Real-Time Knot Tying Simulation

Joel Brown[(1)] Jean-Claude Latombe[(1)] Kevin Montgomery[(2)]
(1) Computer Science Department (2) Department of Surgery
Stanford University, Stanford, CA 94305

Abstract

While rope is arguably a simpler system to simulate than cloth, the real-time simulation of rope, and knot tying in particular, raise unique and difficult issues in contact detection and management. Some practical knots can only be achieved by complicated crossings of the rope, yielding multiple simultaneous contacts, especially when the rope is pulled tight. This paper describes a simulator allowing a user to grasp and smoothly manipulate a virtual rope and to tie arbitrary knots, including knots around other objects, in real-time. One component of the simulator precisely detects self-collisions in the rope, as well as collisions with other objects. Another component manages collisions to prevent penetration, while making the rope slide with some friction along itself and other objects, so that knots can be pulled tight in believable manner. An additional module uses recent results from knot theory to identify which topological knots have been tied, also in real-time. This work was motivated by surgical suturing, but simulation in other domains, such as sailing and rock climbing, could benefit from it.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Physically Based Modeling;

Keywords: rope simulation, knot tying, knot identification, self-collision detection, collision management, friction modeling, surgical suturing

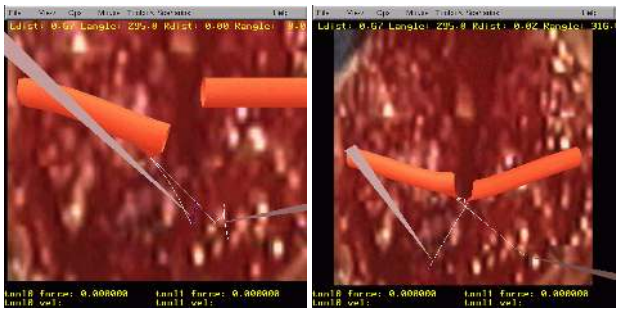

Figure 1: Forceps pull a suture which (a) Pulls a vessel down, (b) Pulls two vessels together

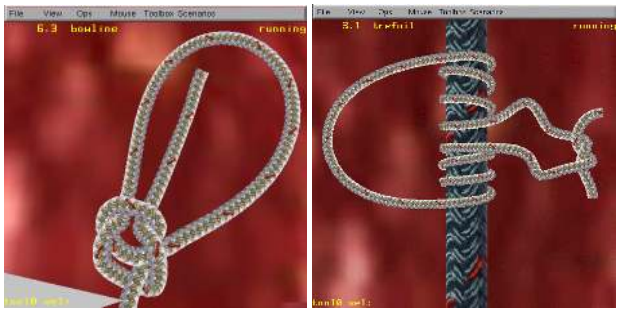

Figure 2: Two practical knots: (a) bowline (b) prusik knot

1 Introduction

Today there is growing interest in *real-time* surgical simulation, especially as a tool for training surgeons [Dawson 2002]. Recent research has focused on creating efficient computational models of visco-elastic tissue structures [Delingette 1998]. But surgical simulation involves many aspects other than tissue deformation. In particular, a key component of many surgical procedures is suturing. It involves manipulating a non-elastic suture through tissue, pulling the strands to bring tissue surfaces in contact with one another, and tying knots. Figure 1 shows two snapshots during the simulation of an anastomosis operation (suturing of a severed blood vessel). Suture simulation is discussed in [Kühnapfel et al. 2000] and a method is presented in [Brown et al. 2002], including the interaction between the suture and both deformable tissue and rigid surgical instruments. However, to our knowledge, there still exists no surgical simulator addressing the issues that arise when the suture comes into contact with itself (self-collision). In [Kühnapfel et al. 2000] these issues are regarded as the most difficult part of collision management. Contacts must be detected and handled to enable the user to naturally tie arbitrary knots. The task of knot-tying in surgical simulation is identified in [Brown et al. 2002], but no solution is offered. It is also mentioned in [Larsson 2001], where some ideas for modeling a suture using masses and springs are sketched, without providing an actual algorithm or results. Specific knots useful in surgery are described in [Silverstein 1999].

Other possible applications of knot-tying simulation include learning to tie knots for sailing and rock-climbing. Many books describe knots used in these domains [Graydon 1992; Pawson 1998]. A system is presented in [Phillips et al. 2002] where a rope in a loosely knotted configuration is pulled tight, and the knot is preserved, using an impulse model for collision handling. The rope is modeled as a spline of linear springs, with spheres placed on the control points to represent the rope volume. The spheres tend to bunch up or stretch apart during the simulation, due to the spring model, but collision handling prevents the rope from passing through itself. This system is not interactive, as it takes several minutes of computation to tighten simple knots, like the overhand or square knots. Moreover, most ropes, including surgical sutures, only stretch by small amounts. Off-line motion planning techniques to undo a knot are also presented in [Ladd and Kavraki 2002]. The notion of knot energy and knot untangling by energy minimization are discussed in [Kusner and Sullivan 1998].

Rope simulation is related to cloth simulation, an area that has recently attracted much attention in Computer Graphics (e.g., [Baraff and Witkin 1998; Bridson et al. 2002; House and Breen 2000; Provot 1995; Volino and Magnenat-Thalmann 1995]). In many ways, a rope is a much simpler system to simulate. While clothes are 2-D objects embedded in 3-D space and can take many shapes, ropes are essentially 1-D objects with relatively simple geometry.

But knot tying is quite specific to ropes and raises unique issues in contact detection and management. Some very practical knots can only be achieved by complicated crossings of the rope, yielding multiple simultaneous contacts when the rope is pulled tight (Figure 2). While collision detection between rigid objects is a well studied problem, collision detection between deformable objects and self-collision detection within a deformable object are relatively new topics (see Section 4). Changes of shape make it more difficult to use precomputed data structures for expediting collision tests during simulation. Moreover, collisions within a thin 1-D object are easy to miss in discrete-step simulation, and missing any would cause the object to unrealistically pass through itself.

In this paper, we describe a simulator allowing a user to grasp and smoothly manipulate a virtual rope and to tie arbitrary knots, including knots around other objects, in real-time. During simulation, the rope's length is kept approximately constant. Its configuration is re-computed at rates up to 140Hz. One component of the simulator precisely detects all self-collisions in the rope and collisions with other objects. Another key component handles collisions to prevent penetration, while making the rope slide against itself and other objects at contact points with some frictional sticking, so that knots can be pulled tight in realistic manner. We propose a new collision response scheme that is based on coordinating local micro-motions of the rope. This scheme creates a natural model of friction, which provides a convenient and efficient alternative to the classical Coulomb's model. An additional module, independent of the main simulator, identifies which topological knots have been tied in the rope, also in real-time. This module is based on recent results from knot theory.

Although our techniques are fairly general, we have created them specifically for surgical suturing. In this application, the mechanical properties of both the suture (very light weight relative to bending stiffness) and its environment (viscous solution) allow our simulator to ignore gravity. At first sight, this may be seen an oversimplification for other applications. In fact, only a small component of the simulator relies on this assumption, and this component could be easily modified to handle gravity. The key components – detection and management of self-collisions, and knot identification – do not depend on it. Note also that many ropes have relatively high bending stiffness, which limits the effect of gravity and actually facilitates knot making.

2 Overview of Rope Simulator

A rope is a cylinder of given length L and radius R, that bends smoothly, while stretching minimally. Because the rope is not elastic, we do not use springs and forces to model its motion. Instead, we preserve key properties of the rope by enforcing a set of physically-motivated constraints that are dynamically updated at each simulation cycle. The most important constraints derive from the fact that the rope should not penetrate itself or other objects, while contacts should allow frictional sliding.

To keep the rope's appearance and motion smooth, while facilitating the algorithmic treatment of these constraints, in particular the management of (self-)collisions, we model the rope's axis as a kinematic chain made of many straight rigid links connected at their endpoints (nodes) by spherical joints. Each joint allows two degrees of rotational freedom. The links all have the same short length. To improve the rope's graphic appearance (especially when one zooms in on a small section of rope), we connect every two successive links by a parabolic arc at rendering time. Figure 2 shows configurations of a rope with 200 links of unit length. In the following, the term *link* will always refer to a straight piece of the rope axis, while the term *segment* will designate the cylinder of radius R around a link.

We move the rope by grasping one or several of its nodes and displacing them. In our experimental system, this is done by means of one or several devices, ranging from a simple mouse to 3-D articulated linkages, allowing the user to position and move a graphic cursor or any representation of grasping tools (e.g., surgical forceps).

Events like rope grasping and self-collisions, as well as collisions with other objects, result in creating new constraints. Ungrasping and contact disappearance yield constraint removal. For example, when a node is grasped, this creates a "grasp" constraint, forcing the node's position to agree with the position of the grasping tool. Also, when a link of the rope comes into contact with another link or with another object, "contact" constraints are created, whose treatment will be described in Section 5.

The overall simulation algorithm is the following, where each execution of the loop is called a *cycle*:

Algorithm Rope-Simulator

Loop:

1. Read the new positions of the grasped nodes
2. Compute the new rope configuration
3. Find all (self-)collisions in this configuration
4. Adjust the rope configuration to remove overlaps
5. Display the final configuration

The computation of the new rope configuration at Step 2 is done by the FTL (Follow The Leader) algorithm presented in Section 3, taking into account contact constraints established at the previous cycle. Finding (self-)collisions at Step 3 is discussed in Section 4.

It is essential that the rope cannot pass through itself at any cycle. This could happen if grasped nodes were moved too quickly. So, we set a threshold δ on the distance that any grasped node may travel between two successive executions of Step 1. As FTL will cause no points in the rope to move by more than δ, we set δ to be slightly smaller than the rope's radius R. Therefore, if the simulator runs at Q Hz, each grasped node can be moved by up to $Q \times R$ units/second without noticeably lagging behind the grasping tool.

In the rest of this paper, without loss of generality, we set the length of each link to one unit. We denote the number of links by n, so that the length L of the rope is equal to n. We denote the nodes by N_i, $i = 1, 2, \ldots, n+1$, and the position of N_i in 3-D space by $\vec{v}_i$.

3 Computation of Rope Configuration

The simplest rope motion occurs when one node of the rope is grasped, and the rest of the rope (in general, two strands) follows along without (self-)collision. To achieve realistic deformation, each link must maintain its length while following in the direction of the grasped node. We compute the motion of the rope in a "follow-the-leader" manner. Let N_i be the grasped node and assume that it is moved from position $\vec{v}_i^{\,\text{old}}$ to $\vec{v}_i^{\,\text{new}}$ within one cycle. We compute the new position of its neighbor N_{i+1}, if $i \leq n$, as the point $\vec{v}_{i+1}^{\,\text{new}}$ a unit distance along the line from $\vec{v}_i^{\,\text{new}}$ to $\vec{v}_{i+1}^{\,\text{old}}$. This motion is propagated along the rope until the new position of node N_{n+1} has been computed. The computation proceeds in the same way along the other strand, from node N_{i-1} toward N_1.

More formally, the algorithm propagating the motion of a grasped node N_i toward another node N_j is as follows:

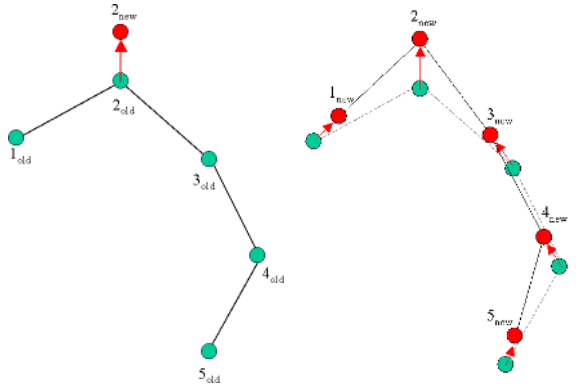

Figure 3: (a) Grasped node 2 is moved. (b) Nodes 1 and 3 follow, then 4, then 5.

Algorithm FTL1(i, j)

1. $\varepsilon = sign(j-i)$
2. $a \leftarrow i$
3. $b \leftarrow a+\varepsilon$
4. while $b \neq j+\varepsilon$:
   4.1 $\vec{u} \leftarrow (\vec{v}_b - \vec{v}_a)/\|(\vec{v}_b - \vec{v}_a)\|$
   4.2 $\vec{v}_b \leftarrow \vec{v}_a + \vec{u}$
   4.3 $a \leftarrow b$
   4.4 $b \leftarrow a+\varepsilon$

The simulation of the entire rope is obtained by calling FTL1$(i,1)$ and FTL1$(i,n+1)$. Figure 3 illustrates this simple technique for a 4-link rope, in which N_2 is the grasped node. The motion between the configurations of Figures 3(a) and (b) is thus achieved by calling FTL1(2,1) and FTL1(2,5).

Often, the user may grasp two nodes and pull both of them, or grasp one node while a second node is constrained by a contact (as described later). In this case, the two constrained nodes N_i and N_j may suggest moving intermediate nodes in different directions, so we average together the results of following the leader in each direction, yielding the following algorithm:

Algorithm FTL2(i, j) (for $i < j$)

1. Save original positions $\vec{v}_i, \ldots, \vec{v}_j$ in array $O(i:j)$
2. Call FTL1$(i, j-1)$
3. Save new positions $\vec{v}_i, \ldots, \vec{v}_j$ in array $C(i:j)$
4. Restore original positions $\vec{v}_i, \ldots, \vec{v}_j$ from O
5. Call FTL1$(j, i+1)$
6. For $a = i+1, \ldots, j-1$, compute final position:
   $\vec{v}_a \leftarrow (\vec{v}_a + C(a))/2$

The averaging may result in changing slightly the length of some links. However, as long as no section of the rope is fully extended, variations in link lengths tend to self-correct over several consecutive cycles, so that the total variation in the rope's length between two constrained nodes remains small. As nodes are not shown in the graphic rendering, this variation is not visually noticeable, nor does it affect such operations as knot tying. Similar averaging, known as "strain limiting," has previously been used in cloth simulation to deal with conflicting forces [Bridson et al. 2002; Provot 1995]. To prevent the user from stretching an already extended section of the rope by pulling its endpoints apart, we could set a maximal average length h per link and freeze the graphic cursors when the user's inputs cause the distance between two constrained nodes to exceed $k \times h$, where k is the number of links between the two nodes. In our current system, we rely on graphical feedback and the discipline of the user not to keep stretching a rope section.

Using the above two algorithms, we create a final algorithm, FTL, shown below. It computes the new configuration of the entire rope at each simulation cycle, based on all the grasp and contact constraints. Prior to calling FTL, the node index of each grasp and contact node is placed into a sorted array $A(1:q)$.

Algorithm FTL

1. If $A(1) > 1$, then call FTL1$(A(1), 1)$
2. If $A(q) < n+1$, then call FTL1$(A(q), n+1)$
3. For $i = 1 \ldots q-1$, call FTL2$(A(i), A(i+1))$

Note that FTL cannot cause any point in the rope to move by more than the maximum displacement of any grasped node. FTL can handle large numbers of grasped and/or contact nodes without significant slowdown.

Despite (or thanks to) its simplicity, experiments with FTL show that it creates very believable rope motions under the assumption that the rope is unaffected by gravity. In applications where gravity affects rope motion, a simple mathematical expression of the state of the rope at each cycle could be used to derive a more realistic configuration of the rope. No other components of the simulator would have to be modified.

4 Collision Detection

When tying knots, a rope often collides with itself and possibly with other objects. All (self-)collisions must be detected at every cycle, as each one of them will affect the rope's motion at the next cycle. Efficient but precise detection and management of collisions are the keys to our knot-tying simulation.

Much research has been devoted to checking collisions between rigid objects (e.g., [Gottschalk et al. 1996; Lin and Gottschalk 1998; Quinlan 1994], to cite only a few works). However, less effort has been spent on collision detection between deformable objects [Brown et al. 2002; Larsson and Akenine-Möller 2001; Joukhadar et al. 1999; Smith et al. 1995; van den Bergen 1997; Volino and Magnenat-Thalmann 1995], and even less on finding self-collisions in a deformable object [Guibas et al. 2002; Lotan et al. 2002; Volino and Magnenat-Thalmann 1995].

When a deformable object A is modeled as a large collection of small rigid pieces $a_1, a_2, \ldots, a_n$ – e.g., the segments in our rope model – an approach to finding self-collisions, called the *grid method*, is to define a uniform grid of cubes over the 3-D space, compute the cubes intersected by each piece a_i, and store the results in a hash-table [Halperin and Overmars 1998]. At each simulation cycle, this data structure is re-computed from scratch; self-collisions are then found by considering every intersected cube and, for each one, testing whether each pair of pieces a_i and a_j intersecting this cube overlap. A is said to be *well-behaved* if all its pieces a_i have (approximately) the same size and the centers of the minimum enclosing spheres of any two pieces cannot come closer than some absolute constant. Under the assumption that A is well-behaved, one can show that, on average, each piece overlaps at most $O(1)$ other pieces [Halperin and Overmars 1998]. So, the grid method always takes $\Theta(n)$ time at each cycle, which is optimal in the worst case. This approach can be used to detect self-collisions in our rope model. Since displacements are short enough at each cycle and we remove all overlaps when they occur, our rope model is well-behaved.

Another approach to finding self-collisions in a deformable object A is to build a *bounding-volume hierarchy* (BVH) representing the shape of A at successive levels of detail. In the past, BVH techniques have been widely used to detect collisions between rigid

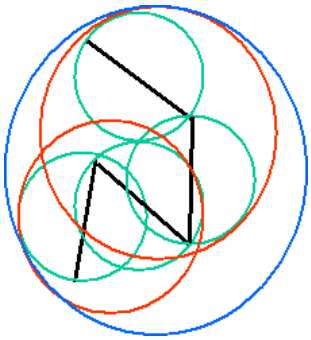

Figure 4: Bottom-up construction of the BVH of an 4-link rope.

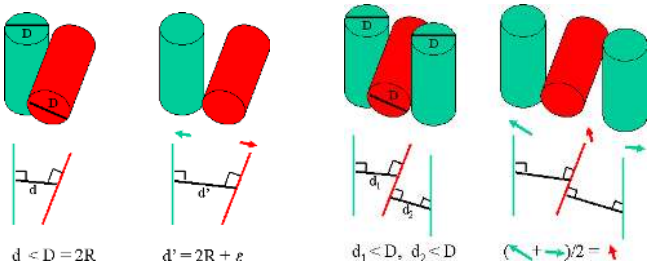

Figure 5: (a) Two rope segments are colliding. They are moved away from each other out of collision. (b) The segment in the middle (red) collides with the other two segments (green). Displacement vectors for the green segments are computed as in (a), but an average is used for the red segment.

objects [Gottschalk et al. 1996; Quinlan 1994]. These techniques must be adapted to deal with deformable objects and self-collisions (see [Brown et al. 2002; Guibas et al. 2002; Lotan et al. 2002; van den Bergen 1997]). For our rope model, we build a BVH from the bottom up (see Figure 4). We first bound each rope segment by its minimal enclosing sphere. The spheres thus obtained are installed as the leaves of the BVH, in the same order as the segments along the rope. Then, we bound pairs of successive spheres by new spheres to form each next level of the hierarchy. Hence, the resulting BVH is a balanced binary tree. Each intermediate sphere bounds tightly its two children. So, it also encloses all the leaf spheres below it, but in general this bounding is not tight. The root sphere encloses the entire rope and all the other spheres. To find the self-collisions in A, we explore two copies of the BVH from the top down. Whenever two BVs (one from each copy) are found not to overlap, we know that they cannot contain colliding segments and hence we do not explore their contents. When two leaf spheres overlap, the shortest distance between the two underlying links is computed. If it is less than the rope diameter $2R$, then the two segments are reported to collide. However, no segment is ever considered to be in collision with itself or its immediate neighbors along the rope chain.

The topology of the BVH is computed once before any simulation and then remains fixed. During simulation, only the positions of the bounding spheres and the positions and radii of the higher level spheres are re-computed at each cycle, and this computation is done from the bottom up, so that no sphere is updated more than once. Moreover, the update of each sphere takes constant time, so that updating the BVH takes $O(n)$ time. It is shown in [Guibas et al. 2002; Lotan et al. 2002] that at each cycle, finding all self-collisions takes $\Theta(n^{4/3})$ in the worst case. (The proof of this result requires the rope model to be well-behaved.)

While the BVH method is only slightly less efficient than the grid method to detect self-collisions, it has a major advantage in environments where the rope may interact with other rigid and non-rigid objects. In such environments, we can efficiently detect collisions between the rope and these objects by comparing their respective BVHs. The grid method would not be as efficient for this purpose. So, our simulator uses the BVH method. A separate BVH is computed for every object in the rope's environment.

At each simulation cycle, the maximum displacement allowed for any grasped node is $\delta < R$. Since FTL can cause no point in the rope to move by more than δ, no link can possibly pass through another link or through any fixed object, even a very thin one, without the simulator detecting a collision.

5 Collision Management

In this section, we focus primarily on the management of self-collisions. Collisions between the rope and other objects are treated in a similar way, as described briefly at the end of the section.

In a discrete simulation process like the one of Section 2, we typically detect self-collisions after they have occurred. So, in the new configuration computed by FTL, the rope may penetrate itself by as much as δ. Since penetrations are physically impossible, we remove them by adjusting the positions of some nodes. We also create contact constraints to be enforced at the next simulation cycle.

The most basic form of self-collision treatment is as follows. When two links are detected to be at a distance $d < 2R$ from each other (i.e., the two corresponding segments are colliding), the pair of closest points in the links is identified and Δ is defined as the line passing through them. Then, an equal (but opposite) displacement vector is applied to each link along Δ. See Figure 5(a). This displacement is just long enough to take the segments out of collision, with a slight "safety margin" ε to allow for a sliding motion discussed below. Hence, each link is shifted away by $R - d/2 + \varepsilon/2$. (This shift results in a slight change of the total length of the rope.) Simultaneously, a contact constraint is created that requires the endpoints of the two links not to move at the next call of FTL.

This method of collision response creates a simple, but effective "algorithmic" model of friction. The fact that the contact constraints fix some nodes in place for one cycle gives the rope a sticky behavior. But since the colliding segments have been pushed slightly out of contact, they will not be in collision during the following cycle, allowing the rope to slide along itself if the local motion computed by FTL has a component tangential to the contact. After this new motion step, a new collision between the two segments may, or may not, happen again. If it happens, then the contact constraint is re-created for one cycle. In other words, this is what happens over three consecutive cycles:

CYCLE k:	- A collision is detected between two segments of underlying links ℓ and ℓ'.
	- A contact constraint C is created.
CYCLE $k+1$:	- Because of C, FTL leaves the four
	- endpoints of ℓ and ℓ' fixed.
	- The contact constraint C is removed.
CYCLE $k+2$:	- FTL is free again to move the four nodes.

Over successive cycles, this treatment produces a frictional sliding behavior, which allows knots to be pulled tight. A higher ε lets the rope slide more easily along itself, but too large an ε leads to unnatural jumps after collisions and a frictionless look, making it

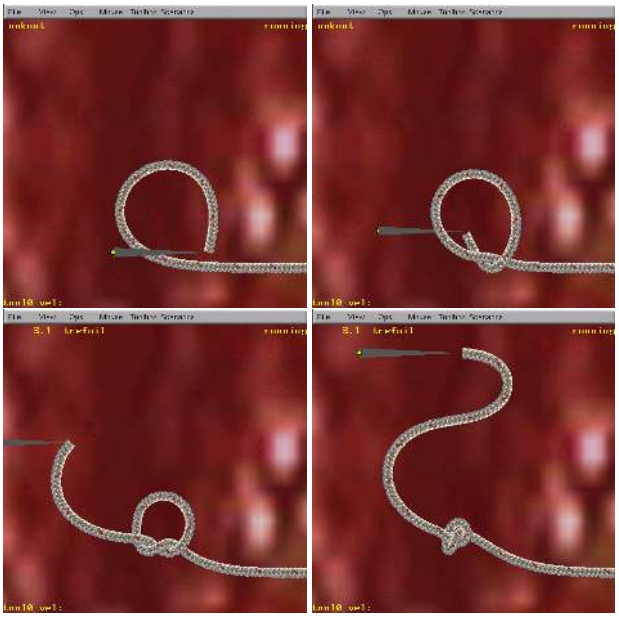

Figure 6: Key steps in the tying of a trefoil (or overhand) knot.

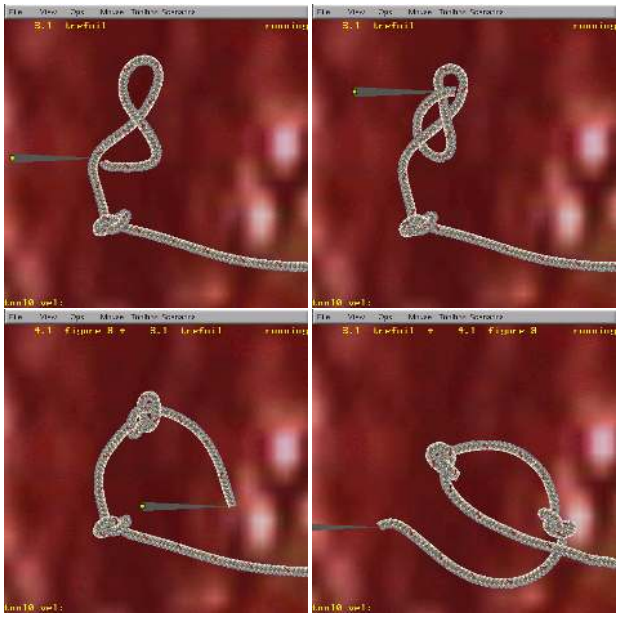

Figure 7: A second knot (figure-8) is tied in the rope from Figure 6.

easy for knots to pull themselves out after being tightened. Conversely, too small an ε causes rope segments in contact to slide along each other very slowly (or to stick), making it difficult or impossible to tighten a knot. However, between these two extremes, there is a range of values of ε that produce believable rope behaviors (see Section 6). Previous research in Robotics and Computer Graphics has mainly used the Coulomb's model of friction [Bridson et al. 2002; Erdmann 1994; Mirtich and Canny 1995]. We believe that our algorithmic model is a convenient and computationally efficient alternative to handle contacts between textured and slightly deformable surfaces, such as those of ropes, which consist of many interlaced fibers.

All collisions between segments can be processed concurrently. However, a slight modification of the above treatment is necessary to handle cases when the same segment collides with several other segments. In that case, the multiple collisions suggest different displacements of the same nodes. We then apply the average of these displacements to each node, as illustrated in Figure 5(b). This averaging no longer guarantees the removal of all segment overlaps. For instance, in the situation of Figure 5(b), the average displacement of the middle segment (in red) could be too small to move it out of collision with the other two segments (in green). But any remaining overlap does not survive long, since it is re-detected at the next simulation cycle, which leads the colliding segments (notably, the green segments in Figure 5(b)) to be pushed further apart. In fact, a situation like that of Figure 5(b), where a segment is being squeezed between two other segments, occurs mainly when a knot gets tight. The side-effect of any remaining overlap is to increase friction inside the knot, which is exactly what is desirable. (Actually, some common knots draw their strength from the fact that a piece of rope "bites" over another piece, when the outgoing strands are pulled apart. This is in particular the case of the alpine butterfly knot shown in Figure 10(a).) On the other hand, overlaps are too small and/or short-lived to be visually noticeable. Moreover, they cannot cause the rope to pass through itself, since the contact constraint prevents FTL from moving the corresponding nodes as long as the overlap remains.

A final improvement is made to allow for knots to move properly in space over successive cycles. Indeed, as a knot becomes tight, the cluster of segments forming the knot will keep containing collisions and the contact constraints will leave several nodes fixed at almost every iteration. The above techniques would then cause the knot to always be constrained in place. To fix this, we cluster together links as follows: colliding links are placed in the same cluster, and if two adjacent links in the rope are in two different clusters, these clusters are merged. This merging is repeated until the rope contains some number of disjoint clusters, with the rest of the links not part of any cluster. After calling FTL (during which the contact constraints keep nodes in place), we then rigidly move each entire cluster based upon the motion of its adjacent, unclustered links. So, as a knot goes from loose to tight, its segments both stick and slide on each other, but once it is tight, the whole knot moves as a unit. If several distinct knots are tied, they move as separate units. These clusters are conceptually similar to the "impact zones" used for cloth simulation in [Bridson et al. 2002; Provot 1997]: when cloth locally bunches together, friction tends to prevent relative motion, which is modeled by collecting the mesh nodes involved in neighboring collisions into impact zones treated as rigid bodies. Figures 6 and 7 shows screenshots as a trefoil knot is tied in the rope, followed by a figure-8 knot. Each knot behaves as a cluster once it is pulled tight.

Collisions between the rope and other objects are handled in almost the same way as self-collisions. Assume that a link has been detected to be at a distance $d < R$ from a fixed rigid object. We then shift the link away from the object by $R - d + \varepsilon$ (the object remaining fixed) and create a contact constraint that requires the link's endpoints not to move at the next call of FTL. Figure 8 shows screenshots during the tying of a knot around a fixed ring.

6 Results

For our experiments, we model a length of rope with 200 unit-length links. The rope's radius R is .5 units. We tested safety margins ε ranging from 0 to .5 units, and values between .1 to .2 provided the most natural "friction."

Any of four devices are used to control cursors on the screen with which to grasp and manipulate the rope. These devices are the mouse (which requires a key to toggle between *xy* and *xz* mo-

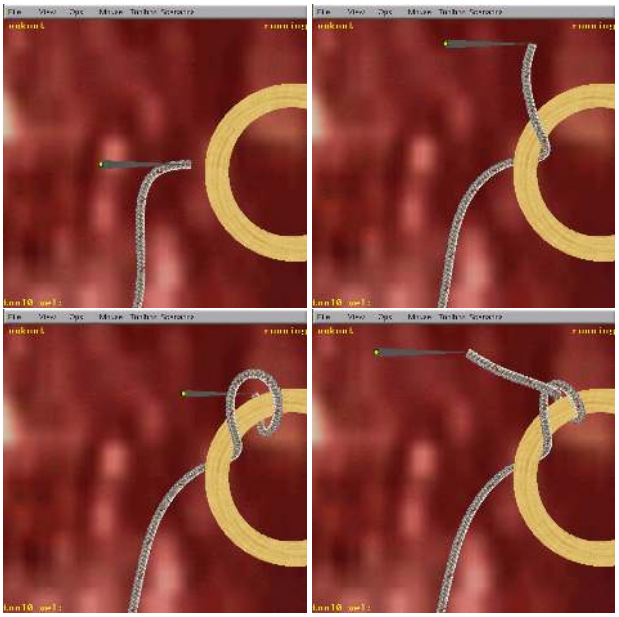

Figure 8: Tying a knot (the Münter hitch) around a ring.

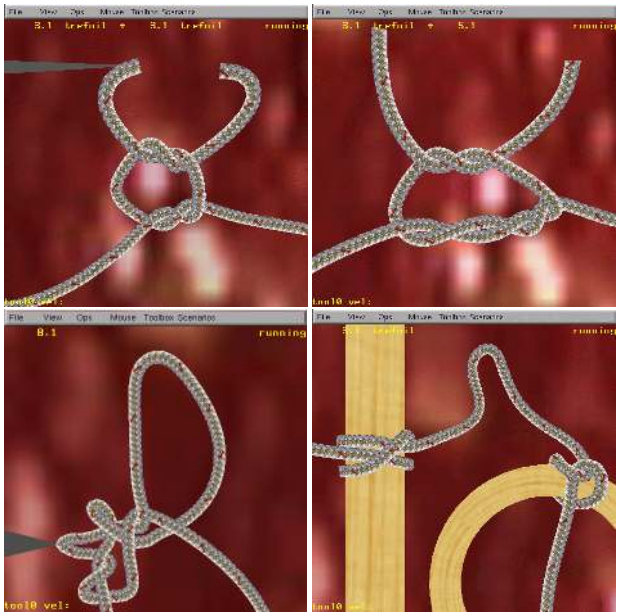

Figure 9: Top-down, left to right: Square knot (or composition of two trefoil knots), Surgeon's knot, Knot 8.1, Clove hitch around pole + trefoil around ring.

tion), surgical forceps equipped with electromagnetic trackers, a bimanual force-feeback laparoscope from Immersion, and a Phantom Desktop from Sensable Technologies. The latter two are only used as positioning devices, as the current version of the simulator does not render forces. Even without stereo glasses, the texture and the thickness of the rope provide fairly good depth cues to the user, so that moving the grasping tools to tie knots is relatively easy.

Users of the system successfully tied many simple and complicated knots, a few examples of which are shown in Figures 2 and 6-10. The rope motion is smooth and natural. In particular, frictional sliding of one piece of rope against another is quite realistic. The increase of friction in a knot as it is pulled tighter gives a feel that is very similar to a real rope. At no time was the rope able to pass through itself or other objects, despite swift motions of the tools.

The software runs on a Sun Ultra 60 with two 450-MHz processors. We use two threads to separate the simulation from the graphic rendering. At all times, the frame rate of the graphical display remained above 30 Hz (50 Hz was our average frame rate, and 76 Hz the maximum, limited by the monitor's refresh rate). The simulation process can complete about 140 simulation cycles per second for an untangled rope, 100 when the rope contains a tightly knotted trefoil, and 50 when containing a 7-crossing knot. Moving the grasping tools very fast does cause the rope to lag behind the tool, due to the threshold δ imposed on the displacement of any grasped node at each update. However, at 50 simulation cycles per second, the maximal velocity without any lag – 25 units/second – makes it possible to comfortably manipulate the rope.

7 Knot Identification

As the user manipulates the virtual rope and ties it into knots, it is helpful for him/her to know which knots have been tied. Certain knots are very similar, and one false move can result in tying the wrong one, or no knot at all (see Figure 11). In applications like surgical training, knot identification is also useful to assess the quality of an operation, since different knots have different properties. For example, the surgeon's knot is more difficult to tie than the similar square knot (see Figure 9), but the extra loop prevents

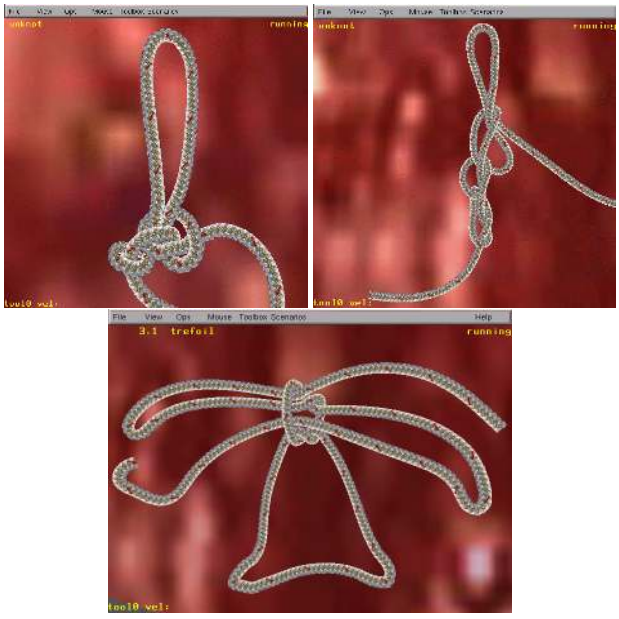

Figure 10: Top-down, left to right: Alpine butterfly knot used in climbing (actually isomorphic to the unknot), Daisy chain (to carry rope conveniently), Shoelace knot (isomorphic to the trefoil).

Figure 11: This five crossing knot and unknot differ only by reversing the over/under of the two bottom crossings.

slippage and loosening [Silverstein 1999].

There exist several texts providing a rigorous introduction to the topic of knot theory (e.g., [Adams 1994; Crowell and Fox 1963; Gilbert and Porter 1994]). A knot is topologically defined as a homeomorphic embedding of the unit circle in $\mathbf{R^3}$ [Crowell and Fox 1963], and two knots are said to be the same if there is an ambient isotopy between them. More simply put, if a piece of rope is manipulated (excluding cutting), and the two ends are glued together, the result is a knot [Adams 1994]. Any further manipulation of the rope (again excluding cutting) is an ambient isotopy which preserves that knot. The *unknot* is ambiently isotopic to $\mathbf{S^1}$.

Knots are often represented by projecting them onto a plane, and labeling the order of the crossings in the planar diagram. The labels also indicate which strand in each crossing passes above the other. A first level of classification for a knot is based on the minimum number of crossings in a planar diagram. It can be shown that any ambient isotopy can be represented as a sequence of three simple manipulations of the planar diagram, known as the Reidemeister moves [Adams 1994]. Two planar projections of the same knot may be quite different, but can always be reached from one another via these moves.

Determining whether a given knot is the unknot is in **NP** [Hass et al. 1997], and a general algorithm given by Haken in 1961 has never been implemented [Adams 1994], although more restrictive algorithms have been successful [Ladd and Kavraki 2002]. A list of all 10-crossing knots, given in 1899, was found to have an error as recently as 1974 [Adams 1994]. More recently, a list has been compiled of all knots with up to 16 crossings, and a list of 17-crossing knots is in progress [Hoste et al. 1998].

We have adapted the Knotscape software of Hoste and Thistlethwaite [Hoste et al. 1998] to identify knots which are currently tied in the rope. We compute the planar diagram of the current configuration of the rope by projecting the links of the rope onto an arbitrary plane P, adding an extra virtual link between the first and last nodes, so that our rope is a closed loop. We then find all the link crossings in P, keeping track of which link is on top in each crossing. These crossings are ordered and translated into Dowker-Thistlethwaite notation [Adams 1994]. The resulting list of crossings is sent to the Knotscape module, which returns the knot name(s). This software accepts as input a list of up to 50 crossings and outputs the identity of any knot of up to 16 crossings. This module is called once per second, in a separate thread of execution. Its output has no influence on the rope's motion, since our simulator computes this motion in a separate thread, without ever knowing which knots are being tied. The knot identities are displayed at the top of the screen while the rope is being manipulated (see the screenshots of Figures 6-11).

There are two possible degenerate cases, neither of which is a problem in practice:

1. The extra link we added to close the rope into a loop can create spurious crossings.

2. Multiple crossings can project to the same point in P, either exacly, or so close that floating-point roundoff causes the order of the crossings to be wrong.

In these cases, the knot identification can return a wrong answer, but as the rope continues to move, the answer is quickly updated with the next call to Knotscape, so an incorrect name is rarely displayed for long. The case of multiple crossings projecting to almost the same point is extremely rare, and while this can be theoretically solved by an arbitrarily small rotation of the projection plane [Crowell and Fox 1963], the practical solution is to simply wait for the next simulation update. The case of spurious crossings caused by the extra link is slightly more common, but can be prevented by an informed user, simply by pulling the two ends of the rope close to each other. Thus, the correct names of the knots in the rope are almost always displayed on screen.

Note that the computed identity of a knot is only based on the topology of the knot. However, friction allows creating complex knots that are important in practice, but are isomorphic to much simpler knots. A good example is the alpine butterfly knot used in climbing, shown in Figure 10(a). In this figure, it is correctly identified as the topological unknot, meaning that without friction, one would easily undo this knot by pulling its two ends apart. The shoelace knot in Figure 10(c) is correctly recognized as the much simpler trefoil knot. To our knowledge, there exists no robust technique to identify more precisely such non-topological knots, whose stability derives from friction at contacts.

8 Conclusion

This paper describes and demonstrates a computational model of rope motion that enables fairly realistic interactive simulations of complex knot-tying operations at real-time rates. The key components of this model are the collision detection and management modules. Our method of resolving (self-)collisions prevents the rope from passing through itself or other thin objects. It also leads to micro-displacements of rope nodes and the establishement of contact constraints, which, together, create a very effective and believable model of friction allowing the rope to be tied into knots in a natural way. We do not think that any prior system was able to simulate knot tying in real-time. An additional module, based on pre-existing software, identifies the topology of the knots tied.

We are close to extending this work to fully simulate knot tying in surgical suturing. Actually, we have already demonstrated suture interactions with deformable tissue, but, so far, we have only tied knots around rigid objects. At this stage, we do not expect that tying knots around deformable tissue will raise new major difficulties, though it will require enough computational power to simulate both the suture and deformable tissue concurrently. Providing force feedback will be another useful addition to the integrated simulator.

Future work should seek to better assess the validity of our model of friction. For example, there exist knots used in rock climbing that are known to be very strong in some variants (due to friction) and weak in other variants (due to less friction). In particular, a prusik knot sticks if tied around a bigger piece of rope, but slides if tied around a rope of smaller radius. Similarly, the strength of a Münter hitch around another object depends on the relative directions along which the two ends are pulled apart. Experimental tests with our simulator on such knots could result in making our model of friction more accurate. This might require modeling the fact that the cross-section of a rope is slightly elastic, which, in the case of the prusik knot, helps the knot to "bite" on the strand of rope around which it is tied. Rope twisting is another aspect of the rope mechanics that might have to be modeled to improve simulation realism.

Finally, an interesting problem for future research is the identification of complex knots that are topogically equivalent to simpler knots, but derive their stability or strength from the friction at the contacts they create.

Acknowledgements: This work was supported by NSF (ACI-0205671) and NASA (NAS-NCC2-1010). Special thanks to Fabian Schwarzer and Dan Brown for valuable discussions. We also thank Jim Hoste and Morwen Thistlethwaite for permission to adapt and use their Knotscape software.

References

ADAMS, C. C. 1994. *The Knot Book*. W. H. Freeman and Co.

BARAFF, D., AND WITKIN, A. 1998. Large steps in cloth simulation. In *Proc. of ACM SIGGRAPH 1998*, ACM Press / ACM SIGGRAPH, New York, E. Fiume, Ed., Comp. Graphics Proc., Annual Conference Series, ACM, 43–54.

BRIDSON, R., FEDKIW, R., AND ANDERSON, J. 2002. Robust treatment of collisions, contact and friction for cloth animation. *ACM Transactions on Graphics 21*, 3, 594–603.

BROWN, J., SORKIN, S., LATOMBE, J.-C., MONTGOMERY, K., AND STEPHANIDES, M. 2002. Algorithmic tools for real-time microsurgery simulation. *Medical Image Analysis 6*, 3, 289–300.

CROWELL, R. H., AND FOX, R. H. 1963. *Introduction to Knot Theory*. Blaisdell Publishing Company.

DAWSON, S. L. 2002. A critical approach to medical simulation. *Bulletin of the American College of Surgeons 87*, 11, 12–18.

DELINGETTE, H. 1998. Towards realistic soft tissue modeling in medical simulation. In *Proceedings of the IEEE : Special Issue on Surgery Simulation*, 512–523.

ERDMANN, M. 1994. On a representation of friction in configuration space. *Int'l J. of Robotics Research 13*, 3, 240–271.

GILBERT, N. D., AND PORTER, T. 1994. *Knots and Surfaces*. Oxford University Press.

GOTTSCHALK, S., LIN, M. C., AND MANOCHA, D. 1996. OBBTree: A hierarchical structure for rapid interference detection. In *Proc. of ACM SIGGRAPH 1996*, ACM Press / ACM SIGGRAPH, New York, E. Fiume, Ed., Comp. Graphics Proc., Annual Conference Series, ACM, 171–180.

GRAYDON, D., Ed. 1992. *Mountaineering: The Freedom of the Hills*. The Mountaineers.

GUIBAS, L., NGUYEN, A., RUSSEL, D., AND ZHANG, L. 2002. Deforming necklaces. In *Proceedings of ACM Symposium on Computational Geometry*, 33–42.

HALPERIN, D., AND OVERMARS, M. H. 1998. Spheres, molecules, and hidden surface removal. *Computational Geometry: Theory and Applications 11*, 2, 83–102.

HASS, J., LAGARIAS, J. C., AND PIPPENGER, N. 1997. The computational complexity of knot and link problems. In *IEEE Symposium on Foundations of Computer Science*, 172–181.

HOSTE, J., THISTLETHWAITE, M., AND WEEKS, J. 1998. The first 1,701,936 knots. *Mathematical Intelligencer 20*, 4, 33–48.

HOUSE, D. H., AND BREEN, D. E., Eds. 2000. *Cloth Modeling and Animation*. A.K. Peters, Ltd., Natick, MA.

JOUKHADAR, A., SCHEUER, A., AND LAUGIER, C. 1999. Fast contact detection between moving deformable polyhedra in motion. In *Proc. of IEEE Int'l Conf. on Intelligent Robots and Systems*, 1810–1815.

KÜHNAPFEL, U. G., ÇAKMAK, H. K., AND MAASS, H. 2000. Endoscopic surgery training using virtual reality and deformable tissue simulation. *Computers & Graphics 24*, 671–682.

KUSNER, R. B., AND SULLIVAN, J. M. 1998. Möbius-invariant knot energies. In *Ideal Knots*, A. Stasiak, V. Katritch, and L. H. Kauffman, Eds. World Scientific, 315–352.

LADD, A. M., AND KAVRAKI, L. E. 2002. Motion planning for knot untangling. In *Preprints of 5th International Workshop on Algorithmic Foundations of Robotics (WAFR 2002)*, 6–22.

LARSSON, T., AND AKENINE-MÖLLER, T. 2001. Collision detection for continuously deforming bodies. In *Proceedings of Eurographics 2001*, 325–333.

LARSSON, A. 2001. Intracorporeal suturing and knot tying in surgical simulation. In *Medicine Meets Virtual Reality*, 266–271.

LIN, M. C., AND GOTTSCHALK, S. 1998. Collision detection between geometric models: A survey. In *Proceedings of IMA Conference on Mathematics of Surfaces*, vol. 1, 602–608.

LOTAN, I., SCHWARZER, F., HALPERIN, D., AND LATOMBE, J.-C. 2002. Efficient maintenance and self-collision testing for kinematic chains. In *Proceedings of ACM Symposium on Computational Geometry*, 43–52.

MIRTICH, B., AND CANNY, J. 1995. Impulse-based simulation of rigid bodies. In *Proceedings of Symposium on Interactive 3D Graphics*, 181–188.

PAWSON, D. 1998. *The Handbook of Knots*. DK Publishing, Inc.

PHILLIPS, J., LADD, A. M., AND KAVRAKI, L. E. 2002. Simulated knot tying. In *Proceedings of the IEEE International Conference on Robotics and Automation*.

PROVOT, X. 1995. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface*, 147–154.

PROVOT, X. 1997. Collision and self-collision handling in cloth model dedicated to design garment. In *Graphics Interface*, 177–189.

QUINLAN, S. 1994. Efficient distance computation between non-convex objects. In *Proceedings of the IEEE International Conference On Robotics and Automation*, 3324–3329.

SILVERSTEIN, L. H. 1999. *Principles of Dental Suturing: The Complete Guide to Surgical Closure*. Montage Media Corp.

SMITH, A., KITAMURA, Y., TAKEMURA, H., AND KISHINO, F. 1995. A simple and efficient method for accurate collision detection among deformable polyhedral objects in arbitrary motion. In *Proc. of the IEEE Virtual Reality Annual Int'l Symp.*, 136–145.

VAN DEN BERGEN, G. 1997. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools 2*, 4, 1–13.

VOLINO, P., AND MAGNENAT-THALMANN, N. 1995. Collision and self-collision detection: Efficient and robust solutions for highly deformable surfaces. In *Eurographics Workshop on Animation and Simulation*, 55–65.