An Introduction to Physically Based Modeling:
An Introduction to Continuum Dynamics for Computer Graphics

Michael Kass
Pixar

Please note: This document is ©1997 by Michael Kass. This chapter may be freely duplicated and distributed so long as no consideration is received in return, and this copyright notice remains intact.

An Introduction to Continuum Dynamics for Computer Graphics

Michael Kass
Pixar

1 Introduction

Mass-spring systems have been widely used in computer graphics because they provide a simple means of generating physically realistic motion for a wide range of situations of interest. Even though the actual mass of a real physical body is distributed through a volume, it is often possible to simulate the motion of the body by lumping the mass into a collection of points. While the exact coupling between the motion of different points on a body may be extremely complex, it can frequently be approximated by a set of springs. As a result, mass-spring systems provide a very versatile simulation technique.

In many cases, however, there are much better simulation techniques than directly approximating a physical system with a set of mass points and springs. The motion of rigid bodies, for example, can be approximated by a small set of masses connected by very stiff springs. Unfortunately, the very stiff springs wreak havoc on the numerical solution methods, so it is much better to use a technique based on the rigid-body equations of motion.

Another important special case arises with our subject here: elastic bodies and fluids. In many cases, they can be approximated by regular lattices of mass points and springs. While these systems can be simulated with simple general methods, the performance is often very disappointing. The reason for the poor performance is relatively easy to understand because the regular lattices are particularly amenable to analysis. Understanding the difficulties leads to formulations of solution methods which exploit the regular structure and thereby achieve far more efficient behavior.

2 Spring with Mass

In idealized mass-spring systems, the springs are considered to have zero mass. This is primarily because such systems are easy to simulate, not because real springs are massless. Here we will use idealized springs to simulate a more realistic spring with mass distributed along its length. With this change, the behavior of the simulated spring becomes a great deal more complex than an idealized spring and includes full wave behavior along its length, much like a “Slinky.”

Figure 1 shows a large zero-rest length spring modeled as a collection of n smaller springs in series. If the mass of the entire spring is m, then the masses between the smaller springs are $m_i = m/(n+1)$. For simplicity and ease of exposition, we will initially consider the one-dimensional problem in which all forces and motion are limited to the x-axis. Let x_i denote the position of the ith mass point. If the spring constant of the ith spring is k_i, the energy of the collection of small springs

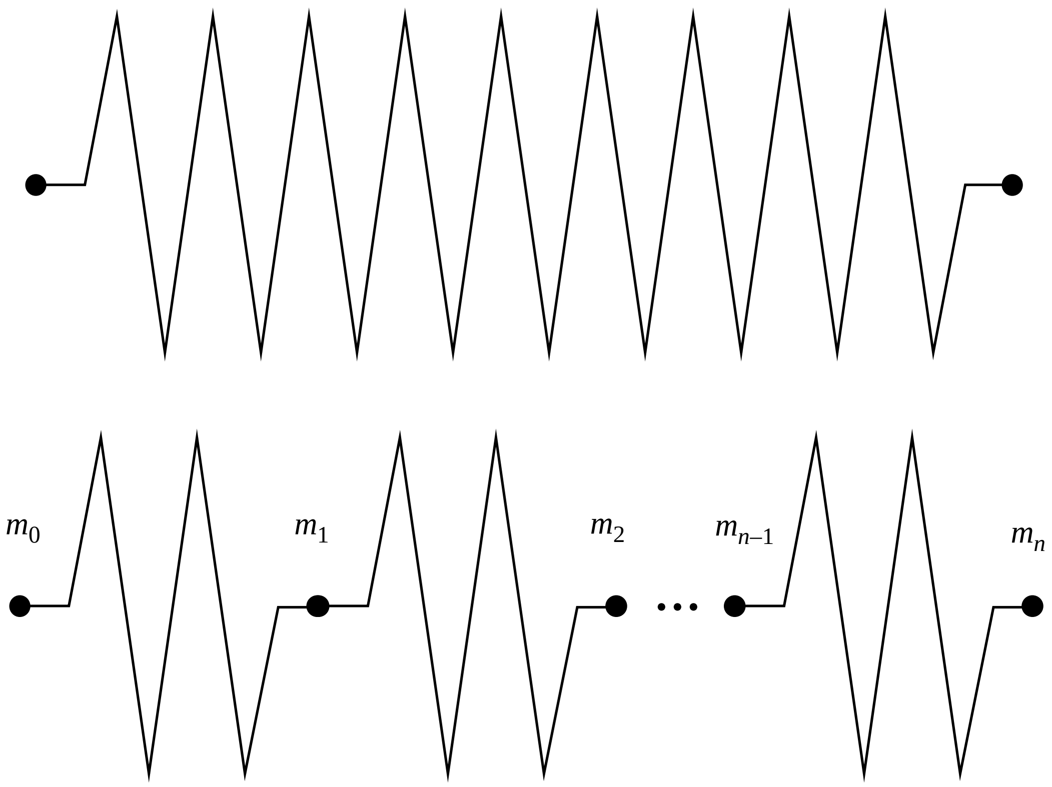

Figure 1: Single spring (top) is simulated by a collection of smaller springs connected in series (bottom).

can be written

$$E = \frac{1}{2}\sum_{i=1}^{n} k_i (x_i - x_{i-1})^2 \tag{1}$$

When the small springs are equally stretched out, their energy should sum to the energy of the entire spring. This will happen when $k_i = nk$ where k is the spring constant of the entire spring. Substituting into the energy expression yields

$$E = \frac{nk}{2}\sum_{i=1}^{n} (x_i - x_{i-1})^2 \tag{2}$$

3 Continuous limit

Additional insight into this energy function can be achieved by examining the limit as the number of springs n goes to infinity. Then x becomes a continuous function of a parameter u instead of being indexed by i . In the limit, the energy function can be written

$$E = \lim_{n\to\infty} \frac{nk}{2}\sum_{i=1}^{n} k_i (x_i - x_{i-1})^2 = \lim_{n\to\infty} \frac{k}{2n}\sum_{i=1}^{n} \left(\frac{x(i/n) - x((i-1)/n)}{1/n} \right)^2 \tag{3}$$

The term being summed on the right-hand side of converges to the square of the derivative of x. The summation converges to an integral. As a consequence, the energy function can be written

$$E = \frac{k}{2}\int_0^1 \left(\frac{\partial x(u)}{\partial u} \right)^2 \tag{4}$$

in the limit as n goes to infinity. It is also possible to arrive at this energy expression directly from arguments about the infinitesimal properties of springs.

The continuous energy expression in equation 4 is very similar to a continuous energy expression of great importance in computer graphics: the energy

$$E = \frac{k}{2}\int_0^1 \left(\frac{\partial^2 x(u)}{\partial u^2} \right)^2 \tag{5}$$

of a natural spline. The only difference is that the spring energy of equation 4 depends on first derivatives, while the spline energy of equation 5 depends on second derivatives. When these energy functions are extended to surfaces, the spring energy corresponds to the behavior of a membrane, while the spline energy corresponds to the energy of a thin plate.

4 Discretization

Suppose we are given a continuous energy function such as equation 4 or 5 and we wish to simulate its behavior with a computer. Before we can simulate the behavior, we must decide on a representation for the continuous function $x(u)$. There are two basic choices: we can represent $x(u)$ by its values at a set of sample points, or we can represent it by a collection of coefficients of local basis functions (e.g. the control points of a spline). The first is known as the *finite-difference* method and the second as the *finite-element* method. Here we will examine the finite-difference method.

If the function $x(u)$ is represented by $n+1$ samples evenly distributed on the interval from zero to one with $x_i = x(i/n)$, $0 \le i \le n$, the derivatives can be approximated by the expressions

$$\left.\frac{\partial x}{\partial u}\right|_{x=ih} \approx \frac{x_{i+1} - x_i}{h} \tag{6}$$

and

$$\left.\frac{\partial^2 x}{\partial u^2}\right|_{x=ih} \approx \frac{x_{i+1} - 2x_i + x_{i-1}}{h^2} \tag{7}$$

where $h = 1/n$ is the spacing between sample values of u. The approximation of equation 7 is derived by using the approximation of equation 6 twice, once for each differentiation. If the approximation of equation 6 is combined with the energy function of equation 4 and the integral in the energy function is converted into a sum over the sample values, then the result is exactly the energy expression of 2. We therefore have two different interpretations of the spring energy in equation 2. One interpretation is the approximation of a large massive spring by a collection of smaller springs connected to point masses. The other interpretation is that the energy in equation 2 is a discrete approximation to the continuous energy function of 4. If the approximation of equation 7 is combined with the continuous energy function of 5, the result is the following discrete energy function:

$$E = \frac{n^2 k}{2} \sum_{i=1}^{n} (x_{i+1} - 2x_i + x_{i-1})^2 \tag{8}$$

Like the energy function in equation 2, this energy has an interpretation in terms of springs, but it is not nearly so simple. The energy corresponds to springs between x_i and the midpoint between x_{i-1} and x_{i+1}.

5 Euler's Method

In order to simulate the motion of physical system from a discrete energy like equation 2, we have to compute the force from the energy expression and then use Newton's second law $F = ma$ to calculate the accelerations. We then integrate the accelerations through time to compute the velocities and integrate the velocities to compute the positions of the mass points.

The force on the ith mass point is given by the partial derivative of the energy with respect to x_i. For the energy of 2, we have

$$F_i = \frac{\partial E}{\partial x_i} = \begin{cases} -nk(x_1 - x_0) & \text{if } i = 0; \\ nk(x_n - x_{n-1}) & \text{if } i = n; \\ nk(-x_{i+1} + 2x_i - x_{i-1} & \text{otherwise.} \end{cases} \tag{9}$$

The reason for the special cases is that there are two springs pulling on each mass point except for the first and the last. Note that if the spring is stretched out evenly $(xi - xj = c(i - j))$ then the force on all the internal mass points $(0 < i < n)$ is zero because the force from the spring on the left precisely balances the force from the spring on the right.

With the above force expressions, we must still choose an integration method for the differential equation $F = ma$. Let us first consider the behavior of the simplest integration method, Euler's method. In order to apply Euler's method, we have to transform the differential equation into the canonical form $dY/dt = f(Y)$. Then if we are given the value of Y at time t, we can compute the value of Y at time $t + \Delta t$ from the Euler update formula

$$Y(t + \Delta t) = Y(t) + (\Delta t)\, f(Y). \tag{10}$$

The equation $F = ma$ is not in canonical form because the acceleration a is a second derivative and the canonical form requires relationships involving first derivatives of the state variables. This difficulty is easily remedied by introducing new variables to represent the velocities. Then we can write two sets of equations involving first derivatives instead of one involving second derivatives. Let v_i denote the velocity of the ith mass point. Then $F = ma$ in canonical form can be written as follows:

$$\begin{aligned} \frac{dv_i}{dt} &= \frac{F_i}{m_i} \\ \frac{dx_i}{dt} &= v_i. \qquad (11) \\ & \qquad\qquad (12) \end{aligned}$$

Using this canonical form, the Euler update formula can be written

$$\begin{aligned} v_i(t + \Delta t) &= v_i(t) + (\Delta t) F_i(t)/m_i \\ x_i(t + \Delta t) &= x_i(t) + (\Delta t) v_i(t). \end{aligned} \tag{13}$$

If the time step Δt is small enough, the above equations provide a means for simulating the behavior of the spring with mass shown in . Unfortunately, the required time step is very small, particularly if n, the number of small springs, is very large. In fact, as we will see shortly, the time step Δt must be of order $1/n$ to ensure that the iteration in equation 13 does not diverge.

Suppose that the spring is initially at rest with $x_i = i/n$ and appropriate forces are applied to both ends in order to keep it at rest. That is, an external force $nk(x_1 - x_0)$ is applied to the left end, and an external force $-nk(x_n - x_{n-1})$ is applied to the right end. The net force on each mass point is zero, so the system is in equilibrium. Now suppose that a large force P is suddenly applied to the left-most mass point m_0. The sudden application of the force will cause a compression wave to move from the left edge towards the right. Physical intuition tells us that the speed of the wave depends on the relationship between the stiffness of the little springs and the mass of the points in between. If the springs are very stiff and the masses are light, we expect the wave to travel very rapidly. Conversely, if the springs are floppy and the masses are heavy, we expect the wave to travel slowly. We will make this notion precise in the next section. For now, we will look at how Euler's method simulates the effect.

Suppose the large force P is applied to the left-most mass point at time zero. In the first Euler step, only the left-most mass point m_0 will accelerate. All the other mass points have zero net force, so they remain at rest. At the end of the first Euler step m_0 will have a non-zero velocity, but it still will not have moved. Only after the second Euler step will m_0 have moved from its equilibrium position. All the other mass points remain in their original positions. Looking at the force expression in equation 12, we see that the force is still zero except on mass point m_1. The reason is that the force on each mass point depends only on the positions of its immediate neighbors. After two more Euler steps, m_1 will have moved and m_2 will feel a non-zero force. Only after $2n$ Euler steps will the right-most mass point feel any effect from the initial disturbance on the left-most mass point.

Let τ be the time that it should take a disturbance to propagate from one end of the spring to the other in the real physical situation. We know that it has to take at least $2n$ Euler steps for this to happen in the simulation, so the time step Δt must be less than $\tau/2n$ in order for the simulation to work.

If we try to simulate the behavior with a larger time step, the Euler iteration will simply diverge. Now the problem with Euler's method should be clear. As we increase the number of samples along the spring, we have to keep reducing the time step because information cannot travel faster than one sample per Euler step. Since each Euler step requires a fixed amount of work per sample, the amount of computational work needed to simulate the disturbance propagating down the length of the spring is n^2 because it requires n iterations each with work proportional to n. If we want to simulate elastic surfaces with thousands of samples, we clearly need a better simulation technique.

6 The Wave Equation

We can gain further insight into the nature of the problem by examining the continuous equations that we are approximating with the collection of springs. Away from the ends of the whole spring, the acceleration F_i/m_i of a mass point is given by

$$\frac{F_i}{m_i} = \frac{n(n+1)}{m} k(-x_{i+1} + 2x_i - x_{i-1}) \tag{14}$$

The right hand side of equation 14 has the familiar look of the second derivative approximation we saw in equation 7. In the limit as n goes to infinity, the second derivative returns

$$\lim_{n\to\infty} \frac{F_i}{m_i} = \frac{k}{m} \frac{\partial^2 x}{\partial u^2} \tag{15}$$

and we are left with the wave equation:

$$\frac{\partial^2 x}{\partial t^2} = \frac{k}{m} \frac{\partial^2 x}{\partial u^2} \tag{16}$$

It is also possible to derive equation 16 directly from the energy expression of equation 4 using a mathematical technique known as the *calculus of variations*.

The wave equation has been studied extensively for a long time because of its importance in a number of areas of physics. Its solution is well known.

$$x(u,t) = T_1\left(u - t\sqrt{k/m}\right) + T_2\left(u + t\sqrt{k/m}\right) \tag{17}$$

The function $x(u,t)$ is given by the sum of a waveform T_1 translating in the positive x direction and another waveform T_2 translating in the negative x direction. The translation velocity is given by the square root of the constant of proportionality between the second spatial derivative and the second temporal derivative, in this case k/m. If this solution is substituted back into equation 16, it is easy to verify that it satisfies the differential equation.

Now we have another perspective on the difficulties with Euler's method. If the spring constant k is very large compared to the mass, then the effect of a force applied at one point in the spring will rapidly spread up and down the length of a spring. A very small time step will be required to combat the problem that Euler's method limits propagation speed to no more than one sample per iteration.

7 Implicit Integration

Not all the difficulty in simulating the massive spring is due to Euler's method. Part of it is inherent in the problem statement. Consider for a moment the energy landscape given by equation 4. If the springs are all equally stretched out, the energy is at a local minimum. If any one mass point is moved from this position by itself, the energy increases very rapidly (approximately as the square of the displacement). On the other hand, if all the mass points are moved together by the same amount, the energy does not change at all. While it is somewhat difficult to visualize because it exists in an n dimensional space, this is the description of an energy landscape that acts like a ravine. If you move in most directions, the energy increases rapidly. If you pick a direction near the axis of the ravine, the energy varies very slowly. Note that the sides of the ravine can be made arbitrarily steep by increasing the spring constant k.

The correct solution to the differential equation will tend to follow along the ravine in the energy function unless large external forces are pulling it away. The problem with Euler's method is that it tries to follow along the ravine without knowing that a ravine is present. Its local knowledge of the energy function is limited to computing the force at the current state. Since the force is the gradient of the energy, Euler's method computes its next step using a local planar model of the energy function. Unfortunately, for Euler's method, a plane is a very poor model of a ravine, so the method ends up doing a very bad job. The only way to make the planar model a good approximation of a ravine is to consider a very small region. As a result, Euler's method will only work on this problem when very small step sizes are used.

To improve the performance, we need to incorporate a better model of the energy function into the differential equation solver. Since first derivatives of the energy are insufficient, the obvious improvement is to look at second derivatives. In this case, second derivatives of the energy correspond to first derivatives of the forces.

7.1 General Implicit Formulation

Consider once again the canonical differential equation form $dY/dt = f(Y)$. Instead of assuming (as Euler's method does) that the derivative dY/dt throughout the time interval is simply $f(Y_0)$, let us assume that it is some weighted average of the derivative $f(Y_0)$ at the beginning of the interval and $f(Y(t+\Delta t))$ at the end of the interval. Then we can write the update

$$Y(t+\Delta t) = Y(t) + (\Delta t)\left[(1-\lambda)f(Y(t)) + \lambda f(Y(t+\Delta t))\right]. \tag{18}$$

where λ is a constant between zero and one. Note that when $\lambda = 0$, this reverts to the ordinary Euler update of equation 10. Any update of this form where $f(Y(t+\Delta t))$ appears on the right is known as an *implicit* update formula. When $\lambda = 1$, this equation is known as the *backwards Euler* or *implicit Euler* update.

Since we do not know $Y(t+\Delta t)$ at the beginning of the step, it may not be clear that equation 18 is helpful in calculating $Y(t+\Delta t)$. This is where we make use of the additional derivative information. Around the current state Y_0, we have the approximation

$$\frac{dY}{dt} \approx f(Y_0) + (Y - Y_0)(\nabla f)|_{Y=Y_0} \tag{19}$$

Substituting this approximation into equation 18 we have

$$\Delta Y = (\Delta t)\left[f(Y_0) + \lambda \Delta Y (\nabla f)|_{Y=Y_0}\right] \tag{20}$$

Solving for ΔY, we obtain the update formula

$$\Delta Y \left[\frac{1}{\Delta t} I - \lambda (\nabla f)|_{Y=Y_0}\right] = f(Y_0) \tag{21}$$

where I denotes the identity matrix. Note that computing the update for Y over a time step now requires solving a linear system. This is the price of using the extra derivative information.

For the greatest accuracy, we should use $\lambda = 1/2$ since the best estimate of the derivative in the middle of the interval is halfway between its value at the beginning and the end. There is another criterion that affects our choice, however, and that is the issue of stability. To understand the stability of the update, let us consider the the single variable linear case $dY/dt = -cY$ where c is a positive constant and where the initial condition is $Y(0) = \gamma$. Then we have

$$\begin{aligned} Y(t+\Delta t) &\approx Y(t) + \Delta t[(1-\lambda)Y'(t) + \lambda Y'(t+\Delta t)] && (22)\\ &= Y(t) - c\Delta t[(1-\lambda)Y(t) + \lambda Y(t+\Delta t)] && (23)\\ &= \frac{(1-c\Delta t(1-\lambda))Y(t)}{1+\lambda c\Delta t} && (24)\\ &= \gamma\left(\frac{(1-c\Delta t(1-\lambda))}{1+\lambda c\Delta t}\right)^{t/\Delta t}. && (25) \end{aligned}$$

Clearly, the update will diverge if the fraction in equation 25 is greater in magnitude than one. Since c is positive, this can occur only when $\Delta t > 2/[c(1-2\lambda)]$. When $\lambda = 0$, we have the ordinary Euler update, and the calculated $Y(t)$ diverges if $\Delta t > 2/c$. In other words, the maximum time step is inversely proportional to the constant of proportionality between Y and Y'. As λ increases, the scheme gets more and more stable. If $\lambda \geq 1/2$, the iteration cannot diverge for this linear differential equation. When applied to non-linear differential equations, however, there is still a possibility of divergence. If λ is increased from $1/2$ to one, the iteration gets even more stable. For severely nonlinear problems, the additional stability, even though it comes at the expense of some reduction in accuracy, can be very useful.

7.2 Application To Springs

When we apply equation 21 to the spring system, there are some simplifications because the force depends only on the positions of the mass points and not on their velocities (unless we add damping forces). As a result, we end up with the same equation for the velocities as in the Euler step, but a very different update formula for the positions:

$$(\Delta x)A = v \tag{26}$$

where

$$A = \left[\frac{1}{\Delta t}I - \lambda H\right] \tag{27}$$

where

$$H = \frac{\partial^2 E}{\partial x_i \partial x_j} \tag{28}$$

is the Hessian matrix of the spring energy.

For some problems, this solution method could take even more work than Euler's method because solving a general linear system requires computational work proportional to n^3. In this case, however, the special structure of the problem makes it possible to solve the linear system with computational work proportional to n. As a result, the method is far faster than Euler's method for cases of practical interest.

Remember that the force on the ith mass point depends only on the positions of mass points $i-1$ through $i+1$. As a result, the derivative of the force on the ith mass point is zero, except with respect to mass points $i-1$, i, and $i+1$. This means that every row of the matrix A has at most three non-zero elements: the element on the diagonal and one entry on either side.

$$A = \begin{pmatrix} a_0 & b_0 & & & & & \\ b_0 & a_1 & b_1 & & & \mathbf{0} & \\ & b_1 & a_2 & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & a_{n-3} & b_{n-3} & \\ & \mathbf{0} & & & b_{n-3} & a_{n-2} & b_{n-2} \\ & & & & & b_{n-2} & a_{n-1} \end{pmatrix} \tag{29}$$

Matrices with this special form are known as tridiagonal matrices, and linear systems involving them are particularly easy to solve. Numerical Recipes [1], for example, includes a seventeen line program which solves tridiagonal matrices in time proportional to n. The algorithm works by factoring the matrix into the product $A = LU$ of a lower triangular matrix L and an upper-triangular matrix U where

$$L = \begin{pmatrix} c_0 & & & & & & \\ d_0 & c_1 & & & & \mathbf{0} & \\ & d_1 & c_2 & & & & \\ & & \ddots & \ddots & & & \\ & & & \ddots & c_{n-3} & & \\ & \mathbf{0} & & & d_{n-3} & c_{n-2} & \\ & & & & & d_{n-2} & c_{n-1} \end{pmatrix} U = \begin{pmatrix} r_0 & s_0 & & & & & \\ & r_1 & s_1 & & & \mathbf{0} & \\ & & r_2 & \ddots & & & \\ & & & \ddots & \ddots & & \\ & & & & r_{n-3} & s_{n-3} & \\ & \mathbf{0} & & & & r_{n-2} & s_{n-2} \\ & & & & & & r_{n-1} \end{pmatrix} \tag{30}$$

It then solves the equation $AX = B$ by first solving $LY = B$ for Y and then solving $Y = UX$ for X. Consider solving $LY = B$ for Y. The first component is easy: $Y_0 = B_0/c_0$. After that, we can solve for the remaining components of Y using the recurrence $Y_i = (B_i - Y_{i-1}d_{i-1})/c_i$. To get an idea of what this recurrence does, let us examine the special case where $d_i = 1/a$ and $c_i = (a-1)/a$. This is representative of the behavior away from the boundaries. In this case, we can rewrite the recurrence as

$$Y_i = \alpha B_i + (1-\alpha)Y_{i-1}. \tag{31}$$

Some will recognize equation 31 as a simple recursive filter. The next output value Y_i is a blend of the last output value Y_{i-1} and the current input value B_i. The effect of any particular input decays exponentially as it gets repeatedly blended in with the subsequent inputs.The result is a low-pass filter with an effective smoothing width determined by α. The output of the filter when given an impulse as input is shown in b. When the subsequent linear system $Y = UX$ is solved for X, the result is to do essentially the same thing but in the reverse direction. Figure 2c shows the effect of the two combined filtering operations is shown. The initial spike is blurred out symmetrically in both directions. Note that the behavior will be somewhat different near the boundaries because d_i and c_i will not be constant.

Now it should be clear how the implicit solution method of avoids the problems with Euler's method. When a force is applied to a single mass point, the low-pass filtering applies the effect of the force to all the neighboring mass points. If k is large, then the coefficients of the recursive filter

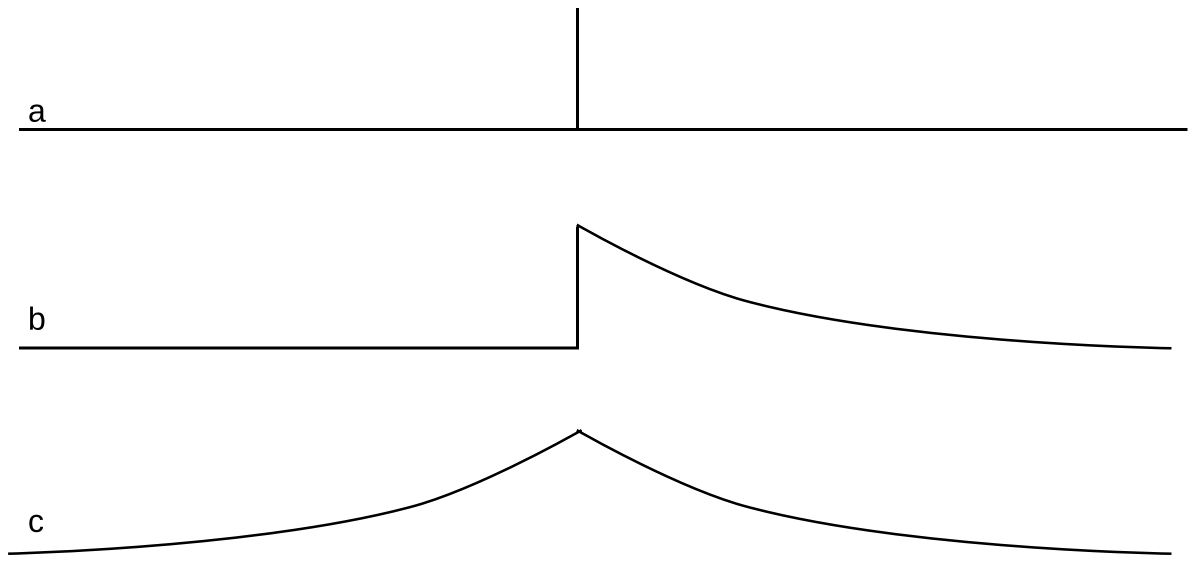

Figure 2: Effect of recursive filter. Input (a) is filtered forward (b) and and then backward (c).

become such that the width of the filter is correspondingly large. In the limit as k grows very large, the filter will essentially move all the mass points together as one. With this solution method, the number of iterations required for a disturbance to propagate from one end of the spring to the other is independent of n. The coefficients of the filters adjust to take account of the different number of samples. Since the computational work per iteration is proportional to n, the entire simulation takes computational work proportional to n instead of the n^2 work required by Euler's method. This can be the difference between having a practical simulation and having an impractical one.

8 Fluids

With a very small modification, the spring with mass can be made to simulate the behavior of shallow water. While the derivation of the equations is fairly different, the end result is much the same. If we divide a volume of water into a collection of vertical columns as in figure 3 and make use of a set of approximations known as the shallow water equations, we can describe the motion of the surface by a wave equation in which the wave velocity is proportional to the square root of the depth:

$$\frac{\partial^2 h}{\partial t^2} = gd\frac{\partial^2 h}{\partial x^2}. \tag{32}$$

where h is the height of the column of water, $d = h - b$ is its depth, and g is the gravitational constant.

In [2], the discretization

$$\frac{\partial^2 h}{\partial t^2} = -g\left(\frac{d_{i-1}+d_i}{2(\Delta x)^2}\right)(h_i - h_{i-1}) + g\left(\frac{d_i+d_{i+1}}{2(\Delta x)^2}\right)(h_{i+1} - h_i) \tag{33}$$

is presented. The main difference between these fluid equations and the spring equations we have been examining is that the spring constant is no longer uniform. As a result, the differential equation becomes nonlinear and it becomes useful to have $\lambda = 1$.

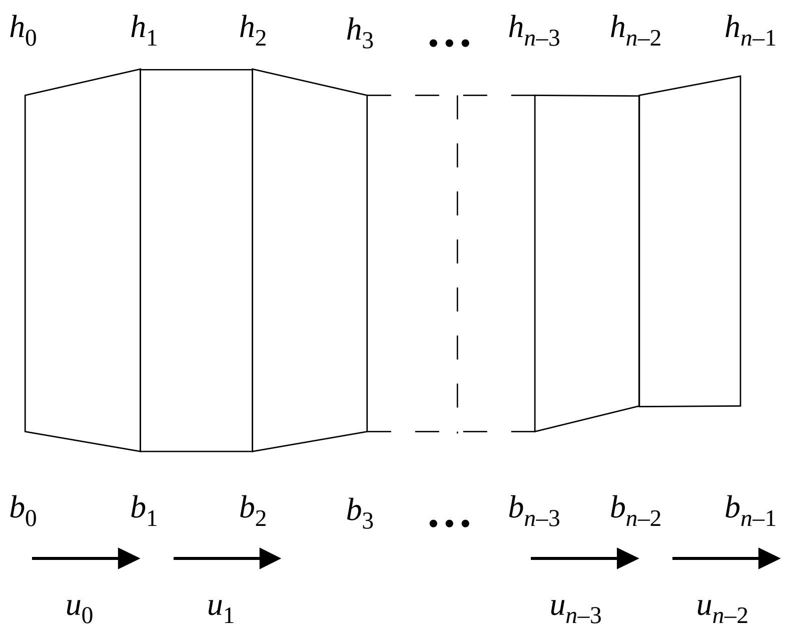

Figure 3: Discrete two-dimensional height-field representation of the water surface *h*, the ground bottom *b*, and the horizontal water velocity *u*.

The iteration required to solve equation 33 with the implicit method of equation 21 consists of solving a tridiagonal linear system. Precise formulas for the entries in the linear system and many further details can be found in [2].

9 Surfaces

So far, we have been considering the one-dimensional problem of a spring with mass. If we use a rectangular lattice of springs, we can simulate an elastic membrane surface. For the most part, the one-dimensional analysis still applies. We still have difficulties with forces propagating over large numbers of samples. On a surface, however, the problem is more severe because the matrix equations we get from the implicit solution technique are no longer tridiagonal. The reason is that mass points are tied to their neighbors along both rows and columns. No matter how the mass points are indexed in the matrix, it will come out with a more complicated structure than the tridiagonal situation in one dimension. One effective solution to this difficulty is to consider rows and columns of the spring lattice alternately. Then the one-dimensional technique can be used on each row or column separately. This works very well, for example, in the case of shallow fluids.

10 Conclusion

Regular lattices of masses and springs have some special properties that are important to be aware of for the purposes of simulation. If the number of nodes in the lattice is large, then the choice of solution technique is very important. A naive use of Euler's method can result in computational cost proportional to the square of the number of nodes. This is because the effects of forces can never propagate faster than one sample per iteration. Using implicit techniques can avoid these problems and produce a computational cost which is linear in the number of nodes.

References

[1] W. Press, B. Flanner, S. Teukolsky and W. Vetterling,. *Numerical Recipes: The Art of Scientific Computing* Cambridge University Press, Cambridge 1986.

[2] Michael Kass and Gavin Miller. Rapid, Stable Fluid Mechanics for Computer Graphics. In *Proceedings of SIGGRAPH '90*, pages 49–57, August 1990.