CS 296: Lab 7

Siddhartha Dutta
120040005

Gangadhar KV
120050078

Sai Krishna Kethan
120050065

March 1, 2014

1 Timing

1.1 Analyzing results of Lab 5

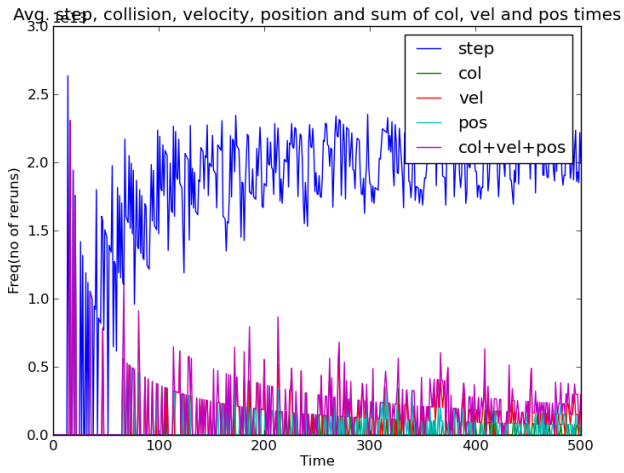

Figure 1: Avg. Step Time vs Iteration Values

In the above graph, we observe that the average step time in the beginning is quite high. As the number of iterations increases, the average step time keeps decreasing. There is a sharp fall after around 30 iterations. But after 50 iterations, the step time increases slowly till around 150 and then levels off. The above graph shows data for 500 iterations.

This can be explained if we realize that initially the wheels and rods are all still. As the wheel begins to move, forces are exerted upon the revolute

joints of various rods. Since in a given loop greater part of the time is spent on resolving collisions, the start time is quite high as various velocity and position calculations have to be made to get things moving.

As time passes on approximate solutions to equations are obtained and much lesser time and iterations of position and velocity solver is required to obtain fine tuned solutions. This explains the drop in time. There are frequent spikes in the graph due to the continuos calls made to the solver when piston reaches an end of the oscillation. The graph levels off and becomes regular with time since most bodies begin to move in a definite order and the same kind of motions are repeated after a certain period.

In every loop "Step" in B2World in called. A solver is used in collision time, velocity time and position time. Box2D also uses a constraint solver for solving constraints. The constraint solver solves all the constraints in the simulation, one at a time. Altough single constraint can be solved perfectly, when we solve one constraint, we slightly disrupt other constraints. Therefore to get a good solution, we need to iterate over all constraints a number of times. Thus, multiple iterations for position and velocity are needed.

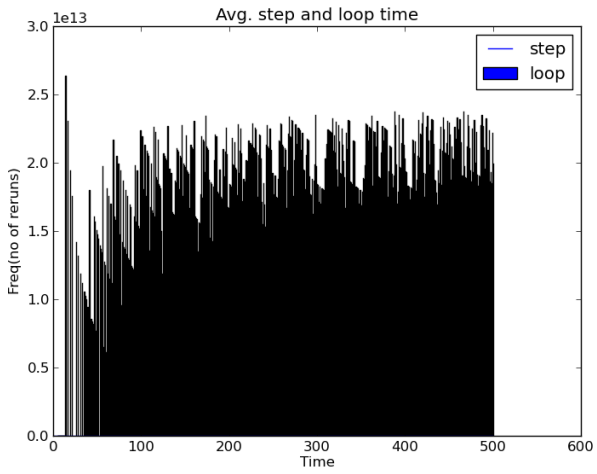

Figure 2: Total loop time

This graph reiterates the observations made earlier.The total loop time is completely linear to the number of iterations. This is in concurrence with the fact that the average step time becomes constant.

The above plot represents average step time and y-error bars. As can be clearly observed, the difference between the maximum and minimum values

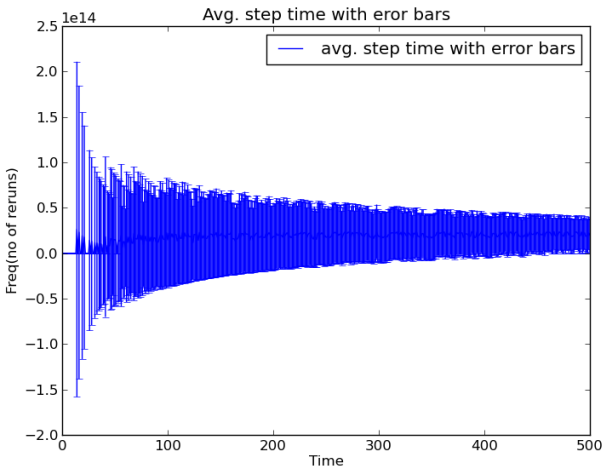

Figure 3: Avg. Step Time vs Iteration Values with error bars

is very large. The values are taken within 50 system iterations(very close times for processing). The differences can be attributed to system processes. The error range is intitially a bit high when step times are high but decreases gradually as the step time levels off.

1.2 Time vs gettimeofday()

Time writes to standard error giving various statistics regarding the running time of the program. These statistics consist of

1. the elapsed real time between invocation and termination,

Time command measures the whole program execution time, including the time it takes to load the binary and all its libraries. It also includes the time it takes to clean up everything once the program is finished.

On the other hand, gettimeofday can only work inside the program, that is after it has finished loading (for the initial measurement) and before it is cleaned up (for the final measurement). It gives the exact time required to execute the function.

The function gettimeofday has higher resolution than time: time can return only in seconds while gettimeofday() returns in microseconds.

1.3 System under heavy loading

While running CPU heavy processes, the time increases since smaller processing time is devoted to our process. I tried this by making several codes run in different terminals.

On the other hand while running RAM intensive processes, it waits and tries to kill other processes. Since the processor may be idle all this while, there's no significant change in time.

2 Profiling

2.1 Introduction

Profiling is a form of dynamic program analysis that measures, the space (memory) or time complexity of a program, the usage of particular instructions, or frequency and duration of function calls. The most common use of profiling information is to aid program optimization.

2.2 Optimization

Turning on optimization flags makes the compiler attempt to improve the performance and/or code size at the expense of compilation time and possibly the ability to debug the program. The compiler performs optimization based on the knowledge it has of the program. Optimization levels -O2 and above, in particular, enable unit-at-a-time mode, which allows the compiler to consider information gained from later functions in the file when compiling a function. Compiling multiple files at once to a single output file in unit-at-a-time mode allows the compiler to use information gained from all of the files when compiling each of them. An example of optimization is inlining the functions.

For what follows, the profiler was run over 1000 iterations.

2.3 Release Build

The release build is faster than the debug build since it uses optimization. Also it eliminates debugging information since it meant for the end-user. We used the O3 flag for compilation and used cmake -DCMAKE_BUILD_TYPE=Release for building. By observing perf report, we conclude that Position Constraint Solver and velocity constraint solver takes more time. But we can't optimise them any further as they are key for constraints.We observe that b2Vec2

Figure 4: Call graph for Release

```
Samples: 5K of event 'cycles', Event count (approx.): 3217562213
+  10.79%  cs296_base  cs296_base               [.] operator-(b2Vec2 const&, b2Vec2 const&)
+   7.32%  cs296_base  cs296_base               [.] b2ContactSolver::SolveTOIPositionConstraints(int, int)
+   7.17%  cs296_base  cs296_base               [.] operator*(float, b2Vec2 const&)
+   6.04%  cs296_base  cs296_base               [.] b2Vec2::b2Vec2(float, float)
+   4.34%  cs296_base  cs296_base               [.] b2Mul(b2Rot const&, b2Vec2 const&)
+   4.30%  cs296_base  cs296_base               [.] b2ContactSolver::SolveVelocityConstraints()
+   4.04%  cs296_base  [kernel.kallsyms]        [k] 0xffffffff8104d24a
+   3.94%  cs296_base  cs296_base               [.] b2Mul(b2Transform const&, b2Vec2 const&)
+   3.50%  cs296_base  i965_dri.so              [.] 0x0000000000075cea
+   2.72%  cs296_base  cs296_base               [.] b2Dot(b2Vec2 const&, b2Vec2 const&)
+   2.41%  cs296_base  cs296_base               [.] b2Cross(float, b2Vec2 const&)
+   2.23%  cs296_base  cs296_base               [.] b2PositionSolverManifold::Initialize(b2ContactPositionConstraint*, b2Transform const&, b2Tr
+   2.17%  cs296_base  cs296_base               [.] b2Cross(b2Vec2 const&, b2Vec2 const&)
+   2.11%  cs296_base  libm-2.17.so             [.] __sinf
+   1.96%  cs296_base  cs296_base               [.] b2Vec2::operator-() const
+   1.87%  cs296_base  cs296_base               [.] operator+(b2Vec2 const&, b2Vec2 const&)
+   1.32%  cs296_base  cs296_base               [.] b2Rot::Set(float)
+   1.12%  cs296_base  cs296_base               [.] b2FindMaxSeparation(int*, b2PolygonShape const*, b2Transform const&, b2PolygonShape const*,
+   1.07%  cs296_base  libdricore9.2.1.so.1.0.0 [.] 0x000000000024bf81
+   0.92%  cs296_base  cs296_base               [.] float b2Clamp<float>(float, float, float)
+   0.86%  cs296_base  cs296_base               [.] float b2Min<float>(float, float)
+   0.84%  cs296_base  cs296_base               [.] b2Mul(b2Mat22 const&, b2Vec2 const&)
+   0.73%  cs296_base  cs296_base               [.] b2Transform::b2Transform()
+   0.70%  cs296_base  cs296_base               [.] float b2Max<float>(float, float)
+   0.69%  cs296_base  cs296_base               [.] b2ContactSolver::InitializeVelocityConstraints()
+   0.68%  cs296_base  cs296_base               [.] b2Vec2::operator-=(b2Vec2 const&)
+   0.68%  cs296_base  cs296_base               [.] b2Min(b2Vec2 const&, b2Vec2 const&)
+   0.67%  cs296_base  cs296_base               [.] b2Vec2::b2Vec2()
+   0.66%  cs296_base  cs296_base               [.] b2Vec2::operator+=(b2Vec2 const&)
+   0.61%  cs296_base  libm-2.17.so             [.] __cosf
+   0.57%  cs296_base  libdrm_intel.so.1.0.0    [.] 0x00000000000075c3
+   0.54%  cs296_base  cs296_base               [.] b2RevoluteJoint::SolveVelocityConstraints(b2SolverData const&)
+   0.53%  cs296_base  cs296_base               [.] b2World::SolveTOI(b2TimeStep const&)
+   0.52%  cs296_base  cs296_base               [.] b2Island::SolveTOI(b2TimeStep const&, int, int)
+   0.45%  cs296_base  cs296_base               [.] b2ContactSolver::b2ContactSolver(b2ContactSolverDef*)
+   0.42%  cs296_base  cs296_base               [.] b2Max(b2Vec2 const&, b2Vec2 const&)
+   0.40%  cs296_base  cs296_base               [.] b2Distance(b2DistanceOutput*, b2SimplexCache*, b2DistanceInput const*)
+   0.37%  cs296_base  [vdso]                   [.] 0x0000000000000d98
+   0.36%  cs296_base  cs296_base               [.] b2Cross(b2Vec2 const&, float)
Press '?' for help on key bindings
```

Figure 5: Call graph for Release

takes significantly more time.This reveals that time taking processes are very less in our simulation.