Dynamics and Time Integration

Computer Graphics
CMU 15-462/15-662
Last time: animation

- Added motion to our model
- Interpolate keyframes
- Still a lot of work!
- Today: physically-based animation
  - often less manual labor
  - often more compute-intensive
- Leverage tools from physics
  - dynamical descriptions
  - numerical integration
- Payoff: beautiful, complex behavior from simple models
- Widely-used techniques in modern film (and games!)
Dynamical Description of Motion

“A change in motion is proportional to the motive force impressed and takes place along the straight line in which that force is impressed.”

—Sir Isaac Newton, 1687

“Dynamics is concerned with the study of forces and their effect on motion, as opposed to kinematics, which studies the motion of objects without reference to its causes.”

—Sir Wiki Pedia, 2015

(Q: Is keyframe interpolation dynamic, or kinematic?)
The Animation Equation

- Already saw the rendering equation
- What’s the animation equation?

\[ F = ma \]
The “Animation Equation,” revisited

- Well actually there are some more equations...
- Let’s be more careful:
  - Any system has a configuration $q(t)$
  - It also has a velocity $\dot{q} := \frac{d}{dt} q$
  - And some kind of mass $M$
  - There are probably some forces $F$
  - And also some constraints $g(q, \dot{q}, t) = 0$
- E.g., could write Newton’s 2nd law as $\ddot{q} = F/m$
- Makes two things clear:
  - acceleration is 2nd time derivative of configuration
  - ultimately, we want to solve for the configuration $q$
Generalized Coordinates

- Often describing systems with many, many moving pieces
- E.g., a collection of billiard balls, each with position $x_i$
- Collect them all into a single vector of generalized coordinates:

$$q = (x_0, x_1, \ldots, x_n)$$

- Can think of $q$ as a single point moving along a trajectory in $\mathbb{R}^n$
- This way of thinking naturally maps to the way we actually solve equations on a computer: all variables are often “stacked” into a big long vector and handed to a solver.
- (…So why not write things down this way in the first place?)
Generalized Velocity

- Not much more to say about generalized velocity: it’s the time derivative of the generalized coordinates!

\[
\dot{q} = (\dot{x}_0, \dot{x}_1, \ldots, \dot{x}_n)
\]

All of life (and physics) is just traveling along a curve...
Ordinary Differential Equations

- Many dynamical systems can be described via an ordinary differential equation (ODE) in generalized coordinates:

\[
\frac{d}{dt} q = f(q, \dot{q}, t)
\]

- ODE doesn’t have to describe mechanical phenomenon, e.g.,

\[
\frac{d}{dt} u(t) = \alpha u
\]

“rate of growth is proportional to value”

- Solution? \( u(t) = be^{\alpha t} \)

- Describes exponential decay \((\alpha < 1)\), or really great stock \((\alpha > 1)\)

- “Ordinary” means “involves derivatives in time but not space”

- We’ll talk about spatial derivatives (PDEs) in another lecture...
Dynamics via ODEs

- Another key example: Newton’s 2nd law!

\[ \ddot{q} = \frac{F}{m} \]

- “Second order” ODE since we take two time derivatives

- Can also write as a system of two first order ODEs, by introducing new “dummy” variable for velocity:

\[
\begin{align*}
\dot{q} &= v \\
\dot{v} &= \frac{F}{m}
\end{align*}
\]

- Splitting things up this way will make it easy to talk about solving these equations numerically (among other things)
Simple Example: Throwing a Rock

Consider a rock* of mass \( m \) tossed under force of gravity \( g \).

Easy to write dynamical equations, since only force is gravity:

\[
\ddot{q} = \frac{g}{m} \quad \text{or} \quad \dot{v} = \frac{g}{m}
\]

Solution:

\[
\begin{align*}
v(t) &= v_0 + \frac{t}{m}g \\
q(t) &= q_0 + tv_0 + \frac{t^2}{2m}g
\end{align*}
\]

*Yes, this rock is spherical and has uniform density.

(What do we need a computer for?!)
Slightly Harder Example: Pendulum

- Mass on end of a bar, swinging under gravity
- What are the equations of motion?
- Same as “rock” problem, but constrained
- Could use a “force diagram”
  - You probably did this for many hours in high school/college
  - Let’s do something new & different!
Lagrangian Mechanics

- Beautifully simple recipe:
  1. Write down kinetic energy $K$
  2. Write down potential energy $U$
  3. Write down Lagrangian $\mathcal{L} := K - U$
  4. Dynamics then given by Euler-Lagrange equation

$$\frac{d}{dt} \frac{\partial \mathcal{L}}{\partial \dot{q}} = \frac{\partial \mathcal{L}}{\partial q}$$

- Why is this useful?
  - often easier to come up with (scalar) energies than forces
  - very general, works in any kind of generalized coordinates
  - helps develop nice class of numerical integrators (symplectic)

Lagrangian Mechanics - Example

- Generalized coordinates for pendulum?
  \[ q = \theta \]
  just one coordinate: angle with the vertical direction

- Kinetic energy (mass \( m \))?
  \[ K = \frac{1}{2} I \omega^2 = \frac{1}{2} m L^2 \dot{\theta}^2 \]

- Potential energy?
  \[ U = mgh = -mgL \cos \theta \]

- Euler-Lagrange equations? (from here, just “plug and chug”—even a computer could do it!)
  \[ \mathcal{L} = K - U = m\left(\frac{1}{2} L^2 \dot{\theta}^2 + gL \cos \theta\right) \]
  \[ \frac{\partial \mathcal{L}}{\partial \dot{q}} = \frac{\partial \mathcal{L}}{\partial \theta} = mL^2 \dot{\theta} \]
  \[ \frac{\partial \mathcal{L}}{\partial q} = \frac{\partial \mathcal{L}}{\partial \theta} = -mgL \sin \theta \]

  \[ \frac{d}{dt} \frac{\partial \mathcal{L}}{\partial \dot{q}} = \frac{\partial \mathcal{L}}{\partial q} \quad \Rightarrow \quad \ddot{\theta} = -\frac{g}{L} \sin \theta \]
Solving the Pendulum

- Great, now we have a nice simple equation for the pendulum:
  \[ \ddot{\theta} = -\frac{g}{L} \sin \theta \]

- For small angles (e.g., clock pendulum) can approximate as
  \[ \ddot{\theta} = -\frac{g}{L} \theta \quad \Rightarrow \quad \theta(t) = a \cos(t \sqrt{g/L} + b) \]
  "harmonic oscillator"

- In general, there is no closed form solution!

- Hence, we must use a numerical approximation

- ...And this was (almost) the simplest system we can think of!

- (What if we want to animate something more interesting?)
Not-So-Simple Example: Double Pendulum

- Blue ball swings from fixed point; green ball swings from blue one
- Simple system... not-so-simple motion!
- Chaotic: perturb input, wild changes to output
- Must again use numerical approximation
Not-So-Simple Example: *n*-Body Problem

- Consider the Earth, moon, and sun—where do they go?
- Solution is trivial for two bodies (e.g., assume one is fixed)
- As soon as $n \geq 3$, again get chaotic solutions (no closed form)
- What if we want to simulate entire *galaxies*?

Credit: Governato et al / NASA
For animation, we want to simulate these kinds of phenomena!
Example: Flocking
Simulated Flocking as an ODE

- Each bird is a particle
- Subject to very simple forces:
  - attraction to center of neighbors
  - repulsion from individual neighbors
  - alignment toward average trajectory of neighbors
- Solve large system of ODEs (numerically!)
- Emergent complex behavior (also seen in fish, bees, ...)

Credit: Craig Reynolds (see http://www.red3d.com/cwr/boids/)
Particle Systems

- More generally, model phenomena as large collection of particles
- Each particle has a behavior described by (physical or non-physical) forces
- Extremely common in graphics/games
  - easy to understand
  - simple equation for each particle
  - easy to scale up/down
- May need many particles to capture certain phenomena (e.g., fluids)
  - may require fast hierarchical data structure (kd-tree, BVH, ...)
  - often better to use continuum model
Example: Crowds

Where are the bottlenecks in a building plan?
Example: Crowds + “Rock” Dynamics
Example: Particle-Based Fluids

Sph particle fluid
300,000 particles
71 min bake
3.5 min per renderframe

(Fluid: particles or continuum?)
Example: Granular Materials

Bell et al, “Particle-Based Simulation of Granular Materials”
Example: Molecular Dynamics

(model of melting ice crystal)
Example: Cosmological Simulation

Tomoaki et al - v²GC simulation of dark matter (~1 trillion particles)
Example: Mass-Spring System

- Connect particles $x_1$, $x_2$ by a spring of length $L_0$
- Potential energy is given by

$$U = \frac{1}{2} k (L - L_0)^2$$

Connect up many springs to describe interesting phenomena

- Extremely common in graphics/games
  - easy to understand
  - simple equation for each particle
- Often good reasons for using continuum model (PDE)
Example: Mass Spring System
Example: Mass Spring + Character
Example: Hair
Ok, I’m convinced. So how do we solve these things numerically?
Numerical Integration

- Key idea: replace derivatives with differences
- In ODE, only need to worry about derivative in time
- Replace time-continuous function \( q(t) \) with samples \( q_k \) in time

\[
\frac{d}{dt} q(t) = f(q(t))
\]

\[
\frac{q_{k+1} - q_k}{\tau} = f(q)
\]

new configuration (unknown—want to solve for this!)

current configuration (known)

“time step,” i.e., interval of time between \( q_k \) and \( q_{k+1} \)

Wait... where do we evaluate the velocity function? At the new or old configuration?
Forward Euler

- Simplest scheme: evaluate velocity at current configuration
- New configuration can then be written *explicitly* in terms of known data:

\[ q_{k+1} = q_k + \tau f(q_k) \]

- Very intuitive: walk a tiny bit in the direction of the velocity
- Unfortunately, not very *stable*—consider pendulum:

Where did all this extra energy come from?

...gradually moves faster & faster!
Forward Euler - Stability Analysis

- Let’s consider behavior of forward Euler for simple linear ODE:
  \[ \dot{u} = -\alpha u, \quad \alpha > 0 \]

- Importantly: \( u \) should decay (exact solution is \( u(t) = e^{-\alpha t} \))

- Forward Euler approximation is
  \[
  u_{k+1} = u_k - \tau \alpha u_k \\
  = (1 - \tau \alpha) u_k
  \]

- Which means after \( n \) steps, we have
  \[
  u_n = (1 - \tau \alpha)^n u_0
  \]

- Decays only if \( |1-\tau \alpha| < 1 \), or equivalently, if \( \tau < 2/\alpha \)

- In practice: need very small time steps if \( \alpha \) is large (“stiff system”)

Backward Euler

- Let’s try something else: evaluate velocity at \textit{next} configuration
- New configuration is then \textit{implicit}, and we must solve for it:

\[ q_{k+1} = q_k + \tau f(q_{k+1}) \]

- Much harder to solve, since in general \( f \) can be very nonlinear!
- Pendulum is now stable... perhaps \textit{too} stable?

starts out slow...

...and eventually stops moving completely.

Where did all the energy go?
Backward Euler - Stability Analysis

- Again consider a simple linear ODE:
  \[ \dot{u} = -au, \quad a > 0 \]

- Remember: \( u \) should decay (exact solution is \( u(t) = e^{-at} \))

- Backward Euler approximation is
  \[ \frac{u_{k+1} - u_k}{\tau} = -au_{k+1} \]
  \[ \iff \quad u_{k+1} = \frac{1}{1+\tau a} u_k \]

- Which means after \( n \) steps, we have
  \[ u_n = \left( \frac{1}{1+\tau a} \right)^n u_0 \]

- Decays if \( |1+\tau a| > 1 \), which is always true!

- \( \Rightarrow \) Backward Euler is unconditionally stable for linear ODEs
Symplectic Euler

- Backward Euler was stable, but we also saw (empirically) that it exhibits *numerical damping* (damping not found in original eqn.)
- Nice alternative is symplectic Euler
  - update velocity using current configuration
  - update configuration using *new* velocity
- Easy to implement; used often in practice (or leapfrog, Verlet, ...)
- Pendulum now conserves energy *almost exactly*, forever:

starts out slow...

...and keeps on ticking.

(Proof? The analysis is not quite as easy...)

CMU 15-462/662
Numerical Integrators

- Barely scratched the surface
- Many different integrators
- Why? Because many notions of “good”:
  - stability
  - accuracy
  - consistency/convergence
  - conservation, symmetry, ...
  - computational efficiency (!)
- No one “best” integrator—*pick the right tool for the job!*
- Could do (at least) an entire course on time integration...
- Great book: Hairer, Lubich, Wanner
Computational Differentiation

- So far, we’ve been taking derivatives by hand
- Very often in simulation, need to differentiate *extremely complicated* functions (e.g., potential energy, to get forces)
- Several different techniques:
  - keep doing it by hand! (laborious & error prone, but potentially fast)
  - numerical differentiation (simple to code, but usually poor accuracy)
  - automatic differentiation (bigger code investment, better accuracy)
  - symbolic differentiation (can help w/ “by-hand”, often messy results)
  - geometric differentiation (sometimes simplifies “by hand” expressions)
Review: Derivatives

- Suppose I have a function $f : \mathbb{R} \to \mathbb{R}; x \mapsto f(x)$

- Q: How do I define its first derivative with respect to $x$, at $x_0$?

  $$f'(x_0) := \lim_{\epsilon \to 0} \frac{f(x_0 + \epsilon) - f(x_0)}{\epsilon}$$

- In dynamical simulation, often need to consider functions
  
  $$f : \mathbb{R}^n \to \mathbb{R}; q \mapsto f(q) \quad \text{(e.g., potential)}$$

- Directional derivative looks a lot like ordinary derivative:

  $$D_X f(q_0) := \lim_{\epsilon \to 0} \frac{f(q_0 + \epsilon X) - f(q_0)}{\epsilon} \quad \text{(Q: is } D_X f \text{ vector or scalar?)}$$

- Gradient is vector $\nabla f$ that yields $D_X f$ when you take inner product:

  $$\langle \nabla f(q_0), X \rangle = D_X f(q_0) \quad \text{(e.g., gradient of potential is force)}$$
Numerical Differentiation

- Taking all those derivatives by hand is a lot of work! (Especially if you're just developing/debugging)

- Idea: replace derivatives with differences (as we did w/ time):

  \[ f'(x_0) \Rightarrow \frac{f(x_0 + h) - f(x_0)}{h} \]

- But how do we pick \( h \)?
  - Smaller is better... right?
  - Not always! Must be careful.
  - Can also be expensive!

  e.g., what if \( f \) were some kind of radiance integral?

But how do we pick \( h \)?
- Smaller is better... right?
- Not always! Must be careful.
- Can also be expensive!

  e.g., what if \( f \) were some kind of radiance integral?
Automatic Differentiation

- Completely different idea: do arithmetic simultaneously on a function and its derivative.
- I.e., rather than work with values $f$, work with tuples $(f, f')$
- Use chain rule to determine rules for manipulating tuples
- Example function: $f(x) = ax^2$
- Suppose we want the value and derivative at $x=2$
- Start with the tuple $(x, \frac{\partial}{\partial x} x) \big|_{x=2} = (2, 1)$
- How do we multiply tuples? $(u, u') \ast (v, v') = (uv, uv' + vu')$
- So, squaring our tuple yields $(2, 1) \ast (2, 1) = (4, 4)$
- And multiplying by $a$ scales the value and derivative: $(4a, 4a)$
- Pros: good accuracy, reasonably fast
- Cons: have to redefine all our arithmetic operators!
Symbolic Differentiation

- Yet another approach (though related to automatic one...)
- Build explicit tree representing expression
- Apply transformations to obtain derivative
- Pros: only needs to happen once!
- Cons: serious development investment
- But, can often use existing tools
  - Mathematica, Maple, etc.
- Current systems not great with vectors, 3D
- Often produce unnecessarily complex formulae...

\[
\frac{\sin(-z)}{x + 5y}
\]
Geometric Differentiation

- Sometimes symbolic differentiation misses the “big picture”
- E.g., gradient of triangle area w.r.t. vertex position $p$

**Mathematica output:**

\[
\frac{\partial}{\partial p} A = \frac{1}{2} N \times e
\]

\[
(2 (b2 - c2) (-b2 c1 + a2 (-b1 + c1) + a1 (b2 - c2) + b1 c2) + 2 (b3 - c3) (-b3 c1 + a3 (-b1 + c1) + a1 (b3 - c3) + b1 c3))/(4 Sqrt((a2 b1 - a1 b2 - a2 c1 + b2 c1 + a1 c2 - b1 c2)^2 + (a3 b1 - a1 b3 - a3 c1 + b3 c1 + a1 c3 - b1 c3)^2 + (a3 b2 - a2 b3 - a3 c2 + b3 c2 + a2 c3 - b2 c3)^2)),
\]

\[
(2 (b1 - c1) (a2 (b1 - c1) + b2 c1 - b1 c2 + a1 (-b2 + c2)) + 2 (b3 - c3) (-b3 c2 + a3 (-b2 + c2)) + a2 (b3 - c3) + b2 c3))/(4 Sqrt((a2 b1 - a1 b2 - a2 c1 + b2 c1 + a1 c2 - b1 c2)^2 + (a3 b1 - a1 b3 - a3 c1 + b3 c1 + a1 c3 - b1 c3)^2 + (a3 b2 - a2 b3 - a3 c2 + b3 c2 + a2 c3 - b2 c3)^2)),
\]

\[
(2 (b1 - c1) (a3 (b1 - c1) + b3 c1 - b1 c3 + a1 (-b3 + c3)) + 2 (b2 - c2) (a3 (b2 - c2) + b3 c2 - b2 c3 + a2 (-b3 + c3)))/(4 Sqrt((a2 b1 - a1 b2 - a2 c1 + b2 c1 + a1 c2 - b1 c2)^2 + (a3 b1 - a1 b3 - a3 c1 + b3 c1 + a1 c3 - b1 c3)^2 + (a3 b2 - a2 b3 - a3 c2 + b3 c2 + a2 c3 - b2 c3)^2))
\]
Not Covered: Contact Mechanics

Smith et al, “Reflections on Simultaneous Impact”
Coming up next: Optimization