PHY 500 Final Project

Newtonian Dynamics based Rope Simulation

Pandey, Shashwat*

M.S. in Computer Science, DigiPen Instiute of Technology, Redmond, WA, 98052

E-mail: shashwat.pandey@digipen.edu

Abstract

Physics simulation of a real-life object usually requires simplifying it to a model, composed of simple physics objects like masses and springs, which mimics the behaviour of that object. For a rope, a good approximation is a mass-spring model. A rope can be modeled as a series of masses connected to each other by springs having a high spring coefficient. This system when constrained to remain stationary at one end quite accurately simulates a hanging rope. The mechanics for such a system are simple and the equations obtained can be easily used with integrators to propogate the solution and obtain the positions of the masses.

Introduction

A simulation is an imitation of the operation of a real-world process or system. The act of simulating something first requires that a model be developed; this model represents the key characteristics, behaviors and functions of the selected physical or abstract system or process. The model represents the system itself, whereas the simulation represents the operation of the system over time.[1]

Therefore, a computer simulation of physical objects also requires that a model be developed that represents the real-world system. This model can then be simulated over time using time-integration techniques. The program which makes this possible is called a Physics Engine. The aim of this project is to create a simple physics engine which can simulate a rope and run it in real-time. The physics engine only calculates the position of all objects in the system based on the model of interaction between them and has nothing to do with the rendering of the scene. For the rendering part any graphics API can be used as the physics engine has no dependence on the graphics. Many simulations do not require any graphical rendering and aim to output data-tables which can be used to study the system. Some systems can be better represented as graphs. To study a rope and its mechanics, we need to have graphics so that we are able to observe the behaviour of the object and are able to tweak the simulation to better emulate the real world.

Theoretical Background

The model used for the physics engine in this project is the Particle model. Every particle is an infinitesimal object of finite mass. The equation of Newton's Second Law governs the motion of all such objects:

$$\vec{F} = m\vec{a}$$

As stated earlier, a rope can be approximated as a series of masses connected with springs. The smallest model for this system is a mass hanging from a spring whose other end is a fixed position in space. This basic model gives a representation of the forces acting on a mass in the rope. To better generalize this, we need to take into account that a mass in the middle of the rope would be connected to two springs and consequently be affected by spring forces from two different directions. The gravitational force will also act on these masses. This gives us a model for each mass in the rope. The Net force on each mass in the rope can be calculated as the sum of the gravitational and the spring forces acting on the mass.

$$\vec{F_N} = -m\vec{g} - k\vec{x_1} - k\vec{x_2}$$

Using this model we can simulate a hanging rope and observe its motion by generating the masses and springs on a horizontal plane and let them fall when the simulation begins. Only the gravitational force will be acting on the system at the start of the simulation. The constrained point will cause the mass attached to it to swing like a pendulum and this effect would trickle down the entire rope and will start making it swing. The spring force which can be calculated using Hooke's Law will determine the bounciness in the system. A suitably high value of the spring coefficient would give a stable swinging motion to the rope. But, we have a problem. Without any damping force in the system, it would never come to rest and keep on swinging forever which is not real-world behaviour.

For this a damping force can be used such as Air Resistance or Spring Internal Resistance. I have used a spring internal resistance as the damping force. This force can be calculated as the product of the spring coefficient with the velocity vector of the spring. The velocity vector of the spring is the difference between the velocities of both the masses attached to the spring.

$$\vec{F_D} = -k_f(\vec{v_1} - \vec{v_2})$$

Adding this force to each mass will have an effect similar to air resistance and would decay the motion of the system to a state of rest much like a real-world rope. The NeHe Productions blog[2] on Rope simulation was very helpful in understanding what goes into simulating a physics system. The author Erkin Tunca has explained an even complex system with ground collisions and air drag.

Numerical Details

The equation of motions obtained in the previous section need to be solved to run the simulation. We can use time-integration techniques to propogate the solution. I have used

a simple integrator called the Semi Implicit Euler method. For a sufficiently small time-step the solution obtained from Semi Implicit Euler gives a result with a small error and can be calculated quickly. It has a clear advantage over plain Euler as the solution stays closer to the analytical one.

Since the spring forces depend on the location of the masses, it changes every timestep. So, we need to recalculate the forces on all the masses every timestep. Once we have the forces, it is trivial to calculate the positions using Euler or any other integrator. The code snippet below shows the use of Semi-Implicit Euler to calculate the position of a mass using the forces acting on the mass every physics timestep. vel is the velocity of the mass at that instant and transform.position is the position of the mass in space. The framerate at the which the scene is rendered is further subdivided into smaller steps to calculate the physics so as to make for a smoother physics simulation as a smaller timestep keeps the errors small and would avoid any random spikes in the data.

```
public void Simulate(float dt)
{
    vel += (force / m) * dt;
    transform.position += vel * dt;
}
```

Results and Discussion

The project was implemented in Unity3D using the C-sharp language. Unity3D has a capable physics engine built into it but Unity was only used in the capacity of a rendering engine for this project. The physics engine was written separately. The Realistic rope[3] example on habrador.com was helpful in setting up the project. Even though it uses Unity3D to create a realistic model, the simulation is limited to movement in only one axis and has the rope

strictly constrained to one end. I wanted to constrain the rope at a minimum of two points so as to study the loop that forms in a hanging rope. For that I setup the project like the NeHe[2] blog post and adapted it to my use case.

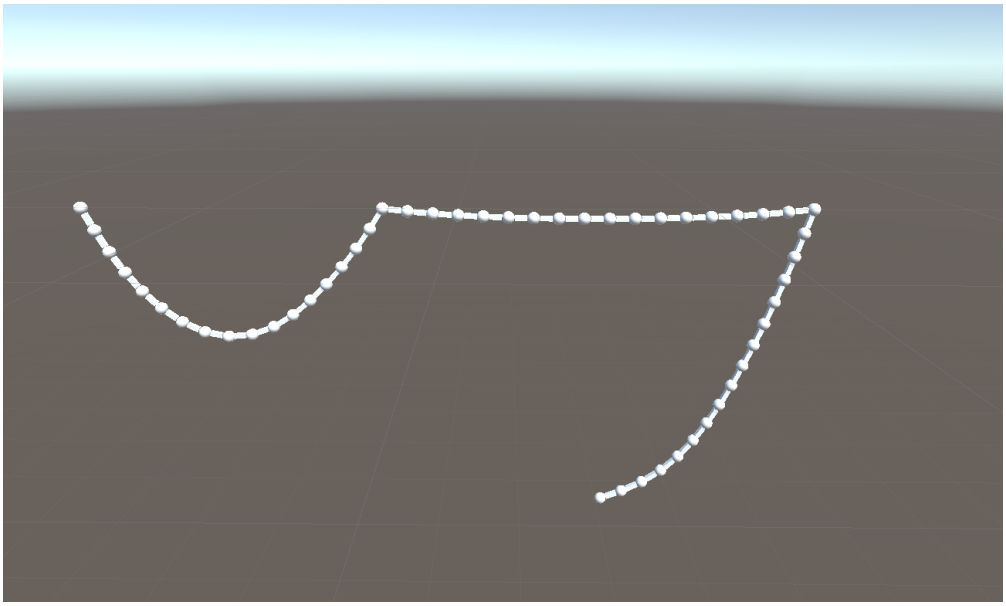


Figure 1: Rope Simulation

Figure 1 is a screenshot taken after a few seconds of running the simulation. There are three stationary points in this rope. The leftmost point can be moved left or right. After moving it towards right for a while the loop created has waves in it created as a result of the displacement. These waves slowly die down and result in the image seen above. The portion between the second and third stationary points from the left are exactly Number of Masses x Spring Length apart. This part of the rope starts to form a small loop after the simulation starts because of the gravitational force stretching the springs in the rope. There is a small up and down movement observed in the masses which slowly dies down to form a shallow loop. The curvature of this loop would depend on the spring constant of the springs.

Figure 2 displays the change in motion of the rope sections. Comparing this to the first screenshot makes it clear how the rope sections are moving and resolving themselves to a rest state due to the damping factor. The size of the damping factor is decisive in the time it takes for the system to resolve into its resting state. A smaller value causes much less decay

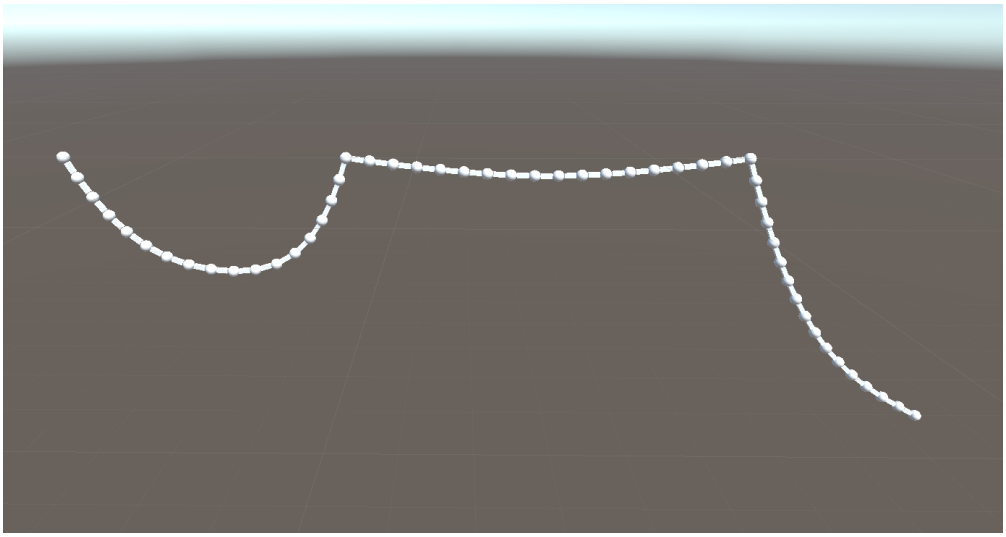


Figure 2: Rope Simulation

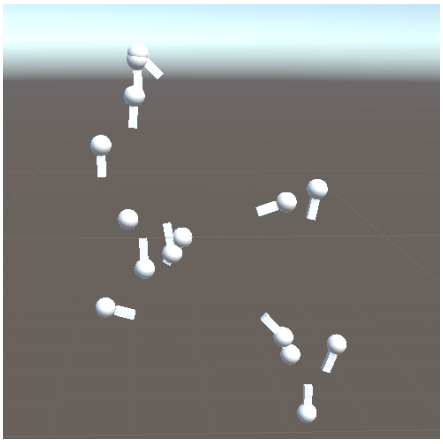
and the overall motion is not as smooth.

The values used for the above simulation are-

Constants	Values
Spring Coeff.	10000
Spring Friction Coeff.	0.05
Mass	50g
Spring Length	1m
TimeStep	0.002s

These values give a pretty realistic simulation with good framerates. The spring coefficient is very high to make the rope less bouncy. But having a high spring coefficient requires a small timestep so that the system doesnt break. The timestep of 0.002s results in 9-10 physics iterations per rendered frame when rendering at 60 frames per second. It may not be feasible to run this rope simulation in all game environments.

When trying to run the simulation at the same rate as the rendering frame rate, the system quickly breaks down at these values. The figure below shows how the system looks when it is broken.



This simulation didn't break right away but after a few swings the system could not resolve the forces in relation to the position of the masses and thus broke down. The physics engine was running at 60 frames per second and the spring coefficient was 200. To run the simulation at this frame rate and to keep it stable the spring coefficient needs to be reduced to 100. But at this value the rope bounces around a lot more and does not look as realistic. This is a clear tradeoff for running this simulation at reduced framerates.

References

(1) Wikipedia contributors, Simulation — Wikipedia, The Free Encyclopedia. 2018; `https://en.wikipedia.org/w/index.php?title=Simulation&oldid=870968164`, [Online; accessed 2-December-2018].

(2) Erkin Tunca, Rope Physics. 2001; `http://nehe.gamedev.net/tutorial/rope_physics/17006/`, [Online; accessed 2-December-2018].

(3) Eric Nordeus, How to create a swinging rope in Unity. 2018; `https://www.habrador.com/tutorials/rope/1-realistic-rope/`, [Online; accessed 2-December-2018].