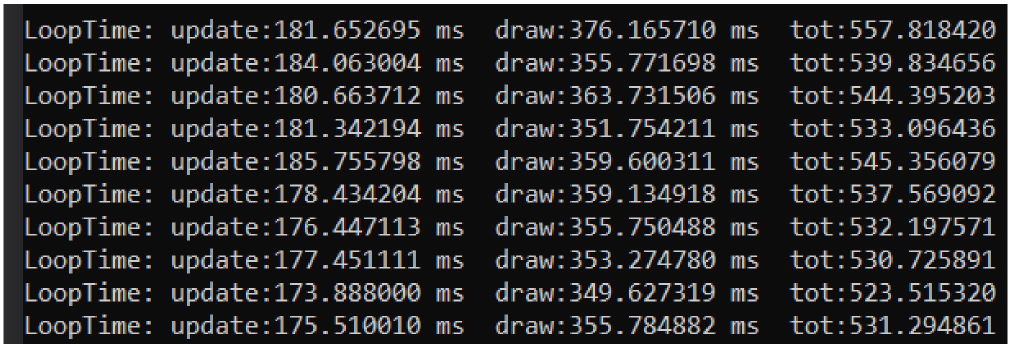
Chien Liang Liu (ID: 1541084)
CSC 461
Professor Ed Keenan

Particles Project Optimization Analysis

This project analysis focuses on the planning process of refactoring the Game Particles system. The average initial or current performance run time of the Game Particles System in debug and release average of about: (Taken from 10 Run Time Samples)

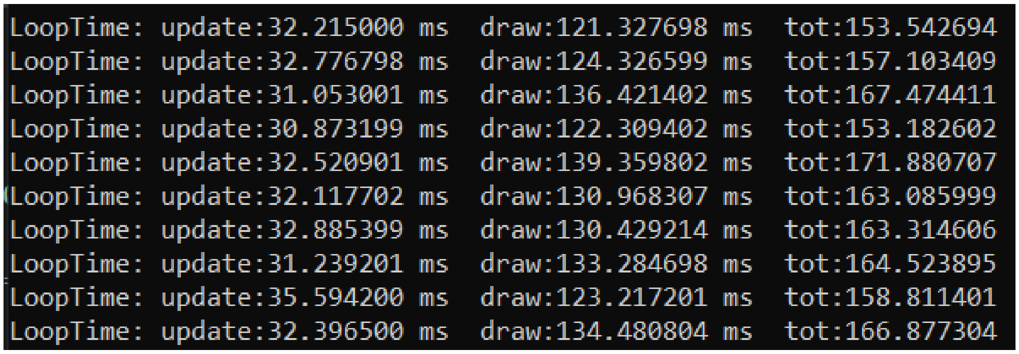
Average Run Times	Update	Draw	Total
Debug:	179.520784	358.0595823	537.5803663
Release:	32.3671901	129.6125127	161.9797028

Debug: (10 Sample Run Times)



```
LoopTime: update:181.652695 ms  draw:376.165710 ms  tot:557.818420
LoopTime: update:184.063004 ms  draw:355.771698 ms  tot:539.834656
LoopTime: update:180.663712 ms  draw:363.731506 ms  tot:544.395203
LoopTime: update:181.342194 ms  draw:351.754211 ms  tot:533.096436
LoopTime: update:185.755798 ms  draw:359.600311 ms  tot:545.356079
LoopTime: update:178.434204 ms  draw:359.134918 ms  tot:537.569092
LoopTime: update:176.447113 ms  draw:355.750488 ms  tot:532.197571
LoopTime: update:177.451111 ms  draw:353.274780 ms  tot:530.725891
LoopTime: update:173.888000 ms  draw:349.627319 ms  tot:523.515320
LoopTime: update:175.510010 ms  draw:355.784882 ms  tot:531.294861
```

Release: (10 Sample Run Times)



```
LoopTime: update:32.215000 ms  draw:121.327698 ms  tot:153.542694
LoopTime: update:32.776798 ms  draw:124.326599 ms  tot:157.103409
LoopTime: update:31.053001 ms  draw:136.421402 ms  tot:167.474411
LoopTime: update:30.873199 ms  draw:122.309402 ms  tot:153.182602
LoopTime: update:32.520901 ms  draw:139.359802 ms  tot:171.880707
LoopTime: update:32.117702 ms  draw:130.968307 ms  tot:163.085999
LoopTime: update:32.885399 ms  draw:130.429214 ms  tot:163.314606
LoopTime: update:31.239201 ms  draw:133.284698 ms  tot:164.523895
LoopTime: update:35.594200 ms  draw:123.217201 ms  tot:158.811401
LoopTime: update:32.396500 ms  draw:134.480804 ms  tot:166.877304
```

The plans to reduce update and draw run times for refactoring the Game Particles project to potentially improve the program's performance through using the optimization techniques learned from the course materials from this class (CSC461). The plans for optimizations outline include:

Include Const & Remove Temporaries

Initialize methods or functions with pass in parameters as const, which allows less prone to error and potentially increasing performance. Remove temporary variables that are not potentially necessary and remove temporary variables through +=, -=, *=, and /= operators. Less variables means speed up in execution of code. Example (X += 5 instead of X = X+5, which creates temporary variables).

Double to Float

Changing variables type from doubles to floats because double has double precision size and float is single precision size, therefore using less memory or space.

Convert Pointers to References

Convert methods or functions parameter pass from pointers to references gives compiler more options, which also removes pointer safety checks to potentially increase performance.

Return Value Optimization (RVO)

Another way of removing temporary variables is to return the results directly. Therefore, changing Name Return Value Optimization (NRVO) into RVO returns can potentially increase performance.

Checking STL (Standard Template Library) & Methods/Functions

Check for unnecessary STL and convert to custom container, because STL can potentially be bottleneck for performance, as well as, check or combine methods or functions into one prevent multiple calls.

Prevent Implicit Conversions

Prevent compiler doing unnecessary implicit conversions, which could slow down execution of code due to the conversions causes more cycles to run. Prevent compiler from implicit conversions by setting methods to private with different types.

Padding & SIMD

Check for data alignment to restructure the data biggest first and most frequent used first. After restructure data for __m128, we can use SIMD to apply faster vector and matrix operation calculations.

Compiler Settings

Changing compiler settings to make the project to run faster. However, according to the instructions do not Exceed SSE4.1, Only use MMX, SSE, NO AVX and VEX instructions.

The above code optimization techniques mentioned above will potentially be used to refactor the Game Particles system. According to the visual studio performance profiler, there is an enormous amount of sqrt() calls approximately 2.2 million within one update and draw run time. The sqrt() is a C++ library function which can potentially slow the performance runtime, especially taking in double types. Therefore, sqrt() can be refactored into custom to potentially decrease runtime. In conclusion, I will use and follow the optimization techniques if possible, to increase the performance runtime of the Game Particle system.