Realtime Rendering and 3D Games Programming - A2

Peter Bui - s3786794

Opening Statement

This is a report on the CR Level of the report component for Realtime Rendering's 2nd Assignment. We will be testing how effective a Uniform Grid is against an approach that does collision checks, regardless if there are objects colliding or not, with most of the CR features being implemented in this application.

Background

Firstly, my data structure to create a uniform grid consisted of a 2D vector that holds a tuple consisting of a grid's position, the scale of the grid, and a vector list of RTRObjects that the grid is in contact with at each frame. While a 2D array would make for a better design choice, the flexibility of not having to resize an array in case there is not enough memory may be worthwhile. Yet, the performance of a raw array would usually be faster than a regular vector, although not by a significant amount, so performance-wise the array may have been the better initial choice. Nonetheless, this application will proceed with a 2D vector that holds a tuple, as I valued the vector's flexibility over the slight minimal performance of an array when designing this grid. In any case, the grid created is a 21 x 15 sized grid, with each grid space being a uniform sized square(measuring around 1 unit on all edges). While this is the final grid design, I did experiment with a grid that doubled in size, but having smaller grid sizes. That was a 40 x 28 grid with grid sizes being 50% less than the original, measuring in 0.5 units in both height and width. While this design did allow for better collision detection, the performance dropped significantly as there were many more fine collisions the grid must test for. Hence, the reversion to the current grid design, with removing and adding the moving objects to the grid. Although, the drop in performance for the previous grid design can be attributed to having to remove and add all the required collidable objects to the uniform grid at the time, rather than just removing and adding the moving ball objects.

For conducting these tests, they will be done without having any debugging methods active, as that may cause a hit in performance when rendering additional objects in the scene, so only the physical objects in the pinball machine will be tested. Additionally, the following data is taking into consideration that any balls that are stuck at the bottom of the machine will not be destroyed, as it will help keep an increasing amount of objects in the scene, and determine the performance with or without the uniform grid. Furthermore, only collisions done through the sphere objects will be taken into account, as both approaches will have the same number of collidable objects, such as the walls and pegs.

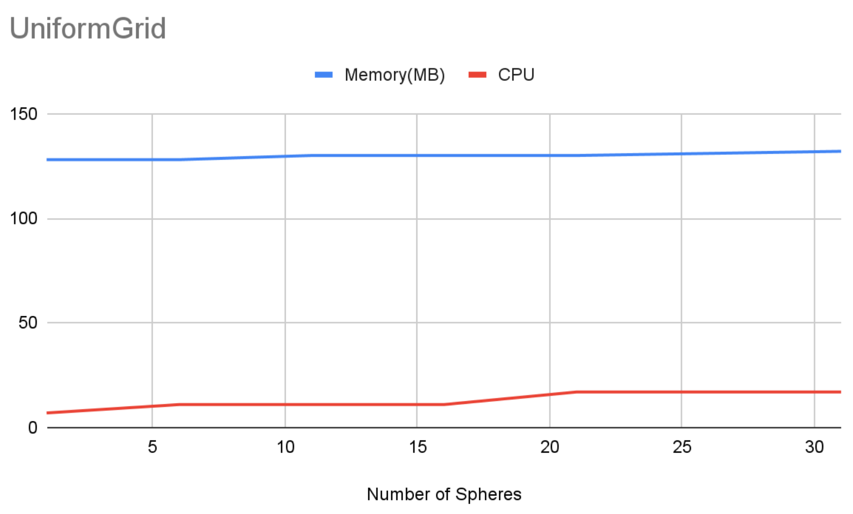
Testing

To test the performance, there will be the monitoring of the amount of memory used in megabytes, and how much capacity of the CPU is required to perform collision testing per a set amount of spheres in the scene.

<u>With Uniform Grid</u>

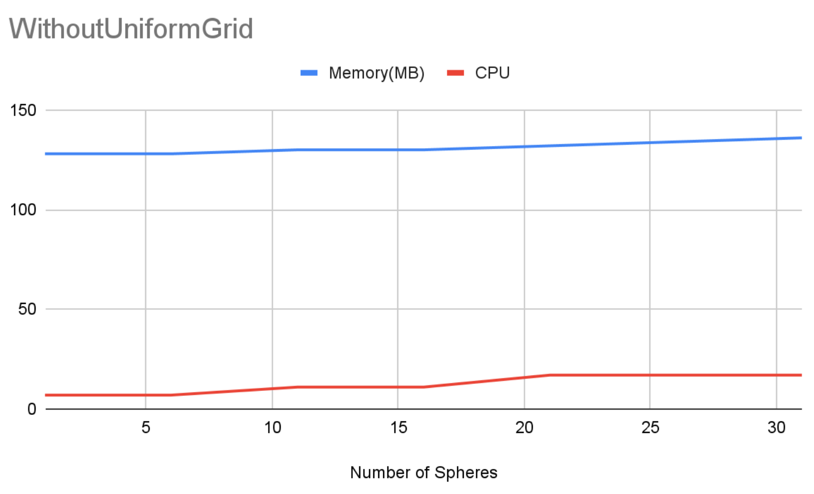
Number of Spheres	Memory(MB)	CPU(%)
1	128	7
6	128	11
11	130	11

16	130	11
21	130	17
31	132	17



Without Uniform Grid

Number of Spheres	Memory(MB)	CPU(%)
1	128	7
6	128	7
11	130	11
16	130	11
21	132	17
31	136	17



Explanation

From the data obtained from testing through an increasing amount of spheres, it can be initially inferred that, with or without a uniform grid, would have similar performances across the memory and CPU usage. Yet, by taking a closer look at the finer details, at around having 6 spheres in the scene, the CPU usage did generally rise for the uniform grid implementation, at around 11% in comparison to the 7% without it. While this may conclude that having a non-uniform grid implementation may result in a slight performance increase with fewer objects, later sphere amounts would say otherwise.

By taking a look at having 21 amount of spheres in the scene, the implementation with the uniform grid used 130 MBs of memory whilst maintaining the same CPU usage, whereas the non-uniform grid implementation used 2 more MBS on average, at 132 MBs. This performance is further pronounced at around 31+ sphere objects in the application, where the non-uniform grid used 4 MBs more worth of memory, which is at around 136 MBs, compared to the uniform grids of 132 MBs. Whilst these numbers seem minor at a glance, they quickly add up by taking into account that each collision test happens per frame, which can be inferred as the non-uniform grid implementation using 4 more MBs per millisecond against the uniform grid implementation. Hence, the data implies that the uniform grid is much more efficient in the long run, by doing collision calculations with less memory overall in comparison to its opposite, at least in the long run.

Although, while doing testing on performance, there were lag issues with the uniform grid, as it seemed the collision computations were affecting the delta time that was being used to move the spheres and other calculations. This implication may be a result of the size of the uniform grid having an effect on performance, as indicated by my earlier attempt at a more complex uniform grid. Hence, size of the uniform grid is a concept that must be taken into consideration, which the non-uniform grid implementation does not have to generally concern itself with.

Conclusive Statement

As a result of the data, it seems with fewer objects in the scene, a non-uniform grid approach is acceptable memory-wise, but when a larger number of objects appear in the scene, a uniform grid approach appears to be much more efficient. Depending on the design of the grid, it is able to perform collision calculations at a more efficient memory rate, rather than having a design that needs to check if a sphere has collided with any object in the scene itself. Hence, a uniform grid design is an efficient data structure that is able to give good performance if there are a large number of objects in the data structure, although the design of the grid itself will have an impact on performance as well.