**Techniques Incorporated Into Game Particle System**

1. **Changed Vector and Matrix classes to use SIMD SSE instrinsics.**
   * Faster mathematical computations overall for particles.
2. **Changed application from being mathematically double based to float based.**
   * Game applications due not need to be so precise.
   * Floating computation is fast
   * Reduces memory usage
3. **General C++ Efficiencies incorporated**
   * const correctness applied to all classes
   * Initializer list used instead of assigning variables inside the body of the constructor
   * Implementing the big four
     + Default constructor
     + Copy constructor
     + Assignment operator
     + Destructor
4. **Rearranged all classes’ data members so that it will use up less space due to padding created due to alignment of biggest data member.**
   * Unnecessary memory usage
5. **Added custom memory system (heap) and used it to allocate particles**
   * Included new and delete overloading
     + Faster than normal new and delete operators
   * Custom memory system was faster than normal heap.
   * 16 byte aligned
6. **Tweaked compiler settings (created custom compiling settings)**
   * Fast floats as precision model
   * Omitting frame pointers
   * Removal of unnecessary runtime checks
7. **Return Value Optimization used to remove unnecessary temporary objects.**
   * A = A+B, changed to A+=B, for all mathematical functions
   * Modified code to not create temporaries and instead immediately return an object.
8. **Cache Conscious Data structure (list) with separation of frequently and infrequently used variables**
   * Changed the list of particles from being an STL list to a custom list that separates frequently used variables from infrequently used variables
   * List was made comprised of a “hot node” and a “cold node”
     + Hot nodes contained frequently used variables and cold nodes contained infrequently used variables.
     + All hot nodes and all cold nodes were contiguous in memory to increase cache hits.
9. **Replaced the TRS transformation matrices multiplications ( T \* R \* S \* V) with a function that returns an optimized TRS transformation matrix.**
   * Prevents multiplying a lot of zeros due to TRS matrix multiplications
   * I did the math and combined all transformations with variables and instead only multiplied the relevant operations within each matrix element instead of multiplying the three matrices together.