Rodolfo Gomez
11/24/15
CSC 461
Game Particles Final Report

Reflection

When working on this project, the first thing that popped into my head was to convert all of the doubles in the programs to floats. Since this is a real-time system and we are not making any calculations that require double precision, it would be wise to switch to single-precision floating point numbers. After that, it was time to split up the data structures in Particle and Particle Emitter to Hot and Cold. Then it was then time to remove the standard template list that was in the program because it was a major performance bottleneck. I got that working and just used the original doubly linked list that was already programed into the system. From there, it was time to implement SIMD into the math library. It was a problem at first to try to find ways of doing 16 byte alignment for Particle, Particle Emitter, and the data, but I figured it out by just overloading the new and delete operators for those respective classes. In the process of trying figure that out, I used a lot of the optimization techniques we learned in class such as RVO, const correctness, passing in parameters by reference instead of a pointer, condensing loops into one, removing functions calls/calculations that were doing nothing, and moving a lot of object construction out of the main loops and into their respective classes' default constructors instead which reduced the amount of construction per cycle. I was then able to implement SIMD for all of the vector math and some of the matrix math operations. I had some issues implementing some of the matrices related to identity, translation, and rotation using the _mm_set_ps functions but it would make more sense to use _mm_setr_ps because we are using Intel processors and the instructions we a pushing into there need to be in little endian hence they need to be in reverse order. In the normal lab computers, I was able to shave off a measly 4-5 milliseconds even with SIMD, but on my home desktop, there was a significant amount of performance improvement of 10-11 milliseconds and this is with SIMD. I am guessing it would have to do with the amount of cache available in my processor and that my GPU is pretty fast also. The use of the Visual Basic 2013 Profiler was pretty good at picking up hot spots in the program that were causing a lot of bottleneck issues. Using that, I was able to pick up calls to some of the default constructors where the data wasn't changing in a loop and moved those out which saw some forms of performance increases. The one thing I wish could get working in time is trying to create some proxy objects for some of the vector multiplications with a scale variable. Overall, these were the techniques I used in trying to optimize the Game Particle System.

Change list

See history on Perforce