Physically Based Modeling
CS 15-860 Spring 1994
Assignment 5: 2D Rigid Bodies
Due April 14

In this assignment, you'll augment your constrained particle system by adding in 2D rigid bodies. Your new system should be able to handle a mix of both particles and rigid bodies.

Each rigid body will be a polygon, described as a list of vertices in body space. During the simulation, each body is displayed by transforming the body-space vertex locations into world-space, according to the (c_x, c_y, θ) state variables of the body, and drawing lines between adjacent vertices. The world-space locations of the vertices are *not* carried along as part of the state vector; you must compute them from the body-space vertex locations and (x, y, θ) whenever they are needed.

If you wish, you may limit yourself to rectangular rigid bodies. If you wish to implement more complicated shapes, you may do so, but you will have to calculate both the center of mass and moment of inertia I for any shape you wish to simulate (not necessarily exactly, but to a reasonable approximation. Essentially, if it behaves well, your approximation is reasonable.) Remember, when you define the vertex locations in body space, the center of mass must lie at the origin, or your objects will behave strangely. (Note: do *not* calculate either the center of mass or I by pretending the body consists of particles located at just the vertices. You should consider the body to be a solid region with constant density, and compute both the center of mass and I accordingly. Treat the summations for the center of mass and I as integrals over the area of the body. The total mass M is the area of the polygon.)

Implement a mouse spring that allows you to grab either a particle, or any vertex of a rigid body when the mouse-button is pressed. (You might also want to let the user grab the center of mass of a body, if the cursor is closer to the center than any other vertex of the body.) As long as the button remains pressed, do not not change which feature of a body you are pulling on (i.e. which particle, which vertex, or the center). If the button is released, a new particle or vertex may be grabbed when the button is re-pressed. The mouse-spring should exert a force $F_m = k(m - q)$ where m is the mouse position, and q is the location of the point grabbed in world space. The mouse-spring also exerts a torque of $r \times F_m$ where r is the displacement of q from the center of mass, in world space. (Or in generalized notation, a world-space force F acting at point p produces the generalized force $(F_x, F_y, (q - c) \times F)$.)

Your system should be able to able to handle the following additional constraints:

- If $q(t)$ is the world-space location of some particular point on a rigid body, you should be able to freeze the rigid body so that $q(t)$ remains a constant.
- If $q_a(t)$ and $q_b(t)$ are points in world-space on rigid bodies A and B, you should be able to constrain $q_a(t) = q_b(t)$.
- If $q_a(t)$ is a point on a rigid body A and $q_b(t)$ is a particle B, you should be able to constrain the distance between $q_a(t)$ and $q_b(t)$ to some constant $K > 0$. Also, handle

the case when you simply want to constrain $q_a(t)$ to remain a fixed distance from some constant point in space.

For all of the above constraints, you will need to determine how many Lagrange multipliers are needed. (One implementation approach is to generalize constraints so that they are not necessarily scalars but vectors i.e. treat $C(x) = 0$ as a vector equation, with all components needing to be zero.)

Please create an example or examples that illustrates all of the above constraints. You should use at least 4 rigid bodies. Interesting things to implement are a chain of bodies with one body fixed (i.e. a multi-body pendulum) and also the case where the connections between the bodies form a complete closed loop.

Collisions

After getting all of the above working, implement a floor (and walls if you like) which bodies and particles can collide with. You will implement collisions just as in assignment 2 (you may wish to reread the discussion on detecting collisions and using numerical tolerances). At the beginning of each time step (i.e. *before* computing any derivatives) verify that all the vertices of your body lie on or above the floor. If a vertex lies on the wrong side of a bounding plane, *and* is moving in the wrong direction (i.e. downwards), you will need to subject the entire system to an impulse. Check the individual particles in the same manner. As in assignment 2, if $\hat{n}$ represents a unit normal to the barrier, pointing towards the legal side, the outgoing normal velocity $\hat{n} \cdot v^+$ of the vertex or particle must satisfy

$$\hat{n} \cdot v^+ = -(1+\epsilon)\hat{n} \cdot v^- \tag{1}$$

where ϵ lies between zero and one. Note that for the rigid-body case, v^- is the velocity of the colliding vertex prior to the collision (and not simply the linear velocity of the body). Similarly, v^+ is the vertex's total velocity after the collision.

To satisfy $\hat{n} \cdot v^+ = -(1+\epsilon)\hat{n} \cdot v^-$, you must subject the body or particle to an impulse, as in assignment 2. However, you must be careful to do so in a manner that does not break the constraints of the system! To make life simple, treat the "bouncing away" as an additional constraint that is temporarily imposed on the system. That is, if you previously had n constraints on your system, then add a temporary constraint C_{n+1} of the form

$$C_{n+1} = \hat{n} \cdot (q - p_0) \tag{2}$$

where q is the world-space location of the particle or vertex, and p_0 is a point on the barrier. For the first n constraints, you will have initially that $\dot{C}_i^- = 0$ and $\dot{C}_{n+1} < 0$. While you want $\dot{C}_i^+ = 0$ for the first n constraints, you want $\dot{C}_{n+1}^+ = -\epsilon\dot{C}_{n+1}^-$ for the $n+1$th constraint. Compute a constraint impulse $\mathbf{J}^T\lambda$ where $\mathbf{J}$ is the derivative of the $(n+1)$-vector of constraints, with respect to the geometric state $\mathbf{x}$ of the system (i.e. just positions, not velocities), and λ is the $(n+1)$-vector of impulse magnitudes. When you apply this impulse, you will have

$$\dot{\mathbf{x}}^+ = \dot{\mathbf{x}}^- + \mathbf{W}\mathbf{J}^T\lambda. \tag{3}$$

You must solve for λ so that $\dot{C}_i^+ = 0$ for the first n constraints, and $\dot{C}_{n+1}^+ = -\epsilon\dot{C_{n+1}}^-$. After applying the impulse, the temporary $n+1$th constraint is removed. (Note that the matrix **W** gains three diagonal components for each rigid body—the first two diagonal components are $1/m$, and the third diagonal component is $1/I$.)

You do not need to worry about allowing particles or bodies to "rest" on the bounding plane—this becomes complicated because of the additional constraints. (The problem is in knowing when to relax the contact constraint preventing a particle or vertex from falling through a barrier.)