Thang Tran
CSC 461
11/03/2018

Particle System Analysis

I. Change List:

a. Particle.h

It can easily be seen that this file has bloated structure. It should be split into 2 separate files so that hot/cold structure can be used. Pointers and keys should be in one file, the other contains data. Data alignment is also re-arranged.

b. Matrix.cpp

All computations in this file are done using normal arithmetic operators, hence it is not optimized. It's better to switch to SSE intrinsics in order to reduce number of operators in each function. However, it needs to be done carefully and efficiently in order to achieve the goal. Otherwise the performance is hurt.

c. Vect.cpp

Same as Matrix.cpp

d. ParticleEmitter.h

STL list is used here. STL is usually slower than specialized list. There is a double linked list that is implemented here. It could be used instead of SLT list. This can potentially speed up the program. By converting Particle.h to hot/cold structure, this double linked list can prove to be superior to STL list.

e. Variable Declaration

A lot of variables are declared as double. This is inefficient. float is the better option here. However, it is machine dependent. If the hardware implements double, then using float means conversion is required which means slower performance. On the other hand, float is better. In general, float should be the better option.

f. Pointer Parameter

Generally speaking, passing by reference is preferred. There are quite a few functions that take pointer as parameter. These functions will be modified so that they take reference as parameter.

g. Loop

There are codes that are called inside loop and do not change data every time new loop begins. These codes can be moved out of loop so that they are only called once.

h. General

Possibly there are codes that have no use when the program is running but called anyway. As of now, it's not clear if such codes can be deleted. Once the above changes are done, it will be more obvious.

Thang Tran
CSC 461
11/03/2018

II. Timings

In previous programming assignments (PA), converting bloated data structure to hot/cold structure proves to be the most efficient in term of improving performance. Accessing such big node is unnecessary, especially, most of data in node is vector which could slow down search even more. In PA2, my hot/cold data structure is 8 times faster than bloated structure. As expected when running performance analysis on Visual Studio, 90% of running time is spent on particle drawing and particle updating. This requires a lot of accessing data on each node in the list. By improving ParticleEmitter.h, performance will improve significantly. At the same time, this should take longest to do in term of implementation. Speed expects to increase by factor of 2 here.

Step further into ParticleEmitter::draw, STL list contributes the most to running time. As mentioned above, STL list is inefficient. There is a double linked that is already included in the program, and it should be used instead of STL list. Performance of said list is expected to be 1.20 time faster than STL list.

All matrices and vectors calculations use normal arithmetic operators. SSE intrinsics reduce number of operators. For example, if matrix-vector dot product is done using standard C++ code, it requires 16 multiplication and 12 add operations. On the other hand, if instruction is coded using SSE, it takes only 4 vector multiplications and 3 horizontal adds. Although that is a lot of saving, performance does not increase as much as hot/cold structure conversion. As seen in PA5, such implementation only speeds up by factor of 2 to 4. Hence, for this particular particle system, the improvement can be expected somewhere between 1.3 to 1.4 time.

There are some small details such as changing parameter from pointer to reference, double data type to float data type. They seem to be minor, but changing these detail, in fact, can have impact on the performance. As said above, expectation is low, therefore it is probably around 1.05 and 1.15 time which is not much.

Obviously, looping contributes the most to running time. Any improvement on looping is noticeably better for the performance. Unfortunately, inefficient loop cannot be detected as easily as inefficient data structure. It would take the longest in term of identifying inefficient holes. Expectation is, by optimizing loop, 1.5 time faster.

III. Conclusion:

As of now, if everything in the change list is done correctly and efficiently, the 3 times mark is within reach. It would take approximately 5 days to do everything in the change list. These changes will not dramatically change the structure of the program. However, there might be more things to optimize along the way.