Physically Based Modeling
CS 15-860 Spring 1994
Assignment 3: A Bead on a Wire
Due Tuesday February 22

In this assignment, you'll be simulating a particle in the plane, subject to a constraint of the form $C(x, y) = 0$. You'll be implementing two different constraints: one that keeps the particle on a circle, and another that keeps it on a funny curve we made up—we'll get to the details shortly.

Your program should draw the particle as it moves, as well as drawing the curve the particle is supposed to stay on. You should be able to use the mouse to pull on the particle, as in the previous assignments, and you will also want gravity and some viscous drag. You should be able to re-use much of your previous code. **Be sure to see the "how-to" document for help on doing the derivations and implementation.**

In class, we discussed a number of variations on the constraint problem, and we want you to be able to try them all:

- Penalty method vs. constraint force. Is the penalty method as losing as we said it is? Try to find out. Can you make your penalty-based simulation fast, accurate, and stable all at the same time?
- First order vs. second order. Try simulating a particle governed by $f = mv$, as well as a real newtonian $f = ma$ particle. (Note: we said you want viscous drag, but only in the $f = ma$ case. Velocity-dependent forces don't make any sense in a first-order world!)
- Constraint feedback on/off (Doesn't apply to the penalty method). If you just solve for $\dot{C} = 0$ or $\ddot{C} = 0$, the particle will drift off the curve, especially at large step sizes. Play with this enough to convince yourself that solving for $\dot{C} = -\alpha C$, or $\ddot{C} = -(\alpha C + \beta \dot{C})$ really prevents drift. Play with the feedback constants.
- Time varying vs. stationary constraints. Try making the radius of your circular wire vary sinusoidally over time (e.g. $r = r_0 + a \sin kt$). Be sure the radius doesn't go negative! Pick k to make the frequency reasonable relative to your step size. See what happens if you forget to include the direct time derivative term in the constraint force calculation.

Here are the particular constraints we want you to implement:[1]

The first constraint is a circle: $C(x, y) = x^2 + y^2 - r^2$, where r is the radius. We recommend using $r = 1$, and scaling your drawing appropriately. To draw the circle on the screen, you'll need the parametric form $[x = r \cos\theta, y = r \sin\theta]$. Take steps in θ from 0 to 2π, calculate $[x, y]$, and draw a line from the previous point to the new one.

The second constraint is like a circle, except its radius varies sinusoidally as a function of the angle, so you get rounded star- or flower-like shapes. We'll start with the parametric form, in polar coordinates (r, θ). Taking θ as the independent variable, we get $r = r_0 + a \cos(n\theta)$, where r_0 is the mean radius, a determines the "amplitude" or waviness of the shape, and n, a positive integer, determines the number of

[1] Both of these constraints go singular at the origin (the derivatives go away.) Don't initialize your particle at the origin! If you're compulsive, you can put a divide-by-zero test in your code.

"petals." Since r must stay positive, r_0 must be greater than a. To draw the curve, you'll take steps in θ, using the polar-to-rectangular conversion [$x = r\cos\theta$, $y = r\sin\theta$]. To get the constraint function $C(x, y)$ we use the rectangular-to-polar conversion [$r = \sqrt{x^2 + y^2}, \theta = \text{atan}(y, x)$], which gives

$$C(x, y) = r_0 + a\cos(n\,\text{atan}(y, x)) - \sqrt{x^2 + y^2}$$

. (To evaluate the inverse tangent, you'll want to use the two-argument atan function, usually called atan2.)

Here's a handy calculus fact:

$$\frac{d}{dx}\text{atan}(y, x) = \frac{-y}{x^2 + y^2}, \qquad \frac{d}{dy}\text{atan}(y, x) = \frac{x}{x^2 + y^2}.$$

In addition to seeing your program run, we'll want to see your constraint force derivations for these two constraints.