6.172 Project 2.2 Writeup

Albert Wang, Kaichen Ma

October 24, 2011

1 4

1.1 4.1

Original Output

```
Number of frames = 4000

1347 Line-Wall Collisions

20276 Line-Line Collisions

88.884 seconds in total
```

1.2 4.2

This is not a problem for quadtree because the Line will just be stored by the inner quadtree node, not be a quadtree leaf. This, however, means that the Line needs to be checked for collisions against each of the node's children (although this probably can be optimized to being just checked against the children that the large Line overlaps).

This method of storing some line segments in inner nodes is not as efficient as the original quadtree implementation because inner node Lines need to be checked for collisions against children lines. However, it is still more efficient than the original implementation because it still reduces the number of collision checks.

1.3 4.3

Quadtree output

```
Number of frames = 4000

1297 Line-Wall Collisions

20094 Line-Line Collisions

18.238 seconds in total
```

This speedup is 4.9 times faster. Since quadtree is supposed to dramatically reduce the number of collision checks, this speedup is expected.

1.4 4.4

The quadtree function was implemented as a recursive function. Each run of the function represents a single node of the quadtree. Instead of using the CollisionWorld variable "lines", the quadtree function passes a vector of lines that it creates to children nodes. When the quadtree function is first run, the quadtree sorts every line that it gets into one of five vectors: a leaf vector if the line can't be stored in a child of the node, or one of the vectors that is sent to one of the node's quadrant children. The quadtree function then checks each of the four quadrant vectors to make sure they have more than 1 line, then instantiates the quadtree function for the child node. Lastly, the quadtree function runs collision checks between Lines in its own vector and Lines in its children.

Each quadtree node has one vector¡Line*¿ variable that stores all the lines. Quadtree nodes will create the vectors for their children nodes. This might be optimized to use an object that is not a vector.

1.5 4.5

Max elements without subdividing.

Found that the best time to stop subdividing quadtrees into children nodes was when the number of Lines in a quadtree was 10.

```
Number of frames = 4000

1310 Line-Wall Collisions

20302 Line-Line Collisions

17.399 seconds in total
```

This is a 5% speedup.

1.6 4.6

Max recursion depth

Maximum recursion depth doesn't seem to matter because any benefits you get from maximum recursion depth are already gained by implementing a maximum ele

1.7 4.7

One part that was optimized was that there are now two functions that manually compare vectors of lines. Since there are some parts of the code that need to compare two different vectors and some parts that compare vectors to themselves, the parts that compare vectors against themselves can be optimized to not recompare Lines that have already been recompared. This can probably be optimized for the former comparisons too so that lines are never compared for collisions more than once in a timestep.

2 5

2.1 5.1

Top six longest functions: 24.55 8.89 8.89 1919921812 0.00 0.00 direction(Vec, Vec, Vec) 23.55 17.41 8.53 1919921812 0.00 0.00 crossProduct(double, double, double, double) 17.20 23.64 6.23 383984338 0.00 0.00 intersectLines(Vec, Vec, Vec, Vec) 6.66 26.05 2.41 95996098 0.00 0.00 intersect(Line*, Line*, double) 3.73 27.40 1.35 95996115 0.00 0.00 pointInParallelogram(Vec, Vec, Vec, Vec, Vec) 3.15 28.54 1.14 287994792 0.00 0.00 Vec::operator-(Vec)

2.2 5.2

Changed detectIntersectionNew and detectIntersectionNewSame functions to save collisions to a list instead of calling collisionSolver directly.

New time:

```
Number of frames = 4000

1265 Line-Wall Collisions

20250 Line-Line Collisions

17.792 seconds in total
```

2.3 5.3

Made the tester into a cilk_for and changed the list into a reducer.

New time:

```
Number of frames = 4000

1265 Line-Wall Collisions

20250 Line-Line Collisions

10.734 seconds in total
```

2.4 5.4

Optimized some parts of code to run faster. Added cilk_spawn to the functions that start the search for intersections.

```
Number of frames = 4000

1227 Line-Wall Collisions

18999 Line-Line Collisions

8.978 seconds in total
```

cilkscreen says no races. Runs are the same time.

2.5 5.5

Span is 337 million instructions

Work is 17986 million insturctions

Parallelism is 53.34

Best maximum recursion depth is 7. Best size of quadtrees is 25 or less. Speed was:

```
Number of frames = 4000

1267 Line-Wall Collisions

20383 Line-Line Collisions
```

```
6.846 seconds in total
```

2.6 5.6

Total speedup from beginning was about 10 times faster from original code.

We want to basically parallelize calls to exterior functions, especially to collisionsolver() and intersect(). We may also look into modifying the intersect function itself to run faster. We currently have problems with iterators because cilk_for cannot use STL list iterators while the output of a reducer_list is a list, and not a vector. Unless there is a way to easily convert from STL lists to vectors, we may not be able to parallelize some loops.

Albert wrote the quadtree, detectIntersectionNew, and allCollisionSolver functions. Kaichen did testing and wrote the lineInsideQuadrant function.