PARTICLE SYSTEM

By

G Hanumanth

Id-1540264

CSC461

hgollapa@mail.depaul.edu

Overview of Course

The optimized C++ course work was to learn all optimization strategies including performance enhancements through extended matrix instruction set, dynamic memory usages, performance related to increasing run-time systems to very large scale, C++ language enhancements and extensions, algorithms, streaming and profiling.

Particle system was my final project to apply all optimization techniques I learnt throughout the course. Re-factored the code non-optimized particle system OpenGL code to make use of efficient algorithmic choices, proxy objects, const correctness, Intel SIMD matrix and vector operations, etc. It was very interesting to witness to code becoming faster for each optimization done.

I have made particle system changes in below three phases

1. Analysis section
2. Log section
3. Reflection section

In Analysis section, I have presented my run time prediction on functions such as Draw(), Update() and Execute(). Here most of the times I have try to reduce the unnecessary matrix computation.

Log section will gives more detailed information about statistic. In this section I have recorded timings for each optimized technique I have used. Reflection section will have summary of final results and things I have learnt and applied in case study. It basically deals with comparison against your analysis section.

Analysis Section:

- In software development the best way to optimize code is to prevent implicit conversions such as from float to double or double to int. The best way to prevent implicit conversions is make all data members and functions private.
- We need to stop use of data type Double instead we can use float. The reason behind this is double will have more precision size than float. For all computations in game engine we can use float instead of double. We can apply this for Vect4D and Matrix classes since lot of variables are defined as Doubles.
- In order to reduce number of cycles per read it is advised that we need to make data aligned. If there are many game objects that should read from CPU then Alignment plays an important role in speeding up the process.
- Reducing number of constructor calls also improves performance of system. This can be achieved by Return value optimization (RVO). It basically calls the constructor when data is ready to initialize. In all functions which are returning objects we can apply this condition.
- One of most important feature in optimizing code is to eliminate temporary variables. For small computation such as add, sub we can use Unary operators such as +=, -=, *=, /=. For high complexity calculation we can use proxy technique which will computes the Complex equation with fewer temporaries.
- Single Instruction Multiple Data (SIMD) will help to increase our performance rapidly. It basically uses special register set for computations such as matrix addition, subtraction, multiplication and division.

Log Section

Date - 11-Nov-2017

Initial Timings:

	Update	Draw	Total
Debug	1166.014404 ms	722.870789 ms	1888.88524 ms
Release	27.2229 ms	79.7339 ms	106.9568 ms

/**********UPDATE 1**********/

Date - 11-Nov-2017
Changes:

- Converting Double to float conversions in Matrix, Vect4D, Particle, ParticeEmitter files.
- Converting to OpenGL double code to float.

Updated Timings:

	Update	Draw	Total
Debug	980.555725 ms	731.281921 ms	1711.837646 ms
Release	17.156250 ms	- 71.034790 ms	88.191040 ms

/********UPDATE 2**********/**

Date - 11-Nov-2017
Changes:

- Remove DrawBuffer STL list.
- Removing unnecessary computation from Draw ().
- Moving OpenGL code and Matrix declaration out of loop in Draw ().

Updated Timings:

	Update	Draw	Total
Debug	454.673370 ms	192.546600 ms	647.220032
Release	8.334164 ms	62.446560 ms	70.780724 ms

/********UPDATE 3**********/**

Date - 12-Nov-2017

Changes:

- Updating Vector and Matrix classes with SIMD code changes.
- In Main (), Moving Matrix operations from loop to outside.
- Adding Align () functionality to Particle.

Updated Timings:

	Update	Draw	Total
Debug	446.538910ms	269.252167 ms	715.791077 ms
Release	6.519125 ms	53.112968 ms	59.632092 ms

/*******************UPDATE 4*********************************/**

Date - 12-Nov-2017

Changes:

- Applying RVO to Vect4D and matrix classes.
- In Main (), Moving Matrix operations from loop to outside.

Updated Timings:

	Update	Draw	Total
Debug	432.138490 ms	240.163457 ms	672.301947ms
Release	5.892775 ms	57.168234 ms	63.061009 ms

/*******************UPDATE 5*********************************/**

Date - 12-Nov-2017

Changes:

- Compiler optimization.
- Removing all unnecessary computations from Update()
- Refactoring repetitive code from Update()

Updated Timings:

	Update	Draw	Total
Debug	156.538910 ms	219.252167ms	375.791077 ms
Release	4.69152 ms	48.213698 ms	52.905218 ms

/********UPDATE 6**********/**

Date - 13-Nov-2017

Changes:

- Removing temporaries, adding constants to variables and functions
- simplifying Matrix calculations in Draw()

Updated Timings:

	Update	Draw	Total
Debug	26.538910 ms	203.192398ms	229.550873 ms
Release	1.775100ms	42.891499 ms	44.666599 ms

Reflection Section:

Last table in above log section gives final optimized values for update =1.775 ms and draw = 42.891499 ms in Particle system. Contrary to my predictions on particle system before development code which I have implemented approach is different. Below is few learning I have recorded during optimization.

1. After initial step in converting all variables from double to float I have faced issue with alignment of data in particle file which in turn leads to run time error. This has been blocker for my other work. But few of discussions (use of Align.h file) on Piazza helped me to get rid of the error. This particular change is not come across my estimate before development.
2. Rather than optimizing matrix computations and other code there is a lot of time I spent working on useless code in which Draw() should be mentioned first. I haven't keep track of this issue in my predictions.

3. Although I have specified proxy technique to minimize the runtime it is not as useful I thought since there is a lot of code which has been removed from calculation such as reducing matrix multiplication of tables has been reduced from 5 to 2.
4. Over all if I can relate this to software development life cycle I can recollect Agile methodology which is helpful for this kind of system.