# COSC1226 Real Time Rendering – Assignment 2, Pinball

S3728977 - Arone Susau

**Spatial Data Structure**

**Octree Implementation**

Data Structure Used: Octree

The project has no uniform grid implemented.

An Octree was used to test for ball-to-ball collisions. The game space is encompassed by a cube that is 3000 units in length on each axis. This cube represents the space in which the Octree will store data, or in our case the balls. All nodes on the Octree are Octree’s themselves but only the child nodes will contain the data that will be compared against.

As mentioned, each leaf node of the Octree represents a slide of the tree and contains at most two balls each before subdividing further. Leaf nodes do have a minimal size parameter that once reached, removes the two-ball limit from the tree. To determine whether a ball is in the leaf node, the balls radius is used as its cubic dimensions, as seen in the image below by the purple/dark-blue outlined box. The balls actual bounding collider can be seen by the pink outline, which in this instance is a Sphere Collider.

A picture containing diagram

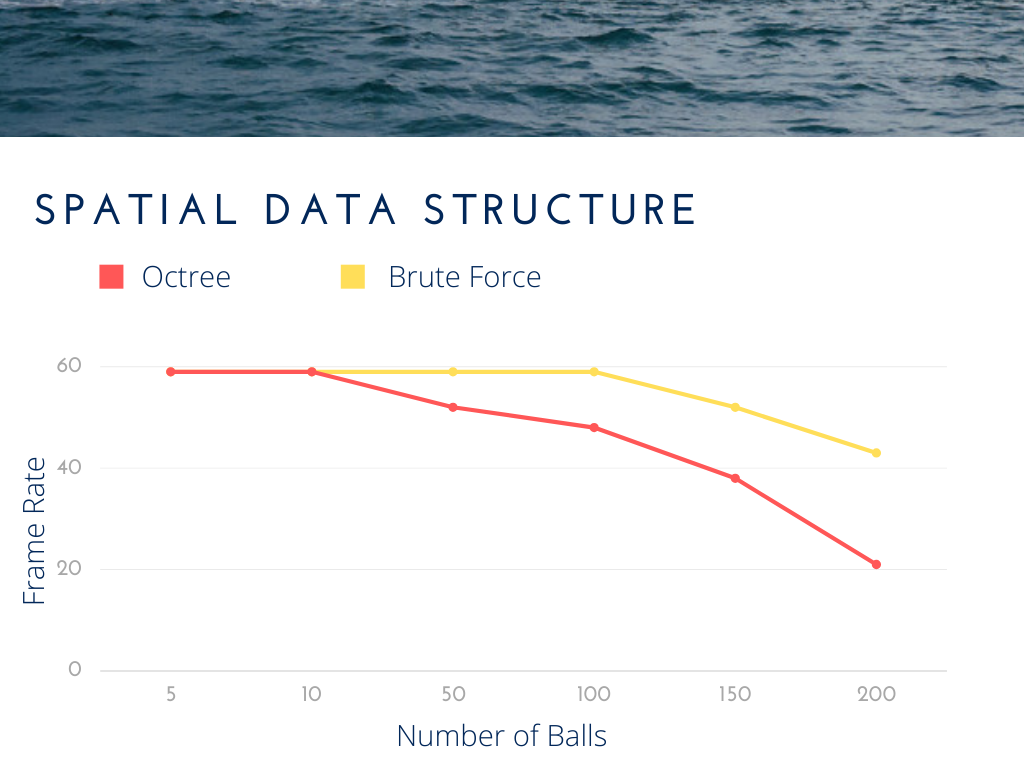
Description automatically generated

The ball is then tested for intersections with the primary top level tree node, if there is a collision then the ball is added to the nodes data if and only if, the number of existing balls in the node is less than the limit. Once the limit is reach, the node is subdivided as seen in the image below where the blue lines are the top-level node, the green the second level, yellow third and so on.

Diagram

Description automatically generated

**Performance Evaluation**



Overall, the Octree performed worse than the brute force approach to collision detection for all cases above fifty balls. At the peak number of two-hundred balls, the brute force approach held a stable framerate at around 42fps while the Octree was at 23fps. Extrapolation of the data would see the Octree implementation run within the 1-2 range as the number of balls increased to 300-400.

**Discussion**

The reason for poor performance of the Octree appears to be in the repeated generate each frame. The entire tree is deleted and rebuilt from scratch which, depending on the size limit, could be quite significant as the branching factor of the tree grows exponentially as the nodes subdivide.

It should also be noted that the poor performance of the Octree regeneration is more to do with the specific implementation in this project. In this project, each ball is represented as a cube which then conducts AABB-AABB like testing to check for intersections. Each ball is added to multiple child nodes if the ball intersects with multiple leaf nodes. Meaning that if a ball resides on the boundary of two major subdivisions and there are several other balls inside of each region, the tree will need to traverse the entire length of its depth to reach the leaf for each branch.

An alternative approach could be to update the branch based on a reference attached to each ball for its parent in the tree. Then instead of deleting the entire branch and rebuilding it from scratch, each ball only needs to review its current parent and recheck for intersection, otherwise the ball can traverse its way up the tree until it finds the leaf nodes it resides in.

**Game Physics**

**Bounding Volumes**

All bounding volumes were either Sphere or Oriented Bounding Boxes (OBB) as the game was simulated in a fully 3D environment.

Sphere to Sphere collision testing is rather simple, all one must do is test that the distance between the spheres centres is less than the combined distance of their radius.

Collision tests between Sphere’s to OBB were slightly more complicated. To begin, the OBB position is subtracted from the Sphere position and an inverse of the OBB rotation is applied to the result. This is done so that the resulting position is in the OBB object space which can then be considered as a regular AABB-Sphere collision test. Each axis of that resulting sphere position is then tested against the OBB dimensions to find if there is an intersection.

**Collision Response**

All collision responses followed a similar formula.

1. Find the normal of the collision.
2. Find the tangent of the collision.
3. Apply elastic collision equations to resolve velocities.

From there, a frictional multiplier was applied to reduce overall velocities if the value was less than one or add an impulse if the value was greater than one.

**Magnetism**

Any object which had a magnetic effect had a bounding radius which indicated the start of its effective distance. The strength of the magnetic intensity would scale linearly as the ball came closer to the object in question. Another multiplier called magnetism is also used to scale the overall strength of the effect. The balls velocities were then either pulled towards an object based on its direction vector to the ball using a position magnetism multiplier or away from the object using a negative multiplier.

**Flippers**

Game flippers were implemented by using two cuboids for each flipper, a parent and child. The parent cuboid was small and position in the world at the flipper’s appropriate location. The child cuboid, which is what is seen, is moved either far left or right of its parents’ position. This is done to hinge the child cuboid onto its parent so then when the parent cuboid rotates, it moves the child in a circular fashion, like a fan blade spinning on its axis.

As the flipper itself uses an OBB for its bounding box, it applies the same reflection of a balls velocity when struck, as with any other sphere to OBB collision. Except, the flippers have a friction value greater than one so that the balls reflection velocities increase on collision rather than slow down.