6.172 Project 2.2 Final Writeup

Albert Wang

November 27, 2011

(Note: My partner dropped the classes soon after the beta for 2.2 was finished)

There have been several changes to the code since the 2.2 beta was submitted. The three largest changes have been the parallelization of recursive quadtree calls (instead of just parallelizing calls to other functions), changing timestep from being a float into an int, and testing for parallel motion in lines.

Originally, the only parallelization was in the loop to check overlapping intersections, and in the calls to those functions. However, since the result of cilk_for loops are always stored in STL lists, which themselves cannot be used later in cilk_for loops, time was wasted in the beta converting STL lists

to vectors. This was removed and all vectors are now lists, and therefore there is no parallelization of cilk_for loops. However, parallelizing parts of the quadtree isn't as needed now since after a few modifications to counting line-line collisions, calls to quadtree are now parallelized, thereby parallelizing almost everything.

The timestep was originally a float equal to 0.5. However, since operations on floats are slow compared to ints, the timestep was doubled to be 1, and changed to an int. I also changed the code in LineDemo to divide the starting velocities by 2 so that the simulation would not speed up.

I also added code in intersectionDetection.cpp to read the relative velocity and prematurely return NO_INTERSECTION if the relative velocity of x and y was below a certain threshold. This basically allows the program to skip needless computation for lines that are travelling parallel to each other.

I also inlined many functions in IntersectionDetection and CollisionWorld.

I tried writing an iterator for STL lists so that they could be iterated over in cilk_for loos, but decided it was not worth it because quadTree itself is already parallelized. This would still have been very useful for the loop that places lines into buckets. I also tried converting the coordinate system to use integers for faster computation, but found that it was not possible.

I also was going to cache low-level functions like cross products, but that was not possible due to the inputs being floats or doubles. I also tried parallelizing the loops in updatePosition and lineWallCollision but found that the computation inside the loops was so fast that coarsening could not decrease the overall time.

The final time is approximately 5.35 seconds.