16 Basic Particle Systems

Steve Marschner
(slides coauthored with Eston Schweickart)
CS5625 Spring 2022

Overview

Particles and Springs

- Matrix notation and the Mass Matrix
- Equations of motion
- Forces as derivatives of energy and the Stiffness matrix

Time Integration Algorithms

- Forward, Backward, and Symplectic Euler

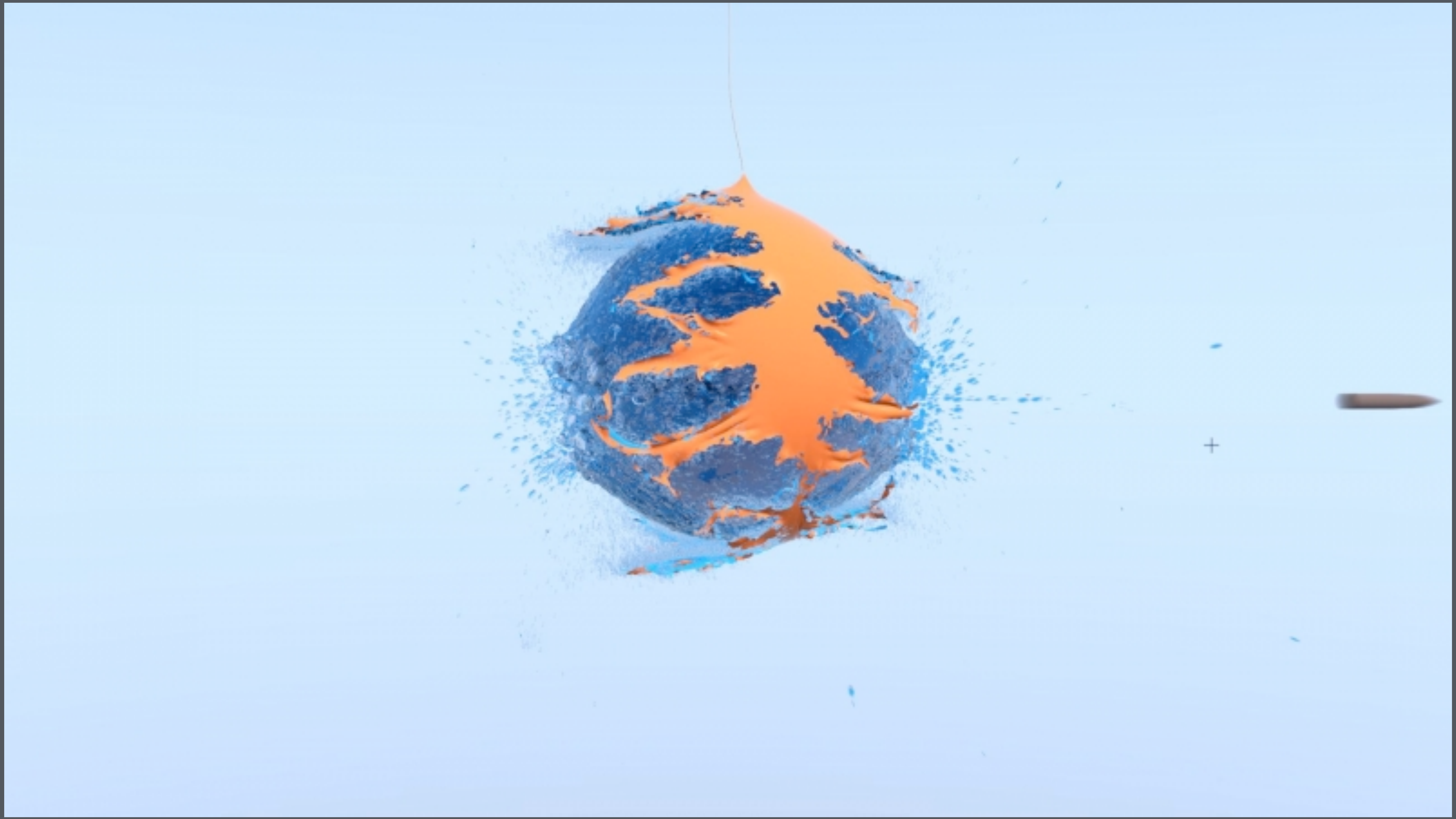
Constraints and Solvers

- Iterative Methods
- Manifold Projection

Examples of Particle Systems

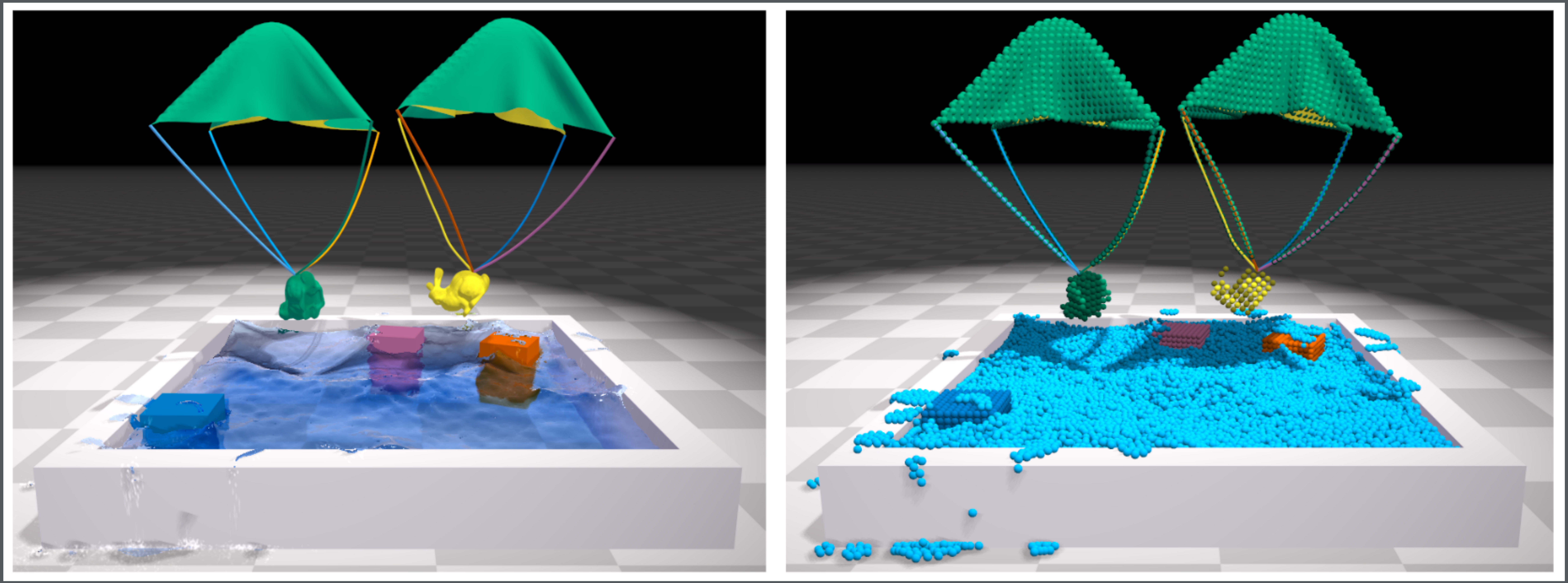
Particle Dreams [Karl Sims, 1988]

Examples of Particle Systems



Balloon Burst [Macklin et. al, SIGGRAPH 2015]

Examples of Particle Systems



Unified Particle Physics for Real-Time Applications [Macklin et. al, SIGGRAPH 2014]

Unified Particle Physics for Real-Time Applications [Macklin et. al, SIGGRAPH 2014]

Particle System Review

Each particle has a mass

Each particle's movement is determined by a sum of forces

- Forces depend on particles' positions and velocities, and maybe time

F = ma determines how the particle moves

- Forces determine acceleration at any given time given the position and velocity
- Differential equation determines the entire motion given initial position and velocity

Basic Algorithm

1) **Clear forces from previous calculations**

2) **Calculate/accumulate forces for each particle**

3) **Solve for particle's state (position, velocity) for the next time step**

Unary Forces

Constant

- Gravity

Position/Time-Dependent

- Force fields, e.g. wind

Velocity-Dependent

- Drag

Matrix Notation

If we have multiple particles, it is nice to group variables together

- Example: 1D System
- Example: 3D System

Mass matrix

- An n x n matrix that represents the mass distribution of n particles
- For simple systems, this is block-diagonal, where the ith block represents the mass of the ith particle
- Each block is a scaled identity matrix $m\mathbf{I}$ where m is the particle's mass and the size of $\mathbf{I}$ is the number of dimensions of the domain

Integration Algorithm 1

Calculating Particle State from Forces: First attempt

- Use forces to update velocity
- Use old velocity to update position

Issues

- Unstable in certain cases!
- Reducing time step can help, but this becomes computationally expensive

This technique is called Forward (Explicit) Euler Integration

Binary, *n*-ary Forces

Much more interesting behaviors to be had from particles that interact

Simplest: binary forces, e.g. springs

$$\mathbf{f}_i(\mathbf{x}_i, \mathbf{x}_j) = -k_s(|\mathbf{x}_i - \mathbf{x}_j| - r_0)\frac{\mathbf{x}_i - \mathbf{x}_j}{|\mathbf{x}_i - \mathbf{x}_j|}$$

Nice example project with mass-spring systems:

- https://vimeo.com/73188339

More sophisticated models for deformable things use forces relating 3 or more particles

Integration Algorithms

Another attempt

- Update velocity with forces at next time step determined by solving a (non-)linear system
- Use new velocity to update position

Benefits

- Unconditionally stable if the system is linear!

Issues

- Solving a system at each step can become expensive
- Can introduce artificial viscous damping

This technique is called Backward (Implicit) Euler Integration

Integration Algorithms

Next attempt: A compromise

- Update velocity using current forces
- Use this updated velocity to update the position

Benefits

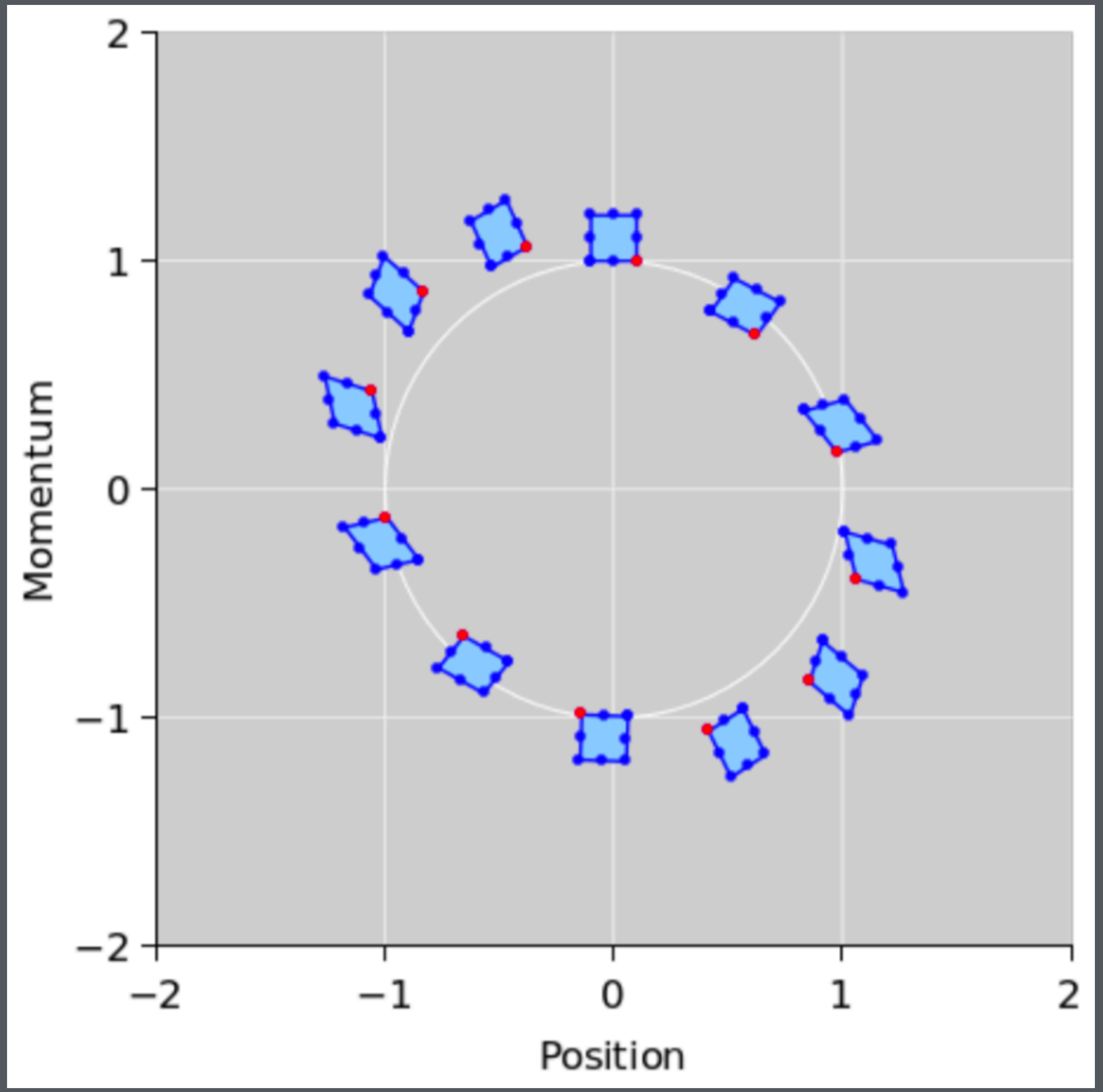
- All the speed benefits of Forward Euler, but much more stable!
- You should basically always choose this algorithm over Forward Euler

Issues

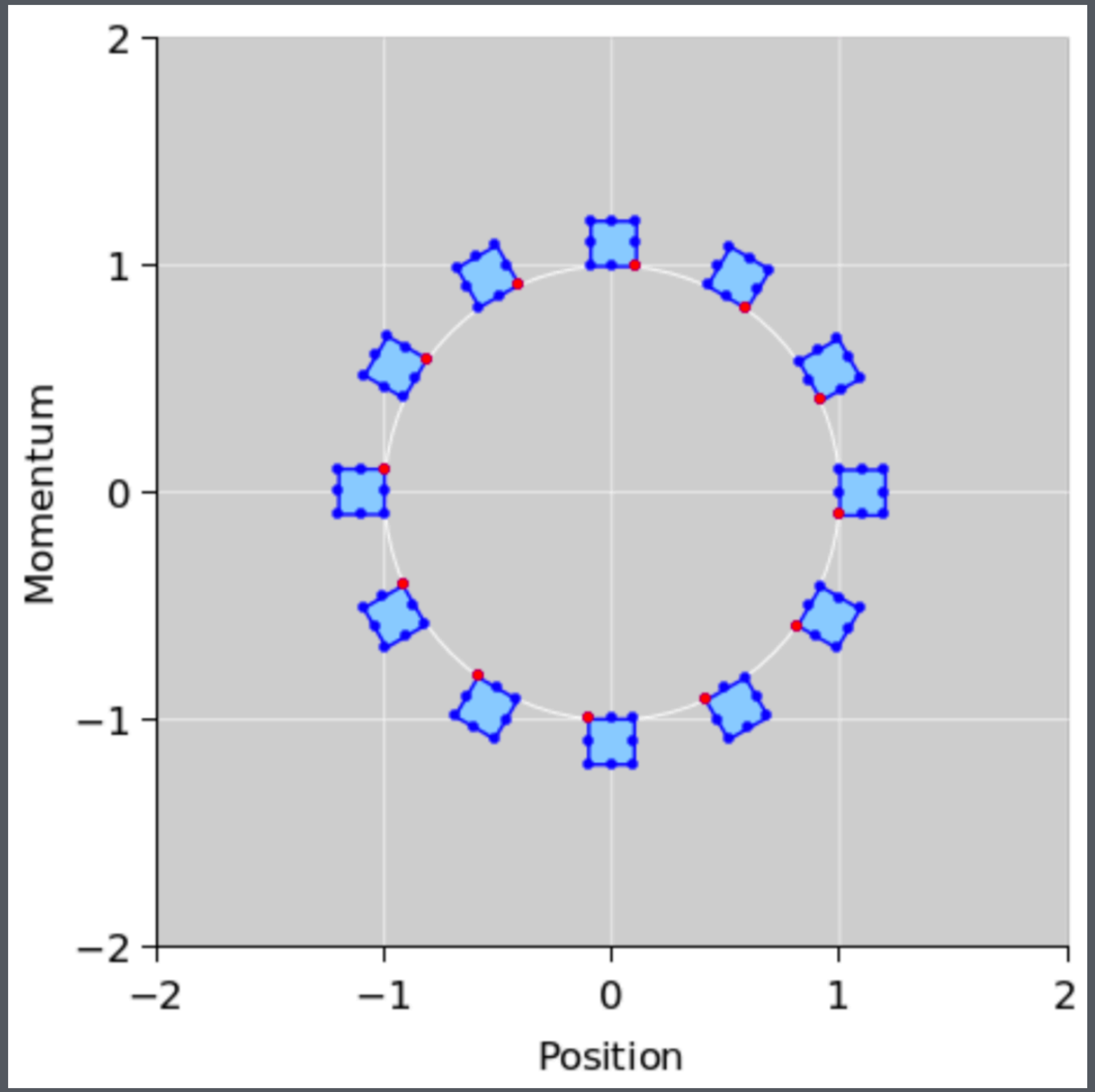
- Still not unconditionally stable, though

This technique is called Symplectic (Semi-implicit) Euler Integration

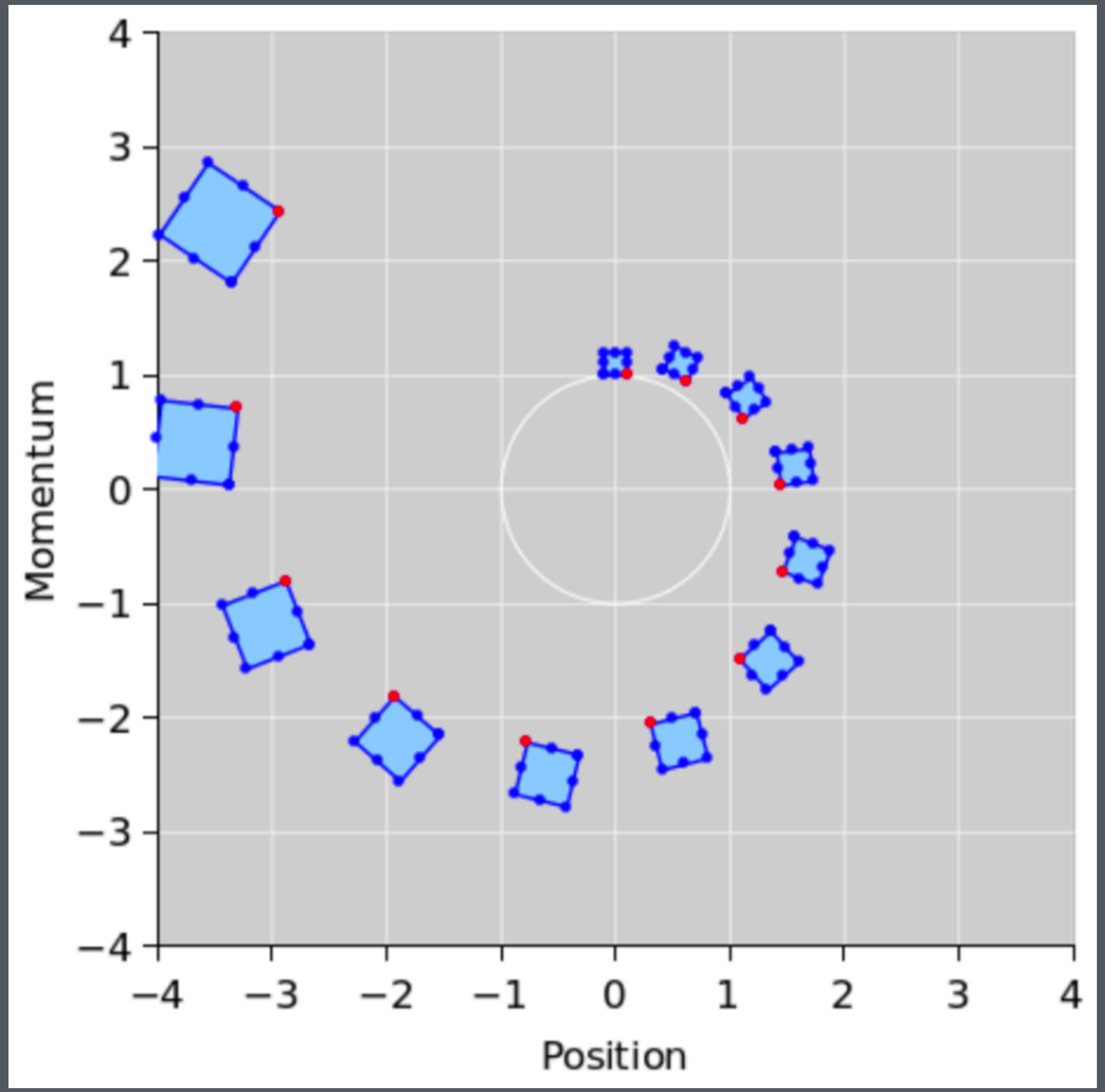
Euler variants viewed in phase space



Symplectic Euler



Exact solution



Forward Euler

Stelian Coros, CMU

Other Integration techniques

Midpoint

Newmark-β

Verlet

RK-4

Many more (complicated) schemes

- RK family
- Exponential Integrators

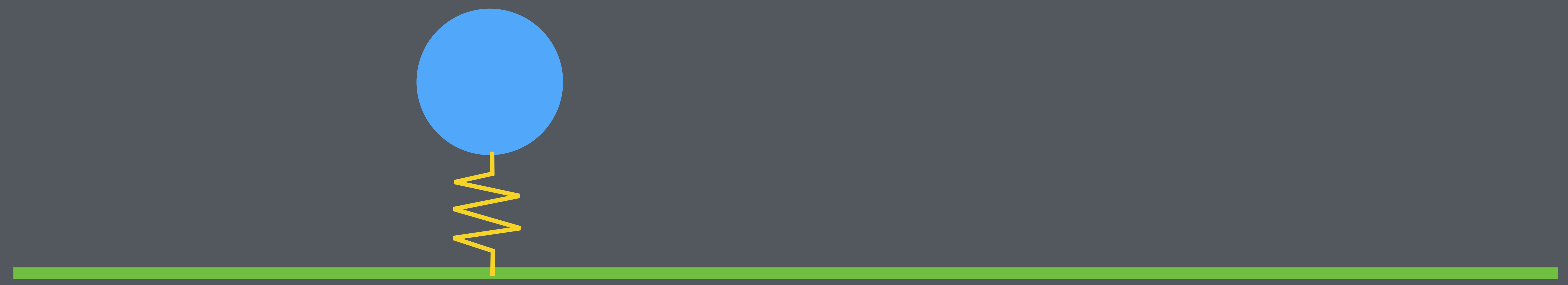
Computational Stiffness

E.g. Bead on Wire

- Can use a spring force to bind bead (particle) to a wire
- If the spring is weak, the particle may drift too far away
- If the spring is strong, we need very small time steps to ensure stability

Known as a “stiff” problem

- One stiff spring makes the whole system stiff!



Constraints

- **At the end of each step (i.e. after integration), enforce certain properties of the system**
  - e.g., the bead should not leave the wire
- **Idea: push unconstrained system towards acceptable configuration by modifying particle momentum as little as possible**

Constraint Equations

Usually of the form $C(x) = 0$ or $C(x) \geq 0$

When finding a solution, we are usually interested in the derivatives of these equations with respect to position (x)

These are similar to forces, but are non-physical

Constraint Jacobian Matrix

Collection of derivatives of constraints into a single matrix

Not necessarily square: relates *n* particle positions to *m* constraints

Similar to a stiffness matrix

Enforcing Constraints

First attempt: Apply constraint equation derivatives iteratively

Benefits

- Fast, parallelizable over particles

Issues

- Constraint application order matters!
- Convergence not guaranteed!
- Successive Over-Relaxation can help (i.e., apply a scaled version of the constraint derivative)
    - But this is finicky, finding the right scaling value can be difficult (or it might not exist)

Enforcing Constraints

Another attempt: Lagrange Multipliers

- Solve a global linear system over all constraints
- Add an extra row/column for each 1D constraint

Benefits

- Order of constraints doesn’t matter
- Solves simultaneous constraints exactly in one pass

Issues

- Non-parallelizable, global linear solve (but this can be done quickly using, e.g., conjugate gradient

Enforcing Constraints

Another attempt: Fast Manifold Projection

- Solve a linear equation over constraints to project particles to “nearest” valid position
- Iterate until convergence

Benefits

- Typically very few iterations needed; system size depends on number of constraints, not number of particles

Issues

- Again, requires a global, non-parallelizable system solve

The New Algorithm

1) **Clear forces from previous calculations**

2) **Calculate/accumulate forces for each particle**

3) **Use time integration algorithm of choice to update particle to unconstrained position**

4) **Enforce constraints with algorithm of choice**