Real-Time Cloth Simulation for Games

Benjamin Kenwright†*

Abstract
Generationg realistic real-time cloth effects on-the-fly for interactive environments, such as games, is challenging and interesting. This article gives a practical explanation for students to enable them to integrate cloth effects into their demos. We explain the principles, computational overheads, and numerical approximations, necessary for achieving an aesthetically pleasing realistic interactive cloth effect.

Keywords
Cloth, Soft-Body, Real-Time, Video Games, C++

1 Edinburgh Napier University, School of Computer Science, United Kingdom: b.kenwright@napier.ac.uk

Contents
Introduction 1
1 Overview 1
2 Particles 1
3 Integration 2
4 Cloth Interconnection Structure 2
5 Forces 3
Acknowledgements 3

Introduction
Cloth Effects The creation of fluid-like cloth effects in real-time that are interactive and dynamics valuable and interesting. Through the exploitation of simple approximations (e.g., velocity-less integration and particle mechanics) we are able to create an undoubtedly elegant and beautiful gaming effect.

Where To Put Cloth In The Scene? Cloth effects can be placed in a number of places in our virtual 3D environment, for example curtains on a window or a cape on an animated character. Furthermore, effects from forces, such as wind are noticeable by including cloth effects in the scene. In conclusion, cloth effects bring an otherwise static and repetitive scene to life.

1. Overview
Principles and Concepts Particle system concept (e.g., how we represent each particle and constraint pairs). We then point out computational limitations and pit-falls using naive approaches (e.g., simple triangle edge constraints aren’t enough). We then gives simple code snippets. Finally, we give the complete source code and output.

2. Particles
To achieve real-time frame-rates, we focus on a particle-based approach. The simulation movement of each particle uses Newtonian mechanics (i.e., f=ma). Hence, each particle needs to have a:
• position
• acceleration
• mass

However, since we are constantly working with 1/mass, it’s more convenient and computationally faster to store the inverse mass (i.e., invmass).

3. Integration
Differentiation is used to represent the ‘rate-of-change’ of something (e.g., change in position with time - velocity) - while integration is used to find the opposite (e.g., how the velocity causes a change in position). We apply forces to our
Figure 2. Verlet Integration (Velocity and Position) - 'Velocity'-less integration system (i.e., known as Verlet Integration), which uses the current and previous position.

particles, either from wind or from neighbouring constraints. From Newton’s second law (i.e., F=ma), we can derive the acceleration. Integrating the acceleration with respect to time gives us the velocity. Integrating the velocity with respect to time gives us the position. Hence, we can predict how the velocity and position change in time in relation to the forces we apply.

**Euler Integration**  A first order approximation method that is very popular is known as Euler’s integration, shown below in Equation 1:

\[
\begin{align*}
    v(t+1) &= v(t) + a\Delta t \\
    p(t+1) &= p(t) + v(t+1)\Delta t
\end{align*}
\]

(1)

However, for the simulation to remain stable and realistic, the time-step must be extremely small and the forces (i.e., accelerations) must remain within reasonable limits. Since we’re interested in real-time interactive environments (i.e., games), stability is very important. Hence, we can modify the first order Euler equation to formulate the Verlet integration method (i.e., a velocity-less) which is more stable.

**Verlet Integration**  The Verlet integration method (i.e., velocity-less approach) works by using the current and previous position to create the velocity (i.e., instead of using the exact velocity), as shown below in Equation 2:

\[
\begin{align*}
    v(t) &\approx p(t+1) - p(t)
\end{align*}
\]

(2)

We the velocity approximation in Equation 2 into the first order Euler Equation 1 and the popular Verlet Equation 3 below:

\[
\begin{align*}
    p(t+1) &= 2p(t) - p(t-1) + a\Delta t^2
\end{align*}
\]

(3)

The Verlet integration scheme is simple and fast to implement, for example, see Listing 3 below:

```plaintext
1 Vector3 temp = pos;
2 pos = 2 * pos - old_pos + acceleration*dt*dt
3 old_pos = temp;
```

**Drag/Damping**  We introduce drag/damping to reduce the amplitude of oscillation and ensure the system of interconnected particles converges on a stable still result. Damping effectively causes energy to be lost (i.e., drained) from the system to overcome numerical inaccuracies and approximations to help ensure our simulation remains stable. We modify the uncomplicated Verlet integration Equation 3 to include damping as shown below in Equation 4:

\[
p(t + 1) = p(t) - (2 - k)p(t - 1) + (2 - k)p(t - 2) + a\Delta t^2
\]

\[
= p(t) + [p(t) - p(t - 1)] - k(p(t - 1) - p(t - 2)) + a\Delta t^2
\]

(4)

where \(k\) is the damping constant between 0 and 1.

Verlet integration ‘with’ damping is a bit more complex and expensive compared to the basic Verlet scheme shown in Listing 3; however, it’s a necessary evil if we want our system to remain stable and converge. For example, the Verlet damped integration implementation is shown below in Listing 3:

```plaintext
1 Vector3 temp = pos;
2 pos += (pos - old_pos) * (1.0f - DAMPING) + acceleration * dt * dt;
3 old_pos = temp;
```

**4. Cloth Interconnection Structure**

**Constraint-Pairs**  The cloth structure is formed by creating an array of constraints for each particle. Each constraint points to ‘two’ particles (i.e., a distance constraint). We can use as many distant constraints as are necessary to create the necessary stiff cloth effect. We iteratively update each constraint individually. As you can imagine, as we update and correct one constraint, it affects the adjacent connected particles and their constraints. However, we find that, after a finite number of iterations, the system of interconnected particles will eventually ‘converge’ on a result with all the constraints in a valid state. We can limit the number of iterations to ensure we maintain a real-time frame-rate.

**Hooke’s Law**  For traditional interconnected set of springs we would use the popular Hooke’s law - where we calculate the force for each constraint and apply it to each particle. We store the rest length for the distance between each particle at the start. The error between the current and stored rest length is used to calculate the correcting force. However, for our velocity Verlet integration scheme, we can simply snap the positions into place. (i.e., see Listing 1)

**Snap-To-Constraints**  Verlet constraints are simple. We adjust the ‘current’ position each frame. We work out the distance error and update the constraint so that each particle is
pushed back into place. As we update each constraint, it will invalidate adjacently connected particle constraints. However, if we iteratively keep updating the constraints, after a finite time the system as a whole will converge on a result.

**Listing 1.** As shown in Figure 3, we need to push each constraint back into place so that the constraint is valid.

```plaintext
1 err_length = cur_length − rest_length
2 err_direction = Length(p1 − p0)
3 p0 += err_direction * err_length * 0.5
4 p1 += err_direction * err_length * 0.5
```

where \( p0 \) and \( p1 \) are the particle positions for the constraint, the rest length is the distance between the two particles initially (i.e., \( \text{rest_length} = \text{Length}(p1-p0) \)).

**Topology - Structure, Shear, and Bending** We need to add additional interconnected constraints to create a cloth effect that looks like cloth (i.e., not like a set of rigid hinges). Hence, as well as the triangle edge constraints, we include shearing and bending constraints, see Figure 4 below:

**Wind** Without wind the cloth effect is rather simple and doesn’t really catch the viewers eyes. We can add in an additional wavy flag type effect by adding a force based on the triangles normal.

**Acknowledgements**
We would like to thank all the reviewers for taking time out of their busy schedules to provide valuable and constructive feedback to make this article more concise, informative, and correct. However, we would be pleased to hear your views on the following:

- Is the article clear to follow?
- Are the examples and tasks achievable?
- Do you understand the objects?
- Did we missed anything?
- Any surprises?

The lessons provide a basic introduction for getting started with cloth effects. So if you can provide any advice, tips, or hints during from your own exploration of cloth simulation development, that you think would be indispensable for a student’s learning and understanding, please don’t hesitate to contact us so that we can make amendments and incorporate them into future tutorials.

**Recommended Reading**

www.napier.ac.uk/games