Numerical Methods in Cloth Simulation

PC3236 Project Report – Wayne Tzu-Wen Wu

Introduction

Cloth simulation has been a problem of interest in the field of computer graphics. The ability to produce visually realistic cloth motion is crucial in many applications such as 3D animation, visual effects and games.

This project stems from multiple well-known researches on cloth simulation with the goal to implement and analyze the common numerical techniques used to model the physical behaviour of cloth. The simulation will be performed entirely in MATLAB.

Physical Model

Physically-based cloth simulation uses Newton's 2nd Law to describe the geometrical state of the cloth:

$$\ddot{x} = M^{-1}(-\frac{\partial E}{\partial x} + F)$$

In this case, x and M represents the state (i.e. position) and mass of the cloth respectively. *E* is the internal energy of the cloth while F consists of the external forces (i.e. gravity, air drag etc.).

In most state-of-the-art researches, the partial differential equation above is discretized and solved by modeling the cloth as a system of particles. These particles are connected to form a mesh that represents the cloth. Naturally, the more particles in the system, the more accurate the representation.

With spatial discretization, the partial differential equation is now transformed into a system of ordinary differential equations, where each ordinary differential equation governs the state of the corresponding mass point (particle) using the same physical model.

Since the internal force is expressed as the gradient of the internal energy, the partial differential equation can be rewritten as

$$\ddot{x} = M^{-1}(F_{int} + F_{ext})$$

or simply,

$$\ddot{x} = M^{-1}f(x, \dot{x})$$

where *f* is the final combined net force. Given that this is a second order differential equation, it is converted into two first order differential equations as

$$\begin{bmatrix} x \\ \dot{x} \end{bmatrix}' = \begin{bmatrix} x \\ v \end{bmatrix}' = \begin{bmatrix} v \\ M^{-1}f \end{bmatrix}$$

which are to be solved numerically.

Dynamics

The essence of the physical model comes from *f*, of which the internal force contribution is responsible for producing cloth-like behaviour. In this project, the internal force equations were taken directly from [1] and [2], which models the internal force using a mass-spring system. The detailed derivation of the equations will not be explained in this report.

In short, there are two types of force interactions between neighbouring particles, which are called Type 1 and Type 2 for simplicity. Type 1 models the stretching force due to adjacent particles (i±1) while Type 2 models the compression force due to particles two spaces apart (i±2). Figure 1 shows the two types of interactions.

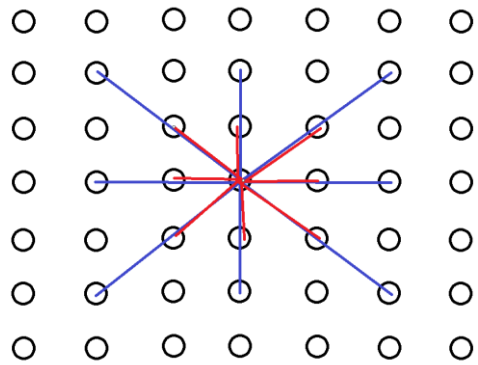

Figure 1. Type 1 interactions (red) and Type 2 interactions (blue)

Type 1 is based on the linear elastic equation and is defined as

$$f_i = \begin{Bmatrix} k_s(|x_{ij}| - L), & |x_{ij}| \geq L \\ 0, & |x_{ij}| < L \end{Bmatrix}$$

$\boldsymbol{f_i}$ is the force acting on i due to its elastic connection with particle j. k_s is the spring constant, and $\mathbf{x_{ij}}$ is the vector defined as $\mathbf{x_j - x_i}$.

Type 2 is less trivial and was derived in [2] to produce a better cloth "buckling" effect. It models the force based on the arc created when being compressed. The equations are given as the following:

$$f_i = f_b{}^*(\frac{x_{ij}}{|x_{ij}|})$$

$$f_b{}^* = \begin{Bmatrix} c_b(|x_{ij}| - L) : f_b < c_b(|x_{ij}| - L) \\ f_b \quad : otherwise \end{Bmatrix}$$

$$f_b = k_b k^2 \left(\cos\left(\frac{kL}{2}\right) - sinc\left(\frac{kL}{2}\right) \right)^{-1}$$

$$k = \frac{2}{L} sinc^{-1}(\frac{|x_{ij}|}{L})$$

In this case, k_b is the flexural rigidity while c_b is the compression constant, similar to k_s.

Finally, to account for energy dissipating in the system, a damping force is included in the model for every interactive force (Type 1 and Type 2). The damping force is defined as

$$fi = k_d(v_i - v_j)$$

where k_d is the damping coefficient.

With these two types of interactions as well as the damping force, the cloth internal characteristics can be reasonably modelled, and the result has been proven to be visually appealing [#].

Numerical Scheme: Explicit

Using the model described above, the cloth simulation involves a system of ordinary differential equations, which is typically solved using explicit or implicit scheme, with the later being more popular for its stability. In this project, both explicit and implicit methods are explored.

For the explicit scheme, Runge-Kutta's 4th order method (RK4) is used to solve the differential equations. The numerical equations of RK4 for the system is defined as

$$\boldsymbol{f_0} = \boldsymbol{F}(t, \boldsymbol{Y})$$

$$\boldsymbol{f_1} = \boldsymbol{F}\left(t + \frac{h}{2}, \boldsymbol{Y} + \frac{h}{2}\boldsymbol{f_0}\right)$$

$$\boldsymbol{f_2} = \boldsymbol{F}\left(t + \frac{h}{2}, \boldsymbol{Y} + \frac{h}{2}\boldsymbol{f_1}\right)$$

$$\boldsymbol{f_3} = \boldsymbol{F}(t + h, \boldsymbol{Y} + h\boldsymbol{f_2})$$

$$\boldsymbol{Y}(t_0 + h) = \boldsymbol{Y}(t_0) + \frac{h}{6}(\boldsymbol{f_0} + \boldsymbol{2f_1} + \boldsymbol{2f_2} + \boldsymbol{f_3})$$

, where for this cloth system

$$\boldsymbol{Y} = \begin{bmatrix} x \\ v \end{bmatrix}, \quad \boldsymbol{F} = \begin{bmatrix} v \\ M^{-1} f(Y) \end{bmatrix}$$

It should be noted that **F** is not directly dependent on time. Therefore, the t component in the RK4 equations are not used in evaluation.

Numerical Scheme: Implicit

It is known that the behaviour of cloth produces a stiff problem [1]. Therefore, explicit scheme is in fact ill-suited for cloth simulation. Since the introduction to using implicit scheme for cloth simulation [1], implicit method has been the more preferred choice.

This project uses the backward Euler's method as the implicit scheme for the simulation, in similar fashion as [1]. To begin with, backward Euler's method is defined as

$$y^{n+1} = y^n + hf^{n+1}$$

Applying it to our differential equations:

$$\begin{bmatrix} \Delta x \\ \Delta v \end{bmatrix} = h \begin{bmatrix} v^{n+1} \\ M^{-1} f^{n+1} \end{bmatrix}$$

where $\Delta\mathbf{v} = \mathbf{v}^{n+1} - \mathbf{v}^n$ and $\Delta\mathbf{x} = \mathbf{x}^{n+1} - \mathbf{x}^n$.

To solve the differential equation, $\boldsymbol{f}^{n+1}$ must first be evaluate, which is usually approximated using first order Taylor Series expansion:

$$f^{n+1} = f^n + \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial v}\Delta v$$

This approximation makes the scheme semi-implicit rather than truly implicit. Replacing $\boldsymbol{f}^{n+1}$ with the approximation, the velocity differential equation becomes

$$\Delta v = h\left(M^{-1}\left(f^n + \frac{\partial f}{\partial x}\Delta x + \frac{\partial f}{\partial v}\Delta v\right)\right)$$

Substituting Δx and rearranging Δv to one side:

$$\left(I - h^2 M^{-1}\frac{\partial f}{\partial x} - h\frac{\partial f}{\partial v}M^{-1}\right)\Delta v = hM^{-1}(f^n + h\frac{\partial f}{\partial x}v^n)$$

Multiply both sides by M, the final equation is obtained as

$$\left(M - h^2\frac{\partial f}{\partial x} - h\frac{\partial f}{\partial v}\right)\Delta v = h(f^n + h\frac{\partial f}{\partial x}v^n)$$

or simply,

$$\boldsymbol{A\Delta v = b}$$

where

$$A = M - h^2\frac{\partial f}{\partial x} - h\frac{\partial f}{\partial v}$$

$$b = h(f^n + h\frac{\partial f}{\partial x}v^n)$$

Therefore, to calculate the position and velocity at the next time step, the A must be solved to find $\Delta\mathbf{v}$. The values can then be obtained by adding the delta values accordingly.

In the above equations, the Jacobians, df/dx and df/dv, must first be calculated to solve the system of equations. Each type of forces has its own Jacobian, which are derived using vector calculus. The derivations will not be shown in this paper.

For Type 1, the Jacobian matrix is

$$\frac{\partial f_i}{\partial x_j} = \begin{cases} k_s\frac{x_{ij}x_{ij}^T}{x_{ij}^T x_{ij}} + k_s\left(1 - \frac{L}{x_{ij}}\right)\left(I - \frac{x_{ij}x_{ij}^T}{x_{ij}^T x_{ij}}\right) & : x_{ij} \geq L \\ 0 & : x_{ij} < L \end{cases}$$

For Type 2, the Jacobian matrix is

$$\frac{\partial f_i}{\partial x_j} = \frac{df_b^*}{d|x_{ij}|}\frac{x_{ij}x_{ij}^T}{x_{ij}^T x_{ij}} + \frac{f_b^*}{|x_{ij}|}(I - \frac{x_{ij}x_{ij}^T}{x_{ij}^T x_{ij}})$$

For the damping force, the Jacobian matrix is

$$\frac{\partial f_i}{\partial v_j} = k_d I$$

Using these equations, the net Jacobians can be obtained by summing up all the elements yielding 3N x 3N sized matrices.

Vectorized Structure

Before implementing the solvers, it is important to first understand and establish the optimal structure for representing the system of particles. Perhaps one of the most intuitive ways is to create a struct type storage for each particle, storing its mass, position and velocity. The solvers would then run on every particle to calculate the state of the particle at the next time step, making it a very slow process.

Since the simulation is carried out in MATLAB®, vectorized computation should be leveraged for optimal performance. As such, rather than storing each particle individually, the geometrical state of all particles is concatenated. The state variables, **x** and **v,** are represented as vectors with size of 3N, where N is the number of particles. M is then created as a diagonal matrix with the size of 3N x 3N. Together, the velocity differential equation can be viewed as

$$\begin{bmatrix} v_{1x} \\ v_{1y} \\ v_{1z} \\ v_{2x} \\ v_{2y} \\ v_{2z} \\ \vdots \end{bmatrix}' = \begin{bmatrix} m_1 & 0 & 0 & & \cdots & & \cdots \\ 0 & m_1 & 0 & & & & \\ 0 & 0 & m_1 & & & & \\ & & & m_2 & 0 & 0 & \\ & \vdots & & 0 & m_2 & 0 & \cdots \\ & & & 0 & 0 & m_2 & \\ & \vdots & & & \vdots & & \ddots \end{bmatrix}^{-1} \begin{bmatrix} f_{1x} \\ f_{1y} \\ f_{1z} \\ f_{2x} \\ f_{2y} \\ f_{2z} \\ \vdots \end{bmatrix}$$

This allows the solvers to do vectorized computation on one whole system rather than individual particles, speeding up the computation drastically.

Implementation

A function is created for calculating the net force vector $\boldsymbol{f}$. The function loops through each particle and calculates the net force acting on each particle based on Type 1 and Type 2 interactions.

Type 2 interaction requires the function $\text{sinc}^{-1}(x)$. Since there is no mathematical representation of such function, typically a look up table is created for linear interpolation. For this project, the function is evaluated numerically, by finding the root of the following equation

$$\text{sinc}(x) - a = \frac{sinx}{x} - a = 0$$

where a = sinc(x). The equation is solved using Newton-Raphson's (NR) method. The use of NR scheme does increase the computational cost though it can provide a more accurate result. The number of iterations can also be limited for faster computation.

Another function is also created for calculating the net Jacobian of the system, which is used in the semi-implicit backward Euler equation. The function also loops through each particle and calculates the Jacobians of each force as defined above.

For Type 2 interaction's Jacobian, the derivative of $df_b/d|x_{ij}|$ is calculated using finite difference for simplicity. First order backward Euler is chosen since $sinc^{-1}(x)$ is not defined for x > 1. If other Euler's methods are chosen, x + h would exceed the limit for x = 1.

Validating the calculation obtained from the analytic Jacobian equations can be challenging especially without a full understanding of how the Jacobian describes the motion of cloth. As such, the Jacobians are also calculated numerically using finite difference, and the results are compared.

The central difference formula for an element of partial derivative is

$$\frac{\partial f_i}{\partial x_j} = \frac{f_i(\boldsymbol{x} + \boldsymbol{e_j}h) - f_i(\boldsymbol{x} - \boldsymbol{e_j}h)}{2h}$$

All Jacobian elements were recalculated using the above equation, which yielded a very similar result to the analytic solution, indicating that the implementation of Jacobian is most likely correct.

The implementation of the solver functions (i.e. RK4, Backward Euler) is trivial and simply follows the equations as defined above. However, slight modification is required in the backward Euler method in order to yield a more correct result at larger time steps.

At first, the implementation of the backward Euler's method simply followed the scheme as defined earlier. This yielded similar results compared to RK4 at small time steps (i.e. h = 0.001). However, when the time step was increased to h = 0.01, the solution differed significantly. Since the semi-implicit method has proven to work at large time steps, it was very likely that the solution at h = 0.01 was incorrect.

Upon further investigation, it was discovered that the solution for h = 0.01 was not accurate enough. By plugging the obtained delta values into the original backward Euler equation, new corrected values are obtained.

$$\begin{bmatrix}\Delta x\\ \Delta v\end{bmatrix}^* = h\begin{bmatrix}v^n + \Delta v^*\\ M^{-1}f(x^n + \Delta x, v^n + \Delta v)\end{bmatrix}$$

The * indicates the corrected values, whereas $\boldsymbol{\Delta x}$ and $\boldsymbol{\Delta v}$ are the values obtained from the semi-implicit backward Euler equation.

Using the corrected values, the result of the backward Euler solver is significantly improved. While the correction method can be done iteratively, only one iteration is used.

Boundary Constraints

For more interesting simulation results, points on the cloth should be constrained. For example, certain points on the cloth can be pinned, while observing the rest of the cloth move with respect to such boundary constraints.

This project adapts from the mass modification technique used in [1]. The idea is to first multiply each element in $\boldsymbol{\Delta v}$ by a constant. In [1], $\mathbf{M}^{-1}$ is directly modified to efficiently impose the boundary constraint hence the name mass modification. However, with MATLAB's ability to do element-wise operations, we define another constant vector, **c**, and simply do element-wise multiplication.

$$\Delta \boldsymbol{v} = \boldsymbol{c}.* \Delta \boldsymbol{v}$$

The vector **c** allows you to enforce homogenous boundary conditions by multiplying $\Delta\mathbf{v}$ to 0. This fixes the velocity at its initial value.

Additionally, specific velocity values can also be enforced. For this purpose, another vector, **z**, is introduced which is added to the above equation.

$$\Delta \boldsymbol{v} = \boldsymbol{c}.* \Delta \boldsymbol{v} + \boldsymbol{z}$$

Without any boundary constraint, **z** is normally 0. If boundary condition is enforced, the specific delta velocity for a mass point can be stored in **z**, and using **c** to reject the otherwise calculated Δ**v**, making Δ**v** = **z**.

Result

To verify the stability of the numerical methods, both methods are run through several simulation tests. Since the number of particles and the size of the timestep are most crucial to the stability of the system, these parameters are varied for different simulation tests.

Two types of homogenous boundary constraints are also tested. One boundary constraint fixes the corner four points and allow the rest of the cloth to fall under gravity (Fig. 2).

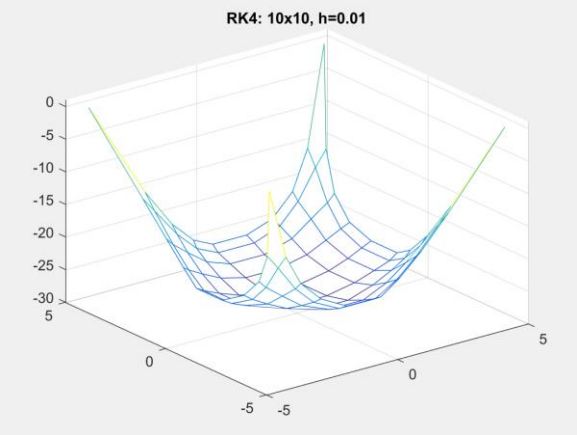

Figure 2. Simple boundary constraint.

The other boundary constraint simulates the cloth being fixed on a table to see the folding effect.

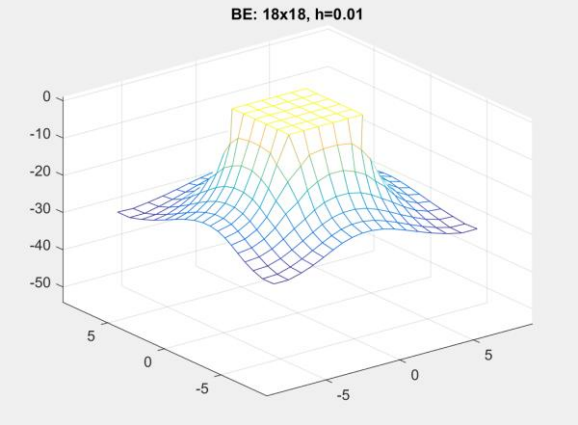

Figure 3. Table boundary constraint.

Below is the result of the simulation tests for both boundary condition.

	RK4	BE
5x5, h = 0.001	Pass	Pass
10x10, h = 0.001	Pass	Pass
20x20, h = 0.001	Pass	Pass
40x40, h = 0.001	Pass	Pass
5x5, h =0.01	Pass	Pass
10x10, h = 0.01	Pass	Pass
20x20, h = 0.01	Pass	Pass
40x40, h = 0.01	Pass	Pass
5x5, h = 0.1	Fail	Fail
10x10, h = 0.1	Fail	Fail

Table 1. Simulation tests result.

Discussion

Base on the simulation result, there are a few problems that should be addressed. To begin with, the implicit scheme did not pass any of the tests at h = 0.1s. This is concerning as all researches done using the same scheme has proven to be stable with timestep as large as h = 0.2s. As such, the backward Euler method that was implemented in this project may in fact not be fully correct.

There are a couple of factors that may have caused this problem, which should be investigated further in the future. The first one concerns with the way the **AΔv = b** is solved. In most researches, the system of equations is solved using the conjugate gradient method [1][2], a type of relaxation-based technique for solving the

system. In this project, the built-in *linsolve* function in MATLAB was used. This may have contributed to the less accurate result, which was fixed only partially when using the correction method.

The inaccuracy may have also come from all the numerical approximations used to calculate derivatives, inverse, etc. Although it is unclear whether other researches have also used similar approaches, coupling numerical methods overwhelmingly may increase the overall error.

The other interesting observation from the simulation test is the unexpected stableness for RK4. Despite the repetitive emphasis on the stiff problem caused by explicit methods, the RK4 has surprisingly passed all the tests that the backward Euler's method did.

This suggests that the test cases may not be sufficient to truly expose the problems of explicit method. In most researches, the test cases involve complex constraints and collision handling which were not implemented in this project for simplicity. However, having such complex system may be necessary in the implementation to understand fully the limitations in explicit methods.

Conclusion

The objective of this project is to implement both explicit and implicit methods for cloth simulation. Despite the implementations for both methods, the slight problem in the backward Euler's method along with the lack of complex test cases yielded an inconclusive result, based entirely from the simulation tests. Further investigation must be conducted to resolve the problem in the implicit scheme at larger time steps.

References

[1] Baraff, David and Witkin A.: Large steps in cloth simulation, Computer Graphics (Proc. SIGGRAPH), pp. 43-54, 1998

[2] Choi, K.-J., Ko H.-S.: Stable but responsive cloth, Computer Graphics (Proc. SIGGRAPH), 2002