# PRACTICAL AND REALISTIC ANIMATION OF CLOTH

*Serkan Bayraktar, Uğur Güdükbay, and Bülent Özgüç*

Bilkent University
Dept. of Computer Engineering
Bilkent 06800 Ankara, Turkey

## ABSTRACT

In this paper, we propose a system for the practical animation of cloth materials. A mass spring based cloth model is used. Explicit time integration methods are used to solve the equations of motion. We update the spring constants dynamically according to the net force acting on them. In this way, spring constants do not grow arbitrarily to introduce numerical instability and realistic cloth appearance without over elongation is obtained.

***Index Terms*—** Mass-spring, collision handling, cloth simulation, physically-based, over-elasticity.

## 1. INTRODUCTION

Techniques developed to model cloth can be categorized as *geometric*, *physically-based*, and *hybrid*. Geometric methods do not consider physical properties of cloth. They emphasize appearance represented as geometric equations, requiring more user intervention than other methods [1]. Physically-based techniques offer more reality and ease of modeling. In these methods cloth is represented as triangular or rectangular grids composed of a finite number of mass points whose equations of motion are solved based on the forces acting on the particles [2, 3, 4]. Breen et al. [5] used particles and energy minimization to model the draping behavior of cloth. The hybrid methods can be considered as a combination of physical and geometrical methods [6].

In this paper, we present our work on cloth simulation by using mass-spring networks. An algorithm based on polygon-to-polygon collision detection is used for detecting collisions of cloth with rigid objects. Bounding volume hierarchies are used to speed up the collision detection process. The simulation algorithm is based on the Newtonian laws of dynamics. An explicit integration method, namely Fourth order Runge-Kutta with adaptive time stepping, is used to solve the equations of motion. Over elasticity is one of the drawbacks of mass-spring models when clothlike objects are simulated. The reason is that real cloth is much more stiff to stretching and shearing than mass-spring models can robustly simulate. The result is a grid that looks like rubber rather than cloth. The proposed solution is to update the spring constants dynamically according to the net force acting on them. This method guarantees robustness since spring constants do not grow to introduce numerical instability and realistic cloth appearance is obtained.

This work is supported by European Union 6th Framework Program under Grant No. FP6-511568 (3DTV NoE Project) and The Scientific and Technical Research Council of Turkey (TÜBİTAK) under grant no EEEAG-105E065.

## 2. MASS-SPRING MODEL

A mass-spring network consists of mass points connected by massless damped springs. By this model, it is assumed that the mass of the body is concentrated at specific points rather than it is scattered along the body. One of the constraints on the realism of the model is the density of the mass points. External and internal forces act on the mass points. Internal forces are spring forces that mass points exert on each other through damped springs, and external forces are environmental forces such as gravity, viscous drag, impulse based forces, and user defined forces such as mouse drag. The mesh is simulated through time by calculating the positions of the mass points at each time step.

Figure 1 shows three kinds of springs used in constructing the mesh. Structural springs connect masses [i, j] to [i + 1, j] and [i, j] to [i, j + 1], shear (diagonal) springs connect mass points [i, j] to [i + 1, j + 1] and [i + 1, j] to [i, j + 1], and finally flexion (bending) springs connect masses [i, j] to [i + 2, j] and [i, j] to [i, j + 2]. Structural springs are constrained by the stretching and compression forces. Diagonal springs are constrained by the shear stresses, whereas bending springs are used to limit bending of the structure, since they are constrained by the flexion stresses. Setting spring constants of these spring types independently enables us to mimic different types cloth and rubbery objects.

We use the Newtonian second law of dynamics to determine position of a mass point $m_{ij}$ at a particular time:

$$F_{ij} = \mu_{ij}\, a_{ij}, \tag{1}$$

where $\mu_{ij}$ is the mass of the point, $a_{ij}$ is the acceleration caused by the net force $F_{ij}$ acting on the mass point. The net

1-4244-0722-2/07/$20.00 ©2007 IEEE

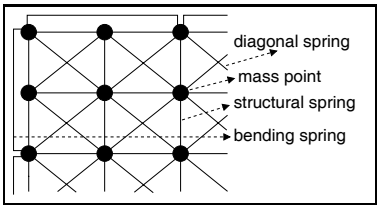



**Fig. 1**. A sample mass-spring mesh

force is calculated by considering the internal forces, namely the forces mass points exert on each other through springs, and external forces such as gravity, viscous air drag, wind etc. The internal force on two mass points $m_{ij}$ and $m_{kl}$ caused by the damped spring connecting them can be calculated by the Hooke's Law:

$$F_{ij}^{spring} = \left(-k_s\left(l_{ij,kl} - l_{ij,kl}^0\right) + k_d \frac{l_{ij,kl} \times l'_{ij,kl}}{|l_{ij,kl}|}\right) \frac{l_{ij,kl}}{|l_{ij,kl}|}$$
$$F_{kl}^{spring} = -F_{ij}^{spring}, \tag{2}$$

where:

- $k_s$ is the spring constant, and $k_d$ is the damping constant,
- $l_{ij,kl} = \overrightarrow{m_{ij}\, m_{kl}}$ and $l_{ij,kl}^0$ is the spring's rest length,
- $l'_{ij,kl}$ is the relative velocity of $m_{ij}$ with respect to $m_{kl}$.

Viscous air drag is another external force to act on all the mass points. It has the effect of dissipation of kinetic energy of mass points. This force can be calculated by using:

$$F_{ij}^{air} = -C_{dis}\, v_{ij}, \tag{3}$$

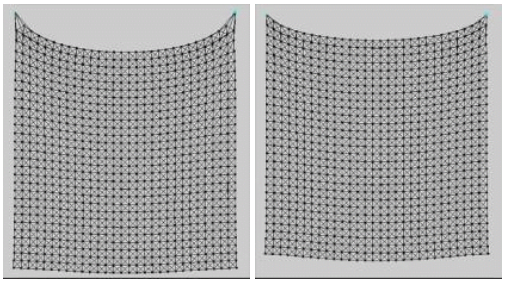
where $C_{dis}$ is the coefficient of air drag and $v_{ij}$ is the velocity vector of the mass point. Adding moderate amount of viscous drag introduces stability to the system, but much of this force gives unrealistically oily look.

Moving air (or fluid) also exerts a force on the mass points which can be found by employing equation 4:

$$F_{ij}^{wind} = C_{wind}[n_{ij} \,.\, (v_{air} - v_{ij})]n_{ij}, \tag{4}$$

where $C_{wind}$ is a coefficient to express the amplitude of wind, $v_{air}$ is the velocity vector of wind, $v_{ij}$ is the velocity vector of the mass point, and $n_{ij}$ is the normal vector to the surface of the model at the mass point $m_{ij}$.

We used an explicit integration method to solve the equations of motion. In order to compute the position and velocity of a mass point at time $t+\Delta t$, where $\Delta t$ is a chosen time step, we used the fourth-order Runge-Kutta method, which, most of the time, gives much more accurate results with larger time steps. Each step of Runge-Kutta involves four function evaluations. In our case each function evaluation involves finding net forces acting on each mass point, updating bounding volumes, finding surface normal vectors, checking for collisions and self-collisions, and resolving any existing collisions.



(a) (b)

**Fig. 2**. Cloth (a) without and (b) with elongation limit.

Efficiency of the numerical solution can be improved by employing adaptive step size control. A numerical solver with adaptive step size control tries to achieve some predetermined accuracy by using as large time steps as possible. To use larger time steps in the integration, an implicit time integration method can be also used [7].

Cloth is much stiffer than a damped spring and it should not behave like a rubber. To prevent over-elongation, we set an elongation limit between rest length and elongation up to the rest length of the spring (see Figure 2). At each step, we measure the ratio of the distance between the mass points to the rest length. In this way, we measure how far we are from the predetermined elasticity of the spring. Then, we multiply the spring coefficient by the ratio of the distance to the allowed elongation to find the new coefficient. The spring coefficient is updated according to the following equation:

$$k_{new} = \begin{cases} k_{original}\frac{d_{current}}{d_{rest}} & \text{if } d_{current} \geq d_{allowed} \\ k_{original} & \text{else} \end{cases} \tag{5}$$

where $k_{new}$ and $k_{original}$ are the new and original spring constants, respectively, $d_{current}$ is the length of the spring at the current time step, and $d_{rest}$ is the rest length of the spring. $d_{allowed}$ is the maximum spring length that is allowed without modifying the spring coefficient.

## 3. COLLISION HANDLING

The most expensive stage of a physically-based simulation is detecting collisions between objects. Collision detection should be accurate, fast, stable, and robust for a simulation to be realistic. We have to check if a part of the cloth collides with another object in the scene and/or with itself. Bounding volumes have been widely used to speed up the colli-

sion detection. The main idea is to encapsulate objects by bounding volumes whose collision detection is much easier. A hierarchy of bounding volumes (usually spheres or boxes) is utilized to make collision-detection faster. The hierarchy is usually designed as a tree where parent nodes encapsulate their children. In the leaf nodes, polygons of the objects to be checked for a possible collision reside.

Two methods can be employed to detect collisions between two polygons. One method is to check if any vertex of the cloth polygon intersects with an object polygon. The other method is to utilize an algorithm that detects intersection of two triangular polygons. The former method is easy to implement and faster in terms of detecting collisions. The latter, however, has several advantages in terms of resolving intersections. It is fast to find mass-point to object polygon intersections, but this does not guarantee coverage of all possible collisions between a cloth polygon and an object polygon. A collision detection algorithm using mass-point to polygon test may fail and object polygon penetrates unrealistically into the cloth. Although computationally more costly, detecting polygon-to-polygon intersection does not have the mentioned drawbacks. The algorithm we used to find out triangle intersections works moderately fast and it includes minimum number of divisions [8].

To resolve collisions, we mark the cloth polygons intersecting with other polygons at each time step. The identifier of the intersecting polygons and the number of them are also stored. At each iteration, we go through the cloth polygons looking for intersecting polygons. During resolution process, we modify velocity and position of the cloth polygon so that any existing intersection is resolved and possible ones are avoided. To modify velocity and position, we first calculate the average of the normal vectors of the object polygons that the cloth polygon intersects. Then we find the components of the vertex velocities in the direction of this average normal and eliminate them, thus avoiding the cloth polygon going deeper into the object polygons. To resolve the intersection, we move the cloth polygon vertices along the average normal as long as the polygons intersect.

During collision resolution, we also simulate friction between the rigid object and the cloth. To achieve this, we find the components of the net forces acting on mass-points perpendicular to the average normal. Multiplying these forces with a friction coefficient provides a friction force.

Self-collision is the case where both of intersecting polygons belong to the same deformable model. Several techniques are proposed to speed up the self-collision detection, such as [9, 10]. Bounding box hierarchies are commonly used to improve the self-collision detection [7]. In our implementation, we also use a bounding volume hierarchy for this purpose. Two cloth polygons are tested for an intersection only if their bounding volumes intersect. Adjacent cloth polygons are not tested for a collision and our data structure can easily reveal the polygon adjacency information.

Collision response in self-collisions should be handled more precisely than collisions with rigid objects since we cannot make the simplifying assumption that one of the polygons is motionless. The method we used to resolve the collision eliminates the force and velocity components that are perpendicular to the collision plane. Although this method gives moderately satisfying results in cases like simulating garments on fixed bodies, it is unrealistic if we consider dynamic movements of a cloth mesh. We should enforce the conservation of momentum to achieve a higher level of reality. Moreover, collision resolution should not create new collisions. Huh et al. [11] develop an algorithm that considers self-collision as a special case of N-body collisions. They group cloth particles into parts and handle collisions between these parts while conserving momentum.

## 4. IMPLEMENTATION

Each mass point is positioned such that the distance between mass points is equal to the predetermined rest length of the mesh. Flexion (bending) springs are inserted between every other mass points and diagonal springs are inserted along the diagonals of the rectangles created by the mass points. 2D texture coordinates are found for each mass point and used in texture mapping. After creating the cloth mesh, we construct the bounding volume hierarchies. Bounding volume hierarchies are stored in binary trees where each bounding sphere residing in a leaf node encapsulates a cloth polygon. The radius of these leaf node bounding spheres are set to the longest edge of the triangular polygon; the center of the sphere is the midpoint of that edge (Figure 3 (a)).

The nonleaf nodes are created in bottom-up fashion where the radius of a higher level sphere is calculated by finding the length of the line connecting the centers of the child spheres (Figure 3 (b)). The higher level spheres store the identifier of the child spheres. Bounding boxes that reside in the leaf node of the tree are found by calculating maximum and minimum coordinates of cloth polygons in three axes. The identifier of each cloth polygon is stored in the corresponding bounding box node. Higher level bounding boxes are created by finding the maximum and minimum coordinates of two child bounding boxes.

At each time step, we update normal vectors of cloth polygons and bounding volumes of the cloth mesh. When updating bounding volumes, we only update the positions and dimensions of the bounding volumes and do not modify the structure of binary trees used to store bounding volume hierarchies. Then, we call the numerical solver. The fourth order Runge-Kutta method updates the velocity and positions of the mass points four times at each time step, thus it needs force calculation to be done four times. Each step of the Runge-Kutta begins with finding net forces acting on mass points. The numerical solver updates the velocity and position of mass points by using these forces. At the end of the

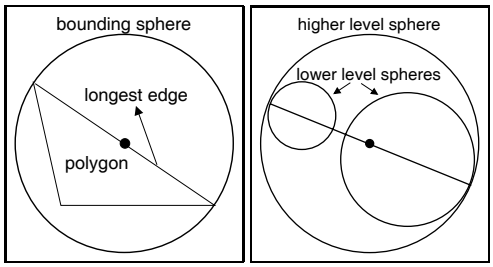



**Fig. 3**. Bounding sphere hierarchies: finding center and radius of (a) leaf-level spheres; and (b) higher-level spheres.

step, collision detection is done to see if any collisions occurred due to position change. Finally, we redraw the scene using the updated positions.

## 5. RESULTS AND CONCLUSIONS

Sample animations produced by our implementation can be found in `http://www.cs.bilkent.edu.tr/~gudukbay/cloth.html`. The simulations are performed on a PC with 1.5 GHz Intel Pentium 4 processor, 256 MB RDRAM, and GeForce2 display adapter.

In the first simulation, a cloth mesh with size 24x30 is constrained in two points and hung while there is gravitation and wind blowing toward the camera point (Figure 4 (a)). The average frames per second (fps) is 27, which is enough for a real time simulation. The cloth is textured and rendered. The cloth mesh consists of 720 mass points, 4,052 springs (1,386 structural, 1,334 diagonal, and 1,332 bending springs), and 1,334 cloth polygons.

In the second simulation, a cloth grid with size 26x26 is draped onto a table (Figure 4 (b)). The cloth grid has reached its equilibrium shape in 24 seconds and in 450 iterations (in average 18.75 fps). The cloth mesh consists of 676 mass points, 3,798 springs (1,300 structural, 1,250 diagonal, and 1,248 bending springs), and 1,250 cloth polygons. There is no wind in the environment and a small degree of friction is applied between the table surface and the cloth. The scene is composed of 51 object polygons constructing the floor and the table. Table 1 shows the processing times of this simulation.

| mesh size | run time (seconds) | frames per second (fps) |
|---|---|---|
| 26x26 | 24 | 18.75 |
| 36x36 | 45 | 10 |
| 48x48 | 86 | 5.2 |

**Table 1**. Processing times of draping cloth with different cloth sizes (450 iterations)

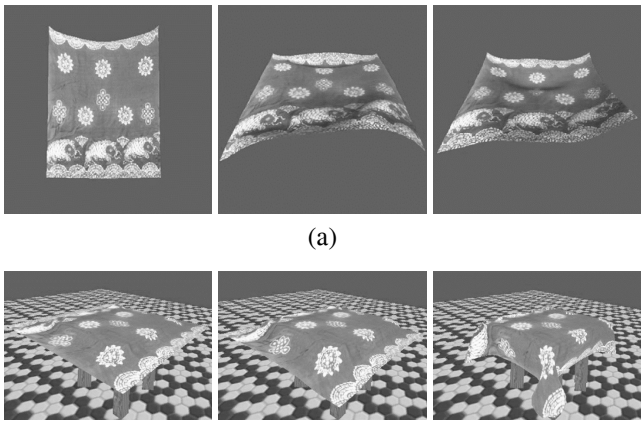



**Fig. 4**. The still frames from the simulation of a hanging cloth (a) and a table cloth draping onto a table (b).

## 6. REFERENCES

[1] J. Weil, "The synthesis of cloth objects," *Proc. ACM SIGGRAPH'86*, pp. 49–54, 1986.

[2] D. Terzopoulos, J. Platt, A. H. Barr, and K. Fleischer, "Elastically deformable models," *Proc. ACM SIGGRAPH'87*, pp. 205–214, 1987.

[3] P. Volino, M. Courchesne, and N. M. Thalmann, "Versatile and efficient techniques for simulating cloth and other deformable objects," *Proc. ACM SIGGRAPH'94*, pp. 137–144, 1994.

[4] X. Provot, "Deformation constraints in a mass-spring model to describe rigid cloth behavior," *Proc. Graphics Interface*, pp. 147–154, 1995.

[5] D. Breen, D. House, and P. Getto, "A physically-based particle model of woven cloth," *the Visual Computer*, vol. 8, pp. 264–277, 1992.

[6] T. Kunii and H. Gotoda, "Singularity theoretical modeling and animation of garment wrinkle formation process," *Visual Computer*, vol. 6, pp. 326–336, 1990.

[7] D. Baraff and A. Witkin, "Large steps in cloth simulation," *Proc. ACM SIGGRAPH'98)*, 1998.

[8] T. Moller, "A fast triangle-triangle intersection test," *Journal of Graphics Tools*, vol. 2, pp. 25–30, 1997.

[9] P. Volino and N. M. Thalmann, "Efficient self-collision detection on smoothly discretized surface animations using geometrical shape regularity." *Computer Graphics Forum*, vol. 13, pp. 155–166, 1994.

[10] X. Provot, "Collision and self-collision handling in cloth model dedicated to design garments," *Proc. Graphics Interface*, pp. 177–189, 1999.

[11] S. Huh, D. Metaxas, and N. Badler, "Collision resolutions in cloth simulation," *Proc. IEEE Computer Animation*, 2001.