[published in: IEEE Computer Graphics and Applications 17(3) , May-June 1997, pp. 31-39.]

Faking Dynamics of Ropes and Springs

Ronen Barzel
Pixar Animation Studios
Richmond, CA

We describe a simple method for modeling flexible linear bodies such as ropes and springs, using no dynamic simulation, procedural animation, nor constraint methods—animators create motion by adjusting the shape of models over time using traditional keyframe methods. The approach taken is to provide a default natural rest shape, and to provide controls that perform gross modification and wave-shaped deformation of the rest shape. The resulting models provide animators with intuitive control and the means to interactively create motion that is both visually plausible and tailored to the frame-to-frame needs of particular animations. This technique has been used successfully in animation production over the past few years, in particular for various ropes and the "Slinky Dog" in the movie Toy Story.

1 Introduction

This paper presents a method to model flexible linear bodies—ropes, cords, springs, and the like—for animation production. The resulting models can easily be animated in a manner that convincingly "fakes" dynamic behavior.

It is perhaps surprising that for character animation, we often prefer convincingly "fake" dynamics to purely "real" dynamics. Although the motion needs to be plausible, high-quality animation uses caricature methods such as anticipation and exaggeration in a non-physically-motivated manner to convey story, mood, and character, and to direct audience's attention.[1] The amount and type of caricature varies—from object to object, scene to scene, pose to pose, and frame to frame—to meet the needs of the moment.

In a production environment, it is essential that animators can easily adjust and edit pose and timing with per-frame accuracy, to respond to the needs and feedback of the director or client. Keyframe animation systems are well-suited to these tasks.

For these reasons, we want to define flexible-body models to be animated using a keyframe system. Our goals are to manage the many degrees of freedom that a flexible body has, to create models that easily lend themselves to convincing motion, and at the same time to provide the necessary direct control and ease of use.

We meet these goals by defining the shape of a flexible body via a few layered deformations; these deformations are chosen so as to create the type of shapes that a dynamically-moving body would adopt. By varying the parameters of the deformations over time, the animator can create convincing motion.

We cannot overemphasize that the keyframe approach requires talented human animators. Our models do not automatically generate animation, they simply provide shape controls for an animator to work with; an unskilled animator (such as this author) can certainly make unconvincing animation. The power of our approach is in its simplicity and its applicability to a specific task.

What about "real" dynamics?

The computer graphics research community (including this author) has done much work in computing realistic motion via physically-based modeling and dynamic simulation. Control over the motion is often provided via constraint methods.[2, 3]

As discussed above, for our character animation application, we require plausible-but-caricatured motion rather than purely physical. and we need quick and direct control. As such, dynamic simulation has not proven appropriate for us: the control methods don't directly lend themselves to the non-physical motion and quick per-pose/per-frame editing that we need, and interaction speed is an issue.

In circumstances where the bodies have less interaction with animated characters, dynamic simulation can be a valuable tool. For example, in *Toy Story*, although almost all the flexible linear-body animation used the kinematic methods of this paper, there was one shot—of a rope thrown from a balcony—that was computed dynamically.

Related work

Pentland and Williams[4] use vibration-mode eigenvalue computation to animate flexible 3D bodies, pointing out the reduction in degrees of freedom that modal analysis provides. Although their focus is on physical characteristics of the bodies, the modal approach is similar in spirit to our work.

Chadwick *et al.*[5] use a layered approach to flexible-body animation, with dynamic vibration computed on top of kinematic skeletal deformation. Terzopoulos and Witkin[6] use dynamic simulation for both a rigid model and its flexible deformations. Our approach is perhaps conceptually similar to these, except that in our work both the basic shape and vibration layer are produced by kinematic deformation.

This author and Hughes and Wood[7] have investigated simulating motion that is "visually plausible" but not necessarily physically accurate. That approach is speculative, and has not seen use in production.

The models in this paper are built using the C-like modeling language of Reeves *et al.*[8] Their keyframe animation system is used as well; it provides traditional spline-based control of the animatable parameters of the models.

2 Overview

A straightforward way to model a rope is to simply provide a collection of movable control vertices (CV's) that define the curve of the body. However, this method is tedious to work with, and hard to make move convincingly.† Having independent control over each CV is overkill, since in practice the CV's do not move independently but rather stay coupled to each other.

To make the animator's job feasible, we capture the physical behavior of the body into relatively few, higher-level controls. In a manner reminiscent of modal analysis (Fig. 1), we observe that the shape of a flexible body can be described at any instant by a gross deformation of a rest shape, perturbed by a wave-shaped deformation.

Our general method for creating models is thus quite simple:

- Define a natural rest shape
- Perform gross-shape deformation
- Perform wave-shape deformation

†The electric cord in the animated short *Luxo Jr.* was modeled using CV's vertices in this manner; it was "by far the hardest and most time consuming"[9] part to animate.

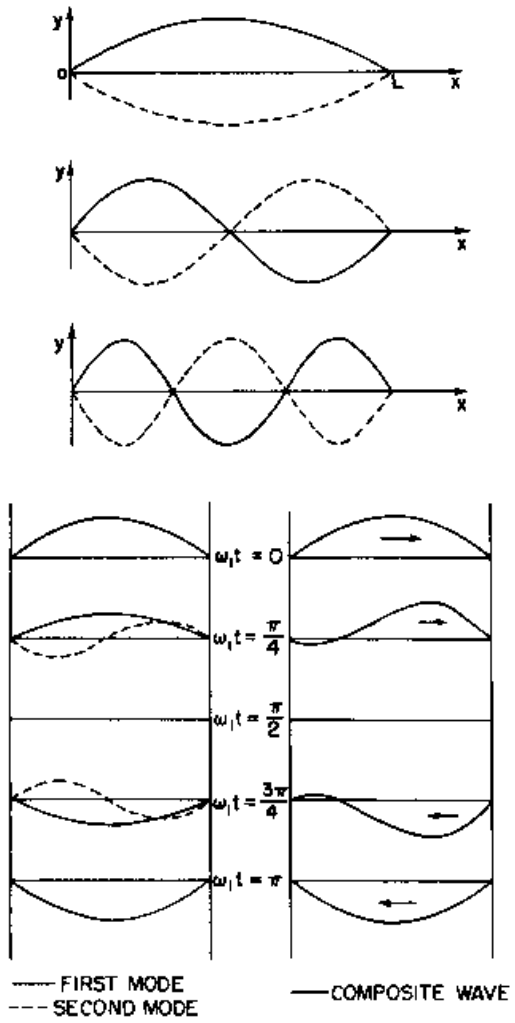


Figure 1: *Top:* the first three modes of a vibrating string. *Bottom:* two modes combine to give a traveling wave. (Figures from Feynman *et al.*[10])

Not needing physically-exact behavior, our gross- and wave-shape deformations are simple operations, chosen for ease of use. A final clamping step can be introduced to keep models from interpenetrating the floor (or other easily computed geometry).

3 Basic Models

This section describes four useful models that illustrate the modular layered modeling method. Sec. 4 discusses how to use and extend these for more complex models.

For each model, we will describe the rest shape, gross shape controls, and wave deformation, followed by a short description of its implementation. Further details are contained in appendix A; in particular, A.1 describes how we represent curves.

Figure 2: Suspended rope. *Left to right:* rest shapes for various end positions; swing side to side; sway back and forth.

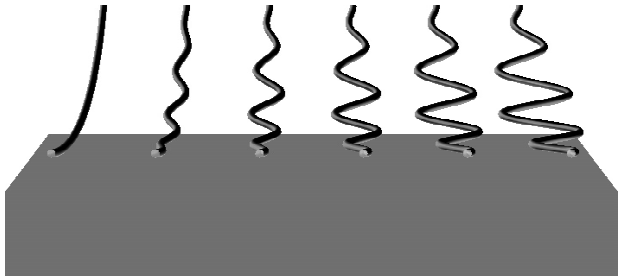


Figure 3: Wave deformations with varying magnitude.

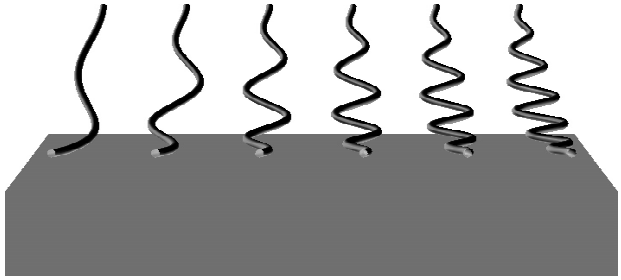


Figure 4: Wave deformations with varying frequency.

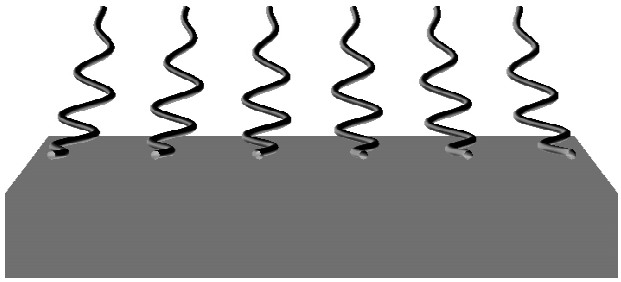


Figure 5: Wave deformations with varying phase.

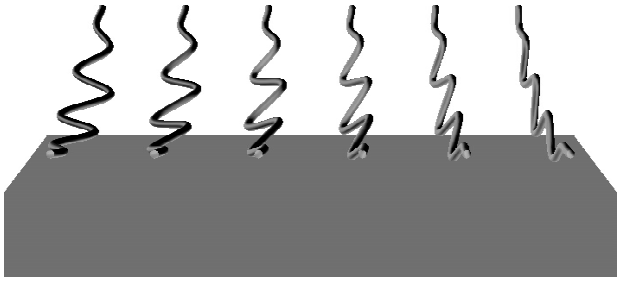


Figure 6: Wave deformations with varying azimuth.

```
func SuspendedRope(end1,end2, length)
parms:  swing, sway
        waves ≡ n× { mag, freq, phase, azm }
; rest and gross shapes
up = rotate(swing, X)*rotate(sway, Y)*Z
rope = catenary(end1, end2, length, up)
; wave deformation
envelope = sin(0..180)
rope = rope + envelope*wave(rope, waves)
return rope
```

Figure 7: Pseudocode for the suspended rope of Figs. 2–6. The *parms* are parameters available to the animators.

3.1 Suspended Rope

A common model is a rope suspended from its two ends. The ends may be completely fixed in space, animated manually, or attached to some other objects, but the ends are not affected by shape adjustments and wave deformations.

Rest shape. The rest shape is a catenary, as would be taken naturally by a rope suspended in gravity. The rest shape is determined by length of the rope and the positions of the ends (Fig. 2, left).

Gross shape. The rope has two gross shape parameters: *swing* from side to side and *sway* back and forth (Fig. 2, center and right).

Wave deformation. A transverse wave deformation has four parameters: *magnitude, frequency*, *phase*, and *azimuth*, as illustrated in Figs. 3–6. Notice that traveling wave motion can easily be generated by animating the phase, and that the wave is largest in the center and falls off so that it does not affect the ends.[†] Any number of different waves can be superposed, although in practice one wave has generally been sufficient.

Implementation. Fig. 7 has pseudocode for a function that computes the curve of the rope. The rest shape is generated by a `catenary` function (ap-

[†]Note also that we do not attempt to maintain the length of the rope: an unduly large wave magnitude can perceptibly increase the length.

pendix A.2); the *swing* and *sway* parameters, while conceptually part of a gross shape deformation, are most easily merged with the rest shape since we implement them by leaning the catenary's up direction. The `wave` function (appendix A.3) returns a displacement which we scale by a $\frac{1}{2}$-cycle sine envelope.

The resulting curve is used as the axis of the final 3D surface. Most simply we define a generalized cylinder (appendix A.4); Fig. 17 shows a more complex example in which a twisted pair of wires is run along the axis, with a light bulb every few inches.

3.2 Loose Rope

Another common model is a rope that is held at one end, with the other end moving freely.

Rest shape. The rest shape of the loose rope is simply a straight line (Fig. 8, left).

Gross shape. The gross shape of the rope is sculpted as if it were a pipe-cleaner (Fig. 8, left middle). One or more bends can be introduced, each having parameters: *position* (how far along the rope the bend starts), *length* (how much of the rope is bent), *angle* (number of degrees rope is bent), and *azimuth* (orientation of bend). In practice we have generally used fewer than 5 bends in any given model.

Wave deformation. The wave deformation is the same as the suspended rope (Fig. 8, right middle). The animator is given a control *env* over the envelope: the held end always stays fixed, but the loose end can be free to displace ($env = 0$) or can be fixed ($env = 1$). Intermediate values of *env* can be useful, for example, if a heavy object is kinematically attached to the loose end so that it should perturb only slightly.

Clamping. We clamp the curve to keep it above the floor plane, so that the animator needn't worry about the rope penetrating the floor (Fig. 8, right).

Implementation. The loose rope function, Fig. 9, has the same structure as the suspended rope of Fig. 7. We generate the rest shape as a straight line, and bend it to yield the gross shape (appendix A.5). The envelope varies between a $\frac{1}{4}$- and $\frac{1}{2}$-cyle sine wave, based on the *env* parameter. The clamping simply prevents the z-component of the curve from being negative.

3.3 Coiled cord

A coiled elastic cord such as a telephone or microphone cord (Fig. 10) is an extension of the suspended rope of Sec. 3.1. The animator controls a "backbone" curve, and the coils are automatically wrapped

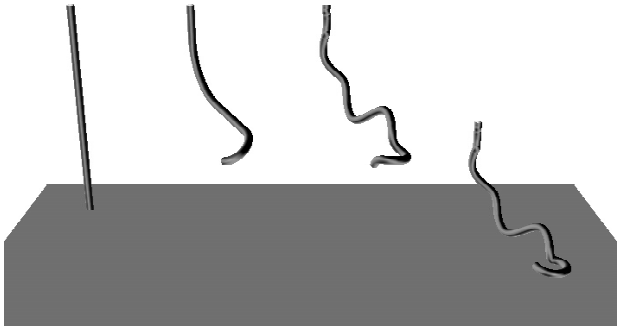


Figure 8: A loose rope. *Left to right:* Rest shape; with two bends; with a wave; lowered to rest on the floor.

```
func LooseRope(end, length)
parms:  env
        bends≡ n× { pos, len, angle, azm }
        waves≡ m× { mag, freq, phase, azm }
; rest shape
rope = end + line((0,0,0)..(0,0,-length))
; gross shape
rope = bend(rope, bends)
; wave deformation
envelope = sin(0..90+env*90)
rope = rope + envelope*wave(rope, waves)
; clamp to floor plane
zcomp(rope) = max(0, zcomp(rope))
return rope
```

Figure 9: Pseudocode for the loose rope of Fig. 8.

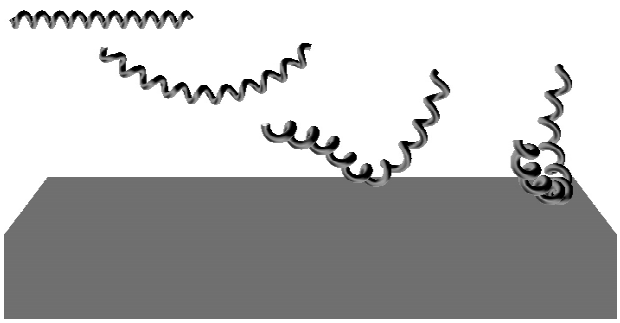


Figure 10: A coiled cord. *Left to right:* Straight rest shape; with sag; with several waves.

```
func CoiledCord(end1, end2, radius, ncoils)
parms:  sag
; construct backbone
length = sqrt((end2-end1)^2+4*sag^2)
rope = SuspendedRope(end1,end2,length)  ; from Fig. 7
; define a u,v cylinder surface
cyl = cylinder(rope, radius)
; construct helix
uv = sawtooth(ncoils)
helix = curve_on_surface(cyl, uv)
return helix
```

Figure 11: Pseudocode for the coiled cord of Fig. 10. The helix is constructed as illustrated in Fig. 12.

around the backbone, coming apart and together as the backbone stretches and shrinks.

Rest shape. Unlike the suspended rope, the rest shape of the backbone is a straight line between the two ends, as if tension in the coils pulls it taut.

Gross shape. The suspended rope had a fixed length. Here, the total length of the coiled cord may be fixed, but the backbone curve can change length as the coils move apart or together. A convenient way to control the length is to provide a *sag* parameter that lengthens the backbone so that the cord sags into a catenary; the *swing* and *sway* controls of the suspended rope are available here.

Wave deformation. Same as the suspended rope.

Implementation. This model illustrates the modularity of the method: the `SuspendedRope` function of Sec. 3.1 generates the backbone of the coiled cord. We compute the backbone length by triangular approximation based on the *sag* parameter (Fig. 11).

To create the axis of the helical curve, we construct a u, v parametric surface in the shape of a generalized cylinder (appendix A.4) then evaluate a u, v sawtooth function on that surface (A.6), as illustrated in Fig. 12.

3.4 Spring

A long, loose spring such as a Slinky† is similar to the coiled cord of Sec. 3.3, but the coils of the spring need to cluster and animate.

The coil clustering is controlled by the animator independently from the shape of the backbone, using the familiar layered approach:

Rest shape. The coils are uniformly distributed. With the coils in their rest shape, this model is the same as the coiled cord of Sec. 3.3.

Gross shape. The animator uses one or more "pincers" to grab coils and bunch them. Each pincer has three parameters: *position* (where along the backbone to place the pincer), *length* (how far along the backbone it extends), and *npinch* (how many coils fall within the pincer).

Wave deformation. A compression wave is a periodic bunching of the coils, with parameters: *magnitude* (how tightly the coils bunch), *frequency* (how many bunches), and *phase* (position of the wave).

Implementation. The backbone is generated using the suspended cord function; here, we also clamp the backbone to one radius above the floor so that the coils will not penetrate it (Fig. 14).

To represent the coils, we generalize the sawtooth mechanism of Sec. 3.3. We extend the u, v space of the

†*Slinky* is a trademark of James Industries.

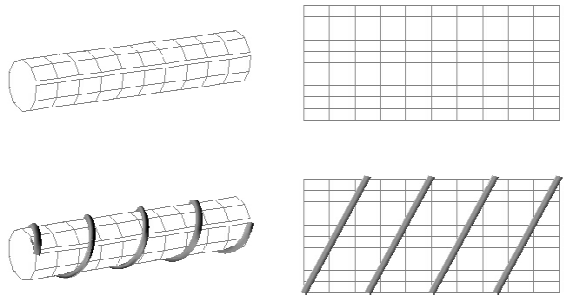


Figure 12: *Top:* u, v-parametrized cylinder, and its unrolled u, v space. *Bottom:* helix wrapped around cylinder corresponds with sawtooth in u, v space.

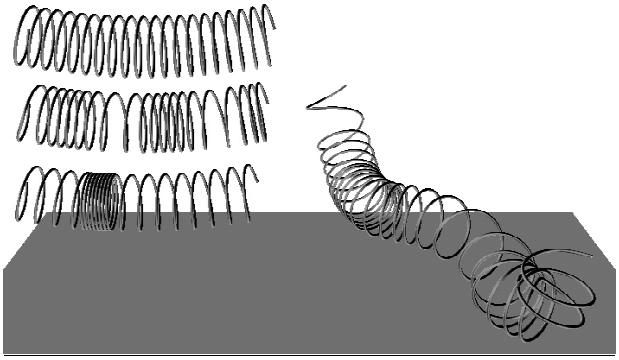


Figure 13: A loose spring. *Left, top to bottom:* Uniform coils; compression wave; pinched coils. *Right:* A little bit of everything.

```
func Spring(end1, end2, radius, ncoils)
parms:  sag
        pincers ≡ n× { pos, len, npinch }
        waves ≡ m× { mag, freq, phase }
; backbone curve
length = sqrt((end2-end1)^2+4*sag^2)
rope = SuspendedRope(end1,end2,length) ; from Fig. 7
; clamp above floor
zcomp(rope) = max(radius, zcomp(rope))
; define u,v cylinder surface
cyl = cylinder(rope, radius)
; rest shape for coils curve
uv = line((0,0)..(1,ncoils))
; gross shape
uv = pincer(uv, pincers)
; compression wave deformation
envelope = plateau(.5/ncoils)
uv = uv + envelope*compressionwave(uv,waves)
; create helix
helix = curve_on_surface(cyl, uv mod 1.0)
return helix
```

Figure 14: Pseudocode for the spring of Fig. 13. The helix is constructed as illustrated in Fig. 15.

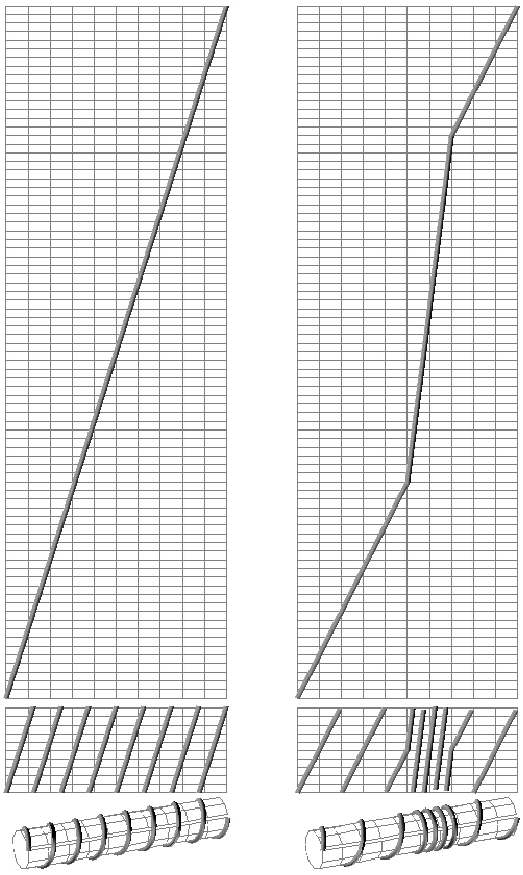


Figure 15: *Left:* u, v space is extended so that sawtooth of Fig. 12 is simply a straight line. *Right:* varying curve slope results in varying coil density.

```
func Lasso(hand, length)
parms:  spanlength
hoop = Hoop()
span = SuspendedRope(hoop[1], hand, spanlength)
tail = LooseRope(hand,length-spanlength)
return concat(hoop,span,tail)
```

Figure 16: Pseudocode for a lasso. Compound of a special-purpose "hoop" model with two rope models.

cylinder surface from $[0,1] \times [0,1]$ to be $[0,1] \times [0,N]$, where there are N coils in the helix. As illustrated in Fig. 15, the sawtooth of Sec. 3.3 is now a straight curve from $(0,0)$ to $(0,N)$. By deforming this curve, we can cluster the coils: where the curve is steeper, the coils are closer together. Note that the curve must be monotonically increasing to keep the coils from doubling back on each other.

We use our standard layered approach to manipulate the u, v curve (Fig. 14): define the rest shape, a straight line; generate a gross shape via `pincers` (appendix A.7); construct an envelope via `plateau` (A.8); and deform using `compressionwave` (A.9). Finally we map the curve back onto the 0-1 surface using `mod`, and evaluate the helix.

4 General Models

In production, each model often has its own special-purpose attachments, widgets, or controls. However, the basic models of Sec. 3 and their components are used as building blocks for the more intricate models.

Compound models. Often, a production model can be composed modularly from a collection of basic pieces. For example, to model a lasso (Fig. 16, Fig. 21): we built a special-purpose hoop, which was twirled and positioned directly; suspended a rope (Sec. 3.1) between the knot of the hoop and the hand of the character; and placed a loose rope (Sec. 3.2) continuing out from the hand.

Variations of models. Because of the internal modularity of the layered-deformation approach, we can easily construct variations on the basic models as needed. For example, the spring in Fig. 20 is based on the spring of Sec. 3.4, but with a more intricate variant for the gross shape: two spline control points can be adjusted to deform the curve; the *sag* is automatically scaled so that there is no sag as the two ends come together; and each end of the curve is constrained to be normal to the character's torso.

Other models. Other deformations than those described in Sec. 3 can be useful. For example, twisting and torsion waves can be implemented analogously to coil compression, by manipulating a rotation curve (see appendix A.4); and a pulse or soliton deformation can be used in addition to cyclic waves. Whatever new modeling deformations are introduced, the same general outline applies:

1. Define the underlying curve model.
2. Create the natural rest shape.
3. Perform gross shape adjustment.
4. Define a deformation envelope.
5. Perform wave deformations.
6. Clamp.
7. Create final object to render.

5 Results

The "fake" dynamics layered approach has proven to be very successful in production. The resulting motion is convincingly realistic in the context of character animation. The animators' reactions—often a matter of personal taste and style—have varied from from mild acceptance[†] to wild enthusiasm. Those who have used other methods to animate flexible bodies

[†]For some, the talk of "phase" and "frequency" was a bit hard to follow; perhaps these parameters should be repackaged in some less technical form.

Figure 17: The string of lights uses the "suspended rope" model of Sec. 3.1.

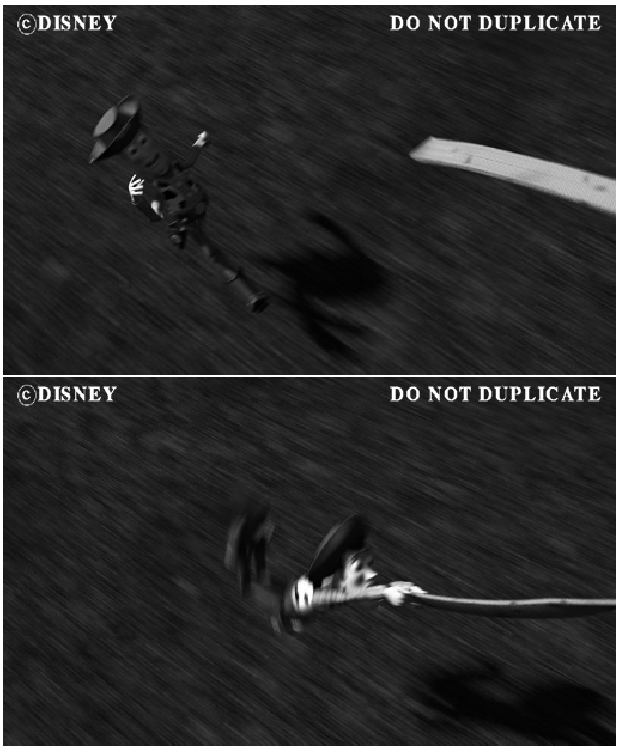


Figure 18: The strap uses a variant of the "loose rope" model of Sec. 3.2.

(such as control vertices or dynamics) have expressed a strong preference for our method.

Figs. 17–21 show pairs of frames from *Toy Story* featuring various models built using the techniques of Secs. 3 and 4. To animate the models, animators adjusted the shape parameters at key frames and used traditional keyframe splining to interpolate between them. Splining these shape parameters tends to yield convincing motion, so that often very few key values need be specified. For example, in the 30-frame scene of Fig. 17, only 8 key values were needed for the lights' *sag* parameter, 3 for wave *magnitude*, and 4 for wave *phase*; the other parameters were constant. In the 120-frame scene of Fig. 20, 4 key values were used for the Slinky's backbone wave *magnitude*, 3 for its *phase*, 4 for compression wave *magnitude*, and 2 each for its *frequency*, and *phase*; the other parameters were constant.

The ability to interactively adjust pose and timing by working within a keyframe system has been essential for integrating the models with character animation. Consider, for example, the frame-accurate choreography needed for the scene of Fig. 18.

The layered approach to modeling—rest shape, gross shape, wave deformations—lends itself to the layered approach to animation that Lasseter[1] suggests is the best practice for computer graphics animation.

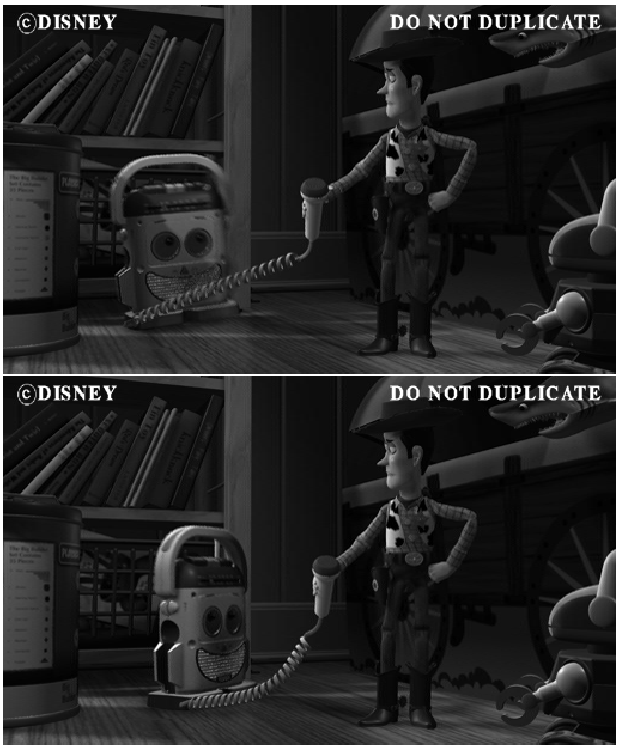


Figure 19: The microphone cord uses the "coiled cord" model of Sec. 3.3.

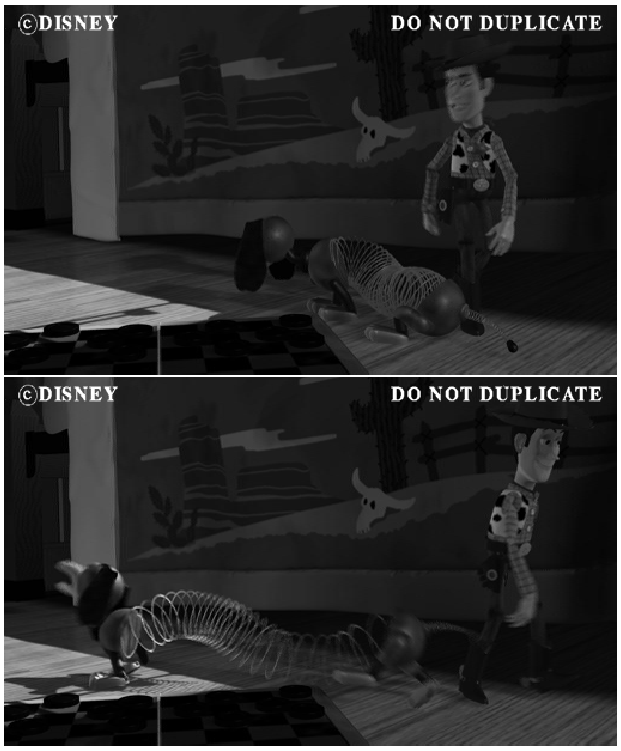


Figure 20: The Slinky uses a variant of the "spring" model of Sec. 3.4.

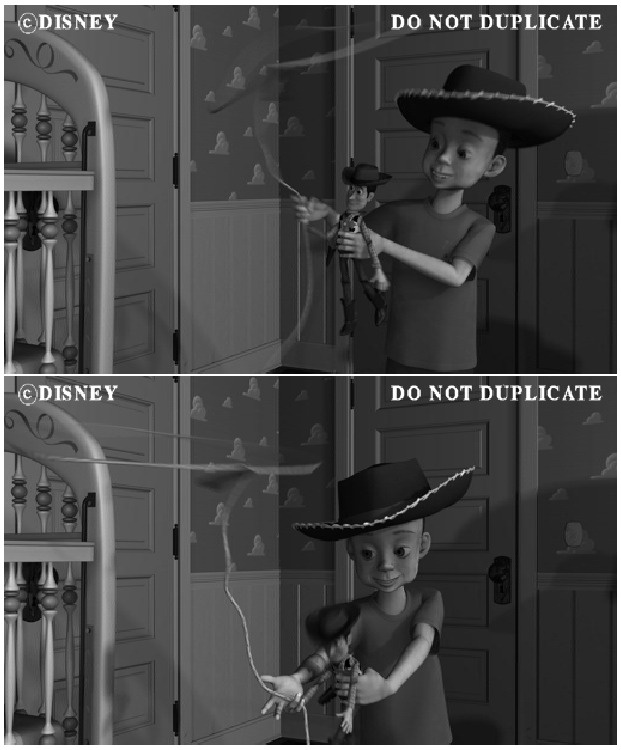


Figure 21: The lasso is a compound model as described in Sec. 4.

The ability to modularly construct special-purpose variants of a model has also been invaluable. For example, the string of lights in Fig. 17 is one of 16 variants (lights being caught, thrown, coiled, carried, dragged, etc.); after building the first few, the remainder were easily put together from available modules. The jump rope in Fig. 21 is similarly one of 10 variants.

6 Conclusion

We have presented a method to build kinematically animatable models of flexible linear bodies. The method is quite simple, both in concept and implementation, and has proven to be effective in practice.

Rather than building a single "does-everything" model, the method supports building simpler models for specific needs. Its strength lies in the modular approach that makes it easy to build new models from existing models and their components.

The method is based on the natural wave behavior of bodies that are constrained at one or both ends and are free in between. If these assumptions do not hold, the method is likely to be less applicable.

7 Future work

What about 2D bodies? Can the same method be used to model cloth and clothing? In principle, the idea of layering wave deformations above a gross shape remains valid. However, 2D waves are more complex than 1D, and the gross shape controls may themselves be non-trivial. We are hopeful, but have not yet investigated this question.

Also of interest is whether our type of kinematic control can be integrated with dynamic simulation methods, so that the choice between interactive control and computed motion is not all-or-nothing.

Acknowledgements

The author is indebted to the *Toy Story* animators for creating the animations that make this work worthwhile. Kurt Fleischer and Uriel Barzel provided valuable suggestions for this paper.

A Implementation Notes

This appendix discusses our representation for a curve, and describes the various library routines used in the pseudocode of Secs. 3 and 4.

A.1 Representing a curve

For our purposes, the easiest way to represent a curve is by brute force: by a constant number of sample points distributed uniformly along it. To deform the curve, we simply deform the sample points. In practice we have used a few 10's to perhaps 100 samples in any given model.

When we need to interpolate a smooth curve from the sample points, we create a B-spline by using a spline-fitting algorithm[11] to compute the knot points from the sample data.

If, as part of building a complex model, we need to change the number of sample points in a curve, we construct its B-spline and sample that spline uniformly at the new rate. If the samples have clustered and we need to resample uniformly, we construct its B-spline, supersample it into short segemnts, compute the lengths of each segment, invert the result, and re-sample the spline.

Each curve has a 0–1 u parameter defined along it. Often it is convenient to have an arc-length parameter s as well: we compute the total length L of the curve by constructing its B-spline, supersampling into segments, and summing the segment lengths; then s is simply a 0–L multiple of u.

A.2 `catenary`

We construct a catenary following the method described by Weill,[12] with some practical modification. First, for simplicity, we transform to canonical coordinates in which the up direction is $+y$, the lower end is at the origin and the other end is at $(1, dy)$ in the x-y plane. Next, given the desired length $L >= \sqrt{1+dy^2}$, we numerically solve for α in

$$\sqrt{L^2 - dy^2} = \frac{\sinh\alpha}{\alpha}.$$

The solution is straightforward since $\frac{\sinh\alpha}{\alpha}$ is monotonic in α: we approximate by bracketing the left-hand-side, sampling, and interpolating. Note that Weill solves for $A = \frac{1}{\alpha}$; we use α because it is close to 0, while A is inconveniently large. Given α, we compute:

$$\begin{aligned} M &= \sinh(2\alpha) \\ N &= \cosh(2\alpha) - 1 \end{aligned}$$

If $N > M$:

$$\begin{aligned} u &= atanh(\tfrac{M}{N}) \\ Q &= \tfrac{M}{\sinh(u)} \\ b &= \sinh^{-1}(\tfrac{2L\alpha}{Q}) - u \end{aligned}$$

If $M \geq N$:

$$\begin{aligned} u &= atanh(\tfrac{N}{M}) \\ Q &= \tfrac{N}{\sinh(u)} \\ b &= \cosh^{-1}(\tfrac{2L\alpha}{Q}) - u \end{aligned}$$

And in all cases:

$$c = \frac{\cosh(b)}{2\alpha}$$

The catenary is nominally given by

$$C(s) = \frac{\cosh(2\alpha s + b)}{2\alpha} - c, \quad s \in [0,1]$$

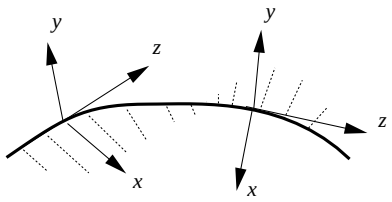
Next, to compensate for our cheap approximation of α, we compute the error e, and subtract it off linearly back along the curve:

$$\begin{aligned} e &= \frac{\cosh(2\alpha + b)}{2\alpha} - c - dy \\ C(s) &= \frac{\cosh(2\alpha s + b)}{2\alpha} - c - se, \quad s \in [0,1] \end{aligned}$$

To generate the final curve, we first sample $C(s)$ in s, then resample uniformly by arc length as described in A.1.

A.3 `wave`

To apply a transverse wave to a curve in space, we create a local coordinate frame at each sample point, by evaluating the tangent to the curve and taking the cross product with some suitable fixed vector such as the world-space z axis, or else by computing the Frenet frame or a rotation-minimizing frame as per Bloomenthal.[13] The displacement is at a constant angle in the local xy plane, specified by the azimuth parameter.



If several waves are to be superposed, the displacements are computed independently based on the same gross shape. The displacements at each sample point are added together to produce the final displacement.

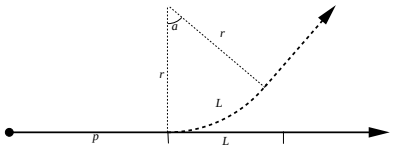
A.4 `cylinder`

We create a generalized cylinder from an axis curve by extruding a profile shape (commonly a circle) along the curve. A curve can be provided to specify the rotation of the profile shape as it is extruded, thus supporting twisting and torsion waves.

The result of the extrusion is a u, v parametric surface, where $u \in [0,1]$ moves axially along the curve and $v \in [0,1]$ moves around the profile.

A.5 `bend`

We perform bends using the circular bend deformation described by Barr.[14] Within the bending region the shape is deformed circularly, and beyond the bend the deformation is a rigid translation and rotation:
If several bends are to be performed, they are sorted by distance from the fixed origin, and applied sequentially.



A.6 curve_on_surface

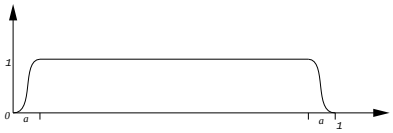
Given a u, v parametric surface, and a 2D curve in u, v space, `curve_on_surface` returns the corresponding curve in 3-space. Since our curves are represented as sample points, this is easy: we simply evaluate the surface at the u, v samples and return the resulting 3D points.

A.7 pincer

The *pincer* function simply constructs the piecewise-constant-slope curve of Fig. 15. If several pincers are defined, the appropriate multi-segment curve is constructed. In practice, we slightly round the corners where segments meet, to avoid kinks in the resulting helix.

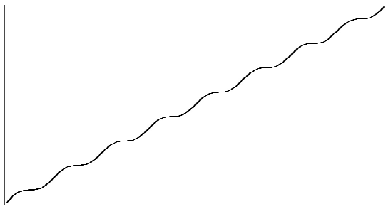
A.8 plateau

The `plateau` function, given a width a returns an envelope that is flat at the top and falls off via two Hermite curves at the ends:



A.9 compressionwave

We are given a monotonically-increasing curve $v = c(u)$ that describes a helix on a cylinder as per Fig. 12. To perform a compression wave on the helix, we need to induce a sinusoidal deformation on the curve, while maintaining the monotonicity.



The following does the trick: given a magnitude k, frequency λ, and phase ϕ, and a sample point (u, v) with $u \in [0, 1]$ and $v \in [0, N]$ for an N-coil helix, displace v by

$$\left(\frac{k}{1+k}\right)\left(N\left(u + \frac{sin(2\pi\lambda u + \phi)}{2\pi\lambda}\right) - v\right)$$

References

[1] J. Lasseter. Principles of traditional animation applied to 3D computer animation. *Computer Graphics* 21(4) (Proc. SIGGRAPH), July 1987, pp. 35–44.

[2] R. Barzel and A.H. Barr. A modeling system based on dynamic constraints. *Computer Graphics* 22(4) (Proc. SIGGRAPH), August 1988, pp. 179–188.

[3] Z. Liu, S.J. Gortler, and M.F. Cohen. Hierarchical Spacetime Control. In *Computer Graphics* Proceedings, Annual Conference Series, ACM SIGGRAPH, 1994, pp. 35–42.

[4] A. Pentland and J. Williams. Good vibrations: modal dynamics for graphics and animation. *Computer Graphics* 23(3) (Proc. SIGGRAPH), July 1989, pp. 215–222.

[5] J.E. Chadwick, D.R. Haumann, and R.E. Parent. Layered construction for deformable animated characters. *Computer Graphics,* 23(3) (Proc. SIGGRAPH), July 1989, pp. 243–252.

[6] D. Terzopoulos and A. Witkin. Physically-based models with rigid and deformable components. In *Proc. Graphics Interface,* Edmonton, Alberta, Canada, June 1988, pp. 146–154.

[7] R. Barzel, J.F. Hughes, and D. Wood. Plausible Motion Simulation for Computer Graphics Animation. Submitted to the Eurographics Workshop on Animation and Simulation, 1996.

[8] W.T. Reeves, E.F. Ostby, and S.J. Leffler. The Menv modelling and animation environment. *The Journal of Visualization and Computer Animation* Vol. 1, 1990, pp. 33–40.

[9] J. Lasseter. Personal communication, 1995.

[10] R.P. Feynman, R.B. Leighton, M. Sands. *The Feynman Lectures on Physics, Volume 1.* Addison-Wesley Publishing Company, Reading MA, 1963.

[11] R.H. Bartels, J.C. Beatty, and B.A. Barsky. *An introduction to splines for use in computer graphics and geometric modeling.* Morgan Kaufmann Publishers, Inc., Los Altos CA, 1987.

[12] J. Weil. The synthesis of cloth objects. *Computer Graphics* 20(4) (Proc. SIGGRAPH), August 1986, pp. 49–55.

[13] J. Bloomenthal. Calculation of reference frames along a space curve. In *Graphics Gems,* Andrew S. Glassner, editor, Academic Press, Boston, 1990, pp. 567–571.

[14] A.H. Barr. Global and local deformations of solid primitives. *Computer Graphics* 18(3) (Proc. SIGGRAPH), July 1984, pp. 21–30.