

# CS296 Project: Torpedo Boat Destroyer System

*Group 13*

*Pintu Lal Meena*

120050018 , [pintulalmeena@hotmail.com](mailto:pintulalmeena@hotmail.com)

*Pranay Dhondi*

120050054 , [pranaydhondi@gmail.com](mailto:pranaydhondi@gmail.com)

*Shivam Garg*

12D020036 , [shivam.garg55@gmail.com](mailto:shivam.garg55@gmail.com)

April 7, 2014

## 1 Introduction

This report discusses the simulation of torpedo boat destroyer system made using box2d. We have used keyboard to control the simulation.

## 2 Design Analysis

The most interesting thing about our design is we have simulated such a complex looking machines easily using prismatic joints, revolute joints and impulses in Box2d.

### 2.1 Parts of the Simulation

#### 2.1.1 Moving Boxes



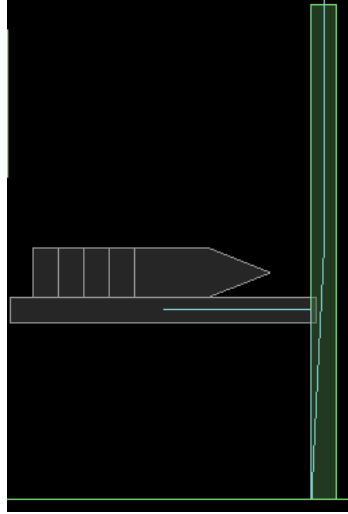
We have used prismatic joints between boxes and horizontal surfaces to push parts of torpedo from surface to lift, to push torpedo from one lift to another and to torpedo launcher.

#### 2.1.2 Torpedo



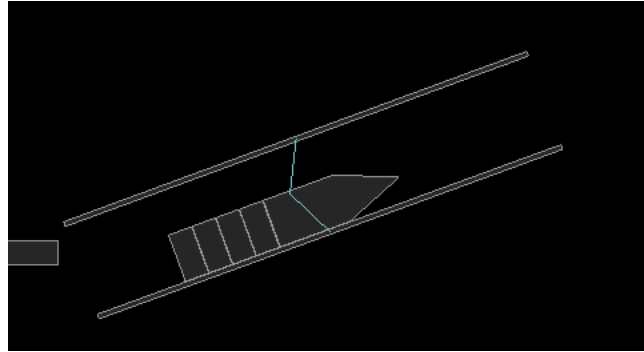
We have made torpedo using 4 boxes and one conical polygon.

### 2.1.3 Lifts



We carry the torpedo up from ground to the launcher using 2 lifts. These lifts slide along vertical walls. We have implemented this using prismatic joints.

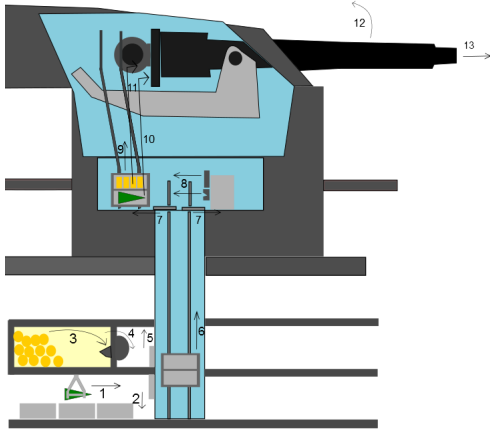
### 2.1.4 Torpedo Launcher



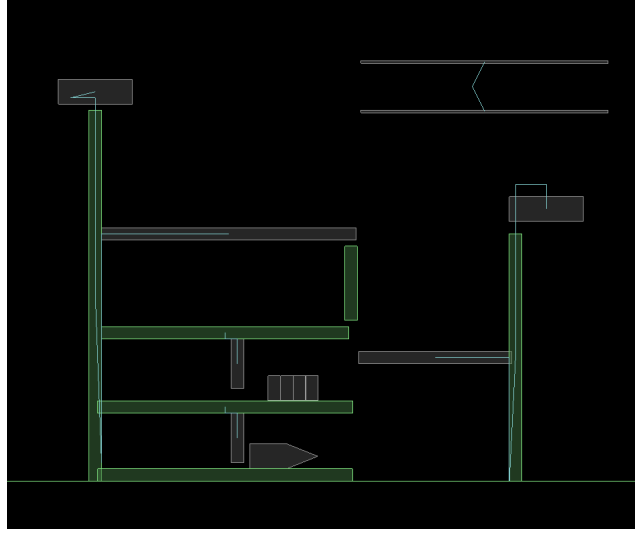
The torpedo launcher is made using two box shapes which independently rotate about a fixed body (using revolute joint). We haven't made any fixture for this fixed body, so it is not visible. Although torpedo is made up of 5 different bodies, but when they reach the torpedo launcher they behave like a single body. This is achieved by increasing friction on the base of the launcher, so that when launcher rotates these bodies do not fall apart. We have used linear impulses to shoot the torpedo. Linear impulse is also used to make the hind section of the torpedo fall down during the motion of torpedo in the air.

## 2.2 Difference between original design and simulation

There is not much of difference between the basic design of our simulation and original design. We have implemented some of the movements in a different manner and added some new things to the simulation also which are discussed below:



*Original Design*



*Design made in Box2d*

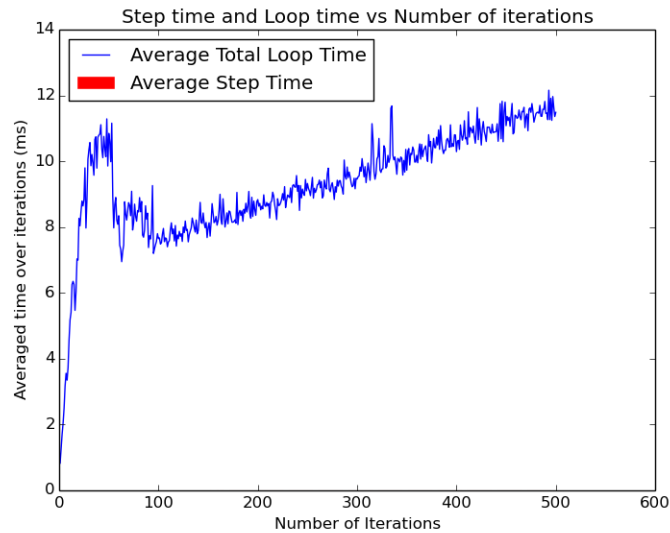
In the original design, cone and the hind section of the torpedo is transferred to lift in a different manner than we implemented it. It is because, the original simulation transfer mechanism was not practically implementable in Box2d. We have kept the hind section and the conical part on the same level on lift whereas in the original design, they were present on two different levels. In our simulation we have also shown the motion of torpedo after being shot (this is not a part of original simulation).

### 3 Timing Experiment Analysis

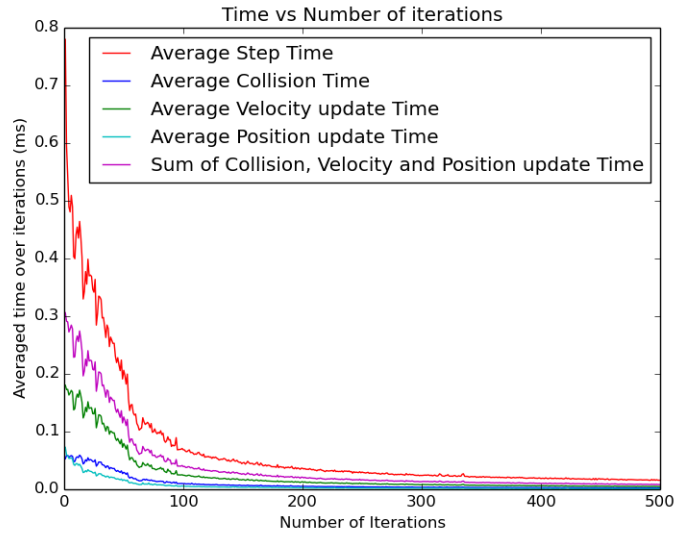
This section includes the observations and inferences from timing experiments and graphical analysis of these experiments.

#### 3.1 Observations

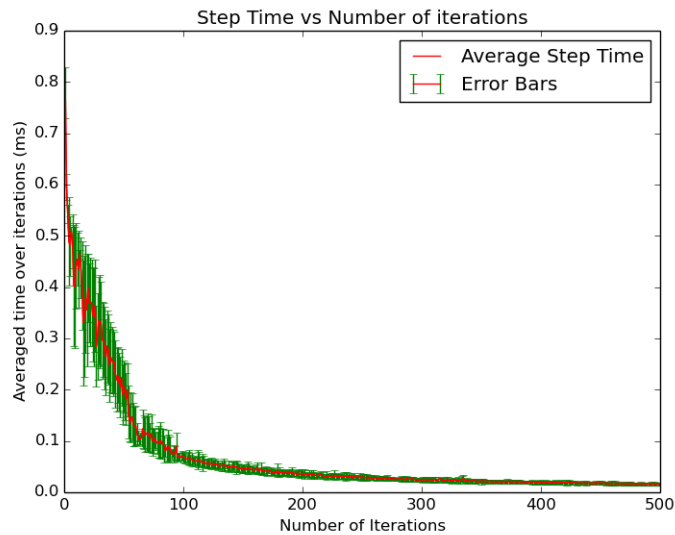
- Loop Time first increases, then decreases and then finally increases as we increase the number of iterations



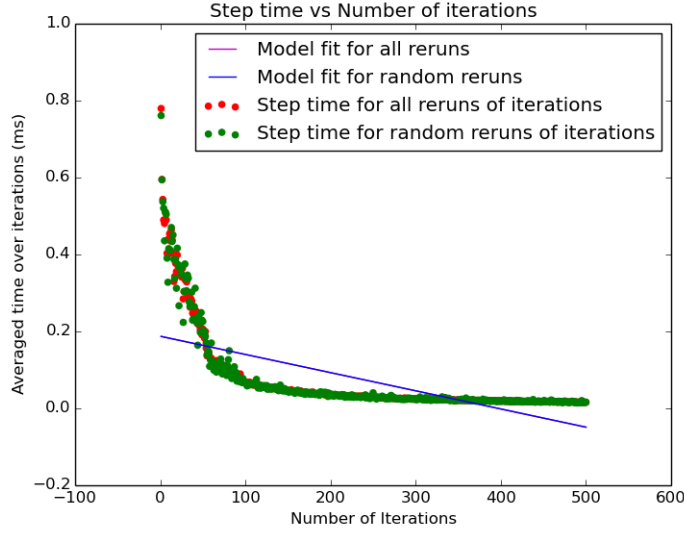
- Average Step time decreases exponentially as we increase the number of iterations
- Velocity update Time > Collision time > Position Update Time
- Step Time is greater than sum of collision, position update and velocity update time



- Variation in average step time decreases as we increase the number of iterations



- Average step time over all reruns and over random reruns is almost the same for higher iteration values



### 3.2 Reasoning and Inferences

- The initial steps take more time as compared to the later steps giving rise to decrease in average step time with increase in number of iterations
- Velocity update takes more time than collision and position update
- Collision and position update take almost same time
- Variation in average step time decreases because number of data points (iteration number) is increasing
- Average step time over all reruns and random reruns is almost same for higher iteration values because with high iteration value, variation in step time decreases, so both quantities lie near each other

## 4 Code Profiling

Flat profile:

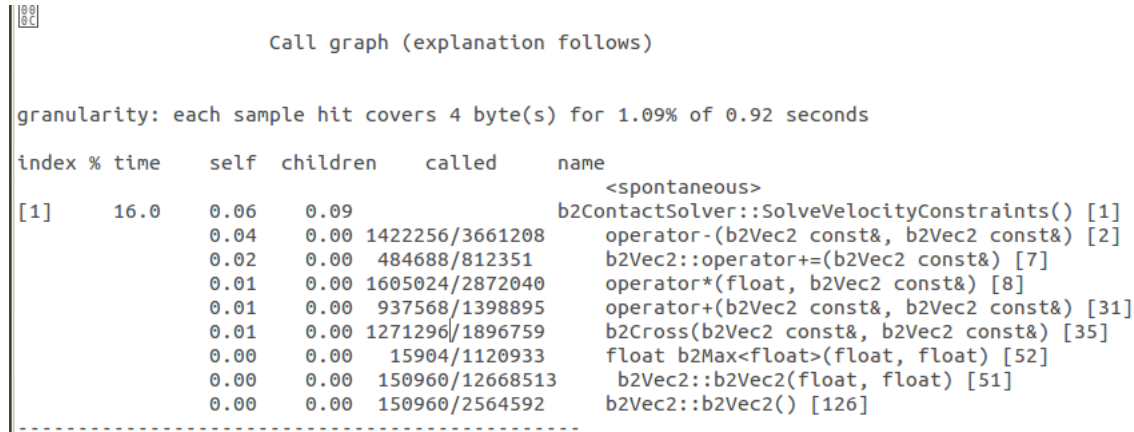
Each sample counts as 0.01 seconds.

% time	cumulative seconds	self seconds	calls	self ns/call	total ns/call	name
10.87	0.10	0.10	3661208	27.31	28.10	operator-(b2Vec2 const&, b2Vec2 const&)
6.52	0.16	0.06				b2ContactSolver::SolveVelocityConstraints()
3.26	0.19	0.03	812351	36.93	36.93	b2Vec2::operator+=(b2Vec2 const&)
3.26	0.22	0.03				b2Cross(b2Vec3 const&, b2Vec3 const&)
3.26	0.25	0.03				b2ContactSolver::b2ContactSolver(b2ContactSolverDef*)
2.72	0.28	0.03	2872040	8.70	9.49	operator*(float, b2Vec2 const&)
2.72	0.30	0.03				b2Dot(b2Vec2 const&, b2Vec2 const&)
2.72	0.33	0.03				b2DynamicTree::MoveProxy(int, b2AABB const&, b2Vec2 const&)

*Some part of Flat Profile*

Profiling data shows that the functions which took most of the time are `operator-(b2Vec2 const&, b2Vec2 const&)`, `b2ContactSolver::SolveVelocityConstraints()` and `b2Vec2::operator+=(b2Vec2 const&)`. This is possibly due to a large number of prismatic joints in our simulation. All these prismatic joints have a lower and upper translation limits. As it has to be checked again and again that whether the body is in its translation limits or not, therefore number of calls to `operator-(b2Vec2 const&, b2Vec2 const&)` is quite high.

`b2Vec2::operator+=(b2Vec2 const&)` is possibly called to change the coordinates of any body. `b2ContactSolver::Solve` is called to solve the velocity constraints. The percentage of total time taken by this function is usually high in all simulations because in all simulations velocity constraints have to be solved at each step.



The image shows a screenshot of a debugger's call graph. At the top, it says 'Call graph (explanation follows)'. Below that, a note states 'granularity: each sample hit covers 4 byte(s) for 1.09% of 0.92 seconds'. The main part of the image is a table with columns: 'index', '% time', 'self', 'children', 'called', and 'name'. The first row, index [1], shows a call to `b2ContactSolver::SolveVelocityConstraints()` which takes 16.0% of the time. This function then calls several other functions, including `operator-(b2Vec2 const&, b2Vec2 const&)`, `b2Vec2::operator+=(b2Vec2 const&)`, `operator*(float, b2Vec2 const&)`, `operator+(b2Vec2 const&, b2Vec2 const&)`, `b2Cross(b2Vec2 const&, b2Vec2 const&)`, `float b2Max<float>(float, float)`, `b2Vec2::b2Vec2(float, float)`, and `b2Vec2::b2Vec2()`. The 'called' column for the first row lists the 'self' and 'children' times for the function being called.

index	% time	self	children	called	name
[1]	16.0	0.06	0.09		<spontaneous> b2ContactSolver::SolveVelocityConstraints() [1]
		0.04	0.00	1422256/3661208	operator-(b2Vec2 const&, b2Vec2 const&) [2]
		0.02	0.00	484688/812351	b2Vec2::operator+=(b2Vec2 const&) [7]
		0.01	0.00	1605024/2872040	operator*(float, b2Vec2 const&) [8]
		0.01	0.00	937568/1398895	operator+(b2Vec2 const&, b2Vec2 const&) [31]
		0.01	0.00	1271296/1896759	b2Cross(b2Vec2 const&, b2Vec2 const&) [35]
		0.00	0.00	15904/1120933	float b2Max<float>(float, float) [52]
		0.00	0.00	150960/12668513	b2Vec2::b2Vec2(float, float) [51]
		0.00	0.00	150960/2564592	b2Vec2::b2Vec2() [126]

*Some part of Call Graph*

It can be seen from this part of call graph that most of the calls to `operator-(b2Vec2 const&, b2Vec2 const&)` are made by `b2ContactSolver::SolveVelocityConstraints()`. Major portion of the time spent on this function is due to this call only. The other major caller of `operator-(b2Vec2 const&, b2Vec2 const&)` is `b2FindMaxSeparation(b2PolygonShape const*, b2Transform const&, b2PolygonShape const*, b2Transform const&)`. `b2FindMaxSeparation(b2PolygonShape const*, b2Transform const&, b2PolygonShape const*, b2Transform const&)` must be called to find the separation between bodies to check if they are in translation limits or not (prismatic joint). We could not find any evident way to change our code so as to decrease calls to above mentioned functions as our simulation is highly dependent on prismatic joints which call these functions repeatedly.