Particle System Analysis

Joey Domino

The particle system currently in use by Mega Awesome games, while functional, is very flawed. I propose making a few changes to the particle system to vastly increase its speed and efficiency. Nearly every file building this particle system can be changed to increase the speed and particle count. Below I will outline the changes I would like to make in each file alongside what I predict will be the percentage increase in speed from the given change. I will be covering the files for Matrix, Particle, ParticleEmitter, VectD4, and the compiler settings. After discussing my improvements, I will outline the actual speed gained from my changes after completing them.

Before jumping into each program, I will mention every single one of them need the big 4 created within them. The default constructors also need to be emptied and told to do nothing to allow easier generation per particle. All doubles in every location need to be changed to floats. Doubles are 8 bytes compared to a float's 4 bytes and no vector is ever going to need the 15 decimal places a double can hold. The 7 decimal places a float can hold is more than enough for anything this system will require. We will never need that amount of precision or size for our particle system. On to the proper list!

Matrix.h and Matrix.cpp hold a lot of storage changes that will impact the system. Every Matrix function handling math equations should also be updated using SIMD functions. This means Matrix's private storage needs to be updated to 16 byte alignment. This will allow us to easily store all of the Vect4D variables with the least amount of empty memory There are also unused I suspect adding both of these changes will increase the speed of the system by 50 to 70%.

Particle.h will require its storage to be aligned just like Matrix. Either pad out the current layout or move variables around to make them line up nicer in memory. I would also recommend moving everything but the previous and next particle pointers to a “cold” storage location to save loading times between particles. The next and previous will be within a much smaller “hot” node to increase speed. Each hot node will point to their cold node to keep all proper data connected. All functions and variables need to be tested for if they’re even used as any bloat needs to be removed. I believe creating this new node layout for all particles will speed up the system by nearly 100%.

Once particle is updated, we need to move on to the ParticleEmitter. This one has a large storage area. Re-align the storage to a 16 byte alignment to best fill the space with the least amount of empty memory. ParticleEmitter also needs to refactor its list to be a linked list to allow faster traversal to each particle node. The constructor needs to be emptied to do nothing on creation. I believe the generation of each new particle can also be updated to build it out quicker. With these changes put in, I predict our system will increase by roughly 5-10%

Vect4D is the last of the files to be updated. The major updates required for this file is the use of SIMD for its calculations. Utlizing intel’s SIMD calculations will drastically increase our simulation speed as it will make short work of matrix math. It just so happens that this program is almost entirely matrix math. Implementing SIMD will increase our system by about 30-40%.

Finally, we’ve come to the compiler setting. There isn’t too much to change here, however there are a few settings we can change to drastically speed up our math computations. Setting Enable Enhanced Instruction Set to “Steaming SIMD Extensions 2” will aid our SIMD commands. Within C/C++, we can set our optimization to maximum optimization (favor speed)

for a boost. Overall, we should prefer fast code over anything else in the settings to keep the simulation rolling. With these setting changes, I believe we can increase our system speed by 30%

If we we're to implement all of the above changes, we would be looking at a ~ 250% improvement with our particle system. It is imperative that we implement these changes as soon as possible to keep ahead of the competition. If we can't keep ahead of the others, we don't deserve the name Mega Awesome Games!

After a long and arduous debugging and coding session, it is safe to say that SIMD and RVO are the saviors of this overhaul. Let's start at the small stuff first. I cleaned up list generation by making a hot/cold linked list. This saved about 3-5ms. Next up was clearing out redundant list iterations. Our new linked list is far faster than these lists. Removing the two or three loops that created and iterated on these lists saved another ~5ms. Next up is storage. Shifting all doubles to floats was a minor increase, but an increase none the less. This net us ~.2ms increase in speed. The same can be said for cleaning up the constructors as much as possible. Less calls on creation saved ~.2ms as well.

Like I said above, RVO and SIMD are the saviors of this overhaul. Before implementing them, I was sitting at about 2-2.5x increase in speed. Once RVO was in, I sat at a safe 3.72x ratio. INSANE! Once I replaced the normal implementation of Matrix X Matrix math with SIMD, I gained another 1.5x increase in speed.

In the end, my ratio ended up sitting at roughly 5.1x times faster than the original. I did NOT expect such a strong improvement. I was aiming for 3x at best, but color me pleasantly surprised! I may need to use SIMD more often…