

# Accuracy and Stability in Mass-Spring Systems for Sound Synthesis

Don Morgan  
Department of Computing and Software,  
McMaster University,  
1280 Main St. West  
Hamilton, Ontario, L8S 4L7, Canada.

Sanzheng Qiao  
Department of Computing and Software,  
McMaster University,  
1280 Main St. West  
Hamilton, Ontario, L8S 4L7, Canada.

## ABSTRACT

This paper examines the stability and accuracy of mass-spring systems used in sound synthesis. We show that the standard method used in mass-spring systems has no numerical damping, but does have frequency warping and is unstable at frequencies above  $1/\pi$  times the sampling rate. We compare the standard method with two higher order numerical methods: the fourth order Runge-Kutta, and the VEFRL algorithm, a fourth order symplectic algorithm. We find that the VEFRL algorithm is much more accurate than the standard method, but that this increase in accuracy does not noticeably affect the quality of the sound produced by the mass-spring system when used to simulate a vibrating string. The increased accuracy of the VEFRL method may, however, be useful for mass-spring systems used in physics or engineering requiring high accuracy.

## Categories and Subject Descriptors

H.5.5 [INFORMATION INTERFACES AND PRESENTATION]: Sound and Music Computing—*Methodologies and techniques, Signal analysis, synthesis, and processing*; G.1.7 [NUMERICAL ANALYSIS]: Ordinary Differential Equations—*Finite difference methods, Error analysis*

## General Terms

Reliability

## Keywords

mass-spring system, symplectic Euler method

## 1. INTRODUCTION

Systems of masses, springs and dampers have been traditionally used by physicists and engineers to model vibration. In recent decades these systems have been used in sound synthesis to model the vibrations of musical instruments.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

C3S2E-08 2008 May 12-13, Montreal [QC, CANADA]  
Copyright ©2008 ACM 978-1-60558-101-9/08/05 ...\$5.00.

Since mass-spring systems model physical objects, they are classified as *physical sound synthesis* models.

Synthesis is creating something new from a combination of pre-existing parts. The name sound synthesis comes from early analog synthesizers, which created sound by combining oscillators, envelope generators and filters. In recent decades it has been increasingly common to use digital computers to synthesize sounds. Most personal computers today come equipped with a sound card. A sound card has D/A converter to convert numbers representing sound pressures (called samples) to analog signals that produce sounds. Since there are many different sound cards, a programmer will usually use an API such as RtAudio or JavaSound to send samples to the sound card. The API will handle the details of communicating with any of the standard sound cards. The programmer can set various parameters such as the sample rate and the number of bits per sample.

There are many ways to synthesize sound. Common methods are additive synthesis, subtractive synthesis and modulation synthesis. While other synthesis methods model musical sound, *physical synthesis* models the sound producing mechanisms of musical instruments [13]. This method takes mathematical models of sound production, developed by physicists and acousticians, and simulates them on computers.

## 1.1 Physical Synthesis

There are many methods used to implement physical synthesis. *Finite-difference models* use numerical techniques to discretize the differential equations of the mathematical model of the instrument. The resulting difference equations can be run on a digital computer, using parameters for the initial conditions and constants.

When a solid object is struck, deformations are caused which propagate through the object. The material and shape of the object determine the possible frequencies—or modes—that the object can vibrate at. *Modal Synthesis* approximates the sound of an instrument by summing up a finite set of modes. Each mode is represented by a sine wave with exponential damping.

Instead of simulating the wave equation itself, as is done in finite difference modelling, *digital waveguides* simulate the solution of the wave equation. Any shape that travels to the left or right at speed  $c = \sqrt{K/\epsilon}$  is a solution to this equation. The left and right traveling waves can be simulated with 2 delay lines. We can then get the solution of the wave equation at any point in time by summing the left and right delay lines. The simulation is exact if the waves are bandlimited

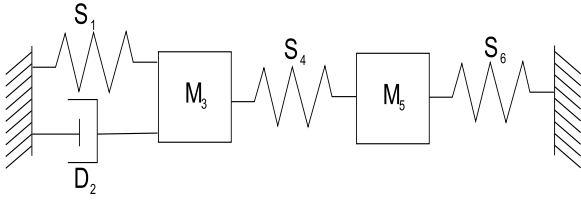


Figure 1: Simple mass-spring system

to half the sampling frequency and it is very efficient. These factors have made digital waveguides the most popular form of physical synthesis.

*Source-filter* models have been used extensively in speech synthesis and in analog music synthesizers. The source can generate periodic wave forms (i.e. sine waves, square waves etc.), random noise, or glottal noise (vibrations from vocal chords). The sound is then processed by the filter before being output. Both the source and the filter are controlled by time varying parameters.

## 1.2 The Mass-Spring Model

The *mass-spring* model builds complex musical instruments from simple components: masses, springs and dampers. Each element is discretized using finite difference methods. The behaviour of the system depends solely on the network, not on physical equations.

Figure 1 shows a simple mass-spring model, where  $M_3$  and  $M_5$  are masses,  $S_1$ ,  $S_4$  and  $S_6$  are springs and  $D_2$  is a damper.

We can derive the behaviour of a mass from Newton's 2nd law:

$$F(t) = ma(t), \quad (1)$$

where  $F(t)$  is the force acting on the mass at time  $t$ ,  $m$  is the mass, and  $a(t)$  is the acceleration of the mass at time  $t$ . Since acceleration is the derivative of the velocity we can write equation (1) as

$$F(t) = mv'(t). \quad (2)$$

We then have a system of two first order differential equations:

$$\begin{pmatrix} x(t) \\ v(t) \end{pmatrix}' = \begin{pmatrix} v(n) \\ \frac{F(t)}{m} \end{pmatrix}. \quad (3)$$

We then use the backward Euler approximation to discretize the equations. The backward Euler approximation is:

$$\mathbf{x}(n+1) \approx \mathbf{x}(n) + h\mathbf{f}(\mathbf{x}(n+1)), \quad (4)$$

where  $\mathbf{f}(\mathbf{x}(n+1))$  is the value of the derivative at the  $(n+1)$ th time step,  $h$  is the length of the time step and the vector  $\mathbf{x}(n+1)$  consists of the position and velocity at time step  $n+1$ . It is an *implicit* numerical method since  $\mathbf{x}(n+1)$  is on both the left and right sides of the equation. So equation (3) is approximated by

$$\begin{pmatrix} x(n+1) \\ v(n+1) \end{pmatrix} = \begin{pmatrix} x(n) \\ v(n) \end{pmatrix} + h \begin{pmatrix} v(n+1) \\ \frac{F(n+1)}{m} \end{pmatrix}. \quad (5)$$

The equations for the spring are derived using Hooke's Law  $F(t) = -kx(t)$ , where  $k$  is the spring stiffness. We

write them as

$$F_a(n+1) = k(x_b(n+1) - x_a(n+1)) \quad (6)$$

$$F_b(n+1) = -F_a(n+1). \quad (7)$$

Here we let  $x_b(n)$ , and  $x_a(n)$  represent the distance from the equilibrium position of *mass<sub>a</sub>* and *mass<sub>b</sub>* at either end of the spring. We use  $F_a(n)$  to denote the force acting on *mass<sub>a</sub>* at one end of the spring at time step  $n$ . The force,  $F_b(n)$ , acting on *mass<sub>b</sub>* at the other end of the spring is, according to Newton's third law, equal and opposite to  $F_a(n)$ .

Note that the equations for the mass and the spring are interdependent: in order to calculate the position of the mass at time  $n+1$  we need to know the force of the spring at time  $n+1$ , but to find the force of the spring at time  $n+1$  we need to know the position of the mass at time  $n+1$ . These are known as *delay-free loops* and make the system non-computable. We will see how this problem is dealt with in section 2.

The damper element is used to represent viscous friction. This is the object's resistance to motion and is proportional to the velocity. The formula for the damper is  $F(t) = -Zv_r(t)$ , where  $Z$  is the coefficient of viscosity,  $F(t)$  the force and  $v_r(t)$  the relative velocity of the two ends of the damper. This can be written as

$$F_a(n+1) = Z(v_b(n+1) - v_a(n+1))$$

$$F_b(n+1) = -F_a(n+1),$$

where  $F_a$  and  $F_b$  represent the forces acting on the masses at the ends of the damper, and  $v_a$  and  $v_b$  are the corresponding velocities.

## 1.3 Previous Work

The first mass-spring system used in sound synthesis was the CORDIS-ANIMA system [5], built in 1978. This system consists of two main modules: the CORDIS module which does the sound synthesis and the ANIMA module which creates the computer graphics. The philosophy behind the system was that instead of simulating the sounds themselves, as in signal processing or wave table synthesis, CORDIS-ANIMA would simulate the sound producing object [4], i.e. simulate the musical instrument rather than just simulating the sound it produces.

Another approach to mass-spring systems is TAO, created by Mark Pearson as part of his PHD thesis [12] at the University of York in 1996 under the supervision of Dr. David M. Howard. The program uses the concept of cellular automata to produce the sound synthesis. One of the main difference between TAO and CORDIS-ANIMA is the user interaction with the instruments. Whereas CORDIS uses physical actions and transducers to produce the excitations of the instruments, TAO uses a script. This makes TAO's interface resemble a musical score created by a composer and given to an orchestra, rather than a musician playing an instrument as in CORDIS.

There is also a survey paper by Välimäki et. al. [14] that contains an analysis of mass-spring systems. They describe the basic elements of mass-spring systems and how they are connected together. They suggest that adding delays between elements to eliminate delay-free loops may cause problems.

None of the previous papers give a quantitative analysis of the stability and accuracy of the numerical methods used

in implementing mass-spring systems, the issue we address in this paper.

## 1.4 Contributions and Outline

Mass-spring networks represent a method of creating complexity from simple building blocks. While each of the components, the mass, the spring and the damper, are easy to understand, the networks built from these components can become complex. Mass-spring models have traditionally been used in physics and engineering to model vibration and can be used to model a large family of vibration patterns. Thus, mass-spring systems have the advantages of conceptual simplicity, uniformity and generality.

When numerical methods are used to approximate the differential equations of a physical system used for sound synthesis, we need to be aware of their stability and accuracy. An unstable system is one in which the output grows without bound. In this case, no usable output is available to produce sound. A system that is inaccurate may, for example, produce sounds that have a different frequency than expected. Errors in frequency can cause a sound to be out of tune if the error is in the fundamental (lowest) frequency, or, give the sound a different timbre (tone colour) than it should, if the error is in the higher frequency components of the sound.

Mass-spring systems have been criticized as being computationally expensive [1], lacking an analysis of stability [1], and of unknown accuracy [6]. This paper addresses the latter two of these criticisms — the questions of stability and accuracy of mass-spring systems. We begin by showing how the “standard” method used in mass-spring systems eliminates *numerical damping* (damping caused by the numerical method that is not in the mathematical model), but that it does have *frequency warping* (alteration of the frequency caused by the numerical method) and can become unstable. We further address the question of accuracy by considering how higher order numerical methods might be used in mass-spring systems. We then examine the simulation of a vibrating string using a series of masses, springs and dampers. We compare the analytical solution to both the “standard” mass-spring model and the two models using higher order numerical methods. We present a method of correcting the frequency error in the fundamental frequency (the lowest frequency) of the “standard” method.

## 2. ANALYSIS OF THE MASS-SPRING SYSTEMS

We have seen in the introduction that using the backward Euler method to discretize the equations of the components of mass-spring systems leads to delay-free loops that make the system non-computable. A simple way to eliminate this problem is to calculate the positions at time step  $n$  based on the forces at time step  $n - 1$ . This is the method used in the TAO system. Equation (5) for the mass then becomes

$$\begin{pmatrix} x(n+1) \\ v(n+1) \end{pmatrix} = \begin{pmatrix} x(n) \\ v(n) \end{pmatrix} + h \begin{pmatrix} F(n) \\ \frac{F(n)}{m} \end{pmatrix}. \quad (8)$$

where equation (8) now uses  $F(n)$  instead of  $F(n+1)$ . We will see in section 4, that this is equivalent to the symplectic Euler method and we will refer to it by this name.

The z-transform takes signals from the time domain and transforms them to signals on the z-plane. The z-plane rep-

resents the signals in terms of amplitude growth (the growth or decay rate of the signal) and frequency, which are particularly useful in the analysis of musical signals. The z-transform is defined:

$$\hat{X}(z) = \sum_{m=-\infty}^{\infty} x(m)z^{-m}$$

where  $z = re^{j\omega}$ . The z-plane uses polar coordinates with  $|z| = r$  the distance from the origin and  $\omega$  as the angle. The amplitude growth of the signal is represented by  $r$  and the frequency by  $\omega$ . The transfer function of a system is defined as the z-transform of its input divided by the z-transform of its output. The *poles* of a system are defined as the roots of the denominator of its transfer function. A system is called stable if when the input is absolutely summable the output is absolutely summable. The system is stable on the z-plane if all its poles lie on or inside the unit circle [9] and asymptotically stable if the poles are inside the unit circle. If the poles are outside the unit circle it is unstable.

Figure 2 shows a simple mass-spring system without a damper. It shows an external force acting on mass  $M_1$ . Spring  $S_3$  has one end connect to  $M_1$  and the other end is fixed at point  $X_2$ . If we regard the equilibrium position of

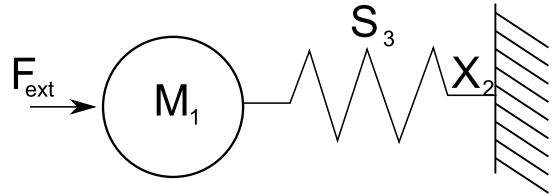


Figure 2: Mass-spring system

the spring to be  $x(t) = 0$ , where  $x(t)$  is the position of  $M_1$ , the equation of this system is

$$d^2x/dt^2 = (1/m)F_{ext} - (k/m)x(t). \quad (9)$$

Using the symplectic Euler approximation from equation (8) on this equation results in

$$\frac{x(n) - 2x(n-1) + x(n-2)}{h^2} = -(k/m)x(n-1) + m^{-1}F_{ext}(n-1).$$

Then we take the z-transform

$$\begin{aligned} (h^{-2}) \left( \hat{X}(z) - 2\hat{X}(z)z^{-1} + \hat{X}(z)z^{-2} \right) \\ = -(k/m)\hat{X}(z)z^{-1} + m^{-1}\hat{F}_{ext}(z)z^{-1}, \end{aligned}$$

where we make use of the fact that the z-transform of a unit delay is equal to  $z^{-1}$ . The z-transform of  $x(n)$  is denoted by  $\hat{X}(z)$  and that of  $F_{ext}(n)$  by  $\hat{F}_{ext}(z)$ . We then find the transfer function:

$$\hat{H}(z) = \frac{\hat{X}(z)}{\hat{F}_{ext}(z)} = \frac{(h^2/m)z^{-1}}{1 + ((h\omega)^2 - 2)z^{-1} + z^{-2}}, \quad (10)$$

where  $\omega^2 = k/m$ . The poles of the transfer function occur when

$$z^2 + ((h\omega)^2 - 2)z + 1 = 0.$$

Solving this quadratic equation gives us:

$$z = \frac{-(h\omega)^2 + 2 \pm (h\omega)\sqrt{(h\omega)^2 - 4}}{2}.$$

If  $(h\omega)^2 < 4$  then

$$z = \frac{-(h\omega)^2 + 2}{2} \pm \frac{j h\omega \sqrt{4 - (h\omega)^2}}{2}.$$

Using the real and imaginary parts, we can calculate  $r$ :

$$r = |z| = \sqrt{\left(\frac{-(h\omega)^2 + 2}{2}\right)^2 + \left(\frac{h\omega \sqrt{4 - (h\omega)^2}}{2}\right)^2} = 1.$$

So on this interval, the poles map exactly to the unit circle. This means there is no numerical damping. We can calculate the frequencies in this range. Using the facts that  $\omega = 2\pi f$ , where  $f$  is the frequency and  $h = 1/f_s$ , where  $f_s$  is the sample rate, we find that  $(h\omega)^2 < 4$  is equivalent to  $f < \frac{1}{\pi} f_s$ , i.e. all cases in which the frequency is less than  $\frac{1}{\pi}$  times the sample rate.

We now find the frequency warping. Using  $\omega_d$  for the digital angular frequency (the actual angular frequency using the numerical method) and  $\omega_a$  as the analog angular frequency (the angular frequency of the original continuous system):

$$\omega_d = \tan^{-1} \left( \frac{\text{im}(z)}{\text{re}(z)} \right) = \tan^{-1} \left( \frac{h\omega_a \sqrt{4 - (h\omega_a)^2}}{-(h\omega_a)^2 + 2} \right). \quad (11)$$

Figure 3 shows the graph of this function. As  $h\omega_a$  approaches 2 (i.e.  $f_d$  approaches  $(1/\pi)f_s$ ) the frequency becomes progressively warped. Notice that the frequency warping of symplectic Euler causes the digital frequency to be higher than the original analog frequency. The digital frequency,  $f_d$ , is in radians per sample, so to convert to radians per second we need to multiply it by the sample rate  $f_s$ .

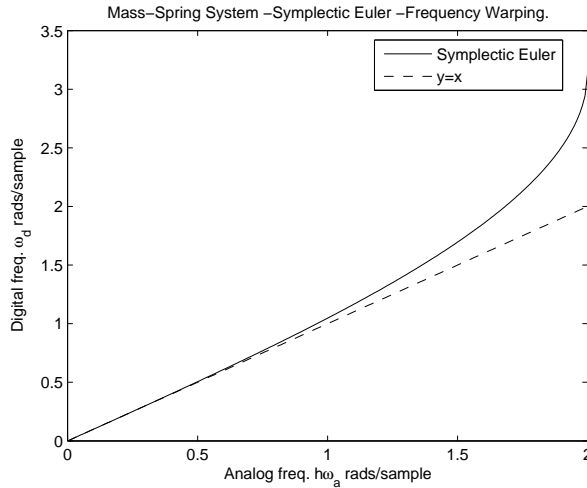


Figure 3:  $\omega_d$  versus  $h\omega_a$  for symplectic Euler

Next we examine the the interval where  $(\omega h)^2 > 4$ . We examine the pole

$$z = \frac{-(h\omega)^2 + 2 - (h\omega)\sqrt{(h\omega)^2 - 4}}{2}.$$

First, notice that this is a real number. Looking at the first 2 terms of the numerator, we see that

$$(-(h\omega)^2 + 2) < (-4 + 2) = -2.$$

The radical in the last term of the numerator is positive when  $(\omega h)^2 > 4$ , so the last term is always positive. Therefore, the numerator is always less than  $-2$  and value of the pole is always less than  $-1$ . Since this pole is outside the unit circle, the system is unstable in all cases in which  $(\omega h)^2 > 4$ . So when the analog frequency is more than  $(1/\pi)f_s$ , the system is unstable.

The final case is when  $(\omega h)^2 = 4$ . This results in  $z = -1$ .

Figure 4 shows the plot of the poles on the  $z$ -plane. The two sets of conjugate poles trace the upper and lower halves of the unit circle. At frequency  $f_a = (1/\pi)f_s$ , they meet at  $z = -1$ . Then one pole continues along the real axis toward the left and the other along the real axis to the right.

So for stability, it is required that

$$f_a = (1/\pi)f_s, \quad (12)$$

where  $f_a$  is the frequency and  $f_s$  the sample rate.

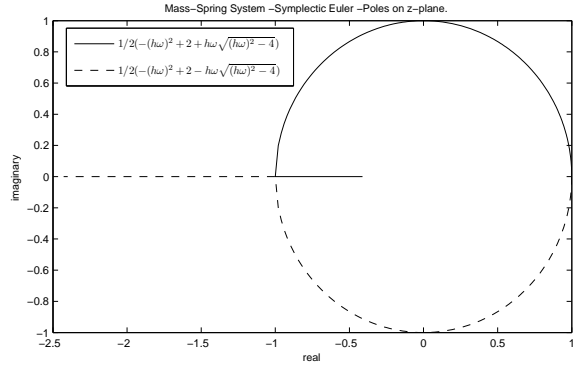


Figure 4: Poles on the  $z$ -plane for the symplectic Euler

### 3. MUTLI-STAGE NUMERICAL METHODS

We have seen in the previous section that although the symplectic Euler method used in mass-spring systems has no numerical damping, it does have frequency warping and does become unstable. The Euler method is a first order method, meaning that its error at each step (local truncation error) is order( $n^2$ ) and the overall error (global truncation error) order( $n$ ). We next look at more accurate methods.

Runge-Kutta methods are a family of methods used to solve ordinary differential equations. They may be either implicit or explicit, and work by sampling the derivative at several points in the interval between time  $n$  and time  $n + 1$ . These methods are called multi-stage methods. The most popular of these —usually just referred to as *the* Runge-Kutta method— has four stages and is explicit. The derivatives are found at four points in the interval — $k_1$  to  $k_4$ — and then these four derivatives are averaged to get an estimate of the derivate for the  $n + 1$  step. For the differential equation  $\mathbf{x}'(t) = \mathbf{f}(t, \mathbf{x})$ , the function  $\mathbf{f}(t(n), \mathbf{x}(n))$  determines the value of the derivative at time step  $n$  using the

time  $t(n)$  and the vector  $\mathbf{x}(n)$ , which consists of the position  $x(n)$  and the velocity  $v(n)$ . The fourth order Runge-Kutta method is then [3]:

$$\begin{aligned} \mathbf{k}_1 &= \mathbf{f}(t(n), \mathbf{x}(n)) \\ \mathbf{k}_2 &= \mathbf{f}(t(n) + (1/2)h, \mathbf{x}(n) + (1/2)h\mathbf{k}_1) \\ \mathbf{k}_3 &= \mathbf{f}(t(n) + (1/2)h, \mathbf{x}(n) + (1/2)h\mathbf{k}_2) \\ \mathbf{k}_4 &= \mathbf{f}(t(n) + h, \mathbf{x}(n) + h\mathbf{k}_3) \\ \mathbf{y}(n+1) &= \mathbf{x}(n) + h \frac{\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4}{6}. \end{aligned}$$

The method has a local truncation error of order  $h^5$  and a global truncation error of order  $h^4$  [3]. This should make it much more accurate than the Euler method.

There are, however, some complications in using the Runge-Kutta method on mass-spring systems. The Runge-Kutta method assumes we can calculate the derivative using the function  $\mathbf{f}(t(n), \mathbf{x}(n))$  at any point. This is not straightforward in a mass-spring system. Recall equation (3):

$$\begin{pmatrix} x(t) \\ v(t) \end{pmatrix}' = \begin{pmatrix} v(t) \\ F(t)/m_i \end{pmatrix}.$$

Here  $F(t)$  represents the total forces acting on the mass at time step  $t$ . Calculating  $\mathbf{k}_1$  is not a problem, since the forces at time step  $n$  have already been calculated. But, calculating  $\mathbf{k}_2$  is a problem, since we need the forces when  $x = x(n) + (1/2)hk_{1,1}$  and  $v = v(n) + (1/2)hk_{1,2}$ , where  $k_{1,1}$  and  $k_{1,2}$  are  $\mathbf{k}_1$ 's components corresponding to the position and velocity respectively. The force for spring  $s_j$ , connected to mass  $m_i$ , is based on the positions of the masses at either end of the spring, so we need to know not just the position of  $m_i$  but the position of the mass  $m_k$  connected to  $m_i$  by spring  $s_j$ . Since a mass can be connected to an arbitrary number of springs, we need to know the positions of all these masses in order to calculate the force acting on mass  $m_i$ . The situation is similar for the damper, except that it needs the velocities of the 2 masses instead of the positions. Since each of these masses may be connected to other masses, we need to know the positions and velocities of all the masses at  $x = x(n) + (1/2)hk_{1,1}$  and  $v = v(n) + (1/2)hk_{1,2}$  in order to compute the forces acting on each mass. After computing all the forces, we can finally compute the derivative  $\mathbf{k}_2$ .

This means, for each stage,  $s = 1 \dots 4$ , we must first calculate the positions of all the masses and then use these positions to calculate the forces of all the springs and dampers. After this we can calculate  $k_s$  for each mass. After stage 4 is completed, we can calculate the new positions and velocities of all the masses and finally, the new forces of all the springs and dampers. This means that instead of one loop through each of the masses, springs and dampers as in the Euler method, we need 5 loops through each of the components. This means that the Runge-Kutta method will take about 5 times longer than the Euler method.

## 4. SYMPLECTIC NUMERICAL METHODS

Some numerical methods, such as the backward Euler method, can cause numerical damping — damping caused by the numerical methods that is not contained in the equation itself. This has proved to be a problem, especially in fields such as molecular and planetary simulation. In such systems conservation of energy over long periods of time

is very important. Although all real systems that produce sound have some damping, many are very lightly damped, so a numerical method that preserves energy is important in physical sound synthesis. A numerical method is called symplectic if it preserves area on the position/momentum phase plane. Among other properties, symplectic systems conserve energy.

The Euler-Cromer method, also called the symplectic Euler, is a first order symplectic method defined as [8]:

$$\begin{pmatrix} x(n+1) \\ v(n+1) \end{pmatrix} = \begin{pmatrix} x(n) \\ v(n) \end{pmatrix} + h \begin{pmatrix} v(n) \\ a(n) \end{pmatrix}. \quad (13)$$

If we replace  $a$  with  $F/m$ , we see that this equation is the same as (8).

Next, we plot the energy of the mass-spring system. The total energy of a mass-spring system is the sum of the kinetic energy of the masses,  $K = \frac{1}{2}mv^2$ , and the potential energy stored in the springs,  $U = \frac{1}{2}kx^2$ . Momentum is mass times velocity so the total energy

$$E = \frac{1}{2}mv^2 + \frac{1}{2}kx^2 = \frac{1}{2}pv + \frac{1}{2}kx^2 = \frac{1}{2m}p^2 + \frac{1}{2}kx^2.$$

Setting  $k = 1$  and  $m = 1$ , we get

$$E = \frac{1}{2}p^2 + \frac{1}{2}x^2; \quad (\sqrt{2E})^2 = p^2 + x^2.$$

This is a circle of radius  $\sqrt{2E}$ . If the starting conditions are  $x = 1$ ,  $v = 0$  the potential energy is  $\frac{1}{2}kx^2 = \frac{1}{2}$  and the kinetic energy is 0. The radius is then  $(\sqrt{2 \times \frac{1}{2}})^2 = 1$ . The phase plot, therefore, should be a unit circle.

Figure 5 shows a plot of the position/momentum phase plane of the symplectic Euler. The plot covers a thousand oscillations. This would be 1 second of sound at 1000 Hz or just .1 seconds at 10,000 Hz, which is still within the range of hearing. So it is important that a numerical method be able to maintain relatively constant energy over this many cycles. From figure 5 we can notice that the shape is somewhat elliptical due to inaccuracies of the first order method. But, the symplectic Euler is able to maintain the energy over a thousands cycles.

In contrast, figure 6 shows the phase plot of the forward Euler. The energy of the forward Euler increases without bound.

Next we try a higher order algorithm —the fourth order Runge-Kutta— which is not symplectic. The results are shown in figure 7. The more accurate Runge-Kutta method results in a circle, not an ellipse. The energy, however, does not stay constant, which can be seen by the fact the the graph slowly spirals inwards toward the center of the circle.

Recently, several higher order symplectic methods have been developed. One of these methods is called the “Velocity Extended Forest-Ruth Like” (VEFRL) algorithm, which is

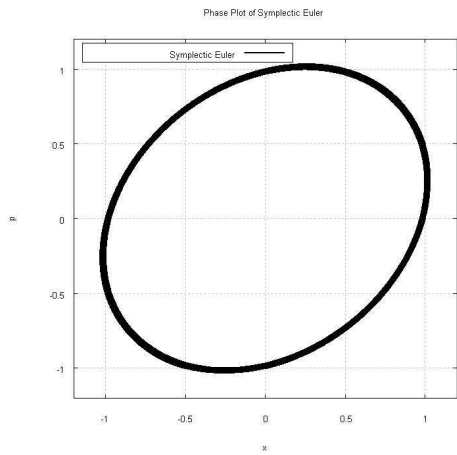


Figure 5: Phase plane of Symplectic Euler

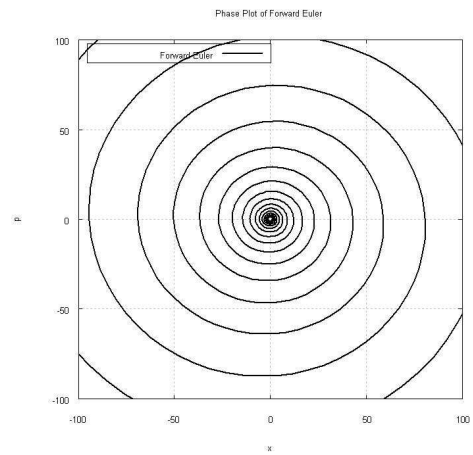


Figure 6: Phase plane of Forward Euler

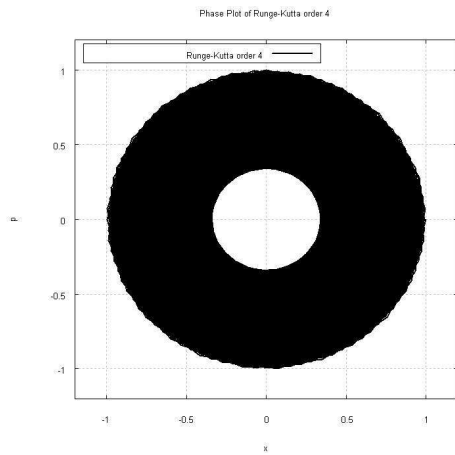


Figure 7: Phase plane of fourth order Runge-Kutta

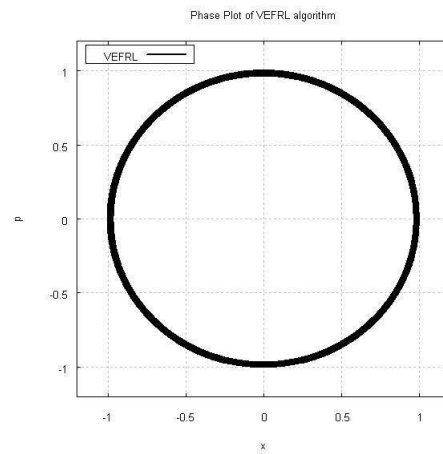


Figure 8: Phase plane of the VEFRL algorithm

a fourth order method defined as [11]

$$\begin{aligned}
v_1 &= v(t) + \frac{1}{m}f[r(t)]\xi h \\
r_1 &= r(t) + v_1(1 - 2\lambda)h/2 \\
v_2 &= v_1 + \frac{1}{m}f[r_1]\chi h \\
r_2 &= r_1 + v_2\lambda h \\
v_3 &= v_2 + \frac{1}{m}f[r_2](1 - 2(\chi + \xi))h \\
r_3 &= r_2 + v_3\lambda h \\
v_4 &= v_3 + \frac{1}{m}f[r_3]\chi h \\
r(t+h) &= r_3 + v_4(1 - 2\lambda)h/2 \\
v(t+h) &= v_4 + \frac{1}{m}f[r(t+h)]\xi h, \tag{14}
\end{aligned}$$

where  $r$  is the position,  $v$  the velocity,  $m$  the mass,  $f$  the force. The constants are:

$$\begin{aligned}
\xi &= +0.1644986515575760E + 00 \\
\lambda &= -0.2094333910398989E - 01 \\
\chi &= +0.1235692651138917E + 01.
\end{aligned}$$

The phase plot for this algorithm is shown in figure 8. This plot shows that the algorithm is both accurate —since it plots a circle— and symplectic —since it maintains a constant energy over a thousand cycles.

This algorithm can be implemented similarly to the Runge-Kutta, with each stage requiring an update of the velocities and positions of all the masses and then the force of each spring and damper is calculated based on the new velocities and positions. The last step has an additional problem, since for mass-spring systems the force calculation is a function of both the position and the velocity (since viscous damping is a function of the velocity). This makes equation in (14) implicit. A simple solution is to calculate the positions of all the masses for  $r(t+h)$  and calculate an estimated velocity for time  $t+h$  as  $v_{est} = v(t) + (f(t)/m)h$ ; then, update all the forces of the springs and dampers using the new positions and estimated velocities; now we can calculate the velocity at time  $t+h$  using  $f(r(t+h), v_{est})$ ; finally we use  $r(t+h)$  and  $v(t+h)$  to calculate the force of the springs and dampers at time  $t+h$ .

ALGORITHM 1 (VEFRL FOR MASS-SPRING SYSTEM).

1. For stage  $s = 1$  to 4
2. For each mass
  3. calculate position and velocity for stage  $s$  according to equation (14).
4. end for
5. For each spring or damper
  6. calculate the force for stage  $s$ .
7. end for
8. end for
9. For each mass
  10. calculate the position of the mass for time step  $n + 1$ .
  11. calculate estimated velocity
$$v_{est} = v(n) + (f(n)/m)h.$$
12. end for

13. For each spring or damper
  14. calculate force using positions at time step  $n + 1$  and estimated velocities.
15. end for
16. For each mass
  17. calculate velocity of the mass for time step  $n + 1$  using forces from step 14.
18. end for
19. For each spring or damper
  20. calculate force at time step  $n + 1$  using positions and velocities at time step  $n + 1$ .
21. end for

## 5. RESULTS AND DISCUSSION

In this section, we simulate a string by connecting springs between 20 masses as shown in Figure 9. The mass of each mass and the stiffness of each string are all identical.

We first find the analytical solution, using the eigenvalue method presented by Meirovitch [10]. The mass matrix,  $M$ , contains each of the masses along its diagonal

$$M = \begin{pmatrix} m_1 & 0 & 0 & \dots & 0 \\ 0 & m_2 & 0 & \dots & 0 \\ \dots & & & & \\ 0 & \dots & 0 & 0 & m_{20} \end{pmatrix}.$$

The stiffness influence coefficients,  $k_{ij}$ , are defined as the forces required for a unit displacement of mass  $i$ , with all other masses  $j \neq i$  having a displacement of zero. On a simulated string such as Figure 9, the force required to displace mass  $m_i$  one unit to the right is  $k_i + k_{i+1}$ , since the spring on the left must be expanded by one unit and the spring on the right compressed by one unit. Masses  $m_{i-1}$  and  $m_{i+1}$  require forces in the opposite direction to keep them in place. The stiffness matrix,  $K$ , for a linear string is then a tridiagonal matrix with the  $i$ th plus the  $(i + 1)$ th spring's stiffness coefficient on the diagonal, the negative of the  $i$ th stiffness coefficient to its left, and the negative of the  $(i + 1)$ th stiffness coefficient to its right:

$$K = \begin{pmatrix} k_1 + k_2 & -k_2 & 0 & \dots & 0 \\ -k_2 & k_2 + k_3 & -k_3 & \dots & 0 \\ 0 & -k_3 & k_3 + k_4 & -k_4 & \dots & 0 \\ \dots & & & & & \\ 0 & \dots & & 0 & -k_{20} & k_{20} + k_{21} \end{pmatrix}$$

We can write the equation of the system in vector form as an eigenvalue problem [10],

$$K\mathbf{u} = \omega^2 M\mathbf{u}, \tag{15}$$

where  $\mathbf{u}$  is a vector of the displacement of each of the masses from its equilibrium position.

By using the Cholesky factorization  $M = LL^T$ , where  $L$  is a lower triangular matrix, we can turn this equation into a standard eigenvalue problem [10]:

$$A\mathbf{v} = \lambda\mathbf{v} \text{ where } A = L^{-1}K(L^{-1})^T \text{ and } \lambda = \omega^2.$$

We can now calculate the eigenvalues,  $\lambda_r$ , and the eigenvectors,  $\mathbf{v}_r$ , numerically using software such as MATLAB. We then use  $\mathbf{u}_r = (L^T)^{-1}\mathbf{v}_r$  to get the eigenvectors of the original system. The eigenvectors,  $\mathbf{u}_r$  make up the *normal modes* of any vibration of the system. This means that any vibration can be described as a linear combination of the

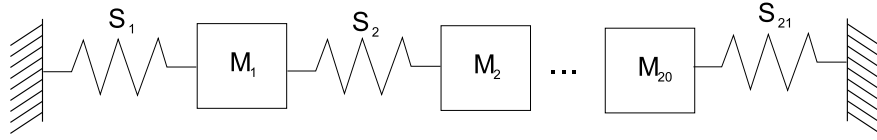


Figure 9: String created from 20 masses

eigenvectors. We can calculate the positions of the masses at any time with

$$\mathbf{x}(t) = \sum_{r=1}^N \mathbf{u}_r^T M \mathbf{x}_0 \cos(\omega_r t) \mathbf{u}_r,$$

where  $\mathbf{x}_0$  is the vector of the initial positions of the masses and  $\omega_r = \sqrt{\lambda_r}$ .

Figure 10 compares the analytical solution with that of a mass-spring system using the symplectic Euler method on a simulated string using 20 masses. The sample rate used was 44,100 samples per second. The initial conditions were that the 6th mass was displaced by one unit and all the other masses had a displacement of zero. The first 200 samples are shown using the displacement of mass 1. Notice that at the start the symplectic Euler is fairly close to the analytical solution, but that it quickly diverges from it. Notice also that the frequency of the Symplectic Euler is a little higher than that of the analytical solution. This is consistent with the frequency warping noted in section 2.

Figure 11 shows the same system comparing the fourth order Runge-Kutta and the VEFRL algorithm with the analytical solution. For the first 200 samples both methods match the analytical solution almost perfectly. The situation, however, changes over time. Figure 12 shows the same simulation after 4,000 samples. The VEFRL algorithm is still quite close to the analytical solution, but the Runge-Kutta is showing the effects of numerical damping.

What is the aural effect of these different methods? The numerical damping of the Runge-Kutta method is clearly audible. But, the symplectic Euler and the VEFRL are virtually indistinguishable. To understand why this is the case, we consider a continuous string. The frequencies that make up the continuous string are all integer multiples of the fundamental frequency [2]. For example, if the fundamental frequency is 440 Hz, the other frequencies (or modes) that make up the sound will be 880 Hz, 1320 Hz etc. In a string simulated with lumped masses, if the all the masses and spring constants are the same, the frequencies are [7]

$$\omega_n = 2\omega_0 \sin\left(\frac{n\pi}{2(N+1)}\right) \quad (16)$$

where  $\omega_n$  is the  $n$ th frequency,  $\omega_0 = \sqrt{k/m}$ , and  $N$  is the number of masses. We can solve this equation for  $\omega_0$  with  $n = 1$

$$\omega_0 = \frac{\omega}{2 \sin\left(\frac{\pi}{2(N+1)}\right)}. \quad (17)$$

This allows us to calculate  $\omega_0$  for a given frequency and number of masses. So, for 20 masses and a fundamental frequency of 440 Hz,  $\omega$  is  $440 \times 2\pi$  and  $\omega_0$  is 18,497.2439. We can now calculate the frequencies for our simulated string. The second frequency is 877.54 Hz as opposed to 880 Hz for the continuous string. The third frequency is 1310.17 Hz as opposed to 1320 Hz. The twentieth frequency is 5,822.09 Hz

as opposed to 8,800 Hz. We can see that the low frequencies are close to those of the continuous string, while the higher frequencies are quite inaccurate. All the frequencies of the mass-spring approximation are *lower* than those of the continuous string. When we use the symplectic Euler method these frequencies become warped according to equation(11) with  $h = 1/44100$ . The fundamental is warped from 440 Hz to 440.07 Hz, the second frequency from 877.54 Hz to 878.11 Hz and the third from 1310.17 Hz to 1312.08 Hz. The change of frequency in the fundamental is almost imperceptible. The other frequencies are shifted by a small amount upwards — toward the frequencies that a continuous string would have. So if anything, the symplectic Euler should sound a little more like a continuous string than the more accurate methods.

Computing the Fourier transform of the output of the simulated string produces results that are consistent with equation (11). Figure 13 shows the Fourier transform of the analytical solution and the symplectic Euler simulation. Each spike in the graph represents the frequency of one of the modes. We used 8192 “buckets”, so, using a sample rate of 44,100 Hz, each bucket is about 5.4 Hz wide. This means the results should be accurate within 2.7 Hz. As expected, the low frequency modes of the symplectic Euler are very close to those of the analytical solution. As the frequency becomes higher the frequency of the symplectic Euler is noticeably higher than that of the analytical solution. For the 20th mode, the spike for the analytical solution occurs at 5873 Hz, which is close to the expected 5971 Hz. The spike for the 20th mode in symplectic Euler simulation is at 6056 Hz, which is close to the calculated 6058 Hz.

If a lower samples rate is used, the frequency warping of the symplectic Euler becomes more noticeable. For example, at 8,000 samples per second, the symplectic Euler warps the fundamental frequency, 440 Hz, to 442.19 Hz, which is noticeably higher. However, since the VEFRL method takes around 6 times longer to run, it is much more efficient to use the symplectic Euler and compensate for the frequency warping. We can do this by using the inverse of equation (11), which comes out to

$$w_a = h^{-1} \sqrt{\frac{2(\alpha + 1 + \sqrt{\alpha + 1})}{\alpha + 1}}, \quad (18)$$

$$(19)$$

where  $\alpha = \tan^2(h\omega_d)$ ,  $\omega_a$  is the analog frequency and  $\omega_d$  the actual frequency resulting from the symplectic Euler method. So if we set  $w_d$  to the desired frequency (i.e.  $440 \times 2\pi$ ) we get back the frequency we should use in equation(17) to calculate  $\omega_0$ . This will correct the fundamental frequency so that it is exactly 440 Hz.

The more masses used in the simulated string, the more accurate the simulation becomes. We note in section 2 that the symplectic Euler becomes unstable at  $1/\pi$  times the sam-

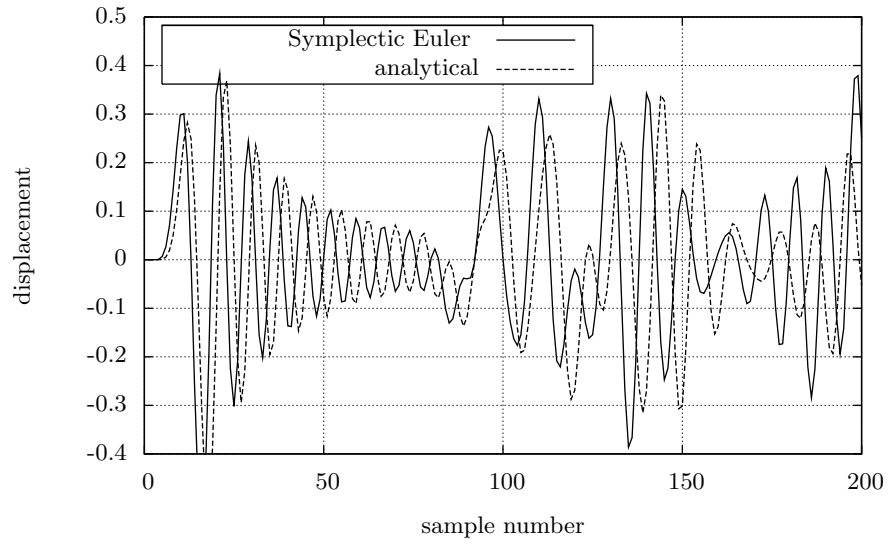


Figure 10: Simulated string comparing symplectic Euler with analytical solution

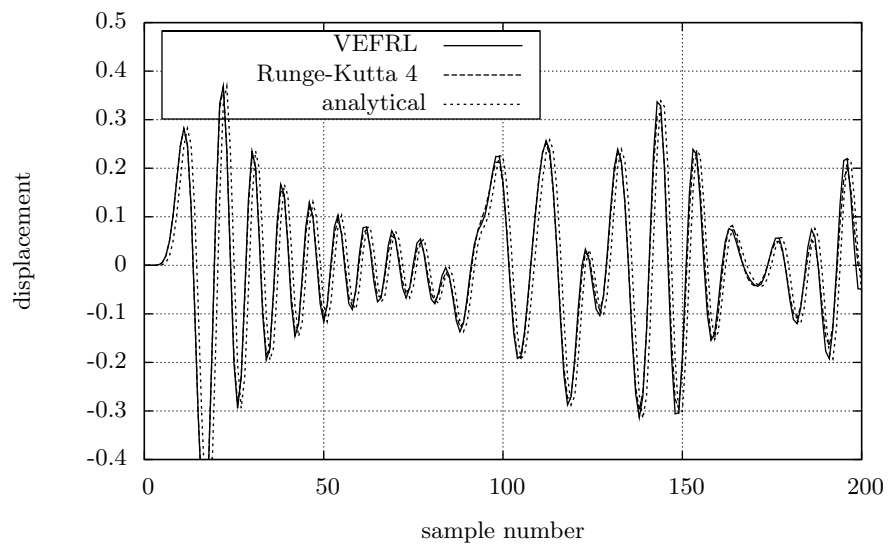


Figure 11: Simulated string comparing Runge-Kutta and the VEFRL with the analytical solution

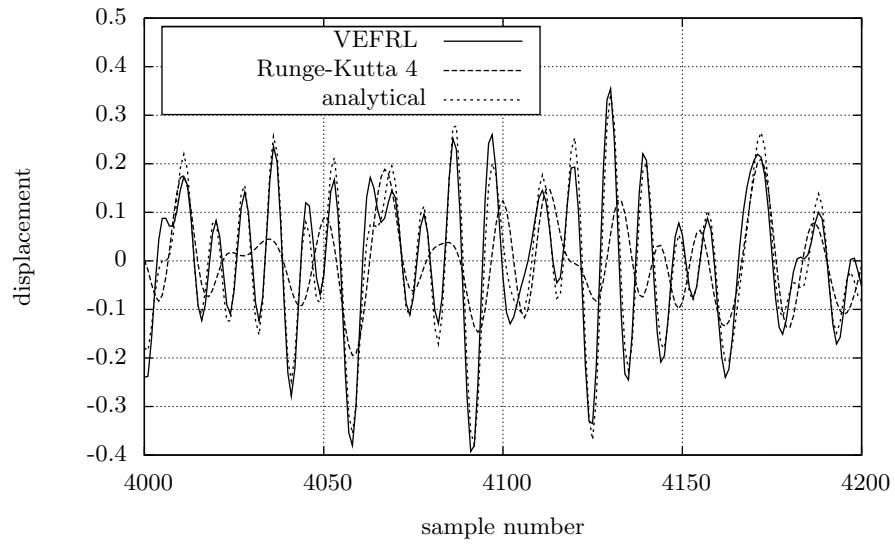


Figure 12: Simulated string comparing Runge-Kutta and VEFRL with the analytical solution after 4,000 samples

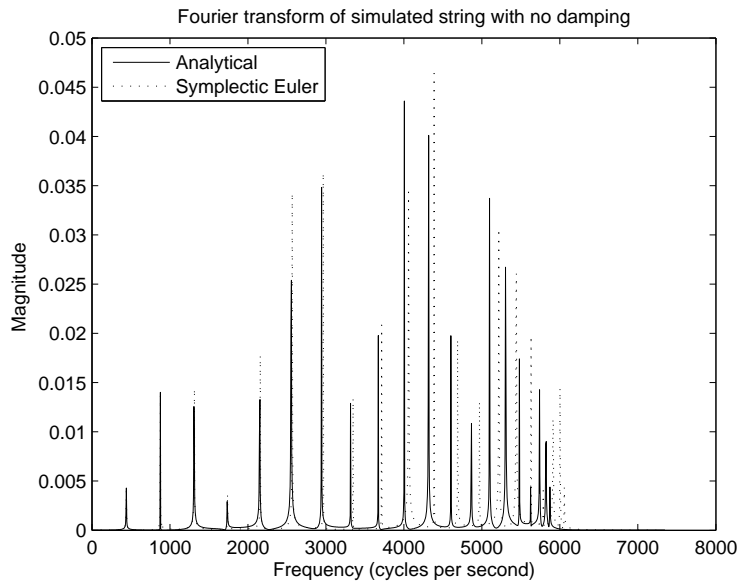


Figure 13: Fourier transform of undamped simulation string

pling frequency. This puts a limitation on the number of masses we can use to simulate a string vibrating at frequency  $\omega$  and sample rate  $f_s$ . Consider for example, the simulation of a string vibrating at 440 Hz and using a sample rate of 44,100 Hz. The maximum frequency that the symplectic Euler can have at this sample rate is  $44,100/\pi = 14,037.5$  Hz. If we list the highest frequencies by using equation (16) with  $n = N$ , where  $N$  is the number of masses, we find that at 44 masses the highest frequency is 13,957 Hz and with 45 masses the highest frequency is 14,045 Hz. This tells us that the maximum number of masses we can use for a string vibrating at 440 Hz with a sample rate of 44,100 is 44. If we want to use more masses we have to go to a higher sampling rate. The Runge-Kutta and VEFRL can go up to near the Nyquist limit,  $f_s/2$ , before becoming unstable. Even if a numerical method can go above the Nyquist limit without becoming unstable, aliasing will occur, resulting in an inaccurate sound. It is computationally more efficient to use the symplectic Euler at a higher sample rate, than it is to use the VEFRL algorithm. At some point, though, the sampling rate of the sound card will be exceeded. At that point the results of the simulation must be put through a low pass filter to remove the frequencies above Nyquist limit, and then downsampled to the maximum sample rate of the sound card.

We also simulated damped strings. The damping gives the simulation a much more realistic sound. The results are similar to those of the undamped string in that the VEFRL algorithm tracks the analytical results very closely, while the symplectic Euler quickly diverges from it. The Runge-Kutta method shows more damping than it should, so after a few thousand samples it is not very accurate. At a sampling rate of 44,100 samples per second, the sounds of the symplectic Euler and the VEFRL are indistinguishable.

## 6. CONCLUSIONS

We have addressed the issues of stability and accuracy of mass-spring systems in sound synthesis. We find that, using the symplectic Euler method (the standard method used in implementing mass-spring systems), mass-spring systems are stable up to frequencies of  $1/\pi$  times the sample rate. When simulating a vibrating string, the highest mode of string, therefore, must be less than  $1/\pi$  times the sample rate for the simulation to be stable.

We find that the symplectic Euler method has no numerical damping, but that it is not accurate in frequency, warping frequencies upward. We compare the symplectic Euler method with two higher-order methods: the fourth order Runge Kutta, and the VEFRL algorithm, a fourth order symplectic algorithm. We find that the Runge-Kutta method has numerical damping which makes it become increasingly inaccurate in long lasting sounds. The VEFRL algorithm, on the other hand, conserves energy, and is much more accurate than either the Symplectic Euler or the Runge-Kutta. We find, however, that perceptually —the way the simulation sounds— there is little or no difference between the symplectic Euler and the VEFRL algorithm for the simulation of a vibrating string. This is because, when a small number of masses are used, the difference between the mathematical model used by the mass-spring system and an actual vibrating string is much greater than the difference in accuracy of the two numerical methods. If a large number of masses are used the model becomes much more accurate,

but a high sample rate is required to avoid aliasing, and so both methods are quite accurate in frequency range of human hearing.

Since the VEFRL method takes about 6 times as long to run as the Symplectic Euler, we conclude that it is not likely to be useful in mass-spring systems used for sound synthesis. However, mass-spring systems are used in other area, and for applications in physics and engineering requiring high accuracy the VEFRL method should be useful.

## 7. REFERENCES

- [1] Federico Avanzini. *Computational Issues in Physically-based Sound Models*. PhD thesis, Università degli Studi di Padova Dipartimento di Elettronica ed Informatica, 2001.
- [2] John Backus. *The Acoustical Foundations of Music*. W.W. Norton and Company Inc., 1969.
- [3] William E. Boyce and Richard C. DiPrima. *Elementary Differential Equations and Boundary Value Problems*. John Wiley and Sons, Inc., 2001.
- [4] Claude Cadoz and Jean Loup Florens. The Physical Model: Modeling and Simulating the Instrumental Universe. In Giovanni DePoli, Aldo Piccialli, and Curtis Roads, editors, *Representations of Musical Signals*. MIT Press., Cambridge, MA, 1991.
- [5] Claude Cadoz, Annie Luciani, and Jean Loup Florens. CORDIS-ANIMA: A Modeling and Simulation System for Sound and Image Synthesis: The General Formalism. *Computer Music Journal*, 17(1):19–29, 1993.
- [6] Giovanni DePoli and Davide Rocchesso. Physically based sound modelling. *Organised Sound*, 3(1):61–76, 1998.
- [7] A.P. French. *Vibrations and Waves*. W.W. Norton and Company Inc., 1971.
- [8] Harvey Gould, Jan Tobochnik, and Wolfgang Christian. *An Introduction to Computer Simulation Methods*. Addison Wesley, 2007.
- [9] Edwards A. Lee and Pravin Varaiya. *Structure and Interpretation of Signals and Systems*. Addison Wesley, 2003.
- [10] Leonard Meirovitch. *Fundamentals of Vibrations*. McGraw Hill, 2001.
- [11] I.P. Omelyan, I.M. Mryglod, and R Folk. Optimized Forest-Ruth and Suzuki-like algorithms for integration of motion in many-body systems. *Computer Physics Communications*, 146:188–202, 2002.
- [12] Mark Pearson. *Synthesis of organic sounds for electroacoustic music*. PhD thesis, Department of Electronics, University of York, 2000.
- [13] Curtis Roads. *The Computer Music Tutorial*. The MIT Press, 1996.
- [14] Vesa Välimäki, Jyri Pakarinen, Cumhur Erkut, and Matti Karjalainen. Discrete-time modelling of musical instruments. *Reports on Progress in Physics*, 69(1):1–78, 2006.