Project Report

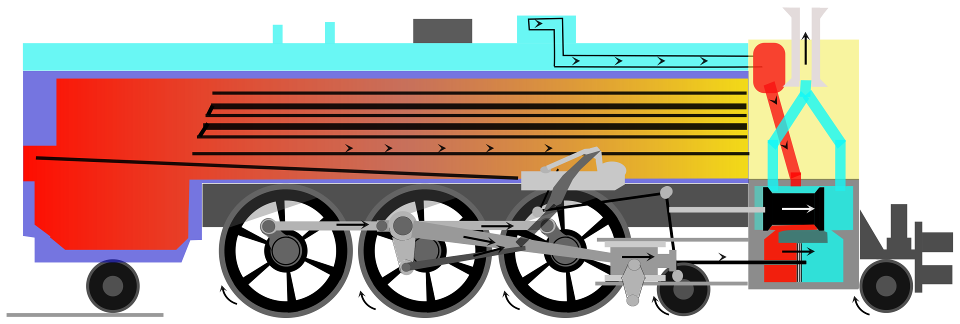
Venkata Dinesh (120050051)
`dineshkota3@cse.iitb.ac.in`

Nitish Chandra (120050071)
`nitish@cse.iitb.ac.in`

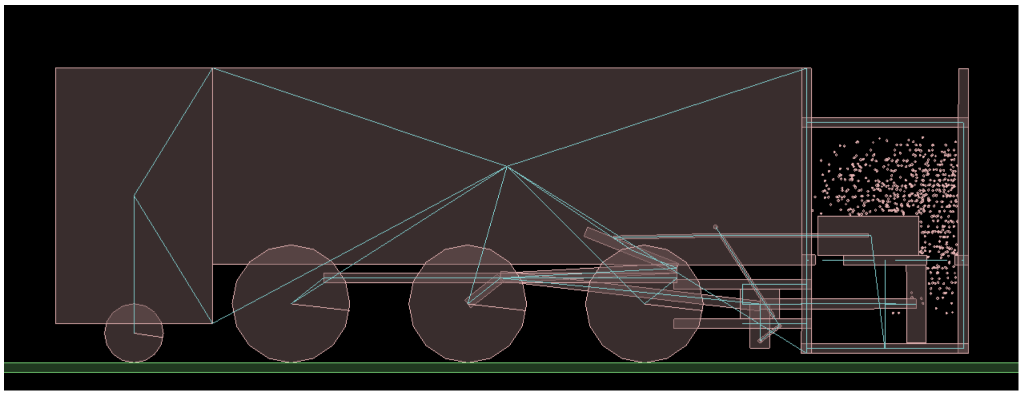
Kota Mahindar (120050073)
`mahindar@cse.iitb.ac.in`

Original design vs Finished design

Original design



Final design



Deviations

1. Unnecessary blocks at the top of train, spokes of wheels have been removed because they don't add to the physics of Steam Engine
2. Outlet for steam are not included
3. Steam has been approximated with small circles constructed using `b2CircleShape`. This is because, Box2D allows only rigid bodies. So, to represent fluids like steam, we need to approximate them with small rigid bodies.

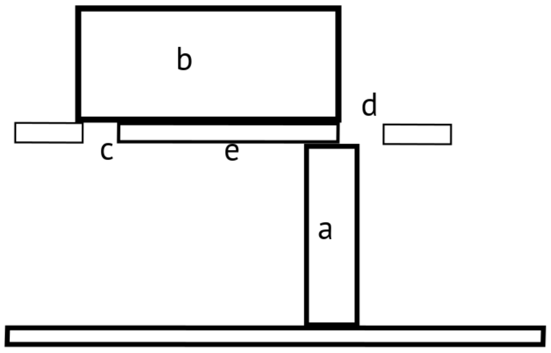
4. We cannot represent temperature changes in Box2D. So, it is difficult to move gas particles produced at the end of engine to the start of engine without temperature gradient. So, steam is created at the front of engine.

Interesting things

We included three keyboard shortcuts to control the motion of steam engine.

- `b` : Apply breaks
- `q` : Accelerate train by applying force on piston
- `c` : To create more steam and thus accelerate the steam engine.

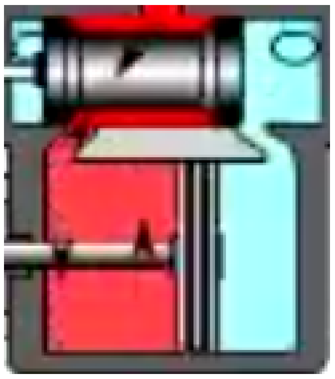
Special arrangement of blocks



Let the above diagram show the arrangement of piston at the front of steam engine. Pistons *a* and *b* move whenever steam particles collide with them. Due to the arrangement of bodies to the left of these pistons, these two bodies always travel in opposite directions. This ensures that whenever piston *a* reaches the right end of block *e*, hole *c* is closed and steam starts entering from *d* making *a* to travel backwards.

Similarly, whenever *a* reaches hole *c*, it is pushed towards right.

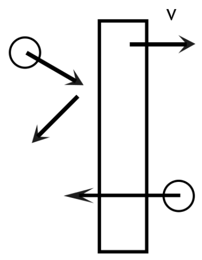
Approximating temperature effect



In the diagram shown above, particles to the left of piston a (shown in red), are of higher temperature and thus impart a larger velocity to piston whereas particles to right of a, are of lower temperature and thus impart very low velocity to piston.

To simulate this effect, we made the following arrangement.

Suppose piston a is moving to right. This means that gas particles on left side have higher temperature. So, potentially, only the particles to left of it can impart considerable velocity to it.



We can simulate the same situation assuming that particles to the right of it don't collide at all. Such an arrangement can be made using `PreSolve` function in `cs296_base.cpp`[2]

Simulating Outlet

In reality, particles which have lost their temperature due to repeated collisions and dissipation of energy go out through outlet. Even without the temperature, we can use the velocities of particles to detect when to oust them. We fixed a minimum value of velocity of particles(w.r.t train) so that if any particle has a speed less than minimum, the program destroys the particle. This is similar to throwing out the particles of lower temperature.

Stacking of objects

We can see that, many objects overlap with wheels of train. This usually leads to collision between overlapping objects (because their AABBs overlap). But, in case of real steam engine, rods lie *in front of* wheels. Since Box2D cannot deal with objects in 3D, we cannot ensure that there is a difference in z component of the objects. So, in 2D, we have to overlap objects and still ensure that they don't collide with each other. We did this using `groupIndex` and `maskBits, categoryBits`[1].

Results of Profiling

We compared the results after profiling our simulation in release profile and debug profile We compared the profiles for iteration number 200000. Highest amount of time is consumed by the function `operation-(b2Vec2 const&, b2Vec2 const&)`.This is due to the fact that it involves a lot of vector calculations.

These results for `Release` profiling were obtained using `-O3` flag.

Following are the profiling results for some functions that have been modified

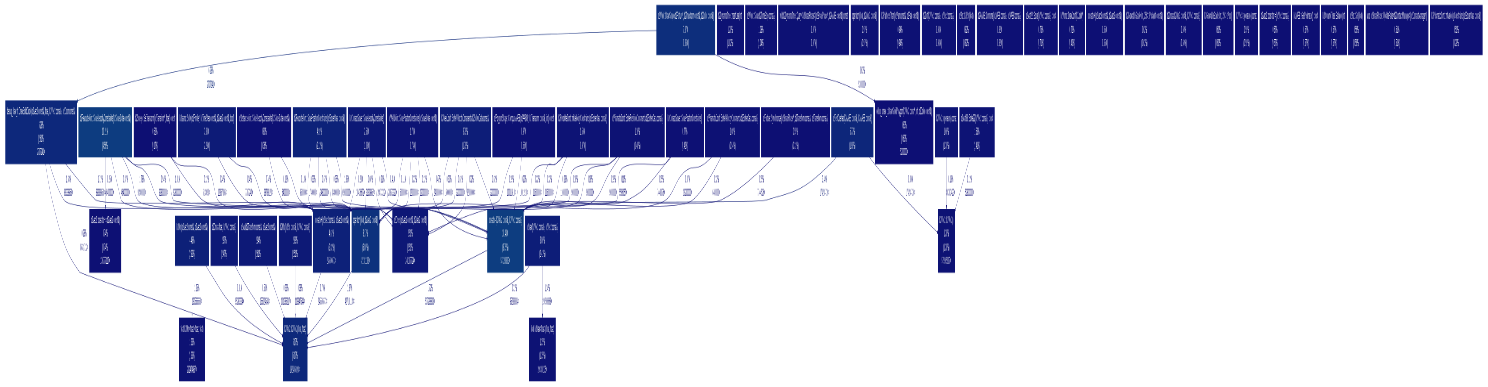
```
0.00 0.00 2 0.00 10.01 cs296::base_sim_t::base_sim_t()
0.00 0.00 2 0.00 10.01 cs296::base_sim_t::base_sim_t()

0.00 0.00 1 0.00 10.07 cs296::dominos_t::dominos_t()
0.00 0.00 1 0.00 30.01 cs296::dominos_t::dominos_t()
```

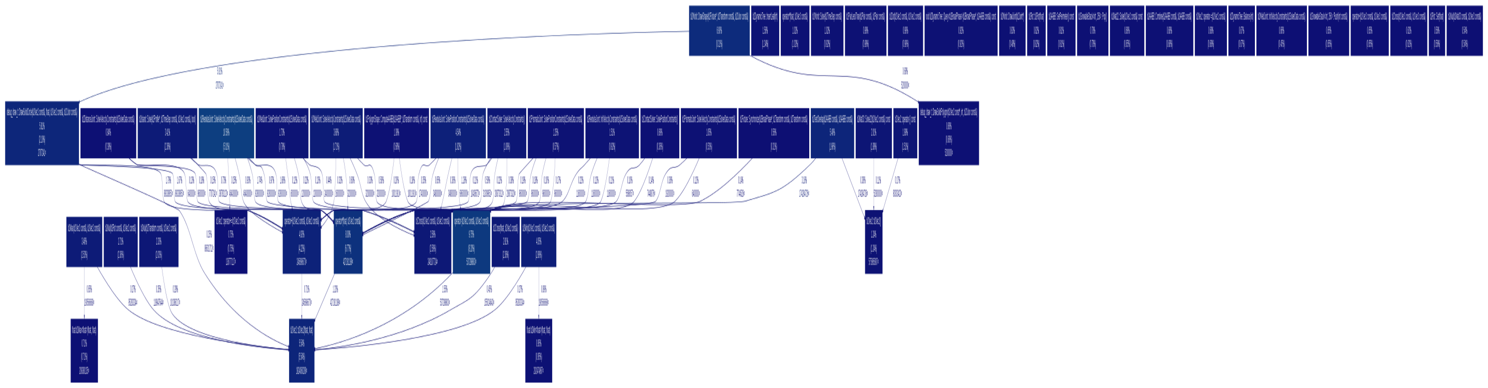
Where first row indicates results with `Release` flag and second row indicates results with `Debug` flag and columns indicate % of total time spent, self time spend in total function calls, self time spent per call in `ms`, total time spent per call in `ms`

We can see that `base_sim_t()` is optimised. But `dominos_t()` where the entire creation of bodies takes place takes more amount of time with `Debug` flag.

With respect to remaining methods, both the profiles have approximately same running times.

This is also evident from Call Graphs

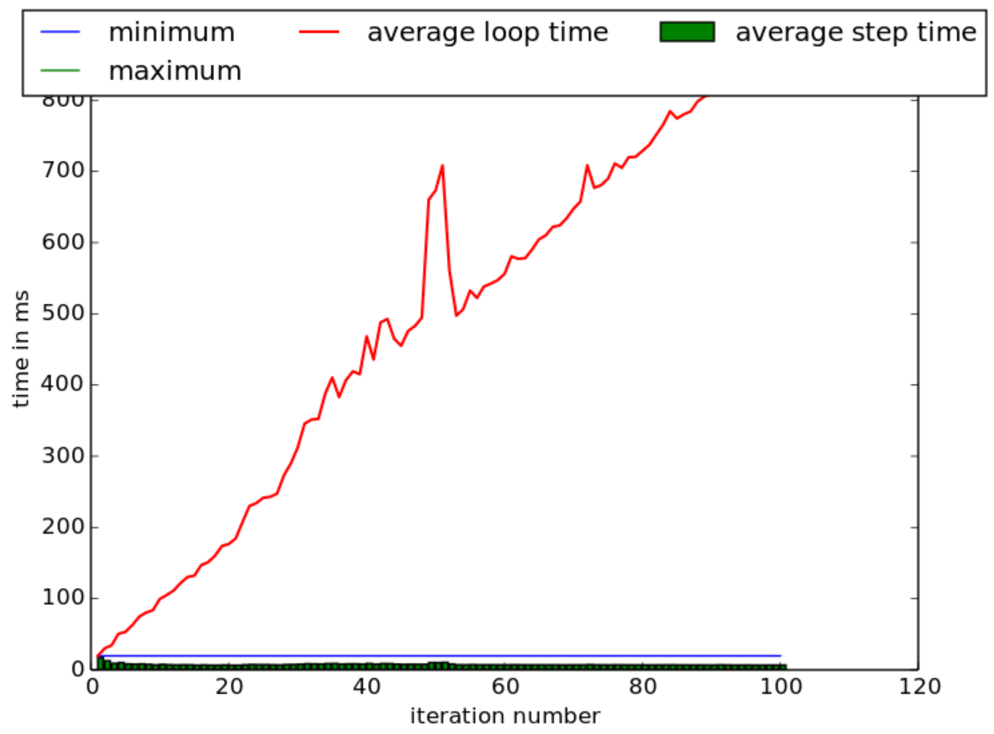




Analysis of the Plots

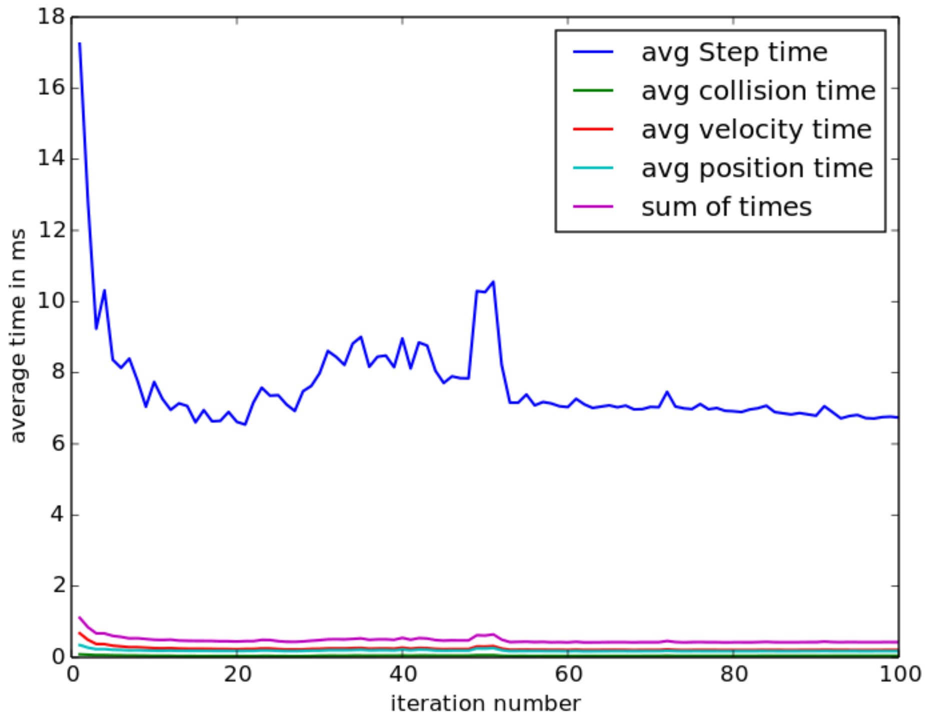
These plots have been generated using 100 iterations and 30 reruns.

g28_plot01



From Plot1 we can observe that **loop time** increases with increasing iteration value. Since with increase in iteration value, we also increase the number of times the loop runs, it is natural for the **loop time** to increase with the iteration value.

g28_plot02



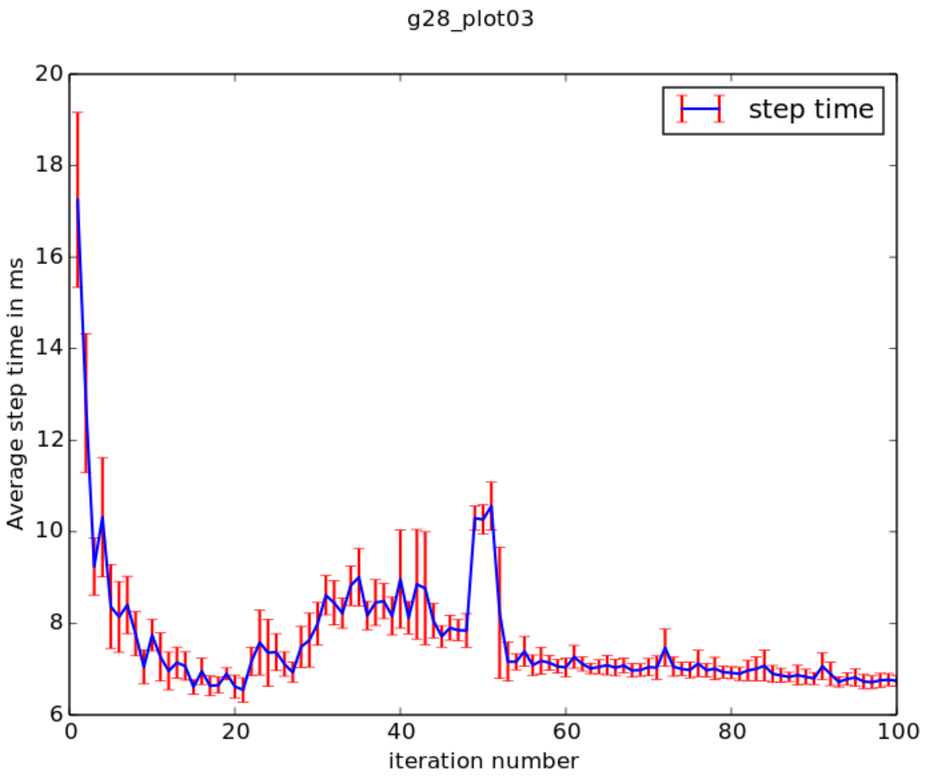
From Plot2 we observe that **collision time, velocity time, position time** decrease with increase in iteration value. This is because collision time is the time taken by program to detect the collision, find the values of collision parameters etc.. To find these values, some subroutines are invoked. When these functions are called, kernel sometimes interrupts the process to allocate time for other processes. So, this interruption time is included in **collision time**. By observing the data, we can conclude that the increase in total interruption time with iteration number is very less.

$$averagecollisiontime = \left(\sum c_i + totalinterruptiontime\right)/v$$

where v is the number of iterations and c_i is the collision time for ith iteration. Since the rate of increase of total interruption time is $<< 1$, and c_i is nearly same except for the instances when collision happens and these instances occur with a gap of many iteration values, average collision time decreases with increase in iteration value.

Similar reason explains the decreasing trend of **velocity time, position time** with respect to iteration value.

We can observe from Plot1 that the average **steptime** is decreasing with increasing iteration number. From Plot2 we can observe that sum of **collision time, position time, velocity time** is strictly less than **step time**, the reason being **step time** includes the time taken to calculate positions, velocities, collision parameters for the next step and also time for which this program is interrupted by kernel. Since all the parameters decrease with increase in iteration value, **step time** also decreases with iteration number.



From the above plot, we can observe that,as the **step time** decreases, error in step time also decreases. The error in **step time** arises because, total interruption time usually varies with time. As the iteration value increases, both the average **step time** and average interruption time decrease, so the error associated with **step time** also decreases.

Drawbacks

If the number of steam particles are too high then there is a distortion among objects in the front part of train and the joint's conditions may not be preserved. Due to their greater number the functions related to the collisions may take more amount of time, so there will be considerable slowdown of simulation.

References

[1] Box2D Manual.

[2] Box2D TestBench OneSidedPlatform.h.