Comp 768 : HW 3

Sahil Narang

November 10, 2014

1.1 Solution to Bead on Wire Problem

For keeping the bead on the wire, I used the following constraint:

$$C(x) = |x| - r = 0 \tag{1.1}$$

The idea of maintaining constraints is simple: assuming we start with a legal position and velocity, the constraints should ensure legal acceleration at each time step. Integrating the acceleration, would consequentially ensure legal velocity and legal position.

The normal N can be computed as

$$N = \frac{\partial C}{\partial x} = \frac{x}{|x|} \tag{1.2}$$

The derivative of the normal is then:

$$\dot{N} = \frac{1}{|x|}[\dot{x} - [\frac{x.\dot{x}}{x.x}]x] \tag{1.3}$$

Its easy to show that

$$\ddot{C} = \frac{\partial[N.\dot{x}]}{\partial t} \tag{1.4}$$

$$\ddot{C} = \dot{N}.\dot{x} + N.\ddot{x} \tag{1.5}$$

Let f denote the sum of the external forces and f_c denote the constraint force. Then,

$$f_c = \lambda * N \tag{1.6}$$

$$\ddot{x} = \frac{f + f_c}{m} \tag{1.7}$$

Now to find λ we can solve the following equation with spring constant α and dampening constant β to account for numerical drift.

$$\ddot{C} = -\alpha * C - \beta * \dot{C} \tag{1.8}$$

Substituting the values from the above equations,

$$\lambda = \frac{(-m * (\dot{N}.\dot{x})) + (-m * (\alpha * C)) + (-m * (\beta * (N.\dot{x}))) + (-N.f)}{N.N} \tag{1.9}$$

1.1.1 External Forces

The two external forces in the system are the graviational forces f_g and the user applied force f_{user}.

$$f_g = (0, mg, 0) \tag{1.10}$$

$$f_{user} = k * (\overrightarrow{cursor}_{pos} - \overrightarrow{bead}_{pos)} \tag{1.11}$$

$$f = f_g + f_{user} \tag{1.12}$$

1.1.2 Algorithm

At every timestep:

1. Compute normal using equation 1.2
2. Compute derivative of normal using equation 1.3
3. Compute sum of external forces by summing up the gravitational force and the force applied by the user
4. Compute λ using 1.9
5. Compute legal acceleration using 1.6 and 1.7
6. Update position and velocity using backward eular

1.2 Application

The application was developed using Unity. The simulation can be started/restarted by pressing the spacebar. The UI allows the user to enter the mass, timestep, initial position, initial velocity, spring constant, dampening constant, spring constant and the scaling factor k. The changes in the parameters are read the next time the simulation is restarted by pressing the spacebar.

1.2.1 User Interaction

The user may click the bead and drag it using the mouse. Depending on the spring constant, dampening constant and the force scaling factor k, the bead may or may not fall off the wire. At each time step, the vector computed as the difference between the cursor position and the bead position is scaled by the force scaling factor k and added to the gravitational force.

1.3 Analysis

1.3.1 Effect of spring constant

Figure 1.1 illustrates the effect of the spring constant on the trajectory of the bead. Its easy to see the numerical instability with with $\alpha = 0$. The drift is reduced significantly by increasing the spring constant. Note: For all three cases, the remaining parameters were constant.

1.3.2 Effect of dampening constant

Figure 1.2 illustrates the effect of the dampening constant on the trajectory of the bead. Its easy to see the numerical instability with with $\alpha = 0$. The drift is reduced significantly by increasing the spring constant. However, too much of an increase in the dampening constant w.r.t. the spring constant can also lead

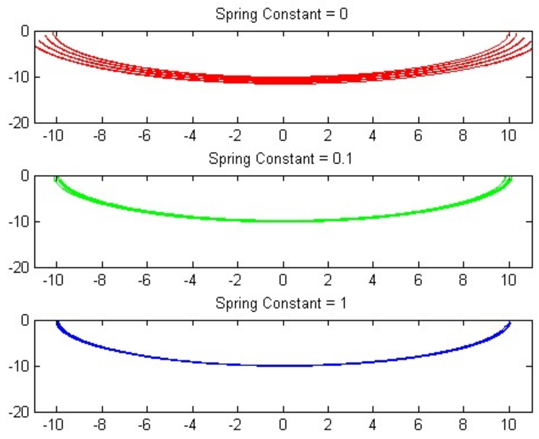

Figure 1.1:

to drift since it increases the acceleration on the bead. Hence, the ratio of the dampening and spring constant is also important. Note: For all three cases, the remaining parameters were constant.

1.3.3 Effect of timestep

Figure 1.3 illustrates the effect of timestep. As expected, with increasing step size, the Eular integration becomes more and more unstable.

1.4 Issues

During the demos, the Eular integration blew up even at a timestep of 0.05. After quite a bit of debugging, I could narrow it down to floating point precision errors. At that point, my circle was of unit radius. Hence, all values were extremely small. Especially, since unity's vector3 class only has floats, this led to a significant loss in precision. However, by simply scaling the scene to a circle of 10 units, the loss in precision was reduced and the simulation was more stable for the same set of parameters.

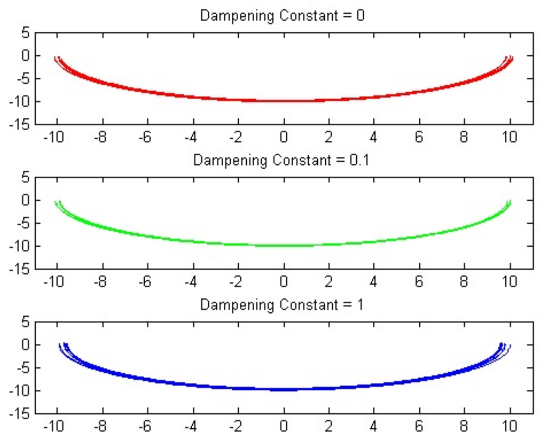

Figure 1.2:

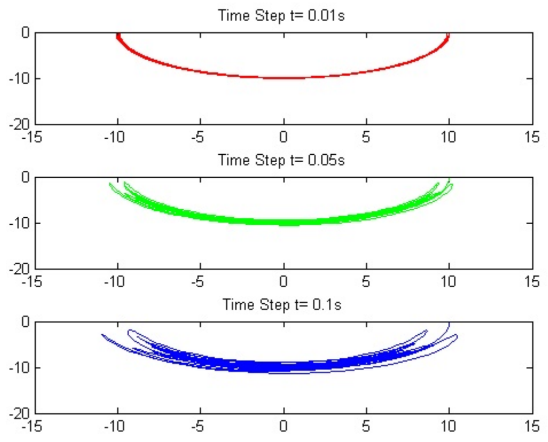

Figure 1.3: