Chapter 9
Collision Detection in Cloth Modeling

R. Bigliani
Visiting Scholar
Department of Mechanical and Aerospace Engineering
NC State University
Raleigh, NC 27695
(919) 515-5263
rbiglian@gateway.net

J. W. Eischen
Associate Professor
Department of Mechanical and Aerospace Engineering
NC State University
Raleigh, NC 27695
(919) 515-5263
eischen@eos.ncsu.edu

Cloth Modeling and Animation
A.K. Peters Publishing

1 Introduction

Realistic modeling of cloth draping and fabric manipulation requires complex algorithmic implementations. The range of possible surface topologies that must be handled is very wide. Phenomena such as folding, wrinkling and crumpling has to be anticipated. Also, cloth interaction with rigid surfaces as well as the self-collision problem must be treated.
The objective of this chapter is to present an easily programmable technique for solving the collision problem when working with a dynamic particle model of cloth. The idea is to develop a general solution applicable both to fabric/external object interaction and to cloth/cloth collision. Particular attention is given to difficult cases, which involve multiple contact with several layers of the fabric or the sliding of two surfaces permanently in contact during the motion. The following issues have received prominent attention while developing the collision detection ideas:

- The dynamic particle method is physically based and it must provide a realistic response to the highly non-linear problem of cloth drape and manipulation. The results obtained must agree with experiments and/or theoretical results. The model must demonstrate stability with a wide range of possible parameters; considering both material properties and external loads. The collision algorithm must not interfere with the physical behavior of the network by creating local instabilities or constraining the natural evolution of the network.

- The base geometric element for the collision detection algorithm is the triangle. Every surface involved in the simulation (deforming objects or fixed objects of the scenery) can be conveniently triangularized by well-known methods. The collision method provides the same kind of response for handling all the collisions with the different parts of the environment.

- Simulation time can be dramatically effected by introduction of collision detection. The search method that detects colliding parts of the deforming structure must be performed at every simulation step. A systematic method that checks every possible colliding configuration requires a considerable amount of time.

- The collision detection algorithm has to be implemented with the possibility of dissipative properties, including viscous effects or Coulomb friction.

2 Overview

Several methods have been applied in computer animation in order to simulate and to control the motion of rigid and flexible objects. Dynamic and static equations are discretized and integrated to calculate the motion of surfaces of rigid bodies and networks of nodes for flexible objects. The efforts are generally addressed to

solve problems of simulation time and realistic motion achievement. The collision detection problem and the related issue of the response to the collision have been the object of several research studies. Moore and Wilhelms [1] propose two different methods for collision detection. One algorithm is based on surface triangularization and it can be applied to rigid and flexible surfaces. The second method is based on objects modeled as rigid polyhedra and it offers a robust and fast solution to the collision problem. A different approach is followed by Wilson and Larsen [2]. They propose a possible algorithm that runs queries on collision detection and object distance computation, based on an overlap graph technique. They use this solution as an alternative to the classic method of bounding volume hierarchies to accelerate the queries. This method, in fact, can only handle relatively small models. Krishnan and Gopi [3] present a method to determine collisions between spline models. They apply a bounding box solution in order to compute the areas of collision and to reduce the simulation time and the memory requirements. A hierarchical representation of general polygonal models using oriented bounding box trees is the technique applied by Gottschalk and Lin [4]. The trees are scanned at run-time, testing for possible overlaps between bounding boxes. A method to report geometric contacts between objects in a large-scaled virtual environment is implemented by Cohen, Lin and Manocha [5]. The algorithms use spatial and temporal coherence between successive instance in order to compute the collisions. The method described in the following sections proposes a solution where the surfaces are triangularized in order to detect the collisions. A spatial enumeration method is adopted to reduce the number of necessary tests needed for collision detection. A completely inelastic impact is simulated in those areas where the contact between surfaces is detected and a hashing list is implemented to reduce memory requirements.

3 Collision Detection Algorithm

The collision algorithm proposed here has been studied and implemented on a dynamic particle model. The particle network is a system of square elements where each vertex is a particle or node. The internal strains considered are: in-plane stretch/compression, in-plane shear, and out-of-plane bending. Gravity, specified forces, and specified displacements can be applied as input to the nodes of the network for simulating different conditions of the cloth draping or manipulation processes. The elastic modulus, thickness, and weight per unit surface area of the cloth are input as well. It is possible to use different parameters for the warp and the weft direction of the fabric.
A direct numerical integration of Newton's second law with Runge-Kutta is employed to compute the evolution of the network. This method acts directly on position and velocity. The network is conveniently triangularized at the beginning of the simulation. The normal vector for each triangle is of paramount importance for the collision algorithm.

3.1 Geometry - collision detection between elements

The fundamental geometric entity for our network is the element triangle. The simulation algorithm operates directly on the nodes, providing position and velocity for each of them at every simulation step. The simplest idea, in terms of minimizing algorithm complexity and reducing simulation time involves checking possible collisions between node-triangle pairs. When a collision is detected, position and velocity corrections are performed for the colliding node. A key issue is to distinguish between the inside and outside of the surface and the relative position of the colliding node for computing the collision response. For dealing with complex situations like folding and multi-layer overlapping, we cannot simply use an initial orientation of the fabric which defines an inside/outside face. We must update the orientation of the fabric while the simulation is running. The idea is that it is not necessary to know which is the inside/outside orientation of all the network elements at every simulation step. For our purpose, it has proved sufficient to know only locally if a particular node is crossing a fabric surface. We have found a solution for determining if a node is penetrating the surface of a triangle and thus compromising the local consistency of the network. This solution is based on:

- The study of the trajectory of the node considering the current consistent position of the node and the future position predicted by the numerical integration,
- The current position of two other nodes associated with a triangle element in relation to the colliding node,
- The components of the normal vector on each triangle being updated every step.

Fig. 1 shows the relative positions of two elements of the network at step k. This configuration is assumed consistent, i.e. there is no intersection between the two elements at this time. Consider now the configuration shown in Fig. 2 where node D has assumed a position at step $k+1$ that has compromised in some way the configuration existing at step k and has produced an intersection. We are considering the triangle ABC as the colliding surface and node D as the colliding particle. The vectors $\underline{X}_F$ and $\underline{X}_D$ are defined by the directions that join the position of nodes F at step k and D at step $k+1$ with vertex A at step k of the triangle ABC.

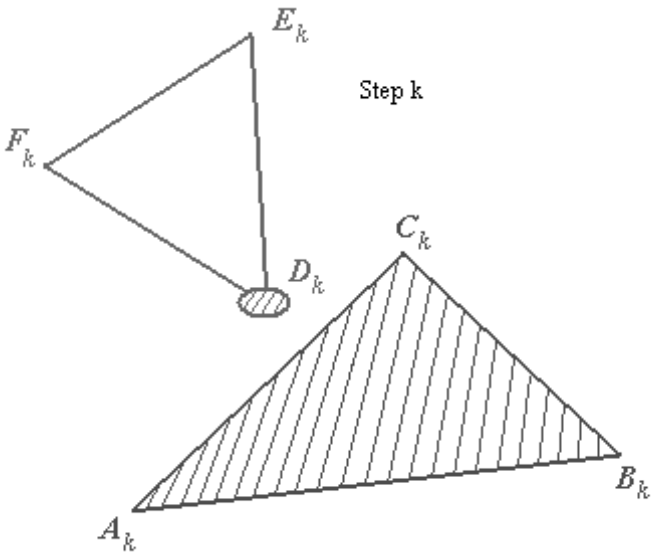

Figure 1: Configuration of two triangles of the network at step k

$$\begin{aligned} \underline{X}_F &= \underline{F}_k - \underline{A}_k, \\ \underline{X}_D &= \underline{D}_{k+1} - \underline{A}_k. \end{aligned} \tag{1}$$

The components $\underline{X}_F^n$ and $\underline{X}_D^n$ of the vectors $\underline{X}_F$ and $\underline{X}_D$ are parallel to the normal vector $\underline{n}$ of the triangle ABC.

$$\begin{aligned} \underline{X}_F^n &= (\underline{X}_F \cdot \underline{n})\underline{n} \\ \underline{X}_D^n &= (\underline{X}_D \cdot \underline{n})\underline{n} \end{aligned} \tag{2}$$

If $\underline{X}_F \cdot \underline{n}$ and $\underline{X}_D \cdot \underline{n}$ are of opposite sign, then node D has crossed the plane during its path between step k and $k+1$, or at least the edge DF of the triangle DEF intersects the plane of the triangle ABC. In fact if the collision involves two edges of different triangles, the configuration shown in Fig. 2 is still verified. In this case, the node D does not intersect the triangle ABC during its path between the step k and the step k+1, but the edge DF still intersects the triangle ABC with an edge-to-edge collision configuration.

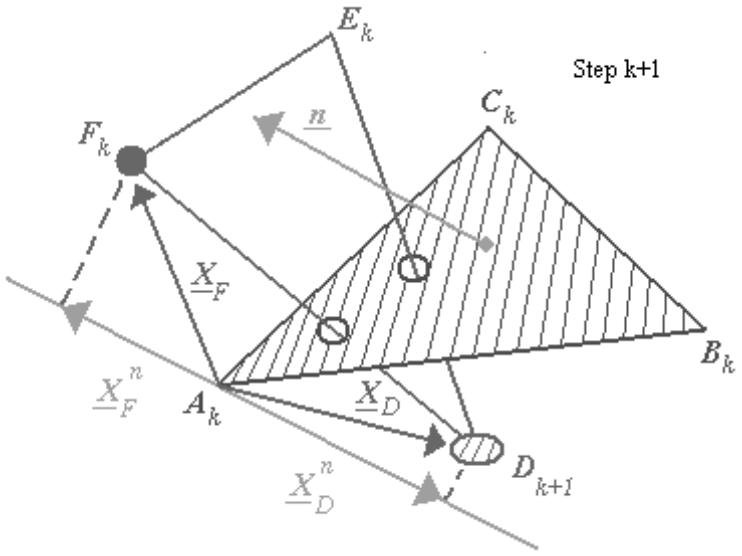

Figure 2: First test for collision detection

The algorithm then computes the intersection between the segment DF, as defined in Fig. 2 and the plane defined by the triangle ABC, as shown in Fig. 3. The equation of the plane of the triangle (Σ) can be written, as

$$(\underline{P} - \underline{A}) \cdot \underline{n} = 0, \tag{3}$$

where $\underline{P} = x\underline{i} + y\underline{j} + z\underline{k}$ is a vector pointing to a general point in the plane, $\underline{A} = x_A\underline{i} + y_A\underline{j} + z_A\underline{k}$ is the vector pointing to the vertex A, and $\underline{n} = a\underline{i} + b\underline{j} + c\underline{k}$ is the normal to the plane. Thus, the equation of the plane is

$$a(x - x_A) + b(y - y_A) + c(z - z_A) = 0 \tag{4}$$

Using the coordinates of node F at step k and the coordinates of node D at step $k + 1$, it is easy to define the equation of the line r as a function of a parameter t. This line describes the new direction of one of the edges of triangle DEF once the updated position of node D is known. The components of the vector $\underline{r} = l\underline{i} + m\underline{j} + n\underline{k}$ along the line r are

$$l = x_D - x_F$$

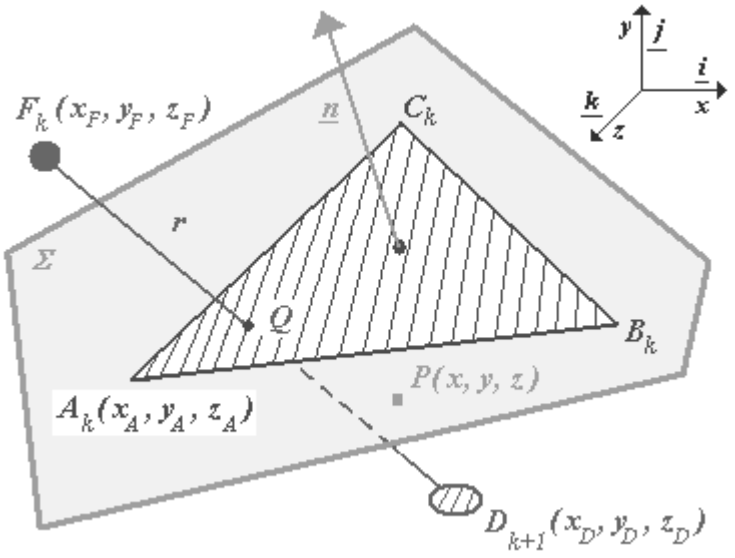

Figure 3: Intersection between the line r and the plane Σ

$$\begin{aligned} m &= y_D - y_F \\ n &= z_D - z_F \end{aligned} \tag{5}$$

Then the parametric equations for coordinates of points on the line r are

$$\begin{aligned} x &= lt + x_F \\ y &= mt + y_F \\ z &= nt + z_F \end{aligned} \tag{6}$$

The parameter t varies between 0 and 1 while a generic point is moving along the line r from node F to D. By simple substitution of the equation of the line into the equation of the plane, we can find the value of t_Q corresponding to the intersection point Q between the plane (Σ) and the line r

$$t_Q = -\frac{p}{s}, \tag{7}$$

where $p = \underline{n} \cdot \underline{F} - \underline{n} \cdot (\underline{A} - \underline{0})$ and $s = al + bm + cn$. By using the parametric equation of the line r it is now possible to obtain the coordinates of the intersection Q,

$$\begin{aligned} x_Q &= lt_Q + x_F \\ y_Q &= mt_Q + y_F \\ z_Q &= nt_Q + z_F \end{aligned} \tag{8}$$

The next step allows us to understand if the intersection between the edge FD and the plane of the triangle falls inside the surface outlined by the perimeter of the triangle ABC. The first test which has to be conducted is to verify if the intersection Q is coincident, within the precision of the machine, with one of the vertices of the triangle, as shown in Fig. 4. This is necessary because otherwise the following steps of this approach do not guarantee a correct response. If the intersection Q is coincident with one of the vertices of the triangle, then the collision algorithm stops the consistency tests considering this vertex. Otherwise the algorithm proceeds.

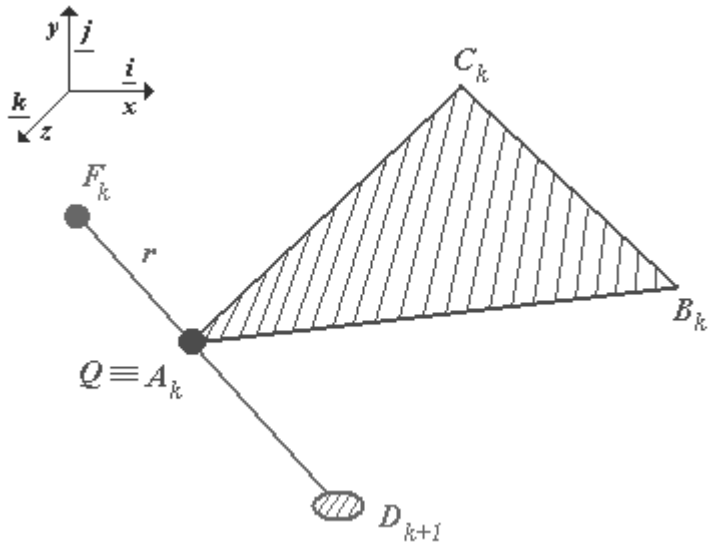

Figure 4: Case where intersection Q is coincident with vertex A of the triangle

At this point the problem is not three-dimensional anymore but the final analysis is made in two dimensions on the plane defined by the vertex of the triangle and the intersection point Q. The outcome of this analysis determines if a correction of the position and the velocity of the node is necessary because a collision is occurring. The inside/outside test shown in Fig. 5 is performed by means of the following steps:

- The triangle is divided into three triangles, defined by joining the intersection point Q with the vertices A, B and C. The angles α, β, and γ surrounding point Q are computed.

- If one of these angles is equal to π (within a certain tolerance $\pm\epsilon$), then the intersection Q must fall on one of the edges of the triangle. That edge is between the two vertices that include the angle π (AB for α, BC for β, CA for γ). If this condition exists, the following step is not necessary and a collision between the node and the triangle is occurring.

- $\theta_{total} = \alpha + \beta + \gamma$ is computed. If θ_{total} is equal to 2π within a certain tolerance ($\pm\epsilon$), then the intersection falls inside the area of the triangle. Otherwise it is falling on the plane of the triangle but outside the area defined by the perimeter.

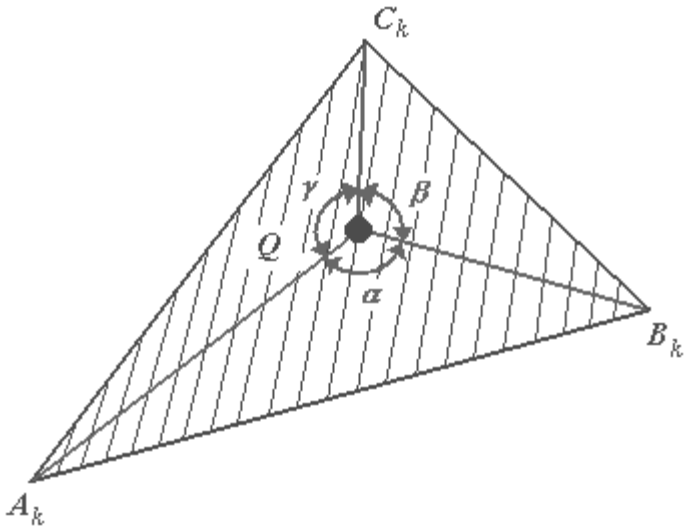

Figure 5: Inside/outside analysis

The angle α, as well as β and γ, can be calculated using the formula:

$$\alpha = \arccos \frac{\underline{v}_1 \cdot \underline{v}_2}{\|\underline{v}_1\| \|\underline{v}_2\|} \tag{9}$$

where $\underline{v}_1 = \underline{A} - \underline{Q}$ and $\underline{v}_2 = \underline{B} - \underline{Q}$.
The angles α, β and γ, are calculated using the *arccos* function that always returns a value between 0 and π. The geometric collision between a node and a triangle has

been tested by simulating the case of a free-falling (gravity load) single node free to move inside an icosahedron. In this case the segment used for checking the collision with a possible triangle was the segment that joined the position of the node at the current step k and the position of the node at the following step $k+1$, provided by the Runge-Kutta routine. Through these numerical experiments (see Fig. 6) we have found that an effective value for the tolerance ϵ is:

$$\epsilon = \frac{2\pi}{100} \tag{10}$$

A possible alternative solution may be applied to verify if the intersection Q falls inside the perimeter of the triangle, avoiding trigonometric functions. The intersection Q needs to be translated to the origin of an XY coordinates system. If the positive X axis intersects the triangle's edges 0 or 2 times, then Q falls outside the triangle, otherwise Q is inside. If the Y coordinate changes the sign along an edge, that edge intersects the X axis.

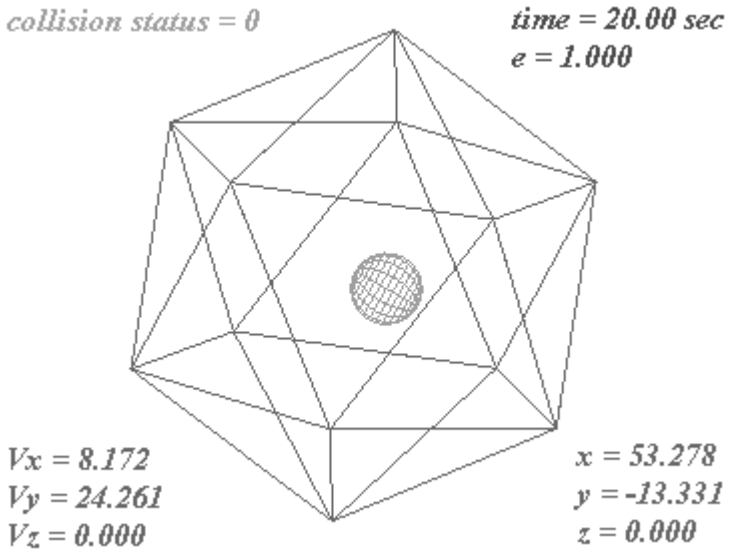

Figure 6: Icosahedron Test Surface

This geometric approach for checking collisions between one node and a triangle of the network provides information about the local consistency of the network but

also information about the modalities of collision. If a collision is occurring, the algorithm can evaluate the possible configuration of this collision: node-to-node, node-to-edge, node-to-triangle. If the line, considered for computing the collision, is the segment FD, the algorithm can even check the edge-to-edge and the triangle-to-triangle type of collision. In case of an edge-to-edge collision, the node D does not intersect the plane of the triangle during its path between the step k and the step $k+1$. However, the segment FD, which is the possible colliding edge, still intersects the plane of the triangle and the intersection Q can be calculated, detecting the collision.

The basic idea of the collision algorithm is that, for every simulation step, our network moves from a consistent configuration, to a new one where possible collisions and intersections can be verified. So we have to provide a correction of the position of those nodes so as not to compromise the consistency of the network.

3.2 Collision Dynamics - post collision position and velocity updates

When a collision between a node and a triangle is detected, a correction has to be applied to the position and velocity of the node involved in the collision. The correction is made according to the momentum conservation law, consistent with a perfectly inelastic collision. We make the following initial assumptions, which allow simplification of the collision response calculations:

- The collision is supposed central and oblique. The collision line passes through the centers of mass of the two bodies that are colliding but the velocities are not necessarily directed along the collision line. The collision line is considered to be the normal line to the triangle surface at the point of intersection with the colliding node.

- If we consider that a single triangle is connected to the rest of the network, then its mass M is much greater then the mass m of the colliding node ($M \gg m$);

- The intersection point Q between the line r and the triangle is considered as the collision point between the node D and the triangle. As shown in Fig. 7, it would be more correct to use the intersection Z between the path of the node D and the surface of the triangle. Since the collision tests are performed every simulation step, the node D can penetrate only very weakly into the surface of the triangle before the collision is detected and the correction in position and velocity is provided. Thus, the approximation of the intersection Z by the point Q will introduce only small errors.

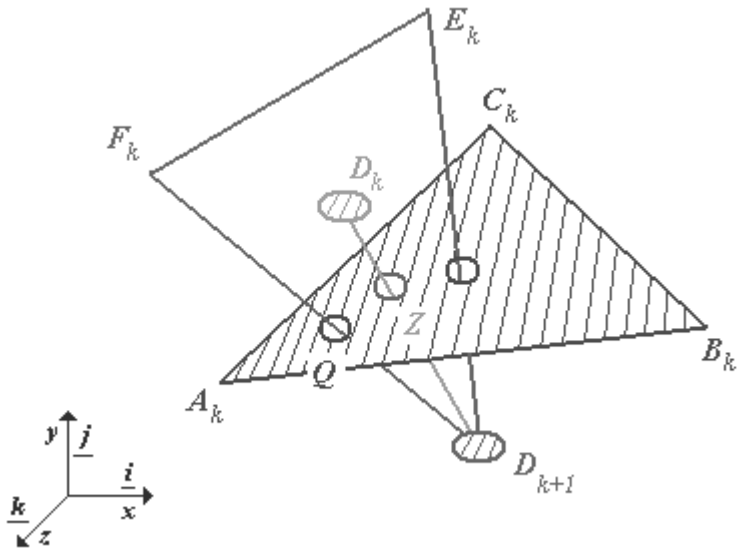

Figure 7: Intersections Q and Z and the trajectory of node D

The first hypothesis insures that the interaction between the node and triangle can only alter the component of momentum along the collision line, normal to the surface. The second hypothesis ($M \gg m$) forces the triangle to keep its velocity after the collision. The relations between the velocities of the node D and the collision point Q before ($^-$) and after ($^+$) the impact are expressed by

$$\begin{aligned} \underline{v}^+_{Dk'} &= -e\underline{v}^-_{Dk'} + (1+e)\underline{v}^-_{Qk'}, \\ \underline{v}^+_{Qk'} &= \underline{v}^-_{Qk'}, \end{aligned} \tag{11}$$

where $\underline{v}^-_{Dk'}$ and $\underline{v}^-_{Qk'}$ are the components of the velocities of D and Q along the direction k' normal to the triangle's surface, before the collision. $\underline{v}^+_{Dk'}$ and $\underline{v}^+_{Qk'}$ are the same components after the collision, and e is the restitution coefficient. Fig. 8 shows the relationships among $\underline{v}_D$, $\underline{v}_{Qk'}$, $\underline{v}_Q$ and $\underline{v}_{Qk'}$. If the collision is assumed to be completely inelastic, $e = 0$ and the colliding node assumes the pre-impact velocity of the triangle.

$$\underline{v}^+_{Dk'} = \underline{v}^-_{Qk'}$$

$$\underline{v}^{+}_{Qk'} = \underline{v}^{-}_{Qk'} \tag{12}$$

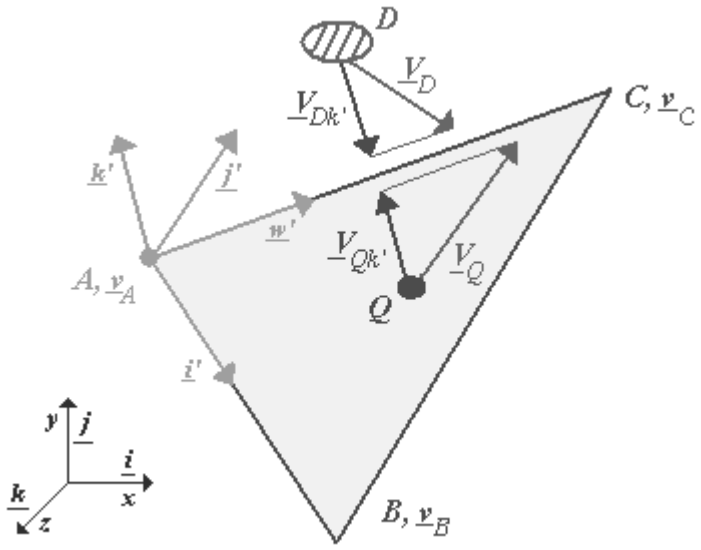

Figure 8: Components of the velocities along the direction normal to the triangle

During any simulation step the position and velocity of the vertices of the triangle ABC are known. For the collision algorithm we also need the velocity of a generic point Q that falls inside the area of the triangle. For this purpose, we define a mobile coordinate system $(\underline{i}', \underline{j}', \underline{k}')$ fixed to the surface of the triangle as shown in Fig. 8. Vector $\underline{k}'$ is chosen normal to the triangle, vector $\underline{i}'$ is parallel to $B - A$ and vector $\underline{j}' = \underline{k}' \times \underline{i}'$. The motion of the triangle is described by the $\underline{i}', \underline{j}', \underline{k}'$ vectors and their time derivatives

$$\begin{aligned}
\underline{i}' &= \frac{\underline{B} - \underline{A}}{\|\underline{B} - \underline{A}\|} \\
\underline{v}_{i'} &= \frac{d\underline{i}'}{dt} = \frac{\underline{v_B} - \underline{v_A}}{\|\underline{B} - \underline{A}\|} \\
\underline{w}' &= \frac{\underline{C} - \underline{A}}{\|\underline{C} - \underline{A}\|} \\
\underline{v}_{w'} &= \frac{d\underline{w}'}{dt} = \frac{\underline{v_C} - \underline{v_A}}{\|\underline{C} - \underline{A}\|}
\end{aligned}$$

$$
\begin{aligned}
\underline{k}' &= \frac{\underline{i}' \times \underline{w}'}{\|\underline{i}' \times \underline{w}'\|} \\
\underline{v}_{k'} &= \frac{d\underline{k}'}{dt} = \frac{\underline{v}_{i'} \times \underline{w}' + \underline{i}' \times \underline{v}_{w'}}{\|\underline{i}' \times \underline{w}'\|} \\
\underline{j}' &= \underline{k}' \times \underline{i}' \\
\underline{v}_{j'} &= \frac{d\underline{j}'}{dt} = (\underline{v}_{k'} \times \underline{i}' + \underline{k}' \times \underline{v}_{i'})
\end{aligned} \tag{13}
$$

The angular velocity $\underline{\Omega}$ of the triangle is given by

$$
\underline{\Omega} = (\underline{v}_{j'} \cdot \underline{k}')\underline{i}' + (\underline{v}_{k'} \cdot \underline{i}')\underline{j}' + (\underline{v}_{i'} \cdot \underline{j}')\underline{k}' \tag{14}
$$

and the velocity of the collision point Q is

$$
\underline{v}_Q = \underline{v}_A + \underline{\Omega} \times (\underline{Q} - \underline{A}) \tag{15}
$$

Therefore the velocity of node D after the collision is:

$$
\underline{v}_D^+ = \underline{v}_D^- - \underline{v}_{Dk'}^- + \underline{v}_{Qk'} \tag{16}
$$

where
$\underline{v}_D^+$ = velocity of node D after the collision, $\underline{v}_D^-$ = velocity of node D before the collision, $\underline{v}_{Dk'}^- = (\underline{v}_D^- \cdot \underline{k}')\underline{k}'$ is the component of $\underline{v}_D^-$ along $\underline{k}'$, and $\underline{v}_{Qk'} = (\underline{v}_Q \cdot \underline{k}')\underline{k}'$ is the component of $\underline{v}_Q$ along $\underline{k}'$.

Every time a collision between a node and a triangle is detected, a new post collision velocity $\underline{v}_D^+$ is calculated and assigned to the node. The impact does not modify the component of the nodal velocity tangent to the triangle surface. This component, before and after the collision, is given by

$$
\underline{v}_{Dt} = \underline{v}_D^- - \underline{v}_{Dk'}^- \tag{17}
$$

It is simple to use this tangential velocity for computing the effect of viscous damping or the effect of Coulomb friction. In these cases a friction force, directed opposite to the velocity $\underline{v}_{Dt}$, would have to be added when computing the total force applied to the node D. The correction of the position of the node is a more difficult task. A reasonable solution seems to be to use the position of the intersection point Q as the new coordinate for the node involved in the collision. However, this solution generates instability with the integration algorithm for certain special configurations of the network. An alternative to this approach is to keep the current position (before the collision) of the colliding node in the next simulation step, as recommended by Volino et. al. [6]. In this way we manipulate the velocity and position of the node according to the physical constraints and we integrate the imposed values at the following simulation step. The most difficult case for our self-collision algorithm is

probably a configuration involving the sliding of two persistently interacting surfaces over each other. In this situation, the nodes involved in the collision are subjected to correction in position and velocity at almost every simulation step. When trying to address the sliding problem between colliding surfaces, the following considerations are important:

- Because the simulated collision between the node and the triangle is completely inelastic, the node loses its relative velocity in the normal direction to the colliding surface. After a collision is detected, the distance in the normal direction to the surface between two positions of the node at different simulation steps, progressively decreases. If the intersecting line r is calculated using the current and updated position of the node, and anticipating some numerical error, after several detected collisions the penetration can no longer be detected. A solution for this problem is to consider the intersecting line r to be along the edge of the triangle containing the colliding node. The nodes at the vertex of the edge are used to perform the collision test as described above. Because the separation between the two nodes is almost constant during the simulation, the collision is always detected including the case where the node that crosses the surface of the triangle loses its relative velocity during the collision

- There are rare situations where the algorithm we have described allows a node to get locked in place along a surface where it should slide. The problem may be experienced when the after a collision the position of the node is chosen to be coincident with the position of the node at the previous step. In this case, if multiple subsequent collisions are detected, the node maintains its position and no tangential displacement is allowed.

3.3 Spatial Enumeration

Collision and self-collision detection are often the most time consuming part of the simulation algorithm. The collision algorithm described above provides a good response for a wide range of cloth drape and manipulation problems involving very complex cloth/cloth interactions. The algorithm can be directly applied to our network, comparing each node with all the triangles by using the described geometric approach. All the calculations must be performed every simulation step and the time complexity of the algorithm as presented is $O(n^2)$, where n is the number of nodes in the network. When comparing all the nodes with all the triangles of the network, many (or most) of the performed tests are superfluous. Regions of a deforming surface that are separated by great distance are very unlikely to collide. Adjacency and surface curvature are two criteria, which allow building a hierarchy between different areas of the network thus avoiding collision checks on large regular regions. This is the fundamental idea of the collision algorithm proposed by Volino et. al.[6] and Volino and Thalmann [7], which reported rapid simulation times. An alternative to

this approach is to use a spatial enumeration method. The entire 3D space where the simulation of our deforming structure is being performed is divided into small volume elements or boxes. Collisions are checked only between those nodes of the network, which fall in the same box. This is the solution that we have used in our algorithm. The entire 3D space where the motion of the network occurs, is discretized into thousands of cubes. Each resulting box is identified by a number or key (K). From the coordinates of the network it is possible to calculate the key of the cube where a particular node has arrived. We number our box system left to right, increasing the key along the direction of the x axis, and up along y axis (see Fig. 9).

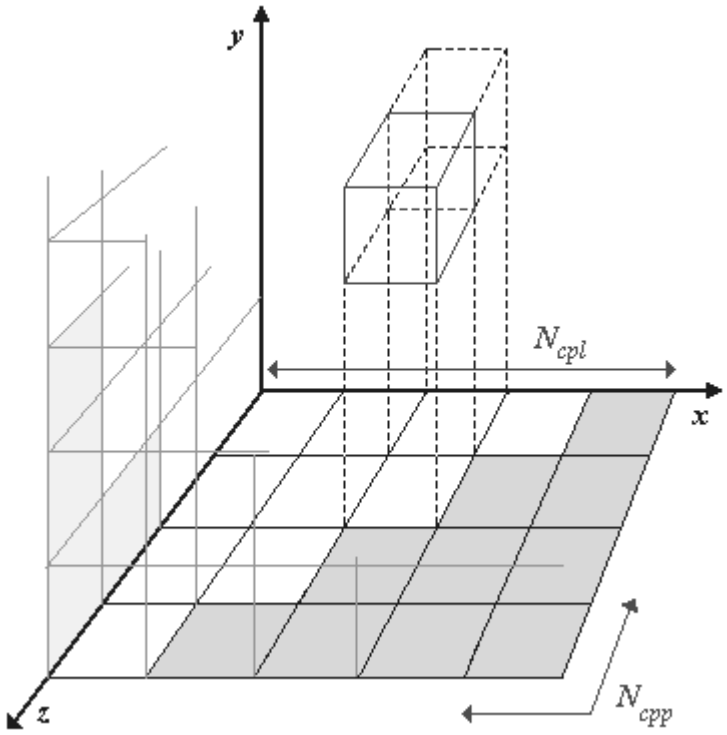

Figure 9: Spatial enumeration system

If N_{cpl} is the number of boxes in the x direction and N_{cpp} is the number of boxes

in the x, z plane. The key for a generic node M with coordinates x_M, y_M, z_M is

$$K = dx_M + (N_{cpl} \cdot dz_M) + (N_{cpp} \cdot dy_M) \tag{18}$$

where:
$dx_M = x_M \mathbf{div} L_{ce}$
$dy_M = y_M \mathbf{div} L_{ce}$
$dz_M = z_M \mathbf{div} L_{ce}$
L_{ce} = length of the edge of a single cube of the spatial enumeration

If two nodes of the network have the same key, they fall inside the same cube. Then the collision detection algorithm is activated and performs the checks for possible intersection between the triangles that have one of these two nodes as a vertex. The problem is to find a data structure that contains all the information about each cube: key, number of nodes which fall inside the cube, and the indices of these nodes. The choice must consider the following peculiarities:

- The data structure must be fast to access, since the insertion of a node of the network into the data structure must be done at every simulation step for each node of the network. The operation of testing if two nodes fall into the same box must also be executed every simulation step for every cube stored in the data structure
- The number of cubes of the spatial enumeration can be very high. Not all the cubes have to be stored in the data structure, since the algorithm has to keep track only of those boxes that contain at least one node.

A possible solution is the use of a hash table indexing an array of structures, that allows storing and retrieving only the boxes that contain one or more nodes of the network. Each structure contains all the information related to a single cube. The dimension N_{array} of the array is a certain fraction of the total number of cubes N_{box} of the enumeration. While key K can vary between 0 and N_{box}, the index h of the array has a smaller range, between 0 and N_{array}. The hashing function that transforms the key K into the index h of the array is given by:

$$h = H(K) = (K) \mathbf{mod} (N_{array}) \tag{19}$$

where $H(K)$=hashing function and mod returns the remainder of the division between two integers. The hashing-collision inside the array is defined as the situation where two elements with different key values provide the same value of the index h. This situation can be handled by a quadratic survey that is equivalent to the following recurrence relation

$$\begin{aligned} h_{i+1} &= h_i + d_i \\ d_{i+1} &= d_i + 2 \end{aligned} \tag{20}$$

with $d_0 = 1$ and $h_0 = H(K)$ When inserting a new element into the hash table, the calculation of the hash index h is iterated until the element with the same key or an empty element is found (see Worth [8] for more details about hashing-functions). The maximum number of tests performed before finding the correct position for a new element is N_{box}/N_{array}. After filling the array with all the nodes of the network, we have available a list of elements, one for each non-empty cube, that we can analyze for finding possible situations which can compromise the consistency of the network.

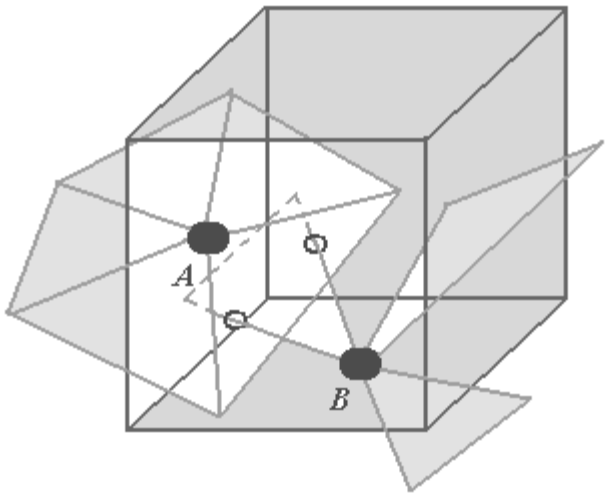

Figure 10: Collision detection

If two nodes fall inside the same box, we believe that applying the geometric method described in this article will account for all possible collision configurations. All the edges of triangles that have node A as one of the vertices (see Fig. 10) are compared with all triangles that have node B as one of the vertices. At the first detected collision, the procedure stops and the correction in position and velocity is applied at node A. Eventually the same procedure is performed for node B. When more than two nodes fall in the same box, the procedure is iterated on all nodes checking all possible combinations. The edge length of the cubes of the spatial enumeration is a delicate parameter to set. In fact, if the box is too small, the collision analysis is faster but possible collision configurations can be missed. For

instance, two nodes in a relative position which compromise the consistency of the network could fall in different cubes. Otherwise, if the box is too large in volume, many useless consistency tests are performed. By experiments conducted with several possible collision configurations we have determined an efficient value for the cube dimensions has to be comparable with the dimensions of triangle of the network. A possible size for the elementary box can be defined using the maximum expected velocity V_{max} of a node of the network and the time step Δt, as

$$L_{ce} > V_{max} \cdot \Delta t, \tag{21}$$

We have tested our collision algorithm with and without using the spatial enumeration. The difference in simulation time is quite considerable. When using the spatial enumeration the complexity of the algorithm becomes proportional to the number of boxes occupied by more than one node of the network.

4 Run-time algorithm path

In every simulation time step, the following operations are performed:

- The external loads are applied to the nodes of the network and internal reactions are computed resulting in a total resultant force for each node.
- The positions and velocities of the nodes are advanced ahead in time using Runge-Kutta integration.
- The hash table is filled by computing the key K for each node of the network and the corresponding hash table index h.
- The hashing table is scanned looking for possible collision configurations. If two or more nodes fall into the same box, the collision detection algorithm is applied to these nodes.
- The positions and velocities for those nodes that compromise the consistency of the network are corrected.
- For those nodes where no self-collision is detected, an algorithm for collision detection with external surfaces is performed. For regular surfaces (such as planes or regular solids) the collision algorithm simply compares the coordinates of the node with the coordinates which define the position of the external surface. If the contact surfaces are more complicated, these surface are also triangularized and the self algorithm is applied again.
- Updated coordinates and velocities define a new configuration of the surface that can be stored as a frame and rendered.

5 Results and Conclusions

For our experiments to validate the algorithm, the simulation and collision detection code were implemented in C^{++}, using Visual C^{++} and MFC on the Windows NT platform. The graphic output was realized with OpenGL. The first example in Fig. 11 shows some frames from the simulation of a vertical strip which falls on a flat horizontal surface under the effect of the gravity. One end of the strip is fixed to the horizontal surface and two vertical feeders (not shown in the pictures) constrain the strip movement. An initial imperfection is imposed at the bottom of the strip. This is necessary for initiating the out of plane buckling. The imperfection constructed to result in an irregular evolution of the motion in 3D space. This case is a strong test for any collision algorithm: multiple layers of the fabric are permanently in mutual contact and inter-penetration is completely avoided. The network is a structure of 125×3 nodes and the base element is a square of 5 mm edge. The total simulation time was 14 hours on a Pentium 200Mhz machine. Fig. 12 shows a second case where collision and self-collision algorithms play an important role. An initially flat tablecloth falls on a solid parallelepiped displaced in a non-symmetric configuration. The effect is that the tablecloth slides off (since there is no friction) the solid under the effect of gravity and crumples to the ground showing several folds and multiple layers overlapping. The sequence of frames clearly shows the interaction between the fabric and external surface. The structure of the tablecloth is modeled with 25×25 nodes: 576 square elements with 10 mm edge length. The total simulation time was 9 hours on the Pentium 200Mhz machine. The collision detection algorithm does not heavily affect the simulation time and it seems to provide an efficient response to difficult cases where folding on multiple layers and buckling occur. However, further improvements are probably required in order to reduce the time spent in detecting collision every simulation step. The number of test performed is still to high and it might probably be reduced using a data structure that allows storing the path of colliding nodes. Also, the after collision response should allow more flexibility to the network of nodes, reducing the constrains imposed to the position of the nodes and allowing a more realistic simulation of the motion.

6 Acknowledgments

Financial support for this work was provided by the National Textile Center which is funded by the US Department of Commerce.

References

[1] Moore, M., and Wilhelms, J., (1988), "Collision Detection and Response for Computer Animation," *Computer Graphics*, Volume 22, Number 4.

[2] Wilson, A., Larsen, E., Manocha, D. and Lin, M., (1998), "A System for Interactive Proximity Queries of Massive Models," UNC Computer Science Technical Report TR98-031.

[3] Krishnan, S., Gopi, M., Lin, M., Manocha, D., A. Pattekar, (1998), "Rapid and Accurate Contact Determination between Spline Models using ShellTrees," to appear in Proceedings of Eurographics'98.

[4] Gottschalk, S., Lin, M., Manocha, D., (1996), "OBBTree: A Hierarchical Structure for Rapid Interference Detection," Proceedings of ACM SIGGRAPH 96.

[5] Cohen, J., Manocha, D., Lin, M., Ponamgi, M., (1994), "Interactive and Exact Collision Detection for Large-Scaled Environments," Technical report TR94-005, Department of Computer Science, University of N. Carolina, Chapel Hill.

[6] Volino, P., Courchesne, M., Thalmann, N. M., (1995), "Versatile and Efficient Techniques for Simulating Cloth and Other Deformable Objects," Proceedings of SIGGRAPH 95.

[7] Volino, P., Thalmann, N. M., (1996), "An Evolving System for Simulating Clothes on Virtual Actors," *IEEE Computer Graphics and Applications*, Vol 16, No. 5.

[8] Wirth, N., (1976), Algorithms + Data Structures = Programs, Prentice-Hall.

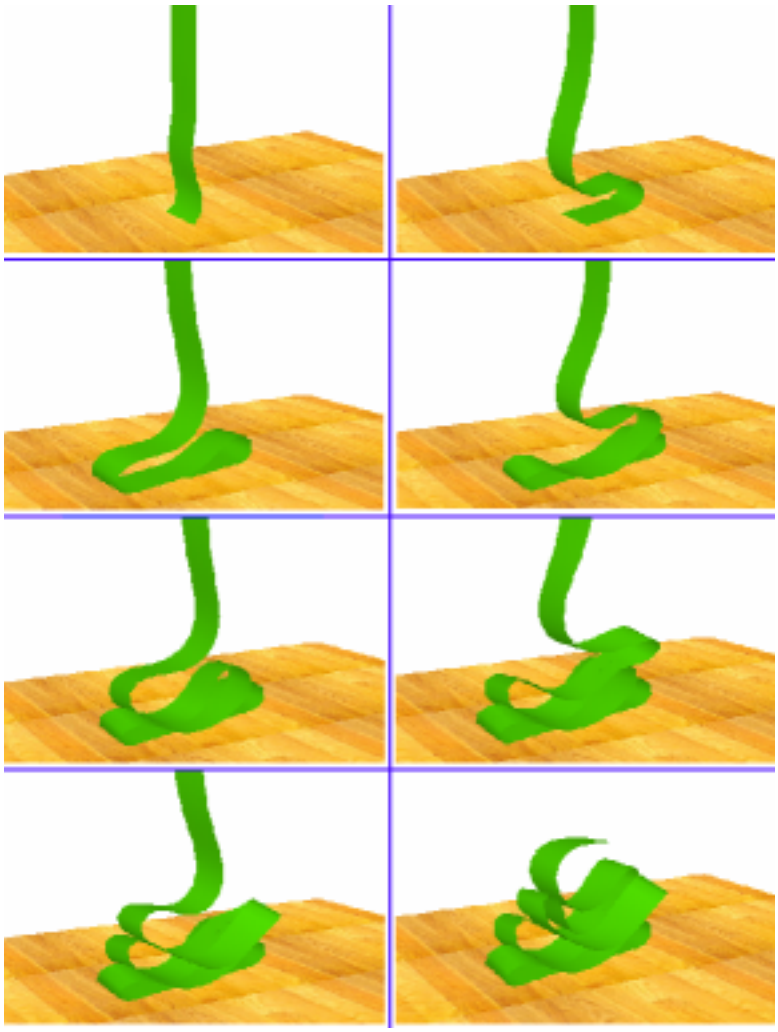

Figure 11: Falling Vertical Strip

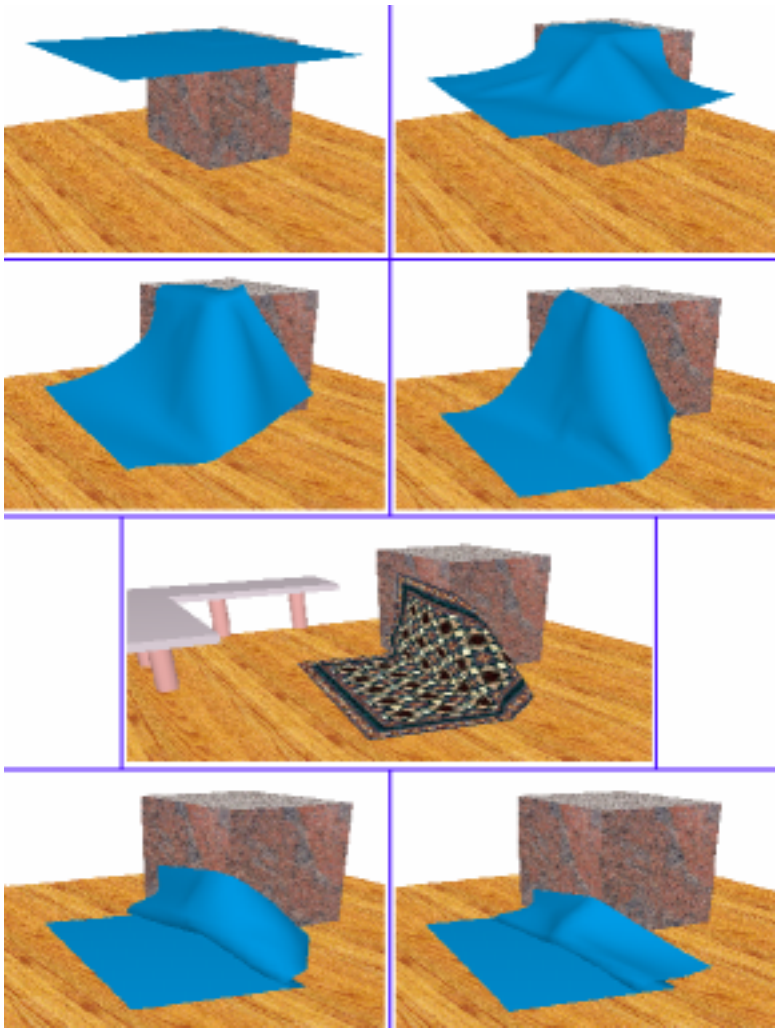

Figure 12: Falling Tablecloth