a CSG model. In G. Marechal, editor, *Proc. Eurographics '87*, pages 65–78, Aug. 1987.

[24] R. Capelli. Fast approximation to the arctangent. In D. Kirk, editor, *Graphics Gems II*, Academic Press, San Diego, pages 389–391, 1992.

[25] F. Cazals, G. Drettakis, and C. Puech. Filtering, clustering and hierarchy construction: A new solution for ray-tracing complex scenes. *Computer Graphics Forum,* 14(3):C371–C382, 1995.

[26] F. Cazals and C. Puech. Bucket-like space partitioning data-structures with applications to ray-tracing. In *13th ACM Symposium on Computational Geometry*, Nice, pages 11–20, 1997.

[27] F. Cazals and M. Sbert. Some integral geometry tools to estimate the complexity of 3d scenes. Technical Report RR-3204, The French National Institue for Research in Computer Science and Control (INRIA), July 1997.

[28] J. Chapman, T. W. Calvert, and J. Dill. Exploiting temporal coherence in ray tracing. In Proceedings of *Graphics Interface '90,* pages 196–204, May 1990.

[29] J. Chapman, T. W. Calvert, and J. Dill. Spatio-temporal coherence in ray tracing. In Proceedings of *Graphics Interface '91,* pages 101–108, June 1991.

[30] M. J. Charney and I. D. Scherson. Efficient traversal of well-behaved hierarchicial trees of extents for ray-tracing complex scenes. *The Visual Computer,* 6(3):167–178, June 1990.

[31] Y.-J. Chiang. Dynamic and I/O-efficient algorithms for computational geometry and graph problems: Theoretical and experimental results. Technical Report CS-95-27, Department of Computer Science, Brown University, Aug. 1995.

[32] J.-H. Chuang and W.-J. Hwang. A new space subdivision for ray tracing CSG solids. *IEEE Computer Graphics and Applications,* 15(6):56–62, Nov. 1995.

[33] J. G. Cleary and G. Wyvill. Analysis of an algorithm for fast ray tracing using uniform space subdivision. *The Visual Computer*, 4(2):65–83, July 1988.

[34] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990 (tenth printing 1993).

[35] L. Dachs. Vizualizace datových struktur pro ukládání prostorových dat. Master thesis, Czech Technical University, May 1999. In Czech.

[36] F. d'Amore and P. G. Franciosa. On the optimal binary plane partition for sets of isothetic rectangles. In *Proc. 4th Canad. Conf. Comput. Geom.*, pages 1–5, 1992.

[37] R. Day. *How to write & Publish a Scientific Paper.* Academic Press, 1997.

[38] D.Cohen and Z.Sheffer. Proximity clouds - an acceleration technique for 3D grid traversal. *The Visual Computer*, 11:27–38, 1994.

[39] M. de Berg, M. de Groot, and M. Overmars. New results on binary space partitions in the plane. *Comput. Geom. Theory Appl.*, 8:317–333, 1997.

[40] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[41] O. Devillers. The macro-regions: an efficient space subdivision structure for ray tracing. In W. Hansmann, F. R. A. Hopgood, and W. Strasser, editors, *Proc. Eurographics '89*, pages 27–38, Sept. 1989.

[42] D.-Z. Du and Y. Zhang. On heuristics for minimum length rectilinear partitions. *Algorithmica*, 5:111–128, 1990.

[43] F. Durand. *3D Visibility: Analytical Study and Applications.* Ph.D. thesis, Universite Grenoble I, July 1999.

[44] R. ENDL. An object-oriented ray tracing architecture for the analysis of ray-generators in spatial subdivisions. In Proceedings of *Compugraphics '95*, pages 268–277, Dec. 1995.

[45] R. Endl and M. Sommer. Classification of ray-generators in uniform subdivisions and octrees for ray tracing. *Computer Graphics Forum*, 13(1):C3–C19, Mar. 1994.

[46] M. Feixas, E. del Acebo, P. Bekaert, and M. Sbert. An information theory framework for the analysis of scene complexity. In P. Brunet and R. Scopigno, editors, *Proc. Eurographics '97*, pages 95–106, Sept. 1999.

[47] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics: Principles and Practice.* Addison-Wesley Publishing Co., Reading, Mass., 2nd edition, 1990.

[48] T. Foris, G. Márton, and L. Szirmay-Kalos. Ray shooting in logarithmic time. In Proceedings of *Winter School of Computer Graphics 96*, pages 84–90, Feb. 1996.

[49] A. Formella and C. Gill. Ray tracing: a quantitative analysis and a new practical algorithm. *The Visual Computer*, 11(9):465–476, 1995.

[50] A. Formella, C. Gill, and V. Hofmeyer. Fast ray tracing of sequences by ray history evaluation. In Proceedings of *Computer Animation '94*, IEEE Computer Society Press, pages 184–191, May 1994.

[51] A. Fournier and P. Poulin. A ray tracing accelerator based on a hierarchy of 1D sorted lists. In Proceedings of *Graphics Interface '93*, pages 53–61, May 1993.

[52] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by a priori tree structures. In *Computer Graphics (SIGGRAPH '80 Proceedings)*, volume 14, pages 124–133, July 1980.

[53] A. Fujimoto, T. Tanaka, and K. Iwata. ARTS: Accelerated ray tracing system. *IEEE Computer Graphics and Applications*, 6(4):16–26, 1986.

[54] I. Gargantini and H. H. Atkinson. Ray tracing an octree: numerical evaluation of the first intersection. *Computer Graphics Forum*, 12(4):C199–C210, Oct. 1993.

[55] J. Genetti, D. Gordon, and G. Williams. Adaptive supersampling in object space using pyramidal rays. In *Computer Graphics Forum*, 17(1):C29–C54, Mar. 1998.

[56] M. Gervautz. Consistent schemes for addressing surfaces when ray tracing transparent CSG objects. *Computer Graphics Forum*, 11(4):C203–C211, Oct. 1992.

[57] M. Gigante. Accelerated ray tracing using non-uniform grids. In Proceedings of *Ausgraph '90*, pages 157–163, 1988.

[58] Global illumination mailing list. `http://w3imagis.imag.fr/~Francois.Sillion/GlobillumList.html`.

[59] A. S. Glassner. Space subdivision for fast ray tracing. *IEEE Computer Graphics and Applications*, 4(10):15–22, Oct. 1984.

[60] A. S. Glassner. Spacetime ray tracing for animation. *IEEE Computer Graphics and Applications*, 8(2):60–70, Mar. 1988.

[61] A. S. Glassner. *An Introduction to Ray Tracing*. Academic Press, 1989.

[62] A. S. Glassner. *Principles of Digital Image Synthesis*. Computer Graphics and Geometric Modeling. Morgan Kaufmann, San Francisco, CA, 1995.

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

[64] P. Gonzalez and F. Gisbert. Object and ray coherence in the optimization of the ray tracing algorithm. In Proceedings of *Computer Graphics International '98 (CGI'98)*, Hannover, Germany, pages 264–267, June 1998.

[65] M. T. Goodrich and R. Tamassia. Dynamic ray shooting and shortest paths via balanced geodesic triangulations. In *Proc. 9th Annu. ACM Sympos. Comput. Geom.*, pages 318–327, 1993.

[66] D. Gordon and S. Chen. Front-to-back display of BSP trees. *IEEE Computer Graphics and Applications*, 11(5):79–85, Sept. 1991.

[67] E. Gröller and W. Purgathofer. Using temporal and spatial coherence for accelerating the calculation of animation sequences. In W. Purgathofer, editor, *Proc. Eurographics '91*, pages 103–113, Sept. 1991.

[68] E. Gröller and W. Purgathofer. Coherence in Computer Graphics. Technical Report TR-186-2-95-04, Institute of Computer Graphics, Vienna University of Technology, Favoritenstrasse 9/186, A-1040 Vienna, Austria, 1995. Human contact: technical-report@cg.tuwien.ac.at.

[69] E. A. Haines. A proposal for standard graphics environments. *IEEE Computer Graphics and Applications*, 7(11):3–5, Nov. 1987. Available from `http://www.acm.org/pubs/tog/resources/SPD/overview.html`.

[70] E. A. Haines and D. P. Greenberg. The light buffer: A ray tracer shadow testing accelerator. *IEEE Computer Graphics and Applications*, 6(9):6–16, Sept. 1986.

[71] E. A. Haines and J. R. Wallace. Shaft culling for efficient ray-traced radiosity. In P. Brunet and F. W. Jansen, editors, *Photorealistic Rendering in Computer Graphics (Proceedings of the Second Eurographics Workshop on Rendering)*, Springer-Verlag, New York, pages 122–138, 1994.

[72] V. Havran. Cache sensitive representation for the BSP tree. In Proceedings of *Compugraphics'97*, GRASP – Graphics Science Promotions & Publications, pages 369–376, Dec. 1997.

[73] V. Havran. Analysis of cache sensitive representation for binary space partitioning trees. *Informatica*, Slovene Society Informatika, ISSN 0350-5596, 23(3):203–210, May 1999.

[74] V. Havran. A summary of octree ray traversal algorithms. *Ray Tracing News*, 12(2):cca 10 pages, Dec. 1999. Available from `http://www.acm.org/tog/resources/RTNews/html/rtnv12n2.html`.

[75] V. Havran. GOLEM rendering system, 2000. HOME page at `http://www.cgg.cvut.cz/GOLEM`.

[76] V. Havran and J. Bittner. Constructing rectilinear BSP trees for preferred ray sets. In Proceedings of *Short Communication Papers of WSCG'99*, poster section, pages 1–2, Feb. 1999.

[77] V. Havran and J. Bittner. Rectilinear BSP trees for preferred ray sets. In Proceedings of *SCCG'99 (Spring Conference on Computer Graphics)*, Budmerice, Slovak Republic, pages 171–179, Apr./May 1999.

[78] V. Havran and J. Bittner. LCTS: Ray shooting using longest common traversal sequences. *Computer Graphics Forum (Proc. Eurographics '2000)*, 19(3):C59–C70, Aug 2000.

[79] V. Havran, J. Bittner, and J. Přikryl. Best efficiency scheme project proposal. Home page at: `http://www.cgg.cvut.cz/GOLEM/bes.html`, Oct 1999.

[80] V. Havran, J. Bittner, and J. Žára. Ray tracing with rope trees. In Proceedings of *SCCG'98 (Spring Conference on Computer Graphics)*, Budmerice, Slovak Republic, pages 130–139, Apr. 1998.

[81] V. Havran, L. Dachs, and J. Žára. VIS-RT: A visualization system for RT spatial data structures. In Proceedings of *WSCG'2000*, short communication papers, pages 28–35, Feb. 2000.

[82] V. Havran, T. Kopal, J. Bittner, and J. Žára. Fast robust BSP tree traversal algorithm for ray tracing. *Journal of Graphics Tools*, 2(4):15–23, Dec. 1997.

[83] V. Havran and W. Purgathofer. Comparison methodology for ray shooting algorithms. Technical Report TR-186-2-00-20, Institute of Computer Graphics, Vienna University of Technology, Favoritenstrasse 9/186, A-1040 Vienna, Austria, Nov. 2000. Human contact: technical-report@cg.tuwien.ac.at.

[84] V. Havran, J. Přikryl, and W. Purgathofer. Statistical comparison of ray-shooting efficiency schemes. Technical Report TR-186-2-00-14, Institute of Computer Graphics, Vienna University of Technology, Favoritenstrasse 9/186, A-1040 Vienna, Austria, May 2000. Human contact: technical-report@cg.tuwien.ac.at.

[85] V. Havran and F. Sixta. Comparison of hierarchical grids. *Ray Tracing News*, 12(1):cca 4 pages, June 1999. Available from `http://www.acm.org/tog/resources/RTNews/html/rtnv12n1.html`.

[86] V. Havran and J. Žára. Evaluation of BSP properties for ray–tracing. In Proceedings of *SCCG'97 (Spring Conference on Computer Graphics)*, pages 155–162, Budmerice, June 1997.

[87] P. S. Heckbert and P. Hanrahan. Beam tracing polygonal objects. *Computer Graphics (SIGGRAPH'84 Proceedings)*, 18(3):119–127, July 1984.

[88] M. Held. ERIT – a collection of efficient and reliable intersection tests. *Journal of Graphics Tools*, 2(4):25–44, Dec. 1997.

[89] T. Horvath, G. Márton, P. Risztics, and L. Szirmay-Kalos. Ray coherence between a sphere and a convex polyhedron. *Computer Graphics Forum*, 11(2):C163–C172, June 1992.

[90] P.-K. Hsiung and R. H. Thibadeau. Accelerating ARTS. *The Visual Computer*, 8(3):181–190, Mar. 1992.

[91] F. W. Jansen. Data structures for ray tracing. In L. R. A. Kessener, F. J. Peters, and M. L. P. van Lierop, editors, *Data Structures for Raster Graphics*, Springer-Verlag, New York,, pages 57–73, 1986.

[92] D. Jevans. Object space temporal coherence for ray tracing. In Proceedings of *Graphics Interface '92*, pages 176–183, May 1992.

[93] D. Jevans and B. Wyvill. Adaptive voxel subdivision for ray tracing. In Proceedings of *Graphics Interface '89*, pages 164–172, June 1989.

[94] M. Kaplan. *Space-Tracing: A Constant Time Ray-Tracer*, pages 149–158, July 1985.

[95] M. R. Kaplan. The use of spatial coherence in ray tracing. In D. E. Rogers and R. A. Earnshaw, editors, *Techniques for Computer Graphics*, Springer-Verlag, pages 173–193, 1987.

[96] T. L. Kay and J. T. Kajiya. Ray tracing complex scenes. In D. C. Evans and R. J. Athay, editors, *SIGGRAPH '86 Proceedings)*, volume 20, pages 269–278, Aug. 1986.

[97] D. Kirk and J. Arvo. Improved ray tagging for voxel-based ray tracing. In J. Arvo, editor, *Graphics Gems II*, Academic Press, San Diego, pages 264–266, 1991.

[98] K. S. Klimaszewski. *Faster ray tracing using adaptive grids and area sampling.* Ph.D. thesis, Brigham Young University, Dec. 1994.

[99] K. S. Klimaszewski. *Faster Ray Tracing Using Adaptive Grids and Area Sampling.* Ph.D. thesis, Dept. of Civil and Environmental Engineering, Brigham Young University, Provo, Utah, 1994.

[100] K. S. Klimaszewski and T. W. Sederberg. Faster ray tracing using adaptive grids. *IEEE Computer Graphics and Applications*, 17(1):42–51, Jan./Feb. 1997.

[101] Y. P. Kuzmin. Ray traversal of spatial structures. *Computer Graphics Forum*, 13(4):C223–C227, Oct. 1994.

[102] B. Kwon, D. S. Kim, K.-Y. Chwa, and S. Y. Shin. Memory-efficient ray classification for visibility operations. *IEEE Transactions on Visualization and Computer Graphics*, 4(3):193–201, July/Sept. 1998.

[103] A. Lingas. Heuristics for minimum edge length rectangular partitions of rectilinear figures. In *Proc. 6th GI Conf. Theoret. Comput. Sci.*, volume 145 of *Lecture Notes Comput. Sci.*, Springer-Verlag, pages 199–210, 1983.

[104] J. D. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. In Proceedings of *Graphics Interface '89*, pages 152–63, June 1989.

[105] J. D. MacDonald and K. S. Booth. Heuristics for ray tracing using space subdivision. *Visual Computer*, 6(6):153–65, 1990.

[106] G. Márton. Acceleration of ray tracing via Voronoi diagrams. In A. W. Paeth, editor, *Graphics Gems V*, Academic Press, Boston Mass., pages 268–284, 1995.

[107] G. Márton and L. Szirmay-Kalos. On average-case complexity of ray tracing algorithms. In Proceedings of *Winter School of Computer Graphics 95*, pages 187–196, Feb. 1995.

[108] H. Maurel, Y. Duthen, and R. Caubet. A 4D ray tracing. *Computer Graphics Forum*, 12(3):C285–C294, Aug 1993.

[109] M. D. J. McNeill, B. C. Shah, M.-P. Hebert, P. F. Lister, and R. L. Grimsdale. Performance of space subdivision techniques in ray tracing. *Computer Graphics Forum*, 11(4):C213–C220, Oct. 1992.

[110] J. S. B. Mitchell, D. M. Mount, and S. Suri. Query-sensitive ray shooting. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 359–368, 1994.

[111] S. Mohaban and M. Sharir. Ray shooting amidst spheres in three dimensions and related problems. *SIAM J. Comput.*, 26(3):654–674, June 1997.

[112] T. Möller and E. Haines. *Real-Time Rendering.* A K Peters, Ltd., 1999.

[113] C. Montani and R. Scopigno. Ray tracing CSG trees using the sticks representation scheme. *Computers and Graphics*, 14(3/4):481–490, 1990.

[114] K. Murakami and K. Hirota. Incremental ray tracing. In K. Bouatouch and C. Bouville, editors, *Photorealism in Computer Graphics*, Springer-Verlag, pages 17–32, 1992.

[115] C. Nyberg, T. Barclay, Z. Cvetanovic, J. Gray, and D. Lomet. Alphasort: A cache-sensitive parallel external sort. *VLDB Journal*, (4):603–627, 1995.

[116] M. Ohta and M. Maekawa. Ray coherence theorem and constant time ray tracing algorithm. In T. L. Kunii, editor, *Computer Graphics 1987 (Proceedings of CG International '87)*, Springer-Verlag, pages 303–314, 1987.

[117] M. Pellegrini. Ray shooting and lines in space. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, CRC Press LLC, Boca Raton, FL, chapter 32, pages 599–614, 1997.

[118] Q. Peng, Y. Zhu, and Y. Liang. A fast ray tracing algorithm using space indexing techniques. In G. Marechal, editor, *Proc. Eurographics '87*, pages 11–23, Aug. 1987.

[119] M. Pharr and P. Hanrahan. Geometry caching for ray-tracing displacement maps. In X. Pueyo and P. Schröder, editors, *Eurographics Rendering Workshop 1996*, Eurographics, Springer-Verlag, Wien, pages 31–40, June 1996.

[120] M. Pharr, C. Kolb, R. Gershbein, and P. Hanrahan. Rendering complex scenes with memory-coherent ray tracing. In T. Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pages 101–108, Aug. 1997.

[121] M. Quail. Space time ray tracing using ray classification. Bachelor thesis, Department of Computing, Macquarie University, Nov. 1996.

[122] E. Reinhard, A. J. F. Kok, and F. W. Jansen. Cost prediction in ray tracing. In *Rendering Techniques '96*, Springer-Verlag, Wien, pages 41–50, 1996.

[123] S. M. Rubin and T. Whitted. A 3-dimensional representation for fast rendering of complex scenes. In *SIGGRAPH '80 Proceedings*, volume 14, pages 110–116, July 1980.

[124] S. S and K. H. Directional safe zones & dual extent algorithms for efficient grid traversal. In Proceedings of *Graphics Interface'97,* pages 76–87, 1997.

[125] H. Samet. *Design and analysis of Spatial Data Structures: Quadtrees, Octrees, and other Hierarchical Methods.* Addison–Wesley, Reading, Mass., 1989.

[126] H. Samet. Implementing ray tracing with octrees and neighbor finding. *Computers and Graphics*, 13(4):445–60, 1989.

[127] H. Samet. *Applications of Spatial Data Structures*. Addison-Wesley, Reading, Mass., 1990.

[128] I. D. Scherson and E. Caspary. Data structures and the time complexity of ray tracing. *The Visual Computer*, 3(4):201–213, Dec. 1987.

[129] C. H. Sequin and E. K. Smyrl. Parameterized ray tracing. In J. Lane, editor, *SIGGRAPH '89 Proceedings*, volume 23, pages 307–314, July 1989.

[130] SGI. *Power Challenge* Technical report, Silicon Graphics Computer Systems, 1996.

[131] M. Shinya, T. Takahashi, and S. Naito. Principles and applications of pencil tracing. In M. C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 45–54, July 1987.

[132] G. Simiakakis. *Accelerating RayTracing with Directional Subdivision and Parallel Processing.* Ph.D. thesis, University of East Anglia, Oct. 1995.

[133] G. Simiakakis and A. M. Day. Five-dimensional adaptive subdivision for ray tracing. *Computer Graphics Forum*, 13(2):C133–C140, June 1994.

[134] F. Sixta. Dělení prostoru pro sledování paprsku. Bachelor thesis, Czech Technical University in Prague, May 1997. In Czech.

[135] F. Sixta. Datové struktury pro ukládání prostorových dat. Master thesis, Czech Technical University, Jan. 1999. In Czech.

[136] J. M. Snyder and A. H. Barr. Ray tracing complex models containing surface tessellations. In M. C. Stone, editor, *Computer Graphics (SIGGRAPH '87 Proceedings)*, volume 21, pages 119–128, July 1987.

[137] SOFTIMAGE. *Mental Ray, A Programmer's Reference Guide*. Gesellschaft für Computerfilm and Maschinintelligenz Gmbh & Co. KG, Berlin, 1995.

[138] H. Solomon. *Geometric Probability.* J.W. Arrowsmith Ltd, 1978.

[139] N. Stolte and R. Caubet. Discrete ray-tracing of huge voxel spaces. *Computer Graphics Forum*, 14(3):C383–C394, Sept. 1995.

[140] L. Stone. *Theory of Optimal Search*. Academic Press, New York, 1975.

[141] B. Stroustrup. *The C++ Programming Language, 3rd edition.* Addison-Wesley, 1997.

[142] W. Sturzlinger. Bounding volume construction using point clouds. In Proceedings of *SCCG'96 (Spring Conference on Computer Graphics)*, pages 239–246, June 1996.

[143] K. R. Subramanian. Personal communication, 1998.

[144] K. R. Subramanian and D. S. Fussel. Factors affecting performance of ray tracing hierarchies. Technical Report Tx 78712, The University of Texas at Austin, July 1990.

[145] K. R. Subramanian and D. S. Fussel. A search structure based on k-d trees for efficient ray tracing. Technical Report (Ph.D. Dissertation), Tx 78712-1188, The University of Texas at Austin, Dec. 1990.

[146] K. R. Subramanian and D. S. Fussell. Automatic termination criteria for ray tracing hierarchies. In Proceedings of *Graphics Interface '91*, pages 93–100, June 1991.

[147] K. Sung. A DDA octree traversal algorithm for ray tracing. In W. Purgathofer, editor, *Proc. Eurographics '91*, pages 73–85, Sept. 1991.

[148] K. Sung and P. Shirley. Ray tracing with the BSP tree. In D. Kirk, editor, *Graphics Gems III*, Academic Press, San Diego, pages 271–274, 1992.

[149] L. Szirmay-Kalos. Monte-Carlo methods in global illumination, script written in Institute of Computer Graphics, Vienna University of Technology, Oct. 1999.

[150] L. Szirmay-Kalos and G. Márton. On the complexity of ray shooting. In *Dagstuhl Seminar on Rendering, 1996*, 1996.

[151] L. Szirmay-Kalos and G. Márton. On the limitations of worst–case optimal ray shooting algorithms. In Proceedings of *Winter School of Computer Graphics 97*, pages 562–571, Feb. 1997.

[152] L. Szirmay-Kalos and G. Márton. Analysis and construction of worst-case optimal ray shooting algorithms. *Computers and Graphics*, 22(2–3):167–174, Mar. 1998.

[153] L. Szirmay-Kalos and G. Márton. Worst-case versus average case complexity of ray-shooting. *Computing*, 61(2):103–131, 1998.

[154] R. Tarjan. *Data Structures and Network Algorithms.* Society for Industrial and Applied Mathematics, Philadelphia, 1987.

[155] S. Teller and J. Allex. Frustum casting for progressive, interactive rendering. Technical Report MIT LCS TR-740, MIT, Jan. 1998.

[156] M. van der Zwaan, E. Reinhard, and F. W. Jansen. Pyramid clipping for efficient ray traversal. In Proceedings of *Eurographics Rendering Workshop 1995*, Dublin, Ireland, pages 1–10, 1995.

[157] A. Watt and M. Watt. *Advanced Animation and Rendering Techniques.* ACM-PRESS, Addison-Wesley, 1992.

[158] H. Weghorst, G. Hooper, and D. P. Greenberg. Improved computational methods for ray tracing. *ACM Transactions on Graphics*, 3(1):52–69, Jan. 1984.

[159] K. Y. Whang, J. W. Song, J. W. Chang, J. Y. Kim, W. S. Cho, C. M. Park, and I. Y. Song. Octree-R: an adaptive octree for efficient ray tracing. *IEEE Transactions on Visualization and Computer Graphics*, 1(4):343–349, Dec. 1995.

[160] T. Whitted. An improved illumination model for shaded display. *Communications of the ACM*, 23(6):343–349, Aug. 1979.

[161] A. Wilkie, R. F. Tobler, and W. Purgathofer. Orientation lightmaps for photon radiosity in complex environments. In Proceedings of *Computer Graphics International ’2000 (CGI’2000)*, pages 279–286, June 2000.

[162] A. Woo. Fast ray-box intersection. In A. S. Glassner, editor, *Graphics Gems*, Academic Press, San Diego, pages 395–396, 1990.

[163] A. Woo. Ray tracing polygons using spatial subdivision. In *Proceedings of Graphics Interface ’92*, pages 184–191, May 1992.

[164] A. Woo, P. Poulin, and A. Fournier. A survey of shadow algorithms. *IEEE Computer Graphics and Applications*, 10(6):13–32, Nov. 1990.

[165] G. Wyvill, T. L. Kunii, and Y. Shirai. Space division for ray tracing in CSG (Constructive Solid Geometry). *IEEE Computer Graphics and Applications*, 6(4):28–34, Apr. 1986.

[166] F. Yamaguchi and M. Niizeki. Some basic geometric test conditions in terms of Pluecker coordinates and pluecker coefficients. *Visual Computer*, 13(1):29–41, 1997.

[167] P. Zemčík and A. Chalmers. Optimised CSG tree evaluation for space subdivision. *Computer Graphics Forum*, 14(2):C139–C146, June 1995.

Notation

Upper Case Roman

AG	adaptive grid
$A = (x, y, z)$	point in $\mathbb{E}^3$, entry point of ray traversal algorithm
$B = (x', y', z')$	point in $\mathbb{E}^3$, exit point of ray traversal algorithm
BES	Best Efficiency Scheme research project
BSP	binary space partitioning (tree)
BVH	bounding volume hierarchy
C	cost – the running time to perform some particular algorithmic operation
C_X	cost of operation X
C_{IT}	cost of ray-object intersection test
C_{IT}^{fail}	cost of failed ray-object intersection test
C_{IT}^{succ}	cost of successful ray-object intersection test
C_T	total cost for shooting an arbitrary ray
C_{TI}	cost of traversing the interior node of the kd-tree
C_{TL}	cost of traversing the leaf node of the kd-tree
C_{TS}	cost of a single RSA traversal step
$COMP$	compiler
$\vec{D}^{\mathrm{R}}$	direction vector for ray R
DS	data structure underlying a particular RSA
E	entry
ESSD	elementary spatial subdivision
$F, F(\nu)$	face, a face associated with $\mathcal{AB}(\nu)$ of the node ν
G^X	group of test scenes with number of objects in range $\langle 10^X + 1, 10^{X+1} \rangle$
G_{SPD}^X	group of test scenes from SPD where number of objects is closest to 10^X
GOLEM	implementation framework used for algorithms within the thesis
$HLCTS$	hierarchical longest common traversal sequence
H^+, H^-	positive or negative halfspace defined by plane in $\mathbb{E}^n$
HSSD	hierarchical spatial subdivision
HUG	hierarchy of uniform grids
HW	hardware
I	point of intersection
I_S^C	mutual visibility information
$IMPL$	implementation
$LCTS$	longest common traversal sequence
K	ray-object intersection test is performed K-times
L	instance of HLCTS
$LSA(b)$	the surface area of $\mathcal{AB}$ of the left child for position b
L	the case left
LO	the case left only
LR	the case left, then right
M	size of memory
N	number of objects
$\mathrm{N1}, \mathrm{N2}, \mathrm{N3}, \mathrm{N4}, \mathrm{N5}$	negative traversal case in a recursive traversal algorithm
N_E	number of elementary nodes in DS

N_{EE}	number of empty elementary nodes in *DS*
N_{ER}	total number of references to objects in elementary nodes of *DS*
N_{ETS}	number of elementary nodes accessed per ray
N_{EETS}	number of empty elementary nodes accessed per ray
N_G	number of generic nodes in *DS*
N_i	number of interior nodes in kd-tree
N_{IT}	number of ray-object intersection tests per ray
N_{IT}^{fail}	number of failed ray-object intersection tests per ray
N_{IT}^{succ}	number of successful ray-object intersection tests per ray
N_l	number of leaves in kd-tree
N_L	number of objects in the left child of the current node
N_{max}	number of objects in kd-tree node, when it is declared as a leaf
N_{rays}	number of rays
N_R	number of objects in the right child of the current node
N_{SP}	number of objects intersecting the splitting plane
N_{TI}	number of interior nodes traversed in kd-tree
N_{TL}	number of leaves traversed in kd-tree
N_{TS}	number of all nodes accessed per ray
N_V	number of voxels in uniform grid
$N(l)$	number of objects stored in the l-th leaf of kd-tree
O	object
O_i	i-th object
$O(g(n))$	upper bound of the worst-case complexity is $g(n)$
O^{R}	origin point of ray R
O84,O89,O93	various versions of octree built with spatial median subdivision
O84A,O93A	two variants of octree built with surface area heuristic
P	polygon
P_C	convex polygon
P1, P2, P3, P4, P5	positive traversal case in a recursive traversal algorithm
Q	center point
Q_i	center point of the $\mathcal{AB}$ of i-th object
R	ray
R	the case right
RL	the case right, then left
RG	recursive grid
RO	the case right only
$RSA(b)$	the surface area of $\mathcal{AB}$ of the right child for position b
RSA	ray shooting algorithm
S	set
S_{R}	set of rays
SA	surface area
$SA(X)$	surface area of the element X
S_F	size factor to generate scenes of different complexity in *SPD*
$SLCTS$	simple longest common traversal sequence
SPD	Standard Procedural Database [69]
SSD	spatial subdivision
T	time
TA	traversal algorithm
T_{MM}	latency of main memory
T_{app}	remaining running time of application excluding the time consumed by a ray shooting algorithm
T_C	cache latency
T_B	preprocessing time required to build data structures for particular ray shooting algorithm
T_{IT}	time for ray-object intersection tests
T_R	running time required to perform given *TP* in the application
T_{TS}	time for traversing the nodes of *DS*
TH	traversal history
TP	*RSA* testing procedure a generating particular set of rays given a scene $\mathcal{S}$

UG	uniform grid
U, V	points in $\mathbb{E}^3$
V_P	viewpoint in $\mathbb{E}^3$
$Vol(X)$	volume of X in $\mathbb{E}^3$
X, Y	entity, spatial region, cell, object, *etc.*
W	number of wins for particular *RSA* within BES project
W_R	viewport
$\mathrm{Z1}, \mathrm{Z2}, \mathrm{Z3}$	zero traversal case in a recursive traversal algorithm

Lower Case Roman

a	entry signed distance corresponding to entry point A
b	exit signed distance corresponding to entry point B
b	position of the splitting plane in $\mathcal{AB}$, $b \in \langle 0, 1\rangle$
b_{OM}	position of the splitting plane for object median
b_{SM}	position of the splitting plane for spatial median
c	real number
d	depth
$d(\nu)$	depth of node ν in the kd-tree (depth of the kd-tree root node is zero)
d_{max}	maximum depth allowed for a leaf in the kd-tree
d_{rec}	depth of rays, for primary rays holds $d_{rec} = 0$.
d_{voxel}	voxel density in context of uniform grid
e	integer number
en^G_{int}	average number of intersections with objects for global line, casting global lines
f	number of tasks where experiment failed due to time limit
h	height, height of a tree
i, j	indices
k	kurtosis
l	length
$\tilde{n}$	average number of objects in the voxel
m	number of tasks where experiment failed due to memory limits
n	integer number
	number of dimensions of space
n^G_{int}	average number of intersections with objects for global line, surface area
p	probability
p_X	probability of case X
$p_{Y\|X}$	conditional probability of case Y when case X occurs
p_0	probability of zero intersections
p^T	blocking factor, *i.e.*, probability of a ray hitting an object
r	ratio
r_{CH}	cache hit ratio
r_{ITM}	ratio of ray-object intersection tests performed to minimum number of intersection tests
r_{SI}	ratio of number of rays hitting objects to number of all rays
s	skewness
s_{len}	average span length
t	signed distance
u	real number
v	variance
w	width
x, y, z	real numbers
$\mathrm{x}, \mathrm{y}, \mathrm{z}$	axis denotation

Scripts

$\mathcal{AB}$($\mathcal{AB}s$)	axis-aligned bounding box(es)
$\mathcal{AB}(X)$	axis-aligned bounding box tightly enclosing entity X
$\mathcal{CS}$	convex shaft
$\mathcal{V}$	cell of SSD
$\mathcal{V}^n$	cell of SSD in $\mathbb{E}^n$ space
$\mathcal{G}$	sequence, sequence of kd-tree nodes
$\mathcal{S}$	scene (region of space)
$\mathcal{S}(N)$	scene containing N objects

Upper Case Greek

Δ	subset of four parameters of minimum testing output describing dynamic use of *DS*
Θ	subset of five hardware/implementation dependent parameters of minimum testing output
Σ	subset of four parameters of minimum testing output describing static properties of *DS*
Π	plane
Π_P	projection plane
Ψ	sparseness
$\Omega(f(n))$	lower bound of the worst-case complexity is $f(n)$
Ω	solid angle in $\mathbb{E}^3$

Lower Case Greek

ε	small positive constant
θ	angle between two vectors in $\mathbb{E}^3$
λ	nonuniformity coefficient
ν	node of *DS*
ν^G	generic node of *DS*
ν^E	elementary node of *DS*
ν^{EE}	empty elementary node of *DS*
ν^{EF}	full elementary node of *DS*
σ	standard deviation

Miscellaneous

$\|S\|$	cardinality of set S
$\|x\|$	absolute value of real number x
$\mathbb{E}^n$	n-dimensional Euclidean space
$\overline{UV}$	line segment with U and V as endpoints
$\log N$	dyadic logarithm of N
$d(U,V)$	distance between points U and V
∂X	boundary of X
$\hat{X}$	mark $\hat{}$ denotes that $\hat{X}$ is an estimate of quantity X
$\tilde{Y}$	mark $\tilde{}$ denotes that $\tilde{Y}$ is the average value of quantity Y
$\vec{Z}$	mark $\vec{}$ denotes that $\vec{Z}$ is the vector in $\mathbb{E}^n$ space
const	constant
$int(X)$	interior of X
$ext(X)$	exterior of X
U_x, U_y, U_z	coordinates of point U in $\mathbb{E}^3$
$lchild(\nu), rchild(\nu)$	left (right) child of the node ν in the kd-tree

Appendix A

C-pseudocode of sequential ray traversal algorithm TA$_{seq}$ for *kd*-tree.

```
/* Possible orientation of the splitting plane in the interior node of the kd-tree, */
/* "No_axis" denotes a leaf. */
enum Axes {X_axis, Y_axis, Z_axis, No_axis};

/* Declaration of the kd-tree node. */
struct KDTNode {
 Point3D min, max; /* extent of node ... six float values */
 GeomObjlist *objlist; /* list of enclosed objects */
 struct KDTNode *left; /* pointer to the left child */
 struct KDTNode *right; /* pointer to the right child */
 Axes axis; /* orientation of the splitting plane */
};

/* Locate leaf containing the point starting given node. */
KDTNode* LocateLeaf(KDTNode *node, Point3D point)
{
  KDTNode *currNode = node;

  if (point lies outside node bounding box)
    return ["no leaf exists"];

  while (currNode points to interior node) {
    if ( point[currNode->axis] < currNode->right.min[currNode->axis])
      currNode = currNode->left;
    else
      currNode = currNode->right;
  } /* while */

  /* return the found leaf that contains point */
  return (KDTNode *)currNode;
} /* LocateLeaf */

/* Sequential ray traversal algorithm */
Object RayTravAlgSEQ(KDTNode *rootNode, Ray ray)
{
  float a, b; /* entry/exit point signed distances */
  Point3D point; /* the point along the ray path */
  KDTNode *currNode; /* pointer to a kd-tree node */

  /* intersect the ray with sceneBox, find the entry and exit signed distance */
  RayBoxIntersect(ray, rootNode, &a, &b);

  if (ray does not intersect sceneBox)
    return ["No object"];

  /* start at the root node */
  currNode = rootNode;
  /* when ray has the origin inside sceneBox, */
  if (a < 0.0)
    /* use the point of origin for initial search, */
    point = ray.origin;
  else
    /* otherwise use ray entry point for sceneBox */
    point = ray.origin + ray.dir * (a + epsilon);

  /* starting from the root node, locate the first leaf */
  currNode = LocateLeaf(rootNode, point);

  /* traverse through whole kd-tree until the object */
  /* is intersected or the ray leaves the scene */
  while (currNode points to leaf) {
    /* find out signed distances to the leaf node bounding box */
    RayBoxIntersect(ray, currNode, &a, &b);

    if (currNode is not empty leaf) {
      "intersect ray with each object in the object list"
      "discarding those lying before (a) or farther than (b)"

      if (any intersection exists)
        return ["object with the closest intersection point"];
    } /* if */

    /* compute the point on the ray path in the next leaf */
    point = ray.origin + ray.dir * (b + epsilon);

    /* locate the next leaf along the ray path, if possible */
    currNode = LocateLeaf(root, point);
  } /* while */

  /* all the leaves of the kd-tree along the ray path were tested */
  return ["No object"];
} /* RayTravAlgSEQ */
```

Appendix B

C-pseudocode of recursive ray traversal algorithm TA^{A}_{rec} for the *kd*-tree.

```
/* Possible orientation of the splitting plane in the interior node of the kd-tree, */
/* "No_axis" denotes a leaf. */
enum Axes {X_axis, Y_axis, Z_axis, No_axis};

/* Declaration of the node of kd-tree */
struct KDTreeNode {
 Point3D min, max; /* extent of node ... six float values */
 GeomObjlist *objlist; /* list of enclosed objects */
 struct KDTreeNode *left; /* pointer to the left child */
 struct KDTreeNode *right; /* pointer to the right child */
 Axes axis; /* orientation of the splitting plane */
};

/* Entry for stack operation. */
struct StackElem {
 KDTreeNode* node;
 float a; /* entry signed distance (a) for the node */
 float b; /* exit signed distance (b) for the node */
};

/* Recursive ray traversal algorithm, which suffers from a lack of robustness. */
Object RayTravAlgRECA(KDTreeNode *rootNode, Ray ray)
{
   float a, b; /* entry/exit point signed distances */
   float t; /* signed distance to the splitting plane */

   /* intersect ray with sceneBox, find the entry and exit signed distance */
   RayBoxIntersect(ray, rootNode, &a, &b);

   if (ray does not intersect sceneBox)
      return ["No object"];

   /* stack to avoid recursive calls, required for efficiency */
   StackElem stack[MAXDEPTH]; /* MAXDEPTH could be 50 */
   int stackPtr = 0; /* pointer to the stack */
   /* pointers to the children node and current node */
   KDTreeNode *farChild, *nearChild, *currNode;

   /* push the initial values onto the stack */
   "store rootNode, a, b onto the stack and increment stackPtr"

   /* until we traversed through the whole kd-tree */
   while ( stack is not empty ) { /* stackPtr > 0 */
      /* pop values from the stack */
      "decrement stackPtr and retrieve currNode, a, and b from the stack"

      /* loop until a leaf is found */
```

```
    while (currNode is not a leaf) {
      /* current node is an interior node */

      /* for X_axis, Y_axis, Z_axis compute difference between position of splitting plane and ray origin */
      float diff = currNode->right.min[axis] - ray.origin[axis];
      /* the signed distance to splitting plane */
      t = diff / ray.dir[axis];

      /* NEGATIVE or POSITIVE cases? */
      /* the case ZERO is not recognized! */
      if (diff > 0.0) /* NEGATIVE */
      { nearChild = currNode->left;
        farChild = currNode->right;
      }
      else /* POSITIVE */
      { nearChild = currNode->right;
        farChild = currNode->left;
      } /* if */

      /* distinguish between cases 1, 3, 4, and 5, */
      /* but case 2 is not taken into account! */
      if ( (t > b) or (t < 0.0) )
        currNode = nearChild; /* case 3 or 1 */
      else {
        if (t < a )
          currNode = farChild; /* case 5 */
        else
        { /* case 4 – push */
          "store farNode, t, b onto the stack and increment stackPtr"

          /* select the near child for further traversal */
          currNode = nearChild;
         /* change the exit signed distance */
          b = t;
        } /* if */
      } /* if */
    } /* while */

    /* current node is the leaf . . . empty or full */

    "intersect ray with each object in the object list"
    "discarding those lying before (a) or farther than (b)"

    if (any intersection exists)
      return ["object with the closest intersection point"];
  } /* while ( stack is not empty) */

  /* if stack is empty, no intersection has been found */
  return ["No object"];
} /* RayTravAlgRECA */
```

Appendix C

C-pseudocode of recursive ray traversal algorithm TA^{B}_{rec} for the kd-tree.

```
/* Possible orientation of the splitting plane in the interior node of the kd-tree, */
/* "No_axis" denotes a leaf. */
enum Axes {X_axis, Y_axis, Z_axis, No_axis};

/* Declaration of the kd-tree node */
struct KDTNode {
 GeomObjlist *objlist; /* list of enclosed objects */
 struct KDTNode *left; /* pointer to the left child */
 struct KDTNode *right; /* pointer to the right child */
 Axes axis; /* orientation of the splitting plane */
 float splitPlane; /* position of the splitting plane */
};

/* Entry for stack operation. */
struct StackElem {
 KDTNode* node; /* pointer to far child */
 float t; /* the entry/exit signed distance */
 Point3D pb; /* the coordinates of entry/exit point */
 int prev; /* the pointer to the previous stack item */
};

/* Recursive ray traversal algorithm. */
Object RayTravAlgRECB(KDTNode *rootNode, Ray ray)
{
  float a, b; /* entry/exit signed distance */
  float t; /* signed distance to the splitting plane */

  /* intersect ray with sceneBox, find the entry and exit signed distance */
  RayBoxIntersect(ray, rootNode, &a, &b);

  if (ray does not intersect sceneBox )
    return ["No object"];

  /* stack required for traversal to store far children */
  StackElem stack[MAXDEPTH]; /* MAXDEPTH could be 50 */

  /* pointers to the far child node and current node */
  KDTNode *farChild, *currNode;
  currNode = root; /* start from the kd-tree root node */

  int enPt = 0; /* setup initial entry point ... enPt corresponds to pointer */
  stack[enPt].t = a; /* set the signed distance */

  /* distinguish between internal and external origin */
  if (a >= 0.0) /* a ray with external origin */
    stack[enPt].pb = ray.origin + ray.dir * a;
  else /* a ray with internal origin */
  stack[enPt].pb = ray.origin;

/* setup initial exit point in the stack */
int exPt = 1; /* pointer to the stack */
stack[exPt].t = b;
stack[exPt].pb = ray.origin + ray.dir * b;
stack[exPt].node = "nowhere"; /* set termination flag */

/* loop, traverse through the whole kd-tree, until an object is intersected or ray leaves the scene */
while (currNode does not point to "nowhere" ) {
  /* loop until a leaf is found */
  while (currNode is not a leaf) {
    /* retrieve position of splitting plane */
    float splitVal = currNode->splitPlane;

    /* similar code for all axes */
    /* nextAxis is x → y, y → z, z → x */
    /* prevAxis is z → x, y → x, y → z */

    if (stack[enPt].pb[axis] <= splitVal) {
      if (stack[exPt].pb[axis] <= splitVal)
      { /* case N1, N2, N3, P5, Z2, and Z3 */
        currNode = currNode->left;
        continue;
      }
      if (stack[exPt].pb[axis] == splitVal) {
        currNode = currNode->right;
        continue; /* case Z1 */
      } /* if */
      /* case N4 */
      farChild = currNode->right;
      currNode = currNode->left;
    }
    else { /* (stack[enPt].pb[axis] > splitVal) */
      if (splitVal < stack[exPt].pb[axis]) {
        /* case P1, P2, P3, and N5 */
        currNode = currNode->right;
        continue;
      } /* if */
      /* case P4*/
      farChild = currNode->left;
      currNode = currNode->right;
    } /* if */
    /* case P4 or N4 ... traverse both children */

    /* signed distance to the splitting plane */
    t = (splitVal - ray.origin[axis]) / ray.dir[axis];

    /* setup the new exit point */
    int tmp = exPt;
    Increment(exPt);
```

```
      /* possibly skip current entry point so not to overwrite the data */
      if (exPt == enPt)
        Increment(exPt);

      /* push values onto the stack */
      stack[exPt].prev = tmp;
      stack[exPt].t = t;
      stack[exPt].node = farChild;
      stack[exPt].pb[axis] = splitVal;
      stack[exPt].pb[nextAxis] = ray.origin[nextAxis] +
                                 t * ray.dir[nextAxis];
      stack[exPt].pb[prevAxis] = ray.origin[prevAxis] +
                                 t * ray.dir[prevAxis];
    } /* while */

    /* current node is the leaf ... empty or full */
    "intersect ray with each object in the object list, discarding "
    "those lying before stack[enPt].t or farther than stack[exPt].t"

    if ( any intersection exists )
      return ["object with closest intersection point"];

    /* pop from the stack */
    enPt = exPt; /* the signed distance intervals are adjacent */

    /* retrieve the pointer to the next node, it is possible that ray traversal terminates */
    currNode = stack[exPt].node;

    exPt = stack[enPt].prev;
  } /* while */

  /* currNode = ”nowhere”, ray leaves the scene */
  return ["No object"];
} /* RayTravAlgRECB */
```

Appendix D

C-pseudocodes concerning ray traversal algorithms for the *kd*-tree with neighbor-links.

Algorithm determining a single neighbor-link for one face of a leaf of the *kd*-tree.

```
/* Possible orientation of the splitting plane in the interior node of the kd-tree, */
/* "No_axis" denotes a leaf. */
enum Axes {X_axis, Y_axis, Z_axis, No_axis};

/* Denote the faces of kd-tree node */
enum Faces {FLeft, FRight, FFront, FBack, FBottom, FTop};

/* Declaration of the kd-tree node */
struct KDTNode {
 Point3D min, max; /* extent of node ... six float values */
 GeomObjlist *objlist; /* list of enclosed objects */
 struct KDTNode *left; /* pointer to the left child */
 struct KDTNode *right; /* pointer to the right child */
 /* links from faces to neighbors, either single neighbor-links or neighbor-links trees */
 struct KDTNode *flinks[6];
 Axes  axis; /* orientation of the splitting plane */
 float splitPlane; /* position of the splitting plane */
};

/* Given a node and one of its faces, find out the single neighbor-link */
KDTNode* FindSingleNeighborLink(KDTNode *node, Faces face, KDTNode *rootNode)
{
  KDTNode *currNode; /* currently accessed node */

  if (node->box[face] is coplanar with a face of sceneBox)
    /* ray leaves the scene from the face of given node */
    return ["No neighbor link exists"];

  /* stack required for traversal to store far child nodes */
  KDTNode *stack[MAXDEPTH]; /* MAXDEPTH could be 50 */

  /* search starts from the root node */
  stack.push(rootNode);

  /* loop until we find out the neighbor-link */
  while (stack is not empty) {
    /* get currNode node on the path */
    currNode = stack.pop();

    if (currNode is not leaf) {
      /* currNode node is interior one */
      if (currNode->axis == node->axis) {
          /* splitting plane parallel with face of node */
          if (node->box.GetExtent(face) < currNode->splittingPlane - epsilon)
            stack.push(currNode->left);
          else {
            if (node->box.GetExtent(face) > currNode->splittingPlane + epsilon)
              stack.push(currNode->right);
            else {
              /* the splitting plane underlying the face, select the opposite, */
              /* part of kd-tree that does not include node */
              if (face is a min face of the node)
                /* it was a min limit – go to the left */
                stack.push(currNode->left);
              else
                /* it was a max limit – go to the right */
                stack.push(currNode->right);
            } /* if */
          } /* if */
        }
        else {
          /* it is some other axis so test if it splits the face or not */
          if (node->min[currNode->axis] >= currNode->splittingPlane)
            /* greater, it avoids splitting the face of node */
            stack.push(currNode->right);
          else {
            if (node->max[currNode->axis] <= currNode->splittingPlane)
              /* smaller, it avoids splitting the face of node */
              stack.push(currNode->left);
          } /* if */
        } /* if axis ... */
      } /* if cuurent node is a leaf*/
    } /* while stack is not empty */

    /* neighbor-link is located, it points either to leaf or to the interior node */
    return ["currNode"];
} /*  FindSingleNeighborLink */
```

Algorithm replacing a single link by a neighbor-links tree decomposed into two procedures.

```
/* Given a node and face, possibly replaces indirect neighbor-link by corresponding neighbor-links tree */
void BuildNeighborLinksTree(KDTNode *node, Faces face)
{
 if (node->flinks[face] == NULL) return;
 if (node->flinks[face] is leaf) return;

 /* the neighbor-link points to an interior kd-tree node, */
 /* it will be replaced a neighbor-links tree, it has sense */
 node->flinks[face] = CreateNeighborLinkTree(node, node->flink[face], face);
} /* BuildNeighborLinksTree */
```

```
/* Always creates neighbor-links tree. */
KDTNode* CreateNeighborLinksTree(KDTNode *node, KDTNode *subtree, Faces face)
{
  /* current node of the kd-tree */
  KDTNode *currNode = subtree;

  while(currNode is not leaf) {
    /* currNode is an interior node, we are descending to a leaf */
    if (currNode->axis is perpendicular to face) {
      if (node->box[face] is to the left of currNode->splitPlane)
       currNode = currNode->left;
      else {
       if (node->box[face] is to the right of currNode->splitPlane)
         currNode = currNode->right;
       else {
         /* limits are equal make decision regarding to the oposite limit */
         if (face is a min face of currNode)
           currNode = currNode->left; /* it was a min limit – go left */
         else
           currNode = currNode->right; /* it was a max limit – go right */
       } /* if */
      } /* if */
    }
    else {
      /* it is some other axis so test if it splits the face or not */
      if (node->box is to the right of currNode->splitPlane)
       currNode = currNode->right; /* greater */
      else {
       if (node->box is to the left of currNode->splitPlane)
         currNode = currNode->left;   /* smaller */
       else {
         /* this node intersects the face – it must be added to the created neighbor-links tree */
         KDTNode *result = new KDTNode;
         result->splitPlane = currNode->splitPlane;
         result->axis = currNode->axis;
         /* recusively call itself to get whole neighbor-links tree */
         result->left = NextRopeTreeNode(node, currNode->left, face);
         result->right = NextRopeTreeNode(node, currNode->right, face);
         return result;
       } /* if */
      } /* if axis .. */
    } /* if perpendicular to face */
  } /* while curr node is not a leaf */

  /* return the leaf node of neigbhour-links tree */
  return currNode;
} /* CreateNeighborLinksTree */
```

Ray traversal algorithm for the *kd*-tree with single neighbor-links and neighbor-links trees.

```
/* Ray traversal algorithm for neighbor-links, */
/* leaf node where ray origin is located may be specified, if known. */
Object RayTravAlgNL(Ray ray, KDTNode *leafOrigin, KDTNode *rootNode)
{
  KDTNode *currLeaf = leafOrigin; /* currently accessed leaf node of kd-tree */
  Point3D exitPoint; /* exit point of current leaf node */

  if (currLeaf is not valid leaf od kd-tree) {
    float a, b; /* entry/exit point signed distances */

    /* intersect ray with sceneBox, find the entry and exit signed distance */
    RayBoxIntersect(ray, rootNode, &a, &b);

    if (ray does not intersect sceneBox)
      return ["No object"];

    /* entry intersection point of ray and sceneBox */
    exitPoint = ray.origin + ray.dir * (a + epsilon);

    /* locate the first leaf along the ray path */
    currLeaf = LocateLeaf(rootNode, exitPoint);
  }

  /* traverse through whole kd-tree until the object is intersected or the ray leaves the scene */
  while (true) { /* endless loop */
    if (currLeaf is not empty leaf) {
      "intersect ray with each primitive in the object list"
      "discarding those lying outside the leaf bounding box"

      if (any intersection exists)
       return ["object with the closest intersection point"];
    } /* if */

    /* exit-face determination for exit intersection point */
    nextExitFace = GetExitFace(currNode, ray, exitPoint);

    if (currLeaf.flinks[nextExitFace] does point outside the sceneBox)
      return ["No object"]; /* the ray exits the scene */

    if (currLeaf.flinks[nextExitFace] is a leaf)
      currLeaf = currLeaf.flinks[nextExitFace];
    else
      currLeaf = LocateLeaf(currLeaf.flinks[nextExitFace], exitPoint);
  } /* while */
} /* RayTravAlgNL */
```

Appendix E

The experimental results for *SPD* scenes are presented here primarily for testing procedure TP_D. The experiments for setting described on lines 0–47 were conducted on PC, Intel Pentium II MMX, 466Mhz, 128MB RAM, running *Linux* operating system (kernel version 2.2.12-20), egcs-1.1.2 compiler release, optimization switches "-O2". In addition, for lines 33–47 compiler setting "-DNDEBUG" was used (for this reason the results for line 45 and 13 differ in subset Θ). The results are presented in tabular form using the minimum testing output (described in Chapter 2). The results for each out of 30 *SPD* scenes are presented, with several invariants of the measurement related to TP_D:

$N^{hit}_{\mathcal{AB}}$:	number of primary rays intersecting the scene axis-aligned bounding box,
N^{hit}_{prim}:	number of primary rays intersecting an object,
N_{sec}:	number of secondary rays (reflected rays and refracted rays),
N^{hit}_{sec}:	number of secondary rays intersecting an object (reflected and refracted rays),
N_{shad}:	number of shadow rays,
N^{hit}_{shad}:	number of shadow rays hitting opaque objects,
T^{MIN}_{R}:	minimum application running time,
T_{app}:	remaining application time,
T^{MIN}_{RSA}:	ideal ray shooting time. (See Chapter 2 for details.)

Several other scene characteristics are given in Table 3.2, Chapter 3.

In experiments for the *kd*-trees built for special ray sets (*parallel*, *perspective*, *spherical*), these invariants do not hold, we do not state them here. Other settings used in Tables:

Line 0: "naïve *RSA*" (for several experiments, integral counters overflew; the results are not fully reported),

Line 48: BVH – bounding volume hierarchy (Subsection 3.5.2 and Subsection 1.6.2),

Line 49: O84 – octree (Subsection 3.5.2 and Subsubsection 1.6.3.2),

Line 50: O89 – octree (Subsection 3.5.2 and Subsubsection 1.6.3.2),

Line 51: BSP – binary space partitioning tree (Subsection 3.5.2 and Subsubsection 1.6.3.1),

Line 52: O93 – octree (Subsection 3.5.2 and Subsubsection 1.6.3.2),

Line 53: UG – uniform grid (Subsection 3.5.2 and Subsubsection 1.6.3.3),

Line 54: AG – adaptive grid (Subsection 3.5.2 and Subsubsection 1.6.3.4),

Line 55: HUG – hierarchy of uniform grids (Subsection 3.5.2 and Subsubsection 1.6.3.4),

Line 56: RG – recursive grids (Subsection 3.5.2 and Subsubsection 1.6.3.4),

Line 57: O84A – Octree-R (Subsection 3.5.2 and Subsubsection 1.6.3.2),

Line 58: O93A – Octree-R (Subsection 3.5.2 and Subsubsection 1.6.3.2),

Line 59: KD – *kd*-tree (Subsection 3.5.2, Chapter 4, and Chapter 5),

Mnemonic Notation for Tables in Appendix E

The setting for the experiments is described in two Sections. In Section 4.10, lines 1–32, the setting is described for *kd*-tree construction algorithms. In Section 5.5, lines 33–47, the setting is described for ray traversal algorithms. The second column in the tables gives mnemonic symbols for the setting used in the experiments, where the following symbols are used:

(atc):	automatic termination criteria (Section 4.5),
(d_{max}, N_{max}):	ad hoc termination criteria with maximum leaf depth d_{max} and maximum number of objects in a leaf N_{max} (Subsubsection 4.2.4.1),
objmed:	object median subdivision (Subsection 4.2.2),
spatmed:	spatial median subdivision (*i.e.* BSP tree, Subsection 4.2.2),
xyz:	cyclical change of splitting plane orientation starting with the *x*-axis (Subsection 4.2.1),
GCM,GCM2,GCM3:	general cost model (Section 4.7),
LC:	late empty space cutting off (Subsection 4.4.3),
OSAH:	ordinary surface area heuristic (Subsection 4.2.2),
PAR:	ray set induced by parallel projection (Section 4.8),
PARSAH:	parallel surface area heuristic (Subsection 4.8.1),
PER:	ray set induced by perspective projection (Section 4.8),
PERSAH:	perspective surface area heuristic (Subsection 4.8.2),
RMI:	search restricted to median interval (Subsubsection 4.2.3.4),
SPH:	ray set induced by spherical projection (Section 4.8),
SPHSAH:	spherical surface area heuristic (Subsection 4.8.3),
TA_{seq}:	sequential ray traversal algorithm (Subsection 5.3.1),
TA^{A}_{rec}:	slower recursive ray traversal algorithm (Subsection 5.3.2),
TA^{B}_{rec}:	faster recursive ray traversal algorithm (Section 5.4),
TA_{SNL}:	ray traversal algorithm with single neighbour-links (Section 5.4),
TA_{NLT}:	ray traversal algorithm with neighbour-links trees (Section 5.4),
TPC:	two-plane empty space cutting off (Subsection 4.4.4).

If not stated explicitly by PAR, PER, or SPH, the distribution of rays was induced by the testing procedure TP_D. The same holds for the ray traversal algorithm. If not stated explicitly, the recursive ray traversal algorithm TA^{B}_{rec} is used. Symbol "–" denotes unknown value.

Scene = “*balls3*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	821	–	0	0	0	0.00	314.77	12.07	1.0	749.45
1	spatmed-xyz(16,2)	526	527	219	2373	78.07	57.46	11.02	5.83	0.02	27.02	12.07	0.42	52.26
2	objmed-xyz(16,2)	4323	4324	185	8025	56.10	66.82	14.68	0.96	0.13	28.28	12.07	0.31	55.26
3	objmed(16,2)	3829	3830	110	7192	61.00	63.32	14.92	0.43	0.16	29.44	12.07	0.34	58.02
4	OSAH(16,2)	1569	1570	295	2608	9.78	19.56	3.98	1.01	0.13	12.66	12.07	0.21	18.07
5	OSAH-RMI(16,2)	1569	1570	295	2608	9.78	19.56	3.98	1.01	0.13	12.26	12.07	0.21	17.12
6	OSAH-xyz(16,2)	1779	1780	306	2964	9.68	21.87	4.64	1.74	0.07	12.49	12.07	0.19	17.67
7	OSAH(8,1)	65	66	3	989	64.44	11.36	2.69	0.17	0.06	18.20	12.07	0.75	31.26
8	OSAH(8,2)	64	65	3	988	64.45	11.36	2.69	0.17	0.06	18.01	12.07	0.75	30.81
9	OSAH(16,1)	2411	2412	663	3076	9.01	21.47	4.34	1.24	0.13	12.64	12.07	0.18	18.02
10	OSAH(16,2)	1569	1570	295	2608	9.78	19.56	3.98	1.01	0.13	12.66	12.07	0.21	18.07
11	OSAH(24,1)	9063	9064	1120	10930	8.30	24.15	4.78	1.38	0.30	13.22	12.07	0.16	19.40
12	OSAH(24,2)	3667	3668	340	6321	9.49	20.51	4.13	1.03	0.18	12.54	12.07	0.20	17.79
13	OSAH(atc)	767	768	158	1674	12.38	18.57	3.87	0.96	0.09	12.61	12.07	0.26	17.95
14	OSAH2(atc)	1098	1099	145	2154	11.91	22.26	4.39	1.23	0.13	13.02	12.07	0.22	18.93
15	OSAH+LC(atc)	790	791	158	1732	12.38	18.57	3.87	0.96	0.09	12.61	12.07	0.26	17.95
16	OSAH+TPC(atc)	732	733	148	1646	12.39	18.51	3.86	0.96	0.11	12.56	12.07	0.26	17.83
17	OSAH+TPC+LC(atc)	813	814	148	1829	12.38	18.57	3.87	0.96	0.13	11.60	12.07	0.26	15.55
18	OSAH+LC(16,1)	2411	2412	663	3076	9.01	21.47	4.34	1.24	0.17	11.72	12.07	0.18	15.83
19	OSAH+TPC(16,1)	2409	2410	665	3074	9.01	21.48	4.34	1.24	0.17	11.82	12.07	0.18	16.07
20	OSAH+TPC+LC(16,1)	2409	2410	665	3074	9.01	21.48	4.34	1.24	0.17	11.75	12.07	0.18	15.90
21	OSAH+PR(atc)	767	768	158	1492	11.97	18.44	3.86	0.98	0.12	11.83	12.07	0.25	16.10
22	OSAH+SC(atc)	720	721	202	1468	11.22	18.29	3.85	1.19	0.15	12.03	12.07	0.25	16.57
23	OSAH+GCM(atc)	906	907	182	1859	11.95	19.54	4.05	1.05	2.52	12.39	12.07	0.25	17.43
24	OSAH+GCM2(atc)	1535	1536	87	3695	14.25	22.71	4.58	0.92	4.65	13.35	12.07	0.25	19.71
25	OSAH+GCM3(atc)	791	792	76	2040	19.44	21.63	4.13	0.58	2.38	13.81	12.07	0.32	20.81
26	OSAH+PAR(atc)	767	768	158	1674	6.08	21.25	4.84	1.39	0.10	4.16	13.00	0.36	7.80
27	PARSAH+PAR(atc)	812	813	174	1723	5.98	21.33	4.83	1.38	0.14	4.15	13.00	0.35	7.75
28	OSAH+PER(atc)	767	768	158	1674	5.93	21.06	4.84	1.56	0.11	4.23	12.43	0.39	7.71
29	PERSAH+PER(atc)	1241	1242	284	2142	5.17	23.85	5.56	1.90	24.21	4.33	12.43	0.35	8.19
30	SPHSAH+PER(atc)	702	703	129	1688	6.98	20.78	4.82	1.12	0.78	4.33	12.43	0.43	8.19
31	OSAH+SPH(atc)	767	768	158	1674	5.86	20.74	4.76	1.53	0.11	4.50	12.04	0.43	7.52
32	SPHSAH+SPH(atc)	702	703	129	1688	6.90	20.50	4.75	1.10	0.78	4.62	12.04	0.48	8.04
33	OSAH+TA$_{seq}$(16,2)	1569	1570	295	2608	10.30	46.11	4.10	1.01	0.10	16.31	12.07	0.12	26.76
34	OSAH+TA$^{A}_{rec}$(16,2)	1569	1570	295	2608	9.78	19.56	3.98	1.01	0.10	12.55	12.07	0.18	17.81
35	OSAH+TA$^{B}_{rec}$(16,2)	1569	1570	295	2608	9.78	19.56	3.98	1.01	0.10	11.47	12.07	0.21	15.24
36	OSAH+TA$_{SNL}$(16,2)	1569	1570	295	2608	10.30	18.24	4.11	1.01	0.11	12.06	12.07	0.19	16.64
37	OSAH+TA$_{NLT}$(16,2)	10510	1570	295	2608	10.30	16.54	4.10	1.01	0.14	12.33	12.07	0.19	17.29
38	OSAH+TA$_{seq}$(18,2)	2196	2197	337	3492	10.03	48.44	4.20	1.03	0.12	16.65	12.07	0.11	27.57
39	OSAH+TA$^{A}_{rec}$(18,2)	2196	2197	337	3492	9.51	20.11	4.07	1.03	0.12	12.72	12.07	0.17	18.21
40	OSAH+TA$^{B}_{rec}$(18,2)	2196	2197	337	3492	9.51	20.11	4.07	1.03	0.11	11.58	12.07	0.20	15.50
41	OSAH+TA$_{SNL}$(18,2)	2196	2197	337	3492	10.04	18.76	4.20	1.03	0.16	12.11	12.07	0.18	16.76
42	OSAH+TA$_{NLT}$(18,2)	14308	2197	337	3492	10.03	16.97	4.20	1.03	0.18	12.20	12.07	0.18	16.98
43	OSAH+TA$_{seq}$(atc)	767	768	158	1674	12.92	42.39	3.97	0.96	0.08	16.16	12.07	0.15	26.40
44	OSAH+TA$^{A}_{rec}$(atc)	767	768	158	1674	12.38	18.57	3.87	0.96	0.09	12.64	12.07	0.23	18.02
45	OSAH+TA$^{B}_{rec}$(atc)	767	768	158	1674	12.38	18.57	3.87	0.96	0.08	11.67	12.07	0.26	15.71
46	OSAH+TA$_{SNL}$(atc)	767	768	158	1674	12.92	17.25	3.97	0.96	0.09	12.15	12.07	0.24	16.86
47	OSAH+TA$_{NLT}$(atc)	5207	768	158	1674	12.92	15.72	3.97	0.96	0.09	12.23	12.07	0.24	17.05
48	BVH	106	520	0	821	37.19	25.89	17.40	0.00	0.05	197.49	–	–	–
49	O84	262	1835	853	4500	69.15	119.30	19.56	12.16	0.04	73.33	–	–	–
50	O89	262	1835	853	4500	69.34	76.73	19.55	12.16	0.03	50.37	–	–	–
51	BSP	526	527	219	2373	78.07	57.46	11.02	5.83	0.85	82.40	–	–	–
52	O93	262	1835	853	4500	68.21	66.33	26.33	19.02	0.03	68.55	–	–	–
53	UG	0	4107	2720	2502	141.67	5.21	5.21	2.58	0.03	45.45	–	–	–
54	AG	22	2754	1257	3734	17.88	7.07	6.28	3.13	0.08	23.18	–	–	–
55	HUG	14	852	366	1436	681.67	6.48	3.37	1.33	0.03	281.01	–	–	–
56	RG	127	2922	577	6856	25.92	10.31	7.36	2.63	0.03	28.96	–	–	–
57	O84A	1350	9451	1418	15373	13.09	41.51	8.61	4.60	0.19	32.29	–	–	–
58	KD	1569	1570	295	2608	9.78	19.56	3.98	1.01	0.30	21.40	–	–	–

Table 1: Experimental results for scene “*balls3*”.

$$N = 821,$$
$$TP_D:\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 263169,\quad N^{hit}_{prim} = 263169,\quad N_{sec} = 151293,\quad N^{hit}_{sec} = 104440,$$
$$N_{shad} = 921077,\quad N^{hit}_{shad} = 236800,\quad T^{MIN}_{R}[s] = 5.49,\quad T_{app}[s] = 5.07,\quad T^{MIN}_{RSA}[s] = 0.42.$$

Scene = “*gears2*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	1169	–	0	0	0	0.00	1166.83	6.82	1.0	783.11
1	spatmed-xyz(16,2)	2247	2248	156	9196	13.79	23.38	3.95	1.20	0.06	27.75	6.82	0.49	11.81
2	objmed-xyz(16,2)	6356	6357	490	13596	19.66	46.04	8.95	4.07	0.20	36.85	6.82	0.41	17.91
3	objmed(16,2)	8543	8544	619	15906	45.29	66.50	13.09	0.62	0.32	61.04	6.82	0.52	34.15
4	OSAH(16,2)	3295	3296	40	7546	7.55	17.00	2.98	0.53	0.19	23.71	6.82	0.42	9.09
5	OSAH-RMI(16,2)	3295	3296	40	7546	7.55	17.00	2.98	0.53	0.19	23.45	6.82	0.42	8.92
6	OSAH-xyz(16,2)	2157	2158	90	5325	8.07	20.78	4.04	1.70	0.10	24.74	6.82	0.38	9.79
7	OSAH(8,1)	90	91	11	1386	40.52	12.38	2.54	0.51	0.07	35.40	6.82	0.84	16.94
8	OSAH(8,2)	90	91	11	1386	40.52	12.38	2.54	0.51	0.07	35.36	6.82	0.84	16.91
9	OSAH(16,1)	5624	5625	165	9338	6.10	18.54	3.18	0.56	0.23	22.69	6.82	0.34	8.41
10	OSAH(16,2)	3295	3296	40	7546	7.55	17.00	2.98	0.53	0.19	23.71	6.82	0.42	9.09
11	OSAH(24,1)	14969	14970	1021	21134	5.70	19.30	3.34	0.60	0.49	22.90	6.82	0.32	8.55
12	OSAH(24,2)	6977	6978	190	14573	7.31	17.26	3.02	0.54	0.30	22.97	6.82	0.40	8.60
13	OSAH(atc)	3297	3298	78	6229	6.57	17.89	3.10	0.55	0.18	22.73	6.82	0.37	8.44
14	OSAH2(atc)	3265	3266	159	6267	6.17	21.09	3.71	1.34	0.24	23.98	6.82	0.32	9.28
15	OSAH+LC(atc)	3842	3843	78	7629	6.50	18.10	3.12	0.55	0.22	23.04	6.82	0.36	8.64
16	OSAH+TPC(atc)	2412	2413	81	5095	6.87	17.56	3.07	0.56	0.17	23.04	6.82	0.38	8.64
17	OSAH+TPC+LC(atc)	3912	3913	86	8016	6.59	18.06	3.12	0.56	0.28	21.80	6.82	0.37	7.81
18	OSAH+LC(16,1)	5626	5627	167	9338	6.10	18.54	3.18	0.56	0.30	21.49	6.82	0.34	7.60
19	OSAH+TPC(16,1)	5620	5621	175	9334	6.12	18.58	3.20	0.56	0.29	21.78	6.82	0.34	7.80
20	OSAH+TPC+LC(16,1)	5622	5623	177	9334	6.12	18.58	3.20	0.56	0.30	22.70	6.82	0.34	8.42
21	OSAH+PR(atc)	3297	3298	78	5691	6.40	17.90	3.11	0.55	0.55	22.59	6.82	0.36	8.34
22	OSAH+SC(atc)	3302	3303	81	6182	6.57	17.91	3.10	0.55	0.29	23.09	6.82	0.37	8.68
23	OSAH+GCM(atc)	4934	4935	109	8655	6.85	19.61	3.48	0.43	15.45	23.41	6.82	0.36	8.89
24	OSAH+GCM2(atc)	8020	8021	240	14595	7.81	22.92	3.79	0.50	40.60	24.54	6.82	0.35	9.65
25	OSAH+GCM3(atc)	3491	3492	228	6697	9.48	26.39	4.37	0.95	11.54	26.00	6.82	0.36	10.63
26	OSAH+PAR(atc)	3297	3298	78	6229	2.34	11.60	2.05	0.21	0.21	5.86	8.02	0.70	3.25
27	PARSAH+PAR(atc)	3889	3890	136	6925	2.24	11.86	2.09	0.27	0.26	5.70	8.02	0.66	2.94
28	OSAH+PER(atc)	3297	3298	78	6229	2.66	13.47	2.17	0.25	0.21	5.61	8.17	0.65	3.52
29	PERSAH+PER(atc)	4046	4047	170	7290	2.49	13.41	2.31	0.46	21.93	5.40	8.17	0.60	3.08
30	SPHSAH+PER(atc)	4264	4265	634	9174	4.51	18.20	3.02	0.79	1.04	5.87	8.17	0.59	4.06
31	OSAH+SPH(atc)	3297	3298	78	6229	2.66	13.30	2.14	0.24	0.21	5.79	8.02	0.64	3.12
32	SPHSAH+SPH(atc)	4264	4265	634	9174	4.50	18.13	3.02	0.81	1.04	6.21	8.02	0.61	3.92
33	OSAH+TA$_{seq}$(16,2)	3295	3296	40	7546	10.58	31.36	3.01	0.53	0.15	26.30	6.82	0.31	10.83
34	OSAH+TA$^{A}_{rec}$(16,2)	3295	3296	40	7546	7.55	17.00	2.98	0.53	0.15	23.41	6.82	0.37	8.89
35	OSAH+TA$^{B}_{rec}$(16,2)	3295	3296	40	7546	7.55	17.00	2.98	0.53	0.16	21.93	6.82	0.42	7.90
36	OSAH+TA$_{SNL}$(16,2)	3295	3296	40	7546	10.58	15.41	3.01	0.53	0.19	21.77	6.82	0.43	7.79
37	OSAH+TA$_{NLT}$(16,2)	14579	3296	40	7546	10.58	14.60	3.01	0.53	0.21	23.19	6.82	0.38	8.74
38	OSAH+TA$_{seq}$(18,2)	4799	4800	98	10190	10.26	31.99	3.03	0.53	0.20	26.54	6.82	0.29	10.99
39	OSAH+TA$^{A}_{rec}$(18,2)	4799	4800	98	10190	7.33	17.20	3.01	0.53	0.16	23.55	6.82	0.36	8.99
40	OSAH+TA$^{B}_{rec}$(18,2)	4799	4800	98	10190	7.33	17.20	3.01	0.53	0.19	21.87	6.82	0.41	7.86
41	OSAH+TA$_{SNL}$(18,2)	4799	4800	98	10190	10.26	15.58	3.04	0.53	0.25	21.71	6.82	0.42	7.75
42	OSAH+TA$_{NLT}$(18,2)	20364	4800	98	10190	10.26	14.75	3.04	0.53	0.30	22.15	6.82	0.40	8.05
43	OSAH+TA$_{seq}$(atc)	3297	3298	78	6229	9.24	33.53	3.13	0.55	0.14	26.44	6.82	0.26	10.93
44	OSAH+TA$^{A}_{rec}$(atc)	3297	3298	78	6229	6.57	17.89	3.10	0.55	0.16	23.36	6.82	0.32	8.86
45	OSAH+TA$^{B}_{rec}$(atc)	3297	3298	78	6229	6.57	17.89	3.10	0.55	0.15	21.54	6.82	0.37	7.64
46	OSAH+TA$_{SNL}$(atc)	3297	3298	78	6229	9.24	16.20	3.13	0.55	0.20	21.62	6.82	0.37	7.69
47	OSAH+TA$_{NLT}$(atc)	14410	3298	78	6229	9.24	15.09	3.13	0.55	0.22	21.80	6.82	0.36	7.81
48	BVH	143	762	0	1169	37.59	35.70	27.66	0.00	0.07	444.67	–	–	–
49	O84	1185	8296	2820	17960	11.36	27.93	5.65	2.81	0.09	48.97	–	–	–
50	O89	1185	8296	2820	17960	10.64	21.48	5.49	2.81	0.07	41.43	–	–	–
51	BSP	2247	2248	156	9196	13.79	23.38	3.95	1.20	1.15	121.18	–	–	–
52	O93	1357	9500	3508	19636	9.74	20.46	7.95	5.55	0.09	49.88	–	–	–
53	UG	0	5887	4371	4121	18.95	8.42	8.42	5.56	0.04	40.22	–	–	–
54	AG	116	5122	0	10112	9.41	4.41	3.40	0.00	0.23	45.39	–	–	–
55	HUG	15	3480	0	8100	36.82	4.95	1.69	0.00	0.08	73.80	–	–	–
56	RG	347	6374	2464	18917	20.30	8.36	6.42	3.18	0.06	50.71	–	–	–
57	O84A	4349	30444	7658	53542	8.60	30.98	6.23	3.33	0.57	49.40	–	–	–
58	KD	3295	3296	40	7546	7.55	17.00	2.98	0.53	0.60	38.96	–	–	–

Table 2: Experimental results for scene “*gears2*”.

$$N = 1169,$$
$$TP_D:\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 263169,\quad N^{hit}_{prim} = 243148,\quad N_{sec} = 270124,\quad N^{hit}_{sec} = 191717,$$
$$N_{shad} = 1565640,\quad N^{hit}_{shad} = 369157,\quad T^{MIN}_{R}[s] = 11.65,\quad T_{app}[s] = 10.16,\quad T^{MIN}_{RSA}[s] = 1.49.$$

Scene = “*jacks3*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	657	–	0	0	0	0.00	155.50	7.65	1.0	676.09
1	spatmed-xyz(16,2)	9091	9092	256	21468	34.87	32.72	6.31	0.84	0.08	9.86	7.65	0.49	35.22
2	objmed-xyz(16,2)	5032	5033	300	9540	37.03	35.27	6.95	0.76	0.13	9.64	7.65	0.49	34.26
3	objmed(16,2)	4879	4880	176	9517	39.74	34.32	6.81	0.23	0.20	10.18	7.65	0.51	36.61
4	OSAH(16,2)	2678	2679	258	4614	22.40	26.50	5.08	1.27	0.14	7.32	7.65	0.44	24.17
5	OSAH-RMI(16,2)	2678	2679	258	4614	22.40	26.50	5.08	1.27	0.14	7.27	7.65	0.44	23.96
6	OSAH-xyz(16,2)	2762	2763	243	4814	22.34	25.46	4.92	1.08	0.09	7.03	7.65	0.44	22.91
7	OSAH(8,1)	209	210	49	1270	56.62	14.92	3.12	1.01	0.05	8.96	7.65	0.78	31.30
8	OSAH(8,2)	196	197	36	1270	57.65	14.55	3.04	0.76	0.05	9.03	7.65	0.78	31.61
9	OSAH(16,1)	5499	5500	1108	6546	16.53	32.57	6.09	2.55	0.20	7.11	7.65	0.32	23.26
10	OSAH(16,2)	2678	2679	258	4614	22.40	26.50	5.08	1.27	0.14	7.32	7.65	0.44	24.17
11	OSAH(24,1)	10584	10585	1184	13622	16.39	35.87	6.59	2.57	0.33	7.42	7.65	0.29	24.61
12	OSAH(24,2)	4611	4612	258	8617	22.78	28.08	5.33	1.27	0.20	7.52	7.65	0.43	25.04
13	OSAH(atc)	1687	1688	630	2330	22.07	24.37	4.71	2.24	0.11	6.78	7.65	0.45	21.83
14	OSAH2(atc)	2636	2637	442	3706	20.74	28.02	5.33	1.82	0.18	7.02	7.65	0.40	22.87
15	OSAH+LC(atc)	2658	2659	631	4607	22.63	27.12	5.21	2.24	0.15	7.25	7.65	0.43	23.87
16	OSAH+TPC(atc)	1054	1055	358	1937	26.42	21.64	4.21	1.87	0.10	6.86	7.65	0.53	22.17
17	OSAH+TPC+LC(atc)	2347	2348	360	4913	26.59	25.89	4.98	1.87	0.18	7.31	7.65	0.48	24.13
18	OSAH+LC(16,1)	5499	5500	1108	6546	16.53	32.57	6.09	2.55	0.24	6.81	7.65	0.32	21.96
19	OSAH+TPC(16,1)	5506	5507	1106	6561	16.54	32.57	6.08	2.54	0.24	6.84	7.65	0.32	22.09
20	OSAH+TPC+LC(16,1)	5506	5507	1106	6561	16.54	32.57	6.08	2.54	0.26	6.93	7.65	0.32	22.48
21	OSAH+PR(atc)	1687	1688	630	1656	17.82	24.45	4.73	2.24	0.18	6.09	7.65	0.40	18.83
22	OSAH+SC(atc)	1990	1991	1161	1512	9.78	25.85	5.00	3.26	0.18	5.27	7.65	0.26	15.26
23	OSAH+GCM(atc)	2591	2592	715	3393	17.95	28.13	5.43	2.43	7.04	6.75	7.65	0.37	21.70
24	OSAH+GCM2(atc)	5517	5518	195	10223	24.69	33.83	6.24	1.38	15.02	8.17	7.65	0.40	27.87
25	OSAH+GCM3(atc)	3810	3811	450	5965	23.66	32.18	5.88	1.63	10.89	7.68	7.65	0.40	25.74
26	OSAH+PAR(atc)	1687	1688	630	2330	12.32	15.37	3.08	1.89	0.12	1.90	7.00	0.37	12.00
27	PARSAH+PAR(atc)	3806	3807	274	11374	4.90	5.91	0.51	0.15	0.34	1.35	7.00	0.55	6.50
28	OSAH+PER(atc)	1687	1688	630	2330	13.23	18.32	3.64	2.29	0.12	2.40	7.31	0.37	11.15
29	PERSAH+PER(atc)	1765	1766	580	3129	8.99	11.66	2.16	1.34	30.37	2.02	7.31	0.46	8.23
30	SPHSAH+PER(atc)	3251	3252	802	8140	22.89	16.82	3.32	1.70	1.56	2.83	7.31	0.56	14.46
31	OSAH+SPH(atc)	1687	1688	630	2330	13.22	18.25	3.63	2.28	0.12	2.75	7.86	0.45	11.79
32	SPHSAH+SPH(atc)	3251	3252	802	8140	22.80	16.73	3.31	1.69	1.58	3.12	7.86	0.59	14.43
33	OSAH+TA$_{seq}$(16,2)	2678	2679	258	4614	22.88	64.46	5.15	1.27	0.12	9.42	7.65	0.29	33.30
34	OSAH+TA$^A_{rec}$(16,2)	2678	2679	258	4614	22.40	26.50	5.08	1.27	0.10	7.55	7.65	0.38	25.17
35	OSAH+TA$^B_{rec}$(16,2)	2678	2679	258	4614	22.40	26.50	5.08	1.27	0.12	6.78	7.65	0.44	21.83
36	OSAH+TA$_{SNL}$(16,2)	2678	2679	258	4614	22.89	22.08	5.15	1.27	0.16	7.74	7.65	0.37	26.00
37	OSAH+TA$_{NLT}$(16,2)	18243	2679	258	4614	22.89	18.53	5.15	1.27	0.16	7.75	7.65	0.37	26.04
38	OSAH+TA$_{seq}$(18,2)	3436	3437	258	6114	23.19	68.50	5.32	1.27	0.14	9.72	7.65	0.28	34.61
39	OSAH+TA$^A_{rec}$(18,2)	3436	3437	258	6114	22.66	27.44	5.24	1.27	0.13	7.84	7.65	0.36	26.43
40	OSAH+TA$^B_{rec}$(18,2)	3436	3437	258	6114	22.66	27.44	5.24	1.27	0.14	6.91	7.65	0.43	22.39
41	OSAH+TA$_{SNL}$(18,2)	3436	3437	258	6114	23.19	22.86	5.32	1.27	0.19	7.86	7.65	0.36	26.52
42	OSAH+TA$_{NLT}$(18,2)	22027	3437	258	6114	23.19	19.13	5.32	1.27	0.21	7.80	7.65	0.37	26.26
43	OSAH+TA$_{seq}$(atc)	1687	1688	630	2330	22.59	55.46	4.76	2.24	0.10	8.59	7.65	0.31	29.70
44	OSAH+TA$^A_{rec}$(atc)	1687	1688	630	2330	22.07	24.37	4.71	2.24	0.09	7.11	7.65	0.39	23.26
45	OSAH+TA$^B_{rec}$(atc)	1687	1688	630	2330	22.07	24.37	4.71	2.24	0.10	6.41	7.65	0.45	20.22
46	OSAH+TA$_{SNL}$(atc)	1687	1688	630	2330	22.59	20.56	4.76	2.24	0.12	7.13	7.65	0.39	23.35
47	OSAH+TA$_{NLT}$(atc)	12617	1688	630	2330	22.59	17.43	4.76	2.24	0.14	7.24	7.65	0.38	23.83
48	BVH	73	383	0	657	63.43	25.04	17.77	0.00	0.05	79.72	–	–	–
49	O84	4143	29002	2046	56486	36.77	48.72	8.48	1.96	0.18	18.58	–	–	–
50	O89	4143	29002	2046	56486	36.74	33.47	8.48	1.96	0.14	15.49	–	–	–
51	BSP	9091	9092	256	21468	34.87	32.72	6.31	0.84	0.31	15.94	–	–	–
52	O93	4143	29002	2046	56486	35.94	32.41	14.45	8.04	0.17	18.15	–	–	–
53	UG	0	3120	1724	4698	36.58	7.05	7.05	3.58	0.03	10.46	–	–	–
54	AG	36	1929	610	3926	48.04	11.06	8.95	2.00	0.12	21.12	–	–	–
55	HUG	75	1866	988	2652	51.83	13.29	8.99	3.80	0.04	19.85	–	–	–
56	RG	147	2130	336	6782	48.27	7.41	5.44	1.64	0.03	14.01	–	–	–
57	O84A	3929	27504	1226	58938	31.80	42.49	7.72	2.00	0.41	16.56	–	–	–
58	KD	2678	2679	258	4614	22.40	26.50	5.08	1.27	0.43	10.63	–	–	–

Table 3: Experimental results for scene “*jacks3*”.

$$N = 657,$$
$$TP_D\colon \quad N_{prim} = 263169, \quad N^{hit}_{\mathcal{AB}} = 191115, \quad N^{hit}_{prim} = 72548, \quad N_{sec} = 119316, \quad N^{hit}_{sec} = 49502,$$
$$N_{shad} = 97328, \quad N^{hit}_{shad} = 33063, \quad T^{MIN}_R[s] = 1.99, \quad T_{app}[s] = 1.76, \quad T^{MIN}_{RSA}[s] = 0.23.$$

Scene = “*lattice6*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	1225	–	0	0	0	0.01	1313.17	8.15	1.0	1193.79
1	spatmed-xyz(16,2)	9791	9792	0	22056	16.02	35.05	6.21	0.00	0.12	32.80	8.15	0.60	21.67
2	objmed-xyz(16,2)	2680	2681	388	4494	11.21	29.70	5.32	0.88	0.11	24.92	8.15	0.56	14.51
3	objmed(16,2)	1944	1945	0	3818	13.91	28.46	5.26	0.00	0.13	27.86	8.15	0.62	17.18
4	OSAH(16,2)	2171	2172	200	3736	10.37	26.82	4.88	0.72	0.14	25.46	8.15	0.56	15.00
5	OSAH-RMI(16,2)	2171	2172	200	3736	10.37	26.82	4.88	0.72	0.14	25.04	8.15	0.56	14.62
6	OSAH-xyz(16,2)	2037	2038	95	3743	10.98	25.92	4.69	0.36	0.09	25.06	8.15	0.58	14.64
7	OSAH(8,1)	255	256	1	2019	37.03	15.76	3.28	0.01	0.08	46.11	8.15	0.89	33.77
8	OSAH(8,2)	255	256	1	2019	37.03	15.76	3.28	0.01	0.08	46.14	8.15	0.89	33.80
9	OSAH(16,1)	8419	8420	682	9502	4.96	35.70	6.11	2.54	0.26	23.35	8.15	0.32	13.08
10	OSAH(16,2)	2171	2172	200	3736	10.37	26.82	4.88	0.72	0.14	25.46	8.15	0.56	15.00
11	OSAH(24,1)	8944	8945	682	10027	4.94	35.73	6.11	2.54	0.28	23.20	8.15	0.31	12.95
12	OSAH(24,2)	2171	2172	200	3736	10.37	26.82	4.88	0.72	0.14	24.84	8.15	0.56	14.44
13	OSAH(atc)	6896	6897	682	7979	5.18	35.41	6.10	2.54	0.24	23.62	8.15	0.33	13.33
14	OSAH2(atc)	7239	7240	577	8427	5.57	36.04	6.19	2.29	0.33	24.20	8.15	0.34	13.85
15	OSAH+LC(atc)	7183	7184	682	8266	5.11	35.49	6.10	2.54	0.28	23.58	8.15	0.32	13.29
16	OSAH+TPC(atc)	4886	4887	675	5976	6.03	34.07	6.01	2.53	0.21	24.08	8.15	0.37	13.75
17	OSAH+TPC+LC(atc)	5466	5467	669	6570	5.61	34.34	5.95	2.49	0.29	22.50	8.15	0.35	12.31
18	OSAH+LC(16,1)	8619	8620	685	9699	4.95	35.72	6.07	2.51	0.35	22.17	8.15	0.31	12.01
19	OSAH+TPC(16,1)	8615	8616	683	9697	4.90	35.83	6.05	2.51	0.31	22.08	8.15	0.31	11.93
20	OSAH+TPC+LC(16,1)	8615	8616	683	9697	4.90	35.83	6.05	2.51	0.37	22.15	8.15	0.31	11.99
21	OSAH+PR(atc)	6896	6897	682	7979	5.18	35.41	6.10	2.54	0.42	23.63	8.15	0.33	13.34
22	OSAH+SC(atc)	7271	7272	688	8348	5.05	35.52	6.11	2.60	0.62	23.76	8.15	0.32	13.45
23	OSAH+GCM(atc)	7518	7519	737	8546	4.61	36.30	6.07	2.93	19.42	22.81	8.15	0.30	12.59
24	OSAH+GCM2(atc)	8425	8426	399	9792	5.88	36.65	6.34	2.11	22.15	23.39	8.15	0.35	13.12
25	OSAH+GCM3(atc)	6858	6859	441	8182	6.80	36.50	5.97	1.42	17.83	23.88	8.15	0.38	13.56
26	OSAH+PAR(atc)	6896	6897	682	7979	11.00	45.40	8.65	4.69	0.28	4.09	8.11	0.35	13.42
27	PARSAH+PAR(atc)	3919	3920	1425	4255	11.36	26.51	6.16	3.06	0.24	3.68	8.11	0.55	11.26
28	OSAH+PER(atc)	6896	6897	682	7979	5.05	47.88	9.35	4.75	0.28	6.73	8.18	0.42	6.78
29	PERSAH+PER(atc)	2011	2012	390	3329	64.50	17.98	3.48	0.50	9.42	15.54	8.18	0.94	26.36
30	SPHSAH+PER(atc)	3989	3990	951	4863	14.82	36.42	6.79	2.52	0.71	7.82	8.18	0.68	9.20
31	OSAH+SPH(atc)	6896	6897	682	7979	5.04	47.70	9.30	4.72	0.28	7.18	8.30	0.47	6.98
32	SPHSAH+SPH(atc)	3989	3990	951	4863	14.47	36.30	6.75	2.52	0.73	8.22	8.30	0.69	9.19
33	OSAH+TA$_{seq}$(16,2)	2164	2165	200	3729	10.39	60.08	4.91	0.75	0.12	30.54	8.15	0.38	19.62
34	OSAH+TA$^{A}_{rec}$(16,2)	2164	2165	200	3729	10.22	26.84	4.83	0.74	0.12	25.88	8.15	0.48	15.38
35	OSAH+TA$^{B}_{rec}$(16,2)	2164	2165	200	3729	10.22	26.84	4.83	0.74	0.11	23.57	8.15	0.56	13.28
36	OSAH+TA$_{SNL}$(16,2)	2164	2165	200	3729	10.39	17.56	4.91	0.75	0.13	24.46	8.15	0.53	14.09
37	OSAH+TA$_{NLT}$(16,2)	7855	2165	200	3729	10.39	16.99	4.91	0.75	0.16	24.87	8.15	0.51	14.46
38	OSAH+TA$_{seq}$(18,2)	2164	2165	200	3729	10.39	60.08	4.91	0.75	0.12	30.65	8.15	0.38	19.72
39	OSAH+TA$^{A}_{rec}$(18,2)	2164	2165	200	3729	10.22	26.84	4.83	0.74	0.11	25.86	8.15	0.48	15.36
40	OSAH+TA$^{B}_{rec}$(18,2)	2164	2165	200	3729	10.22	26.84	4.83	0.74	0.11	23.53	8.15	0.56	13.25
41	OSAH+TA$_{SNL}$(18,2)	2164	2165	200	3729	10.39	17.56	4.91	0.75	0.15	24.47	8.15	0.53	14.10
42	OSAH+TA$_{NLT}$(18,2)	7855	2165	200	3729	10.39	16.99	4.91	0.75	0.16	24.57	8.15	0.52	14.19
43	OSAH+TA$_{seq}$(atc)	6968	6969	685	8048	5.24	84.03	6.13	2.51	0.21	31.54	8.15	0.19	20.53
44	OSAH+TA$^{A}_{rec}$(atc)	6968	6969	685	8048	5.12	35.46	6.06	2.51	0.20	25.35	8.15	0.26	14.90
45	OSAH+TA$^{B}_{rec}$(atc)	6968	6969	685	8048	5.12	35.46	6.06	2.51	0.20	22.17	8.15	0.32	12.01
46	OSAH+TA$_{SNL}$(atc)	6968	6969	685	8048	5.24	22.71	6.13	2.51	0.31	21.99	8.15	0.32	11.85
47	OSAH+TA$_{NLT}$(atc)	29056	6969	685	8048	5.24	21.25	6.13	2.51	0.32	22.01	8.15	0.32	11.86
48	BVH	102	759	0	1225	45.03	43.87	34.39	0.00	0.13	471.90	–	–	–
49	O84	4177	29240	0	45296	15.96	41.30	7.67	0.00	0.16	53.84	–	–	–
50	O89	4177	29240	0	45296	15.84	30.17	7.64	0.00	0.14	46.73	–	–	–
51	BSP	9791	9792	0	22056	16.02	35.05	6.21	0.00	1.42	83.74	–	–	–
52	O93	4177	29240	0	45296	15.30	31.02	13.98	6.62	0.16	56.70	–	–	–
53	UG	0	6859	0	9799	14.19	7.82	7.82	0.00	0.08	33.24	–	–	–
54	AG	514	1279	0	3366	26.92	26.75	21.82	0.00	0.16	270.32	–	–	–
55	HUG	1	2197	0	4843	19.07	6.93	5.93	0.00	0.05	48.26	–	–	–
56	RG	9	1298	0	7064	35.28	5.54	4.50	0.00	0.02	62.30	–	–	–
57	O84A	1769	12384	584	17176	10.46	35.86	6.93	1.60	0.25	45.47	–	–	–
58	KD	2171	2172	200	3736	10.37	26.82	4.88	0.72	1.14	42.97	–	–	–

Table 4: Experimental results for scene “*lattice6*”.

$$N = 1225,$$
$$TP_D:\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 263169,\quad N^{hit}_{prim} = 250614,\quad N_{sec} = 214756,\quad N^{hit}_{sec} = 134665,$$
$$N_{shad} = 1124636,\quad N^{hit}_{shad} = 788080,\quad T^{MIN}_{R}[s] = 10.06,\quad T_{app}[s] = 8.96,\quad T^{MIN}_{RSA}[s] = 1.10.$$

Scene = “*mount4*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	516	–	0	0	0	0.00	253.97	15.85	1.0	746.97
1	spatmed-xyz(16,2)	8348	8349	922	22667	15.60	27.03	5.13	1.01	0.10	16.39	15.85	0.38	32.35
2	objmed-xyz(16,2)	797	798	117	1317	16.24	27.81	4.88	0.50	0.03	15.80	15.85	0.38	30.62
3	objmed(16,2)	318	319	8	609	19.16	31.24	5.86	0.18	0.03	17.58	15.85	0.39	35.85
4	OSAH(16,2)	424	425	88	633	6.89	17.25	3.17	0.81	0.04	12.64	15.85	0.29	21.32
5	OSAH-RMI(16,2)	424	425	88	633	6.89	17.25	3.17	0.81	0.03	12.14	15.85	0.29	19.85
6	OSAH-xyz(16,2)	482	483	95	726	7.27	17.47	3.27	0.73	0.02	12.04	15.85	0.30	19.56
7	OSAH(8,1)	146	147	37	606	12.31	12.39	2.45	0.60	0.03	12.69	15.85	0.51	21.47
8	OSAH(8,2)	143	144	34	606	12.31	12.39	2.45	0.60	0.02	12.66	15.85	0.51	21.38
9	OSAH(16,1)	1068	1069	478	887	5.80	18.95	3.39	1.14	0.05	12.42	15.85	0.24	20.68
10	OSAH(16,2)	424	425	88	633	6.89	17.25	3.17	0.81	0.04	12.64	15.85	0.29	21.32
11	OSAH(24,1)	1082	1083	478	912	5.37	19.60	3.46	1.14	0.04	12.44	15.85	0.22	20.74
12	OSAH(24,2)	433	434	88	653	6.66	17.58	3.21	0.81	0.03	12.36	15.85	0.28	20.50
13	OSAH(atc)	842	843	422	707	6.65	14.97	2.88	1.10	0.04	11.53	15.85	0.32	18.06
14	OSAH2(atc)	1019	1020	404	958	6.87	17.21	3.24	0.87	0.06	12.21	15.85	0.29	20.06
15	OSAH+LC(atc)	848	849	422	722	6.31	16.72	3.12	1.10	0.04	11.99	15.85	0.28	19.41
16	OSAH+TPC(atc)	764	765	373	678	8.21	13.03	2.53	1.06	0.04	11.26	15.85	0.40	17.26
17	OSAH+TPC+LC(atc)	782	783	374	704	9.13	16.70	3.28	1.06	0.05	11.70	15.85	0.36	18.56
18	OSAH+LC(16,1)	1068	1069	478	887	5.80	18.94	3.39	1.14	0.05	11.33	15.85	0.24	17.47
19	OSAH+TPC(16,1)	1067	1068	477	887	5.79	18.91	3.38	1.13	0.05	11.24	15.85	0.24	17.21
20	OSAH+TPC+LC(16,1)	1067	1068	477	887	5.79	18.91	3.38	1.13	0.05	11.24	15.85	0.24	17.21
21	OSAH+PR(atc)	842	843	422	706	6.65	14.97	2.88	1.10	0.05	11.51	15.85	0.32	18.00
22	OSAH+SC(atc)	880	881	438	730	6.48	14.98	2.90	1.11	0.05	11.66	15.85	0.31	18.44
23	OSAH+GCM(atc)	878	879	425	731	6.53	15.46	2.97	1.25	2.37	11.40	15.85	0.31	17.68
24	OSAH+GCM2(atc)	1325	1326	448	1395	6.39	20.64	3.81	0.97	3.81	12.40	15.85	0.24	20.62
25	OSAH+GCM3(atc)	991	992	371	994	7.37	18.87	3.46	0.97	2.86	12.09	15.85	0.29	19.71
26	OSAH+PAR(atc)	842	843	422	707	5.07	20.67	4.31	2.91	0.05	1.97	13.50	0.21	19.33
27	PARSAH+PAR(atc)	824	825	416	688	5.05	20.87	4.35	2.96	0.05	1.98	13.50	0.21	19.50
28	OSAH+PER(atc)	842	843	422	707	5.21	22.38	4.77	2.67	0.05	2.68	14.25	0.35	19.25
29	PERSAH+PER(atc)	819	820	381	714	6.41	19.74	4.48	2.49	3.22	2.60	14.25	0.40	18.25
30	SPHSAH+PER(atc)	842	843	422	707	5.21	22.38	4.77	2.67	0.06	2.51	14.25	0.27	17.12
31	OSAH+SPH(atc)	842	843	422	707	5.20	22.02	4.69	2.64	0.04	2.81	14.50	0.28	13.60
32	SPHSAH+SPH(atc)	842	843	422	707	5.20	22.02	4.69	2.64	0.09	2.95	14.50	0.35	15.00
33	OSAH+TA$_{seq}$(16,2)	424	425	88	633	7.37	29.98	3.17	0.81	0.02	13.79	15.85	0.20	24.71
34	OSAH+TA$^{A}_{rec}$(16,2)	424	425	88	633	6.89	17.25	3.17	0.82	0.02	12.10	15.85	0.25	19.74
35	OSAH+TA$^{B}_{rec}$(16,2)	424	425	88	633	6.89	17.25	3.17	0.81	0.02	11.25	15.85	0.29	17.24
36	OSAH+TA$_{SNL}$(16,2)	424	425	88	633	7.38	11.88	3.17	0.81	0.03	10.26	15.85	0.35	14.32
37	OSAH+TA$_{NLT}$(16,2)	1829	425	88	633	7.38	10.89	3.17	0.81	0.03	10.54	15.85	0.33	15.15
38	OSAH+TA$_{seq}$(18,2)	431	432	88	648	7.16	30.86	3.21	0.81	0.03	13.95	15.85	0.19	25.18
39	OSAH+TA$^{A}_{rec}$(18,2)	431	432	88	648	6.69	17.56	3.21	0.82	0.03	12.15	15.85	0.24	19.88
40	OSAH+TA$^{B}_{rec}$(18,2)	431	432	88	648	6.68	17.56	3.21	0.81	0.03	11.28	15.85	0.28	17.32
41	OSAH+TA$_{SNL}$(18,2)	431	432	88	648	7.17	12.06	3.21	0.81	0.03	10.27	15.85	0.34	14.35
42	OSAH+TA$_{NLT}$(18,2)	1856	432	88	648	7.17	11.04	3.21	0.81	0.04	10.32	15.85	0.33	14.50
43	OSAH+TA$_{seq}$(atc)	842	843	422	707	7.12	25.26	2.88	1.10	0.03	12.77	15.85	0.23	21.71
44	OSAH+TA$^{A}_{rec}$(atc)	842	843	422	707	6.65	14.97	2.88	1.10	0.03	11.20	15.85	0.29	17.09
45	OSAH+TA$^{B}_{rec}$(atc)	842	843	422	707	6.65	14.97	2.88	1.10	0.03	10.61	15.85	0.32	15.35
46	OSAH+TA$_{SNL}$(atc)	842	843	422	707	7.12	10.42	2.88	1.10	0.04	9.75	15.85	0.38	12.82
47	OSAH+TA$_{NLT}$(atc)	3473	843	422	707	7.12	9.66	2.88	1.10	0.05	9.80	15.85	0.38	12.97
48	BVH	42	304	0	516	32.28	21.59	16.22	0.00	0.02	185.02	–	–	–
49	O84	4085	28596	5069	61738	15.84	33.55	6.28	1.62	0.18	31.01	–	–	–
50	O89	4085	28596	5069	61738	15.87	24.05	6.27	1.62	0.15	27.19	–	–	–
51	BSP	8348	8349	922	22667	15.60	27.03	5.13	1.01	0.21	36.70	–	–	–
52	O93	4085	28596	5069	61738	15.34	23.73	10.13	5.63	0.15	32.65	–	–	–
53	UG	0	2548	1839	3629	16.61	6.27	6.27	3.09	0.02	20.28	–	–	–
54	AG	160	4002	1292	9819	22.01	9.71	8.06	3.46	0.14	36.49	–	–	–
55	HUG	1	343	172	1853	28.48	4.43	3.43	0.88	0.01	27.35	–	–	–
56	RG	137	2443	564	9200	18.80	5.83	4.37	1.22	0.03	23.44	–	–	–
57	O84A	5793	40552	3378	98764	11.94	30.41	5.81	1.94	0.64	27.98	–	–	–
58	KD	424	425	88	633	6.89	17.25	3.17	0.81	0.09	18.88	–	–	–

Table 5: Experimental results for scene “*mount4*”.

$$N = 516,$$
$$TP_D\colon\ N_{prim} = 263169,\ N^{hit}_{\mathcal{AB}} = 257871,\ N^{hit}_{prim} = 173250,\quad N_{sec} = 707764,\ N^{hit}_{sec} = 472407,$$
$$N_{shad} = 358042,\ N^{hit}_{shad} = 24257,\quad T^{MIN}_{R}[s] = 5.73,\ T_{app}[s] = 5.39,\ T^{MIN}_{RSA}[s] = 0.34.$$

Scene = “*rings3*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	841	–	0	0	0	0.00	1014.57	4.63	1.0	798.87
1	spatmed-xyz(16,2)	5839	5840	627	18169	29.09	44.65	8.86	3.58	0.09	40.62	4.63	0.62	27.35
2	objmed-xyz(16,2)	11578	11579	1175	25092	64.63	92.97	18.86	5.30	0.29	78.94	4.63	0.64	57.53
3	objmed(16,2)	14628	14629	1873	28296	78.89	109.04	24.31	6.78	0.47	94.97	4.63	0.64	70.15
4	OSAH(16,2)	2974	2975	359	6519	14.00	26.98	5.12	1.60	0.17	25.30	4.63	0.57	15.29
5	OSAH-RMI(16,2)	2974	2975	359	6519	14.00	26.98	5.12	1.60	0.18	24.70	4.63	0.57	14.82
6	OSAH-xyz(16,2)	3257	3258	335	7287	15.41	29.75	5.95	2.04	0.12	26.35	4.63	0.56	16.12
7	OSAH(8,1)	85	86	23	1118	61.97	14.47	3.23	1.25	0.05	47.78	4.63	0.91	32.99
8	OSAH(8,2)	85	86	23	1118	61.97	14.47	3.23	1.25	0.05	47.73	4.63	0.91	32.95
9	OSAH(16,1)	4250	4251	829	7301	13.16	30.33	5.84	2.10	0.19	24.89	4.63	0.52	14.97
10	OSAH(16,2)	2974	2975	359	6519	14.00	26.98	5.12	1.60	0.17	25.30	4.63	0.57	15.29
11	OSAH(24,1)	26501	26502	1794	44050	15.66	38.95	7.38	2.21	0.81	29.95	4.63	0.50	18.95
12	OSAH(24,2)	15943	15944	498	34813	15.82	31.98	5.99	1.63	0.55	28.21	4.63	0.55	17.58
13	OSAH(atc)	629	630	98	2322	21.08	21.31	4.26	1.53	0.10	27.46	4.63	0.71	16.99
14	OSAH2(atc)	1894	1895	212	4295	16.30	30.66	6.13	2.41	0.17	26.95	4.63	0.57	16.59
15	OSAH+LC(atc)	1956	1957	98	7058	22.36	25.99	5.05	1.53	0.16	30.35	4.63	0.68	19.27
16	OSAH+TPC(atc)	380	381	55	1892	25.39	19.64	3.97	1.45	0.10	29.55	4.63	0.76	18.64
17	OSAH+TPC+LC(atc)	1802	1803	55	7244	27.73	25.43	5.00	1.45	0.18	32.76	4.63	0.73	21.17
18	OSAH+LC(16,1)	4270	4271	835	7323	13.16	30.34	5.84	2.11	0.24	23.89	4.63	0.52	14.18
19	OSAH+TPC(16,1)	4269	4270	829	7330	13.22	30.32	5.84	2.09	0.25	23.92	4.63	0.52	14.20
20	OSAH+TPC+LC(16,1)	4269	4270	829	7330	13.22	30.32	5.84	2.09	0.25	24.01	4.63	0.52	14.28
21	OSAH+PR(atc)	629	630	98	1993	19.30	21.37	4.28	1.53	0.17	25.83	4.63	0.69	15.71
22	OSAH+SC(atc)	901	902	316	1961	13.84	23.30	4.66	2.29	0.18	22.27	4.63	0.60	12.91
23	OSAH+GCM(atc)	1742	1743	341	3996	15.16	25.61	5.19	1.90	5.75	24.91	4.63	0.60	14.98
24	OSAH+GCM2(atc)	3983	3984	134	11430	20.14	33.45	6.45	1.82	13.60	31.07	4.63	0.60	19.83
25	OSAH+GCM3(atc)	2016	2017	268	5218	22.30	41.50	7.20	3.28	7.09	33.00	4.63	0.57	21.35
26	OSAH+PAR(atc)	629	630	98	2322	4.37	5.55	1.12	0.49	0.12	2.06	3.52	0.73	2.73
27	PARSAH+PAR(atc)	1874	1875	91	11039	2.63	3.14	0.44	0.01	0.28	1.86	3.52	0.80	2.12
28	OSAH+PER(atc)	629	630	98	2322	8.52	20.01	4.18	1.72	0.11	6.28	3.58	0.77	5.52
29	PERSAH+PER(atc)	653	654	116	2412	14.91	15.97	3.77	1.56	20.88	7.38	3.58	0.86	7.12
30	SPHSAH+PER(atc)	1557	1558	300	6282	17.01	26.16	5.74	2.57	1.11	9.06	3.58	0.82	9.55
31	OSAH+SPH(atc)	629	630	98	2322	8.56	19.79	4.13	1.70	0.10	6.40	3.85	0.77	5.56
32	SPHSAH+SPH(atc)	1557	1558	300	6282	17.18	26.11	5.75	2.59	1.11	9.20	3.85	0.82	9.68
33	OSAH+TA$_{seq}$(16,2)	2974	2975	358	6527	14.95	65.01	5.29	1.60	0.15	31.35	4.63	0.40	20.06
34	OSAH+TA$^A_{rec}$(16,2)	2974	2975	358	6527	14.00	26.95	5.11	1.60	0.14	25.72	4.63	0.51	15.62
35	OSAH+TA$^B_{rec}$(16,2)	2974	2975	358	6527	14.00	26.95	5.11	1.60	0.14	23.73	4.63	0.57	14.06
36	OSAH+TA$_{SNL}$(16,2)	2974	2975	358	6527	14.95	23.89	5.29	1.60	0.20	26.70	4.63	0.49	16.39
37	OSAH+TA$_{NLT}$(16,2)	18194	2975	358	6527	14.95	21.28	5.29	1.60	0.22	26.93	4.63	0.48	16.57
38	OSAH+TA$_{seq}$(18,2)	5344	5345	465	11012	15.12	72.82	5.61	1.63	0.22	33.37	4.63	0.37	21.65
39	OSAH+TA$^A_{rec}$(18,2)	5344	5345	465	11012	14.04	28.69	5.40	1.63	0.20	26.68	4.63	0.49	16.38
40	OSAH+TA$^B_{rec}$(18,2)	5344	5345	465	11012	14.04	28.69	5.40	1.63	0.20	24.39	4.63	0.55	14.57
41	OSAH+TA$_{SNL}$(18,2)	5344	5345	465	11012	15.12	25.49	5.61	1.63	0.30	27.61	4.63	0.47	17.11
42	OSAH+TA$_{NLT}$(18,2)	31614	5345	465	11012	15.12	22.56	5.61	1.63	0.29	27.70	4.63	0.47	17.18
43	OSAH+TA$_{seq}$(atc)	630	631	99	2327	22.10	45.68	4.38	1.53	0.08	32.02	4.63	0.55	20.58
44	OSAH+TA$^A_{rec}$(atc)	630	631	99	2327	20.99	21.35	4.27	1.53	0.09	27.66	4.63	0.66	17.15
45	OSAH+TA$^B_{rec}$(atc)	630	631	99	2327	20.99	21.35	4.27	1.53	0.09	26.20	4.63	0.71	16.00
46	OSAH+TA$_{SNL}$(atc)	630	631	99	2327	22.10	18.91	4.38	1.53	0.07	29.77	4.63	0.60	18.81
47	OSAH+TA$_{NLT}$(atc)	3574	631	99	2327	22.10	17.21	4.38	1.53	0.10	29.69	4.63	0.61	18.75
48	BVH	67	506	0	841	52.44	30.88	22.98	0.00	0.06	282.66	–	–	–
49	O84	2989	20924	2962	47428	29.44	67.92	11.97	5.43	0.14	76.77	–	–	–
50	O89	2989	20924	2962	47428	29.46	45.57	11.97	5.43	0.13	65.08	–	–	–
51	BSP	5839	5840	627	18169	29.09	44.65	8.86	3.58	0.93	119.09	–	–	–
52	O93	2989	20924	2962	47428	29.20	42.62	17.95	11.44	0.12	75.17	–	–	–
53	UG	0	4046	3140	4414	42.37	12.63	12.63	8.71	0.04	59.47	–	–	–
54	AG	11	2523	112	9162	27.88	7.00	5.21	0.12	0.13	61.55	–	–	–
55	HUG	23	1280	0	6406	353.31	6.02	2.73	0.00	0.05	209.44	–	–	–
56	RG	222	3911	1018	11803	36.69	12.81	9.87	5.25	0.04	64.49	–	–	–
57	O84A	3947	27630	1770	73146	24.33	49.30	9.33	3.23	0.45	63.52	–	–	–
58	KD	2974	2975	359	6519	14.00	26.98	5.12	1.60	0.45	40.39	–	–	–

Table 6: Experimental results for scene “*rings3*”.

$$N = 841,$$
$$TP_D:\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 263169,\quad N^{hit}_{prim} = 263168,\quad N_{sec} = 177299,\quad N^{hit}_{sec} = 90933,$$
$$N_{shad} = 937972,\quad N^{hit}_{shad} = 319722,\quad T^{MIN}_R[s] = 7.15,\quad T_{app}[s] = 5.88,\quad T^{MIN}_{RSA}[s] = 1.27.$$

Scene = “*sombrero1*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	1922	–	0	0	0	0.00	415.55	12.33	1.0	4617.22
1	spatmed-xyz(16,2)	9763	9764	1314	35107	35.23	27.40	5.06	2.28	0.18	4.16	12.33	0.46	33.89
2	objmed-xyz(16,2)	3948	3949	359	10348	75.13	31.89	5.78	0.48	0.21	5.72	12.33	0.61	51.22
3	objmed(16,2)	977	978	0	1953	26.80	20.35	4.11	0.00	0.14	3.61	12.33	0.47	27.78
4	OSAH(16,2)	1099	1100	136	1926	6.66	12.76	2.32	1.28	0.16	2.49	12.33	0.26	15.33
5	OSAH-RMI(16,2)	1099	1100	136	1926	6.66	12.76	2.32	1.28	0.16	2.37	12.33	0.26	14.00
6	OSAH-xyz(16,2)	1843	1844	564	2552	7.96	16.36	3.18	1.97	0.13	2.67	12.33	0.25	17.33
7	OSAH(8,1)	159	160	64	1926	58.57	9.17	1.99	1.00	0.12	4.28	12.33	0.81	35.22
8	OSAH(8,2)	158	159	63	1926	58.69	9.14	1.98	0.98	0.12	4.15	12.33	0.81	33.78
9	OSAH(16,1)	3702	3703	1827	2820	6.04	15.64	3.05	1.90	0.21	2.55	12.33	0.21	16.00
10	OSAH(16,2)	1099	1100	136	1926	6.66	12.76	2.32	1.28	0.16	2.49	12.33	0.26	15.33
11	OSAH(24,1)	3722	3723	1827	2840	6.05	15.65	3.05	1.90	0.20	2.59	12.33	0.21	16.44
12	OSAH(24,2)	1099	1100	136	1926	6.66	12.76	2.32	1.28	0.16	2.49	12.33	0.26	15.33
13	OSAH(atc)	3509	3510	1826	2628	5.89	15.48	2.99	1.90	0.20	2.57	12.33	0.20	16.22
14	OSAH2(atc)	3117	3118	1571	2527	6.73	16.89	3.01	1.69	0.26	2.55	12.33	0.21	16.00
15	OSAH+LC(atc)	3509	3510	1826	2628	5.89	15.48	2.99	1.90	0.20	2.57	12.33	0.20	16.22
16	OSAH+TPC(atc)	3510	3511	1831	2630	5.98	15.88	3.12	2.00	0.21	2.61	12.33	0.20	16.67
17	OSAH+TPC+LC(atc)	3510	3511	1831	2630	5.98	15.88	3.12	2.00	0.23	2.31	12.33	0.20	13.33
18	OSAH+LC(16,1)	3702	3703	1827	2820	6.04	15.64	3.05	1.90	0.24	2.41	12.33	0.21	14.44
19	OSAH+TPC(16,1)	3717	3718	1832	2835	6.14	16.07	3.18	2.00	0.24	2.32	12.33	0.20	13.44
20	OSAH+TPC+LC(16,1)	3717	3718	1832	2835	6.14	16.07	3.18	2.00	0.25	2.44	12.33	0.20	14.78
21	OSAH+PR(atc)	3509	3510	1826	2617	5.87	15.48	2.99	1.90	0.22	2.58	12.33	0.20	16.33
22	OSAH+SC(atc)	3515	3516	1828	2632	5.88	15.47	2.99	1.89	0.24	2.33	12.33	0.20	13.56
23	OSAH+GCM(atc)	3544	3545	1757	2734	6.10	18.75	3.71	2.54	10.87	2.46	12.33	0.18	15.00
24	OSAH+GCM2(atc)	4110	4111	1406	4248	7.89	18.64	3.44	1.91	14.86	2.50	12.33	0.22	15.44
25	OSAH+GCM3(atc)	3165	3166	1372	2977	7.55	19.32	3.82	2.37	10.36	2.55	12.33	0.21	16.00
26	OSAH+PAR(atc)	3509	3510	1826	2628	2.89	11.70	2.28	1.50	0.23	1.58	9.50	0.40	10.25
27	PARSAH+PAR(atc)	3407	3408	1724	2630	2.89	10.29	2.05	1.29	0.25	1.59	9.50	0.48	10.38
28	OSAH+PER(atc)	3509	3510	1826	2628	2.95	11.10	2.15	1.38	0.24	1.58	11.38	0.30	8.38
29	PERSAH+PER(atc)	2536	2537	1275	2446	2.91	10.53	2.12	1.37	8.93	1.60	11.38	0.35	8.62
30	SPHSAH+PER(atc)	1930	1931	990	2241	52.20	10.66	1.85	0.92	0.45	3.35	11.38	0.82	30.50
31	OSAH+SPH(atc)	3509	3510	1826	2628	2.95	11.12	2.16	1.39	0.23	2.05	11.30	0.49	9.20
32	SPHSAH+SPH(atc)	1930	1931	990	2241	51.59	10.70	1.86	0.93	0.57	3.72	11.30	0.83	25.90
33	OSAH+TA$_{seq}$(16,2)	1099	1100	136	1926	6.66	25.54	2.32	1.28	0.13	2.82	12.33	0.16	19.00
34	OSAH+TA$^{A}_{rec}$(16,2)	1099	1100	136	1926	6.66	12.76	2.32	1.28	0.11	2.33	12.33	0.22	13.56
35	OSAH+TA$^{B}_{rec}$(16,2)	1099	1100	136	1926	6.66	12.76	2.32	1.28	0.13	2.16	12.33	0.26	11.67
36	OSAH+TA$_{SNL}$(16,2)	1099	1100	136	1926	6.66	12.03	2.32	1.28	0.14	2.17	12.33	0.26	11.78
37	OSAH+TA$_{NLT}$(16,2)	4253	1100	136	1926	6.66	11.56	2.32	1.28	0.14	2.17	12.33	0.26	11.78
38	OSAH+TA$_{seq}$(18,2)	1099	1100	136	1926	6.66	25.54	2.32	1.28	0.12	2.76	12.33	0.17	18.33
39	OSAH+TA$^{A}_{rec}$(18,2)	1099	1100	136	1926	6.66	12.76	2.32	1.28	0.13	2.34	12.33	0.23	13.67
40	OSAH+TA$^{B}_{rec}$(18,2)	1099	1100	136	1926	6.66	12.76	2.32	1.28	0.13	2.21	12.33	0.26	12.22
41	OSAH+TA$_{SNL}$(18,2)	1099	1100	136	1926	6.66	12.03	2.32	1.28	0.13	2.30	12.33	0.24	13.22
42	OSAH+TA$_{NLT}$(18,2)	4253	1100	136	1926	6.66	11.56	2.32	1.28	0.15	2.17	12.33	0.27	11.78
43	OSAH+TA$_{seq}$(atc)	3509	3510	1826	2628	5.89	38.10	2.99	1.90	0.26	3.18	12.33	0.11	23.00
44	OSAH+TA$^{A}_{rec}$(atc)	3509	3510	1826	2628	5.89	15.48	2.99	1.90	0.16	2.48	12.33	0.17	15.22
45	OSAH+TA$^{B}_{rec}$(atc)	3509	3510	1826	2628	5.89	15.48	2.99	1.90	0.16	2.26	12.33	0.20	12.78
46	OSAH+TA$_{SNL}$(atc)	3509	3510	1826	2628	5.89	14.25	2.99	1.90	0.22	2.36	12.33	0.18	13.89
47	OSAH+TA$_{NLT}$(atc)	13132	3510	1826	2628	5.89	13.33	2.99	1.90	0.22	2.24	12.33	0.20	12.56
48	BVH	205	1223	0	1922	60.37	33.45	25.24	0.00	0.18	82.50	–	–	–
49	O84	5027	35190	8886	85306	37.29	43.37	7.49	3.80	0.26	9.07	–	–	–
50	O89	5027	35190	8886	85306	37.18	28.91	7.48	3.80	0.25	7.23	–	–	–
51	BSP	9763	9764	1314	35107	35.23	27.40	5.06	2.28	0.29	6.70	–	–	–
52	O93	5027	35190	8886	85306	36.04	26.35	10.68	7.09	0.24	8.69	–	–	–
53	UG	0	9583	7356	12374	29.38	7.18	7.18	5.47	0.09	4.48	–	–	–
54	AG	360	11472	3905	30027	35.07	6.78	5.46	2.81	0.43	7.49	–	–	–
55	HUG	1	432	205	4448	99.32	3.31	2.65	1.32	0.04	9.28	–	–	–
56	RG	319	5788	2984	15420	30.02	6.37	5.02	3.47	0.05	5.39	–	–	–
57	O84A	6353	44472	5260	123036	34.76	39.57	6.83	2.82	0.98	8.51	–	–	–
58	KD	1099	1100	136	1926	6.66	12.76	2.32	1.28	0.33	3.82	–	–	–

Table 7: Experimental results for scene “*sombrero1*”.

$$N = 1922,$$
$$TP_D\text{:}\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 136638,\quad N^{hit}_{prim} = 112209,\quad N_{sec} = 0,\quad N^{hit}_{sec} = 0,$$
$$N_{shad} = 110241,\quad N^{hit}_{shad} = 2247,\quad T^{MIN}_{R}[s] = 1.20,\quad T_{app}[s] = 1.11,\quad T^{MIN}_{RSA}[s] = 0.09.$$

Scene = “*teapot4*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	1008	–	0	0	0	0.00	414.85	20.11	1.0	2183.42
1	spatmed-xyz(16,2)	6074	6075	959	21619	37.24	30.46	6.38	2.79	0.11	12.31	20.11	0.47	44.68
2	objmed-xyz(16,2)	3130	3131	397	7407	82.77	47.73	9.75	1.40	0.11	18.26	20.11	0.55	76.00
3	objmed(16,2)	3917	3918	482	8654	76.44	71.14	13.69	1.04	0.18	20.78	20.11	0.43	89.26
4	OSAH(16,2)	2635	2636	375	5373	19.56	19.12	4.07	0.95	0.14	9.54	20.11	0.42	30.11
5	OSAH-RMI(16,2)	2635	2636	375	5373	19.56	19.12	4.07	0.95	0.15	9.22	20.11	0.42	28.42
6	OSAH-xyz(16,2)	2094	2095	339	4566	18.82	19.61	4.14	1.05	0.09	9.11	20.11	0.41	27.84
7	OSAH(8,1)	172	173	34	1461	48.61	14.93	3.47	1.58	0.06	11.58	20.11	0.70	40.84
8	OSAH(8,2)	154	155	24	1461	55.06	13.06	3.12	0.43	0.05	12.01	20.11	0.75	43.11
9	OSAH(16,1)	3743	3744	881	5933	11.60	22.77	4.76	2.46	0.17	9.09	20.11	0.27	27.74
10	OSAH(16,2)	2635	2636	375	5373	19.56	19.12	4.07	0.95	0.14	9.54	20.11	0.42	30.11
11	OSAH(24,1)	5591	5592	987	10207	11.55	23.20	4.83	2.48	0.25	9.16	20.11	0.26	28.11
12	OSAH(24,2)	4078	4079	402	9322	19.54	19.42	4.12	0.96	0.21	9.45	20.11	0.42	29.63
13	OSAH(atc)	1534	1535	436	3069	14.54	20.74	4.41	2.34	0.11	8.88	20.11	0.33	26.63
14	OSAH2(atc)	2090	2091	459	4082	14.18	22.91	4.67	2.28	0.17	9.42	20.11	0.31	29.47
15	OSAH+LC(atc)	1787	1788	446	3955	13.83	21.24	4.48	2.38	0.13	8.86	20.11	0.32	26.53
16	OSAH+TPC(atc)	1222	1223	370	2767	16.31	19.93	4.27	2.25	0.11	8.92	20.11	0.37	26.84
17	OSAH+TPC+LC(atc)	1681	1682	397	4169	14.84	20.84	4.41	2.29	0.15	8.46	20.11	0.34	24.42
18	OSAH+LC(16,1)	3748	3749	886	5933	11.51	22.80	4.76	2.50	0.21	8.37	20.11	0.26	23.95
19	OSAH+TPC(16,1)	3755	3756	885	5952	11.59	22.78	4.76	2.47	0.21	8.33	20.11	0.27	23.74
20	OSAH+TPC+LC(16,1)	3760	3761	890	5952	11.50	22.82	4.76	2.50	0.20	8.36	20.11	0.26	23.89
21	OSAH+PR(atc)	1534	1535	436	2863	14.27	20.51	4.35	2.37	0.14	8.14	20.11	0.32	22.74
22	OSAH+SC(atc)	1579	1580	584	2741	11.52	20.30	4.30	2.43	0.20	8.81	20.11	0.29	26.26
23	OSAH+GCM(atc)	2006	2007	479	3747	12.71	21.45	4.47	2.21	5.94	8.82	20.11	0.30	26.32
24	OSAH+GCM2(atc)	3728	3729	521	9721	16.33	25.60	4.94	2.18	11.77	9.53	20.11	0.31	30.05
25	OSAH+GCM3(atc)	1777	1778	332	4263	18.87	25.22	4.75	1.85	5.51	9.85	20.11	0.35	31.74
26	OSAH+PAR(atc)	1534	1535	436	3069	5.24	15.23	3.30	2.05	0.14	2.59	14.64	0.41	8.91
27	PARSAH+PAR(atc)	1452	1453	392	2971	5.38	14.75	3.17	1.78	0.15	2.53	14.64	0.39	8.36
28	OSAH+PER(atc)	1534	1535	436	3069	4.61	18.59	4.12	2.66	0.13	3.15	17.08	0.24	7.15
29	PERSAH+PER(atc)	1504	1505	399	3002	4.85	16.40	3.69	2.05	10.15	3.15	17.08	0.33	7.15
30	SPHSAH+PER(atc)	1111	1112	291	2782	6.37	17.33	3.84	2.34	0.39	3.23	17.08	0.35	7.77
31	OSAH+SPH(atc)	1534	1535	436	3069	4.61	18.19	4.03	2.60	0.19	3.46	17.36	0.33	7.36
32	SPHSAH+SPH(atc)	1111	1112	291	2782	6.37	16.95	3.76	2.29	0.40	3.52	17.36	0.41	7.79
33	OSAH+TA$_{seq}$(16,2)	2635	2636	375	5373	19.69	42.00	4.09	0.95	0.12	11.69	20.11	0.26	41.42
34	OSAH+TA$^{A}_{rec}$(16,2)	2635	2636	375	5373	19.56	19.12	4.07	0.95	0.12	9.43	20.11	0.36	29.53
35	OSAH+TA$^{B}_{rec}$(16,2)	2635	2636	375	5373	19.56	19.12	4.07	0.95	0.13	8.68	20.11	0.42	25.58
36	OSAH+TA$_{SNL}$(16,2)	2635	2636	375	5373	19.70	15.62	4.09	0.95	0.16	8.78	20.11	0.41	26.11
37	OSAH+TA$_{NLT}$(16,2)	12142	2636	375	5373	19.70	14.26	4.09	0.95	0.16	8.91	20.11	0.40	26.79
38	OSAH+TA$_{seq}$(18,2)	3418	3419	391	7375	19.65	42.93	4.13	0.96	0.15	11.75	20.11	0.26	41.74
39	OSAH+TA$^{A}_{rec}$(18,2)	3418	3419	391	7375	19.52	19.34	4.11	0.96	0.14	9.47	20.11	0.36	29.74
40	OSAH+TA$^{B}_{rec}$(18,2)	3418	3419	391	7375	19.52	19.34	4.11	0.96	0.15	8.73	20.11	0.42	25.84
41	OSAH+TA$_{SNL}$(18,2)	3418	3419	391	7375	19.66	15.75	4.13	0.96	0.20	8.83	20.11	0.41	26.37
42	OSAH+TA$_{NLT}$(18,2)	14817	3419	391	7375	19.66	14.37	4.13	0.96	0.19	8.80	20.11	0.41	26.21
43	OSAH+TA$_{seq}$(atc)	1534	1535	436	3069	14.70	44.85	4.42	2.34	0.09	11.60	20.11	0.19	40.95
44	OSAH+TA$^{A}_{rec}$(atc)	1534	1535	436	3069	14.54	20.74	4.41	2.34	0.09	9.17	20.11	0.28	28.16
45	OSAH+TA$^{B}_{rec}$(atc)	1534	1535	436	3069	14.54	20.74	4.41	2.34	0.08	8.35	20.11	0.33	23.84
46	OSAH+TA$_{SNL}$(atc)	1534	1535	436	3069	14.70	16.74	4.42	2.34	0.12	8.47	20.11	0.32	24.47
47	OSAH+TA$_{NLT}$(atc)	7550	1535	436	3069	14.70	15.28	4.42	2.34	0.13	8.44	20.11	0.32	24.32
48	BVH	113	623	0	1008	78.49	36.66	25.50	0.00	0.07	195.22	–	–	–
49	O84	2964	20749	4990	50544	34.31	46.95	8.99	5.03	0.17	25.83	–	–	–
50	O89	2964	20749	4990	50544	34.28	33.44	8.98	5.03	0.13	20.48	–	–	–
51	BSP	6074	6075	959	21619	37.24	30.46	6.38	2.79	0.83	53.66	–	–	–
52	O93	2964	20749	4990	50544	33.67	30.48	12.54	8.63	0.14	24.51	–	–	–
53	UG	0	4968	3717	6466	48.30	12.25	12.25	9.23	0.05	17.37	–	–	–
54	AG	472	12374	3583	30866	46.52	11.93	10.10	4.72	0.40	29.66	–	–	–
55	HUG	303	2906	1566	7550	45.89	10.45	8.62	6.35	0.07	21.57	–	–	–
56	RG	626	9861	2399	38017	44.85	11.11	8.85	5.03	0.11	20.39	–	–	–
57	O84A	5732	40125	4606	109596	28.64	45.35	8.94	4.94	0.81	25.02	–	–	–
58	KD	2635	2636	375	5373	19.56	19.12	4.07	0.95	0.37	13.94	–	–	–

Table 8: Experimental results for scene “*teapot4*”.

$$N = 1008,$$
$$TP_D\colon\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 226198,\quad N^{hit}_{prim} = 161203,\qquad N_{sec} = 228252,\quad N^{hit}_{sec} = 71224,$$
$$N_{shad} = 408292,\quad N^{hit}_{shad} = 37577,\qquad T^{MIN}_{R}[s] = 4.01,\quad T_{app}[s] = 3.82,\quad T^{MIN}_{RSA}[s] = 0.19.$$

Scene = “*tetra5*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	1024	–	0	0	0	0.01	121.80	26.58	1.0	3690.91
1	spatmed-xyz(16,2)	13915	13916	4140	46080	43.38	28.85	5.62	3.85	0.18	3.09	26.58	0.40	67.06
2	objmed-xyz(16,2)	735	736	480	1024	16.17	18.41	3.70	2.93	0.04	2.01	26.58	0.28	34.33
3	objmed(16,2)	831	832	576	1024	10.62	27.52	6.02	5.51	0.07	2.17	26.58	0.15	39.18
4	OSAH(16,2)	751	752	496	1024	10.62	11.76	2.32	1.81	0.07	1.80	26.58	0.29	27.97
5	OSAH-RMI(16,2)	751	752	496	1024	10.62	11.76	2.32	1.81	0.07	1.69	26.58	0.29	24.64
6	OSAH-xyz(16,2)	727	728	472	1024	11.35	13.32	2.68	2.15	0.05	1.74	26.58	0.28	26.15
7	OSAH(8,1)	171	172	60	1024	41.69	9.05	1.97	1.12	0.05	2.22	26.58	0.67	40.70
8	OSAH(8,2)	171	172	60	1024	41.69	9.05	1.97	1.12	0.05	2.22	26.58	0.67	40.70
9	OSAH(16,1)	751	752	496	1024	10.62	11.76	2.32	1.81	0.07	1.70	26.58	0.29	24.94
10	OSAH(16,2)	751	752	496	1024	10.62	11.76	2.32	1.81	0.07	1.80	26.58	0.29	27.97
11	OSAH(24,1)	751	752	496	1024	10.62	11.76	2.32	1.81	0.07	1.71	26.58	0.29	25.24
12	OSAH(24,2)	751	752	496	1024	10.62	11.76	2.32	1.81	0.06	1.70	26.58	0.29	24.94
13	OSAH(atc)	751	752	496	1024	10.62	11.76	2.32	1.81	0.07	1.69	26.58	0.29	24.64
14	OSAH2(atc)	747	748	492	1024	10.62	13.31	2.40	1.89	0.09	1.69	26.58	0.26	24.64
15	OSAH+LC(atc)	751	752	496	1024	10.62	11.76	2.32	1.81	0.06	1.68	26.58	0.29	24.33
16	OSAH+TPC(atc)	751	752	496	1024	10.62	11.79	2.32	1.81	0.07	1.69	26.58	0.29	24.64
17	OSAH+TPC+LC(atc)	751	752	496	1024	10.62	11.79	2.32	1.81	0.07	1.58	26.58	0.29	21.30
18	OSAH+LC(16,1)	751	752	496	1024	10.62	11.76	2.32	1.81	0.08	1.57	26.58	0.29	21.00
19	OSAH+TPC(16,1)	751	752	496	1024	10.62	11.79	2.32	1.81	0.07	1.57	26.58	0.29	21.00
20	OSAH+TPC+LC(16,1)	751	752	496	1024	10.62	11.79	2.32	1.81	0.08	1.57	26.58	0.29	21.00
21	OSAH+PR(atc)	751	752	496	1024	10.62	11.76	2.32	1.81	0.07	1.69	26.58	0.29	24.64
22	OSAH+SC(atc)	751	752	496	1024	10.62	11.76	2.32	1.81	0.08	1.63	26.58	0.29	22.82
23	OSAH+GCM(atc)	839	840	584	1024	10.62	12.79	2.44	1.94	2.55	1.73	26.58	0.27	25.85
24	OSAH+GCM2(atc)	839	840	584	1024	10.62	12.79	2.44	1.94	2.59	1.72	26.58	0.27	25.55
25	OSAH+GCM3(atc)	887	888	632	1024	11.05	16.46	3.02	2.50	2.80	1.75	26.58	0.23	26.45
26	OSAH+PAR(atc)	751	752	496	1024	8.37	10.32	2.12	1.75	0.08	1.38	17.67	0.60	28.33
27	PARSAH+PAR(atc)	803	804	548	1024	8.37	11.14	2.31	1.93	0.07	1.27	17.67	0.50	24.67
28	OSAH+PER(atc)	751	752	496	1024	7.53	10.54	2.15	1.76	0.08	1.30	31.00	0.49	34.00
29	PERSAH+PER(atc)	840	841	585	1024	7.53	11.28	2.34	1.96	1.99	1.42	31.00	0.53	40.00
30	SPHSAH+PER(atc)	641	642	425	1024	23.15	13.46	2.64	2.09	0.13	1.70	31.00	0.59	54.00
31	OSAH+SPH(atc)	751	752	496	1024	7.52	10.41	2.12	1.74	0.06	1.67	23.25	0.54	18.50
32	SPHSAH+SPH(atc)	641	642	425	1024	22.91	13.27	2.60	2.06	0.13	2.05	23.25	0.61	28.00
33	OSAH+TA$_{seq}$(16,2)	751	752	496	1024	10.62	23.98	2.32	1.81	0.05	2.16	26.58	0.15	38.88
34	OSAH+TA$^{A}_{rec}$(16,2)	751	752	496	1024	10.62	11.76	2.32	1.81	0.05	1.71	26.58	0.23	25.24
35	OSAH+TA$^{B}_{rec}$(16,2)	751	752	496	1024	10.62	11.76	2.32	1.81	0.06	1.55	26.58	0.29	20.39
36	OSAH+TA$_{SNL}$(16,2)	751	752	496	1024	10.62	11.15	2.32	1.81	0.06	1.65	26.58	0.25	23.42
37	OSAH+TA$_{NLT}$(16,2)	2427	752	496	1024	10.62	10.46	2.32	1.81	0.07	1.64	26.58	0.26	23.12
38	OSAH+TA$_{seq}$(18,2)	751	752	496	1024	10.62	23.98	2.32	1.81	0.04	2.15	26.58	0.15	38.58
39	OSAH+TA$^{A}_{rec}$(18,2)	751	752	496	1024	10.62	11.76	2.32	1.81	0.05	1.72	26.58	0.23	25.55
40	OSAH+TA$^{B}_{rec}$(18,2)	751	752	496	1024	10.62	11.76	2.32	1.81	0.05	1.55	26.58	0.29	20.39
41	OSAH+TA$_{SNL}$(18,2)	751	752	496	1024	10.62	11.15	2.32	1.81	0.06	1.64	26.58	0.26	23.12
42	OSAH+TA$_{NLT}$(18,2)	2427	752	496	1024	10.62	10.46	2.32	1.81	0.07	1.64	26.58	0.26	23.12
43	OSAH+TA$_{seq}$(atc)	751	752	496	1024	10.62	23.98	2.32	1.81	0.05	2.17	26.58	0.15	39.18
44	OSAH+TA$^{A}_{rec}$(atc)	751	752	496	1024	10.62	11.76	2.32	1.81	0.06	1.72	26.58	0.23	25.55
45	OSAH+TA$^{B}_{rec}$(atc)	751	752	496	1024	10.62	11.76	2.32	1.81	0.05	1.55	26.58	0.29	20.39
46	OSAH+TA$_{SNL}$(atc)	751	752	496	1024	10.62	11.15	2.32	1.81	0.06	1.64	26.58	0.26	23.12
47	OSAH+TA$_{NLT}$(atc)	2427	752	496	1024	10.62	10.46	2.32	1.81	0.07	1.64	26.58	0.26	23.12
48	BVH	102	623	0	1024	88.29	21.10	16.86	0.00	0.09	48.29	–	–	–
49	O84	6997	48980	19428	128000	47.59	44.16	7.61	5.53	0.31	6.98	–	–	–
50	O89	6997	48980	19428	128000	47.49	29.92	7.61	5.53	0.27	5.34	–	–	–
51	BSP	13915	13916	4140	46080	43.38	28.85	5.62	3.85	0.30	4.95	–	–	–
52	O93	6997	48980	19428	128000	45.83	25.13	9.36	7.35	0.28	6.55	–	–	–
53	UG	0	5832	4456	8984	51.51	8.05	8.05	6.47	0.06	3.56	–	–	–
54	AG	151	5941	2791	16868	72.05	9.69	8.29	5.45	0.19	6.66	–	–	–
55	HUG	1	3375	2429	6736	63.02	7.41	6.74	5.08	0.07	4.51	–	–	–
56	RG	261	5164	1116	30784	73.71	6.24	5.08	3.03	0.05	4.67	–	–	–
57	O84A	7257	50800	18656	138944	29.86	33.90	6.26	4.87	0.91	5.49	–	–	–
58	KD	751	752	496	1024	10.62	11.76	2.32	1.81	0.10	2.48	–	–	–

Table 9: Experimental results for scene “*tetra5*”.

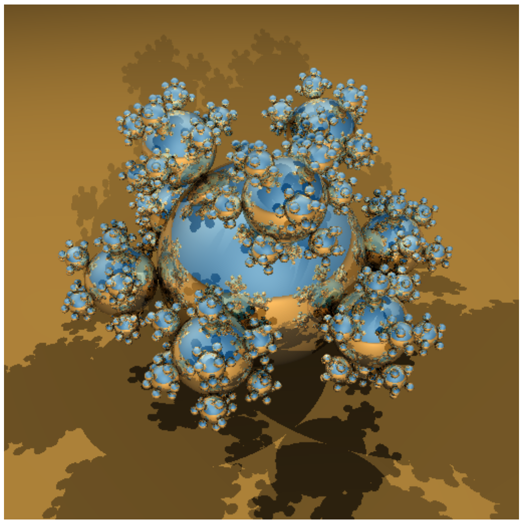
$$N = 1024,$$
$$TP_D: \quad N_{prim} = 263169, \quad N^{hit}_{\mathcal{AB}} = 159213, \quad N^{hit}_{prim} = 53807, \quad N_{sec} = 0, \quad N^{hit}_{sec} = 0,$$
$$N_{shad} = 50135, \quad N^{hit}_{shad} = 4650, \quad T^{MIN}_{R}[s] = 0.91, \quad T_{app}[s] = 0.877, \quad T^{MIN}_{RSA}[s] = 0.033.$$

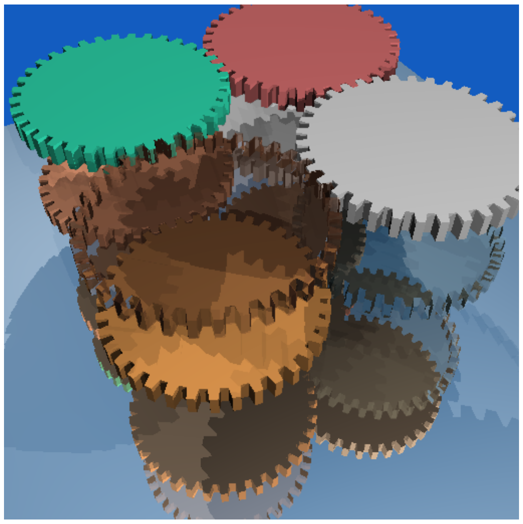
Scene = “*tree8*”

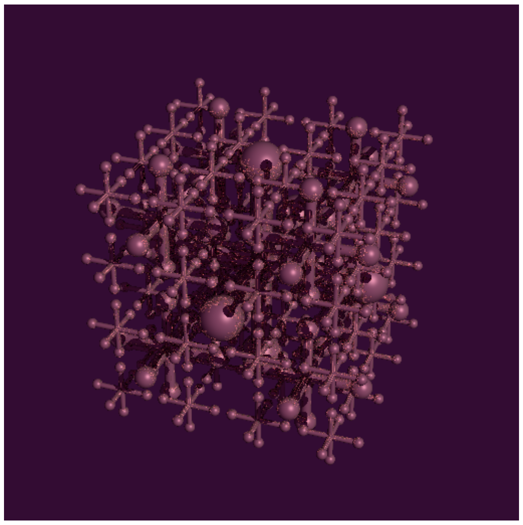
Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	1023	–	0	0	0	0.01	1544.43	18.67	1.0	7354.43
1	spatmed-xyz(16,2)	307	308	218	1569	480.05	61.95	12.99	6.69	0.02	109.94	18.67	0.76	504.86
2	objmed-xyz(16,2)	4427	4428	661	7239	198.68	87.03	19.39	3.33	0.14	58.63	18.67	0.48	260.52
3	objmed(16,2)	5041	5042	809	8477	342.38	125.39	30.85	5.32	0.22	102.18	18.67	0.52	467.90
4	OSAH(16,2)	1107	1108	312	1930	21.51	11.96	3.17	0.48	0.16	10.09	18.67	0.42	29.38
5	OSAH-RMI(16,2)	1107	1108	312	1930	21.51	11.96	3.17	0.48	0.15	9.89	18.67	0.42	28.43
6	OSAH-xyz(16,2)	1114	1115	334	1961	19.34	13.98	3.90	1.65	0.08	10.48	18.67	0.36	31.24
7	OSAH(8,1)	57	58	8	1126	99.09	9.95	2.84	0.12	0.09	22.95	18.67	0.80	90.62
8	OSAH(8,2)	54	55	7	1126	100.14	9.69	2.79	0.12	0.08	22.99	18.67	0.80	90.81
9	OSAH(16,1)	1433	1434	545	2019	19.16	12.59	3.28	0.59	0.15	9.63	18.67	0.38	27.19
10	OSAH(16,2)	1107	1108	312	1930	21.51	11.96	3.17	0.48	0.16	10.09	18.67	0.42	29.38
11	OSAH(24,1)	8578	8579	1747	9546	18.47	13.03	3.36	0.62	0.33	9.66	18.67	0.36	27.33
12	OSAH(24,2)	3306	3307	376	5649	20.99	12.15	3.20	0.48	0.22	9.81	18.67	0.41	28.05
13	OSAH(atc)	521	522	192	1416	22.98	12.04	3.19	0.51	0.12	10.43	18.67	0.43	31.00
14	OSAH2(atc)	900	901	234	1818	20.10	13.19	3.39	0.91	0.16	10.18	18.67	0.38	29.81
15	OSAH+LC(atc)	527	528	192	1431	22.99	12.06	3.19	0.51	0.12	10.48	18.67	0.43	31.24
16	OSAH+TPC(atc)	514	515	193	1403	23.03	11.98	3.17	0.51	0.14	10.30	18.67	0.43	30.38
17	OSAH+TPC+LC(atc)	538	539	193	1463	22.95	12.04	3.18	0.51	0.14	9.54	18.67	0.43	26.76
18	OSAH+LC(16,1)	1433	1434	545	2019	19.16	12.59	3.28	0.59	0.18	9.07	18.67	0.38	24.52
19	OSAH+TPC(16,1)	1438	1439	548	2021	19.11	12.58	3.27	0.59	0.19	8.94	18.67	0.38	23.90
20	OSAH+TPC+LC(16,1)	1438	1439	548	2021	19.11	12.58	3.27	0.59	0.19	8.99	18.67	0.38	24.14
21	OSAH+PR(atc)	521	522	192	1388	23.04	11.75	3.11	0.52	0.14	9.78	18.67	0.43	27.90
22	OSAH+SC(atc)	512	513	228	1335	22.15	11.97	3.18	0.61	0.17	9.91	18.67	0.42	28.52
23	OSAH+GCM(atc)	569	570	220	1436	22.33	12.21	3.20	0.58	1.96	9.81	18.67	0.42	28.05
24	OSAH+GCM2(atc)	880	881	180	2377	23.97	13.65	3.47	0.56	3.24	10.53	18.67	0.41	31.48
25	OSAH+GCM3(atc)	548	549	110	1895	47.63	16.19	3.56	0.43	2.23	15.19	18.67	0.54	53.67
26	OSAH+PAR(atc)	521	522	192	1416	4.75	15.97	4.46	0.82	0.14	5.60	18.24	0.57	8.43
27	PARSAH+PAR(atc)	566	567	227	1506	3.38	12.66	3.40	0.84	0.19	5.18	18.24	0.55	6.43
28	OSAH+PER(atc)	521	522	192	1416	8.33	18.59	4.95	1.08	0.14	4.48	17.87	0.51	12.00
29	PERSAH+PER(atc)	847	848	329	1827	10.11	14.83	4.12	1.94	34.83	4.62	17.87	0.63	12.93
30	SPHSAH+PER(atc)	639	640	227	2175	10.68	15.63	4.18	1.70	1.23	4.93	17.87	0.67	15.00
31	OSAH+SPH(atc)	521	522	192	1416	8.28	18.43	4.90	1.07	0.13	4.76	16.53	0.55	11.47
32	SPHSAH+SPH(atc)	639	640	227	2175	10.66	15.55	4.16	1.68	1.24	5.21	16.53	0.69	14.12
33	OSAH+TA$_{seq}$(16,2)	1107	1108	312	1930	21.60	25.40	3.18	0.48	0.13	13.13	18.67	0.25	43.86
34	OSAH+TA$^{A}_{rec}$(16,2)	1107	1108	312	1930	21.51	11.96	3.17	0.48	0.12	9.99	18.67	0.38	28.90
35	OSAH+TA$^{B}_{rec}$(16,2)	1107	1108	312	1930	21.51	11.96	3.17	0.48	0.12	9.35	18.67	0.42	25.86
36	OSAH+TA$_{SNL}$(16,2)	1107	1108	312	1930	21.61	11.82	3.18	0.48	0.15	9.77	18.67	0.39	27.86
37	OSAH+TA$_{NLT}$(16,2)	7341	1108	312	1930	21.61	10.99	3.18	0.48	0.14	10.42	18.67	0.35	30.95
38	OSAH+TA$_{seq}$(18,2)	1630	1631	356	2554	21.17	25.81	3.20	0.48	0.15	13.12	18.67	0.24	43.81
39	OSAH+TA$^{A}_{rec}$(18,2)	1630	1631	356	2554	21.08	12.06	3.19	0.48	0.14	9.95	18.67	0.36	28.71
40	OSAH+TA$^{B}_{rec}$(18,2)	1630	1631	356	2554	21.08	12.06	3.19	0.48	0.15	9.29	18.67	0.41	25.57
41	OSAH+TA$_{SNL}$(18,2)	1630	1631	356	2554	21.18	11.92	3.20	0.48	0.14	9.71	18.67	0.38	27.57
42	OSAH+TA$_{NLT}$(18,2)	10377	1631	356	2554	21.18	11.08	3.20	0.48	0.18	9.87	18.67	0.37	28.33
43	OSAH+TA$_{seq}$(atc)	521	522	192	1416	23.07	25.16	3.19	0.51	0.10	13.55	18.67	0.26	45.86
44	OSAH+TA$^{A}_{rec}$(atc)	521	522	192	1416	22.98	12.04	3.19	0.51	0.10	10.37	18.67	0.39	30.71
45	OSAH+TA$^{B}_{rec}$(atc)	521	522	192	1416	22.98	12.04	3.19	0.51	0.10	9.70	18.67	0.43	27.52
46	OSAH+TA$_{SNL}$(atc)	521	522	192	1416	23.08	11.89	3.20	0.51	0.10	10.69	18.67	0.37	32.24
47	OSAH+TA$_{NLT}$(atc)	3644	522	192	1416	23.08	11.06	3.20	0.51	0.12	10.29	18.67	0.39	30.33
48	BVH	163	672	0	1023	32.72	10.33	6.59	0.00	0.08	99.60	–	–	–
49	O84	145	1016	859	2013	419.41	128.56	21.27	14.00	0.03	176.30	–	–	–
50	O89	145	1016	859	2013	418.74	83.80	21.27	14.00	0.03	152.31	–	–	–
51	BSP	307	308	218	1569	480.05	61.95	12.99	6.69	1.31	232.96	–	–	–
52	O93	145	1016	859	2013	418.25	71.05	28.38	21.12	0.03	167.79	–	–	–
53	UG	0	4324	2158	3217	1137.20	6.29	6.29	2.87	0.04	299.79	–	–	–
54	AG	170	6285	4014	5248	19.75	7.05	6.40	4.62	0.13	18.49	–	–	–
55	HUG	10	3064	2322	2550	55.55	4.53	1.94	1.18	0.06	35.93	–	–	–
56	RG	45	3511	1778	4177	53.77	14.39	11.75	6.48	0.02	29.01	–	–	–
57	O84A	471	3298	1364	4836	22.74	31.57	7.99	5.62	0.12	28.84	–	–	–
58	KD	1107	1108	312	1930	21.51	11.96	3.17	0.48	0.34	18.39	–	–	–

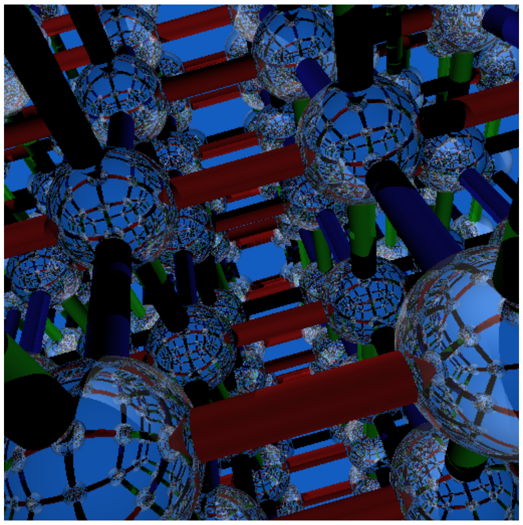
Table 10: Experimental results for scene “*tree8*”.

$$N = 1023,$$
$$TP_D\colon\ \ N_{prim} = 263169,\ \ N^{hit}_{\mathcal{AB}} = 263169,\ \ N^{hit}_{prim} = 166900,\quad N_{sec} = 0,\ \ N^{hit}_{sec} = 0,$$
$$N_{shad} = 1088879,\ \ N^{hit}_{shad} = 632,\quad T^{MIN}_{R}[s] = 4.13,\ \ T_{app}[s] = 3.92,\ \ T^{MIN}_{RSA}[s] = 0.21.$$


balls3


gears2


jacks3


lattice6

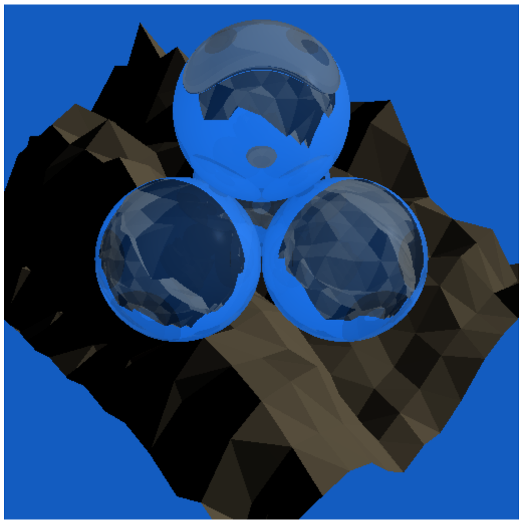
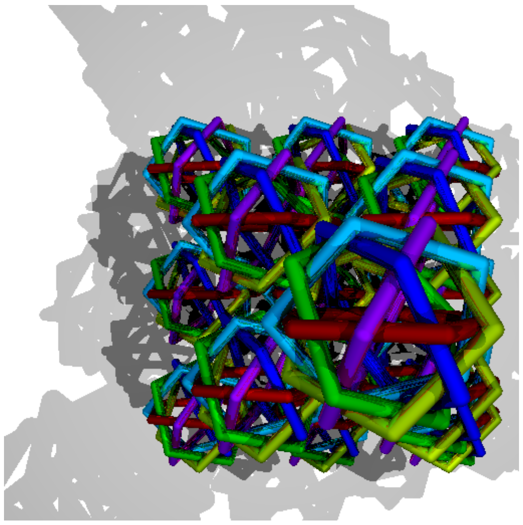
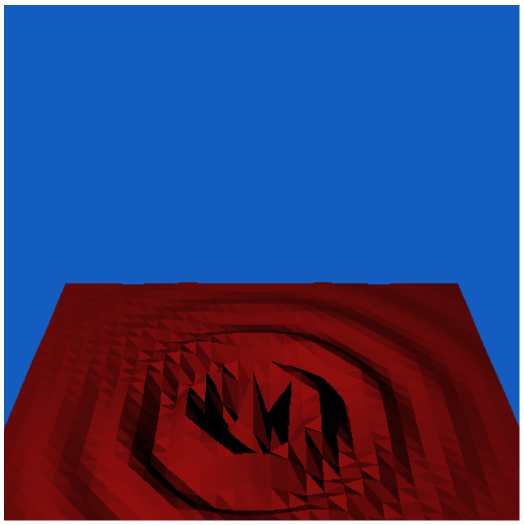
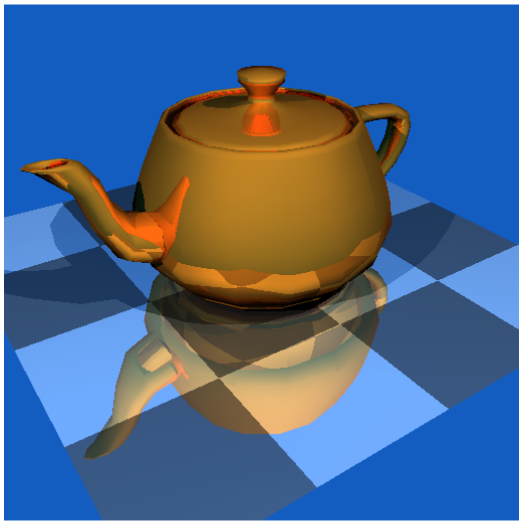
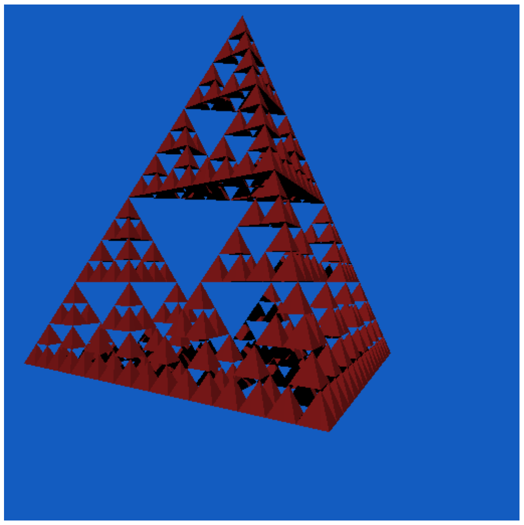

mount4

Figure 1: Visualization of the G^3_{SPD} scenes using the testing procedure TP_D.


rings3


sombrero1


teapot4


tetra5

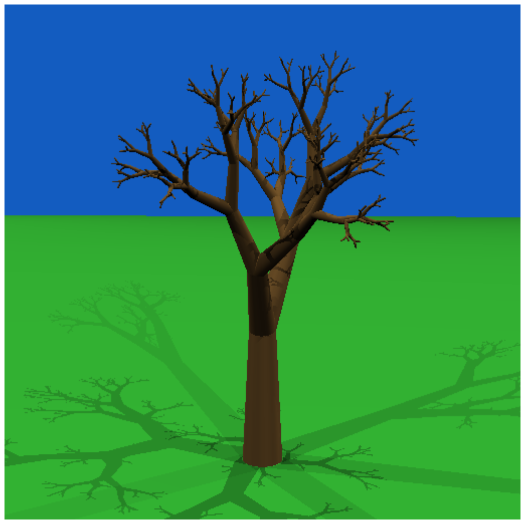

tree8

Figure 2: Visualization of the G^3_{SPD} scenes using the testing procedure TP_D.

Scene = “*balls4*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	7382	–	0	0	0	0.02	6248.25	15.05	1.0	16887.16
1	spatmed-xyz(16,2)	583	584	258	11433	330.66	56.74	10.76	5.96	0.18	81.39	15.05	0.77	204.92
2	objmed-xyz(16,2)	31398	31399	1213	64190	104.21	133.92	28.30	2.98	1.24	57.94	15.05	0.31	141.54
3	objmed(16,2)	26852	26853	502	55119	109.17	130.18	27.86	1.12	1.71	60.17	15.05	0.32	147.57
4	OSAH(16,2)	4374	4375	746	13784	16.23	24.10	4.77	1.26	1.38	16.07	15.05	0.28	28.38
5	OSAH-RMI(16,2)	4374	4375	746	13784	16.23	24.10	4.77	1.26	1.35	15.68	15.05	0.28	27.32
6	OSAH-xyz(16,2)	4833	4834	828	14839	17.18	25.44	5.13	1.40	0.78	15.84	15.05	0.28	27.76
7	OSAH(8,1)	82	83	6	7931	410.73	13.16	3.12	0.36	0.93	87.10	15.05	0.95	220.35
8	OSAH(8,2)	79	80	5	7930	411.16	12.61	2.99	0.27	0.92	85.93	15.05	0.95	217.19
9	OSAH(16,1)	5243	5244	938	14410	15.01	26.09	5.18	1.51	1.40	15.94	15.05	0.25	28.03
10	OSAH(16,2)	4374	4375	746	13784	16.23	24.10	4.77	1.26	1.38	16.07	15.05	0.28	28.38
11	OSAH(24,1)	64019	64020	9756	79636	9.72	31.83	6.13	1.99	2.85	16.55	15.05	0.15	29.68
12	OSAH(24,2)	29159	29160	2724	51935	11.73	27.47	5.29	1.44	2.12	15.91	15.05	0.19	27.95
13	OSAH(atc)	7892	7893	1479	17323	12.79	27.13	5.34	1.61	1.48	15.74	15.05	0.21	27.49
14	OSAH2(atc)	12720	12721	1641	23484	11.85	33.66	6.02	2.34	2.07	16.61	15.05	0.17	29.84
15	OSAH+LC(atc)	8014	8015	1485	17592	12.41	27.35	5.35	1.80	1.49	15.67	15.05	0.20	27.30
16	OSAH+TPC(atc)	7707	7708	1469	17092	12.81	27.08	5.33	1.61	1.51	15.80	15.05	0.21	27.65
17	OSAH+TPC+LC(atc)	8253	8254	1476	18312	12.41	27.36	5.35	1.80	1.81	14.72	15.05	0.20	24.73
18	OSAH+LC(16,1)	5249	5250	944	14410	14.62	26.29	5.18	1.71	1.58	14.92	15.05	0.24	25.27
19	OSAH+TPC(16,1)	5264	5265	959	14410	14.99	26.11	5.18	1.52	1.67	14.93	15.05	0.25	25.30
20	OSAH+TPC+LC(16,1)	5270	5271	965	14410	14.61	26.31	5.18	1.71	1.67	15.00	15.05	0.24	25.49
21	OSAH+PR(atc)	7892	7893	1479	14565	11.55	27.21	5.37	1.65	1.72	14.81	15.05	0.19	24.97
22	OSAH+SC(atc)	7427	7428	2003	14055	10.94	26.67	5.27	1.94	2.26	15.32	15.05	0.19	26.35
23	OSAH+GCM(atc)	9515	9516	1761	19438	11.99	28.35	5.55	1.85	28.87	15.59	15.05	0.19	27.08
24	OSAH+GCM2(atc)	16783	16784	1067	39824	14.03	31.60	6.02	1.40	49.25	16.78	15.05	0.20	30.30
25	OSAH+GCM3(atc)	12955	12956	1446	27378	16.52	34.89	6.21	1.43	39.85	17.73	15.05	0.21	32.86
26	OSAH+PAR(atc)	7892	7893	1479	17323	6.55	31.18	6.86	2.39	1.66	4.58	12.85	0.31	10.05
27	PARSAH+PAR(atc)	7751	7752	1465	17185	6.41	29.76	6.49	2.09	2.06	4.57	12.85	0.34	10.00
28	OSAH+PER(atc)	7892	7893	1479	17323	6.30	30.98	6.87	2.66	1.69	4.61	14.11	0.34	11.50
29	PERSAH+PER(atc)	13264	13265	2942	22531	4.93	32.53	7.04	3.22	302.22	4.65	14.11	0.32	11.72
30	SPHSAH+PER(atc)	7535	7536	1447	17392	6.42	30.42	6.58	2.46	10.21	4.57	14.11	0.33	11.28
31	OSAH+SPH(atc)	7892	7893	1479	17323	6.21	30.51	6.76	2.61	1.71	4.99	12.30	0.37	9.39
32	SPHSAH+SPH(atc)	7535	7536	1447	17392	6.35	30.02	6.50	2.42	10.27	4.88	12.30	0.35	8.91
33	OSAH+TA_{seq}(16,2)	4374	4375	746	13784	17.23	59.40	4.92	1.27	1.15	21.33	15.05	0.16	42.59
34	OSAH+TA^A_{rec}(16,2)	4374	4375	746	13784	16.23	24.10	4.77	1.26	1.12	16.29	15.05	0.24	28.97
35	OSAH+TA^B_{rec}(16,2)	4374	4375	746	13784	16.23	24.10	4.77	1.26	1.14	14.76	15.05	0.28	24.84
36	OSAH+TA_{SNL}(16,2)	4374	4375	746	13784	17.24	21.84	4.92	1.27	1.21	15.50	15.05	0.26	26.84
37	OSAH+TA_{NLT}(16,2)	31999	4375	746	13784	17.23	19.53	4.92	1.27	1.21	15.87	15.05	0.25	27.84
38	OSAH+TA_{seq}(18,2)	9171	9172	1545	19724	13.66	65.76	5.18	1.38	1.28	21.63	15.05	0.12	43.41
39	OSAH+TA^A_{rec}(18,2)	9171	9172	1545	19724	12.87	25.67	5.01	1.37	1.28	16.30	15.05	0.18	29.00
40	OSAH+TA^B_{rec}(18,2)	9171	9172	1545	19724	12.87	25.67	5.01	1.37	1.28	14.49	15.05	0.22	24.11
41	OSAH+TA_{SNL}(18,2)	9171	9172	1545	19724	13.68	23.28	5.19	1.38	1.44	15.35	15.05	0.20	26.43
42	OSAH+TA_{NLT}(18,2)	65310	9172	1545	19724	13.67	20.70	5.19	1.38	1.53	15.49	15.05	0.20	26.81
43	OSAH+TA_{seq}(atc)	7892	7893	1479	17323	13.56	69.73	5.50	1.61	1.22	22.37	15.05	0.11	45.41
44	OSAH+TA^A_{rec}(atc)	7892	7893	1479	17323	12.78	27.13	5.34	1.61	1.22	16.53	15.05	0.17	29.62
45	OSAH+TA^B_{rec}(atc)	7892	7893	1479	17323	12.79	27.13	5.34	1.61	1.22	14.69	15.05	0.21	24.65
46	OSAH+TA_{SNL}(atc)	7892	7893	1479	17323	13.59	24.46	5.51	1.61	1.35	15.56	15.05	0.19	27.00
47	OSAH+TA_{NLT}(atc)	56963	7893	1479	17323	13.57	21.66	5.50	1.61	1.41	15.66	15.05	0.19	27.27
48	BVH	1107	4967	0	7382	122.94	141.01	97.54	0.00	0.80	1251.74	–	–	–
49	O84	295	2066	1032	15491	210.05	118.54	19.17	12.37	0.22	120.07	–	–	–
50	O89	295	2066	1032	15491	211.91	75.40	19.16	12.37	0.21	96.63	–	–	–
51	BSP	583	584	258	11433	330.66	56.74	10.76	5.96	1.03	164.56	–	–	–
52	O93	295	2066	1032	15497	207.90	64.76	25.30	18.59	0.22	116.43	–	–	–
53	UG	0	40836	33867	15959	215.52	10.25	10.25	7.11	0.26	78.59	–	–	–
54	AG	1762	48624	13627	58410	23.49	13.37	11.57	7.41	3.16	32.25	–	–	–
55	HUG	21	22226	17429	17954	67.25	15.51	11.92	8.61	0.42	73.71	–	–	–
56	RG	1578	40061	11669	122122	31.07	16.32	12.63	7.41	0.36	38.12	–	–	–
57	O84A	2291	16038	3486	34692	16.57	46.88	9.12	4.97	1.05	37.96	–	–	–
58	KD	4374	4375	746	13784	16.23	24.10	4.77	1.26	1.78	27.06	–	–	–

Table 11: Experimental results for scene “*balls4*”.

$$N = 7382,$$
$$TP_D\!: \quad N_{prim} = 263169, \quad N^{hit}_{\mathcal{AB}} = 263169, \quad N^{hit}_{prim} = 263169, \quad N_{sec} = 179881, \quad N^{hit}_{sec} = 134360,$$
$$N_{shad} = 959197, \quad N^{hit}_{shad} = 285156, \quad T^{MIN}_{R}[s] = 5.94, \quad T_{app}[s] = 5.57, \quad T^{MIN}_{RSA}[s] = 0.37.$$

Scene = “*gears4*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	9345	–	0	0	0	0.01	11753.20	6.93	1.0	9794.33
1	spatmed-xyz(16,2)	7647	7648	800	61644	29.81	23.49	3.42	0.89	0.41	29.66	6.93	0.62	17.78
2	objmed-xyz(16,2)	29232	29233	1942	88854	45.42	63.23	12.48	3.38	1.66	47.83	6.93	0.48	32.92
3	objmed(16,2)	44931	44932	619	100913	72.90	95.02	18.89	0.76	2.50	73.31	6.93	0.50	54.16
4	OSAH(16,2)	10703	10704	254	34667	9.85	19.73	3.04	0.55	1.60	21.78	6.93	0.39	11.22
5	OSAH-RMI(16,2)	10703	10704	254	34667	9.85	19.73	3.04	0.55	1.61	21.43	6.93	0.39	10.93
6	OSAH-xyz(16,2)	6487	6488	366	24030	11.92	24.96	4.43	2.12	0.86	23.23	6.93	0.38	12.43
7	OSAH(8,1)	155	156	0	10146	148.88	10.84	1.99	0.00	1.03	69.48	6.93	0.95	50.97
8	OSAH(8,2)	155	156	0	10146	148.88	10.84	1.99	0.00	0.91	69.67	6.93	0.95	51.12
9	OSAH(16,1)	13258	13259	399	35902	8.41	21.02	3.19	0.64	1.63	21.00	6.93	0.34	10.57
10	OSAH(16,2)	10703	10704	254	34667	9.85	19.73	3.04	0.55	1.60	21.78	6.93	0.39	11.22
11	OSAH(24,1)	123517	123518	5805	187329	6.09	23.81	3.62	0.76	4.55	21.04	6.93	0.25	10.60
12	OSAH(24,2)	64863	64864	1509	137422	8.29	21.23	3.23	0.58	3.06	21.22	6.93	0.33	10.75
13	OSAH(atc)	27471	27472	964	53681	7.55	21.85	3.29	0.68	1.99	21.26	6.93	0.31	10.78
14	OSAH2(atc)	28542	28543	1158	57549	7.14	23.89	3.57	1.00	2.87	21.38	6.93	0.28	10.88
15	OSAH+LC(atc)	35245	35246	1133	71987	7.33	22.37	3.35	0.69	2.51	21.31	6.93	0.30	10.82
16	OSAH+TPC(atc)	17292	17293	493	42666	8.22	21.16	3.19	0.64	1.85	21.30	6.93	0.33	10.82
17	OSAH+TPC+LC(atc)	33368	33369	646	74810	7.71	22.05	3.29	0.65	3.00	19.96	6.93	0.31	9.70
18	OSAH+LC(16,1)	13310	13311	451	35902	8.41	21.02	3.19	0.65	1.83	20.24	6.93	0.34	9.93
19	OSAH+TPC(16,1)	13592	13593	794	36039	8.37	21.06	3.19	0.69	1.98	20.08	6.93	0.34	9.80
20	OSAH+TPC+LC(16,1)	13621	13622	823	36039	8.37	21.06	3.19	0.69	1.86	20.69	6.93	0.34	10.31
21	OSAH+PR(atc)	27471	27472	964	45855	7.37	21.86	3.31	0.68	5.18	20.36	6.93	0.30	10.03
22	OSAH+SC(atc)	27823	27824	1034	53622	7.59	21.92	3.30	0.68	3.43	21.10	6.93	0.31	10.65
23	OSAH+GCM(atc)	40823	40824	1429	74910	7.33	23.29	3.54	0.49	133.51	21.26	6.93	0.29	10.78
24	OSAH+GCM2(atc)	67659	67660	3167	125698	8.49	27.24	4.02	0.71	329.59	22.52	6.93	0.29	11.83
25	OSAH+GCM3(atc)	29857	29858	1773	60050	12.78	31.51	4.57	0.95	108.55	25.07	6.93	0.34	13.96
26	OSAH+PAR(atc)	27471	27472	964	53681	2.83	15.09	2.33	0.31	2.26	5.79	6.76	0.62	3.22
27	PARSAH+PAR(atc)	30419	30420	1190	58126	2.57	14.88	2.25	0.35	2.72	5.90	6.76	0.65	3.41
28	OSAH+PER(atc)	27471	27472	964	53681	3.25	17.26	2.50	0.37	2.27	5.52	6.74	0.57	3.48
29	PERSAH+PER(atc)	32118	32119	1362	60436	2.90	17.38	2.48	0.52	180.33	5.46	6.74	0.55	3.37
30	SPHSAH+PER(atc)	33660	33661	3526	73892	6.10	23.71	3.51	0.98	8.70	6.18	6.74	0.56	4.70
31	OSAH+SPH(atc)	27471	27472	964	53681	3.24	17.03	2.47	0.36	2.24	5.87	6.89	0.59	3.40
32	SPHSAH+SPH(atc)	33660	33661	3526	73892	6.06	23.59	3.50	1.00	8.65	6.50	6.89	0.57	4.51
33	OSAH+TA$_{seq}$(16,2)	10703	10704	254	34667	13.54	39.72	3.12	0.56	1.24	24.99	6.93	0.27	13.89
34	OSAH+TA$^{A}_{rec}$(16,2)	10703	10704	254	34667	9.85	19.73	3.04	0.55	1.27	22.14	6.93	0.33	11.52
35	OSAH+TA$^{B}_{rec}$(16,2)	10703	10704	254	34667	9.85	19.73	3.04	0.55	1.36	19.95	6.93	0.39	9.69
36	OSAH+TA$_{SNL}$(16,2)	10703	10704	254	34667	13.54	17.96	3.12	0.56	1.42	20.37	6.93	0.38	10.04
37	OSAH+TA$_{NLT}$(16,2)	52747	10704	254	34667	13.54	16.98	3.12	0.56	1.48	21.41	6.93	0.35	10.91
38	OSAH+TA$_{seq}$(18,2)	23394	23395	371	57919	11.71	42.84	3.23	0.57	1.76	24.79	6.93	0.25	13.72
39	OSAH+TA$^{A}_{rec}$(18,2)	23394	23395	371	57919	8.80	20.75	3.16	0.56	1.59	22.22	6.93	0.30	11.58
40	OSAH+TA$^{B}_{rec}$(18,2)	23394	23395	371	57919	8.80	20.75	3.16	0.56	1.59	20.13	6.93	0.35	9.84
41	OSAH+TA$_{SNL}$(18,2)	23394	23395	371	57919	11.71	18.81	3.23	0.57	2.03	19.94	6.93	0.36	9.68
42	OSAH+TA$_{NLT}$(18,2)	105680	23395	371	57919	11.71	17.68	3.23	0.57	2.14	20.39	6.93	0.34	10.06
43	OSAH+TA$_{seq}$(atc)	27471	27472	964	53681	10.18	45.22	3.34	0.68	1.67	24.58	6.93	0.22	13.55
44	OSAH+TA$^{A}_{rec}$(atc)	27471	27472	964	53681	7.56	21.85	3.29	0.68	1.66	22.24	6.93	0.26	11.60
45	OSAH+TA$^{B}_{rec}$(atc)	27471	27472	964	53681	7.55	21.85	3.29	0.68	1.67	20.10	6.93	0.31	9.82
46	OSAH+TA$_{SNL}$(atc)	27471	27472	964	53681	10.18	19.61	3.34	0.68	2.16	19.65	6.93	0.32	9.44
47	OSAH+TA$_{NLT}$(atc)	120779	27472	964	53681	10.18	18.15	3.34	0.68	2.28	20.05	6.93	0.31	9.78
48	BVH	1166	6104	0	9345	123.90	131.42	95.25	0.00	1.76	1764.88	–	–	–
49	O84	4033	28232	10996	111612	23.11	28.43	5.15	2.50	0.62	47.29	–	–	–
50	O89	4033	28232	10996	111612	22.10	22.01	5.10	2.50	0.51	41.45	–	–	–
51	BSP	7647	7648	800	61644	29.81	23.49	3.42	0.89	1.54	93.96	–	–	–
52	O93	4217	29520	12092	114496	18.78	17.73	6.62	4.45	0.56	46.64	–	–	–
53	UG	0	46284	35148	39226	22.86	12.15	12.15	8.96	0.44	38.07	–	–	–
54	AG	2374	64448	5992	126321	14.90	5.14	3.78	0.25	5.12	48.29	–	–	–
55	HUG	25	27984	17304	50030	22.89	11.36	6.37	3.26	0.89	91.52	–	–	–
56	RG	8503	117394	28116	512128	23.41	11.62	9.04	5.52	1.31	50.42	–	–	–
57	O84A	6937	48560	11408	148456	13.69	33.34	6.38	3.64	2.32	45.75	–	–	–
58	KD	10703	10704	254	34667	9.85	19.73	3.04	0.55	2.38	36.24	–	–	–

Table 12: Experimental results for scene “*gears4*”.

$$N = 9345,$$
$$TP_D: \quad N_{prim} = 263169, \quad N^{hit}_{\mathcal{AB}} = 263169, \quad N^{hit}_{prim} = 332, \quad N_{sec} = 181015, \quad N^{hit}_{sec} = 122023,$$
$$N_{shad} = 1388067, \quad N^{hit}_{shad} = 356484, \quad T^{MIN}_{R}[s] = 9.52, \quad T_{app}[s] = 8.32, \quad T^{MIN}_{RSA}[s] = 1.20.$$

Scene = “*jacks4*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	5265	–	0	0	0	0.01	2844.89	8.56	1.0	8890.28
1	spatmed-xyz(16,2)	20041	20042	1042	60390	36.75	40.56	7.49	1.47	0.36	17.95	8.56	0.65	47.53
2	objmed-xyz(16,2)	32567	32568	1664	70776	44.85	58.12	11.09	1.42	1.12	22.03	8.56	0.61	60.28
3	objmed(16,2)	29962	29963	645	64557	52.25	60.54	11.98	0.24	1.62	24.66	8.56	0.64	68.50
4	OSAH(16,2)	13793	13794	1343	28098	24.62	36.71	6.70	1.42	1.25	14.73	8.56	0.58	37.47
5	OSAH-RMI(16,2)	13793	13794	1343	28098	24.62	36.71	6.70	1.42	1.24	14.45	8.56	0.58	36.59
6	OSAH-xyz(16,2)	14392	14393	1384	29831	24.93	36.91	6.73	1.56	0.76	14.45	8.56	0.58	36.59
7	OSAH(8,1)	239	240	30	7969	180.92	15.09	3.00	0.71	0.61	38.40	8.56	0.96	111.44
8	OSAH(8,2)	227	228	18	7969	181.64	14.72	2.91	0.45	0.61	38.38	8.56	0.96	111.38
9	OSAH(16,1)	20374	20375	4485	31264	19.68	41.58	7.58	3.05	1.38	13.85	8.56	0.49	34.72
10	OSAH(16,2)	13793	13794	1343	28098	24.62	36.71	6.70	1.42	1.25	14.73	8.56	0.58	37.47
11	OSAH(24,1)	92622	92623	7933	133860	18.63	53.91	9.64	3.36	3.25	15.82	8.56	0.41	40.87
12	OSAH(24,2)	46495	46496	1731	94355	24.93	43.23	7.80	1.46	2.18	15.85	8.56	0.54	40.97
13	OSAH(atc)	17878	17879	4883	25902	21.04	39.51	7.20	3.08	1.33	13.70	8.56	0.52	34.25
14	OSAH2(atc)	27834	27835	3456	39802	20.86	45.75	8.06	2.78	2.09	14.78	8.56	0.48	37.62
15	OSAH+LC(atc)	28499	28500	4883	52653	21.83	43.52	7.92	3.08	1.78	14.79	8.56	0.50	37.66
16	OSAH+TPC(atc)	10982	10983	2929	20034	24.15	35.72	6.52	2.71	1.27	13.90	8.56	0.58	34.88
17	OSAH+TPC+LC(atc)	25060	25061	2939	54064	24.90	41.80	7.61	2.71	2.06	14.94	8.56	0.55	38.12
18	OSAH+LC(16,1)	20374	20375	4485	31264	19.68	41.58	7.58	3.05	1.63	13.56	8.56	0.49	33.81
19	OSAH+TPC(16,1)	20419	20420	4506	31306	19.66	41.57	7.56	3.04	1.68	13.59	8.56	0.49	33.91
20	OSAH+TPC+LC(16,1)	20419	20420	4506	31306	19.66	41.57	7.56	3.04	1.71	13.60	8.56	0.49	33.94
21	OSAH+PR(atc)	17878	17879	4883	15941	14.82	39.71	7.24	3.08	2.25	11.98	8.56	0.43	28.88
22	OSAH+SC(atc)	20021	20022	11476	14186	8.90	40.38	7.36	4.78	2.56	10.32	8.56	0.31	23.69
23	OSAH+GCM(atc)	32010	32011	6011	45479	18.83	45.63	8.22	3.11	89.81	14.22	8.56	0.46	35.88
24	OSAH+GCM2(atc)	71773	71774	1265	130911	24.07	53.33	9.38	1.98	201.24	17.24	8.56	0.48	45.31
25	OSAH+GCM3(atc)	47308	47309	3604	79081	24.60	53.50	9.08	2.57	140.55	16.74	8.56	0.48	43.75
26	OSAH+PAR(atc)	17878	17879	4883	25902	11.14	24.19	4.71	2.86	1.54	2.64	7.17	0.40	14.83
27	PARSAH+PAR(atc)	35331	35332	3093	182417	7.22	8.15	0.59	0.16	4.64	2.12	7.17	0.71	10.50
28	OSAH+PER(atc)	17878	17879	4883	25902	12.09	28.95	5.58	3.48	1.64	3.22	6.79	0.43	16.21
29	PERSAH+PER(atc)	18430	18431	4913	32776	8.27	18.73	3.30	2.04	322.41	2.65	6.79	0.51	12.14
30	SPHSAH+PER(atc)	40984	40985	7339	134683	46.45	32.21	6.21	2.93	21.44	6.12	6.79	0.72	36.93
31	OSAH+SPH(atc)	17878	17879	4883	25902	12.06	28.79	5.55	3.45	1.54	3.53	7.84	0.37	10.74
32	SPHSAH+SPH(atc)	40984	40985	7339	134683	46.27	32.02	6.17	2.90	21.43	6.32	7.84	0.71	25.42
33	OSAH+TA$_{seq}$(16,2)	13793	13794	1343	28098	25.22	98.26	6.81	1.42	1.07	18.64	8.56	0.40	49.69
34	OSAH+TA$^{A}_{rec}$(16,2)	13793	13794	1343	28098	24.62	36.71	6.70	1.42	1.04	15.10	8.56	0.52	38.62
35	OSAH+TA$^{B}_{rec}$(16,2)	13793	13794	1343	28098	24.62	36.71	6.70	1.42	1.04	13.80	8.56	0.58	34.56
36	OSAH+TA$_{SNL}$(16,2)	13793	13794	1343	28098	25.23	29.26	6.81	1.42	1.29	15.55	8.56	0.50	40.03
37	OSAH+TA$_{NLT}$(16,2)	107720	13794	1343	28098	25.23	24.12	6.81	1.42	1.38	15.40	8.56	0.51	39.56
38	OSAH+TA$_{seq}$(18,2)	22195	22196	1694	42201	24.82	111.17	7.35	1.46	1.30	19.74	8.56	0.37	53.12
39	OSAH+TA$^{A}_{rec}$(18,2)	22195	22196	1694	42201	24.21	39.72	7.22	1.46	1.23	15.66	8.56	0.49	40.38
40	OSAH+TA$^{B}_{rec}$(18,2)	22195	22196	1694	42201	24.21	39.72	7.22	1.46	1.25	14.23	8.56	0.55	35.91
41	OSAH+TA$_{SNL}$(18,2)	22195	22196	1694	42201	24.83	31.59	7.35	1.46	1.61	16.06	8.56	0.47	41.62
42	OSAH+TA$_{NLT}$(18,2)	158186	22196	1694	42201	24.83	25.93	7.35	1.46	1.76	15.85	8.56	0.48	40.97
43	OSAH+TA$_{seq}$(atc)	17878	17879	4883	25902	21.67	107.51	7.30	3.08	1.17	18.38	8.56	0.34	48.87
44	OSAH+TA$^{A}_{rec}$(atc)	17878	17879	4883	25902	21.04	39.51	7.20	3.08	1.14	14.50	8.56	0.46	36.75
45	OSAH+TA$^{B}_{rec}$(atc)	17878	17879	4883	25902	21.04	39.51	7.20	3.08	1.13	13.05	8.56	0.52	32.22
46	OSAH+TA$_{SNL}$(atc)	17878	17879	4883	25902	21.67	31.43	7.30	3.08	1.42	14.77	8.56	0.45	37.59
47	OSAH+TA$_{NLT}$(atc)	135333	17879	4883	25902	21.67	25.92	7.30	3.08	1.55	14.71	8.56	0.45	37.41
48	BVH	580	3387	0	5265	187.42	145.28	108.17	0.00	0.82	647.28	–	–	–
49	O84	10071	70498	8366	145262	33.89	64.27	10.24	2.92	0.51	31.37	–	–	–
50	O89	10071	70498	8366	145262	33.82	41.11	10.23	2.92	0.49	25.86	–	–	–
51	BSP	20041	20042	1042	60390	36.75	40.56	7.49	1.47	0.63	25.49	–	–	–
52	O93	10071	70498	8366	145262	32.80	39.01	16.72	9.59	0.46	31.06	–	–	–
53	UG	0	25792	16345	36966	38.00	11.86	11.86	7.27	0.32	19.82	–	–	–
54	AG	360	19909	7652	37576	51.68	19.93	17.02	7.61	3.91	39.96	–	–	–
55	HUG	131	16092	9937	21771	59.71	20.02	15.43	7.92	0.36	38.66	–	–	–
56	RG	1627	24050	3588	85556	51.46	12.85	9.60	3.73	0.28	29.39	–	–	–
57	O84A	12833	89832	7534	184006	31.29	65.26	10.54	3.10	1.83	31.04	–	–	–
58	KD	13793	13794	1343	28098	24.62	36.71	6.70	1.42	2.15	20.22	–	–	–

Table 13: Experimental results for scene “*jacks4*”.

$$N = 5265,$$
$$TP_D\text{:}\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 220685,\quad N^{hit}_{prim} = 98411,\quad N_{sec} = 206471,\quad N^{hit}_{sec} = 123014,$$
$$N_{shad} = 162163,\quad N^{hit}_{shad} = 73920,\quad T^{MIN}_{R}[s] = 3.06,\quad T_{app}[s] = 2.74,\quad T^{MIN}_{RSA}[s] = 0.32.$$

Scene = “*lattice12*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	8281	–	0	0	0	0.02	10596.70	8.72	1.0	9057.01
1	spatmed-xyz(16,2)	22015	22016	464	49328	15.01	38.15	6.44	0.20	0.43	38.28	8.72	0.59	24.00
2	objmed-xyz(16,2)	18238	18239	2462	31902	11.65	36.15	6.00	1.01	0.88	31.56	8.72	0.54	18.26
3	objmed(16,2)	14529	14530	0	27981	14.05	35.67	5.96	0.00	1.26	34.89	8.72	0.59	21.10
4	OSAH(16,2)	15225	15226	2044	25350	9.79	34.14	5.71	1.33	1.26	31.45	8.72	0.51	18.16
5	OSAH-RMI(16,2)	15225	15226	2044	25350	9.79	34.14	5.71	1.33	1.26	30.80	8.72	0.51	17.61
6	OSAH-xyz(16,2)	14185	14186	981	25856	10.91	33.35	5.51	0.75	0.82	31.06	8.72	0.54	17.83
7	OSAH(8,1)	255	256	0	11103	103.39	12.23	2.07	0.00	0.72	120.54	8.72	0.97	94.31
8	OSAH(8,2)	255	256	0	11103	103.39	12.23	2.07	0.00	0.71	120.53	8.72	0.97	94.30
9	OSAH(16,1)	32452	32453	5155	39466	5.49	41.93	6.88	3.04	1.63	29.35	8.72	0.32	16.37
10	OSAH(16,2)	15225	15226	2044	25350	9.79	34.14	5.71	1.33	1.26	31.45	8.72	0.51	18.16
11	OSAH(24,1)	61634	61635	5155	68648	4.83	43.32	6.95	3.04	2.28	28.66	8.72	0.29	15.78
12	OSAH(24,2)	15645	15646	2044	25770	9.78	34.16	5.71	1.33	1.26	30.71	8.72	0.51	17.53
13	OSAH(atc)	42624	42625	5155	49638	5.13	42.76	6.93	3.04	1.84	28.86	8.72	0.30	15.95
14	OSAH2(atc)	41769	41770	4139	49812	5.69	42.55	6.86	2.53	2.53	29.60	8.72	0.33	16.58
15	OSAH+LC(atc)	43079	43080	5155	50093	5.12	42.78	6.93	3.04	2.02	29.41	8.72	0.30	16.42
16	OSAH+TPC(atc)	32549	32550	4961	39757	5.79	40.83	6.52	3.19	1.77	28.94	8.72	0.34	16.02
17	OSAH+TPC+LC(atc)	36455	36456	4991	43782	5.38	41.74	6.73	3.01	2.34	27.40	8.72	0.32	14.70
18	OSAH+LC(16,1)	32557	32558	5197	39529	5.36	41.83	6.79	3.12	2.04	27.95	8.72	0.32	15.17
19	OSAH+TPC(16,1)	32756	32757	5245	39680	5.33	41.83	6.77	3.12	1.97	27.77	8.72	0.32	15.02
20	OSAH+TPC+LC(16,1)	32756	32757	5245	39680	5.33	41.83	6.77	3.12	2.08	27.86	8.72	0.32	15.09
21	OSAH+PR(atc)	42624	42625	5155	49638	5.13	42.76	6.93	3.04	2.98	28.89	8.72	0.30	15.97
22	OSAH+SC(atc)	43552	43553	5152	50569	5.06	42.85	6.94	3.08	5.17	29.65	8.72	0.30	16.62
23	OSAH+GCM(atc)	43635	43636	5404	50400	4.77	43.11	6.82	3.22	111.58	28.45	8.72	0.29	15.60
24	OSAH+GCM2(atc)	52846	52847	3776	61268	6.06	45.04	7.23	2.41	134.96	29.51	8.72	0.33	16.50
25	OSAH+GCM3(atc)	44736	44737	3506	53405	7.08	44.19	6.96	1.69	114.56	30.33	8.72	0.37	17.21
26	OSAH+PAR(atc)	42624	42625	5155	49638	18.91	82.86	15.63	9.11	2.15	6.21	8.63	0.31	24.05
27	PARSAH+PAR(atc)	22812	22813	8645	26248	20.02	44.07	10.63	5.19	2.08	5.17	8.63	0.53	18.58
28	OSAH+PER(atc)	42624	42625	5155	49638	5.36	54.13	10.22	5.16	2.16	7.27	7.98	0.42	7.49
29	PERSAH+PER(atc)	14633	14634	3037	22943	316.69	17.46	3.19	0.33	65.51	66.80	7.98	0.99	134.15
30	SPHSAH+PER(atc)	24185	24186	5611	30861	61.92	45.95	8.12	3.43	5.12	23.69	7.98	0.91	42.43
31	OSAH+SPH(atc)	42624	42625	5155	49638	5.35	53.96	10.17	5.12	2.11	7.69	9.16	0.44	8.32
32	SPHSAH+SPH(atc)	24185	24186	5611	30861	59.31	45.80	8.06	3.40	4.91	23.48	9.16	0.91	44.20
33	OSAH+TA$_{seq}$(16,2)	15233	15234	2053	25349	9.84	85.88	5.70	1.37	1.08	38.27	8.72	0.34	23.99
34	OSAH+TA$^{A}_{rec}$(16,2)	15233	15234	2053	25349	9.63	33.90	5.58	1.35	1.06	31.65	8.72	0.44	18.33
35	OSAH+TA$^{B}_{rec}$(16,2)	15233	15234	2053	25349	9.63	33.90	5.58	1.35	1.07	28.68	8.72	0.51	15.79
36	OSAH+TA$_{SNL}$(16,2)	15233	15234	2053	25349	9.84	22.39	5.70	1.37	1.31	29.58	8.72	0.49	16.56
37	OSAH+TA$_{NLT}$(16,2)	58797	15234	2053	25349	9.84	21.18	5.70	1.37	1.38	29.90	8.72	0.48	16.84
38	OSAH+TA$_{seq}$(18,2)	15641	15642	2053	25757	9.83	85.89	5.70	1.37	1.09	38.56	8.72	0.33	24.24
39	OSAH+TA$^{A}_{rec}$(18,2)	15641	15642	2053	25757	9.62	33.90	5.58	1.35	1.09	31.81	8.72	0.44	18.47
40	OSAH+TA$^{B}_{rec}$(18,2)	15641	15642	2053	25757	9.62	33.90	5.58	1.35	1.09	28.66	8.72	0.51	15.78
41	OSAH+TA$_{SNL}$(18,2)	15641	15642	2053	25757	9.83	22.40	5.70	1.37	1.37	29.59	8.72	0.49	16.57
42	OSAH+TA$_{NLT}$(18,2)	60837	15642	2053	25757	9.83	21.18	5.70	1.37	1.39	29.73	8.72	0.48	16.69
43	OSAH+TA$_{seq}$(atc)	42904	42905	5197	49876	5.09	113.84	6.94	3.12	1.58	39.53	8.72	0.17	25.07
44	OSAH+TA$^{A}_{rec}$(atc)	42904	42905	5197	49876	4.94	42.78	6.85	3.12	1.55	30.99	8.72	0.24	17.77
45	OSAH+TA$^{B}_{rec}$(atc)	42904	42905	5197	49876	4.94	42.78	6.85	3.12	1.58	27.03	8.72	0.30	14.38
46	OSAH+TA$_{SNL}$(atc)	42904	42905	5197	49876	5.09	27.37	6.94	3.12	2.35	26.86	8.72	0.30	14.24
47	OSAH+TA$_{NLT}$(atc)	178081	42905	5197	49876	5.09	25.26	6.94	3.12	2.42	26.69	8.72	0.31	14.09
48	BVH	810	5374	0	8281	123.84	155.40	119.25	0.00	3.02	1908.83	–	–	–
49	O84	8777	61440	7744	88184	11.58	51.96	8.44	1.71	0.53	59.89	–	–	–
50	O89	8777	61440	7744	88184	11.52	33.60	8.41	1.71	0.48	48.88	–	–	–
51	BSP	22015	22016	464	49328	15.01	38.15	6.44	0.20	1.82	68.72	–	–	–
52	O93	8777	61440	7744	88184	10.40	31.98	13.17	7.03	0.49	61.09	–	–	–
53	UG	0	42875	9616	61003	13.55	8.46	8.46	1.37	0.63	39.95	–	–	–
54	AG	3614	4614	0	14548	66.28	82.96	62.19	0.00	2.16	1005.04	–	–	–
55	HUG	1	12167	0	61535	35.88	6.83	5.83	0.00	0.32	85.77	–	–	–
56	RG	289	11568	0	61156	36.97	6.09	5.00	0.00	0.17	75.90	–	–	–
57	O84A	7441	52088	6928	66424	9.24	48.75	8.01	2.58	1.48	54.82	–	–	–
58	KD	15225	15226	2044	25350	9.79	34.14	5.71	1.33	3.97	52.88	–	–	–

Table 14: Experimental results for scene “*lattice12*”.

$$N = 8281,$$
$$TP_D:\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 263169,\quad N^{hit}_{prim} = 261170,\quad N_{sec} = 243215,\quad N^{hit}_{sec} = 178786,$$
$$N_{shad} = 1180750,\quad N^{hit}_{shad} = 943153,\quad T^{MIN}_{R}[s] = 11.37,\quad T_{app}[s] = 10.20,\quad T^{MIN}_{RSA}[s] = 1.17.$$

Scene = “*mount6*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	8196	–	0	0	0	0.01	6971.35	16.03	1.0	20503.97
1	spatmed-xyz(16,2)	8837	8838	1989	43940	21.34	27.24	5.13	1.41	0.33	18.72	16.03	0.38	39.03
2	objmed-xyz(16,2)	12089	12090	1398	23189	34.04	53.05	9.51	1.37	0.80	25.87	16.03	0.34	60.06
3	objmed(16,2)	4414	4415	17	8754	63.15	98.72	17.55	0.05	0.93	46.40	16.03	0.34	120.44
4	OSAH(16,2)	6701	6702	1302	10382	7.49	22.07	3.90	1.45	1.07	13.73	16.03	0.21	24.35
5	OSAH-RMI(16,2)	6701	6702	1302	10382	7.49	22.07	3.90	1.45	1.07	13.27	16.03	0.21	23.00
6	OSAH-xyz(16,2)	8003	8004	1848	13451	8.67	24.30	4.20	1.48	0.71	13.66	16.03	0.22	24.15
7	OSAH(8,1)	194	195	43	9016	94.45	13.11	2.48	0.80	0.74	35.12	16.03	0.85	87.26
8	OSAH(8,2)	194	195	43	9016	94.45	13.11	2.48	0.80	0.77	34.96	16.03	0.85	86.79
9	OSAH(16,1)	10102	10103	3866	11210	6.81	22.90	3.97	1.59	1.14	13.50	16.03	0.19	23.68
10	OSAH(16,2)	6701	6702	1302	10382	7.49	22.07	3.90	1.45	1.07	13.73	16.03	0.21	24.35
11	OSAH(24,1)	16495	16496	6633	14567	5.67	24.82	4.22	1.73	1.27	13.57	16.03	0.15	23.88
12	OSAH(24,2)	7005	7006	1311	10409	6.90	22.81	3.97	1.45	1.05	13.64	16.03	0.19	24.09
13	OSAH(atc)	13191	13192	5649	12237	6.57	20.73	3.71	1.69	1.17	12.74	16.03	0.20	21.44
14	OSAH2(atc)	16578	16579	5795	16317	6.75	24.75	4.27	1.66	1.69	13.82	16.03	0.18	24.62
15	OSAH+LC(atc)	13203	13204	5651	12258	5.99	22.19	3.87	1.69	1.19	13.05	16.03	0.18	22.35
16	OSAH+TPC(atc)	12996	12997	5562	12137	7.48	19.42	3.52	1.69	1.19	12.66	16.03	0.23	21.21
17	OSAH+TPC+LC(atc)	13028	13029	5564	12176	7.17	21.70	3.88	1.69	1.38	12.26	16.03	0.21	20.03
18	OSAH+LC(16,1)	10104	10105	3868	11210	6.81	22.90	3.97	1.59	1.28	12.52	16.03	0.19	20.79
19	OSAH+TPC(16,1)	10094	10095	3838	11251	6.81	22.89	3.97	1.59	1.29	12.37	16.03	0.19	20.35
20	OSAH+TPC+LC(16,1)	10096	10097	3840	11251	6.81	22.89	3.97	1.59	1.29	12.41	16.03	0.19	20.47
21	OSAH+PR(atc)	13191	13192	5649	11919	6.55	20.73	3.72	1.70	1.25	12.74	16.03	0.20	21.44
22	OSAH+SC(atc)	14301	14302	6193	12837	6.15	20.68	3.72	1.73	1.76	12.71	16.03	0.19	21.35
23	OSAH+GCM(atc)	15020	15021	6460	13237	6.53	21.20	3.79	1.72	45.38	12.51	16.03	0.20	20.76
24	OSAH+GCM2(atc)	20943	20944	6166	22986	7.25	27.75	4.73	1.40	66.00	14.07	16.03	0.17	25.35
25	OSAH+GCM3(atc)	15240	15241	5296	16054	7.38	28.56	4.63	1.86	47.69	14.05	16.03	0.17	25.29
26	OSAH+PAR(atc)	13191	13192	5649	12237	4.71	28.48	5.51	4.08	1.49	2.28	13.83	0.13	24.17
27	PARSAH+PAR(atc)	13186	13187	5718	12127	4.65	28.55	5.50	4.10	1.41	2.27	13.83	0.12	24.00
28	OSAH+PER(atc)	13191	13192	5649	12237	4.51	30.94	6.09	3.96	1.55	2.77	12.67	0.16	18.11
29	PERSAH+PER(atc)	13281	13282	5612	12503	6.19	30.09	6.07	4.00	61.17	2.84	12.67	0.21	18.89
30	SPHSAH+PER(atc)	13191	13192	5649	12237	4.51	30.94	6.09	3.96	2.06	2.83	12.67	0.19	18.78
31	OSAH+SPH(atc)	13191	13192	5649	12237	4.50	30.36	5.97	3.89	1.56	3.06	13.18	0.16	14.64
32	SPHSAH+SPH(atc)	13191	13192	5649	12237	4.50	30.36	5.97	3.89	1.95	3.06	13.18	0.16	14.64
33	OSAH+TA$_{seq}$(16,2)	6701	6702	1302	10382	8.01	44.39	3.90	1.45	0.84	16.17	16.03	0.14	31.53
34	OSAH+TA$^{A}_{rec}$(16,2)	6701	6702	1302	10382	7.49	22.07	3.90	1.45	0.83	13.73	16.03	0.18	24.35
35	OSAH+TA$^{B}_{rec}$(16,2)	6701	6702	1302	10382	7.49	22.07	3.90	1.45	0.83	12.49	16.03	0.21	20.71
36	OSAH+TA$_{SNL}$(16,2)	6701	6702	1302	10382	8.01	15.53	3.90	1.45	0.94	11.44	16.03	0.25	17.62
37	OSAH+TA$_{NLT}$(16,2)	32628	6702	1302	10382	8.01	13.92	3.90	1.45	0.98	11.81	16.03	0.23	18.71
38	OSAH+TA$_{seq}$(18,2)	7000	7001	1311	10397	7.47	45.91	3.95	1.45	0.85	16.20	16.03	0.13	31.62
39	OSAH+TA$^{A}_{rec}$(18,2)	7000	7001	1311	10397	6.98	22.70	3.95	1.45	0.84	13.77	16.03	0.17	24.47
40	OSAH+TA$^{B}_{rec}$(18,2)	7000	7001	1311	10397	6.98	22.70	3.95	1.45	0.85	12.44	16.03	0.20	20.56
41	OSAH+TA$_{SNL}$(18,2)	7000	7001	1311	10397	7.47	15.90	3.96	1.45	0.95	11.35	16.03	0.24	17.35
42	OSAH+TA$_{NLT}$(18,2)	33397	7001	1311	10397	7.47	14.21	3.96	1.45	0.99	11.35	16.03	0.24	17.35
43	OSAH+TA$_{seq}$(atc)	13191	13192	5649	12237	7.02	41.74	3.72	1.69	0.96	15.48	16.03	0.13	29.50
44	OSAH+TA$^{A}_{rec}$(atc)	13191	13192	5649	12237	6.57	20.73	3.72	1.69	0.93	12.96	16.03	0.17	22.09
45	OSAH+TA$^{B}_{rec}$(atc)	13191	13192	5649	12237	6.57	20.73	3.71	1.69	0.93	11.89	16.03	0.20	18.94
46	OSAH+TA$_{SNL}$(atc)	13191	13192	5649	12237	7.02	14.68	3.72	1.69	1.16	10.95	16.03	0.23	16.18
47	OSAH+TA$_{NLT}$(atc)	58078	13192	5649	12237	7.02	13.23	3.72	1.69	1.24	10.94	16.03	0.23	16.15
48	BVH	1058	5398	0	8196	170.39	175.62	124.24	0.00	0.97	1583.37	–	–	–
49	O84	4358	30507	10649	85916	17.68	34.39	6.37	2.27	0.46	32.36	–	–	–
50	O89	4358	30507	10649	85916	17.75	24.43	6.37	2.27	0.44	28.15	–	–	–
51	BSP	8837	8838	1989	43940	21.34	27.24	5.13	1.41	0.50	39.38	–	–	–
52	O93	4358	30507	10649	85916	16.91	23.46	9.65	5.71	0.42	33.95	–	–	–
53	UG	0	41650	36218	37369	24.23	14.90	14.90	9.53	0.40	25.97	–	–	–
54	AG	3342	91277	34893	217830	29.47	21.48	18.96	12.50	3.76	46.60	–	–	–
55	HUG	1	512	310	14641	131.92	4.92	3.93	1.31	0.15	72.35	–	–	–
56	RG	10035	136153	16704	713095	26.98	12.47	10.30	5.22	1.67	31.80	–	–	–
57	O84A	7352	51465	11955	145308	13.32	33.32	6.22	2.69	2.19	29.95	–	–	–
58	KD	6701	6702	1302	10382	7.49	22.07	3.90	1.45	1.48	21.06	–	–	–

Table 15: Experimental results for scene “*mount6*”.

$$N = 8196,$$
$$TP_D\colon\ \ N_{prim} = 263169,\ \ N^{hit}_{\mathcal{AB}} = 257871,\ \ N^{hit}_{prim} = 173685,\quad N_{sec} = 707764,\ \ N^{hit}_{sec} = 472358,$$
$$N_{shad} = 361043,\ \ N^{hit}_{shad} = 30032,\quad T^{MIN}_{R}[s] = 5.79,\ \ T_{app}[s] = 5.45,\ \ T^{MIN}_{RSA}[s] = 0.34.$$

Scene = “*rings7*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	8401	–	0	0	0	0.02	15987.20	5.33	1.0	10517.89
1	spatmed-xyz(16,2)	12403	12404	1196	63348	42.38	51.03	9.56	3.87	0.42	71.81	5.33	0.72	41.91
2	objmed-xyz(16,2)	46174	46175	4973	121417	87.28	106.86	20.87	4.66	1.84	141.02	5.33	0.72	87.45
3	objmed(16,2)	48671	48672	4085	120391	83.20	109.24	23.20	6.46	2.69	137.09	5.33	0.70	84.86
4	OSAH(16,2)	11835	11836	1600	32785	19.51	36.59	6.47	2.63	1.58	42.62	5.33	0.62	22.71
5	OSAH-RMI(16,2)	11835	11836	1600	32785	19.51	36.59	6.47	2.63	1.58	42.19	5.33	0.62	22.43
6	OSAH-xyz(16,2)	12417	12418	2063	34704	21.13	39.47	7.21	3.07	0.99	44.81	5.33	0.63	24.15
7	OSAH(8,1)	182	183	32	9506	221.71	16.75	3.54	1.24	0.83	196.30	5.33	0.98	123.82
8	OSAH(8,2)	182	183	32	9506	221.71	16.75	3.54	1.24	0.82	196.74	5.33	0.98	124.11
9	OSAH(16,1)	13332	13333	2323	33401	19.09	37.70	6.74	2.92	1.60	41.94	5.33	0.61	22.26
10	OSAH(16,2)	11835	11836	1600	32785	19.51	36.59	6.47	2.63	1.58	42.62	5.33	0.62	22.71
11	OSAH(24,1)	195354	195355	16920	303710	16.98	55.97	9.96	3.50	6.36	47.68	5.33	0.49	26.04
12	OSAH(24,2)	111055	111056	4959	231712	17.12	46.91	8.11	2.79	4.58	44.91	5.33	0.53	24.22
13	OSAH(atc)	14913	14914	2525	35825	19.23	37.39	6.67	2.90	1.72	42.05	5.33	0.62	22.34
14	OSAH2(atc)	34413	34414	3882	66152	17.07	49.77	8.70	3.50	2.96	45.25	5.33	0.52	24.44
15	OSAH+LC(atc)	29750	29751	2527	82883	19.34	41.83	7.34	2.90	2.34	44.56	5.33	0.59	23.99
16	OSAH+TPC(atc)	9466	9467	1591	28073	22.04	35.08	6.29	2.75	1.63	44.28	5.33	0.66	23.80
17	OSAH+TPC+LC(atc)	31616	31617	1596	97448	22.68	42.03	7.39	2.75	2.91	46.84	5.33	0.63	25.49
18	OSAH+LC(16,1)	13348	13349	2328	33432	19.13	37.71	6.74	2.92	1.84	40.78	5.33	0.61	21.50
19	OSAH+TPC(16,1)	13349	13350	2345	33412	19.12	37.71	6.74	2.92	1.90	41.12	5.33	0.61	21.72
20	OSAH+TPC+LC(16,1)	13351	13352	2347	33412	19.12	37.71	6.74	2.92	1.90	40.81	5.33	0.61	21.52
21	OSAH+PR(atc)	14913	14914	2525	27424	16.79	37.53	6.71	2.90	2.82	39.08	5.33	0.58	20.38
22	OSAH+SC(atc)	16196	16197	5846	24791	12.41	38.82	6.97	4.00	3.13	34.36	5.33	0.50	17.28
23	OSAH+GCM(atc)	28565	28566	5726	55492	15.86	43.31	7.73	3.27	87.98	40.69	5.33	0.53	21.44
24	OSAH+GCM2(atc)	71878	71879	2372	183905	21.48	53.92	9.35	2.78	233.09	52.27	5.33	0.55	29.06
25	OSAH+GCM3(atc)	29923	29924	3578	67553	125.24	47.34	7.19	2.80	98.25	144.65	5.33	0.89	89.84
26	OSAH+PAR(atc)	14913	14914	2525	35825	7.82	31.48	5.80	2.95	1.96	6.74	5.17	0.65	7.55
27	PARSAH+PAR(atc)	24119	24120	1680	210932	7.61	13.07	1.12	0.09	5.15	6.49	5.17	0.85	7.08
28	OSAH+PER(atc)	14913	14914	2525	35825	10.66	41.58	8.18	4.39	1.94	8.27	5.05	0.65	9.21
29	PERSAH+PER(atc)	4920	4921	1029	21254	85.89	26.48	5.71	2.84	202.02	24.16	5.05	0.94	36.60
30	SPHSAH+PER(atc)	20335	20336	4429	74874	21.43	55.89	11.17	6.50	13.41	11.94	5.05	0.72	15.53
31	OSAH+SPH(atc)	14913	14914	2525	35825	10.68	41.13	8.11	4.37	1.97	8.44	5.02	0.66	8.60
32	SPHSAH+SPH(atc)	20335	20336	4429	74874	21.51	55.59	11.14	6.55	13.43	12.06	5.02	0.72	14.44
33	OSAH+TA$_{seq}$(16,2)	11842	11843	1598	32814	20.68	92.74	6.64	2.63	1.34	52.83	5.33	0.46	29.43
34	OSAH+TA$^{A}_{rec}$(16,2)	11842	11843	1598	32814	19.56	36.59	6.47	2.63	1.30	44.21	5.33	0.57	23.76
35	OSAH+TA$^{B}_{rec}$(16,2)	11842	11843	1598	32814	19.56	36.59	6.47	2.63	1.31	40.55	5.33	0.63	21.35
36	OSAH+TA$_{SNL}$(16,2)	11842	11843	1598	32814	20.68	30.28	6.64	2.63	1.53	45.50	5.33	0.55	24.61
37	OSAH+TA$_{NLT}$(16,2)	73110	11843	1598	32814	20.68	27.12	6.64	2.63	1.56	45.62	5.33	0.54	24.68
38	OSAH+TA$_{seq}$(18,2)	24269	24270	2827	55360	17.39	107.41	7.24	2.71	1.70	53.01	5.33	0.39	29.55
39	OSAH+TA$^{A}_{rec}$(18,2)	24269	24270	2827	55360	16.35	40.26	7.02	2.71	1.67	43.30	5.33	0.49	23.16
40	OSAH+TA$^{B}_{rec}$(18,2)	24269	24270	2827	55360	16.35	40.26	7.02	2.71	1.69	39.15	5.33	0.56	20.43
41	OSAH+TA$_{SNL}$(18,2)	24269	24270	2827	55360	17.39	33.50	7.24	2.71	2.11	43.89	5.33	0.49	23.55
42	OSAH+TA$_{NLT}$(18,2)	150093	24270	2827	55360	17.39	29.81	7.24	2.71	2.21	43.85	5.33	0.49	23.52
43	OSAH+TA$_{seq}$(atc)	14951	14952	2527	35894	20.43	97.14	6.84	2.90	1.46	53.21	5.33	0.44	29.68
44	OSAH+TA$^{A}_{rec}$(atc)	14951	14952	2527	35894	19.32	37.40	6.67	2.90	1.45	44.16	5.33	0.55	23.72
45	OSAH+TA$^{B}_{rec}$(atc)	14951	14952	2527	35894	19.32	37.40	6.67	2.90	1.45	40.29	5.33	0.62	21.18
46	OSAH+TA$_{SNL}$(atc)	14951	14952	2527	35894	20.43	30.91	6.85	2.90	1.68	45.42	5.33	0.53	24.55
47	OSAH+TA$_{NLT}$(atc)	92519	14952	2527	35894	20.43	27.58	6.85	2.90	1.78	45.29	5.33	0.54	24.47
48	BVH	960	5385	0	8401	185.15	179.96	128.64	0.00	1.45	2222.09	–	–	–
49	O84	6476	45333	8289	130108	35.22	79.26	13.05	6.28	0.65	114.89	–	–	–
50	O89	6476	45333	8289	130108	35.20	51.13	13.05	6.28	0.52	98.64	–	–	–
51	BSP	12403	12404	1196	63348	42.38	51.03	9.56	3.87	1.32	144.15	–	–	–
52	O93	6476	45333	8289	130108	34.97	47.10	19.18	12.44	0.55	114.91	–	–	–
53	UG	0	41650	32408	50233	45.40	19.54	19.54	14.45	0.50	91.48	–	–	–
54	AG	3998	73611	11870	182079	47.32	14.76	10.60	0.19	2.65	154.98	–	–	–
55	HUG	1749	13765	2305	61621	65.25	13.71	9.03	4.90	0.50	132.47	–	–	–
56	RG	2083	35411	8840	111974	48.54	17.93	14.43	7.75	0.35	111.94	–	–	–
57	O84A	11302	79115	7387	193220	26.00	65.41	11.03	4.49	2.25	94.90	–	–	–
58	KD	11835	11836	1600	32785	19.51	36.59	6.47	2.63	2.38	64.76	–	–	–

Table 16: Experimental results for scene “*rings7*”.

$$N = 8401,$$
$$TP_D: \quad N_{prim} = 263169, \quad N^{hit}_{\mathcal{AB}} = 263169, \quad N^{hit}_{prim} = 263168, \quad N_{sec} = 312998, \quad N^{hit}_{sec} = 175756,$$
$$N_{shad} = 1077448, \quad N^{hit}_{shad} = 510854, \quad T^{MIN}_{R}[s] = 9.62, \quad T_{app}[s] = 8.10, \quad T^{MIN}_{RSA}[s] = 1.52.$$

Scene = “*sombrero2*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	7938	–	0	0	0	0.02	2229.08	12.22	1.0	24767.56
1	spatmed-xyz(16,2)	9405	9406	1892	48754	46.92	27.46	5.08	2.77	0.35	4.62	12.22	0.54	39.11
2	objmed-xyz(16,2)	16777	16778	1695	40753	96.63	51.02	9.38	0.80	1.09	7.59	12.22	0.57	72.11
3	objmed(16,2)	4025	4026	0	8049	50.09	38.64	7.69	0.00	0.83	5.30	12.22	0.47	46.67
4	OSAH(16,2)	4641	4642	521	8142	7.78	16.10	2.85	1.59	1.01	2.73	12.22	0.25	18.11
5	OSAH-RMI(16,2)	4641	4642	521	8142	7.78	16.10	2.85	1.59	1.12	2.57	12.22	0.25	16.33
6	OSAH-xyz(16,2)	4320	4321	885	10208	13.68	19.39	3.61	2.14	0.72	2.91	12.22	0.33	20.11
7	OSAH(8,1)	175	176	68	8016	240.30	9.57	2.09	1.05	0.75	10.81	12.22	0.95	107.89
8	OSAH(8,2)	174	175	67	8016	240.30	9.57	2.09	1.05	0.74	10.72	12.22	0.95	106.89
9	OSAH(16,1)	11616	11617	7124	8482	5.82	18.30	3.35	2.35	1.10	2.72	12.22	0.18	18.00
10	OSAH(16,2)	4641	4642	521	8142	7.78	16.10	2.85	1.59	1.01	2.73	12.22	0.25	18.11
11	OSAH(24,1)	15712	15713	7688	12008	6.36	19.29	3.64	2.39	1.20	2.76	12.22	0.19	18.44
12	OSAH(24,2)	4641	4642	521	8142	7.78	16.10	2.85	1.59	1.11	2.65	12.22	0.25	17.22
13	OSAH(atc)	13927	13928	7677	10234	6.05	18.88	3.51	2.39	1.12	2.72	12.22	0.18	18.00
14	OSAH2(atc)	14000	14001	7167	11093	8.07	22.81	3.94	2.24	1.47	2.81	12.22	0.20	19.00
15	OSAH+LC(atc)	13927	13928	7677	10234	6.05	18.88	3.51	2.39	1.15	2.69	12.22	0.18	17.67
16	OSAH+TPC(atc)	13892	13893	7666	10210	6.06	18.85	3.50	2.39	1.13	2.72	12.22	0.18	18.00
17	OSAH+TPC+LC(atc)	13896	13897	7666	10214	6.06	18.85	3.51	2.39	1.47	2.44	12.22	0.18	14.89
18	OSAH+LC(16,1)	11616	11617	7124	8482	5.82	18.30	3.35	2.35	1.38	2.40	12.22	0.18	14.44
19	OSAH+TPC(16,1)	11618	11619	7126	8482	5.83	18.29	3.35	2.35	1.28	2.40	12.22	0.18	14.44
20	OSAH+TPC+LC(16,1)	11618	11619	7126	8482	5.83	18.29	3.35	2.35	1.25	2.39	12.22	0.18	14.33
21	OSAH+PR(atc)	13927	13928	7677	10170	6.03	18.88	3.51	2.39	1.17	2.75	12.22	0.18	18.33
22	OSAH+SC(atc)	14208	14209	7772	10424	5.89	18.91	3.51	2.42	1.37	2.50	12.22	0.18	15.56
23	OSAH+GCM(atc)	14312	14313	7379	10916	6.22	21.58	4.04	2.86	42.23	2.60	12.22	0.17	16.67
24	OSAH+GCM2(atc)	15561	15562	5359	15924	8.15	22.60	3.92	2.28	55.24	2.71	12.22	0.20	17.89
25	OSAH+GCM3(atc)	12464	12465	5199	12888	11.31	24.33	4.47	2.95	41.00	2.90	12.22	0.24	20.00
26	OSAH+PAR(atc)	13927	13928	7677	10234	2.89	14.08	2.60	1.83	1.38	1.80	9.62	0.46	12.88
27	PARSAH+PAR(atc)	13729	13730	7287	10436	2.92	11.88	2.23	1.48	1.27	1.61	9.62	0.44	10.50
28	OSAH+PER(atc)	13927	13928	7677	10234	2.98	13.42	2.48	1.71	1.36	1.68	9.64	0.14	5.64
29	PERSAH+PER(atc)	10072	10073	5396	9696	2.94	12.53	2.38	1.63	40.18	1.69	9.64	0.21	5.73
30	SPHSAH+PER(atc)	6092	6093	3236	8683	56.40	11.65	2.27	1.62	2.31	4.05	9.64	0.85	27.18
31	OSAH+SPH(atc)	13927	13928	7677	10234	2.97	13.46	2.49	1.72	1.46	2.07	10.62	0.23	5.31
32	SPHSAH+SPH(atc)	6092	6093	3236	8683	55.54	11.72	2.29	1.63	2.30	4.32	10.62	0.84	22.62
33	OSAH+TA$_{seq}$(16,2)	4641	4642	521	8142	7.79	35.79	2.86	1.59	0.77	3.23	12.22	0.15	23.67
34	OSAH+TA$^{A}_{rec}$(16,2)	4641	4642	521	8142	7.78	16.10	2.85	1.59	0.75	2.62	12.22	0.21	16.89
35	OSAH+TA$^{B}_{rec}$(16,2)	4641	4642	521	8142	7.78	16.10	2.85	1.59	0.74	2.36	12.22	0.25	14.00
36	OSAH+TA$_{SNL}$(16,2)	4641	4642	521	8142	7.79	14.87	2.86	1.59	0.84	2.52	12.22	0.22	15.78
37	OSAH+TA$_{NLT}$(16,2)	18971	4642	521	8142	7.79	14.06	2.86	1.59	1.03	2.35	12.22	0.25	13.89
38	OSAH+TA$_{seq}$(18,2)	4641	4642	521	8142	7.79	35.79	2.86	1.59	0.77	3.19	12.22	0.16	23.22
39	OSAH+TA$^{A}_{rec}$(18,2)	4641	4642	521	8142	7.78	16.10	2.85	1.59	0.77	2.63	12.22	0.21	17.00
40	OSAH+TA$^{B}_{rec}$(18,2)	4641	4642	521	8142	7.78	16.10	2.85	1.59	0.77	2.41	12.22	0.25	14.56
41	OSAH+TA$_{SNL}$(18,2)	4641	4642	521	8142	7.79	14.87	2.86	1.59	0.82	2.43	12.22	0.25	14.78
42	OSAH+TA$_{NLT}$(18,2)	18971	4642	521	8142	7.79	14.06	2.86	1.59	0.82	2.52	12.22	0.23	15.78
43	OSAH+TA$_{seq}$(atc)	13927	13928	7677	10234	6.05	49.54	3.51	2.39	0.89	3.64	12.22	0.09	28.22
44	OSAH+TA$^{A}_{rec}$(atc)	13927	13928	7677	10234	6.05	18.88	3.51	2.39	0.89	2.75	12.22	0.15	18.33
45	OSAH+TA$^{B}_{rec}$(atc)	13927	13928	7677	10234	6.05	18.88	3.51	2.39	0.87	2.43	12.22	0.18	14.78
46	OSAH+TA$_{SNL}$(atc)	13927	13928	7677	10234	6.05	17.01	3.51	2.39	1.12	2.55	12.22	0.17	16.11
47	OSAH+TA$_{NLT}$(atc)	55873	13928	7677	10234	6.05	15.81	3.51	2.39	1.19	2.40	12.22	0.18	14.44
48	BVH	1094	5255	0	7938	181.14	135.81	92.42	0.00	1.38	325.22	–	–	–
49	O84	4741	33188	11479	96034	41.16	42.70	7.45	4.54	0.50	9.16	–	–	–
50	O89	4741	33188	11479	96034	40.96	28.70	7.43	4.54	0.43	7.29	–	–	–
51	BSP	9405	9406	1892	48754	46.92	27.46	5.08	2.77	0.50	7.40	–	–	–
52	O93	4741	33188	11479	96034	39.13	25.40	9.84	7.04	0.44	8.80	–	–	–
53	UG	0	39672	33848	38651	35.69	11.34	11.34	9.71	0.37	4.97	–	–	–
54	AG	1534	49014	18116	108836	35.28	9.50	8.01	5.26	2.38	7.80	–	–	–
55	HUG	1	675	343	14548	242.76	3.67	3.01	1.53	0.15	19.37	–	–	–
56	RG	1484	29607	12558	87792	37.16	9.71	7.86	5.65	0.28	6.43	–	–	–
57	O84A	7034	49239	10432	150250	39.81	42.40	7.32	3.92	2.07	8.91	–	–	–
58	KD	4641	4642	521	8142	7.78	16.10	2.85	1.59	1.38	4.00	–	–	–

Table 17: Experimental results for scene “*sombrero2*”.

$$N = 7938,$$
$$TP_D\text{:}\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 136638,\quad N^{hit}_{prim} = 112239,\quad N_{sec} = 0,\quad N^{hit}_{sec} = 0,$$
$$N_{shad} = 110608,\quad N^{hit}_{shad} = 2523,\quad T^{MIN}_{R}[s] = 1.19,\quad T_{app}[s] = 1.10,\quad T^{MIN}_{RSA}[s] = 0.09.$$

Scene = "*teapot12*"

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	9264	–	0	0	0	0.02	6036.10	17.32	1.0	27436.82
1	spatmed-xyz(16,2)	6866	6867	1800	41612	61.95	36.01	7.57	4.52	0.34	15.11	17.32	0.52	51.36
2	objmed-xyz(16,2)	26569	26570	3712	61429	161.76	105.53	20.58	3.38	1.24	34.04	17.32	0.49	137.41
3	objmed(16,2)	22186	22187	2390	47489	219.60	181.84	35.38	3.30	1.58	52.53	17.32	0.43	221.45
4	OSAH(16,2)	10198	10199	1942	27347	17.71	26.28	5.12	2.79	1.46	10.32	17.32	0.30	29.59
5	OSAH-RMI(16,2)	10198	10199	1942	27347	17.71	26.28	5.12	2.79	1.48	9.99	17.32	0.30	28.09
6	OSAH-xyz(16,2)	10248	10249	1983	28294	20.73	30.76	6.25	3.78	0.93	10.58	17.32	0.30	30.77
7	OSAH(8,1)	213	214	26	11345	305.93	15.76	3.47	1.62	0.90	36.32	17.32	0.92	147.77
8	OSAH(8,2)	213	214	26	11345	305.93	15.76	3.47	1.62	0.92	36.55	17.32	0.92	148.82
9	OSAH(16,1)	11852	11853	3030	27771	14.50	27.56	5.26	3.16	1.51	10.03	17.32	0.25	28.27
10	OSAH(16,2)	10198	10199	1942	27347	17.71	26.28	5.12	2.79	1.46	10.32	17.32	0.30	29.59
11	OSAH(24,1)	51751	51752	11450	79971	10.98	30.16	5.71	3.40	2.66	10.06	17.32	0.19	28.41
12	OSAH(24,2)	34811	34812	4295	70563	14.82	27.67	5.35	2.85	2.23	10.13	17.32	0.25	28.73
13	OSAH(atc)	23502	23503	6337	38220	12.01	28.97	5.50	3.33	1.80	9.72	17.32	0.21	26.86
14	OSAH2(atc)	35521	35522	8102	56119	13.16	36.64	6.53	3.80	2.91	10.90	17.32	0.18	32.23
15	OSAH+LC(atc)	24880	24881	6364	43086	11.80	29.15	5.53	3.33	1.93	9.69	17.32	0.20	26.73
16	OSAH+TPC(atc)	18888	18889	5132	34319	13.53	27.99	5.33	3.21	1.87	9.63	17.32	0.23	26.45
17	OSAH+TPC+LC(atc)	22949	22950	5255	45003	12.71	28.55	5.41	3.22	2.30	9.29	17.32	0.22	24.91
18	OSAH+LC(16,1)	11858	11859	3036	27771	14.48	27.57	5.26	3.16	1.77	9.29	17.32	0.25	24.91
19	OSAH+TPC(16,1)	11875	11876	3051	27892	14.51	27.56	5.26	3.15	1.78	9.30	17.32	0.25	24.95
20	OSAH+TPC+LC(16,1)	11881	11882	3057	27892	14.50	27.56	5.26	3.16	1.81	9.32	17.32	0.25	25.05
21	OSAH+PR(atc)	23502	23503	6337	33869	12.68	28.02	5.29	3.26	2.21	7.28	17.32	0.19	15.77
22	OSAH+SC(atc)	23822	23823	9466	32983	9.34	28.32	5.36	3.68	3.65	9.64	17.32	0.17	26.50
23	OSAH+GCM(atc)	30936	30937	8478	46548	11.16	30.78	5.73	3.52	100.93	9.77	17.32	0.18	27.09
24	OSAH+GCM2(atc)	55335	55336	9907	108251	13.67	33.81	6.07	3.54	183.16	10.36	17.32	0.20	29.77
25	OSAH+GCM3(atc)	28719	28720	6446	52848	13.67	38.24	6.76	4.20	95.24	10.88	17.32	0.18	32.14
26	OSAH+PAR(atc)	23502	23503	6337	38220	3.74	19.32	3.75	2.63	2.13	2.62	16.33	0.36	12.78
27	PARSAH+PAR(atc)	23233	23234	6227	38196	3.74	19.27	3.69	2.62	2.37	2.70	16.33	0.40	13.67
28	OSAH+PER(atc)	23502	23503	6337	38220	3.42	23.39	4.59	3.25	2.18	3.28	13.64	0.35	9.79
29	PERSAH+PER(atc)	21147	21148	5836	35610	3.35	21.58	4.36	3.09	129.81	3.23	13.64	0.38	9.43
30	SPHSAH+PER(atc)	18627	18628	5154	34050	3.86	21.85	4.40	3.09	5.82	3.28	13.64	0.39	9.79
31	OSAH+SPH(atc)	23502	23503	6337	38220	3.42	22.92	4.49	3.17	2.12	3.59	16.14	0.34	9.50
32	SPHSAH+SPH(atc)	18627	18628	5154	34050	3.87	21.42	4.31	3.02	5.90	3.57	16.14	0.38	9.36
33	OSAH+TA$_{seq}$(16,2)	10198	10199	1942	27347	17.83	60.99	5.13	2.79	1.23	13.44	17.32	0.17	43.77
34	OSAH+TA$^{A}_{rec}$(16,2)	10198	10199	1942	27347	17.71	26.28	5.12	2.79	1.22	10.55	17.32	0.25	30.64
35	OSAH+TA$^{B}_{rec}$(16,2)	10198	10199	1942	27347	17.71	26.28	5.12	2.79	1.19	9.42	17.32	0.30	25.50
36	OSAH+TA$_{SNL}$(16,2)	10198	10199	1942	27347	17.84	21.13	5.13	2.79	1.38	9.39	17.32	0.30	25.36
37	OSAH+TA$_{NLT}$(16,2)	59876	10199	1942	27347	17.84	18.96	5.13	2.79	1.42	9.48	17.32	0.30	25.77
38	OSAH+TA$_{seq}$(18,2)	18672	18673	3159	37728	15.36	64.58	5.28	2.84	1.45	13.47	17.32	0.15	43.91
39	OSAH+TA$^{A}_{rec}$(18,2)	18672	18673	3159	37728	15.26	27.18	5.27	2.84	1.41	10.50	17.32	0.21	30.41
40	OSAH+TA$^{B}_{rec}$(18,2)	18672	18673	3159	37728	15.26	27.18	5.27	2.84	1.41	9.30	17.32	0.26	24.95
41	OSAH+TA$_{SNL}$(18,2)	18672	18673	3159	37728	15.36	21.82	5.28	2.84	1.77	9.31	17.32	0.26	25.00
42	OSAH+TA$_{NLT}$(18,2)	100779	18673	3159	37728	15.36	19.50	5.28	2.84	1.84	9.26	17.32	0.26	24.77
43	OSAH+TA$_{seq}$(atc)	23502	23503	6337	38220	12.13	70.11	5.52	3.33	1.52	13.61	17.32	0.11	44.55
44	OSAH+TA$^{A}_{rec}$(atc)	23502	23503	6337	38220	12.01	28.97	5.50	3.33	1.50	10.45	17.32	0.17	30.18
45	OSAH+TA$^{B}_{rec}$(atc)	23502	23503	6337	38220	12.01	28.97	5.50	3.33	1.50	9.16	17.32	0.21	24.32
46	OSAH+TA$_{SNL}$(atc)	23502	23503	6337	38220	12.13	22.81	5.52	3.33	1.89	9.15	17.32	0.21	24.27
47	OSAH+TA$_{NLT}$(atc)	117412	23503	6337	38220	12.13	20.27	5.52	3.33	2.03	9.12	17.32	0.21	24.14
48	BVH	1393	6001	0	9264	290.51	183.11	119.20	0.00	1.66	1121.33	–	–	–
49	O84	3372	23605	9421	72493	44.06	55.80	10.41	7.26	0.48	29.25	–	–	–
50	O89	3372	23605	9421	72493	44.08	38.91	10.40	7.26	0.43	22.91	–	–	–
51	BSP	6866	6867	1800	41612	61.95	36.01	7.57	4.52	1.14	58.51	–	–	–
52	O93	3372	23605	9421	72493	42.76	33.88	13.13	10.05	0.46	27.54	–	–	–
53	UG	0	48020	42398	37819	60.94	26.00	26.00	23.03	0.45	20.93	–	–	–
54	AG	2983	103883	38995	248398	53.59	19.88	16.85	11.83	4.60	32.59	–	–	–
55	HUG	2305	30596	21099	51033	51.47	20.44	17.54	14.83	0.63	26.97	–	–	–
56	RG	5541	93034	23297	461085	53.29	19.27	16.84	12.95	1.08	23.24	–	–	–
57	O84A	7066	49463	12222	142944	29.21	55.29	10.50	7.56	2.19	27.43	–	–	–
58	KD	10198	10199	1942	27347	17.71	26.28	5.12	2.79	2.22	15.66	–	–	–

Table 18: Experimental results for scene "*teapot12*".

$$N = 9264,$$
$$TP_D:\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 226198,\quad N^{hit}_{prim} = 161546,\quad N_{sec} = 226089,\quad N^{hit}_{sec} = 67517,$$
$$N_{shad} = 406274,\quad N^{hit}_{shad} = 34744,\quad T^{MIN}_{R}[s] = 4.03,\quad T_{app}[s] = 3.81,\quad T^{MIN}_{RSA}[s] = 0.22.$$

Scene = “*tetra6*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	4096	–	0	0	0	0.01	821.40	29.00	1.0	27380.00
1	spatmed-xyz(16,2)	12623	12624	4392	49152	53.58	29.07	5.75	4.10	0.27	3.30	29.00	0.39	81.00
2	objmed-xyz(16,2)	2891	2892	1868	4096	16.28	30.27	6.24	5.50	0.25	2.37	29.00	0.16	50.00
3	objmed(16,2)	3199	3200	2176	4096	10.47	52.50	11.07	10.59	0.36	2.86	29.00	0.07	66.33
4	OSAH(16,2)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.40	1.83	29.00	0.20	32.00
5	OSAH-RMI(16,2)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.39	1.84	29.00	0.20	32.33
6	OSAH-xyz(16,2)	2819	2820	1820	4096	13.24	17.20	3.36	2.79	0.26	1.87	29.00	0.21	33.33
7	OSAH(8,1)	175	176	30	4096	166.63	9.37	2.05	0.98	0.30	4.43	29.00	0.86	118.67
8	OSAH(8,2)	175	176	30	4096	166.63	9.37	2.05	0.98	0.28	4.44	29.00	0.86	119.00
9	OSAH(16,1)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.40	1.78	29.00	0.20	30.33
10	OSAH(16,2)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.40	1.83	29.00	0.20	32.00
11	OSAH(24,1)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.43	1.78	29.00	0.20	30.33
12	OSAH(24,2)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.39	1.77	29.00	0.20	30.00
13	OSAH(atc)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.40	1.77	29.00	0.20	30.00
14	OSAH2(atc)	2859	2860	1836	4096	10.47	18.51	3.04	2.56	0.49	1.92	29.00	0.17	35.00
15	OSAH+LC(atc)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.38	1.77	29.00	0.20	30.00
16	OSAH+TPC(atc)	2961	2962	1938	4096	10.47	14.81	2.78	2.30	0.38	1.75	29.00	0.20	29.33
17	OSAH+TPC+LC(atc)	2961	2962	1938	4096	10.47	14.81	2.78	2.30	0.44	1.76	29.00	0.20	29.67
18	OSAH+LC(16,1)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.44	1.76	29.00	0.20	29.67
19	OSAH+TPC(16,1)	2961	2962	1938	4096	10.47	14.81	2.78	2.30	0.44	1.66	29.00	0.20	26.33
20	OSAH+TPC+LC(16,1)	2961	2962	1938	4096	10.47	14.81	2.78	2.30	0.42	1.76	29.00	0.20	29.67
21	OSAH+PR(atc)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.40	1.85	29.00	0.20	32.67
22	OSAH+SC(atc)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.43	1.82	29.00	0.20	31.67
23	OSAH+GCM(atc)	3210	3211	2187	4096	10.47	16.03	2.96	2.48	10.39	1.72	29.00	0.19	28.33
24	OSAH+GCM2(atc)	3288	3289	2265	4096	10.47	16.38	3.01	2.53	10.77	1.73	29.00	0.18	28.67
25	OSAH+GCM3(atc)	3467	3468	2447	4096	11.02	25.52	4.44	3.94	11.29	2.03	29.00	0.13	38.67
26	OSAH+PAR(atc)	2971	2972	1948	4096	8.42	13.01	2.55	2.21	0.50	1.33	26.50	0.46	40.00
27	PARSAH+PAR(atc)	3155	3156	2132	4096	8.42	13.85	2.75	2.41	0.42	1.35	26.50	0.44	41.00
28	OSAH+PER(atc)	2971	2972	1948	4096	7.55	13.33	2.59	2.24	0.49	1.45	30.50	0.48	42.00
29	PERSAH+PER(atc)	3291	3292	2268	4096	7.55	15.80	3.10	2.75	8.22	1.46	30.50	0.39	42.50
30	SPHSAH+PER(atc)	2459	2460	1636	4096	33.13	18.33	3.40	2.81	0.70	1.93	30.50	0.54	66.00
31	OSAH+SPH(atc)	2971	2972	1948	4096	7.54	13.17	2.56	2.21	0.46	1.71	19.80	0.40	14.40
32	SPHSAH+SPH(atc)	2459	2460	1636	4096	32.92	18.07	3.35	2.77	0.72	2.25	19.80	0.53	25.20
33	OSAH+TA$_{seq}$(16,2)	2971	2972	1948	4096	10.47	32.60	2.79	2.31	0.32	2.48	29.00	0.11	53.67
34	OSAH+TA$^{A}_{rec}$(16,2)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.31	1.88	29.00	0.17	33.67
35	OSAH+TA$^{B}_{rec}$(16,2)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.30	1.73	29.00	0.20	28.67
36	OSAH+TA$_{SNL}$(16,2)	2971	2972	1948	4096	10.47	13.62	2.79	2.31	0.35	1.69	29.00	0.21	27.33
37	OSAH+TA$_{NLT}$(16,2)	10067	2972	1948	4096	10.47	12.55	2.79	2.31	0.37	1.70	29.00	0.21	27.67
38	OSAH+TA$_{seq}$(18,2)	2971	2972	1948	4096	10.47	32.60	2.79	2.31	0.32	2.49	29.00	0.10	54.00
39	OSAH+TA$^{A}_{rec}$(18,2)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.32	1.87	29.00	0.16	33.33
40	OSAH+TA$^{B}_{rec}$(18,2)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.31	1.65	29.00	0.20	26.00
41	OSAH+TA$_{SNL}$(18,2)	2971	2972	1948	4096	10.47	13.62	2.79	2.31	0.35	1.72	29.00	0.18	28.33
42	OSAH+TA$_{NLT}$(18,2)	10067	2972	1948	4096	10.47	12.55	2.79	2.31	0.37	1.70	29.00	0.19	27.67
43	OSAH+TA$_{seq}$(atc)	2971	2972	1948	4096	10.47	32.60	2.79	2.31	0.32	2.49	29.00	0.11	54.00
44	OSAH+TA$^{A}_{rec}$(atc)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.30	1.88	29.00	0.17	33.67
45	OSAH+TA$^{B}_{rec}$(atc)	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.31	1.73	29.00	0.20	28.67
46	OSAH+TA$_{SNL}$(atc)	2971	2972	1948	4096	10.47	13.62	2.79	2.31	0.36	1.81	29.00	0.18	31.33
47	OSAH+TA$_{NLT}$(atc)	10067	2972	1948	4096	10.47	12.55	2.79	2.31	0.37	1.71	29.00	0.20	28.00
48	BVH	458	2629	0	4096	196.66	60.58	45.93	0.00	0.76	148.45	–	–	–
49	O84	6189	43324	21100	110592	47.07	44.33	7.71	5.96	0.40	6.91	–	–	–
50	O89	6189	43324	21100	110592	46.96	30.14	7.71	5.97	0.32	5.27	–	–	–
51	BSP	12623	12624	4392	49152	53.58	29.07	5.75	4.10	0.41	5.36	–	–	–
52	O93	6189	43324	21100	110592	44.82	24.96	9.12	7.45	0.36	6.46	–	–	–
53	UG	0	21952	18444	25612	70.84	12.82	12.82	11.08	0.21	4.16	–	–	–
54	AG	969	35045	18069	88028	80.99	15.88	14.03	10.95	1.60	7.48	–	–	–
55	HUG	1	12167	9913	19600	78.62	11.22	10.56	8.91	0.26	4.98	–	–	–
56	RG	1565	40404	14220	218944	118.31	12.31	9.98	6.83	0.47	6.96	–	–	–
57	O84A	6449	45144	20976	120336	25.43	35.86	6.59	5.55	1.33	5.54	–	–	–
58	KD	2971	2972	1948	4096	10.47	14.83	2.79	2.31	0.48	2.66	–	–	–

Table 19: Experimental results for scene “*tetra6*”.

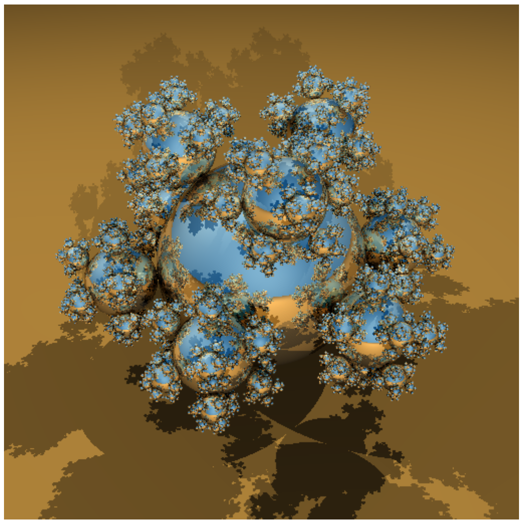
$$N = 4096,$$
$$TP_D: \quad N_{prim} = 263169, \quad N^{hit}_{\mathcal{AB}} = 159213, \quad N^{hit}_{prim} = 49950, \quad N_{sec} = 0, \quad N^{hit}_{sec} = 0,$$
$$N_{shad} = 46262, \quad N^{hit}_{shad} = 5552, \quad T^{MIN}_{R}[s] = 0.9, \quad T_{app}[s] = 0.87, \quad T^{MIN}_{RSA}[s] = 0.03.$$

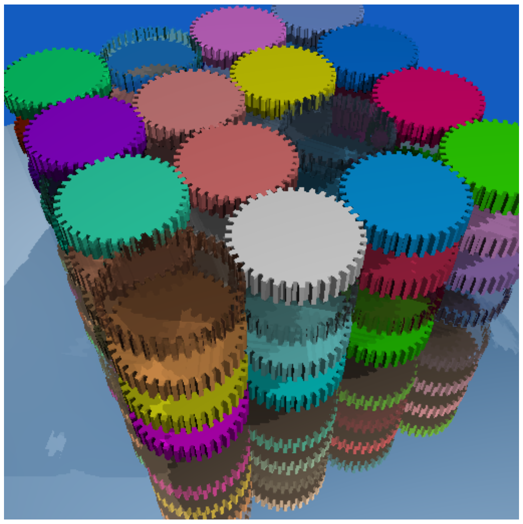
Scene = “*tree11*”

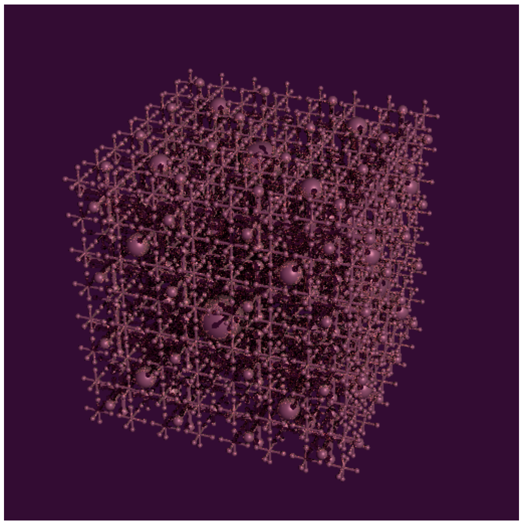
Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	8191	–	0	0	0	0.01	14902.70	18.14	1.0	67739.55
1	spatmed-xyz(16,2)	318	319	226	9377	2680.80	63.27	13.21	7.02	0.23	746.48	18.14	0.94	3374.95
2	objmed-xyz(16,2)	24036	24037	4478	41989	390.41	175.41	37.70	5.27	1.11	125.86	18.14	0.45	553.95
3	objmed(16,2)	20719	20720	2295	40789	625.25	198.81	48.32	3.49	1.67	198.83	18.14	0.54	885.64
4	OSAH(16,2)	2736	2737	937	10501	27.22	14.17	3.57	0.77	1.59	12.30	18.14	0.41	37.77
5	OSAH-RMI(16,2)	2736	2737	937	10501	27.22	14.17	3.57	0.77	1.58	12.04	18.14	0.41	36.59
6	OSAH-xyz(16,2)	2868	2869	955	10843	33.52	17.43	4.61	1.74	0.91	14.44	18.14	0.41	47.50
7	OSAH(8,1)	58	59	6	8451	632.46	10.33	2.90	0.13	1.14	155.95	18.14	0.96	690.73
8	OSAH(8,2)	56	57	6	8451	633.45	10.08	2.85	0.13	1.14	157.25	18.14	0.96	696.64
9	OSAH(16,1)	3014	3015	1159	10543	25.04	14.78	3.68	0.87	1.61	11.74	18.14	0.38	35.23
10	OSAH(16,2)	2736	2737	937	10501	27.22	14.17	3.57	0.77	1.59	12.30	18.14	0.41	37.77
11	OSAH(24,1)	32483	32484	10236	35110	18.71	15.76	3.84	1.00	2.50	10.69	18.14	0.30	30.45
12	OSAH(24,2)	17687	17688	3105	27487	21.38	14.80	3.67	0.83	2.18	10.86	18.14	0.35	31.23
13	OSAH(atc)	4369	4370	1743	11327	22.24	14.94	3.70	0.91	1.65	11.53	18.14	0.35	34.27
14	OSAH2(atc)	9579	9580	2424	16533	17.41	18.58	4.22	1.82	2.17	11.39	18.14	0.26	33.64
15	OSAH+LC(atc)	4397	4398	1743	11396	22.24	14.97	3.71	0.91	1.66	11.39	18.14	0.35	33.64
16	OSAH+TPC(atc)	4364	4365	1775	11274	22.37	14.88	3.69	0.91	1.74	11.40	18.14	0.36	33.68
17	OSAH+TPC+LC(atc)	4463	4464	1775	11511	22.32	14.97	3.71	0.91	2.00	10.82	18.14	0.35	31.05
18	OSAH+LC(16,1)	3014	3015	1159	10543	25.04	14.78	3.68	0.87	1.76	11.19	18.14	0.38	32.73
19	OSAH+TPC(16,1)	3073	3074	1215	10540	24.96	14.79	3.68	0.87	1.90	11.17	18.14	0.38	32.64
20	OSAH+TPC+LC(16,1)	3073	3074	1215	10540	24.96	14.79	3.68	0.87	1.91	11.19	18.14	0.38	32.73
21	OSAH+PR(atc)	4369	4370	1743	11107	21.58	14.65	3.63	0.92	1.74	10.59	18.14	0.34	30.00
22	OSAH+SC(atc)	4301	4302	1935	10684	21.07	14.68	3.66	1.01	2.26	10.76	18.14	0.34	30.77
23	OSAH+GCM(atc)	4834	4835	1885	11764	21.30	15.24	3.72	0.97	18.36	10.69	18.14	0.34	30.45
24	OSAH+GCM2(atc)	8042	8043	1742	19799	22.89	18.65	4.37	1.44	30.71	11.87	18.14	0.31	35.82
25	OSAH+GCM3(atc)	5868	5869	1517	15894	58.22	23.21	4.77	0.97	24.57	18.80	18.14	0.48	67.32
26	OSAH+PAR(atc)	4369	4370	1743	11327	4.59	19.90	5.18	1.44	1.87	5.79	16.22	0.51	8.96
27	PARSAH+PAR(atc)	5196	5197	2031	12654	3.64	16.05	4.01	0.92	2.44	5.48	16.22	0.54	7.61
28	OSAH+PER(atc)	4369	4370	1743	11327	7.94	23.89	5.96	1.95	1.91	4.76	11.76	0.56	16.24
29	PERSAH+PER(atc)	7047	7048	2750	14300	4.62	19.69	5.13	2.75	376.04	4.16	11.76	0.54	12.71
30	SPHSAH+PER(atc)	4400	4401	1456	19110	126.16	21.46	5.25	2.30	13.69	26.10	11.76	0.96	141.76
31	OSAH+SPH(atc)	4369	4370	1743	11327	7.88	23.69	5.90	1.94	1.93	5.04	16.00	0.45	12.00
32	SPHSAH+SPH(atc)	4400	4401	1456	19110	126.27	21.33	5.21	2.26	13.67	26.92	16.00	0.96	133.56
33	OSAH+TA$_{seq}$(16,2)	2736	2737	937	10501	27.33	32.45	3.58	0.77	1.33	15.84	18.14	0.26	53.86
34	OSAH+TA$^{A}_{rec}$(16,2)	2736	2737	937	10501	27.22	14.17	3.57	0.77	1.32	12.20	18.14	0.38	37.32
35	OSAH+TA$^{B}_{rec}$(16,2)	2736	2737	937	10501	27.22	14.17	3.57	0.77	1.33	11.54	18.14	0.41	34.32
36	OSAH+TA$_{SNL}$(16,2)	2736	2737	937	10501	27.35	13.78	3.58	0.77	1.38	12.80	18.14	0.35	40.05
37	OSAH+TA$_{NLT}$(16,2)	19843	2737	937	10501	27.35	12.54	3.58	0.77	1.37	12.61	18.14	0.36	39.18
38	OSAH+TA$_{seq}$(18,2)	5370	5371	1792	12405	23.12	33.85	3.63	0.81	1.46	15.22	18.14	0.22	51.05
39	OSAH+TA$^{A}_{rec}$(18,2)	5370	5371	1792	12405	23.01	14.51	3.62	0.81	1.45	11.51	18.14	0.33	34.18
40	OSAH+TA$^{B}_{rec}$(18,2)	5370	5371	1792	12405	23.01	14.51	3.62	0.81	1.43	10.64	18.14	0.37	30.23
41	OSAH+TA$_{SNL}$(18,2)	5370	5371	1792	12405	23.13	14.11	3.63	0.81	1.53	11.10	18.14	0.35	32.32
42	OSAH+TA$_{NLT}$(18,2)	37771	5371	1792	12405	23.13	12.81	3.63	0.81	1.57	11.27	18.14	0.34	33.09
43	OSAH+TA$_{seq}$(atc)	4369	4370	1743	11327	22.34	34.95	3.71	0.91	1.38	15.38	18.14	0.21	51.77
44	OSAH+TA$^{A}_{rec}$(atc)	4369	4370	1743	11327	22.24	14.94	3.70	0.91	1.36	11.58	18.14	0.31	34.50
45	OSAH+TA$^{B}_{rec}$(atc)	4369	4370	1743	11327	22.24	14.94	3.70	0.91	1.38	10.71	18.14	0.35	30.55
46	OSAH+TA$_{SNL}$(atc)	4369	4370	1743	11327	22.36	14.52	3.71	0.91	1.43	12.02	18.14	0.29	36.50
47	OSAH+TA$_{NLT}$(atc)	31626	4370	1743	11327	22.36	13.16	3.71	0.91	1.47	11.26	18.14	0.32	33.05
48	BVH	1411	5513	0	8191	118.86	50.11	33.17	0.00	1.38	495.04	–	–	–
49	O84	150	1051	886	10402	2002.70	132.22	21.82	14.57	0.26	676.30	–	–	–
50	O89	150	1051	886	10402	2000.20	85.94	21.81	14.57	0.25	653.08	–	–	–
51	BSP	318	319	226	9377	2680.80	63.27	13.21	7.02	1.54	908.03	–	–	–
52	O93	150	1051	886	10402	1997.50	72.66	28.90	21.65	0.26	664.61	–	–	–
53	UG	0	34968	26207	17297	2311.90	12.40	12.40	8.68	0.30	679.65	–	–	–
54	AG	916	52609	37188	37463	19.63	13.31	12.57	10.65	1.78	20.61	–	–	–
55	HUG	9	24680	21625	15584	58.40	6.70	4.07	3.18	0.50	38.62	–	–	–
56	RG	498	32174	19626	60830	74.48	14.55	12.83	8.25	0.27	32.55	–	–	–
57	O84A	931	6518	3001	15157	32.13	38.13	9.10	6.10	1.16	35.03	–	–	–
58	KD	2736	2737	937	10501	27.22	14.17	3.57	0.77	2.00	22.26	–	–	–

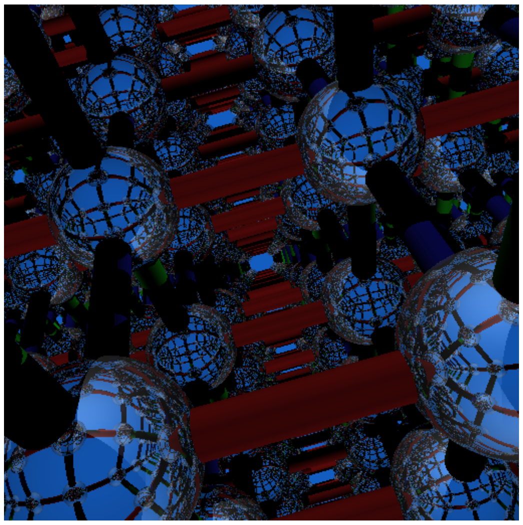
Table 20: Experimental results for scene “*tree11*”.

$$N = 8191,$$
$$TP_D\colon\ \ N_{prim} = 263169,\ \ N^{hit}_{\mathcal{AB}} = 263169,\ \ N^{hit}_{prim} = 169904,\quad N_{sec} = 0,\ \ N^{hit}_{sec} = 0,$$
$$N_{shad} = 1097802,\ \ N^{hit}_{shad} = 42522,\quad T^{MIN}_{R}[s] = 4.21,\ \ T_{app}[s] = 3.99,\ \ T^{MIN}_{RSA}[s] = 0.22.$$


balls4


gears4


jacks4


lattice12

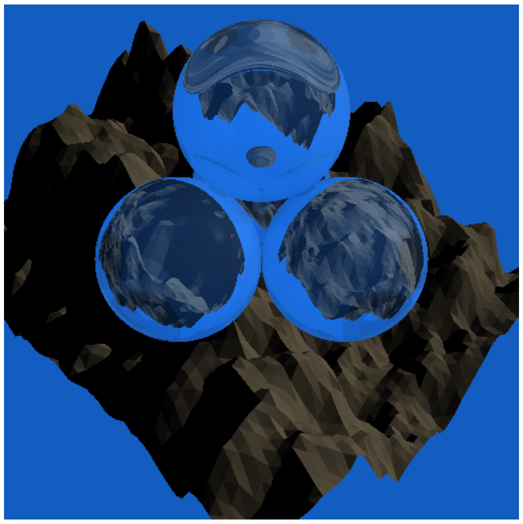
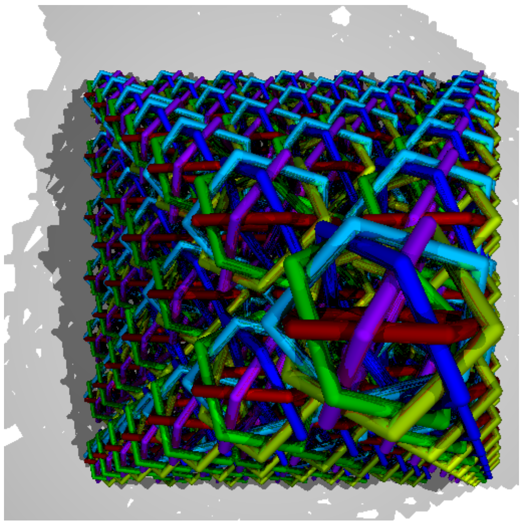
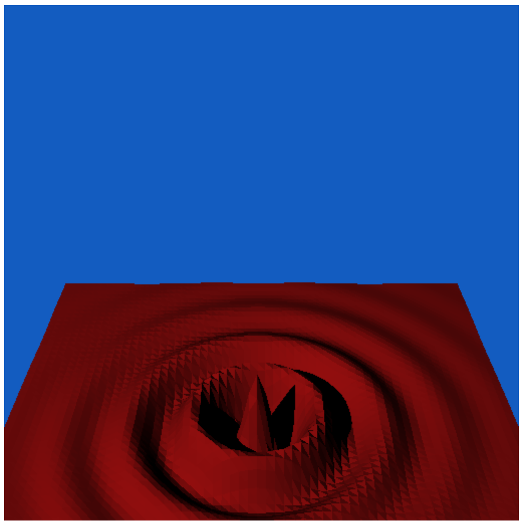
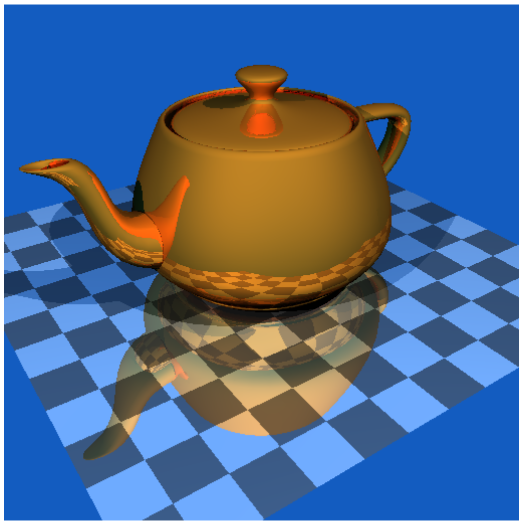
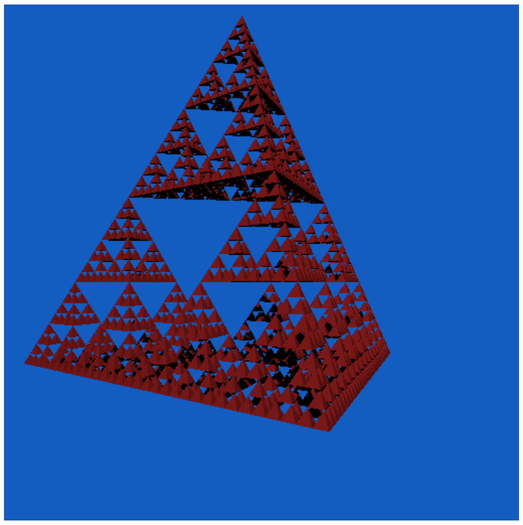

mount6

Figure 3: Visualization of the G^4_{SPD} scenes using the testing procedure TP_D.


rings7


sombrero2


teapot12


tetra6

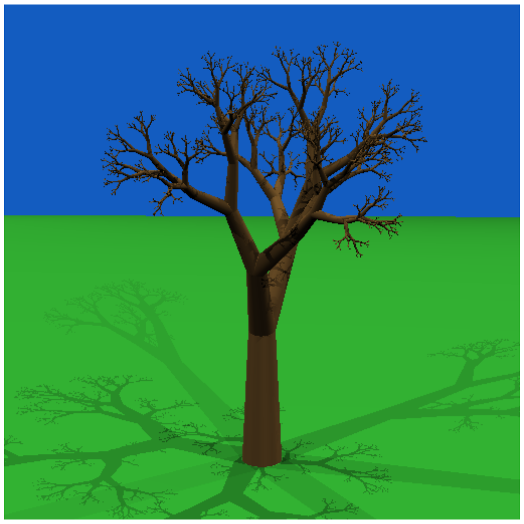

tree11

Figure 4: Visualization of the G^4_{SPD} scenes using the testing procedure TP_D.

Scene = “*balls5*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	66431	–	0	0	0	0.13	56977.20	12.21	1.0	121228.09
1	spatmed-xyz(16,2)	589	590	260	77205	2094.20	55.39	10.41	5.73	2.00	798.14	12.21	0.96	1685.96
2	objmed-xyz(16,2)	65201	65202	506	268019	290.57	178.26	37.57	0.86	9.21	136.08	12.21	0.51	277.32
3	objmed(16,2)	65390	65391	19	234572	288.46	194.32	42.20	0.02	14.37	147.99	12.21	0.49	302.66
4	OSAH(16,2)	7130	7131	888	84927	46.33	27.48	5.26	1.44	14.57	24.85	12.21	0.52	40.66
5	OSAH-RMI(16,2)	7130	7131	888	84927	46.33	27.48	5.26	1.44	14.64	24.33	12.21	0.52	39.55
6	OSAH-xyz(16,2)	8073	8074	1048	87604	47.58	29.90	5.89	2.28	8.72	24.90	12.21	0.51	40.77
7	OSAH(8,1)	99	100	13	68241	3100.80	13.68	3.15	0.32	10.27	1387.11	12.21	0.99	2939.09
8	OSAH(8,2)	97	98	13	68240	3100.80	13.46	3.08	0.32	10.17	1382.27	12.21	0.99	2928.79
9	OSAH(16,1)	7949	7950	1105	85475	45.76	28.90	5.55	1.58	14.63	24.74	12.21	0.51	40.43
10	OSAH(16,2)	7130	7131	888	84927	46.33	27.48	5.26	1.44	14.57	24.85	12.21	0.52	40.66
11	OSAH(24,1)	243317	243318	50718	347811	11.44	37.63	7.01	2.36	22.50	19.48	12.21	0.17	29.23
12	OSAH(24,2)	161104	161105	21479	295835	12.82	34.38	6.38	1.99	20.81	18.92	12.21	0.20	28.04
13	OSAH(atc)	80093	80094	14316	173040	13.52	35.17	6.59	2.14	18.02	19.27	12.21	0.20	28.79
14	OSAH2(atc)	139692	139693	16599	249757	12.24	44.56	7.40	3.10	25.30	20.97	12.21	0.15	32.40
15	OSAH+LC(atc)	80573	80574	14363	174144	12.95	35.49	6.60	2.44	18.16	19.15	12.21	0.19	28.53
16	OSAH+TPC(atc)	79446	79447	14231	172312	13.52	35.16	6.59	2.14	19.12	19.26	12.21	0.20	28.77
17	OSAH+TPC+LC(atc)	80549	80550	14274	174925	12.95	35.49	6.60	2.44	21.76	18.30	12.21	0.19	26.72
18	OSAH+LC(16,1)	7982	7983	1138	85475	45.19	29.21	5.56	1.87	16.38	23.54	12.21	0.50	37.87
19	OSAH+TPC(16,1)	8065	8066	1212	85468	45.70	28.96	5.57	1.60	17.20	23.60	12.21	0.51	38.00
20	OSAH+TPC+LC(16,1)	8099	8100	1246	85468	45.13	29.27	5.57	1.90	17.26	23.56	12.21	0.50	37.91
21	OSAH+PR(atc)	80093	80094	14316	136996	11.12	35.69	6.70	2.19	21.02	18.17	12.21	0.17	26.45
22	OSAH+SC(atc)	73702	73703	20822	130584	10.69	34.42	6.42	2.74	32.29	18.34	12.21	0.17	26.81
23	OSAH+GCM(atc)	97257	97258	17400	196761	12.96	36.04	6.68	2.25	293.40	19.11	12.21	0.19	28.45
24	OSAH+GCM2(atc)	184866	184867	10488	425021	14.49	41.04	7.36	2.10	545.81	20.86	12.21	0.19	32.17
25	OSAH+GCM3(atc)	146592	146593	16774	305799	28.27	50.31	8.42	2.67	469.09	25.87	12.21	0.27	42.83
26	OSAH+PAR(atc)	80093	80094	14316	173040	6.82	39.41	8.27	3.14	20.75	5.09	10.65	0.33	11.48
27	PARSAH+PAR(atc)	80741	80742	14464	174575	6.72	38.47	8.00	2.98	25.29	5.07	10.65	0.34	11.39
28	OSAH+PER(atc)	80093	80094	14316	173040	6.57	39.13	8.24	3.37	20.75	5.06	9.92	0.29	9.54
29	PERSAH+PER(atc)	134766	134767	26449	233120	5.85	39.52	8.12	3.21	3515.50	5.27	9.92	0.34	10.35
30	SPHSAH+PER(atc)	72559	72560	13467	168828	6.72	38.14	8.03	3.34	118.20	5.07	9.92	0.31	9.58
31	OSAH+SPH(atc)	80093	80094	14316	173040	6.48	38.55	8.11	3.32	20.68	5.36	10.74	0.29	9.11
32	SPHSAH+SPH(atc)	72559	72560	13467	168828	6.67	37.66	7.94	3.29	118.09	5.36	10.74	0.31	9.11
33	OSAH+TA$_{seq}$(16,2)	7130	7131	888	84927	50.30	69.09	5.45	1.44	11.95	31.75	12.21	0.35	55.34
34	OSAH+TA$^{A}_{rec}$(16,2)	7130	7131	888	84927	46.33	27.48	5.26	1.44	11.93	24.98	12.21	0.47	40.94
35	OSAH+TA$^{B}_{rec}$(16,2)	7130	7131	888	84927	46.33	27.48	5.26	1.44	11.93	23.14	12.21	0.52	37.02
36	OSAH+TA$_{SNL}$(16,2)	7130	7131	888	84927	50.34	24.78	5.45	1.44	12.10	24.89	12.21	0.47	40.74
37	OSAH+TA$_{NLT}$(16,2)	55966	7131	888	84927	50.34	22.04	5.45	1.44	12.10	25.17	12.21	0.47	41.34
38	OSAH+TA$_{seq}$(18,2)	18889	18890	2866	101020	25.66	79.35	5.87	1.67	12.92	28.20	12.21	0.20	47.79
39	OSAH+TA$^{A}_{rec}$(18,2)	18889	18890	2866	101020	23.68	29.95	5.66	1.66	12.87	21.15	12.21	0.29	32.79
40	OSAH+TA$^{B}_{rec}$(18,2)	18889	18890	2866	101020	23.68	29.95	5.66	1.66	12.92	19.06	12.21	0.34	28.34
41	OSAH+TA$_{SNL}$(18,2)	18889	18890	2866	101020	25.69	26.95	5.87	1.67	13.24	20.09	12.21	0.32	30.53
42	OSAH+TA$_{NLT}$(18,2)	145833	18890	2866	101020	25.69	23.77	5.87	1.67	13.40	20.23	12.21	0.31	30.83
43	OSAH+TA$_{seq}$(atc)	80093	80094	14316	173040	14.56	102.19	6.84	2.15	14.87	29.14	12.21	0.11	49.79
44	OSAH+TA$^{A}_{rec}$(atc)	80093	80094	14316	173040	13.51	35.17	6.59	2.14	14.82	20.63	12.21	0.17	31.68
45	OSAH+TA$^{B}_{rec}$(atc)	80093	80094	14316	173040	13.52	35.17	6.59	2.14	14.79	18.22	12.21	0.20	26.55
46	OSAH+TA$_{SNL}$(atc)	80093	80094	14316	173040	14.60	31.30	6.85	2.15	16.44	19.40	12.21	0.18	29.06
47	OSAH+TA$_{NLT}$(atc)	590522	80094	14316	173040	14.59	27.12	6.85	2.15	16.92	19.38	12.21	0.18	29.02
48	BVH	9936	45674	0	66431	827.35	1116.40	780.87	0.00	17.11	12335.30	–	–	–
49	O84	301	2108	1071	87856	1117.00	115.41	18.63	12.07	1.93	463.25	–	–	–
50	O89	301	2108	1071	87856	1131.40	73.33	18.62	12.07	1.91	447.03	–	–	–
51	BSP	589	590	260	77205	2094.20	55.39	10.41	5.73	2.88	862.52	–	–	–
52	O93	301	2108	1071	87856	1107.10	62.98	24.49	18.02	1.91	464.01	–	–	–
53	UG	0	329219	298274	106007	403.52	19.49	19.49	15.89	2.67	157.70	–	–	–
54	AG	13736	508722	147921	583874	31.44	24.57	22.17	16.36	287.72	43.76	–	–	–
55	HUG	21	199570	176549	142830	36.00	25.61	22.03	18.77	4.55	67.30	–	–	–
56	RG	20704	669181	127121	5540218	65.97	25.09	20.81	13.82	10.91	60.05	–	–	–
57	O84A	3063	21442	5275	120197	37.28	51.08	9.59	5.45	8.83	48.61	–	–	–
58	KD	7130	7131	888	84927	46.33	27.48	5.26	1.44	16.77	42.00	–	–	–

Table 21: Experimental results for scene “*balls5*”.

$$N = 66431,$$
$$TP_D\colon\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 263169,\quad N^{hit}_{prim} = 263169,\quad N_{sec} = 195150,\quad N^{hit}_{sec} = 150302,$$
$$N_{shad} = 979728,\quad N^{hit}_{shad} = 314095,\quad T^{MIN}_{R}[s] = 6.21,\quad T_{app}[s] = 5.74,\quad T^{MIN}_{RSA}[s] = 0.47.$$

Scene = "*gears9*"

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	106435	–	0	0	0	0.22	129294.00	6.49	1.0	99456.92
1	spatmed-xyz(16,2)	14427	14428	1760	344324	79.83	20.55	2.76	0.74	4.12	45.91	6.49	0.93	28.82
2	objmed-xyz(16,2)	57571	57572	4708	492923	198.54	74.87	14.40	1.99	16.71	127.53	6.49	0.91	91.61
3	objmed(16,2)	65535	65536	0	386404	187.29	79.83	18.06	0.00	21.34	130.08	6.49	0.89	93.57
4	OSAH(16,2)	21814	21815	4114	163635	19.53	19.97	2.62	0.48	19.36	31.77	6.49	0.78	17.95
5	OSAH-RMI(16,2)	21814	21815	4114	163635	19.53	19.97	2.62	0.48	19.90	32.83	6.49	0.78	18.76
6	OSAH-xyz(16,2)	16173	16174	2576	151279	27.50	25.01	4.04	2.08	12.91	33.93	6.49	0.80	19.61
7	OSAH(8,1)	155	156	0	109968	1128.00	9.59	1.56	0.00	14.28	560.61	6.49	1.00	424.75
8	OSAH(8,2)	155	156	0	109968	1128.00	9.59	1.56	0.00	14.30	563.81	6.49	1.00	427.21
9	OSAH(16,1)	23474	23475	4114	163747	18.77	20.49	2.66	0.48	20.02	30.03	6.49	0.77	16.61
10	OSAH(16,2)	21814	21815	4114	163635	19.53	19.97	2.62	0.48	19.36	31.77	6.49	0.78	17.95
11	OSAH(24,1)	864109	864110	37583	1386750	6.52	27.47	3.65	0.76	67.35	42.04	6.49	0.46	25.85
12	OSAH(24,2)	498216	498217	17194	1081377	8.96	24.91	3.29	0.58	46.74	35.97	6.49	0.56	21.18
13	OSAH(atc)	392145	392146	16164	757083	7.56	26.16	3.41	0.66	38.07	31.51	6.49	0.51	17.75
14	OSAH2(atc)	425150	425151	19119	811415	7.53	27.58	3.55	0.79	54.57	34.85	6.49	0.50	20.32
15	OSAH+LC(atc)	425416	425417	17751	838913	7.42	26.43	3.44	0.67	41.07	33.57	6.49	0.50	19.33
16	OSAH+TPC(atc)	271549	271550	10789	618988	8.40	25.37	3.30	0.63	33.68	28.37	6.49	0.54	15.33
17	OSAH+TPC+LC(atc)	361216	361217	12355	811187	8.02	25.97	3.37	0.63	47.72	33.15	6.49	0.53	19.01
18	OSAH+LC(16,1)	23486	23487	4126	163747	18.77	20.49	2.66	0.48	22.69	27.96	6.49	0.77	15.02
19	OSAH+TPC(16,1)	24355	24356	4857	164056	18.48	20.57	2.68	0.50	23.20	27.83	6.49	0.76	14.92
20	OSAH+TPC+LC(16,1)	24359	24360	4861	164056	18.48	20.57	2.68	0.50	23.44	29.68	6.49	0.76	16.34
21	OSAH+PR(atc)	392145	392146	16164	603661	7.03	26.35	3.52	0.68	106.75	35.90	6.49	0.48	21.12
22	OSAH+SC(atc)	399893	399894	18269	757150	7.51	26.18	3.41	0.67	79.88	33.99	6.49	0.51	19.65
23	OSAH+GCM(atc)	606852	606853	26635	1056116	7.71	28.26	3.74	0.54	1982.32	38.46	6.49	0.50	23.09
24	OSAH+GCM2(atc)	997410	997411	43167	1812599	7.83	30.39	4.04	0.70	4111.98	47.49	6.49	0.48	30.04
25	OSAH+GCM3(atc)	592775	592776	27821	1086071	13.27	36.44	4.60	0.84	2257.84	41.25	6.49	0.57	25.24
26	OSAH+PAR(atc)	392145	392146	16164	757083	2.79	18.24	2.46	0.39	43.20	12.56	6.55	0.89	15.10
27	PARSAH+PAR(atc)	408406	408407	17952	770637	2.67	18.00	2.42	0.41	49.98	14.30	6.55	0.91	18.10
28	OSAH+PER(atc)	392145	392146	16164	757083	3.13	20.87	2.62	0.44	44.93	13.55	6.05	0.89	15.81
29	PERSAH+PER(atc)	472765	472766	20764	872136	3.04	20.75	2.65	0.51	2379.36	13.20	6.05	0.89	15.24
30	SPHSAH+PER(atc)	481568	481569	37096	1048342	6.58	28.49	3.79	0.98	133.25	16.93	6.05	0.89	21.26
31	OSAH+SPH(atc)	392145	392146	16164	757083	3.12	20.56	2.59	0.43	45.11	13.27	6.03	0.89	14.70
32	SPHSAH+SPH(atc)	481568	481569	37096	1048342	6.52	28.42	3.80	1.02	136.83	18.80	6.03	0.90	23.34
33	OSAH+TA$_{seq}$(16,2)	21814	21815	4114	163635	26.06	38.74	2.70	0.49	15.81	32.18	6.49	0.63	18.26
34	OSAH+TA$^{A}_{rec}$(16,2)	21814	21815	4114	163635	19.53	19.97	2.62	0.48	15.94	29.81	6.49	0.70	16.44
35	OSAH+TA$^{B}_{rec}$(16,2)	21814	21815	4114	163635	19.53	19.97	2.62	0.48	16.00	27.73	6.49	0.78	14.84
36	OSAH+TA$_{SNL}$(16,2)	21814	21815	4114	163635	26.07	18.03	2.70	0.49	16.78	25.48	6.49	0.88	13.11
37	OSAH+TA$_{NLT}$(16,2)	106625	21815	4114	163635	26.07	17.29	2.70	0.49	16.96	26.42	6.49	0.84	13.83
38	OSAH+TA$_{seq}$(18,2)	57821	57822	4647	253474	17.20	46.19	3.00	0.50	17.18	27.75	6.49	0.61	14.85
39	OSAH+TA$^{A}_{rec}$(18,2)	57821	57822	4647	253474	12.72	21.96	2.89	0.49	17.38	27.78	6.49	0.60	14.88
40	OSAH+TA$^{B}_{rec}$(18,2)	57821	57822	4647	253474	12.72	21.96	2.89	0.49	17.16	25.63	6.49	0.68	13.22
41	OSAH+TA$_{SNL}$(18,2)	57821	57822	4647	253474	17.20	19.93	3.00	0.50	18.92	25.71	6.49	0.68	13.28
42	OSAH+TA$_{NLT}$(18,2)	311851	57822	4647	253474	17.20	18.95	3.00	0.50	20.48	35.13	6.49	0.44	20.53
43	OSAH+TA$_{seq}$(atc)	392145	392146	16164	757083	9.70	60.34	3.49	0.67	35.25	41.06	6.49	0.35	25.09
44	OSAH+TA$^{A}_{rec}$(atc)	392145	392146	16164	757083	7.56	26.16	3.41	0.66	33.16	34.35	6.49	0.44	19.93
45	OSAH+TA$^{B}_{rec}$(atc)	392145	392146	16164	757083	7.56	26.16	3.41	0.66	33.56	30.69	6.49	0.51	17.12
46	OSAH+TA$_{SNL}$(atc)	392145	392146	16164	757083	9.71	23.16	3.49	0.67	45.09	44.82	6.49	0.31	27.98
47	OSAH+TA$_{NLT}$(atc)	1656236	392146	16164	757083	9.71	21.35	3.49	0.67	67.40	57.12	6.49	0.23	37.45
48	BVH	13245	70137	0	106435	456.47	459.87	340.89	0.00	160.09	10934.30	–	–	–
49	O84	7461	52228	16228	583908	56.62	22.89	4.13	1.89	5.22	60.22	–	–	–
50	O89	7461	52228	16228	583908	54.38	18.62	4.13	1.89	4.81	55.77	–	–	–
51	BSP	14427	14428	1760	344324	79.83	20.55	2.76	0.74	5.93	112.41	–	–	–
52	O93	7461	52228	16228	583908	41.55	14.26	5.04	3.26	4.67	56.16	–	–	–
53	UG	0	528384	420540	523636	25.16	22.64	22.64	19.67	7.78	44.49	–	–	–
54	AG	12	285159	83202	879690	28.40	7.47	5.73	0.48	306.02	1308.26	–	–	–
55	HUG	1	8712	5600	238592	270.82	7.14	6.14	4.12	2.43	184.99	–	–	–
56	RG	20325	659907	184514	4277637	45.72	17.57	15.16	11.38	15.10	134.40	–	–	–
57	O84A	14601	102208	25372	709952	29.41	32.77	5.99	3.47	22.52	54.41	–	–	–
58	KD	21814	21815	4114	163635	19.53	19.97	2.62	0.48	22.97	40.59	–	–	–

Table 22: Experimental results for scene "*gears9*".

$$N = 106435,$$
$$TP_D:\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 263169,\quad N^{hit}_{prim} = 247173,\quad N_{sec} = 163990,\quad N^{hit}_{sec} = 111495,$$
$$N_{shad} = 1365418,\quad N^{hit}_{shad} = 365632,\quad T^{MIN}_{R}[s] = 9.74,\quad T_{app}[s] = 8.44,\quad T^{MIN}_{RSA}[s] = 1.30.$$

Scene = “*jacks5*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	42129	–	0	0	0	0.09	26711.80	7.79	1.0	55649.58
1	spatmed-xyz(16,2)	25669	25670	1678	172204	52.52	36.70	6.41	1.73	1.55	28.38	7.79	0.79	51.33
2	objmed-xyz(16,2)	64941	64942	940	267797	76.95	61.14	11.54	0.48	6.77	41.88	7.79	0.77	79.46
3	objmed(16,2)	65146	65147	67	242024	78.18	66.66	12.88	0.01	10.67	44.81	7.79	0.75	85.56
4	OSAH(16,2)	36148	36149	2911	132833	33.52	38.63	6.68	1.81	10.23	23.39	7.79	0.69	40.94
5	OSAH-RMI(16,2)	36148	36149	2911	132833	33.52	38.63	6.68	1.81	10.20	23.26	7.79	0.69	40.67
6	OSAH-xyz(16,2)	36456	36457	2868	138336	35.88	38.77	6.72	1.68	6.05	24.23	7.79	0.71	42.69
7	OSAH(8,1)	249	250	18	53569	756.18	13.94	2.66	0.45	6.55	235.41	7.79	0.99	482.65
8	OSAH(8,2)	249	250	18	53569	756.18	13.94	2.66	0.45	6.55	235.31	7.79	0.99	482.44
9	OSAH(16,1)	39427	39428	5273	133547	31.55	39.97	6.92	2.57	10.28	22.81	7.79	0.67	39.73
10	OSAH(16,2)	36148	36149	2911	132833	33.52	38.63	6.68	1.81	10.23	23.39	7.79	0.69	40.94
11	OSAH(24,1)	622889	622890	52924	966375	19.48	65.98	11.47	4.01	25.83	24.32	7.79	0.43	42.88
12	OSAH(24,2)	350355	350356	11861	737969	24.67	54.98	9.57	2.16	19.58	24.17	7.79	0.54	42.56
13	OSAH(atc)	193204	193205	39615	288851	20.08	53.98	9.40	3.83	14.98	21.76	7.79	0.49	37.54
14	OSAH2(atc)	291149	291150	26100	438877	21.10	61.02	10.25	3.25	22.83	23.80	7.79	0.47	41.79
15	OSAH+LC(atc)	288425	288426	39623	542868	20.83	57.71	10.07	3.83	19.19	23.09	7.79	0.48	40.31
16	OSAH+TPC(atc)	128243	128244	27301	222654	21.77	50.00	8.69	3.54	14.31	21.47	7.79	0.53	36.94
17	OSAH+TPC+LC(atc)	250710	250711	27328	534275	22.57	55.61	9.70	3.54	21.86	22.70	7.79	0.51	39.50
18	OSAH+LC(16,1)	39430	39431	5276	133547	31.55	39.97	6.92	2.57	11.85	22.28	7.79	0.67	38.62
19	OSAH+TPC(16,1)	39609	39610	5443	133601	31.54	40.00	6.93	2.58	12.51	22.27	7.79	0.67	38.60
20	OSAH+TPC+LC(16,1)	39614	39615	5448	133601	31.54	40.00	6.93	2.58	12.55	22.29	7.79	0.67	38.65
21	OSAH+PR(atc)	193204	193205	39615	152791	12.71	54.33	9.47	3.83	26.34	18.43	7.79	0.38	30.60
22	OSAH+SC(atc)	200237	200238	106481	140781	8.16	53.38	9.26	6.00	45.07	15.48	7.79	0.28	24.46
23	OSAH+GCM(atc)	372561	372562	46743	576245	19.40	60.89	10.55	3.83	1111.27	23.14	7.79	0.45	40.42
24	OSAH+GCM2(atc)	888130	888131	8329	1640883	24.40	70.04	11.88	2.21	2736.52	27.82	7.79	0.48	50.17
25	OSAH+GCM3(atc)	500942	500943	28712	887100	25.71	71.16	11.57	3.18	1584.85	27.08	7.79	0.48	48.62
26	OSAH+PAR(atc)	193204	193205	39615	288851	10.19	32.57	6.09	3.69	17.05	3.42	4.09	0.40	11.45
27	PARSAH+PAR(atc)	318387	318388	24304	3198608	11.98	10.31	0.64	0.16	66.62	3.10	4.09	0.78	10.00
28	OSAH+PER(atc)	193204	193205	39615	288851	11.15	38.94	7.24	4.52	17.05	4.10	5.83	0.40	16.94
29	PERSAH+PER(atc)	204890	204891	43813	363136	7.67	25.30	4.28	2.66	3176.03	3.29	5.83	0.47	12.44
30	SPHSAH+PER(atc)	525576	525577	70994	1803037	68.63	51.01	9.42	4.69	253.52	10.85	5.83	0.76	54.44
31	OSAH+SPH(atc)	193204	193205	39615	288851	11.10	38.72	7.20	4.50	17.11	4.53	7.16	0.43	16.68
32	SPHSAH+SPH(atc)	525576	525577	70994	1803037	68.55	50.72	9.36	4.66	253.71	11.09	7.16	0.76	51.21
33	OSAH+TA$_{seq}$(16,2)	36148	36149	2911	132833	34.39	103.31	6.78	1.81	8.58	28.29	7.79	0.53	51.15
34	OSAH+TA$^{A}_{rec}$(16,2)	36148	36149	2911	132833	33.52	38.63	6.68	1.81	8.60	24.01	7.79	0.64	42.23
35	OSAH+TA$^{B}_{rec}$(16,2)	36148	36149	2911	132833	33.52	38.63	6.68	1.81	8.59	22.63	7.79	0.69	39.35
36	OSAH+TA$_{SNL}$(16,2)	36148	36149	2911	132833	34.39	29.66	6.78	1.81	9.21	24.63	7.79	0.62	43.52
37	OSAH+TA$_{NLT}$(16,2)	312886	36149	2911	132833	34.39	24.66	6.78	1.81	9.51	24.54	7.79	0.63	43.33
38	OSAH+TA$_{seq}$(18,2)	88006	88007	7223	209203	26.18	132.74	8.02	2.04	10.17	28.75	7.79	0.42	52.10
39	OSAH+TA$^{A}_{rec}$(18,2)	88006	88007	7223	209203	25.49	45.43	7.89	2.04	10.01	23.03	7.79	0.54	40.19
40	OSAH+TA$^{B}_{rec}$(18,2)	88006	88007	7223	209203	25.49	45.43	7.89	2.04	10.05	21.44	7.79	0.59	36.88
41	OSAH+TA$_{SNL}$(18,2)	88006	88007	7223	209203	26.19	35.10	8.02	2.05	11.76	23.40	7.79	0.53	40.96
42	OSAH+TA$_{NLT}$(18,2)	717084	88007	7223	209203	26.19	28.71	8.02	2.05	12.46	23.18	7.79	0.54	40.50
43	OSAH+TA$_{seq}$(atc)	193204	193205	39615	288851	20.74	170.30	9.54	3.83	12.82	30.22	7.79	0.32	55.17
44	OSAH+TA$^{A}_{rec}$(atc)	193204	193205	39615	288851	20.08	53.98	9.40	3.83	12.54	22.92	7.79	0.44	39.96
45	OSAH+TA$^{B}_{rec}$(atc)	193204	193205	39615	288851	20.08	53.98	9.40	3.83	12.55	20.78	7.79	0.49	35.50
46	OSAH+TA$_{SNL}$(atc)	193204	193205	39615	288851	20.75	41.62	9.55	3.83	16.31	23.00	7.79	0.43	40.12
47	OSAH+TA$_{NLT}$(atc)	1421499	193205	39615	288851	20.75	33.72	9.55	3.83	20.80	37.59	7.79	0.25	70.52
48	BVH	5214	27779	0	42129	676.63	759.71	551.99	0.00	23.41	5106.45	–	–	–
49	O84	14001	98008	12164	329450	36.09	56.84	8.87	3.06	2.15	39.53	–	–	–
50	O89	14001	98008	12164	329450	36.05	36.06	8.85	3.06	1.89	33.80	–	–	–
51	BSP	25669	25670	1678	172204	52.52	36.70	6.41	1.73	2.09	39.28	–	–	–
52	O93	14001	98008	12164	329450	34.63	33.80	13.70	8.10	1.92	40.03	–	–	–
53	UG	0	210145	139360	299206	38.81	18.95	18.95	13.49	2.80	30.69	–	–	–
54	AG	4511	194294	72699	360849	62.72	35.50	31.43	20.21	191.34	64.66	–	–	–
55	HUG	131	126560	83138	194877	59.80	25.33	20.92	13.06	3.56	55.27	–	–	–
56	RG	21873	324096	30464	1493122	63.32	19.36	15.05	7.40	3.87	50.37	–	–	–
57	O84A	22039	154274	17066	423070	32.27	63.54	9.87	3.39	8.74	40.20	–	–	–
58	KD	36148	36149	2911	132833	33.52	38.63	6.68	1.81	12.47	31.82	–	–	–

Table 23: Experimental results for scene “*jacks5*”.

$$N = 42129,$$
$$TP_D\text{:}\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 231363,\quad N^{hit}_{prim} = 115562,\quad N_{sec} = 281028,\quad N^{hit}_{sec} = 197780,$$
$$N_{shad} = 218826,\quad N^{hit}_{shad} = 116281,\quad T^{MIN}_{R}[s] = 4.22,\quad T_{app}[s] = 3.74,\quad T^{MIN}_{RSA}[s] = 0.48.$$

Scene = “*lattice29*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	105300	–	0	0	0	0.22	129143.00	8.73	1.0	107619.17
1	spatmed-xyz(16,2)	65535	65536	0	330512	28.28	35.71	5.03	0.00	3.75	53.39	8.73	0.72	35.76
2	objmed-xyz(16,2)	65223	65224	2640	243821	21.18	34.70	4.78	0.02	12.70	40.54	8.73	0.67	25.05
3	objmed(16,2)	65535	65536	0	233480	20.73	35.70	5.17	0.00	19.46	41.70	8.73	0.66	26.02
4	OSAH(16,2)	64531	64532	4969	216163	19.62	36.86	5.29	0.06	19.39	42.76	8.73	0.64	26.90
5	OSAH-RMI(16,2)	64531	64532	4969	216163	19.62	36.86	5.29	0.06	19.39	42.89	8.73	0.64	27.01
6	OSAH-xyz(16,2)	64433	64434	2668	221091	18.80	35.43	4.98	0.02	12.65	41.71	8.73	0.64	26.02
7	OSAH(8,1)	255	256	0	120512	708.01	11.58	1.53	0.00	14.40	835.01	8.73	1.00	687.11
8	OSAH(8,2)	255	256	0	120512	708.01	11.58	1.53	0.00	14.39	831.06	8.73	1.00	683.82
9	OSAH(16,1)	65159	65160	5597	216163	19.61	36.87	5.29	0.06	19.46	43.29	8.73	0.64	27.34
10	OSAH(16,2)	64531	64532	4969	216163	19.62	36.86	5.29	0.06	19.39	42.76	8.73	0.64	26.90
11	OSAH(24,1)	799987	799988	75291	881297	5.01	54.96	7.63	3.50	37.82	35.69	8.73	0.23	21.01
12	OSAH(24,2)	188841	188842	23435	322007	11.57	42.61	5.87	0.76	22.80	34.30	8.73	0.47	19.85
13	OSAH(atc)	710837	710838	75291	792147	5.11	54.80	7.62	3.50	34.46	31.86	8.73	0.23	17.82
14	OSAH2(atc)	710635	710636	53642	813609	6.24	54.50	7.60	2.55	46.59	34.37	8.73	0.27	19.91
15	OSAH+LC(atc)	713210	713211	75291	794520	5.11	54.80	7.62	3.50	36.84	31.90	8.73	0.23	17.85
16	OSAH+TPC(atc)	574762	574763	75161	656202	5.66	53.99	7.58	3.46	32.65	32.72	8.73	0.26	18.53
17	OSAH+TPC+LC(atc)	608309	608310	75168	689767	5.40	53.81	7.43	3.47	42.59	30.78	8.73	0.25	16.92
18	OSAH+LC(16,1)	65159	65160	5596	216164	18.93	36.86	5.23	0.07	23.43	41.80	8.73	0.63	26.10
19	OSAH+TPC(16,1)	65159	65160	5592	216168	20.03	36.88	5.29	0.07	23.69	43.39	8.73	0.64	27.42
20	OSAH+TPC+LC(16,1)	65159	65160	5592	216168	20.03	36.88	5.29	0.07	23.61	43.20	8.73	0.64	27.27
21	OSAH+PR(atc)	710837	710838	75291	791392	5.11	54.82	7.63	3.51	50.74	31.84	8.73	0.23	17.80
22	OSAH+SC(atc)	764409	764410	78072	842938	4.98	54.49	7.57	3.53	142.20	33.20	8.73	0.23	18.93
23	OSAH+GCM(atc)	704739	704740	74907	786433	4.81	52.26	7.45	3.62	1892.45	32.18	8.73	0.23	18.08
24	OSAH+GCM2(atc)	795680	795681	40160	912121	7.19	52.24	7.52	1.58	2130.64	35.18	8.73	0.31	20.58
25	OSAH+GCM3(atc)	697883	697884	47683	806865	7.73	52.79	7.84	1.89	1860.60	35.12	8.73	0.32	20.53
26	OSAH+PAR(atc)	710837	710838	75291	792147	37.94	190.83	35.81	22.44	39.62	12.57	8.56	0.34	61.28
27	PARSAH+PAR(atc)	371502	371503	114060	412682	44.15	95.94	24.04	10.08	40.72	10.08	8.56	0.57	47.44
28	OSAH+PER(atc)	710837	710838	75291	792147	4.89	65.95	10.02	5.26	39.74	7.33	9.28	0.33	9.51
29	PERSAH+PER(atc)	242344	242345	42679	341976	1971.70	17.40	3.06	0.63	997.97	577.37	9.28	1.00	1471.15
30	SPHSAH+PER(atc)	293285	293286	62804	395893	11.07	48.59	8.36	2.18	73.71	7.96	9.28	0.58	11.13
31	OSAH+SPH(atc)	710837	710838	75291	792147	4.87	65.77	9.98	5.23	39.34	8.74	7.86	0.49	9.98
32	SPHSAH+SPH(atc)	293285	293286	62804	395893	10.69	48.21	8.29	2.18	73.69	8.15	7.86	0.58	8.78
33	OSAH+TA$_{seq}$(16,2)	64529	64530	4966	216164	19.36	93.30	5.32	0.07	16.87	50.22	8.73	0.49	33.12
34	OSAH+TA$^{A}_{rec}$(16,2)	64529	64530	4966	216164	18.94	36.85	5.22	0.07	16.65	44.15	8.73	0.58	28.06
35	OSAH+TA$^{B}_{rec}$(16,2)	64529	64530	4966	216164	18.94	36.85	5.22	0.07	16.70	41.24	8.73	0.63	25.63
36	OSAH+TA$_{SNL}$(16,2)	64529	64530	4966	216164	19.36	18.93	5.32	0.07	18.04	41.91	8.73	0.62	26.19
37	OSAH+TA$_{NLT}$(16,2)	148943	64530	4966	216164	19.36	18.37	5.32	0.07	18.19	41.75	8.73	0.62	26.06
38	OSAH+TA$_{seq}$(18,2)	142744	142745	22200	277145	12.57	105.44	5.75	0.71	18.68	43.02	8.73	0.35	27.12
39	OSAH+TA$^{A}_{rec}$(18,2)	142744	142745	22200	277145	12.29	41.23	5.64	0.70	18.42	37.49	8.73	0.42	22.51
40	OSAH+TA$^{B}_{rec}$(18,2)	142744	142745	22200	277145	12.29	41.23	5.64	0.70	18.54	33.77	8.73	0.49	19.41
41	OSAH+TA$_{SNL}$(18,2)	142744	142745	22200	277145	12.57	23.12	5.75	0.71	21.43	33.14	8.73	0.50	18.88
42	OSAH+TA$_{NLT}$(18,2)	519230	142745	22200	277145	12.57	22.33	5.75	0.71	22.37	35.86	8.73	0.45	21.15
43	OSAH+TA$_{seq}$(atc)	721357	721358	75310	802648	5.18	154.48	7.61	3.48	33.48	60.08	8.73	0.09	41.33
44	OSAH+TA$^{A}_{rec}$(atc)	721357	721358	75310	802648	5.00	54.68	7.49	3.48	28.85	35.35	8.73	0.18	20.73
45	OSAH+TA$^{B}_{rec}$(atc)	721357	721358	75310	802648	5.00	54.68	7.49	3.48	29.30	30.09	8.73	0.23	16.34
46	OSAH+TA$_{SNL}$(atc)	721357	721358	75310	802648	5.18	32.33	7.61	3.48	119.94	44.79	8.73	0.13	28.59
47	OSAH+TA$_{NLT}$(atc)	-	-	-	-	-	-	-	-	-	-	-	-	-
48	BVH	11839	69959	0	105300	490.70	758.26	583.97	0.00	294.53	13803.80	–	–	–
49	O84	37449	262144	26944	561600	20.31	51.66	7.38	0.29	4.82	71.58	–	–	–
50	O89	37449	262144	26944	561600	20.21	32.17	7.32	0.29	4.51	61.47	–	–	–
51	BSP	65535	65536	0	330512	28.28	35.71	5.03	0.00	5.51	79.71	–	–	–
52	O93	37449	262144	26944	561600	17.34	29.71	11.87	5.67	4.44	72.89	–	–	–
53	UG	0	531441	138248	749107	14.09	9.23	9.23	2.81	8.54	43.65	–	–	–
54	AG	0	0	0	0	0.00	0.00	0.00	0.00	0.00	1.00	–	–	–
55	HUG	1	166375	0	1596184	78.06	7.37	6.37	0.00	5.53	157.03	–	–	–
56	RG	1	106032	0	1602704	84.67	5.87	4.87	0.00	4.23	140.24	–	–	–
57	O84A	37449	262144	14408	507656	17.90	50.76	7.25	0.07	18.48	66.32	–	–	–
58	KD	64531	64532	4969	216163	19.62	36.86	5.29	0.06	24.42	71.33	–	–	–

Table 24: Experimental results for scene “*lattice29*”.

$$N = 105300,$$
$$TP_D\!: \quad N_{prim} = 263169, \quad N^{hit}_{\mathcal{AB}} = 263169, \quad N^{hit}_{prim} = 262875, \quad N_{sec} = 253547, \quad N^{hit}_{sec} = 193215,$$
$$N_{shad} = 1143451, \quad N^{hit}_{shad} = 927614, \quad T^{MIN}_{R}[s] = 11.68, \quad T_{app}[s] = 10.48, \quad T^{MIN}_{RSA}[s] = 1.2.$$

Scene = “*mount8*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	131076	–	0	0	0	0.29	106528.00	16.94	1.0	343638.71
1	spatmed-xyz(16,2)	7121	7122	1895	227180	58.19	22.19	4.23	1.05	3.76	27.68	16.94	0.72	72.35
2	objmed-xyz(16,2)	62451	62452	2156	415452	222.20	83.55	16.01	0.27	17.34	99.02	16.94	0.73	302.48
3	objmed(16,2)	65535	65536	0	136619	324.91	420.70	69.76	0.00	24.44	282.94	16.94	0.44	895.77
4	OSAH(16,2)	13321	13322	3247	153865	18.63	18.90	3.44	1.23	24.31	16.21	16.94	0.50	35.35
5	OSAH-RMI(16,2)	13321	13322	3247	153865	18.63	18.90	3.44	1.23	24.39	15.99	16.94	0.50	34.65
6	OSAH-xyz(16,2)	14912	14913	3443	180978	22.80	20.65	3.60	1.55	16.02	17.33	16.94	0.52	38.97
7	OSAH(8,1)	161	162	30	134845	921.70	13.14	2.60	0.64	18.57	458.56	16.94	0.99	1462.29
8	OSAH(8,2)	161	162	30	134845	921.70	13.14	2.60	0.64	18.53	459.26	16.94	0.99	1464.55
9	OSAH(16,1)	13766	13767	3595	153961	18.17	19.78	3.54	1.24	24.39	16.64	16.94	0.48	36.74
10	OSAH(16,2)	13321	13322	3247	153865	18.63	18.90	3.44	1.23	24.31	16.21	16.94	0.50	35.35
11	OSAH(24,1)	253236	253237	112343	217276	5.42	22.63	4.00	1.66	30.18	12.92	16.94	0.19	24.74
12	OSAH(24,2)	106592	106593	22695	160289	6.37	20.84	3.76	1.44	27.36	13.09	16.94	0.23	25.29
13	OSAH(atc)	199904	199905	94141	182738	6.11	20.84	3.77	1.63	29.01	12.90	16.94	0.23	24.68
14	OSAH2(atc)	268007	268008	101388	262321	5.83	25.69	4.14	1.86	41.45	13.44	16.94	0.19	26.42
15	OSAH+LC(atc)	199934	199935	94157	182770	5.63	21.79	3.88	1.63	29.15	12.85	16.94	0.21	24.52
16	OSAH+TPC(atc)	199330	199331	93448	183062	6.30	20.20	3.67	1.63	30.54	12.58	16.94	0.24	23.65
17	OSAH+TPC+LC(atc)	199368	199369	93465	183094	5.80	21.31	3.79	1.63	34.64	11.98	16.94	0.21	21.71
18	OSAH+LC(16,1)	13776	13777	3605	153961	18.17	19.78	3.54	1.24	28.81	15.19	16.94	0.48	32.06
19	OSAH+TPC(16,1)	13859	13860	3652	154140	18.19	19.83	3.54	1.24	28.81	15.22	16.94	0.48	32.16
20	OSAH+TPC+LC(16,1)	13868	13869	3661	154140	18.18	19.83	3.54	1.25	28.84	15.54	16.94	0.48	33.19
21	OSAH+PR(atc)	199904	199905	94141	164729	6.08	20.98	3.81	1.66	31.55	12.96	16.94	0.22	24.87
22	OSAH+SC(atc)	216996	216997	104728	190004	5.53	21.02	3.74	1.76	71.46	12.68	16.94	0.21	23.97
23	OSAH+GCM(atc)	242670	242671	111817	207413	6.07	22.29	3.69	1.86	767.14	12.85	16.94	0.21	24.52
24	OSAH+GCM2(atc)	340273	340274	106521	379434	5.93	23.74	4.13	1.54	1145.86	13.11	16.94	0.20	25.35
25	OSAH+GCM3(atc)	240175	240176	87303	267534	6.16	26.23	4.32	1.84	792.90	14.34	16.94	0.19	29.32
26	OSAH+PAR(atc)	199904	199905	94141	182738	4.46	29.63	5.50	4.50	34.27	2.38	12.33	0.14	27.33
27	PARSAH+PAR(atc)	199293	199294	94879	180812	4.42	29.84	5.55	4.56	36.79	2.37	12.33	0.13	27.17
28	OSAH+PER(atc)	199904	199905	94141	182738	4.43	30.07	5.83	4.04	34.35	2.75	17.83	0.15	28.00
29	PERSAH+PER(atc)	189614	189615	89494	178835	5.14	29.23	5.92	4.21	1086.66	2.82	17.83	0.21	29.17
30	SPHSAH+PER(atc)	199904	199905	94141	182738	4.43	30.07	5.83	4.04	43.82	2.74	17.83	0.15	27.83
31	OSAH+SPH(atc)	199904	199905	94141	182738	4.42	29.54	5.74	3.98	34.35	3.04	15.22	0.16	18.56
32	SPHSAH+SPH(atc)	199904	199905	94141	182738	4.42	29.54	5.74	3.98	43.42	3.03	15.22	0.16	18.44
33	OSAH+TA$_{seq}$(16,2)	13321	13322	3247	153865	20.04	34.36	3.44	1.23	20.29	18.18	16.94	0.38	41.71
34	OSAH+TA$^{A}_{rec}$(16,2)	13321	13322	3247	153865	18.64	18.90	3.44	1.23	20.23	16.03	16.94	0.45	34.77
35	OSAH+TA$^{B}_{rec}$(16,2)	13321	13322	3247	153865	18.63	18.90	3.44	1.23	20.24	15.04	16.94	0.50	31.58
36	OSAH+TA$_{SNL}$(16,2)	13321	13322	3247	153865	20.05	12.68	3.44	1.23	20.45	14.32	16.94	0.54	29.26
37	OSAH+TA$_{NLT}$(16,2)	83081	13322	3247	153865	20.05	11.42	3.44	1.23	20.49	13.95	16.94	0.56	28.06
38	OSAH+TA$_{seq}$(18,2)	35967	35968	9251	157995	10.90	37.85	3.58	1.34	21.12	16.01	16.94	0.25	34.71
39	OSAH+TA$^{A}_{rec}$(18,2)	35967	35968	9251	157995	10.13	19.77	3.58	1.34	21.12	14.10	16.94	0.30	28.55
40	OSAH+TA$^{B}_{rec}$(18,2)	35967	35968	9251	157995	10.13	19.77	3.58	1.34	21.20	13.03	16.94	0.34	25.10
41	OSAH+TA$_{SNL}$(18,2)	35967	35968	9251	157995	10.90	13.38	3.58	1.34	21.79	12.71	16.94	0.35	24.06
42	OSAH+TA$_{NLT}$(18,2)	202751	35968	9251	157995	10.90	11.98	3.58	1.34	21.93	11.81	16.94	0.40	21.16
43	OSAH+TA$_{seq}$(atc)	199904	199905	94141	182738	6.58	43.72	3.77	1.63	24.08	15.30	16.94	0.15	32.42
44	OSAH+TA$^{A}_{rec}$(atc)	199904	199905	94141	182738	6.11	20.84	3.77	1.63	23.75	13.06	16.94	0.20	25.19
45	OSAH+TA$^{B}_{rec}$(atc)	199904	199905	94141	182738	6.11	20.84	3.77	1.63	23.81	12.00	16.94	0.23	21.77
46	OSAH+TA$_{SNL}$(atc)	199904	199905	94141	182738	6.58	14.13	3.77	1.63	28.16	10.80	16.94	0.28	17.90
47	OSAH+TA$_{NLT}$(atc)	866851	199905	94141	182738	6.58	12.60	3.77	1.63	30.10	16.60	16.94	0.14	36.61
48	BVH	19371	88075	0	131076	531.51	713.61	500.34	0.00	189.59	10436.00	–	–	–
49	O84	3450	24151	11599	301800	34.28	26.68	5.19	1.60	4.36	35.60	–	–	–
50	O89	3450	24151	11599	301800	34.35	19.53	5.19	1.60	4.36	31.82	–	–	–
51	BSP	7121	7122	1895	227180	58.19	22.19	4.23	1.05	4.69	53.49	–	–	–
52	O93	3450	24151	11599	301809	32.41	19.37	8.02	4.57	4.20	36.61	–	–	–
53	UG	0	649522	614359	393479	37.48	38.27	38.27	27.56	6.37	37.94	–	–	–
54	AG	0	0	0	0	0.00	0.00	0.00	0.00	0.00	1.00	–	–	–
55	HUG	1	2925	2328	220930	407.04	7.81	6.82	3.62	2.59	186.87	–	–	–
56	RG	1	129792	116818	685163	97.90	23.23	22.24	15.09	1.87	56.14	–	–	–
57	O84A	8048	56337	20760	493257	22.09	28.71	5.63	2.54	21.19	30.69	–	–	–
58	KD	13321	13322	3247	153865	18.63	18.90	3.44	1.23	25.82	25.14	–	–	–

Table 25: Experimental results for scene “*mount8*”.

$$N = 131076,$$
$$TP_D:\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 256915,\quad N^{hit}_{prim} = 145240,\quad N_{sec} = 707764,\quad N^{hit}_{sec} = 461009,$$
$$N_{shad} = 290405,\quad N^{hit}_{shad} = 20438,\quad T^{MIN}_{R}[s] = 5.56,\quad T_{app}[s] = 5.25,\quad T^{MIN}_{RSA}[s] = 0.31.$$

Scene = “*rings17*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	107101	–	0	0	0	0.23	231049.00	5.58	1.0	132028.00
1	spatmed-xyz(16,2)	18516	18517	1248	306974	101.10	50.36	9.12	3.85	3.63	145.11	5.58	0.87	77.34
2	objmed-xyz(16,2)	65529	65530	323	488212	230.62	89.41	17.19	0.21	16.10	355.92	5.58	0.89	197.81
3	objmed(16,2)	65521	65522	34	408570	166.27	81.06	15.94	0.01	23.69	260.66	5.58	0.87	143.37
4	OSAH(16,2)	27155	27156	4828	201209	39.12	40.25	6.88	3.70	21.23	69.33	5.58	0.76	34.04
5	OSAH-RMI(16,2)	27155	27156	4828	201209	39.12	40.25	6.88	3.70	21.22	69.94	5.58	0.76	34.39
6	OSAH-xyz(16,2)	30217	30218	2798	216262	41.21	44.34	7.84	4.25	13.68	74.08	5.58	0.75	36.75
7	OSAH(8,1)	214	215	34	114951	1460.10	15.82	3.34	0.75	15.32	1795.05	5.58	1.00	1020.17
8	OSAH(8,2)	214	215	34	114951	1460.10	15.82	3.34	0.75	15.30	1813.31	5.58	1.00	1030.60
9	OSAH(16,1)	27282	27283	4918	201228	39.09	40.28	6.89	3.71	21.19	69.67	5.58	0.76	34.23
10	OSAH(16,2)	27155	27156	4828	201209	39.12	40.25	6.88	3.70	21.23	69.33	5.58	0.76	34.04
11	OSAH(24,1)	1104245	1104246	174514	1703735	16.55	66.00	11.35	4.97	56.93	70.27	5.58	0.45	34.58
12	OSAH(24,2)	671174	671175	60011	1389504	17.19	57.92	9.65	4.18	43.38	54.19	5.58	0.49	25.39
13	OSAH(atc)	304937	304938	54285	615672	17.93	54.64	9.33	4.55	33.04	52.27	5.58	0.51	24.29
14	OSAH2(atc)	782981	782982	92071	1347114	18.03	70.28	11.73	5.05	59.81	59.37	5.58	0.45	28.35
15	OSAH+LC(atc)	443984	443985	54291	1052788	17.75	57.85	9.77	4.55	39.47	53.85	5.58	0.50	25.19
16	OSAH+TPC(atc)	185686	185687	32735	448858	19.85	51.82	8.85	4.36	30.25	53.69	5.58	0.55	25.10
17	OSAH+TPC+LC(atc)	415491	415492	32780	1166698	19.64	57.12	9.60	4.36	46.30	55.14	5.58	0.52	25.93
18	OSAH+LC(16,1)	27287	27288	4931	201228	39.04	40.28	6.89	3.71	24.18	68.46	5.58	0.76	33.54
19	OSAH+TPC(16,1)	27282	27283	4917	201262	39.09	40.27	6.89	3.71	24.78	68.41	5.58	0.76	33.51
20	OSAH+TPC+LC(16,1)	27287	27288	4922	201262	39.09	40.28	6.89	3.71	24.80	68.44	5.58	0.76	33.53
21	OSAH+PR(atc)	304937	304938	54285	415765	14.86	55.05	9.42	4.56	53.26	48.43	5.58	0.46	22.10
22	OSAH+SC(atc)	346502	346503	103509	438117	10.48	55.97	9.51	5.91	87.78	43.00	5.58	0.38	18.99
23	OSAH+GCM(atc)	723135	723136	140521	1199435	16.00	63.80	10.80	5.09	2182.55	56.09	5.58	0.45	26.47
24	OSAH+GCM2(atc)	2066109	2066110	54421	4462230	22.36	79.27	13.23	4.70	7393.55	104.61	5.58	0.47	54.20
25	OSAH+GCM3(atc)	759588	759589	99396	1425964	902.25	58.92	7.66	3.69	2492.78	1173.18	5.58	0.98	664.81
26	OSAH+PAR(atc)	304937	304938	54285	615672	19.37	77.76	14.82	6.25	37.97	13.36	5.41	0.66	17.24
27	PARSAH+PAR(atc)	278251	278252	32148	4029917	36.45	19.34	1.19	0.06	92.79	20.58	5.41	0.95	29.47
28	OSAH+PER(atc)	304937	304938	54285	615672	11.62	71.19	13.84	8.41	37.99	10.14	5.26	0.55	12.22
29	PERSAH+PER(atc)	165924	165925	46239	416731	312.51	46.13	9.29	5.64	3801.09	82.71	5.26	0.97	137.34
30	SPHSAH+PER(atc)	480358	480359	96766	1472702	64.70	111.80	21.64	14.35	256.53	26.95	5.26	0.79	41.21
31	OSAH+SPH(atc)	304937	304938	54285	615672	11.61	70.46	13.73	8.41	37.87	10.29	5.37	0.55	11.23
32	SPHSAH+SPH(atc)	480358	480359	96766	1472702	68.72	110.89	21.51	14.35	256.39	27.80	5.37	0.80	39.47
33	OSAH+TA$_{seq}$(16,2)	27154	27155	4835	201209	40.78	102.45	6.99	3.70	17.36	83.38	5.58	0.60	42.07
34	OSAH+TA$^{A}_{rec}$(16,2)	27154	27155	4835	201209	39.07	40.24	6.88	3.70	17.28	72.27	5.58	0.70	35.72
35	OSAH+TA$^{B}_{rec}$(16,2)	27154	27155	4835	201209	39.07	40.24	6.88	3.70	17.31	67.71	5.58	0.76	33.11
36	OSAH+TA$_{SNL}$(16,2)	27154	27155	4835	201209	40.78	32.46	6.99	3.70	17.77	109.78	5.58	0.44	57.15
37	OSAH+TA$_{NLT}$(16,2)	145233	27155	4835	201209	40.78	29.27	6.99	3.70	17.99	76.33	5.58	0.66	38.04
38	OSAH+TA$_{seq}$(18,2)	78599	78600	13421	297456	25.77	129.29	8.10	3.93	19.82	72.33	5.58	0.46	35.75
39	OSAH+TA$^{A}_{rec}$(18,2)	78599	78600	13421	297456	24.51	46.65	7.94	3.93	19.79	60.60	5.58	0.56	29.05
40	OSAH+TA$^{B}_{rec}$(18,2)	78599	78600	13421	297456	24.51	46.65	7.94	3.93	19.77	55.12	5.58	0.63	25.92
41	OSAH+TA$_{SNL}$(18,2)	78599	78600	13421	297456	25.77	37.61	8.10	3.93	21.28	61.06	5.58	0.56	29.31
42	OSAH+TA$_{NLT}$(18,2)	466434	78600	13421	297456	25.77	33.57	8.10	3.93	21.69	61.02	5.58	0.56	29.29
43	OSAH+TA$_{seq}$(atc)	304350	304351	54206	614753	18.98	167.09	9.55	4.56	27.75	72.06	5.58	0.33	35.60
44	OSAH+TA$^{A}_{rec}$(atc)	304350	304351	54206	614753	17.94	54.63	9.33	4.56	27.33	56.91	5.58	0.44	26.94
45	OSAH+TA$^{B}_{rec}$(atc)	304350	304351	54206	614753	17.94	54.63	9.33	4.56	27.46	50.63	5.58	0.51	23.35
46	OSAH+TA$_{SNL}$(atc)	304350	304351	54206	614753	18.98	44.46	9.55	4.56	33.48	56.42	5.58	0.45	26.66
47	OSAH+TA$_{NLT}$(atc)	1897332	304351	54206	614753	18.98	39.10	9.55	4.56	48.70	87.07	5.58	0.27	44.18
48	BVH	13107	70607	0	107101	738.71	912.51	664.19	0.00	145.68	18030.50	–	–	–
49	O84	10123	70862	7846	493503	59.94	83.55	12.77	6.77	4.90	168.69	–	–	–
50	O89	10123	70862	7846	493503	60.08	50.66	12.77	6.77	4.45	149.94	–	–	–
51	BSP	18516	18517	1248	306974	101.10	50.36	9.12	3.85	5.14	234.40	–	–	–
52	O93	10123	70862	7846	493503	59.86	46.03	18.14	12.15	4.42	168.63	–	–	–
53	UG	0	536726	417221	677753	48.27	32.10	32.10	25.76	7.51	116.81	–	–	–
54	AG	44528	757129	159833	1625581	91.19	32.03	23.39	0.44	45.51	371.98	–	–	–
55	HUG	24356	174237	30198	791828	71.18	19.37	13.62	9.20	6.99	170.66	–	–	–
56	RG	26722	478957	112574	1526782	54.53	26.07	22.17	14.42	4.95	144.67	–	–	–
57	O84A	19177	134240	17096	573705	39.03	71.95	11.32	6.03	20.09	128.81	–	–	–
58	KD	27155	27156	4828	201209	39.12	40.25	6.88	3.70	23.55	106.61	–	–	–

Table 26: Experimental results for scene “*rings17*”.

$$N = 107101,$$
$$TP_D\text{:}\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 263169,\quad N^{hit}_{prim} = 263168,\quad N_{sec} = 386859,\quad N^{hit}_{sec} = 220131,$$
$$N_{shad} = 1147577,\quad N^{hit}_{shad} = 609678,\quad T^{MIN}_{R}[s] = 11.55,\quad T_{app}[s] = 9.76,\quad T^{MIN}_{RSA}[s] = 1.75.$$

Scene = “*sombrero4*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	130050	–	0	0	0	0.27	39868.90	11.70	1.0	398689.00
1	spatmed-xyz(16,2)	8904	8905	2759	230502	217.75	26.97	4.98	3.23	4.17	11.73	11.70	0.86	105.60
2	objmed-xyz(16,2)	63668	63669	1280	390996	315.83	84.09	15.17	0.49	19.55	19.51	11.70	0.73	183.40
3	objmed(16,2)	65243	65244	0	130696	190.33	159.75	29.25	0.00	21.77	21.08	11.70	0.47	199.10
4	OSAH(16,2)	18329	18330	2966	135207	41.70	19.17	3.28	1.81	24.98	4.23	11.70	0.61	30.60
5	OSAH-RMI(16,2)	18329	18330	2966	135207	41.70	19.17	3.28	1.81	24.95	4.14	11.70	0.61	29.70
6	OSAH-xyz(16,2)	14525	14526	2594	168265	101.52	27.41	5.07	2.68	17.52	6.92	11.70	0.73	57.50
7	OSAH(8,1)	203	204	72	131812	3645.70	10.12	2.18	1.10	17.46	268.58	11.70	1.00	2674.10
8	OSAH(8,2)	200	201	69	131812	3645.80	10.10	2.17	1.08	17.59	268.46	11.70	1.00	2672.90
9	OSAH(16,1)	19544	19545	4113	135263	41.29	19.32	3.31	1.90	24.93	4.39	11.70	0.61	32.20
10	OSAH(16,2)	18329	18330	2966	135207	41.70	19.17	3.28	1.81	24.98	4.23	11.70	0.61	30.60
11	OSAH(24,1)	259695	259696	127424	198890	7.14	26.44	4.66	3.20	30.20	3.37	11.70	0.17	22.00
12	OSAH(24,2)	85618	85619	16730	135619	9.21	23.02	3.84	2.27	27.00	3.23	11.70	0.23	20.60
13	OSAH(atc)	232128	232129	127080	171667	6.85	26.02	4.53	3.20	29.58	3.27	11.70	0.16	21.00
14	OSAH2(atc)	250211	250212	122973	198040	8.82	38.33	5.85	3.92	41.42	3.75	11.70	0.14	25.80
15	OSAH+LC(atc)	232130	232131	127082	171667	6.82	26.02	4.53	3.21	29.72	3.27	11.70	0.16	21.00
16	OSAH+TPC(atc)	232027	232028	126996	171663	6.85	26.01	4.53	3.20	31.38	3.26	11.70	0.16	20.90
17	OSAH+TPC+LC(atc)	232033	232034	126998	171667	6.82	26.02	4.53	3.21	34.89	3.03	11.70	0.16	18.60
18	OSAH+LC(16,1)	19546	19547	4115	135263	41.26	19.32	3.31	1.91	29.21	3.92	11.70	0.61	27.50
19	OSAH+TPC(16,1)	19566	19567	4132	135303	41.27	19.32	3.31	1.91	29.39	3.93	11.70	0.61	27.60
20	OSAH+TPC+LC(16,1)	19568	19569	4134	135303	41.25	19.33	3.31	1.92	29.37	3.93	11.70	0.61	27.60
21	OSAH+PR(atc)	232128	232129	127080	161899	6.29	26.02	4.53	3.20	32.20	3.25	11.70	0.15	20.80
22	OSAH+SC(atc)	240183	240184	137075	169784	6.24	26.04	4.53	3.36	41.31	3.09	11.70	0.15	19.20
23	OSAH+GCM(atc)	260242	260243	128272	201278	7.12	29.27	4.97	3.53	797.48	3.23	11.70	0.15	20.60
24	OSAH+GCM2(atc)	241938	241939	87307	242076	8.52	31.28	5.15	3.42	850.17	3.40	11.70	0.17	22.30
25	OSAH+GCM3(atc)	217347	217348	97756	203990	10.57	34.68	5.73	3.92	713.35	3.65	11.70	0.18	24.80
26	OSAH+PAR(atc)	232128	232129	127080	171667	3.08	18.97	3.26	2.40	35.24	2.06	8.10	0.42	12.50
27	PARSAH+PAR(atc)	235137	235138	125205	176143	2.88	15.51	2.71	1.96	33.27	2.08	8.10	0.53	12.70
28	OSAH+PER(atc)	232128	232129	127080	171667	3.24	18.16	3.12	2.25	35.23	2.07	7.55	0.44	11.27
29	PERSAH+PER(atc)	165968	165969	90071	160178	2.96	17.06	3.01	2.25	846.63	2.13	7.55	0.50	11.82
30	SPHSAH+PER(atc)	196334	196335	105204	166309	24.95	19.82	3.50	2.75	65.01	3.54	7.55	0.72	24.64
31	OSAH+SPH(atc)	232128	232129	127080	171667	3.24	18.25	3.15	2.27	35.15	2.46	7.67	0.47	8.73
32	SPHSAH+SPH(atc)	196334	196335	105204	166309	24.88	19.92	3.52	2.76	64.97	3.94	7.67	0.73	18.60
33	OSAH+TA$_{seq}$(16,2)	18329	18330	2966	135207	41.72	45.29	3.28	1.81	20.35	5.00	11.70	0.44	38.30
34	OSAH+TA$^{A}_{rec}$(16,2)	18329	18330	2966	135207	41.70	19.17	3.28	1.81	20.30	4.32	11.70	0.53	31.50
35	OSAH+TA$^{B}_{rec}$(16,2)	18329	18330	2966	135207	41.70	19.17	3.28	1.81	20.34	3.91	11.70	0.61	27.40
36	OSAH+TA$_{SNL}$(16,2)	18329	18330	2966	135207	41.72	17.74	3.28	1.81	20.63	3.93	11.70	0.61	27.60
37	OSAH+TA$_{NLT}$(16,2)	87481	18330	2966	135207	41.72	16.39	3.28	1.81	20.74	3.94	11.70	0.60	27.70
38	OSAH+TA$_{seq}$(18,2)	50412	50413	13578	135600	16.83	55.23	3.68	2.21	21.29	4.43	11.70	0.22	32.60
39	OSAH+TA$^{A}_{rec}$(18,2)	50412	50413	13578	135600	16.83	21.69	3.68	2.21	21.27	3.57	11.70	0.29	24.00
40	OSAH+TA$^{B}_{rec}$(18,2)	50412	50413	13578	135600	16.83	21.69	3.68	2.21	21.20	3.13	11.70	0.36	19.60
41	OSAH+TA$_{SNL}$(18,2)	50412	50413	13578	135600	16.84	19.80	3.68	2.21	22.09	3.15	11.70	0.36	19.80
42	OSAH+TA$_{NLT}$(18,2)	238564	50413	13578	135600	16.84	18.17	3.68	2.21	22.27	3.16	11.70	0.35	19.90
43	OSAH+TA$_{seq}$(atc)	232128	232129	127080	171667	6.85	77.55	4.53	3.20	24.19	4.80	11.70	0.08	36.30
44	OSAH+TA$^{A}_{rec}$(atc)	232128	232129	127080	171667	6.85	26.02	4.53	3.20	23.98	3.52	11.70	0.12	23.50
45	OSAH+TA$^{B}_{rec}$(atc)	232128	232129	127080	171667	6.85	26.02	4.53	3.20	24.14	3.00	11.70	0.16	18.30
46	OSAH+TA$_{SNL}$(atc)	232128	232129	127080	171667	6.85	23.13	4.53	3.20	28.65	3.01	11.70	0.16	18.40
47	OSAH+TA$_{NLT}$(atc)	981401	232129	127080	171667	6.85	20.94	4.53	3.20	31.82	7.22	11.70	0.05	60.50
48	BVH	20050	87858	0	130050	1048.30	906.27	614.35	0.00	172.53	3324.59	–	–	–
49	O84	4422	30955	15168	305664	134.48	41.07	7.23	5.12	4.63	14.40	–	–	–
50	O89	4422	30955	15168	305664	133.70	27.87	7.22	5.12	4.27	12.91	–	–	–
51	BSP	8904	8905	2759	230502	217.75	26.97	4.98	3.23	4.65	19.08	–	–	–
52	O93	4422	30955	15168	305664	125.40	24.02	8.85	6.83	4.38	13.74	–	–	–
53	UG	0	643860	606223	396633	58.06	28.11	28.11	26.43	6.45	7.08	–	–	–
54	AG	41042	1275612	386764	2950955	42.64	22.34	19.21	15.43	205.66	23.11	–	–	–
55	HUG	1	1936	1332	173298	1202.80	4.95	4.29	3.00	2.87	87.69	–	–	–
56	RG	11610	491228	264152	2172750	74.94	21.03	18.83	16.17	5.62	9.58	–	–	–
57	O84A	8706	60943	20848	461512	105.39	46.40	7.93	5.19	22.41	12.96	–	–	–
58	KD	18329	18330	2966	135207	41.70	19.17	3.28	1.81	27.63	6.90	–	–	–

Table 27: Experimental results for scene “*sombrero4*”.

$$N = 130050,$$
$$TP_D\text{:}\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 136638,\quad N^{hit}_{prim} = 112239,\quad N_{sec} = 0,\quad N^{hit}_{sec} = 0,$$
$$N_{shad} = 110608,\quad N^{hit}_{shad} = 2622,\quad T^{MIN}_{R}[s] = 1.27,\quad T_{app}[s] = 1.17,\quad T^{MIN}_{RSA}[s] = 0.10.$$

Scene = “*teapot40*”

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	103680	–	0	0	0	0.22	67216.90	17.32	1.0	305531.36
1	spatmed-xyz(16,2)	8598	8599	2943	180581	251.00	40.45	8.61	5.97	3.24	35.97	17.32	0.80	146.18
2	objmed-xyz(16,2)	63340	63341	2356	371822	670.84	142.05	29.53	0.88	12.99	110.67	17.32	0.75	485.73
3	objmed(16,2)	65496	65497	798	265120	1372.80	486.86	98.65	1.15	18.40	318.57	17.32	0.64	1430.73
4	OSAH(16,2)	18593	18594	3495	181085	57.40	30.48	5.58	3.41	19.38	14.80	17.32	0.54	49.95
5	OSAH-RMI(16,2)	18593	18594	3495	181085	57.40	30.48	5.58	3.41	19.38	14.68	17.32	0.54	49.41
6	OSAH-xyz(16,2)	18737	18738	4197	185251	67.47	36.55	7.20	4.79	12.10	16.06	17.32	0.54	55.68
7	OSAH(8,1)	225	226	30	112307	2803.00	16.73	3.68	1.89	13.97	454.21	17.32	0.99	2047.27
8	OSAH(8,2)	225	226	30	112307	2803.00	16.73	3.68	1.89	14.01	455.98	17.32	0.99	2055.32
9	OSAH(16,1)	19630	19631	3883	181133	55.42	31.04	5.61	3.54	19.37	14.92	17.32	0.53	50.50
10	OSAH(16,2)	18593	18594	3495	181085	57.40	30.48	5.58	3.41	19.38	14.80	17.32	0.54	49.95
11	OSAH(24,1)	425323	425324	117627	638251	11.87	39.24	6.96	4.49	31.41	11.70	17.32	0.16	35.86
12	OSAH(24,2)	290100	290101	49364	574014	15.55	36.91	6.63	3.97	28.58	11.77	17.32	0.21	36.18
13	OSAH(atc)	264722	264723	75531	435017	12.77	38.29	6.79	4.42	27.20	11.31	17.32	0.17	34.09
14	OSAH2(atc)	456849	456850	111778	707674	12.97	52.13	8.42	5.75	43.85	13.28	17.32	0.13	43.05
15	OSAH+LC(atc)	266473	266474	75670	443763	12.73	38.32	6.79	4.42	27.59	11.33	17.32	0.17	34.18
16	OSAH+TPC(atc)	247373	247374	70794	418298	13.27	37.94	6.73	4.38	28.43	11.33	17.32	0.18	34.18
17	OSAH+TPC+LC(atc)	257162	257163	71050	446531	13.08	38.12	6.76	4.38	32.72	11.05	17.32	0.18	32.91
18	OSAH+LC(16,1)	19652	19653	3905	181133	55.42	31.04	5.61	3.54	22.42	14.15	17.32	0.53	47.00
19	OSAH+TPC(16,1)	19765	19766	4021	181186	55.41	31.06	5.62	3.55	22.83	14.28	17.32	0.53	47.59
20	OSAH+TPC+LC(16,1)	19786	19787	4042	181186	55.40	31.06	5.62	3.55	22.83	14.13	17.32	0.53	46.91
21	OSAH+PR(atc)	264722	264723	75531	377180	12.77	37.35	6.61	4.36	32.04	9.88	17.32	0.16	27.59
22	OSAH+SC(atc)	264471	264472	106643	375965	10.17	37.64	6.63	4.83	92.72	11.21	17.32	0.14	33.64
23	OSAH+GCM(atc)	344056	344057	101428	523736	11.89	39.81	6.94	4.59	1138.43	11.50	17.32	0.16	34.95
24	OSAH+GCM2(atc)	616614	616615	119164	1234612	14.16	45.00	7.69	4.93	2144.94	12.74	17.32	0.16	40.59
25	OSAH+GCM3(atc)	194922	194923	43680	467933	19.98	50.58	8.01	5.18	701.48	13.77	17.32	0.20	45.27
26	OSAH+PAR(atc)	264722	264723	75531	435017	3.61	23.84	4.32	3.19	32.11	2.95	14.70	0.39	14.80
27	PARSAH+PAR(atc)	256679	256680	72330	428579	3.54	23.42	4.17	3.09	35.37	2.91	14.70	0.38	14.40
28	OSAH+PER(atc)	264722	264723	75531	435017	3.31	29.31	5.34	3.98	33.42	3.74	12.50	0.36	10.88
29	PERSAH+PER(atc)	233036	233037	68652	393621	3.28	27.60	5.11	3.80	1653.61	3.60	12.50	0.34	10.00
30	SPHSAH+PER(atc)	215290	215291	64843	386727	3.62	27.60	5.10	3.75	80.40	3.62	12.50	0.35	10.12
31	OSAH+SPH(atc)	264722	264723	75531	435017	3.32	28.73	5.23	3.90	32.14	3.95	16.14	0.35	12.07
32	SPHSAH+SPH(atc)	215290	215291	64843	386727	3.64	27.09	5.00	3.68	80.36	3.93	16.14	0.38	11.93
33	OSAH+TA$_{seq}$(16,2)	18593	18594	3495	181085	57.83	70.26	5.59	3.41	16.04	18.93	17.32	0.37	68.73
34	OSAH+TA$^A_{rec}$(16,2)	18593	18594	3495	181085	57.40	30.48	5.58	3.41	15.98	15.68	17.32	0.47	53.95
35	OSAH+TA$^B_{rec}$(16,2)	18593	18594	3495	181085	57.40	30.48	5.58	3.41	16.04	14.15	17.32	0.54	47.00
36	OSAH+TA$_{SNL}$(16,2)	18593	18594	3495	181085	57.84	23.81	5.59	3.41	16.47	13.95	17.32	0.55	46.09
37	OSAH+TA$_{NLT}$(16,2)	122836	18594	3495	181085	57.84	21.34	5.59	3.41	16.62	13.90	17.32	0.55	45.86
38	OSAH+TA$_{seq}$(18,2)	48421	48422	9840	227380	30.41	81.23	6.05	3.68	17.39	16.94	17.32	0.22	59.68
39	OSAH+TA$^A_{rec}$(18,2)	48421	48422	9840	227380	30.18	33.18	6.04	3.68	17.28	13.46	17.32	0.30	43.86
40	OSAH+TA$^B_{rec}$(18,2)	48421	48422	9840	227380	30.18	33.18	6.04	3.68	17.26	11.84	17.32	0.36	36.50
41	OSAH+TA$_{SNL}$(18,2)	48421	48422	9840	227380	30.42	25.88	6.05	3.68	18.56	11.61	17.32	0.37	35.45
42	OSAH+TA$_{NLT}$(18,2)	313831	48422	9840	227380	30.42	23.00	6.05	3.68	18.57	11.60	17.32	0.37	35.41
43	OSAH+TA$_{seq}$(atc)	264722	264723	75531	435017	12.88	101.70	6.80	4.42	22.64	16.94	17.32	0.09	59.68
44	OSAH+TA$^A_{rec}$(atc)	264722	264723	75531	435017	12.77	38.29	6.79	4.42	22.18	12.61	17.32	0.13	40.00
45	OSAH+TA$^B_{rec}$(atc)	264722	264723	75531	435017	12.77	38.29	6.79	4.42	22.37	10.76	17.32	0.17	31.59
46	OSAH+TA$_{SNL}$(atc)	264722	264723	75531	435017	12.89	29.39	6.80	4.42	27.79	10.94	17.32	0.17	32.41
47	OSAH+TA$_{NLT}$(atc)	1371881	264723	75531	435017	12.89	25.78	6.80	4.42	36.00	32.08	17.32	0.04	128.50
48	BVH	16404	69920	0	103680	1011.90	835.87	557.33	0.00	118.35	7116.55	–	–	–
49	O84	4066	28463	14364	231325	132.13	64.02	11.48	8.84	3.68	43.98	–	–	–
50	O89	4066	28463	14364	231325	132.05	43.01	11.48	8.84	3.41	37.49	–	–	–
51	BSP	8598	8599	2943	180581	251.00	40.45	8.61	5.97	4.27	93.80	–	–	–
52	O93	4066	28463	14364	231325	126.61	36.52	13.67	11.11	3.47	41.44	–	–	–
53	UG	0	515570	487803	289369	94.13	57.12	57.12	54.15	4.94	31.46	–	–	–
54	AG	20337	1006860	495117	2225562	64.22	45.94	42.08	36.38	287.18	41.59	–	–	–
55	HUG	19260	316436	264487	380083	77.49	42.19	37.10	34.02	6.51	42.48	–	–	–
56	RG	4820	352189	196005	2025201	92.30	37.92	35.69	31.86	4.43	31.62	–	–	–
57	O84A	8704	60929	21144	360490	64.62	58.95	10.98	8.43	17.01	33.27	–	–	–
58	KD	18593	18594	3495	181085	57.40	30.48	5.58	3.41	21.77	23.85	–	–	–

Table 28: Experimental results for scene “*teapot40*”.

$$N = 103680,$$
$$TP_D\text{:}\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 226226,\quad N^{hit}_{prim} = 161581,\quad N_{sec} = 225988,\quad N^{hit}_{sec} = 67302,$$
$$N_{shad} = 406161,\quad N^{hit}_{shad} = 33950,\quad T^{MIN}_{R}[s] = 4.03,\quad T_{app}[s] = 3.81,\quad T^{MIN}_{RSA}[s] = 0.22.$$

Scene = "*tetra8*"

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	65536	–	0	0	0	0.13	12855.80	28.67	1.0	428526.67
1	spatmed-xyz(16,2)	11651	11652	4588	151552	214.14	30.12	6.04	4.44	2.05	6.46	28.67	0.72	186.67
2	objmed-xyz(16,2)	42239	42240	25856	65536	42.04	75.87	14.84	13.06	6.54	4.61	28.67	0.17	125.00
3	objmed(16,2)	49151	49152	32768	65536	23.77	215.24	42.25	41.24	10.85	9.27	28.67	0.04	280.33
4	OSAH(16,2)	19097	19098	7026	65536	27.34	20.17	3.71	2.92	9.97	2.40	28.67	0.33	51.33
5	OSAH-RMI(16,2)	19097	19098	7026	65536	27.34	20.17	3.71	2.92	9.99	2.39	28.67	0.33	51.00
6	OSAH-xyz(16,2)	19932	19933	7865	65536	30.07	24.48	4.66	3.79	6.69	2.56	28.67	0.30	56.67
7	OSAH(8,1)	179	180	28	65536	2948.70	9.70	2.17	1.01	8.20	82.19	28.67	0.99	2711.00
8	OSAH(8,2)	179	180	28	65536	2948.70	9.70	2.17	1.01	8.18	83.23	28.67	0.99	2745.67
9	OSAH(16,1)	19097	19098	7026	65536	27.34	20.17	3.71	2.92	9.93	2.45	28.67	0.33	53.00
10	OSAH(16,2)	19097	19098	7026	65536	27.34	20.17	3.71	2.92	9.97	2.40	28.67	0.33	51.33
11	OSAH(24,1)	48264	48265	31881	65536	10.12	22.00	3.98	3.54	10.74	2.08	28.67	0.14	40.67
12	OSAH(24,2)	48264	48265	31881	65536	10.12	22.00	3.98	3.54	10.87	2.09	28.67	0.14	41.00
13	OSAH(atc)	48264	48265	31881	65536	10.12	22.00	3.98	3.54	10.82	2.08	28.67	0.14	40.67
14	OSAH2(atc)	44203	44204	27820	65536	10.12	34.13	4.74	4.30	13.73	2.32	28.67	0.10	48.67
15	OSAH+LC(atc)	48264	48265	31881	65536	10.12	22.00	3.98	3.54	10.73	2.10	28.67	0.14	41.33
16	OSAH+TPC(atc)	48252	48253	31869	65536	10.12	21.98	3.98	3.54	11.39	2.07	28.67	0.14	40.33
17	OSAH+TPC+LC(atc)	48252	48253	31869	65536	10.12	21.98	3.98	3.54	12.72	1.96	28.67	0.14	36.67
18	OSAH+LC(16,1)	19097	19098	7026	65536	27.34	20.17	3.71	2.92	11.85	2.28	28.67	0.33	47.33
19	OSAH+TPC(16,1)	19003	19004	7068	65536	27.51	20.14	3.70	2.92	11.90	2.28	28.67	0.33	47.33
20	OSAH+TPC+LC(16,1)	19003	19004	7068	65536	27.51	20.14	3.70	2.92	11.79	2.28	28.67	0.33	47.33
21	OSAH+PR(atc)	48264	48265	31881	65536	10.12	22.00	3.98	3.54	11.44	2.07	28.67	0.14	40.33
22	OSAH+SC(atc)	48264	48265	31881	65536	10.12	22.00	3.98	3.54	13.15	2.03	28.67	0.14	39.00
23	OSAH+GCM(atc)	51845	51846	35462	65536	10.12	26.10	4.71	4.28	176.69	2.13	28.67	0.12	42.33
24	OSAH+GCM2(atc)	51682	51683	35299	65536	10.12	25.84	4.50	4.06	180.27	2.13	28.67	0.12	42.33
25	OSAH+GCM3(atc)	48503	48504	33020	65536	40.06	44.63	7.06	6.40	173.04	3.28	28.67	0.24	80.67
26	OSAH+PAR(atc)	48264	48265	31881	65536	8.52	19.23	3.62	3.32	12.57	1.57	18.33	0.38	34.00
27	PARSAH+PAR(atc)	48232	48233	31849	65536	8.52	19.38	3.70	3.41	13.08	1.59	18.33	0.38	34.67
28	OSAH+PER(atc)	48264	48265	31881	65536	7.53	19.84	3.71	3.40	12.60	1.64	19.67	0.38	35.00
29	PERSAH+PER(atc)	51060	51061	34677	65536	7.53	23.54	4.43	4.12	154.96	1.81	19.67	0.36	40.67
30	SPHSAH+PER(atc)	35813	35814	22688	65536	73.92	28.15	4.98	4.38	17.97	3.04	19.67	0.62	81.67
31	OSAH+SPH(atc)	48264	48265	31881	65536	7.51	19.62	3.67	3.36	12.55	1.97	30.67	0.38	35.00
32	SPHSAH+SPH(atc)	35813	35814	22688	65536	72.43	27.79	4.92	4.33	17.95	3.31	30.67	0.62	79.67
33	OSAH+TA$_{seq}$(16,2)	19097	19098	7026	65536	27.34	50.29	3.71	2.92	8.39	3.38	28.67	0.18	84.00
34	OSAH+TA$^A_{rec}$(16,2)	19097	19098	7026	65536	27.34	20.17	3.71	2.92	8.34	2.59	28.67	0.27	57.67
35	OSAH+TA$^B_{rec}$(16,2)	19097	19098	7026	65536	27.34	20.17	3.71	2.92	8.46	2.26	28.67	0.33	46.67
36	OSAH+TA$_{SNL}$(16,2)	19097	19098	7026	65536	27.34	18.39	3.71	2.92	10.06	2.79	28.67	0.24	64.33
37	OSAH+TA$_{NLT}$(16,2)	85575	19098	7026	65536	27.34	16.36	3.71	2.92	8.71	2.40	28.67	0.30	51.33
38	OSAH+TA$_{seq}$(18,2)	41905	41906	25556	65536	11.81	56.11	3.94	3.43	8.86	3.22	28.67	0.08	78.67
39	OSAH+TA$^A_{rec}$(18,2)	41905	41906	25556	65536	11.81	21.71	3.94	3.43	8.80	2.36	28.67	0.12	50.00
40	OSAH+TA$^B_{rec}$(18,2)	41905	41906	25556	65536	11.81	21.71	3.94	3.43	8.76	1.99	28.67	0.16	37.67
41	OSAH+TA$_{SNL}$(18,2)	41905	41906	25556	65536	11.81	19.42	3.94	3.43	9.53	2.08	28.67	0.15	40.67
42	OSAH+TA$_{NLT}$(18,2)	151873	41906	25556	65536	11.81	17.21	3.94	3.43	9.69	2.10	28.67	0.15	41.33
43	OSAH+TA$_{seq}$(atc)	48264	48265	31881	65536	10.12	57.23	3.98	3.54	9.03	3.20	28.67	0.07	78.00
44	OSAH+TA$^A_{rec}$(atc)	48264	48265	31881	65536	10.12	22.00	3.98	3.54	8.96	2.34	28.67	0.10	49.33
45	OSAH+TA$^B_{rec}$(atc)	48264	48265	31881	65536	10.12	22.00	3.98	3.54	8.95	1.95	28.67	0.14	36.33
46	OSAH+TA$_{SNL}$(atc)	48264	48265	31881	65536	10.12	19.63	3.98	3.54	9.85	2.09	28.67	0.12	41.00
47	OSAH+TA$_{NLT}$(atc)	168703	48265	31881	65536	10.12	17.39	3.98	3.54	9.92	2.09	28.67	0.12	41.00
48	BVH	9339	43903	0	65536	1109.90	450.18	326.98	0.00	88.62	1418.19	–	–	–
49	O84	5697	39880	22496	212992	119.72	47.28	8.22	6.61	2.41	9.48	–	–	–
50	O89	5697	39880	22496	212992	119.29	31.85	8.22	6.61	2.20	7.33	–	–	–
51	BSP	11651	11652	4588	151552	214.14	30.12	6.04	4.44	2.51	9.86	–	–	–
52	O93	5697	39880	22496	212992	110.35	26.16	9.50	7.97	2.22	8.56	–	–	–
53	UG	0	328509	307819	232864	105.60	32.84	32.84	31.22	3.17	5.97	–	–	–
54	AG	16921	656550	376506	1455360	86.38	38.10	35.04	31.83	57.81	9.64	–	–	–
55	HUG	1	195112	180318	202552	136.68	28.28	27.63	25.92	4.09	7.42	–	–	–
56	RG	8281	386424	151012	3447984	253.12	25.12	22.48	18.98	6.37	11.63	–	–	–
57	O84A	7121	49848	25744	311008	134.85	44.46	7.86	6.39	10.78	9.40	–	–	–
58	KD	19097	19098	7026	65536	27.34	20.17	3.71	2.92	10.61	3.57	–	–	–

Table 29: Experimental results for scene "*tetra8*".

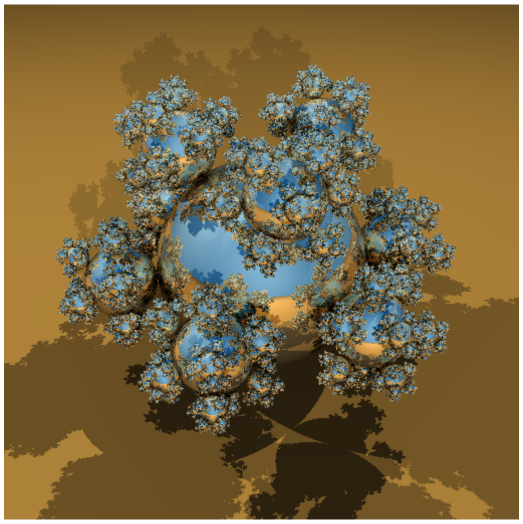
$$N = 65536,$$
$$TP_D\!: \quad N_{prim} = 263169, \quad N^{hit}_{\mathcal{AB}} = 159213, \quad N^{hit}_{prim} = 43709, \quad N_{sec} = 0, \quad N^{hit}_{sec} = 0,$$
$$N_{shad} = 40256, \quad N^{hit}_{shad} = 7098, \quad T^{MIN}_{R}[s] = 0.89, \quad T_{app}[s] = 0.86, \quad T^{MIN}_{RSA}[s] = 0.03.$$

Scene = “*tree15*”

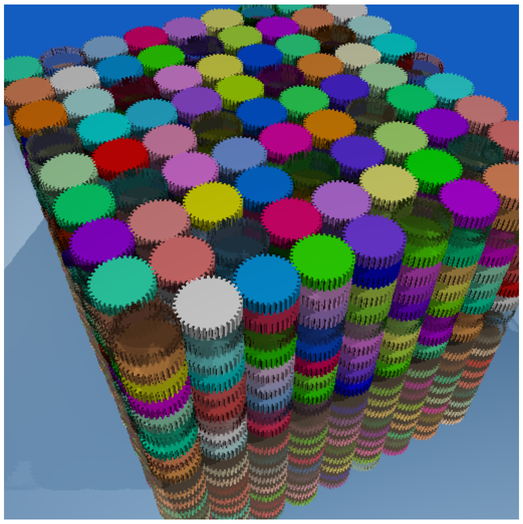
Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	131071	–	0	0	0	0.26	232025.00	18.59	1.0	1054659.09
1	spatmed-xyz(16,2)	319	320	227	134391	17792.00	63.02	13.12	6.99	4.81	10901.20	18.59	0.99	49532.32
2	objmed-xyz(16,2)	65425	65426	996	314658	1659.90	274.45	59.06	0.60	17.19	554.75	18.59	0.65	2503.00
3	objmed(16,2)	65531	65532	252	267726	1871.50	346.00	74.89	0.02	27.43	651.41	18.59	0.62	2942.36
4	OSAH(16,2)	5019	5020	1364	140681	113.69	18.00	4.31	0.94	31.83	33.78	18.59	0.66	134.95
5	OSAH-RMI(16,2)	5019	5020	1364	140681	113.69	18.00	4.31	0.94	31.90	33.96	18.59	0.66	135.77
6	OSAH-xyz(16,2)	5281	5282	1394	142057	133.18	21.21	5.36	1.88	18.62	39.41	18.59	0.65	160.55
7	OSAH(8,1)	75	76	7	132226	9127.30	12.31	3.32	0.14	21.03	2757.39	18.59	1.00	12515.00
8	OSAH(8,2)	70	71	4	132225	9128.50	12.10	3.27	0.09	21.04	2751.83	18.59	1.00	12489.73
9	OSAH(16,1)	5356	5357	1618	140755	110.96	18.61	4.41	1.11	31.82	33.48	18.59	0.64	133.59
10	OSAH(16,2)	5019	5020	1364	140681	113.69	18.00	4.31	0.94	31.83	33.78	18.59	0.66	134.95
11	OSAH(24,1)	143468	143469	56123	231368	21.75	20.83	4.80	1.52	40.28	12.70	18.59	0.24	39.14
12	OSAH(24,2)	114142	114143	34755	223827	25.35	19.84	4.63	1.25	39.90	13.18	18.59	0.28	41.32
13	OSAH(atc)	72922	72923	28766	184136	23.56	20.44	4.73	1.48	37.42	13.14	18.59	0.26	41.14
14	OSAH2(atc)	192425	192426	48787	299524	17.02	26.25	5.31	2.78	49.95	13.61	18.59	0.16	43.27
15	OSAH+LC(atc)	73046	73047	28784	184414	21.98	20.71	4.74	1.74	37.53	13.02	18.59	0.24	40.59
16	OSAH+TPC(atc)	72735	72736	29186	183277	23.69	20.35	4.72	1.48	39.89	13.14	18.59	0.26	41.14
17	OSAH+TPC+LC(atc)	73268	73269	29203	184559	22.05	20.67	4.73	1.73	44.70	12.34	18.59	0.24	37.50
18	OSAH+LC(16,1)	5362	5363	1624	140755	109.38	18.86	4.41	1.37	35.50	32.03	18.59	0.64	127.00
19	OSAH+TPC(16,1)	5774	5775	2017	140538	109.05	18.67	4.43	1.14	37.71	31.62	18.59	0.64	125.14
20	OSAH+TPC+LC(16,1)	5783	5784	2026	140538	107.47	18.93	4.43	1.39	37.67	31.35	18.59	0.63	123.91
21	OSAH+PR(atc)	72922	72923	28766	171576	22.94	20.26	4.69	1.51	39.45	12.24	18.59	0.25	37.05
22	OSAH+SC(atc)	70835	70836	30429	174110	22.14	19.88	4.61	1.67	57.57	12.12	18.59	0.24	36.50
23	OSAH+GCM(atc)	78022	78023	30619	188289	22.86	20.29	4.66	1.45	324.74	12.65	18.59	0.25	38.91
24	OSAH+GCM2(atc)	114150	114151	26050	288861	23.69	23.43	5.15	1.93	472.11	14.04	18.59	0.23	45.23
25	OSAH+GCM3(atc)	82132	82133	24601	237759	102.49	29.76	5.56	1.71	408.52	33.54	18.59	0.51	133.86
26	OSAH+PAR(atc)	72922	72923	28766	184136	5.10	27.24	6.81	2.29	42.74	6.11	20.83	0.42	13.11
27	PARSAH+PAR(atc)	80420	80421	30990	200666	3.50	20.16	4.79	1.69	54.22	5.66	20.83	0.47	10.61
28	OSAH+PER(atc)	72922	72923	28766	184136	8.38	32.43	7.75	3.04	42.78	5.20	17.40	0.37	17.27
29	PERSAH+PER(atc)	111065	111066	44207	226182	4.51	24.36	5.78	3.28	9227.67	4.52	17.40	0.36	12.73
30	SPHSAH+PER(atc)	64191	64192	20383	314352	17361.00	31.00	6.91	3.09	294.09	4036.76	17.40	1.00	26894.33
31	OSAH+SPH(atc)	72922	72923	28766	184136	8.29	32.11	7.66	3.01	42.77	5.51	17.18	0.37	15.24
32	SPHSAH+SPH(atc)	64191	64192	20383	314352	17545.00	30.90	6.85	3.05	294.16	4055.84	17.18	1.00	23840.71
33	OSAH+TA$_{seq}$(16,2)	5019	5020	1364	140681	113.93	43.09	4.32	0.94	25.97	38.40	18.59	0.54	155.95
34	OSAH+TA$^{A}_{rec}$(16,2)	5019	5020	1364	140681	113.68	18.00	4.31	0.94	25.93	33.42	18.59	0.63	133.32
35	OSAH+TA$^{B}_{rec}$(16,2)	5019	5020	1364	140681	113.68	18.00	4.31	0.94	25.93	32.15	18.59	0.66	127.55
36	OSAH+TA$_{SNL}$(16,2)	5019	5020	1364	140681	114.20	17.16	4.33	0.94	26.05	43.74	18.59	0.47	180.23
37	OSAH+TA$_{NLT}$(16,2)	39548	5020	1364	140681	114.20	15.39	4.33	0.94	26.04	42.56	18.59	0.48	174.86
38	OSAH+TA$_{seq}$(18,2)	12900	12901	3804	147446	52.00	46.42	4.46	1.08	27.40	24.68	18.59	0.32	93.59
39	OSAH+TA$^{A}_{rec}$(18,2)	12900	12901	3804	147446	51.88	18.79	4.45	1.08	27.48	19.60	18.59	0.42	70.50
40	OSAH+TA$^{B}_{rec}$(18,2)	12900	12901	3804	147446	51.88	18.79	4.45	1.08	27.49	18.54	18.59	0.45	65.68
41	OSAH+TA$_{SNL}$(18,2)	12900	12901	3804	147446	52.16	17.87	4.46	1.08	27.67	22.25	18.59	0.36	82.55
42	OSAH+TA$_{NLT}$(18,2)	99423	12901	3804	147446	52.16	15.94	4.46	1.08	27.74	24.75	18.59	0.31	93.91
43	OSAH+TA$_{seq}$(atc)	72922	72923	28766	184136	23.65	53.12	4.74	1.48	30.53	19.00	18.59	0.15	67.77
44	OSAH+TA$^{A}_{rec}$(atc)	72922	72923	28766	184136	23.56	20.44	4.73	1.48	30.49	13.77	18.59	0.23	44.00
45	OSAH+TA$^{B}_{rec}$(atc)	72922	72923	28766	184136	23.56	20.44	4.73	1.48	30.54	12.51	18.59	0.26	38.27
46	OSAH+TA$_{SNL}$(atc)	72922	72923	28766	184136	23.72	19.35	4.75	1.48	31.94	13.32	18.59	0.24	41.95
47	OSAH+TA$_{NLT}$(atc)	528558	72923	28766	184136	23.72	17.14	4.75	1.48	32.31	13.46	18.59	0.23	42.59
48	BVH	21992	91171	0	131071	803.76	404.27	274.16	0.00	107.54	5220.49	–	–	–
49	O84	151	1058	892	137338	6108.80	131.76	21.72	14.53	5.18	9452.42	–	–	–
50	O89	151	1058	892	137338	6076.50	85.62	21.72	14.53	5.16	9153.53	–	–	–
51	BSP	319	320	227	134391	17792.00	63.02	13.12	6.99	6.10	13032.90	–	–	–
52	O93	151	1058	892	137340	6032.40	72.36	28.74	21.55	4.91	9261.85	–	–	–
53	UG	0	443680	388122	188973	6723.40	29.36	29.36	25.14	6.43	2347.53	–	–	–
54	AG	2	393009	372191	182880	79.01	30.31	29.83	27.56	370.43	43.38	–	–	–
55	HUG	9	394682	370210	200497	74.93	12.70	10.05	9.00	10.65	46.15	–	–	–
56	RG	41	275886	230133	670903	236.59	29.32	27.62	22.80	2.61	86.53	–	–	–
57	O84A	1328	9297	3935	150854	144.65	42.82	9.72	6.14	23.06	73.28	–	–	–
58	KD	5019	5020	1364	140681	113.69	18.00	4.31	0.94	36.57	52.10	–	–	–

Table 30: Experimental results for scene “*tree15*”.

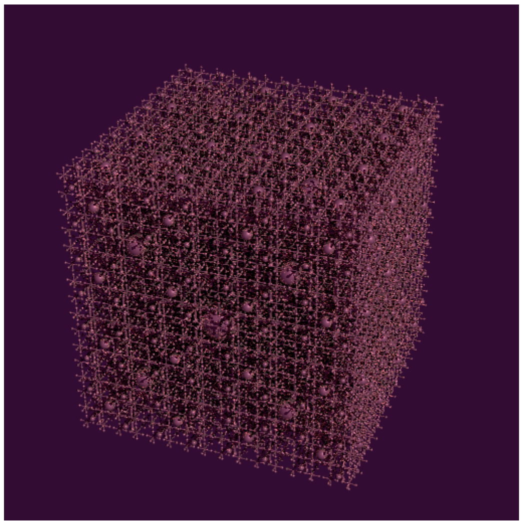
$$N = 131071,$$
$$TP_D{:}\quad N_{prim} = 263169,\quad N^{hit}_{\mathcal{AB}} = 263169,\quad N^{hit}_{prim} = 173215,\quad N_{sec} = 0,\quad N^{hit}_{sec} = 0,$$
$$N_{shad} = 1107745,\quad N^{hit}_{shad} = 48134,\quad T^{MIN}_{R}[s] = 4.31,\quad T_{app}[s] = 4.09,\quad T^{MIN}_{RSA}[s] = 0.22.$$



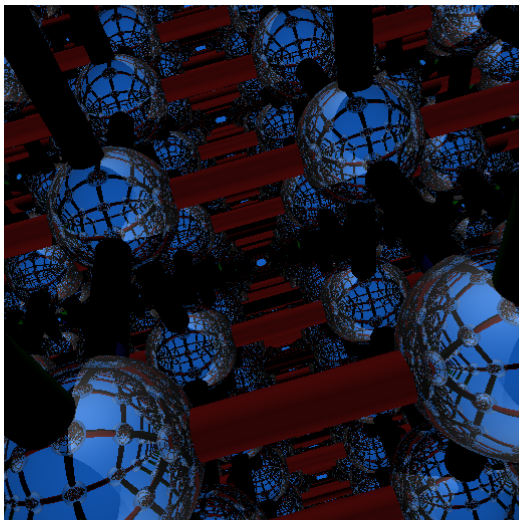
balls5



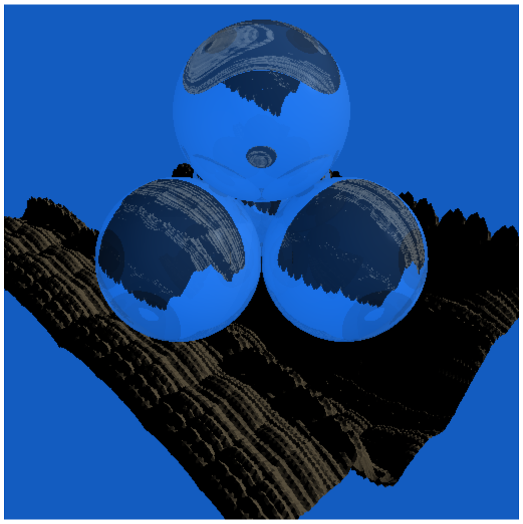
gears9



jacks5

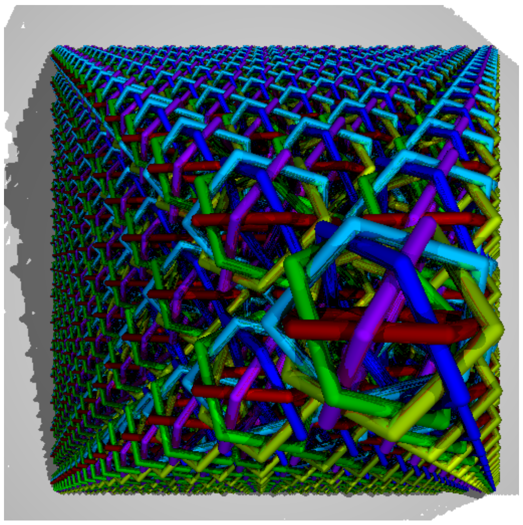


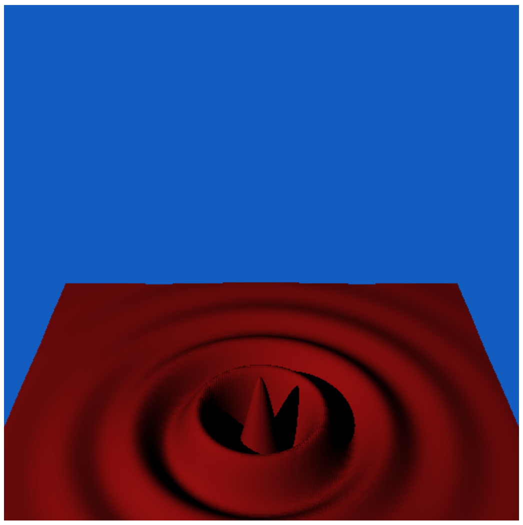
lattice29

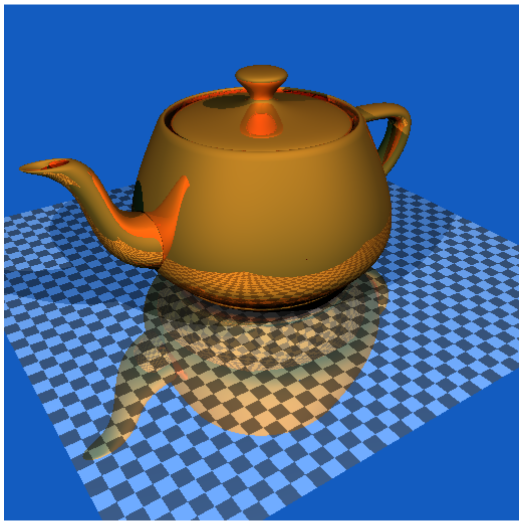


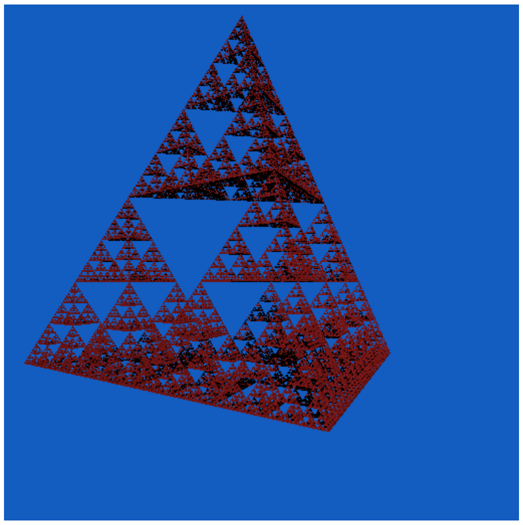
mount8

Figure 5: Visualization of the G^5_{SPD} scenes using the testing procedure TP_D.


rings17


sombrero4


teapot40


tetra8

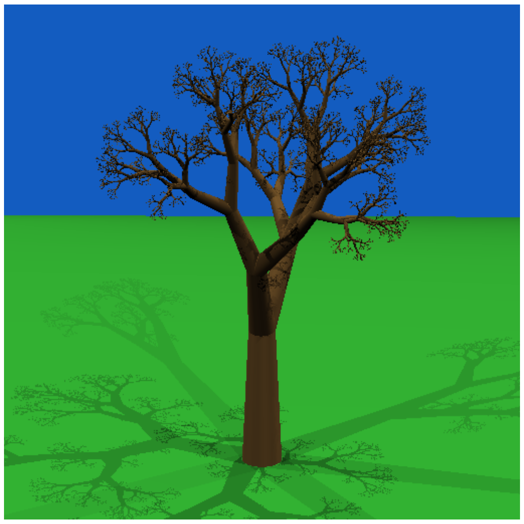

tree15

Figure 6: Visualization of the G^5_{SPD} scenes using the testing procedure TP_D.

Scenes = group G^3_{SPD}

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	1020	–	0.00	0.00	0.00	0.00	671.54	13.29	1.00	2279.43
1	spatmed-xyz(16,2)	6590	6591	881	20030	78.33	36.90	7.15	2.81	0.10	28.39	13.29	0.51	83.12
2	objmed-xyz(16,2)	4300	4301	455	8808	57.76	48.37	9.83	2.06	0.14	27.91	13.29	0.47	63.22
3	objmed(16,2)	4490	4491	465	8544	71.42	57.73	12.49	2.01	0.19	36.98	13.29	0.46	87.61
4	OSAH(16,2)	1870	1871	255	3590	12.93	18.97	3.71	1.05	0.13	13.10	13.29	0.39	20.57
5	OSAH-RMI(16,2)	1870	1871	255	3590	12.93	18.97	3.71	1.05	0.13	12.80	13.29	0.39	19.48
6	OSAH-xyz(16,2)	1825	1826	287	3496	13.12	20.45	4.14	1.45	0.08	13.17	13.29	0.38	20.33
7	OSAH(8,1)	140	141	29	1292	52.09	12.44	2.76	0.74	0.07	21.02	13.29	0.77	37.51
8	OSAH(8,2)	137	138	26	1292	52.95	12.18	2.71	0.59	0.06	21.03	13.29	0.77	37.59
9	OSAH(16,1)	3690	3691	767	4844	10.30	22.03	4.24	1.69	0.17	12.61	13.29	0.31	19.43
10	OSAH(16,2)	1870	1871	255	3590	12.93	18.97	3.71	1.05	0.13	13.10	13.29	0.39	20.57
11	OSAH(24,1)	8978	8979	1133	12429	10.30	23.72	4.52	1.73	0.31	13.22	13.29	0.29	20.23
12	OSAH(24,2)	4303	4304	298	8663	13.02	19.83	3.85	1.05	0.20	13.19	13.29	0.38	20.19
13	OSAH(atc)	2043	2044	501	2937	12.80	19.25	3.78	1.55	0.13	12.83	13.29	0.37	19.51
14	OSAH2(atc)	2400	2401	469	3525	11.92	22.16	4.25	1.67	0.18	13.12	13.29	0.33	20.15
15	OSAH+LC(atc)	2385	2386	502	3905	12.86	20.25	3.94	1.55	0.15	13.24	13.29	0.36	20.07
16	OSAH+TPC(atc)	1622	1623	458	2504	14.12	18.40	3.65	1.50	0.13	13.09	13.29	0.40	19.68
17	OSAH+TPC+LC(atc)	2160	2161	460	3856	14.24	19.95	3.92	1.50	0.17	12.96	13.29	0.38	18.53
18	OSAH+LC(16,1)	3712	3713	769	4866	10.29	22.04	4.23	1.69	0.21	11.88	13.29	0.30	17.30
19	OSAH+TPC(16,1)	3714	3715	769	4871	10.30	22.09	4.24	1.69	0.20	11.88	13.29	0.31	17.14
20	OSAH+TPC+LC(16,1)	3715	3716	770	4871	10.29	22.09	4.24	1.70	0.21	12.01	13.29	0.30	17.41
21	OSAH+PR(atc)	2043	2044	501	2740	12.11	19.20	3.77	1.55	0.21	12.37	13.29	0.36	18.19
22	OSAH+SC(atc)	2142	2143	602	2793	10.31	19.54	3.84	1.77	0.22	12.08	13.29	0.33	17.65
23	OSAH+GCM(atc)	2552	2553	554	3612	11.48	20.98	4.10	1.73	7.39	12.45	13.29	0.34	18.85
24	OSAH+GCM2(atc)	3836	3837	419	6850	13.80	24.09	4.55	1.43	13.23	13.72	13.29	0.34	21.33
25	OSAH+GCM3(atc)	2433	2434	428	3925	17.41	25.43	4.62	1.60	7.35	14.58	13.29	0.36	23.97
26	OSAH+PAR(atc)	2043	2044	501	2937	6.24	17.31	3.62	1.77	0.15	3.12	11.32	0.47	11.44
27	PARSAH+PAR(atc)	2135	2136	540	4413	5.22	13.85	2.93	1.37	0.20	2.93	11.32	0.50	9.99
28	OSAH+PER(atc)	2043	2044	501	2937	6.40	20.19	4.23	2.01	0.15	3.84	13.12	0.45	11.55
29	PERSAH+PER(atc)	1626	1627	450	2731	12.79	15.57	3.40	1.56	16.59	4.81	13.12	0.54	13.99
30	SPHSAH+PER(atc)	1892	1893	517	3907	16.38	19.78	4.10	1.84	0.75	4.56	13.12	0.58	16.98
31	OSAH+SPH(atc)	2043	2044	501	2937	6.39	20.00	4.19	1.99	0.15	4.14	12.30	0.49	9.51
32	SPHSAH+SPH(atc)	1892	1893	517	3907	16.26	19.63	4.07	1.83	0.77	4.88	12.30	0.61	13.61
33	OSAH+TA$_{seq}$(16,2)	1869	1870	255	3591	13.50	41.39	3.75	1.05	0.11	15.75	13.29	0.25	27.84
34	OSAH+TA$^A_{rec}$(16,2)	1869	1870	255	3591	12.92	18.97	3.70	1.05	0.10	13.07	13.29	0.34	19.98
35	OSAH+TA$^B_{rec}$(16,2)	1869	1870	255	3591	12.92	18.97	3.70	1.05	0.11	12.05	13.29	0.39	17.30
36	OSAH+TA$_{SNL}$(16,2)	1869	1870	255	3591	13.51	15.97	3.75	1.05	0.13	12.54	13.29	0.37	18.44
37	OSAH+TA$_{NLT}$(16,2)	9737	1870	255	3591	13.51	14.61	3.75	1.05	0.14	12.88	13.29	0.35	19.09
38	OSAH+TA$_{seq}$(18,2)	2526	2527	282	4806	13.43	43.10	3.83	1.06	0.13	16.07	13.29	0.24	28.22
39	OSAH+TA$^A_{rec}$(18,2)	2526	2527	282	4806	12.83	19.38	3.77	1.06	0.12	13.23	13.29	0.33	20.29
40	OSAH+TA$^B_{rec}$(18,2)	2526	2527	282	4806	12.83	19.38	3.77	1.05	0.13	12.13	13.29	0.38	17.49
41	OSAH+TA$_{SNL}$(18,2)	2526	2527	282	4806	13.43	16.32	3.83	1.06	0.16	12.65	13.29	0.36	18.69
42	OSAH+TA$_{NLT}$(18,2)	12989	2527	282	4806	13.43	14.89	3.83	1.06	0.18	12.72	13.29	0.36	18.66
43	OSAH+TA$_{seq}$(atc)	2050	2051	502	2945	13.35	41.84	3.82	1.54	0.11	15.80	13.29	0.24	27.88
44	OSAH+TA$^A_{rec}$(atc)	2050	2051	502	2945	12.78	19.26	3.78	1.54	0.11	13.11	13.29	0.32	19.89
45	OSAH+TA$^B_{rec}$(atc)	2050	2051	502	2945	12.78	19.26	3.78	1.54	0.10	12.05	13.29	0.37	17.15
46	OSAH+TA$_{SNL}$(atc)	2050	2051	502	2945	13.35	16.01	3.82	1.54	0.13	12.56	13.29	0.34	18.51
47	OSAH+TA$_{NLT}$(atc)	9509	2051	502	2945	13.35	14.65	3.82	1.54	0.15	12.54	13.29	0.35	18.26
48	BVH	111	637	0	1020	52.78	28.45	21.06	0.00	0.08	208.71	–	–	–
49	O84	3197	22382	4791	49927	71.71	60.18	10.50	5.23	0.16	52.07	–	–	–
50	O89	3197	22382	4791	49927	71.56	40.75	10.47	5.23	0.13	43.16	–	–	–
51	BSP	6590	6591	881	20030	78.33	36.90	7.15	2.81	0.76	75.73	–	–	–
52	O93	3214	22503	4860	50094	70.75	36.96	15.18	10.05	0.14	50.86	–	–	–
53	UG	0	5127	3148	6020	153.68	8.12	8.12	4.76	0.05	53.43	–	–	–
54	AG	201	5368	1756	12312	32.55	10.14	8.40	2.63	0.20	52.04	–	–	–
55	HUG	44	1979	804	4657	143.50	6.78	4.61	1.99	0.05	73.10	–	–	–
56	RG	224	4340	1323	14902	38.76	8.84	6.87	3.19	0.04	30.34	–	–	–
57	O84A	4095	28666	4592	69335	21.62	38.09	7.46	3.50	0.53	30.31	–	–	–
58	KD	1870	1871	255	3590	12.93	18.97	3.71	1.05	0.42	21.19	–	–	–

Table 31: Experimental results, summary for G^3_{SPD} scenes, average values are reported.

Scenes = group G^4_{SPD}

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	7635	–	0.00	0.00	0.00	0.02	7839.09	13.73	1.00	22297.46
1	spatmed-xyz(16,2)	10073	10074	1405	43897	331.92	39.30	7.44	3.22	0.33	102.73	13.73	0.61	392.16
2	objmed-xyz(16,2)	23997	23998	2540	54859	99.25	81.36	16.21	2.98	1.12	49.61	13.73	0.47	121.40
3	objmed(16,2)	21948	21949	1272	47813	130.01	100.12	20.79	2.60	1.51	63.60	13.73	0.46	171.67
4	OSAH(16,2)	8317	8318	1263	19515	15.07	24.47	4.49	1.61	1.26	16.76	13.73	0.38	25.98
5	OSAH-RMI(16,2)	8317	8318	1263	19515	15.07	24.47	4.49	1.61	1.27	16.43	13.73	0.38	25.12
6	OSAH-xyz(16,2)	8057	8058	1311	19615	17.59	26.92	5.10	2.08	0.77	17.29	13.73	0.39	27.46
7	OSAH(8,1)	172	173	24	8757	250.54	12.62	2.67	0.69	0.80	75.45	13.73	0.94	175.32
8	OSAH(8,2)	171	172	22	8757	250.75	12.50	2.64	0.65	0.78	75.52	13.73	0.94	175.62
9	OSAH(16,1)	12421	12422	3042	21654	13.03	26.67	4.86	2.14	1.34	16.18	13.73	0.32	24.75
10	OSAH(16,2)	8317	8318	1263	19515	15.07	24.47	4.49	1.61	1.26	16.76	13.73	0.38	25.98
11	OSAH(24,1)	65655	65656	8352	91893	10.84	31.37	5.65	2.35	2.73	16.86	13.73	0.26	25.45
12	OSAH(24,2)	33433	33434	2414	66189	13.32	26.92	4.88	1.66	2.02	16.77	13.73	0.33	25.27
13	OSAH(atc)	16873	16874	3836	25848	12.31	26.70	4.86	2.19	1.45	16.01	13.73	0.31	24.14
14	OSAH2(atc)	22381	22382	3960	34095	11.85	31.69	5.52	2.42	2.12	16.85	13.73	0.28	26.39
15	OSAH+LC(atc)	20396	20397	3856	35627	12.26	27.79	5.03	2.21	1.65	16.43	13.73	0.30	24.66
16	OSAH+TPC(atc)	13109	13110	3351	21965	13.29	25.58	4.67	2.14	1.43	16.24	13.73	0.33	24.18
17	OSAH+TPC+LC(atc)	19204	19205	3384	37141	13.18	27.39	4.97	2.14	1.97	16.04	13.73	0.32	23.33
18	OSAH+LC(16,1)	12440	12441	3054	21663	12.98	26.68	4.85	2.17	1.55	15.46	13.73	0.32	22.82
19	OSAH+TPC(16,1)	12500	12501	3101	21710	13.01	26.66	4.85	2.16	1.59	15.44	13.73	0.32	22.45
20	OSAH+TPC+LC(16,1)	12504	12505	3106	21710	12.97	26.68	4.85	2.17	1.59	15.50	13.73	0.32	22.86
21	OSAH+PR(atc)	16873	16874	3836	22458	11.30	26.62	4.85	2.19	2.17	15.03	13.73	0.29	21.84
22	OSAH+SC(atc)	17462	17463	5282	22824	9.78	26.81	4.89	2.56	2.60	14.82	13.73	0.27	22.04
23	OSAH+GCM(atc)	22286	22287	4672	33228	11.45	28.85	5.21	2.35	66.90	15.75	13.73	0.28	23.41
24	OSAH+GCM2(atc)	38410	38411	3708	71266	13.66	33.03	5.81	2.05	129.40	17.91	13.73	0.29	27.05
25	OSAH+GCM3(atc)	23053	23054	3481	38924	28.78	35.13	5.91	2.34	72.16	28.32	13.73	0.35	38.10
26	OSAH+PAR(atc)	16873	16874	3836	25848	7.16	27.96	5.49	2.98	1.69	3.98	12.31	0.42	15.85
27	PARSAH+PAR(atc)	17893	17894	3946	57241	6.72	19.95	3.93	1.94	2.46	3.77	12.31	0.50	14.64
28	OSAH+PER(atc)	16873	16874	3836	25848	6.41	27.79	5.51	2.92	1.72	4.28	11.89	0.41	13.97
29	PERSAH+PER(atc)	13820	13821	3514	23614	44.33	21.23	4.28	2.32	168.79	11.71	11.89	0.50	28.72
30	SPHSAH+PER(atc)	17146	17147	3948	40987	36.64	29.24	5.70	3.01	8.35	9.07	11.89	0.62	37.44
31	OSAH+SPH(atc)	16873	16874	3836	25848	6.38	27.50	5.45	2.88	1.71	4.60	11.70	0.40	9.63
32	SPHSAH+SPH(atc)	17146	17147	3948	40987	36.26	28.99	5.65	2.98	8.32	9.34	11.70	0.61	30.29
33	OSAH+TA$_{seq}$(16,2)	8319	8320	1264	19518	15.79	58.22	4.54	1.62	1.04	20.72	13.73	0.25	36.61
34	OSAH+TA$^{A}_{rec}$(16,2)	8319	8320	1264	19518	15.06	24.45	4.48	1.61	1.02	17.04	13.73	0.33	26.41
35	OSAH+TA$^{B}_{rec}$(16,2)	8319	8320	1264	19518	15.06	24.45	4.48	1.61	1.03	15.53	13.73	0.38	22.94
36	OSAH+TA$_{SNL}$(16,2)	8319	8320	1264	19518	15.80	20.07	4.54	1.62	1.17	16.43	13.73	0.35	24.42
37	OSAH+TA$_{NLT}$(16,2)	46575	8320	1264	19518	15.80	18.10	4.54	1.62	1.22	16.61	13.73	0.35	24.51
38	OSAH+TA$_{seq}$(18,2)	13332	13333	1722	27372	14.16	62.58	4.72	1.65	1.20	20.83	13.73	0.22	36.78
39	OSAH+TA$^{A}_{rec}$(18,2)	13332	13333	1722	27372	13.54	25.56	4.65	1.65	1.16	16.96	13.73	0.30	26.20
40	OSAH+TA$^{B}_{rec}$(18,2)	13332	13333	1722	27372	13.54	25.56	4.65	1.65	1.17	15.31	13.73	0.35	22.24
41	OSAH+TA$_{SNL}$(18,2)	13332	13333	1722	27372	14.17	20.99	4.72	1.65	1.40	16.07	13.73	0.33	23.56
42	OSAH+TA$_{NLT}$(18,2)	74109	13333	1722	27372	14.17	18.84	4.72	1.65	1.46	16.14	13.73	0.32	23.67
43	OSAH+TA$_{seq}$(atc)	16905	16906	3840	25879	12.89	66.24	4.92	2.20	1.22	20.87	13.73	0.19	37.06
44	OSAH+TA$^{A}_{rec}$(atc)	16905	16906	3840	25879	12.30	26.70	4.86	2.20	1.20	16.80	13.73	0.27	25.82
45	OSAH+TA$^{B}_{rec}$(atc)	16905	16906	3840	25879	12.30	26.70	4.86	2.20	1.20	15.11	13.73	0.31	21.95
46	OSAH+TA$_{SNL}$(atc)	16905	16906	3840	25879	12.90	21.64	4.92	2.20	1.49	15.87	13.73	0.29	23.72
47	OSAH+TA$_{NLT}$(atc)	85673	16906	3840	25879	12.90	19.36	4.92	2.20	1.57	15.78	13.73	0.29	22.88
48	BVH	1003	5001	0	7635	170.08	135.83	96.38	0.00	1.40	1146.82	–	–	–
49	O84	4846	33924	8996	86609	246.65	65.19	10.98	6.04	0.46	112.75	–	–	–
50	O89	4846	33924	8996	86609	246.45	43.14	10.97	6.04	0.41	102.82	–	–	–
51	BSP	10073	10074	1405	43897	331.92	39.30	7.44	3.22	1.04	151.56	–	–	–
52	O93	4864	34053	9105	86898	244.60	38.09	15.16	10.40	0.42	111.15	–	–	–
53	UG	0	38369	28449	36013	283.89	13.97	13.97	10.12	0.39	100.36	–	–	–
54	AG	2185	54303	18640	111948	42.26	21.62	17.56	6.67	3.11	139.56	–	–	–
55	HUG	424	16086	10026	32831	81.41	11.44	8.77	5.44	0.42	58.44	–	–	–
56	RG	3320	55985	13861	243468	50.17	13.31	10.85	6.33	0.62	40.67	–	–	–
57	O84A	6963	48746	9532	120079	23.67	46.46	8.48	4.46	1.79	37.13	–	–	–
58	KD	8317	8318	1263	19515	15.07	24.47	4.49	1.61	2.02	26.68	–	–	–

Table 32: Experimental results, summary for G^4_{SPD} scenes, average values are reported.

Scenes = group G^5_{SPD}

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	98880	–	0.00	0.00	0.00	0.21	103166.96	13.40	1.00	304702.66
1	spatmed-xyz(16,2)	16132	16133	1735	215542	2088.90	38.15	7.07	3.37	3.31	1205.40	13.40	0.84	5192.23
2	objmed-xyz(16,2)	61558	61559	4176	331923	372.87	109.84	22.01	1.89	13.51	149.05	13.40	0.68	427.09
3	objmed(16,2)	63808	63809	3393	237074	452.42	208.61	40.90	4.25	19.24	190.85	13.40	0.59	639.95
4	OSAH(16,2)	23113	23114	3580	147514	41.69	26.99	4.71	1.78	19.52	26.35	13.40	0.60	46.27
5	OSAH-RMI(16,2)	23113	23114	3580	147514	41.69	26.99	4.71	1.78	19.60	26.44	13.40	0.60	46.09
6	OSAH-xyz(16,2)	22873	22874	3145	155665	52.60	30.38	5.54	2.50	12.50	28.11	13.40	0.61	53.52
7	OSAH(8,1)	181	182	23	104396	2659.95	12.66	2.62	0.63	14.01	883.41	13.40	0.99	2696.34
8	OSAH(8,2)	180	181	22	104396	2660.08	12.62	2.61	0.62	14.01	884.45	13.40	0.99	2698.10
9	OSAH(16,1)	24068	24069	4124	147680	40.80	27.54	4.79	1.91	19.60	26.24	13.40	0.59	46.44
10	OSAH(16,2)	23113	23114	3580	147514	41.69	26.99	4.71	1.78	19.52	26.35	13.40	0.60	46.27
11	OSAH(24,1)	476453	476454	83642	663728	11.53	38.32	6.55	3.00	35.32	23.46	13.40	0.26	31.60
12	OSAH(24,2)	251440	251441	28940	498597	14.18	33.74	5.76	2.21	28.70	21.09	13.40	0.34	30.14
13	OSAH(atc)	249915	249916	55707	366588	12.36	35.23	6.01	2.90	27.26	19.94	13.40	0.29	28.78
14	OSAH2(atc)	356130	356131	62027	519386	11.99	43.45	6.90	3.33	39.95	21.98	13.40	0.26	33.00
15	OSAH+LC(atc)	277145	277146	55889	445138	12.13	36.11	6.14	2.95	28.95	20.41	13.40	0.28	29.28
16	OSAH+TPC(atc)	203940	203941	51251	314085	12.94	34.28	5.86	2.84	27.16	19.79	13.40	0.31	28.49
17	OSAH+TPC+LC(atc)	252635	252636	51449	442823	12.64	35.61	6.05	2.89	33.99	20.04	13.40	0.29	27.55
18	OSAH+LC(16,1)	24077	24078	4134	147680	40.50	27.60	4.78	1.97	22.63	25.16	13.40	0.59	43.20
19	OSAH+TPC(16,1)	24243	24244	4291	147725	40.63	27.57	4.80	1.92	23.20	25.28	13.40	0.59	43.23
20	OSAH+TPC+LC(16,1)	24252	24253	4300	147725	40.41	27.63	4.80	1.98	23.22	25.44	13.40	0.59	43.26
21	OSAH+PR(atc)	249915	249916	55707	304152	10.90	35.28	6.04	2.90	40.48	19.32	13.40	0.26	26.87
22	OSAH+SC(atc)	262549	262550	73790	328496	9.60	35.10	5.97	3.40	66.34	18.51	13.40	0.25	26.11
23	OSAH+GCM(atc)	348137	348138	71380	500124	11.89	37.90	6.42	3.10	1066.65	21.13	13.40	0.27	29.78
24	OSAH+GCM2(atc)	629685	629686	53090	1146337	13.87	42.23	7.07	2.72	2171.18	28.14	13.40	0.28	36.30
25	OSAH+GCM3(atc)	348085	348086	50674	575455	115.65	45.55	7.08	3.13	1145.45	137.11	13.40	0.39	111.59
26	OSAH+PAR(atc)	249915	249916	55707	366588	10.19	47.77	9.10	5.16	31.55	6.21	10.96	0.44	21.83
27	PARSAH+PAR(atc)	227704	227705	55818	963815	12.48	29.04	5.72	2.84	44.81	6.77	10.96	0.54	21.59
28	OSAH+PER(atc)	249915	249916	55707	366588	6.42	36.59	6.77	3.87	31.88	5.56	11.13	0.42	16.64
29	PERSAH+PER(atc)	197143	197144	50704	325145	232.42	27.09	5.17	3.03	2683.95	69.67	11.13	0.54	175.09
30	SPHSAH+PER(atc)	256487	256488	58838	600446	1762.56	41.47	7.76	4.35	133.65	411.75	11.13	0.62	2717.62
31	OSAH+SPH(atc)	249915	249916	55707	366588	6.40	36.23	6.71	3.84	31.71	5.91	12.40	0.44	15.13
32	SPHSAH+SPH(atc)	256487	256488	58838	600446	1781.15	41.11	7.69	4.33	133.96	414.12	12.40	0.62	2410.13
33	OSAH+TA$_{seq}$(16,2)	23113	23114	3581	147514	43.17	65.02	4.76	1.78	16.16	30.97	13.40	0.45	58.86
34	OSAH+TA$^A_{rec}$(16,2)	23113	23114	3581	147514	41.61	26.99	4.70	1.78	16.12	26.73	13.40	0.54	47.46
35	OSAH+TA$^B_{rec}$(16,2)	23113	23114	3581	147514	41.61	26.99	4.70	1.78	16.15	25.00	13.40	0.60	43.02
36	OSAH+TA$_{SNL}$(16,2)	23113	23114	3581	147514	43.21	21.36	4.76	1.78	16.76	30.54	13.40	0.54	52.82
37	OSAH+TA$_{NLT}$(16,2)	118817	23114	3581	147514	43.21	19.25	4.76	1.78	16.73	27.10	13.40	0.57	49.04
38	OSAH+TA$_{seq}$(18,2)	57566	57567	11238	187225	22.93	76.98	5.24	2.06	17.48	26.53	13.40	0.31	47.69
39	OSAH+TA$^A_{rec}$(18,2)	57566	57567	11238	187225	21.95	30.04	5.17	2.06	17.44	22.31	13.40	0.38	35.63
40	OSAH+TA$^B_{rec}$(18,2)	57566	57567	11238	187225	21.95	30.04	5.17	2.06	17.43	20.36	13.40	0.44	30.83
41	OSAH+TA$_{SNL}$(18,2)	57566	57567	11238	187225	22.95	23.91	5.24	2.06	18.63	21.52	13.40	0.42	33.55
42	OSAH+TA$_{NLT}$(18,2)	316687	57567	11238	187225	22.95	21.36	5.24	2.06	19.06	22.88	13.40	0.39	35.40
43	OSAH+TA$_{seq}$(atc)	250908	250909	55701	367546	12.92	98.77	6.08	2.90	23.46	29.18	13.40	0.17	48.11
44	OSAH+TA$^A_{rec}$(atc)	250908	250909	55701	367546	12.35	35.22	6.00	2.89	22.61	21.55	13.40	0.24	32.13
45	OSAH+TA$^B_{rec}$(atc)	250908	250909	55701	367546	12.35	35.22	6.00	2.89	22.75	19.06	13.40	0.29	26.51
46	OSAH+TA$_{SNL}$(atc)	250908	250909	55701	367546	12.94	27.85	6.09	2.90	35.77	22.86	13.40	0.25	30.41
47	OSAH+TA$_{NLT}$(atc)	1053664	198637	53522	319202	13.80	23.90	5.92	2.83	32.66	30.29	13.92	0.17	54.49
48	BVH	14049	66508	0	98880	769.52	731.69	519.51	0.00	131.74	8772.62	–	–	–
49	O84	8712	60985	12877	324543	781.94	64.12	10.56	6.08	3.93	1035.91	–	–	–
50	O89	8712	60985	12877	324543	779.80	41.87	10.55	6.08	3.70	999.11	–	–	–
51	BSP	16132	16133	1735	215542	2088.90	38.15	7.07	3.37	4.38	1453.74	–	–	–
52	O93	8712	60985	12877	324544	768.76	36.52	14.20	9.92	3.65	1016.39	–	–	–
53	UG	0	471705	381796	385702	754.85	28.81	28.81	24.21	5.67	282.33	–	–	–
54	AG	14108	507733	209423	1026475	48.60	23.63	20.89	14.87	175.17	190.84	–	–	–
55	HUG	4378	158654	111416	414167	241.48	18.07	15.50	12.07	4.98	100.59	–	–	–
56	RG	11437	387369	141279	2344246	106.91	23.06	20.49	15.19	6.00	72.52	–	–	–
57	O84A	13023	91166	17164	411170	62.75	49.14	8.61	4.71	17.31	49.79	–	–	–
58	KD	23113	23114	3580	147514	41.69	26.99	4.71	1.78	22.26	40.39	–	–	–

Table 33: Experimental results, summary for G^5_{SPD} scenes, average values are reported.

Scenes = group G^3_{SPD}, G^4_{SPD}, and G^5_{SPD}

Line	Mnemonic Notation	Minimum Testing Output												
		Σ				Δ				Θ				
		N_G	N_E	N_{EE}	N_{ER}	r_{ITM}	$\tilde{N}_{TS}$	$\tilde{N}_{ETS}$	$\tilde{N}_{EETS}$	T_B	T_R	Θ_{APP}	Θ_{rat}	Θ_{RUN}
0	naïve *RSA*	0	0	1	35845	–	0.00	0.00	0.00	0.07	37225.86	13.47	1.00	109759.85
1	spatmed-xyz(16,2)	10932	10933	1340	93156	833.05	38.11	7.22	3.13	1.25	445.51	13.47	0.65	1889.17
2	objmed-xyz(16,2)	29952	29953	2390	131863	176.63	79.85	16.02	2.31	4.92	75.52	13.47	0.54	203.90
3	objmed(16,2)	30082	30083	1710	97811	217.95	122.15	24.73	2.95	6.98	97.15	13.47	0.50	299.74
4	OSAH(16,2)	11100	11101	1700	56873	23.23	23.48	4.30	1.48	6.97	18.74	13.47	0.46	30.94
5	OSAH-RMI(16,2)	11100	11101	1700	56873	23.23	23.48	4.30	1.48	7.00	18.56	13.47	0.46	30.23
6	OSAH-xyz(16,2)	10918	10919	1581	59592	27.77	25.92	4.93	2.01	4.45	19.52	13.47	0.46	33.77
7	OSAH(8,1)	165	166	25	38149	987.52	12.57	2.68	0.69	4.96	326.62	13.47	0.90	969.73
8	OSAH(8,2)	162	163	23	38148	987.93	12.43	2.65	0.62	4.95	327.00	13.47	0.90	970.44
9	OSAH(16,1)	13393	13394	2644	58059	21.38	25.41	4.63	1.91	7.04	18.34	13.47	0.41	30.20
10	OSAH(16,2)	11100	11101	1700	56873	23.23	23.48	4.30	1.48	6.97	18.74	13.47	0.46	30.94
11	OSAH(24,1)	183695	183696	31042	256017	10.89	31.14	5.57	2.36	12.79	17.85	13.47	0.27	25.76
12	OSAH(24,2)	96392	96393	10551	191150	13.51	26.83	4.83	1.64	10.31	17.01	13.47	0.35	25.20
13	OSAH(atc)	89610	89611	20014	131791	12.49	27.06	4.89	2.21	9.61	16.26	13.47	0.32	24.14
14	OSAH2(atc)	126970	126971	22152	185669	11.92	32.43	5.56	2.48	14.08	17.31	13.47	0.29	26.51
15	OSAH+LC(atc)	99975	99976	20082	161557	12.42	28.05	5.04	2.24	10.25	16.70	13.47	0.31	24.67
16	OSAH+TPC(atc)	72890	72891	18353	112851	13.45	26.09	4.73	2.16	9.57	16.37	13.47	0.35	24.12
17	OSAH+TPC+LC(atc)	91333	91334	18431	161273	13.36	27.65	4.98	2.18	12.04	16.35	13.47	0.33	23.14
18	OSAH+LC(16,1)	13410	13411	2652	58070	21.26	25.44	4.62	1.94	8.13	17.50	13.47	0.41	27.77
19	OSAH+TPC(16,1)	13486	13487	2720	58102	21.31	25.44	4.63	1.92	8.33	17.54	13.47	0.41	27.60
20	OSAH+TPC+LC(16,1)	13490	13491	2725	58102	21.22	25.47	4.63	1.95	8.34	17.65	13.47	0.41	27.84
21	OSAH+PR(atc)	89610	89611	20014	109783	11.44	27.04	4.89	2.22	14.29	15.57	13.47	0.30	22.30
22	OSAH+SC(atc)	94051	94052	26558	118038	9.90	27.15	4.90	2.58	23.05	15.14	13.47	0.28	21.94
23	OSAH+GCM(atc)	124325	124326	25535	178988	11.61	29.25	5.24	2.39	380.31	16.44	13.47	0.30	24.01
24	OSAH+GCM2(atc)	223977	223978	19072	408151	13.77	33.12	5.81	2.06	771.27	19.92	13.47	0.30	28.23
25	OSAH+GCM3(atc)	124524	124525	18194	206101	53.95	35.37	5.87	2.36	408.32	60.00	13.47	0.37	57.89
26	OSAH+PAR(atc)	89610	89611	20014	131791	7.86	31.01	6.07	3.30	11.13	4.43	11.53	0.44	16.37
27	PARSAH+PAR(atc)	82577	82578	20101	341823	8.14	20.95	4.19	2.05	15.82	4.49	11.53	0.52	15.41
28	OSAH+PER(atc)	89610	89611	20014	131791	6.41	28.19	5.50	2.93	11.25	4.56	12.05	0.42	14.05
29	PERSAH+PER(atc)	70863	70864	18223	117163	96.51	21.29	4.28	2.30	956.44	28.73	12.05	0.53	72.60
30	SPHSAH+PER(atc)	91842	91843	21101	215113	605.19	30.16	5.85	3.07	47.58	141.79	12.05	0.60	924.01
31	OSAH+SPH(atc)	89610	89611	20014	131791	6.39	27.91	5.45	2.91	11.19	4.88	12.13	0.44	11.42
32	SPHSAH+SPH(atc)	91842	91843	21101	215113	611.22	29.91	5.80	3.05	47.68	142.78	12.13	0.62	818.01
33	OSAH+TA$_{seq}$(16,2)	11100	11101	1700	56874	24.16	54.88	4.35	1.48	5.77	22.48	13.47	0.32	41.11
34	OSAH+TA$^A_{rec}$(16,2)	11100	11101	1700	56874	23.20	23.47	4.29	1.48	5.75	18.94	13.47	0.40	31.28
35	OSAH+TA$^B_{rec}$(16,2)	11100	11101	1700	56874	23.20	23.47	4.29	1.48	5.76	17.52	13.47	0.46	27.75
36	OSAH+TA$_{SNL}$(16,2)	11100	11101	1700	56874	24.17	19.13	4.35	1.48	6.02	19.84	13.47	0.42	31.89
37	OSAH+TA$_{NLT}$(16,2)	58376	11101	1700	56874	24.17	17.32	4.35	1.48	6.03	18.86	13.47	0.42	30.88
38	OSAH+TA$_{seq}$(18,2)	24475	24476	4414	73134	16.84	60.89	4.60	1.59	6.27	21.14	13.47	0.26	37.56
39	OSAH+TA$^A_{rec}$(18,2)	24475	24476	4414	73134	16.11	24.99	4.53	1.59	6.24	17.50	13.47	0.34	27.37
40	OSAH+TA$^B_{rec}$(18,2)	24475	24476	4414	73134	16.11	24.99	4.53	1.58	6.24	15.93	13.47	0.39	23.52
41	OSAH+TA$_{SNL}$(18,2)	24475	24476	4414	73134	16.85	20.40	4.60	1.59	6.73	16.75	13.47	0.37	25.27
42	OSAH+TA$_{NLT}$(18,2)	134595	24476	4414	73134	16.85	18.37	4.60	1.59	6.90	17.25	13.47	0.36	25.91
43	OSAH+TA$_{seq}$(atc)	89955	89956	20014	132123	13.06	68.95	4.94	2.21	8.26	21.95	13.47	0.20	37.69
44	OSAH+TA$^A_{rec}$(atc)	89955	89956	20014	132123	12.48	27.06	4.88	2.21	7.97	17.15	13.47	0.28	25.95
45	OSAH+TA$^B_{rec}$(atc)	89955	89956	20014	132123	12.48	27.06	4.88	2.21	8.02	15.41	13.47	0.32	21.87
46	OSAH+TA$_{SNL}$(atc)	89955	89956	20014	132123	13.06	21.83	4.94	2.21	12.46	17.10	13.47	0.29	24.21
47	OSAH+TA$_{NLT}$(atc)	359820	68183	18107	109002	13.33	19.15	4.85	2.17	10.73	19.17	13.64	0.27	31.10
48	BVH	5055	24049	0	35845	330.80	298.66	212.32	0.00	44.41	3376.05	–	–	–
49	O84	5585	39097	8888	153693	366.77	63.16	10.68	5.78	1.52	400.24	–	–	–
50	O89	5585	39097	8888	153693	365.94	41.92	10.66	5.78	1.41	381.70	–	–	–
51	BSP	10932	10933	1340	93156	833.05	38.11	7.22	3.13	2.06	560.34	–	–	–
52	O93	5597	39180	8947	153845	361.37	37.19	14.85	10.12	1.41	392.80	–	–	–
53	UG	0	171734	137798	142578	397.47	16.97	16.97	13.03	2.03	145.37	–	–	–
54	AG	5498	189135	76606	383578	41.14	18.46	15.61	8.06	59.49	127.48	–	–	–
55	HUG	1615	58906	40749	150552	155.46	12.10	9.62	6.50	1.81	77.38	–	–	–
56	RG	4994	149231	52154	867538	65.28	15.07	12.74	8.24	2.22	47.84	–	–	–
57	O84A	8027	56192	10429	200194	36.01	44.57	8.19	4.22	6.54	39.08	–	–	–
58	KD	11100	11101	1700	56873	23.23	23.48	4.30	1.48	8.23	29.42	–	–	–

Table 34: Experimental results, summary for G^3_{SPD}, G^4_{SPD}, and G^5_{SPD} scenes, average values are reported.