Physically Based Animation

CSE 558 Spring 2000
Zoran Popović
Project 1: Tinkertoys — Constrained particle system

Out: April 6, 2000
Due: Thursday, April 20

In this assignment, you will be simulating particles in the plane, subject to two types of constraints. You will need to implement at least one constraint of the form $C(p_x, p_y) = 0$: one that keeps the particle p on a circle, and one constraint of the form $C(p_x, p_y, q_x, q_y) = 0$: one that keeps the particles p and q at a fixed distance d apart from each other.

Your program should draw the particles as they move, as well as drawing the curve the particle is supposed to stay on for each constraint of the type $C(p_x, p_y) = 0$ and a line between all particles with a distant constraint. You should be able to use the mouse to pull on the particle, and you will also want gravity and some viscous drag.

1 Single particle constraint

The constraint is a circle: $C(p_x, p_y) = p_x^2 + p_y^2 - d^2$, where d is the radius. To draw the circle on the screen, you'll need the parametric form $[x = r\cos(t), y = r\sin(t)]$. Take steps in t, from 0 to 2π , calculate $[x, y]$, and draw a line from the previous point to the new one.

The constraint becomes 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.

2 Multi-particle constraints

You'll implement all constraints using Lagrange multipliers, regardless of the number of particles that they affect. Rather than hand-coding the whole system, you'll build the constraint matrix on the fly by allowing each constraint to make its own contribution to the global J and $\dot{J}$ matrices.

We talked about sparse matrices in class, but you won't need to use them for the assignment. The web page has a pointer to the C code which does the basic vector/matrix manipulations you need, including solving the linear system.

You should implement distance constraints as little structures, analogous to a structure that represents a spring in the particle systems. The structure should point to the pair of particles it influences, and should also point to the three numerical functions that define its behavior: One that computes the value of the constraint function $C = ||p_1 - p_2||^2 - d^2$, one that stuffs the derivatives $\frac{\partial C}{\partial p_1}$ and $\frac{\partial C}{\partial p_2}$ into the global J matrix, and another that stuffs their time derivatives into the global $\dot{J}$ matrix.

In order to build the global matrices and vectors, the constraints need to know where to put their derivatives. This is a simple matter of indexing: number all your constraints, and all your particles. The index of the i-th constraint is just i, and the indices of p_j (j-th particle) are $2j$ and $2j+1$, for the x and y components respectively. So, e.g. the derivative of constraint C_i with respect to the y component of particle p_j goes into element $i, 2j+1$ of the matrix.

You can in principle build arbitrary ”tinkertoy” structures interactively. However, you’re not required you to do any interactive construction. You can read a model in from a file, or just wire it into the code. If the latter, do at least the ”triangle with a tail”.

3 Mouse interaction

To make your simulations interactive, implement a mouse-spring as follows: each time the mouse button is pressed, find the particle closest to the mouse at that moment. Apply an attractive spring force (damped, with zero rest-length) between that particle and the mouse until the button is released. Let the ”mouse force” F_M be computed by $F_M = k_m(M_x - p_x, M_y - p_y)$ with $k_m > 0$. In order for your simulation not to get out of control, you’ll want to add a damping force F_d of the form $F_d = -k_d v$ where $k_d > 0$, so that the total force F is $F = F_M + F_D$.

In order to keep things on the screen, you’ll want to have at least one particle of the constrained structure fixed to lie on a circle constraint.

You should also be able to turn gravity forces on/off in your system.

4 Experiments

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

- 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 $\dot{C} = -\alpha\ddot{C}$, or $\ddot{C} = -\alpha\,\ddot{C} + \beta\dot{C}$ 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.

- Make the distance d in the distance constraint time varying.

- If you team has 3 people, or if you want to learn how it's really done, or if you just want the general warm and fuzzy feeling that you solved the problem in a righteous way implement the sparse matrix representation of J and $\dot{J}$ and use the conjugate gradient method to solve for the λ vector. I suggest using a good vector library such as VL. You'll find the pointer to the VL source on the project web page. The web page also contains pointers to the short and long descriptions for the conjugate gradient method. Don't be detracted by a fancy name — the basic conjugate gradient method is extremely simple to implement. This implementation enables interactive speeds for extremely large constrained particle systems.

- Add additional types of constraints to your system. The formulation and the implementation should be flexible enough to handle any algebraic constraint.