An Object Oriented Library for Acoustics Simulation Based on the Physolator Simulation Framework

Dirk EISENBIEGLER
University of Furtwangen, Germany

Keywords: Physical simulation, Acoustics, Physolator, JAVA framework, Object oriented modeling.

Abstract. This paper presents a library of components for acoustics simulation. With this library, the user can assemble a sound producing device by assembling some core components, simulate the device and produce the sound of the device. The work is based on Physolator, an object oriented physical simulation framework.

Introduction

Different kinds of mechanical devices can be used to produce sounds. The sound produced by a mechanical device very much depends on the shape of the device, the materials being used, and on the way the vibration is triggered. Tonal sounds produced by such a device are usually complex vibrations [6,8]. From the point in time where the device is triggered until the tone fades away due to damping there are different kinds of overlapping frequencies with varying intensities. The library presented in this paper is something like a laboratory kit for such devices. The library is used to assemble a device from core components and to simulate the device in order to hear, how the device sounds.

This paper explains the underlying concepts and ideas by an example device consisting of a metal string and resonance body. Figure 1 visually represents such a system. The circle at the origin of the coordinate system represents the resonance body. The string is connected to the resonance body at the left hand side and to a fixed pivot point on the right hand side. Figure 1 shows a snapshot of a simulation with such a device. In the beginning of the simulation the device had been triggered by plucking the string in the middle. In reality the deflection of the string is very small. This is why figure 1 uses different scales in x and y directions. The y direction is magnified with a factor of 20.

Figure 1. String during simulation.

The next section introduces the Physolator framework and to the concept of object oriented physical modeling. The succeeding sections explain, how to implement core acoustic components such as strings and resonance bodies and how to build up physical systems by joining together these components. Then the sound recorder is introduced. It will be explained, how the sound recorder works and how it is installed inside a physical system. The final section describes, how such a physical system is simulated inside the Physolator framework and how to produce the sound.
The Physolator Framework

Physolator is an object oriented software framework based on the Java programming language for physical simulation [2,4,5]. Physolator supports an object oriented style of implementing physical systems, simulating these systems and visually representing the simulation run on the screen. Physical systems, numerical procedures and visualization components are developed independently. The Physolator binds these building blocks together during run time and executes the simulation.

Also the physical systems are implemented in a modular and object oriented style. Starting from simple physical components like vectors, point masses and springs more complex physical components are derived by joining together these components. Object oriented programming techniques such as composition, inheritance and delegation. Physical systems are complex physical components.

Resonance Bodies

A simple resonance body shall be implemented. The resonance body consists of a point mass connected to a pivot point via a spring. The position of the pivot point is predetermined and will not change during simulation time. Initially, the point mass is located at the same position as the pivot point. Its initial speed is zero. The physical behavior of this simple resonance body is characterized by three physical variables \( m \), \( k \) and \( D \). \( m \) is the mass of the point mass. \( k \) represents the spring constant. \( k \) describes the counterforce that is applied to the movement of the point mass.

The program code below declares a class that represents a resonance body. This physical component has been implemented using predefined physical components: vector, point mass and spring. Be aware, that this program code does not contain any physical formula. The pointMass object and the spring object already internally contain their physical behavior. The physical behavior has already been implemented inside these classes. The program code simply uses these components and binds them together. The constructor of class ResonanceBody builds a resonance body by creating a pivot, a point mass and a spring and by connecting the pivot with the spring and the spring with the point mass. Besides the constructor, the class provides a method for connecting the point mass with an external spring.
public class ResonanceBody {
    public Vector2D pivot;
    public PointMass2D pointMass;
    public Spring2D spring;

    public ResonanceBody(double x, double y, double m, double D, double k) {
        pivot = new Vector2D(x, y);
        pointMass = new PointMass2D(x, y, 0, 0, m);
        pointMass.k = k;
        spring = new Spring2D(D, Double.MAX_VALUE);
        spring.r1 = pivot;
        spring.r2 = pointMass.r;
        pointMass.springs = new Spring2D[] {spring};
    }

    public void connectToSpring(Spring2D spring) {
        pointMass.springs = new Spring2D[] {this.spring, spring};
    }
}

Strings

Strings are wires with a cylindrical shape. Both ends of the strings shall be fixed to specific positions, the string shall be stretched and then be triggered by plucking. This leads to a transverse oscillation of the string.

The physical parameters of a string are its length $l_0$ (unstretched), its diameter $d$, the mass density $\rho$ and the elastic modulus $E$ of the material being used. Strings used in instruments usually consist of a core part and a metal braiding that is wound around the string. The purpose of this metal braiding is to increase the mass of the string without increasing the spring constant. The variable braidingFactor shall be used to describe the increment of mass due to the braiding. Furthermore, one has to consider, that damping also applies to a string. Basically, there are two kinds of vibrations applying to a string: a transverse oscillation and a longitudinal oscillation. The variables $k_L$ and $k_T$ shall represent the longitudinal and the transverse damping, respectively.

Strings shall be represented by chains consisting of point masses that are interconnected with string segments (see figure 1). The physical behavior of a string segment very much resembles a spring as implemented in class Spring2D. However, string segments shall have damping. StringSegment2D is declared as a son class of Spring2D (see program code below). It inherits the variables and the physical behavior from Spring2D and adds damping. There is an extra variable $k$ representing the coefficient of friction. The Spring2D class only needs access to the positions of the end points. Additionally, StringSegment2D needs access to the end points velocities $v1$ and $v2$. In the Physolator framework, the method $f$ of a physical system is used to compute the values of depending values. In this case force $F$ is computed using the physics from the father class by invoking super.f and then adding the force that applies due to friction.

For sake of space, the class String2D is not printed in this article. String2D is nothing but a composition of some string segments and point masses. The concept used for composing these elements is very much the same as in the resonance body example. The parameters of the string and the number of segments define the parameters of the string segments.
public class StringSegment2D extends Spring2D implements G, F {

    @Crossref
    public Vector2D v1;
    @Crossref
    public Vector2D v2;
    public double k;

    public StringSegment2D(double D, double l0, double k, double breakForce) {
        super(D, l0, breakForce);
        this.k = k;
    }

    public void f(double t, double h) {
        super.f(t, h);
        Vector2D segmentDirection = new Vector2D(r2.x, r2.y);
        segmentDirection.sub(r1);
        Vector2D vl = new Vector2D(v2.x, v2.y);
        vl.sub(v1);
        double vla = -vl.mult(segmentDirection) / segmentDirection.abs();
        segmentDirection.normalize();
        F.add(-k * segmentDirection.x * vla, -k * segmentDirection.y * vla);
    }
}

The Sound Recorder

The following program code is a complete physical system representing a mechanical device that resembles an instrument consisting of an e^2 string of a violin and a resonance body. The length of the string is 0.35 cm. The string is connected to the resonance body on the left hand side and to a fixed pivot point on the right (see figure 1). Besides the string, the resonance body and the pivot point, the system also contains a sound recorder. The sound recorder logs a specific value. The value to be logged is specified in the expression handed over to the constructor of the class SoundRecorder. A Java lambda term is used for representing this expression (see [7]). In this case, the vertical deflection of the resonance body shall be recoded. The values expected by the sound recorder shall range from -1 to 1. This is why the vertical deflection of the resonance body has to be magnified with the factor of 2e5.

In the program code of ViolinE, the methods initGraphicsComponents, initSimulationParameters and initPlotterDescriptors are overwritten in order to provide the physical system with an appropriate graphics component, set the right parameters for the simulation and provide the system with a plotter for printing the values logged inside the recorder. Figure 2 shows the result of the simulation run.

![Figure 2. Data logged by the sound recorder.](image)
public class ViolinE extends PhysicalSystem {
    public double l = 0.35;
    public ResonanceBody resonanceBody = new ResonanceBody(0.0, 0.0, 0.05, 1e5, 10);
    public Vector2D pivot = new Vector2D(l, 0);
    public String2D string = new String2D(0.3481206582093056);
    public SoundRecorder soundRecorder =
        new SoundRecorder(() -> 2e5 * resonanceBody.pointMass.r.y);

    public ViolinE() {
        string.setEndpointA(resonanceBody.pointMass.r, resonanceBody.pointMass.v);
        resonanceBody.connectToSpring(string.stringSegments[0]);
        string.setEndpointB(pivot, new Vector2D(0, 0));
    }

    public void initGraphicsComponents(GraphicsComponents g, Structure s, Recorder r, SimulationParameters sp) {
        g.addTVG(new StringTVG(this, s, r, l));
    }

    public void initSimulationParameters(SimulationParameters s) {
        super.initSimulationParameters(s);
        s.fastMotionFactor = 0.001;
        s.iterationsPerFrame = 1000;
        s.recorderCapacity = 20000;
        s.iterativeMethod = IterativeMethodCatalogue.adamsBashforth;
    }

    public void initPlotterDescriptors(PlotterParameters r) {
        r.add("soundRecorder.value", 0.03, -1.2, 1.2);
    }
}

Summary
This library is a first step towards acoustics simulations using the Physolator framework. The devices implemented so far are pretty simple. More complex devices shall be implemented and the numerical settings shall be optimized in order to handle larger devices.

Reference