Simulated Knot Tying

Jeff Phillips Andrew Ladd Lydia E. Kavraki
Department of Computer Science
Rice University
Houston, TX, 77005
`jeffp|aladd|kavraki@cs.rice.edu`

Abstract

Applications such as suturing in medical simulations require the modeling of knot tying in physically realistic rope. The paper describes the design and implementation of such a system. Our model uses a spline of linear springs, adaptive subdivision and a dynamics simulation. Collisions are discrete event simulated and follow the impulse model. Although some care must be taken to maintain stable knots, we demonstrate our simple model is sufficient for this task. In particular, we do not use friction or explicit constraints to maintain the knot. As examples, we tie an overhand knot and a reef knot.

1 Introduction

It is often necessary to model deformable materials for graphics and simulation applications. Examples include cloth modeling [1], soft tissue simulation [10] and virtual surgery simulators such as [13]. The simulation of deformation on a computer has led to development and use of many approximate models and numerical integration schemes. Some models use meshes of linear or non-linear springs [14] and [15]. Other models use Boundary Element Methods `BEM` [7] and Finite Element Methods `FEM` [11]. These methods are at various places in speed efficiency, realism and stability trade off.

Surgical simulation makes new demands of the physically based computer modeling of deformables. Real time models sufficiently realistic to train surgeons are needed. In microsurgery simulation [3], in laproscopic surgery simulation [2] and others, realistic real-time simulation of rope is identified as a task. In a virtual surgery world, we would like to be able to tie knots to simulate suturing. To do this we must provide a robust dynamics model and integration as well as a robust solution to the collision detection problem for deformable self-intersecting shapes.

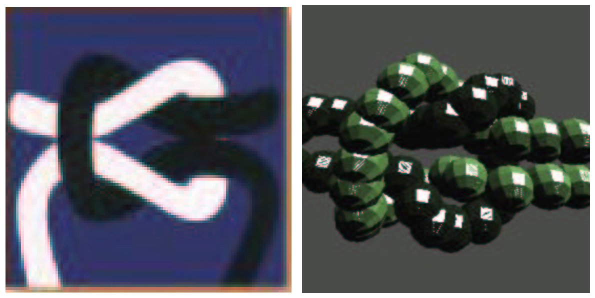

Figure 1: Illustration of a standard reef knot and our reef knot

This paper describes a method for knot tying with simulated rope. The rope is a physically based dynamics simulation [15]. Our experiments consist of pulling a loosely knotted rope configuration tight and witnessing that the knot was maintained.

Our rope is a spline of linear springs [17]. The topology of the rope varies adaptively to maintain the surface of the rope. Self-intersections in the rope are discrete event simulated using the impulse model of collision [8] in the interpolation step of the physical simulation. During the adaptive reparametrization of the rope, some care must be taken to maintain conservation of energy, mass and momentum. This also must be done during the numerical integration of the resulting differential equations. The errors in our solution are sufficiently small that they can be balanced with a reasonable amount of damping [11]. These errors come from modeling approximations, geometric approximations, and dynamics integration.

There are several interdependent difficulties that must be addressed by our solution or they will cause instabilities which will lead to system divergence. Divergence manifests itself either as the knot slipping or as degenerate oscillatory behavior. In the area around a tightening knot, the high curvature of the rope, multiple collisions and opposing internal forces create a potentially unstable situation. The dynamics simulation must be able to cope adaptively as these conditions vary at some parts of rope and the collision surface must be approximately continuous over time. Otherwise numerical errors will aggregate rapidly. We demonstrate that a simple but carefully chosen model can be used successfully in simulating knot tying.

2 Model

Our rope model is constructed of a spline of linear springs with control points at the end points of the springs. Each spring knows its tension, its original length, and the control points on either end of it. Each control point is shared by two springs, except for the terminal control points which only attach to one spring. Using the tension and the original length, each spring can calculate the force acting on a control point using Hooke's Law

$$F = -k(d - d_o).$$

Each control point knows its mass, its velocity, and the springs on either side. Any external energy accumulated in the system is stored in the control points, either in the form of momentum or from an external force such as gravity or manipulation. Each control point also represents the mass of half of each rope segment on either side of it. To describe the volume of the rope, we tile it with spheres of fixed radius centered at the control points.

For realistic collision detection, we need the surface of the rope to be sufficiently consistent. We also need to be more expressive when the rope is stretched tightly. We address these issues by maintaining the spacing of control points online. In particular

1. We ensure that small oscillations of a spring close to equilibrium do not lead to too many oscillations in the structure of the rope. (Section 2.2)

2. Conservation of energy, mass and momentum is maintained as much as possible during insertion and removal of control points. (Section 2.3)

To simulate the rope we calculate the trajectories of all of the control points individually every time step. From these trajectories we can predict collision events over the time step, which we can handle individually at the appropriate simulation times.

2.1 Main Loop

Algorithm 1 Toplevel loop

```
1: loop
2:    recompute rope topology
3:    predict control point trajectories
4:    generate collision events and queue
5:    while queue is not empty do
6:       simulate collision
7:       dequeue stale events
8:       queue secondary events
9:    end while
10: end loop
```

Algorithm 1 describes the main loop of our simulation. Passing through the loop once simulates a fixed block of time. We first examine the rope and adaptively reparametrize if necessary. Afterward, the equations of motion are integrated and trajectories for the time step are set. The trajectories of the control points are then examined and collisions are scheduled in the discrete event simulation queue. During the discrete event simulation, the events are dequeued and simulated in order. During a collision, impulse occurs and the control points' trajectories are adjusted. Stale collision events are dequeued and secondary events, resulting from the changes in trajectory, are queued.

2.2 Adaptive Subdivision

Structural oscillation during simulation is avoided by employing a hysteresis threshold [4] using $\alpha < \beta$ as the low and high constants respectively. When the distance between adjacent control points is larger than β, subdivision [16] occurs, and when the distance between the end points of an adjacent triple of control points is less than α, the center point of the triple is contracted. The precise implementation of the rope topology recomputation pass is given in Algorithm 2.

Algorithm 2 Recompute rope topology

```
1: for $i = 1, ..., n-1$ do
2:    if DIST($p[i], p[i+1]$) > $\beta$ then
3:       INSERT($i$)
4:       continue
5:    else if DIST($p[i], p[i+2]$) < $\alpha$ then
6:       DELETE($i+1$)
7:       $i = i + 1$
8:       continue
9:    end if
10: end for
```

In Algorithm 2, the array p contains the positions of control points, DIST is the metric, INSERT(i) inserts a control point just after control point i and DELETE(i) contracts the ith control point.

We chose to use a hysteresis threshold instead of a single threshold value to provide a range in which an intermediate control point can exist between control points. Small oscillations occur naturally in the simulation when settling to an equilibrium. Each change to the shape of the surface causes small numerical errors in the dynamics and the efficiency of the collision detection. Repetitive changes in a small time span can aggregate error and lead to system divergence.

2.3 Insertion and Deletion Operators

The stability of the system is dependent on conservative equations describing adaptive changes to the system. During insertion and removal of control points we need to conserve momentum.

The springs hold the potential energy of the system. The control points hold the mass and the kinetic energy of the system. Since a control point represents the mass of two separate pieces of the rope, half the segment on the left and half the segment on the right, it needs to distinguish between the two. Each control point is given a left mass and a right mass. It also has an associated left and right velocity corresponding to each mass.

Whenever a new control point is added, it receives half of its right neighbor's left mass and half of its left neighbor's right mass. It stores them as right and left masses respectively. The masses maintain their velocities to conserve momentum. The new control point is added along the line between the two old control points based on a weighted average of the left mass of the right old control point and the right mass of the left old control point. Notice that the new control point is placed at the center of mass of the old spring. The original distances of the springs are divided in the same ratio. In Figure 2, Control Point B is inserted between Control Points A and C.

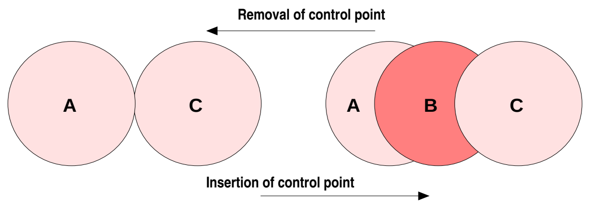

Figure 2: Insertion and removal of control points

Removing a control point is the inverse of adding a control point. The merged velocities may not be equal, so they are recalculated with a mass-weighted average.

Removal of a point can cause numerical error. The new current length of the spring is almost surely shorter than the sum of the old lengths, but this added energy will quickly dissipate as the control points are pushed out toward an equilibrium position. Removal can change the center of mass and the internal energy of the system. If a control point with more left mass than right mass is closer to its right neighbor than its left neighbor and is removed, then the center of mass will shift toward the left mass: more mass is being transferred a longer distance left than is being transferred to the right. Additionally, internal energy will be altered any time a control point is removed that is not on the line segment joining its two neighbors. This will cause the current distance of the new spring to be less than the current distance of the sum of the two old springs. We reduce the effect of numerical error by adding damping into the system. Damping has the effect of stabilizing any oscillations around the equilibrium energy position and of smoothing discontinuities in the energy distribution. With the appropriate transfer of mass and energy as well as damping, we can insert and remove control points with sufficient stability.

3 Physical Dynamics

We opt for a dynamics simulation since self-intersections cause changes in the boundary conditions of the physical equations of energy. To simplify this task, we assume the forces are constant over a small time interval and integrate the predicted change in position of the control points viewed as a particle field. We use Stoermer's Rule [12], which computes change in position directly from the second derivative rather than via a double integration. Our experience suggests that this method is more stable than methods such as Euler's method. Algorithm 3 describes the operation of this integration scheme for a single variable, x, which evolves over finite time steps (Δt) with second derivative $\ddot{x} = f(t, x)$, a function of time and the variable. Notice the change in x is integrated directly from the second derivative and the first derivative is integrated independently. This is valid for conservative second order differential equations.

We adapted Stoermer's rule to handle the notion of left and right mass and to be used in a dynamics simulation. Left and right sides of the control point are integrated separately and then the total change in position is accumulated. The change is then simulated with interpolation over the time step.

Algorithm 3 Stoermer's rule

```
1: $\Delta x = \Delta t[\dot{x} + \frac{1}{2}f(t,x)]$
2: repeat
3:     $x = x + \Delta x$
4:     $t = t + \Delta t$
5:     $\dot{x} = \frac{\Delta x}{\Delta t} + \frac{\Delta t}{2}f(t,x)$
6:     $\Delta x = \Delta x + (\Delta t)^2 f(t,x)$
7: until we are done simulating
```

If a change in boundary conditions occurs over the course of the interpolation, for example, a collision causes the velocity to change, then the appropriate variables are updated and then interpolation continues. After such a change, the loop in Algorithm 3 is reset for the affected control points and restarted with the new values.

3.1 Collision Detection

Our collision detection consists of checking for collisions between the spheres centered on the control points, an easy calculation. There are no gaps between spheres because the threshold for adding control points is twice the radius. When a gap is about to form, a new sphere is added as in Figure 3, ensuring that a collision between the ropes is not missed by a part of the rope passing through a gap in a different part of the rope.

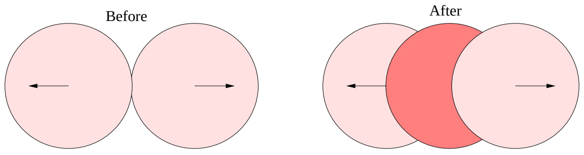

Figure 3: Insertion of a control point

We want to handle every event at the time it occurs, but we wish to avoid the expense and difficulty of advancing the dynamics simulation every time we handle an event. To avoid this problem, we run a discrete event simulator during the interpolation step between dynamics simulation samples. We calculate when a collision will occur within a time step by comparing the paths of control points over a time step.

Adjacent control points will be in collision with each other. In fact, control points which are each other's neighbor's neighbor can also be in contact. These collisions are simply not checked for.

When two control points collide, they affect each other's trajectories, disrupting the momentum of the system and changing the boundary conditions. The points must not pass through each other. We choose to use a simple impulse model [9]. This ignores any friction forces and assumes elastic collision. Our algorithm also ensures that the control points' new velocities repel them from each other as in Figure 4.

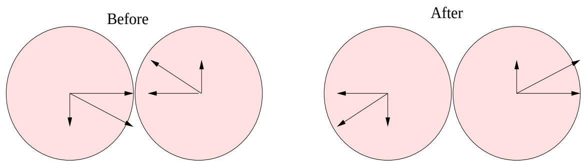

Figure 4: Collision of control points

An error can occur when a new control point is added and its sphere is already in contact with another sphere. Simply reversing the control points' velocities will not ensure that they will come out of collision. A naive implementation of the collision dynamics will generate instantaneous collisions repeatedly, causing a loop. To overcome this issue we force them apart, as shown in Figure 5.

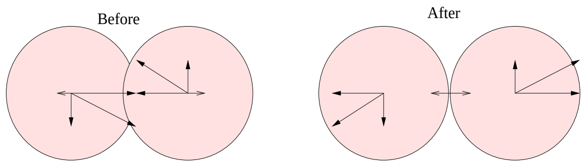

Figure 5: Overlapping control points

We also want to maintain locality in the effect of the collision. When the velocity of a control point changes, the integration in Algorithm 3 is reseeded which increases the numerical error during the next time step. In our model, each collision causes the velocity of only two control points to change.

4 Experiments

In order to verify our rope model, we ran simulations in our OpenGL visualizer. A sphere is drawn at each control point to visually approximate the volume of the rope. A frame is redrawn after each time step. Our simulations represent systems reacting to physical forces. To verify the correctness of our model, we rely on the visual feedback from the simulator. Errors in stability are apparent because if they are an issue they amplify and become obvious; if they do not amplify they are dissipated by damping and are not a problem. The images below are snapshots from such simulations. All experiments are run on a Pentium III 900MHz workstation.

4.1 Simulation of Gravity

We wish to verify the correctness of the stability and realism of our system when in a simple simulation with physical forces. We interpolated the points between two control points to form a horizontal line. We fix the end points and apply a universal downward force on the rope, simulating gravity. As shown in Figure 6, the center of the rope falls, points are inserted adaptively, the rope rebounds, oscillates briefly, and reaches an equilibrium.

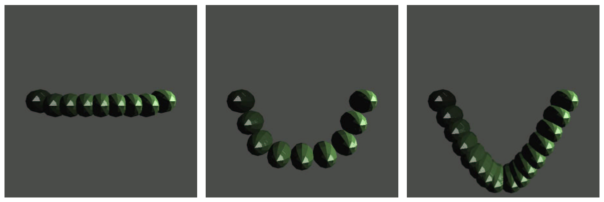

Figure 6: Effects of gravity

4.2 Overhand Knot

We wish to demonstrate that our model can tie a knot, so we begin with a simple overhand knot. Our initial configuration consists of a loose overhand knot. The ends of the rope are given forces to repel them from the center of the knot. This is done by applying an external force on the section of rope at the end being pulled. The knot tightens without slipping, eventually reaches an equilibrium, and stabilizes. The force on the end portion of the rope causes that portion to stretch considerably, but minimal stretching occurs around the knotted part. The rope stretches to 118% of its original length throughout the simulation. The screenshots in Figure 7 are taken at 40 seconds and then again at 3 minutes and 30 seconds.

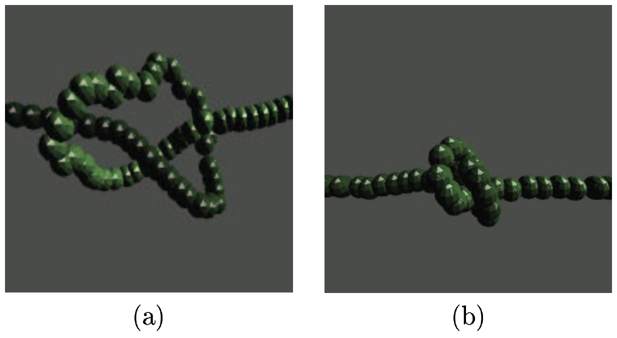

Figure 7: (a) Loose configuration close to beginning of simulation. (b) Tied overhand knot.

4.3 Reef Knot

A reef knot is often used in surgical suturing and provides another example of a knot for our model. We simulate it by initially forming a loose configuration of the knot with two pieces of rope. When the ends are pulled in the same manner as with the overhand knot, the knot tightens and holds. The screenshots in Figure 8 are taken at 3 minutes and 5 minutes into the simulation and the rope stretches to 116% of its original length, but mainly around the ends where the force is directly being applied.

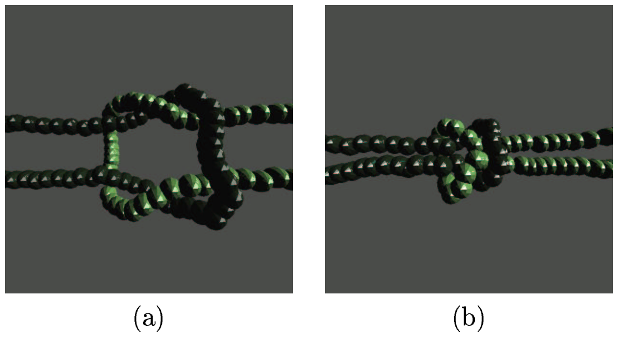

Figure 8: (a) Tightening rope in the middle of the simulation. (b) Tied reef knot.

4.4 Discussion

Our simulation decouples the particle simulation from momentum changing events: insertion and removal of control points and collisions. This allows us to find and handle every collision at the appropriate time without slowing down the broader simulation of particles.

We can automatically insert and remove control points to maintain the consistency of the collision surface. Our model presents a way of reconciling the adaptive subdivision with conservation of energy and mass. Our experiments have shown that by varying parameters for damping, tension, and external forces we can stabilize the model. Generally, as tension increases, the amount of external force and damping also needs to increase; otherwise the system becomes volatile.

We have chosen a conservative integration scheme rather than a double integration scheme or predictor. This avoids expensive, multiple force recomputations that cannot be reliably cached because of the dynamic nature of the system. Also, we have observed that the conservative integration scheme we use is more numerically stable than common double integration schemes such as Improved Euler's method and Runge-Kutta order 4.

Our model does not incorporate explicit frictional forces, it does not factor inelasticity into collisions, and it does not maintain a predetermined structure representing a knot, yet the energy values reduce to the correct position. We use only an impulse model to model the collision dynamics. Adjacent sliding rope may snag around abutting notches between spheres, producing effects which are qualitatively like friction. As the knots become tighter, the control points have more frequent but weaker collisions. Our simple approach is able to maintain and recognize convergence of knots.

5 Future Work

The collision detection is the current speed bottleneck in the code. It can potentially be optimized through the use of kinetic data structures [5]. The algorithm presented in [6] is close to having the right operations. We believe that more complex data structuring would allow us to consider fewer potential collisions.

Investigation into using more realistic models to model torsion, stretching, bending and squeezing of the rope, can improve the realism of our model.

Our collision geometry is somewhat simplistic. We would like to incorporate a more continuous collision surface into the simulation. Also, we would like to simulate more complex environments.

Acknowledgments

Jeff Phillips is partially supported by a Brown Undergraduate Research award and by the REU portion of NSF IRI-970228. Andrew Ladd is partially supported by FCAR 70577, by NSF IRI-970228 and by a Whitaker Biomedical Engineering Grant. Work on this paper by Lydia Kavraki was partially supported by NSF CAREER Award IRI-970228, a Whitaker Biomedical Engineering Grant, an ATP Award and a Sloan Fellowship.

References

[1] D. Baraff and A. Witkin. Large steps in cloth simulation. In *Proc. Computer Graphics*, pages 43–54, 1998.

[2] C. Basdogan, C.-H. Ho, and M. Srinivasan. Virtual environments in medical training: graphic and haptic simulation of laparoscope common bile duct exploration. IEEE/ASME Transactions on Mechatronics, September 2001.

[3] J. Brown, K. Montgomery, J.-C. Latombe, and M. Stephanides. A microsurgery simulation system. In *Medical Image Computing and Computer-Assisted Interventions (MICCAI)*, Utrecht, The Netherlands, October 2001.

[4] J. F. Canny. Finding edges and lines in images. *Techincal Report AI-TR-720, Massachusetts Institute of Technology Artificial Intellegence Laboratory*, 1983.

[5] L. Guibas. Kinetic data structures: A state of the art report. In P. Agarwal, L. Kavraki, and M. Mason, editors, *Robotics: The Algorithmic Perspective*. A. K. Peters, 1998.

[6] L. Guibas, F. Xie, and L. Zhang. Kinetic collision detection : Algortithms and experiments. In *IEEE Int. Conf. Robot. & Autom.*, pages 2903–2910, Seoul, Korea, May 2001.

[7] D. James and D. Pai. Artdefo : Accurate real time deformable objects. *Computer Graphics (SIGGRAPH'90)*, pages 65–72, August 1999.

[8] B. Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems*. PhD thesis, University of California, Berkeley, 1996.

[9] B. Mirtich and J. Canny. Impulse-based simulation of rigid bodies. In *Proceedings of ACM Interactive 3D Graphics Conference*, 1995.

[10] V. Ng-Thow-Hing, A. Agur, K. Ball, E. Fiume, and N. McKee. Shape reconstruction and subsequent deformation of soleus muscle models using b-spline solid primitives. In *Proc. SPIE 3254 Laser-Tissue Interaction IX*, pages 423–434, 1998.

[11] G. Picinbono, H. Delingette, and N. Ayache. Nonlinear and anisotropic elastic soft tissue models for medical simulation. In *IEEE Int. Conf. Robot. & Autom.*, 2001.

[12] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery. *Numerical Recipes in C: The Art of Scientific Computing*. Cambridge University Press, New York, NY, second edition, 1992.

[13] J. Rotnes, J. Kaasa, G. Westgaard, and al. Digital trainer developed for robotic assisted cardiac surgery. In *Medicine Meets Virtual Reality (MMVR)*, Sept. 2001.

[14] D. Terzopoulos, J. Platt, A. Barr, and K. Fleischer. Elastically deformable models. *Computer Graphics (SIGGRAPH'87)*, 21(4):205–214, 1987.

[15] D. Terzopoulos and A. Witkin. Physically based models with rigid and deformable components. *IEEE Computer Graphics and Applications*, pages 41–51, November 1988.

[16] J. Warren and H. Weimer. *Subdivision Methods for Geometric Design: A constructive Approach*. Morgan Kaufmann, 2002.

[17] A. Witkin and W. Welch. Fast animation and control of non-rigid structures. *Computer Graphics (SIGGRAPH'90)*, pages 243–252, 1990.