Andrew Richards

Particle System Optimization

11/12/2019

On my desktop PC the Particle system is running at 33ms for update, 39ms for draw, and 72ms total while in release. The loops inside of the Particle Emitter update method and draw method appear to be consuming the most time. Refactoring these loops and optimizing the methods being called within them should result in a large performance increase. These loops iterate 100k times so slight performance improvements will makes a significant difference.

Before any refactoring takes place, all doubles should be converted into floats and all data in each class should be aligned. Each class should be instantiating its data through initializer lists in the correct order rather than in the body of the default constructor. The big four needs to have custom implementation for every method that is used while stepping through the update and draw methods. Compiler knobs will be adjusted for maximizing speed.

Draw is looping through an STL buffer of Particle objects and calling GL methods using Particle data. A handful of math operations are taking place during this loop as well. Some of the GL calls and math operations only need to be called once and will be pulled outside of the loop. The math operations can be converted to SIMD and consolidated into fewer method calls. Temporaries created during math operations can be reduced by overloading and using *=, -=, etc. The switch statements involved in the Matrix math for getting and setting are slow and will be removed as well. Since Particle buffer uses STL and index looping, converting this to a custom linked list and using pointer iteration will increase the iteration speed.

Update is spawning particles and iterating through the STL particle buffer in order to update the data in each particle. It loops through the buffer a second time to add the spawned particles to the buffer. The particles only need to be spawned once when the program is first initialized rather than with each update. After the particles are initially spawned, they can be reused each time the particle system resets. Their values will be reset each time spawn particle is called.

To achieve this an array buffer of particles will be created when Particle Emitter is first constructed. A pointer to a Particle object in the buffer will be set instead of calling a new particle constructor 100k times in the spawn particle method. This array of particles will make the particles contiguous in memory which will in turn make iterating them much faster. Using a custom linked list instead of STL will make the iteration even faster. The second loop which adds a spawned particle to the STL buffer can be removed entirely since STL will no longer be used. This will increase speed since adding an element to an STL container is very slow.

The update loop can be further optimized by optimizing the update method that is called on each particle in the loop. A large quantity of math operations take place in this method which can be improved with SIMD implementation. Temporaries will be reduced by overloading *=, -=, etc. There is a slow sqrt method that can be converted to sqrtf or by taking pow(0.5) instead. When implementing SIMD, an anonymous union between an __m128 variable and the Matrix data will be created and aligned to 16 bytes.

There may be a slight improvement by refactoring the main loop and pulling out the variable declaration. Const will be added everywhere possible and pointers will be converted

to references wherever they can be. Any methods which return a temporary will be converted to RVO by returning an object constructor.