OPTIMIZING BOUNDING VOLUME HIERARCHY GENERATION BY USING GEOMETRY SIMPLIFICATION AND LEVEL OF DETAIL ON GPU

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF INFORMATICS INSTITUTE

OF

MIDDLE EAST TECHNICAL UNIVERSITY

BY

SEMİH KEKÜL

# ABSTRACT

OPTIMIZING BOUNDING VOLUME HIERARCHY GENERATION BY GEOMETRY SIMPLIFICATION , LEVEL OF DETAIL AND LBVH METHODS WITH CUDA

Kekül, Semih

Supervisor :

May 2017, tbd pages

In computer games, processing massive number of objects with high-precision collision detection is a challenging task. A well-known approach being used to tackle this important task is using two phases; a broad phase which includes bounding volume hierarchy(BVH) generation for preventing brute force collision testing among all the objects against each other and narrow phase for more precise collision detection among the objects filtered by broad phase. Also, a relatively new approach used is to use Graphical Processing Units(GPU) instead of Central Processing Units (CPU).

In this thesis, a geometric simplification method([]) is and improved by using Level Of Detail (LOD) optimization on remesh calculation and other parallel algorithm for broad phase collision (LBVH[]) is used for generating BVH structure.

Keywords: GPU , Level Of Detail Optimization, Collision Detection, Geometry Simplification

# ÖZ

GEOMETRİ SADELEŞTİRMESİ VE SINIRLAYICI HACİM AŞAMALARINI BİRLEŞTİREREK GRAFİK İŞLEMCİSİ ÜZERİNDE ÇARPIŞMA ALGILAMAYI İYİLEŞTİRME

Kekül, Semih

Supervisor :

May 2017, tbd pages

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Keywords: GPU , Level Of Detail Optimization, Collision Detection, Mesh simplification

# **ACKNOWLEDGMENTS**

To

**TABLE OF CONTENTS**

[**ABSTRACT**](#_r42rcdrkfh0) **2**

[**ÖZ**](#_cnedllmygmbt) **3**

[**ACKNOWLEDGMENTS**](#_gdyup5zhb6x7) **4**

[**LIST OF FIGURES**](#_gnyupfj9xwg6) **6**

[**LIST OF TABLES**](#_fzqxso29ftcm) **7**

[**INTRODUCTION**](#_mlakjxnb5d1t) **8**

[**BACKGROUND AND PREVIOUS WORK**](#_n0x9o5qwcrtr) **9**

[2.1 Morton Codes](#_ogimb1eje3cx) 10

[**Algorithm**](#_m2dchinsaa3j) **12**

[3.1 MERGING GEOMETRY SIMPLIFICATION AND BVHs](#_ftvp4rmd5dwm) 12

[3.1.1 Geometry Simplification](#_5sa5omt3q7cw) 12

[3.1.2 LBVH construction For Broad Phase](#_5sa5omt3q7cw) 12

[**RESULTS AND DISCUSSION**](#_gac88chtanzb) **16**

[**CONCLUSION AND FUTURE WORK**](#_mbzgrk9bxvwd) **16**

# LIST OF FIGURES

FIGURES

Figure 1.1. . . . 5

# LIST OF TABLES

TABLES

Table 1.1. . . . 5

# 

**CHAPTER 1**

# **INTRODUCTION**

Physics engine is an application providing approximation of a rule based system (mostly real world). When we are talking about a game real world physics is not always applicable, however, real or not the engine should be consistent in itself.

Collision detection is the core of the physics engines. The physical consistency of the game world depends on how consistent the collision system works. If , for example, we use the real world physic rule as a base, if two solid objects penetrates unrealistically, the flow of the game will be influenced catastrophically.

Moreover, collision detection is used for artificial intelligence algorithms such as line of sight and path planning. For instance, an AI entity may check the user with a ray cast. If the ray hits the target, it is seen.

While developing a collision detection mechanism, one should note that physics engine is an approximation not a real system and it’s realism depends on minimizing approximation errors. However, in games, one can not ignore the resource usage of the collision detection mechanism while focusing on fidelity. Because, games not only have a physics engine but also other systems like graphics, AI, network, I/O, etc. As a result, if collision detection becomes a bottleneck of the physics engine (and of the game as a whole) game would not reach necessary FPS limit depending on the platform.

There are many different studies for optimizing the collision detection mechanism. Despite the fact that they are mostly CPU based implementations, in the recent years ; especially with the development of *General Purpose Graphics Processor Unit (GPGPU)* implementations such as *Compute Unified Device Architecture (CUDA)* [5] ; parallel algorithms such as [..][..][..] are in demand.

In this work, two different methods are connected to have a optimized but consistent collision detection engine; the first algorithm is [] which is actually developed for graphic system transforms an input triangle mesh M to simplified M’ and the second algorithm use this M’ to create Bounding Volume Hierarchy BVH.

**CHAPTER 2**

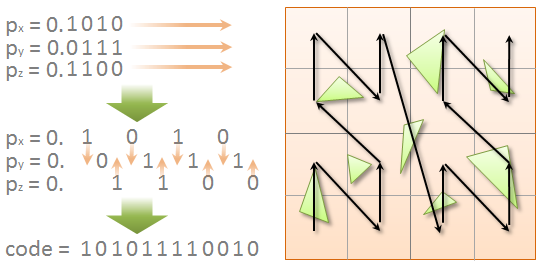
# **BACKGROUND AND PREVIOUS WORK**

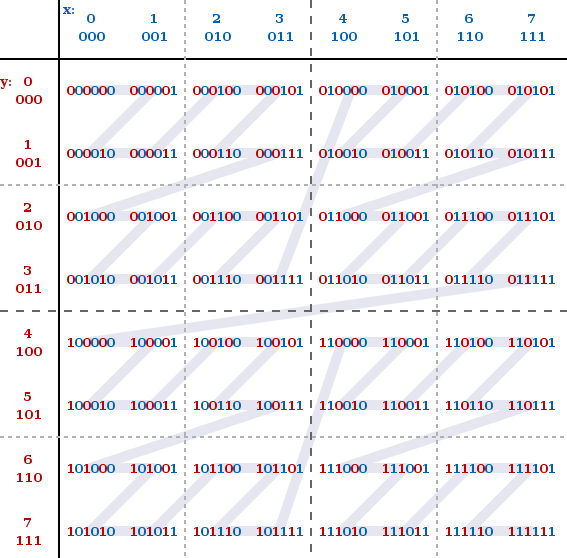
In this work the main approach is connecting two distinguished method which are developed for quite different problems and add and LOD method: first one is geometry simplification and the second one is BVH creation for broad phase collision detection. However, these algorithms use same approach; *Morton Codes* for parallelization of the process. [3] uses it to generate a BVH tree which can be used in collision detection and [4] uses it to generate a K-d tree for simplification of the geometry. [4] also uses a error measuring metric *Quadric Error Metric.*

## 2.1 Morton Codes

Mortoncode (Z-order) is used as a mapping from 3D to 1D without losing the positional relativity of the vertices.

The method is simple; the binary values of the coordinates of the vertex are interleaved and merged with each other (Figure 1). The bit differences between codes are reflected to spatial relativity of the vertices. For instance, if a vertex has 00000001 morton code, its neighbour will have 00000010 morton code and these codes are also neighbours in the tree(Figure 2)





[3] uses Quadric error metrics[] for polygon mesh simplification;

2.2 Geometric Simplification

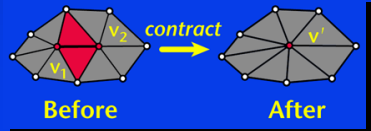
2.4 Quadric Error Metrics

This error metric is used in *Vertex Pair Contraction* operation to measure error. Vertex Pair Contraction is a method of geometric simplification(figure3). The equation of plane P is ax + by + cz +d = 0 where **n** = (a,b,c) is the normal; and the square distance between vertex v and plane P is

(2.1)

So the sum of squared distance of the planes is the error:





In the figure 3; v’ is the new vertex(contraction of V1 and V2) and the error of this operation is the sum of square distances of V’ to the planes of V1 and V2. The parametric plan

2.3 Broad Phase Collision Detection

**CHAPTER 3**

# **Algorithm**

## 3.1 MERGING GEOMETRY SIMPLIFICATION AND BVHs

### 3.1.1 Geometry Simplification

........... createMortonCodesWithCuda ...........

........... sortMortonCodesWithCuda ...........

........... removeDuplicatesWithCuda ...........

........... QuadricsprocessWithCuda ...........

........... mortonIntegralsWithCuda ...........

........... ParallelTreeConstructionWithCuda ...........

........... AdaptiveSamplingAndTriangleMarking ...........

........... Remeshing ...........

### 3.1.2 LBVH construction For Broad Phase

........... CreatePrimitiveMortonsByCuda ...........

In this research, the method written in [1] is going to be used; which is called Linear Bounding

Volume Hierarchy (LBVH). The LBVH creation is started with with generating Morton

Codes of all the objects in the scene and continues with sorting objects according to these

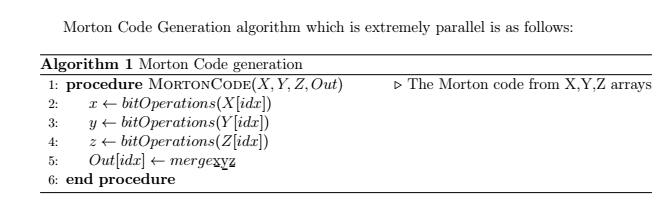
codes using parallel radix sort on GPU. Afterwards the LBVH is created using a top-down

hierarchy on GPU.

Morton Codes will be used to index objects spatially; the Morton Codes which are close

means objects having them are close position in space, vice versa. As a result, in order to

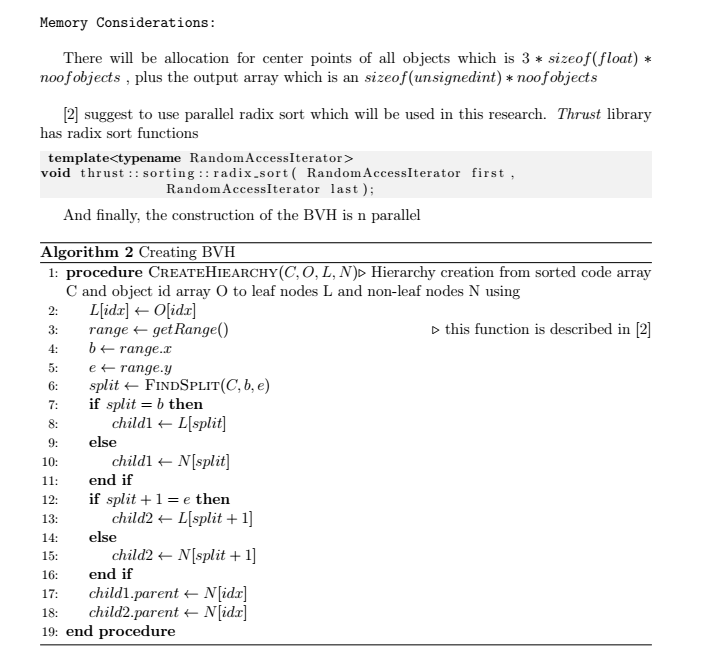
create a spatially sorted object list, it is enough to sort the Morton codes of objects[3].



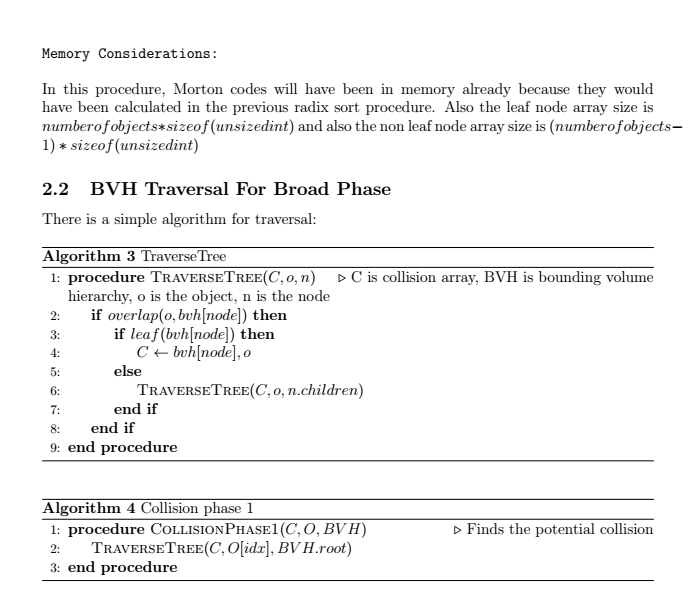
........... sortPrimitiveMortonCodesWithCuda ...........

........... generateAABBsWithCuda ...........

........... generateHierarchyWithCuda ...........

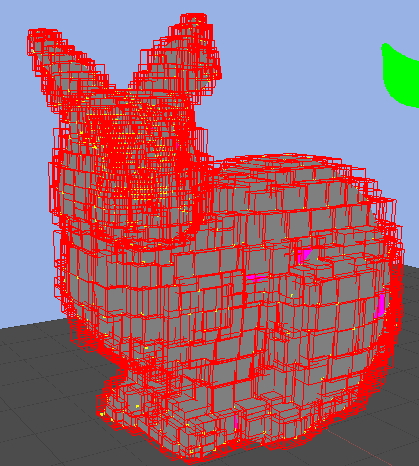


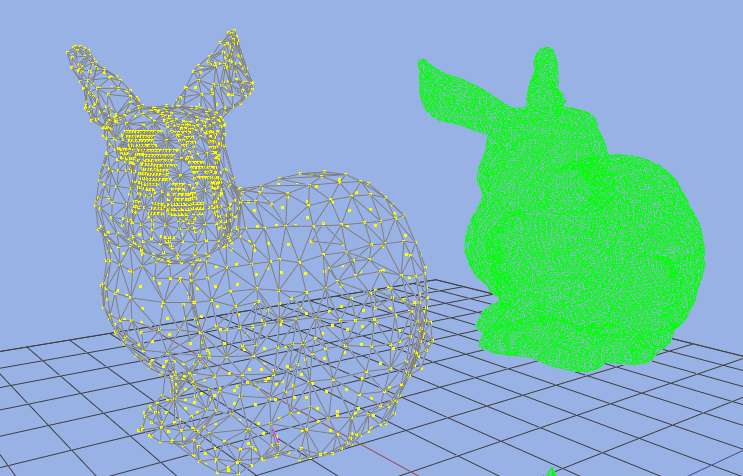
........... calculateBoundingBoxesWithCuda ...........

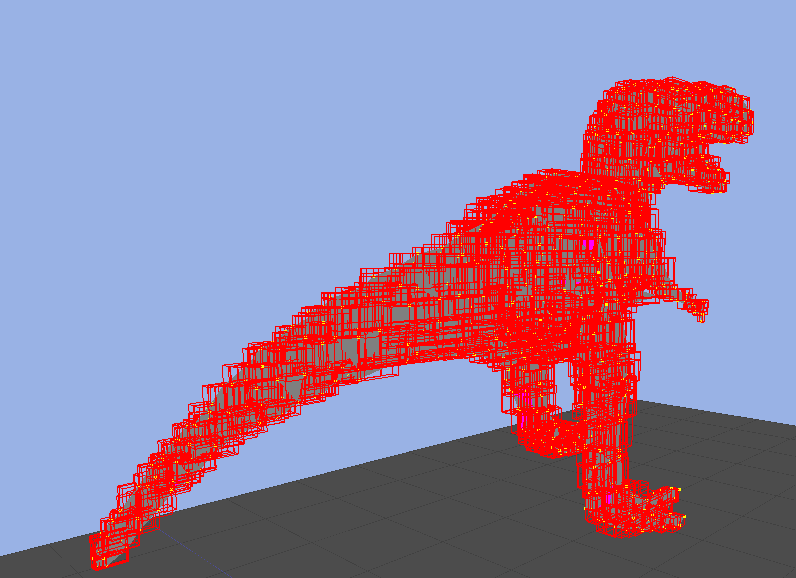
****

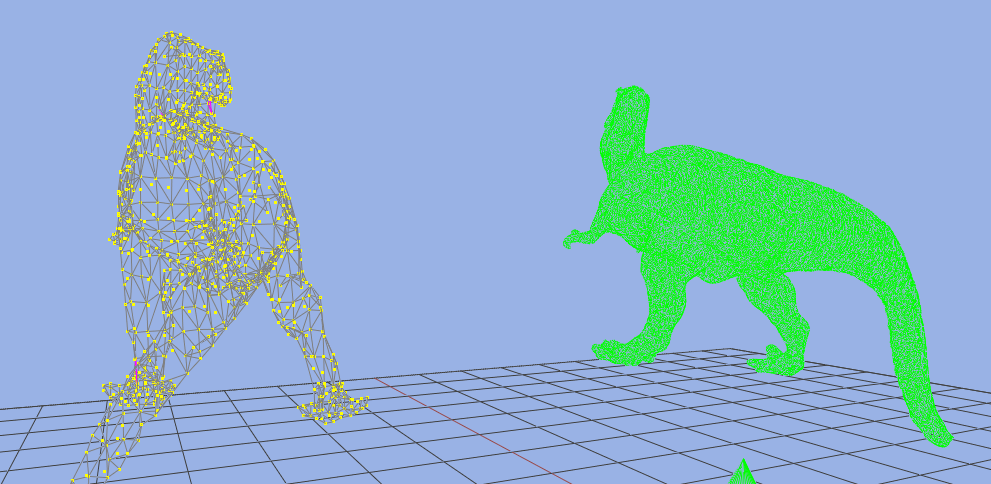
**CHAPTER 4**

# RESULTS AND DISCUSSION

****

****

****

****

**CHAPTER 5**

# CONCLUSION AND FUTURE WORK

REFERENCES

[1] Maximizing Parallelism in the Construction of BVHs, Octrees, and k-d Trees , Tero Karras

[2] Real-Time Collision Detection , Christer Ericson

Sony Computer Entertainment America

[3] Morton Integrals for High Speed Geometry Simplification, Hélène Legrand & Tamy Boubekeur

Telecom ParisTech - CNRS LTCI - Institut Mines-Telecom

[4] Surface Simplification Using Quadric Error Metrics ,Michael Garland,Paul S. Heckbert

[5] HLBVH: Hierarchical LBVH Construction for Real-Time Ray Tracing of Dynamic Geometry , J. Pantaleoni and D. Luebke

[6] Real-time mesh simplifications using the GPU, Christopher DeCoro, Natalya Tatarchuk

[]http://www.nvidia.com/object/cuda\_home\_new.html