ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

SCHOOL OF INFORMATION SCIENCES & TECHNOLOGY

DEPARTMENT OF INFORMATICS

BSc THESIS

TITLE : ACCELERATION OF A RAY TRACING DATA STRUCTURE

DIMITRIOS TSIOMPIKAS

SUPERVISORS :
GEORGIOS PAPAIOANNOU, ASSOCIATE PROFESSOR
IORDANIS EVAGGELOU, DOCTORATE STUDENT

AUGUST 2022

ABSTRACT

The following thesis has the purpose of accelerating the already optimised ray tracing algorithm by accelerating one of the many algorithms that are used for this reason , the K-d tree. In this experiment , the goal is to reduce the initial primitive population of a scene , which will be placed in a K-d tree for the experiment, to a smaller subsample getting the same or a little bit worse cost during ray tracing. In order to achieve this, we initially created the tree with all primitives and afterwards we created trees using a subsample of the initial population , we removed all the primitives, filled their leaves with primitives that were inside the Axis Aligned Bounding Box (AABB) and computed the cost that a ray takes to traverse the whole tree from root to leaves , finally finding the best possible percentage that would reduce the initial population without having a big setback in cost and time.

ACKNOWLEDGEMENTS

First of all, i'd like to sincerely thank my supervisors , Professor Georgios Papaioannou and his Doctorate student, Iordanis Evaggelou, for their tremendous help and guidance throughout this thesis.

In addition, i'm grateful to all my colleagues for their mental support and feedback for writing this thesis.

Finally , i'd like to thank my friends and my mother for providing me with mental support and motivation throughout my undergraduate years.

Contents

1 Introduction

1.1 Purpose of the experiment

Ray tracing ,nowadays, is one of the most used lighting algorithms in Computer Graphics and it can achieve very accurate simulations of shadows and lighting in most virtual environments (which i'm going to refer to as "scene/s" from now on). The problem lies in the fact that if there is a great amount of primitives in a scene , ray tracing can get extremely cost-inefficient which results in a need for stronger hardware in order for it to execute.

There are many ways to accelerate ray tracing but in this thesis we're going to focus on only one of them , that helps with faster ray intersection. It's called K-d tree accelerator and it used the K-d tree to split the AABBs of a scene to smaller ones until the AABBs contain very few primitives.

1.2 Thesis structure

The thesis is structured as follows :

Chapter 2 : Background
Chapter 3 : Method overview
Chapter 4 : Implementation
Chapter 5 : Conclusion
Chapter 6 : Bibliography
Chapter 7 : Acronyms

2 Background

2.1 Ray Tracing

Ray tracing ,as previously mentioned, is a lighting algorithm that simulates light and shadows in a scene. The algorithm functions as follows :

Firstly, rays start from a point (usually the light source) and they go to a specific direction. Those that "hit" a primitive are considered hit. The rest are considered missed. Lighting and the rest of the attributes are evaluated at the intersection point. These rays are considered the primary rays. New rays start from the rays that hit a primitive and they head towards the light sources of the scene. These are called secondary rays or shadow rays and they create the shadows and reflections of the scene.

Lastly, the algorithm is recursively executed , which means that in a large scene with many primitives it's going to need a lot of time to finish. This is the reason that accelerators have been created , just like the K-d tree that we're going to examine in this thesis.

2.2 K-d Tree

The K-d Tree is a specialization of the Binary Tree , which splits data based on the best axis instead of the classic way of insertion that binary trees use. K stands for the number of dimensions out of which the tree is able to find the best axis to split the data, in computer graphics the K is usually equal to 3 as the scenes are represented in 3-D space.

The data can be split in 3 ways : Median cut, midpoint and the Surface Area Heuristic (SAH). In this experiment we're going to use the SAH algorithm , in which the tree splits as follows : a cost function is computed by using the surface area of the examined tree node and the surface areas of it's children nodes. This way , SAH finds the best axis from x,y,z and the best point that the AABB splitting will occur.

Last but not least, the K-d tree is a reliable ray tracing accelerator but it can get cost-inefficient as well if the scene is too large and the number of primitives is big. The purpose of this experiment is to reduce the initial size that it takes as input in order to make it faster.

Sources : Pharr, Jakob, and Humphreys 2016

Figure 1: Construction and splitting of a k-d tree

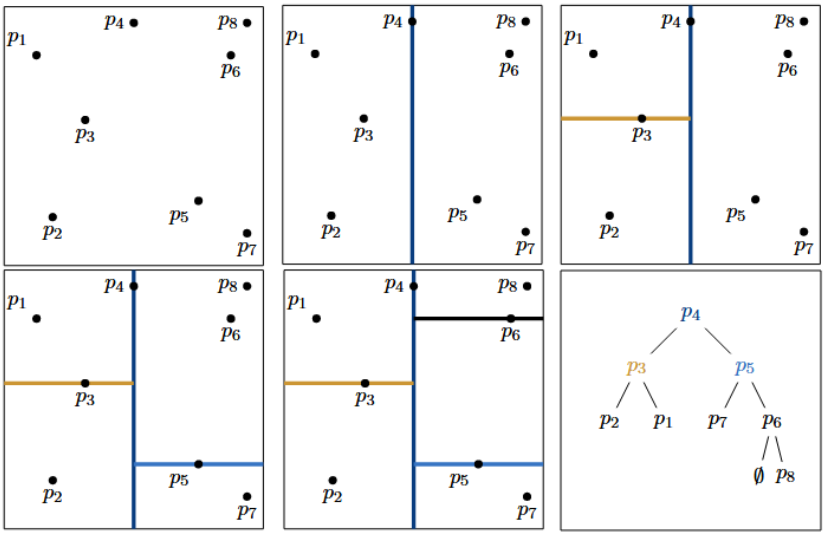


Image from the paper : Skrodzki 2019

3 Method Overview

3.1 Introduction to the experiment

Firstly, we'll start with the basic concept of the experiment. We're going to use 4 scenes which contain an increasing number of primitives in order to check the K-d tree's complexity. We will store the primitives in a vector (Dynamic C++ data structure) in order to use it as input in our trees.

Furthermore, we're going to create 11 trees for each scene with the following population percentages : The first tree will contain all the primitives (100% of the population) while the rest will contain subsamples of the initial population , more specifically : 50% , 40%, 30% , 20% ,10% ,5% , 4% , 3% , 2% and 1%.

3.2 Tree Manipulation

After the trees have been constructed we'll have to empty them from primitives in order to have faster traversal for the rest of the methods in the experiment.

In addition, while having the initial primitives stored in a vector , we're going to fill the leaves of the trees with pointers pointing to the locations of the primitives in the starting vector by checking the node's AABBs to maintain the fast traversal of the trees.

Lastly, we're going to compute the traversal cost of the trees in order to check how fast a ray can reach the leaves of a tree and find out if there's an intersection with the primitives.

3.3 Conclusion of the experiment

The experiment will be completed with the accumulation of data that we're going to extract from the trees and perform statistical

analysis on them in order to find out the proper population percentage of primitives that we can take so as to reduce the initial size and maintain the speed of the K-d tree accelerator.

4 Implementation

4.1 Introduction to the program

First of all , the experiment will be implemented in C++ , on a program that will run on a custom-built renderer made by the Computer Graphics Lab of AUEB. Two more files will be added to support the K-d tree accelerator.

We will retrieve the primitives from 4 scenes (in .obj form) which are named (from smallest to largest primitive size): living room 1 , bmw , bistro , dragon. The following steps are executed for each scene.

4.2 Vertices

We take the data in the form of points in the 3-D space , which if connected , will create the respective primitive and that's why they're called vertices.

Furthermore, we'll store them in a vector and , with the usage of the shuffle method, we'll "shuffle" them , removing duplicates in the process, in order to see if the trees have the same "behavior" with this dataset.

Lastly, we find the initial AABB that contains the whole scene with which we will build the trees.

4.3 Trees

The trees are supported by 3 new added classes : KdTreeNode which simulates the k-d tree nodes , KdTree which simulates our accelerator and Bounds which simulates the AABBs of each node and the scene's. KdTree also contains some basic methods which

will be used later in the experiment and have the following functionalities :

- Build Tree
- Empty Tree
- Populate leaves of the tree with pointers
- Computation of the tree's cost

which will be explained in more detail later in the thesis.

4.3.1 Build Tree

Starting from the tree construction , the KdTree class contains variables for the tree's depth in order to have the capability to perform tests with whatever depth we desire. For the experiment , we used a depth of 50 levels , which means that the tree will have 2^{50} nodes at most.

The splitting of the nodes happens thanks to the heuristic function SAH ,as mentioned previously, which functions as follows: We compute a cost for each axis and each point that is in the boundaries of the examined node's AABB in order to find a good spot for splitting the AABB (also known as split). The following formula is used:

$$C(n) = traversalCost + isectCost * (1 - eb) * (pLeft * nLeft + pRight * nRight) \quad (1)$$

with pLeft and pRight being described by the following formulas

:

$$pLeft = leftSA * invertedSA \tag{2}$$

$$pRight = rightSA * invertedSA \tag{3}$$

Where :

- C(n) = node's cost
- traversalCost = traversal cost of a node
- isectCost = intersection cost of a ray with a node
- eb = empty bonus , variables that helps with the formula's error
- pLeft = probability to have a good split in the left child
- pRight = probability to have a good split in the right child
- nLeft = integer that starts from the beginning of the vertices vector.
- nRight = integer that starts from the end of the vertices vector. Helps with finding the good split faster.
- leftSA = left child's Surface Area
- rightSA = right child's Surface Area
- invertedSA = $\frac{1}{SA}$
- SA = Examined node's Surface Area

With the conclusion of the procedure described above , there's a chance that no good splits were found so this happens : The procedure is repeated for two more times, if there is no good split then the tree building method stops and a new leaf node is created. In the opposite case , the method with the new split creates two new AABBs and splits the data that will go to the children as follows : Every point that is before the best offset will go to the left child and the rest of the data will go to the right child.

Finally, leaf nodes are also created when the tree has reached it's maximum depth that we have set , or if it has a number of primitives ≥ 1 and ≤ 32. This helps in eliminating empty leaf nodes.

4.3.2 Empty Tree

Moving on with the experiment, we will have to empty the tree from the vertices that it contains as this will help us later with the other two methods. This happens because by emptying the tree we will be able to traverse it faster.

4.3.3 Filling the leaves with pointers

The next step is the filling of the tree leaves. Using the KdTree method we fill the leaves with integers contained in vectors which indicate the index of the vertex in the starting vector.

For faster traversal of the tree we have done the following : The method takes the index of a vertex as an argument and the if the examined node isn't a leaf it checks if the vertex is inside the boundaries of the AABB of the node's children. If it's in the boundaries of the left child we traverse left otherwise we traverse right.

4.3.4 Computation of the tree's cost

Finally , after the above methods have finished we come to the final part of the trees, the computation of the tree cost. KdTree has a special method for this purpose that functions as follows : We begin a bottom-up algorithm , going from the leaves to the root, and we check each node.

The formula that's used for the cost is described by the bidirectional equation below and it's computed recursively :

$$C(n) = \begin{cases} traversalCost + WL * C(nLeft) + WR * C(nRight) & , internal \\ isectCost * nPrims & , leaf \end{cases} \tag{4}$$

With WL and WR being described by the formulas :

$$WL = \frac{leftSA}{SA} \tag{5}$$

$$WR = \frac{rightSA}{SA} \tag{6}$$

Where :

- $C(n)$ = Cost of node
- $traversalCost$ = Traversal cost of each node
- $isectCost$ = Intersection cost of a ray with a node
- WL = Weight of left child
- $C(nLeft)$ = Cost of left child
- WR = Weight of right child
- $C(nRight)$ = Cost of right child

- $nPrims$ = number of primitives
- $leftSA$ = Left child Surface Area
- $rightSA$ = Right child Surface Area
- SA = Examined node's Surface Area

4.4 Extraction of statistical values

In the final step of the experiment we extracted some statistics which will help us determine the conclusion of this thesis. The following statistics were extracted from each scene for each tree :

- Primitive population percentage of the tree
- Total number of nodes of the tree
- Number of internal nodes
- Number of leaf nodes
- Number of leaf nodes with 1 primitive
- Number of empty nodes (0 primitives)

5 Conclusion

At the end of the experiment , our goal was to find the best possible population percentage of primitives that would have the same results in time and cost as the initial primitive population. For this purpose, we created the following tables:

5.1 Primitive population and levels

The levels that were examined for this experiment were 50. This means that every tree had 2^{50} nodes at most. The number of primitives was increasing as seen in figure 2 in order to obtain a better sample from large scenes like dragon. The experiment could bolster larger levels if we had stronger hardware.

Figure 2: Tree levels and initial population of each scene

Column1	Column2
Tree levels	50

Column1	living room 1	bmw	bistro	dragon
prim numbe	429807	2378178	3139827	5545806

5.2 Tree cost

We came to the conclusion that the SAH costs were increasing in ascending order because the less primitives we had the larger the cost.

Figure 3: Costs for every tree

	SAH Costs			
Percent	bmw	bistro	Dragon	living roo
100	20.3705	22.3338	29.94	13.379
50	21.8756	23.1208	32.0152	16.2415
40	23.3739	25.8116	33.6649	18.0391
30	26.2615	27.2772	34.975	20.7466
20	28.2949	28.5868	37.7269	22.5436
10	30.2203	30.1757	39.8926	25.007
5	31.6427	32.141	41.9352	27.7984
4	32.889	34.0449	43.0366	29.9927
3	35.2765	36.8446	45.2008	32.8045
2	37.9099	39.6257	47.5889	34.4543
1	39.3723	42.5385	49.1755	36.5801

5.3 Tree building time

The table below shows the times that each tree of every scene took to build. The time as we lower the population percentage is decreasing as we have less primitives to take care of.

Figure 4: Build time for each tree

	Build Time	in seconds		
Percent	living room 1	bmw	bistro	Dragon
100	42	267	382	737
50	23	138	203	405
40	18	115	165	379
30	13	90	120	258
20	9	56	84	182
10	4	27	46	86
5	2	13	21	42
4	1	10	15	32
3	1	7	11	21
2	0	4	7	13
1	0	2	3	6

5.4 Node numbers per tree

In the tables below we recorded the total node numbers of each tree for every scene and the number of levels that each tree requires at most.

Figure 5: Node numbers and levels living room 1

Total	node	numbers	for	LR1
	Percent	nodeNumbers	levels (at most)	
	100	50409	16	
	50	23710	15	
	40	37836	16	
	30	23100	15	
	20	31494	15	
	10	18630	15	
	5	15967	14	
	4	11630	14	
	3	8868	15	
	2	7051	13	
	1	5750	13	

Figure 6: Node numbers and levels bmw

Total	node	numbers	for	BMW
	Percent	nodeNumbers	Levels(at most)	
	100	95684	17	
	50	95566	17	
	40	82932	17	
	30	134353	18	
	20	88615	17	
	10	57849	17	
	5	45630	16	
	4	38245	16	
	3	37297	16	
	2	30093	15	
	1	21028	15	

Figure 7: Node numbers and levels bistro

Total	node	numbers	for	Bistro
Percen	nodeNu	Levels(at most		
100	166267	18		
50	161630	18		
40	197065	18		
30	106791	17		
20	164324	18		
10	73402	17		
5	46994	16		
4	81396	17		
3	38016	16		
2	32471	15		
1	25732	15		

Figure 8: Node numbers and levels dragon

Total	node	numbers	for	Dragon
Percen	nodeNu	Levels(at most		
100	307718	19		
50	293976	19		
40	315234	19		
30	175303	18		
20	246390	18		
10	113867	17		
5	115305	17		
4	106568	17		
3	78805	17		
2	61315	16		
1	41397	16		

We noticed that the levels are placed between the interval [13,19] for these 4 scenes with the dragon scene needing the most levels and

the living room 1 scene requiring the least because of the primitives number.

5.5 SAH cost vs build time diagrams

Lastly, we created SAH cost vs build time diagrams in order to find out which percentage is the optimal to use as a subsample instead of the initial size , while having the least (if not at all) possible losses in cost and time. As that was the purpose of this experiment , this part is considered the most important of all the statistical analysis.

Figure 9: Living room 1 - SAH Cost vs build time

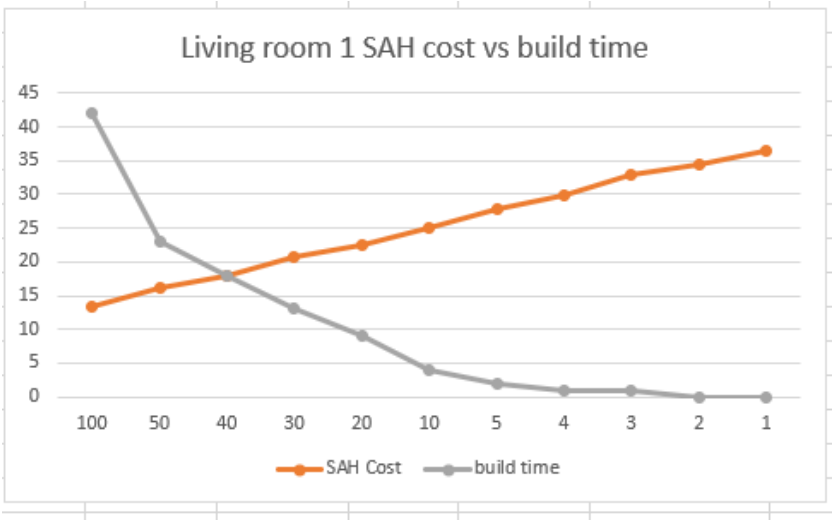


Figure 10: BMW - SAH Cost vs build time

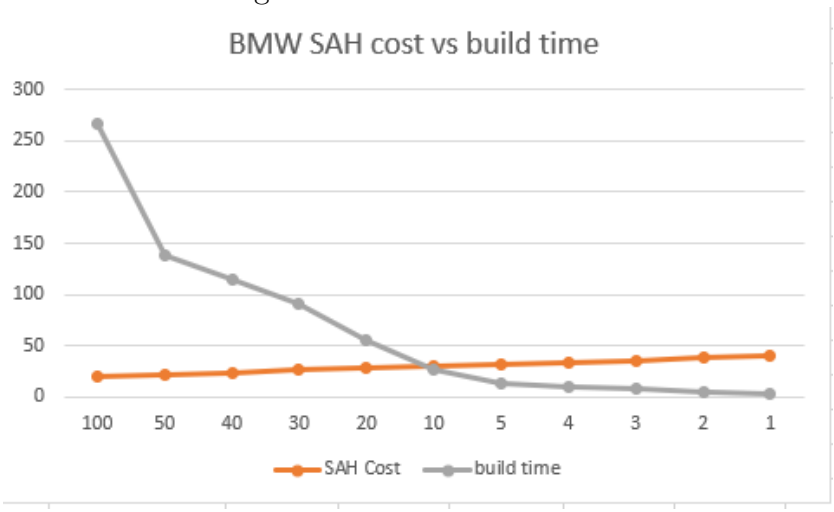


Figure 11: Bistro - SAH Cost vs build time

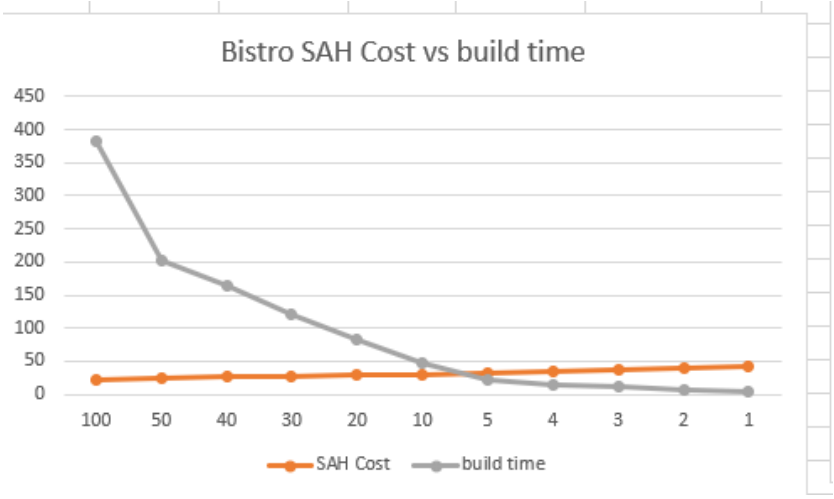
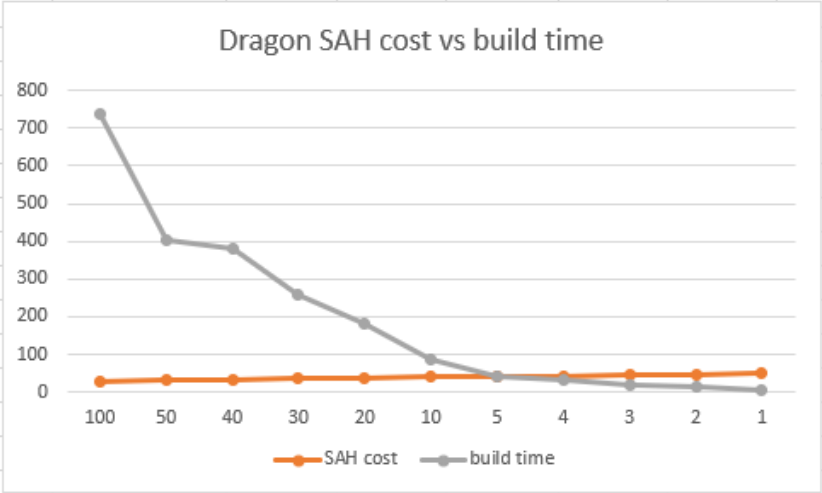


Figure 12: Dragon - SAH Cost vs build time



5.6 Final outcome

In conclusion , by carefully observing figures 9 , 10, 11 and 12 we noticed that the best possible percentage that we could use without having many losses in time and cost was 50% of the initial population because that's where we had the greatest downfall in time with the least cost increase. The rest of the percentages had very little time decrease and since the cost is gradually increasing we discarded them.

6 Bibliography

References

Pharr, Matt, Wenzel Jakob, and Greg Humphreys (2016). *Physically based rendering: From theory to implementation*. Morgan Kaufmann.

Skrodzki, Martin (2019). "The kd tree data structure and a proof for neighborhood computation in expected logarithmic time". In: *arXiv preprint arXiv:1903.04936*.

7 Acronyms

- SAH = Surface Area Heuristic
- AABB = Axis Aligned Bounding Box