

# Comparison of Spatial Partitioning Strategies

Linus Kåreblom

Technical program Academic year 2024/2025

**School:** Österänggymnasiet

**Class:** TE3A

**Supervisor:** Tony Tieu

January 23, 2025

## **Abstract**

Spatial partitioning is a widely used optimization strategy to efficiently optimize objects in space. It can be found in many applications, from simulating galaxies to video games. Checking collision between objects is an expensive operation that quickly becomes a bottleneck as the number of objects grow. This paper compares three different strategies, alongside a naive solution. The aim is to evaluate the use cases for the different strategies. Each strategy was implemented to fit the same interface to make it as fair of a comparison as possible. The results show how spatial hashing out performs every other method when the objects are distributed evenly while quadtrees excel with uneven object distribution.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Theory</b>	<b>3</b>
2.1	Spatial Partitioning . . . . .	3
2.2	Quadtree . . . . .	3
2.3	The Stack and Heap . . . . .	4
2.4	Big O notation . . . . .	4
2.5	Static Grid . . . . .	5
2.6	Spatial Hashing Grid . . . . .	5

# 1 Introduction

When I was developing a small physics engine, I encountered performance issues during the collision detection phase. In this phase the program figures out which objects are close enough to be touching and overlapping. This wouldn't be an issue in a small simulation with few objects, but I was striving for something grand running in real time which made every millisecond count. Due to these constraints I had to find a way to speed things up.

On my optimization journey I encountered the concept of spatial partitioning - an elegant solution to my problem. Diving deeper I discovered the three techniques I'll be comparing in this paper, along side the naive solution. The basic theory of the technique is simply to group objects close to each other to avoid checking two objects on the opposite side of the world. This drastically decreases the number of collision checks performed each frame.

As a consequence of this constraint I had to find a way to optimize this phase. That is what sent me down the rabbit hole of spatial partitioning as an optimization to do query objects close to each other in space. During my research I found out about the three strategies I'm comparing in this paper. I'm doing this paper within the programming course.

## 2 Theory

### 2.1 Spatial Partitioning

Spatial partitioning is an optimization technique used to make programs run faster. It's often found within physics applications. The theory behind this technique is to reduce the number of times one object is checked against another. Suppose we have one hundred objects in our world. The naive way to check each one of these against the others results in a ten thousand checks. Since checks are expensive to make, we want to perform as few of them as possible.

### 2.2 Quadtree

A quadtree is a recursive data structure, meaning it contains pointers to each objects of the same type as itself. It's in the family of tree data structures. The data structure is comprised of nodes. Each node has four pointers to other nodes, could also be referred to as branches or leaves. This strategy works by inserting points into a leaf until the leaf reaches its capacity, then it splits into four equally large sub leaves. This subdivision pattern is shown in figure 1.

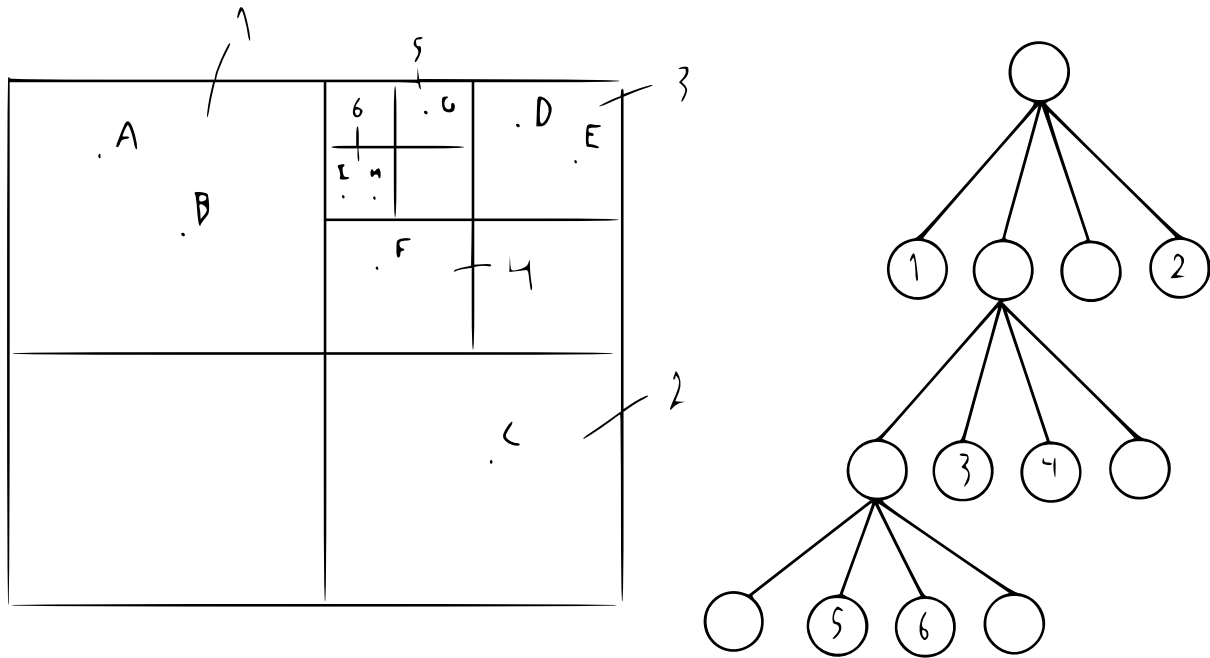


Figure 1: Visual representation of a quadtree.

## 2.3 The Stack and Heap

A program uses two different types of memory, the stack and the heap. When the program starts a fixed amount of bytes are allocated for the program to use, this is what is called the stack. When the program creates a variable it is allocated on the stack, which is done by incrementing the stack pointer by the size of the variable data type. Since a stack allocation is only a single instruction it is incredibly fast. The heap is much larger than the stack but slower because you have to request space within it from the operating system. Allocating is considered to be a slow operation since there's a lot of overhead introduced.

## 2.4 Big O notation

Big O notation is a notation system to describe the time complexity of an algorithm. If an algorithm has the time complexity of  $O(n)$  then that means its time complexity grows linearly with the amount of inputs  $n$ . This could be deceiving due to a hidden constant within the expression. If  $O(n)$  would be written as a function to describe the actual execution time of the algorithm then it would look like this  $T(n) = C \cdot n$ . A  $O(n^2)$  algorithm would be considered worse than an  $O(n)$  one. This assumption is, however, not always correct. If  $n$  is small enough then the  $O(n^2)$  algorithm could run faster due to the constant being smaller than its  $O(n)$  equivalent.

## 2.5 Static Grid

The static grid is built using a simple array. Indexing into the array is an  $O(1)$  operation. The caveat with this approach is you're forced to know how big the area you want the to partition to be before compiling the program. Both insertion and querying only requires a simple index into the grid array which makes this approach super fast.

## 2.6 Spatial Hashing Grid

The spatial hashing grid is built using the same technique as a hash map. The method I used to write my spatial hashing grid was to allocate an arbitrarily sized array. When inserting into the structure I first hash the coordinate of the object, then run a modulo operation on it giving me a number within the array bounds, and then it is inserted into the cell.