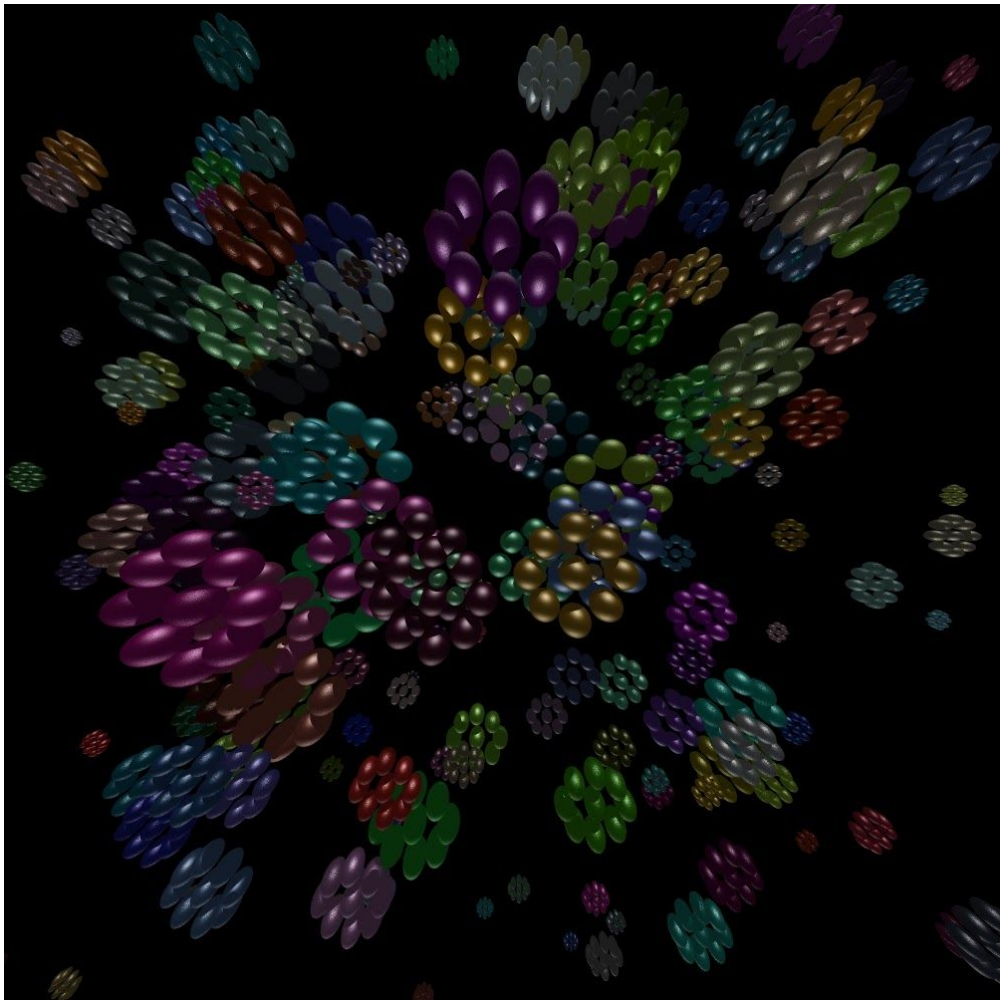
BVH / Multithreading Analysis

To analyze the effects that these two features have on my renders, I generate a scene with 200 clumps of spheres. Each clump contains 9 spheres (1 in the middle of the clump and 8 surrounding it) so the scene has 1800 spheres in total. The spheres are placed randomly around the scene. Each coordinate has a min and max value that they are bounded by. Also, due to the nature of perspective projection, I apply a multiplier to the x and y values based on how far away the clump is in the z direction.

Even though the spheres are positioned randomly, no seed is being used so that the spheres are given the exact same locations each time I run the program to make the comparisons more fair.

Each render was done multiple times to make sure that the results were consistent, but I've only included one set of post-render statistics per render to keep the document concise.



BVH Analysis

16 samples (Multi-jittered) were used for this section.

Without BVH

total elapsed: 630s
BBox hit calls: 0
Shape hit calls: 28800000000
Total: 28,800,000,000

With BVH

bvh tree building took: 16s
total elapsed: 32s
BBox hit calls: 709116646
Shape hit calls: 19280930

Total: 728,397,576

39.5x fewer hit() calls are made with the BVH version of this render and 97% of the hit() calls are on a bounding box instead of a shape. The total render time (including bvh tree building) is 19.7x faster with the BVH.

Multithreading Analysis (With BVH)

512 samples (Multi-jittered) were used for this section.

Without multithreading

bvh tree building took: 16s
total elapsed: 375s

With multithreading

bvh tree building took: 16s
num_threads: 8
slabs per thread: 6.000000
slab size(w,h): (125,166)
total elapsed: 131s

The multithreaded execution is roughly 2.86x faster. It doesn't cut down nearly as much render time as the BVH, but it's still a significant speed up. Because the BVH tree building cannot be parallelized, removing that time from the renders results in 359s for the single threaded execution and 115s for the multithreaded execution. This is 3.12x faster.