Notes

- Review IK
- Start summary of course
- Evaluations
- Will get into water simulations on Monday, more summary

Inverse Kinematics

- IK is just a tool for helping you set joint angles
- FK computes positions and orientations of links given the joint angles
- IK solves for joint angles that give you specific positions/orientations
- Could be used interactively to pose characters
- Could be used automatically to maintain constraints (e.g. fixing footskate)
- Analytic formulas preferable if possible
  - Two link example
  - Numerical methods can work too

Jacobian

- Write down formula for position of end-effector in terms of joint angles:
  \[
  \begin{pmatrix}
  x(\theta_1, \theta_2, \theta_3, \theta_4) \\
  y(\theta_1, \theta_2, \theta_3, \theta_4)
  \end{pmatrix}
  \]
- Jacobian is matrix of partial derivatives w.r.t. angles
  \[
  J = \begin{pmatrix}
  \frac{\partial x}{\partial \theta_1} & \frac{\partial x}{\partial \theta_2} & \frac{\partial x}{\partial \theta_3} & \frac{\partial x}{\partial \theta_4} \\
  \frac{\partial y}{\partial \theta_1} & \frac{\partial y}{\partial \theta_2} & \frac{\partial y}{\partial \theta_3} & \frac{\partial y}{\partial \theta_4}
  \end{pmatrix}
  \]

Linearization

- Jacobian shows how position is perturbed if you perturb the angles (a small amount): \(dx = J \, d\theta\)
- So we can solve for the \(d\theta\) that gives us the \(dx\) we want: \(d\theta = J^{-1}(x_{\text{target}} - x)\)
- But \(J\) is typically rectangular, with no inverse
  - Too many degrees of freedom for there to be a unique solution
  - Also, sometimes there is no solution...
Jacobian transpose algorithm

- Instead of $J^{-1}$ use $J^T$
- Well defined, makes some sense
  - The joints with the biggest change are the ones that (roughly speaking) have the largest effect on the end-effector
  - Helps to minimize total change in angles
- But could be slow to converge (if we’re close, we really prefer something more like $J^{-1}$)
  - Technically: this is related to something called “steepest descent”, which works but can be slow

Regularized Least Squares

- Look at least-squares problem: find $d\theta$ that minimizes $|x_{\text{target}} - x|^2 = |d|^2$
  - That is, minimize $|Jd\theta - d|^2$
    - Which is just $d\theta^T J^T J d\theta - 2d\theta^T J^T d + d^T d$
    - Take the gradient w.r.t. $d\theta$ and solve equal to zero: $J^T J d\theta = J^T d$
    - Called “the normal equations”
    - Problem: $J^T J$ is singular
- Regularized least squares: add a scaled identity to singular matrix
  - Solve $\left( \beta J^T J + (1 - \beta) |J|^2 I \right) d\theta = J^T d$

Other issues

- Note: limit as regularization parameter goes to zero is “pseudo-inverse” $J^+$
  - Also equal to $J^T (J J^T)^{-1}$ when it exists
- In singular cases, may have $J^T d = 0$ even though $d$ is not zero
  - E.g. arm fully extended the wrong way
  - Can detect this, bump one or more angles to avoid singularity
- For big steps, linearization could be unreliable - use “line search”
  - Scale down the size of $d\theta$ until you verify it’s an improvement

Other algorithms

- Cyclic Coordinate Descent
  - Even simpler than Jacobian method
  - Adjust first angle to get end-effector as close as you can to target
  - Then adjust second angle, and third, etc.
  - Repeat until close enough
  - Can be very slow, and searches aren’t usually as simple
- Phrase it as optimization
  - Minimize distance to target: regularized least-squares corresponds to “regularized Gauss-Newton”
  - But can add additional stuff - total change in angles, penalties for violating joint constraints, ...
Summary: The Basics

- Perception of Motion
- Traditional Animation Principles
- Keyframes, Layering
- Motion Curves
- Splines
- Parameterizing and Retiming Motion Curves
- Kinematics, DOF
- Forward Kinematics
- Inverse Kinematics
- Character Rigging

Summary: Rendering

- Compositing
- Sampling
  - motion blur, antialiasing, depth-of-field
- REYES
- Shading
- Differential Rendering
- Image-Based Lighting
- Camera Control
  - Parameterization, path-following, through-the-lens constraints
- Match Move

Summary: Basic Particles

- Particle Systems
- 1st and 2nd Order Motion
- Particle Rendering
- Implicit Surfaces
- Geometric Acceleration Structures
- Spring Forces
  - Elastic objects, liquids
- Noise
- Time Integration
  - Accuracy, stability; explicit and implicit methods

Summary: Geometry & Collisions

- Geometry questions:
  - Is particle inside/outside?
  - Does particle trajectory cross?
  - Object normal at some point on surface?
  - Distance/direction to surface in space?
- Plane, Sphere, Heightfield, Triangle Mesh, Implicit Surface
- Repulsion Fields
- Spring-like Forces, Damping, Time Steps
- Newtonian Collisions, Relative Velocity
- Friction
- Collision Resolution Algorithm
Summary: Advanced Dynamics

- Flocking Behaviour
- Optimized Control
- Rigid Body Dynamics
  - World/Object space, centre of mass, angular velocity, angular momentum, torque, inertia tensor, ...
- Quaternions and Rotations
- Rigid Body Collisions
- Lagrangian Dynamics
  - Handling constraints with reduced coordinates
  - Articulated rigid bodies

Summary: Deformations, Mocap

- Morphing, Image Warping
- Free-Form Deformations
- Character Skinning
  - Skeletal Subspace Deformations
  - Solving SSD problems
- Motion Capture Process
- Foot skate clean-up
- Motion Graphs
  - Defining, traversing, searching

Summary: Advanced Character Motion

- Spacetime Constraints
- Motion Warping
- Motion Parameterization
- Fourier Analysis
  - Applications to retiming, understanding noise
- Motion Texturing
- Motion Controllers