Differential Constraints

Beyond Points and Springs
- You can make just about anything out of point masses and springs, in principle
- In practice, you can make anything you want as long as it’s jello
- Constraints will buy us:
  - Rigid links instead of goopy springs
  - Ways to make interesting contraptions

A bead on a wire
- Desired Behavior:
  - The bead can slide freely along the circle
  - It can never come off, however hard we pull
- Question:
  - How does the bead move under applied forces?

Penalty Constraints
- Why not use a spring to hold the bead on the wire?
- Problem:
  - Weak springs ⇒ goopy constraints
  - Strong springs ⇒ neptune express!
- A classic stiff system
The basic trick (f = mv version)

- 1st order world.
- Legal velocity: tangent to circle (N·v = 0)
- Project applied force f onto tangent: $\mathbf{f}' = \mathbf{f} + \mathbf{f}_c$
- Added normal-direction force $\mathbf{f}_c$: constraint force
- No tug-of-war, no stiffness

$$f_c = -\frac{\mathbf{f} \cdot \mathbf{N}}{\mathbf{N} \cdot \mathbf{N}} \quad \mathbf{f}' = \mathbf{f} + \mathbf{f}_c$$

\[ f = ma \]

Now for the Algebra …

- Fortunately, there’s a general recipe for calculating the constraint force
- First, a single constrained particle
- Then, generalize to constrained particle systems

Representing Constraints

I. Implicit:
$$C(x) = |x| - r = 0$$

II. Parametric:
$$x = r \begin{bmatrix} \cos \theta \\ \sin \theta \end{bmatrix}$$
Maintaining Constraints Differentially

- Start with legal position and velocity.
- Use constraint forces to ensure legal curvature.

\[ C = 0 \quad \text{legal position} \]
\[ \dot{C} = 0 \quad \text{legal velocity} \]
\[ C = 0 \quad \text{legal curvature} \]

Constraint Gradient

Implicit:
\[ C(x) \equiv |x| - r = 0 \]
Differentiating \( C \) gives a normal vector.
This is the direction our constraint force will point in.

Constraint Forces

Constraint force: gradient vector times a scalar \( \lambda \).
Just one unknown to solve for.
Assumption: constraint is passive—no energy gain or loss.

Point-on-circle

Constraint Force Derivation

\[ N = \frac{\partial C}{\partial x} \]
\[ C(x(t)) = N \cdot x \]
\[ \dot{C} = \frac{\partial}{\partial t} (N \cdot x) \]
\[ = N \cdot \dot{x} + N \cdot \ddot{N} \]
\[ \lambda \left( \mathbf{f} + \mathbf{f}_c \right) = \frac{m}{\mathbf{f}} \]
Set \( \dot{C} = 0 \), solve for \( \lambda \):
\[ \lambda = -m \frac{N \cdot \dot{x} - N \cdot f}{N \cdot N} \]
Constraint force is \( \lambda N \).
Example: Point-on-circle

\[ C = |x| - r \]
\[ N = \frac{\partial C}{\partial x} = \frac{x}{|x|} \]
\[ N = \frac{\partial^2 C}{\partial x \partial t} = \frac{1}{|x|} \left[ x - \frac{x \cdot x}{x \cdot x} \right] \]

\[ \lambda = -m \frac{N \cdot x}{N \cdot N} - \frac{N \cdot f}{N \cdot N} = \left[ m \left( \frac{x \cdot x}{x \cdot x} - m \cdot (x \cdot x) - x \cdot f \right) \right] \frac{1}{|x|} \]

Drift and Feedback

- In principle, clamping \( C \) at zero is enough
- Two problems:
  - Constraints might not be met initially
  - Numerical errors can accumulate
- A feedback term handles both problems:
  \[ C = - \alpha C - \beta C, \text{ instead of } C = 0 \]
  \( \alpha \) and \( \beta \) are magic constants.

Tinkertoys

- Now we know how to simulate a bead on a wire.
- Next: a constrained particle system.
  - E.g. constrain particle/particle distance to make rigid links.
- Same idea, but…

Constrained particle systems

- Particle system: a point in state space.
- Multiple constraints:
  - each is a function \( C_i(x_1, x_2, \ldots) \)
  - Legal state: \( C_i = 0, \forall i \).
  - Simultaneous projection.
  - Constraint force: linear combination of constraint gradients.
- Matrix equation.
Compact Particle System Notation

\[ \ddot{\mathbf{q}} = \mathbf{WQ} \]

- \( \mathbf{q} \): 3\(n\)-long state vector.
- \( \mathbf{Q} \): 3\(n\)-long force vector.
- \( \mathbf{M} \): 3\(n\) x 3\(n\) diagonal mass matrix.
- \( \mathbf{W} \): \( \mathbf{M} \)-inverse (element-wise reciprocal)

\[ \mathbf{q} = [x_1, x_2, ..., x_n] \]
\[ \mathbf{Q} = [f_1, f_2, ..., f_n] \]
\[ \mathbf{M} = \begin{bmatrix}
  m_1 & m_{12} & \cdots & m_{1n} \\
  m_{21} & m_2 & \cdots & m_{2n} \\
  \vdots & \vdots & \ddots & \vdots \\
  m_{n1} & m_{n2} & \cdots & m_n
\end{bmatrix} \]
\[ \mathbf{W} = \mathbf{M}^{-1} \]

Particle System Constraint Equations

Matrix equation for \( \lambda \)

\[ [\mathbf{JWJ}^T] \lambda = -\mathbf{JQ} - [\mathbf{JW}] \mathbf{Q} \]

More Notation

\[ \mathbf{C} = [C_1, C_2, ..., C_m] \]
\[ \lambda = [\lambda_1, \lambda_2, ..., \lambda_m] \]
\[ \mathbf{J} = \frac{\partial \mathbf{C}}{\partial \mathbf{q}} \]
\[ \mathbf{J} = \frac{\partial^2 \mathbf{C}}{\partial^2 \mathbf{q}} \]

Constrained Acceleration

\[ \ddot{\mathbf{q}} = \mathbf{W} (\mathbf{Q} + \mathbf{J}^T \lambda) \]

How do you implement all this?

- We have a global matrix equation.
- We want to build models on the fly, just like masses and springs.
- Approach:
  - Each constraint adds its own piece to the equation.

Matrix Block Structure

- Each constraint contributes one or more blocks to the matrix.
- Sparsity: many empty blocks.
- Modularity: let each constraint compute its own blocks.
- Constraint and particle indices determine block locations.

Derivation: just like bead-on-wire.
Global and Local

Constraint Structure

Each constraint must know how to compute these

\[
\begin{align*}
\frac{\partial C}{\partial x_1}, & \quad \frac{\partial C}{\partial x_2}, \\
\frac{\partial^2 C}{\partial x_1^2}, & \quad \frac{\partial^2 C}{\partial x_1 \partial t}, \\
& \quad \frac{\partial^2 C}{\partial x_2 \partial t}
\end{align*}
\]

Distance Constraint

\[ C = |x_1 - x_2| - r \]

Constrained Particle Systems

Modified Deriv Eval Loop

1. Clear Force Accumulators
2. Apply forces
3. Compute and apply Constraint Forces
4. Return to solver

Added Step

Added Stuff
**Constraint Force Eval**

- After computing ordinary forces:
  - Loop over constraints, assemble global matrices and vectors.
  - Call matrix solver to get $\lambda$, multiply by $J^T$ to get constraint force.
  - Add constraint force to particle force accumulators.

**Impress your Friends**

- The requirement that constraints not add or remove energy is called the *Principle of Virtual Work*.
- The $\lambda$’s are called *Lagrange Multipliers*.
- The derivative matrix, $J$, is called the *Jacobian Matrix*.

---

**A whole other way to do it.**

**I. Implicit:**

$$C(x) = |x| - r = 0$$

**II. Parametric:**

$$x = r [\cos \theta, \sin \theta]$$

**Parametric Constraints**

**Parametric:**

$$x = r [\cos \theta, \sin \theta]$$

- Constraint is always met exactly.
- One DOF: $\theta$.
- Solve for $\theta$. 
**Parametric bead-on-wire \( f = mv \)**

- **x** is not an independent variable.
- First step—get rid of it:
  - \( x = \frac{f + f_c}{m} \)
  - \( x = T\dot{\theta} \)
  - \( T\dot{\theta} = \frac{f + f_c}{m} \)

For our next trick...

As before, assume \( f_c \) points in the normal direction, so \( T f_c = 0 \)

We can nuke \( f_c \) by dotting \( T \) into both sides:

\[
T \theta = \frac{f + f_c}{m}
\]

\[
T \cdot T \theta = \frac{T \cdot f + T \cdot f_c}{m}
\]

\[
\theta = \frac{1}{m} \frac{T \cdot f}{T \cdot T}
\]

**General case**

Lagrange dynamics:

\[
J^T \dot{J} u + J^T M J u - J^T Q = 0
\]

where

\[
J = \frac{\partial q}{\partial u}
\]

Not to be confused with:

\[
\left[ J W J^T \right] \lambda = -J q - [JW] Q
\]

where

\[
J = \frac{\partial C}{\partial q}
\]

**Parametric Constraints: Summary**

- **Generalizations:** \( f = ma \), particle systems
  - Like implicit case (see notes.)
- **Big advantages:**
  - Fewer DOF’s.
  - Constraints are always met.
- **Big disadvantages:**
  - Hard to formulate constraints.
  - No easy way to combine constraints.
- **Official name:** *Lagrangian dynamics.*
Hybrid systems

\[ \lambda = -Ju - [JW]Q \]

where

\[ W = M^{-1} = \int m_s \mathbf{q}_s \mathbf{q}_s \]

\[ C(q(u)) \]

\[ J = \frac{\partial C}{\partial q} \frac{\partial q}{\partial u} \]

Project 1:

- A bead on a wire (implicit)
- A double pendulum
- A *triple* pendulum
- Simple interactive tinkertoys