Physics in Games

The Pedagogical Problem

- Physics simulation is a **very** complex topic
  - No way I can address this in a few lectures
  - Could spend an entire course talking about it
  - **CS 5643**: Physically Based Animation
- This is why we have **physics engines**
  - Libraries that handle most of the dirty work
  - But you have to understand how they work
  - **Examples**: Box2D, Bullet, PhysX

Approaching the Problem

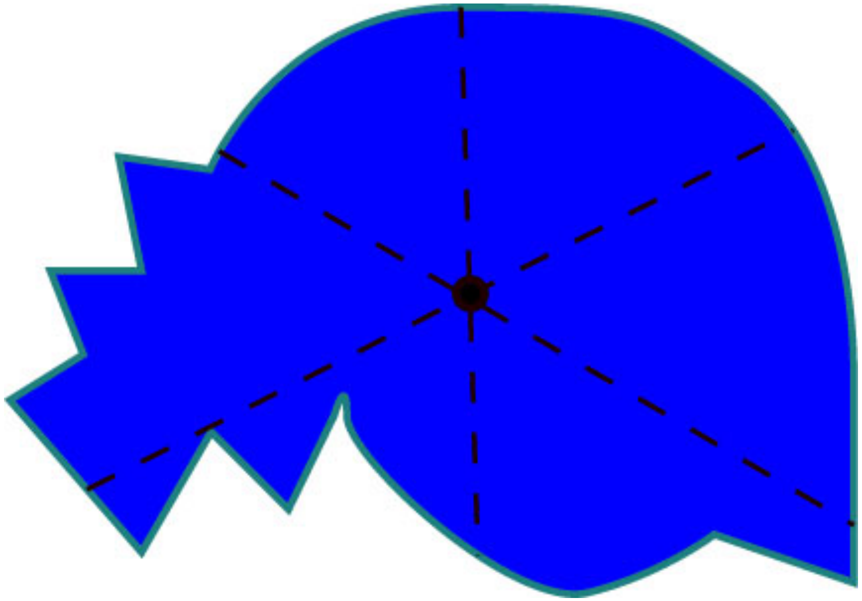
- Want to start with the **problem description**
  - Squirrel Eiserloh's *Problem Overview* slides
  - http://www.essentialmath.com/tutorial.htm
- Will help you understand the Engine APIs
  - Understand the limitations of physics engines
  - Learn where to go for other solutions
- Will cover Box2D API next time in depth

Physics in Games

- **Moving** objects about the screen
  - **Kinematics**: Motion ignoring external forces (Only consider position, velocity, acceleration)
  - **Dynamics**: The effect of forces on the screen
- **Collisions** between objects
  - **Collision Detection**: Did a collision occur?
  - **Collision Resolution**: What do we do?

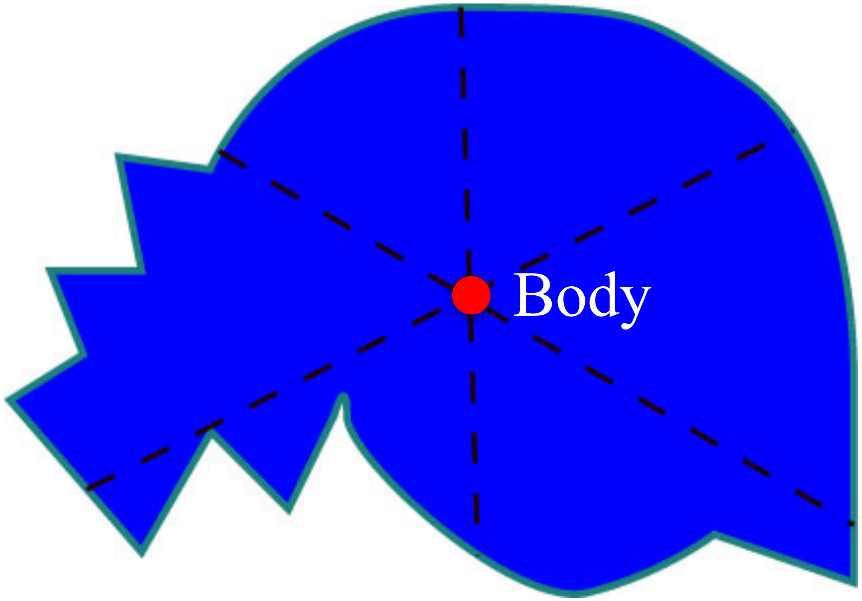
Motion: Modeling Objects

- Typically ignore **geometry**
  - Don’t worry about shape
  - Only needed for *collisions*
- Every object is a **point**
  - *Centroid*: average of points
  - Also called: *center of mass*
  - Same if density uniform
- Use **rigid body** if needed
  - Multiple points together
  - Moving one moves them all



Motion: Modeling Objects

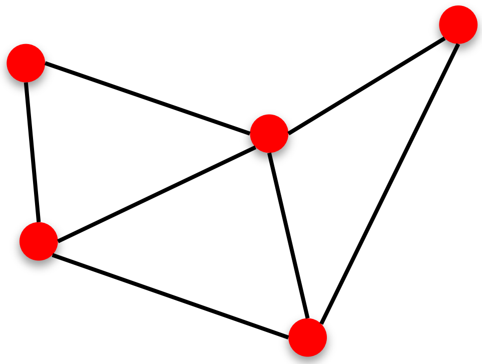
- Typically ignore **geometry**
  - Don’t worry about shape
  - Only needed for *collisions*
- Every object is a **point**
  - *Centroid*: average of points
  - Also called: *center of mass*
  - Same if density uniform
- Use **rigid body** if needed
  - Multiple points together
  - Moving one moves them all



Motion: Modeling Objects

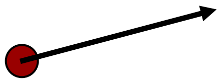
- Typically ignore **geometry**
  - Don’t worry about shape
  - Only needed for *collisions*
- Every object is a **point**
  - *Centroid*: average of points
  - Also called: *center of mass*
  - Same if density uniform
- Use **rigid body** if needed
  - Multiple points together
  - Moving one moves them all

Rigid Body



Time-Stepped Simulation

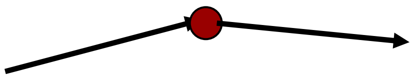
- Physics is **time-stepped**
  - Assume velocity is constant (or the acceleration is)
  - Compute the position
  - Move for next frame



- Movement is very linear
  - Piecewise approximations
  - Remember your calculus
- Smooth = smaller steps
  - More frames a second?

Time-Stepped Simulation

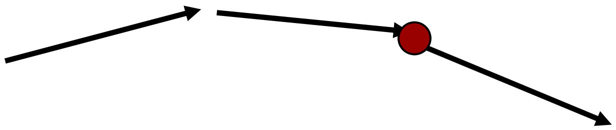
- Physics is **time-stepped**
  - Assume velocity is constant (or the acceleration is)
  - Compute the position
  - Move for next frame



- Movement is very linear
  - Piecewise approximations
  - Remember your calculus
- Smooth = smaller steps
  - More frames a second?

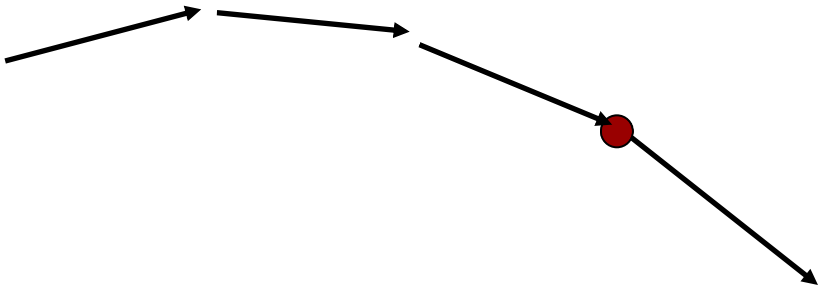
Time-Stepped Simulation

- Physics is **time-stepped**
  - Assume velocity is constant (or the acceleration is)
  - Compute the position
  - Move for next frame
- Movement is very linear
  - Piecewise approximations
  - Remember your calculus
- Smooth = smaller steps
  - More frames a second?



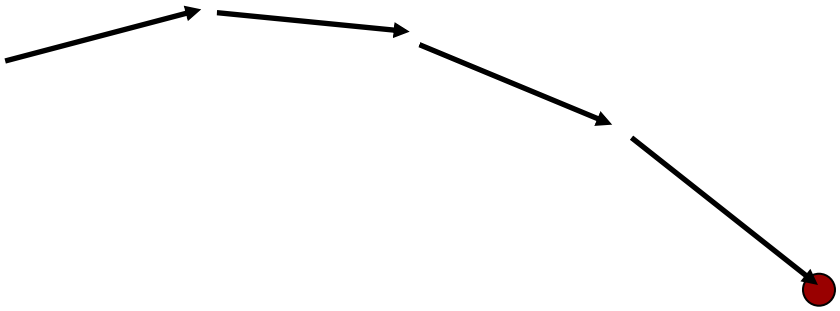
Time-Stepped Simulation

- Physics is **time-stepped**
  - Assume velocity is constant (or the acceleration is)
  - Compute the position
  - Move for next frame
- Movement is very linear
  - Piecewise approximations
  - Remember your calculus
- Smooth = smaller steps
  - More frames a second?



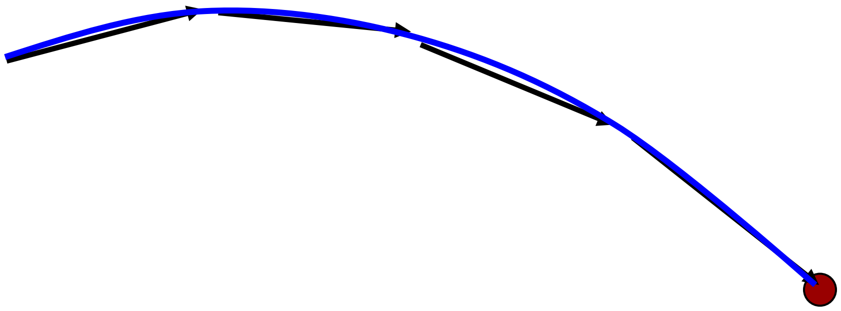
Time-Stepped Simulation

- Physics is **time-stepped**
  - Assume velocity is constant (or the acceleration is)
  - Compute the position
  - Move for next frame
- Movement is very linear
  - Piecewise approximations
  - Remember your calculus
- Smooth = smaller steps
  - More frames a second?



Time-Stepped Simulation

- Physics is **time-stepped**
  - Assume velocity is constant (or the acceleration is)
  - Compute the position
  - Move for next frame
- Movement is very linear
  - Piecewise approximations
  - Remember your calculus
- Smooth = smaller steps
  - More frames a second?



Kinematics

- **Goal**: determine an object position *p* at time *t*
  - Typically know it from a previous time
- **Assume**: constant velocity *v*
  - $p(t+\Delta t) = p(t) + v\Delta t$
  - Or $\Delta p = p(t+\Delta t) - p(t) = v\Delta t$
- **Alternatively**: constant acceleration *a*
  - $v(t+\Delta t) = v(t) + a\Delta t$ (or $\Delta v = a\Delta t$)
  - $\mathrm{p}(t+\Delta t) = p(t) + v(t)\Delta t + \frac{1}{2}a(\Delta t)^2$
  - Or $\Delta p = v_0\Delta t + \frac{1}{2}a(\Delta t)^2$

Formulas commonly in use

Kinematics

- **Goal**: determine an object position p at time t
  - Typically know it from a previous time
- **Assume**: constant velocity v
  - $p(t+\Delta t) = p(t) + v\Delta t$
  - O [illegible]

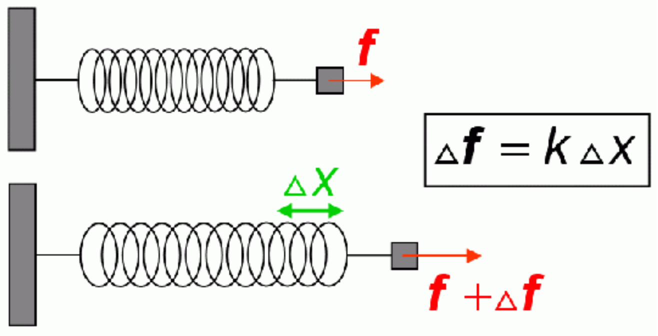
High School Physics w/o Calculus

- **Al**[illegible] constant acceleration a
  - $v(t+\Delta t) = v(t) + a\Delta t$ (or $\Delta v = a\Delta t$)
  - $p(t+\Delta t) = p(t) + v(t)\Delta t + ½a(\Delta t)^2$
  - Or $\Delta p = v_0\Delta t + ½a(\Delta t)^2$

Formulas commonly in use

Linear Dynamics

- **Forces** affect movement
  - Springs, joints, connections
  - Gravity, repulsion

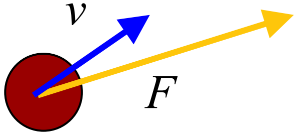


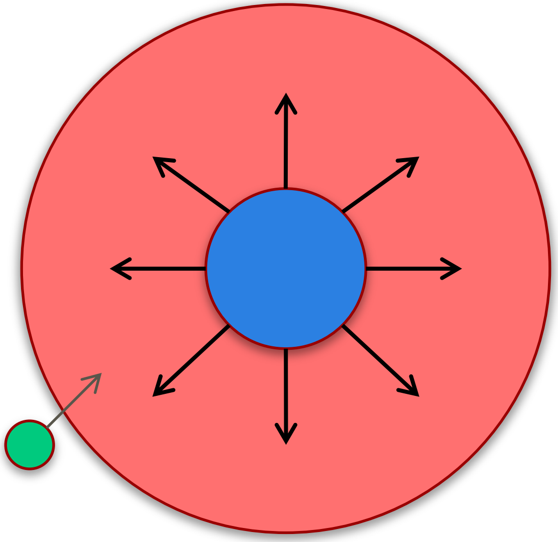
- Get velocity from forces
  - Compute current force F
  - ***F* constant entire frame**
  - Formulas:

$$\Delta a = F/m$$

$$\Delta v = F\Delta t/m$$

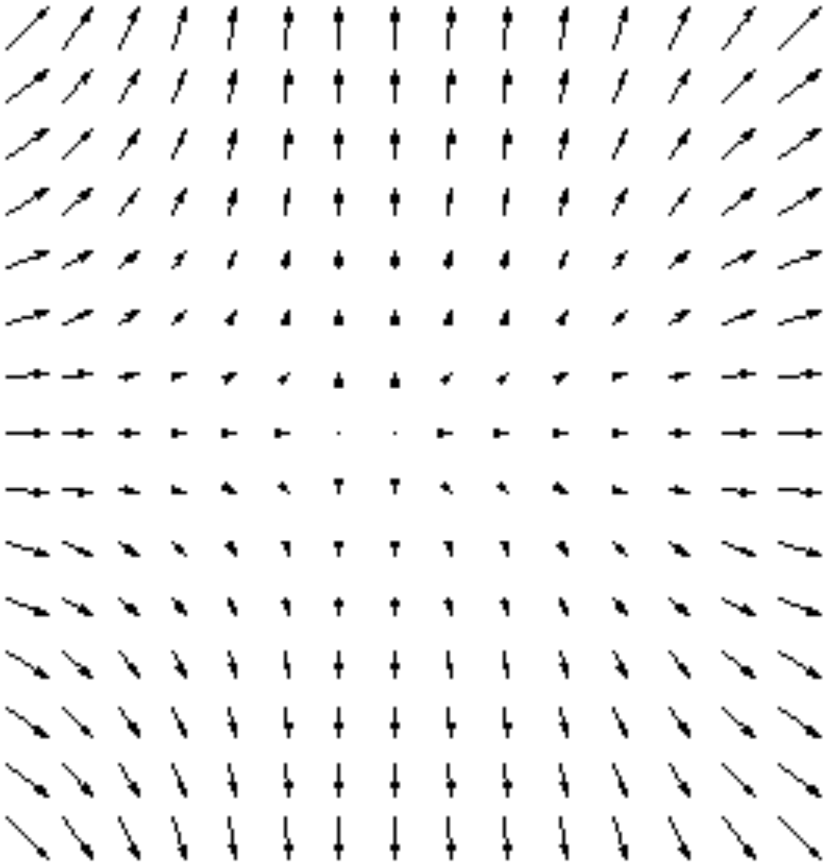
$$\Delta p = F(\Delta t)^2/m$$





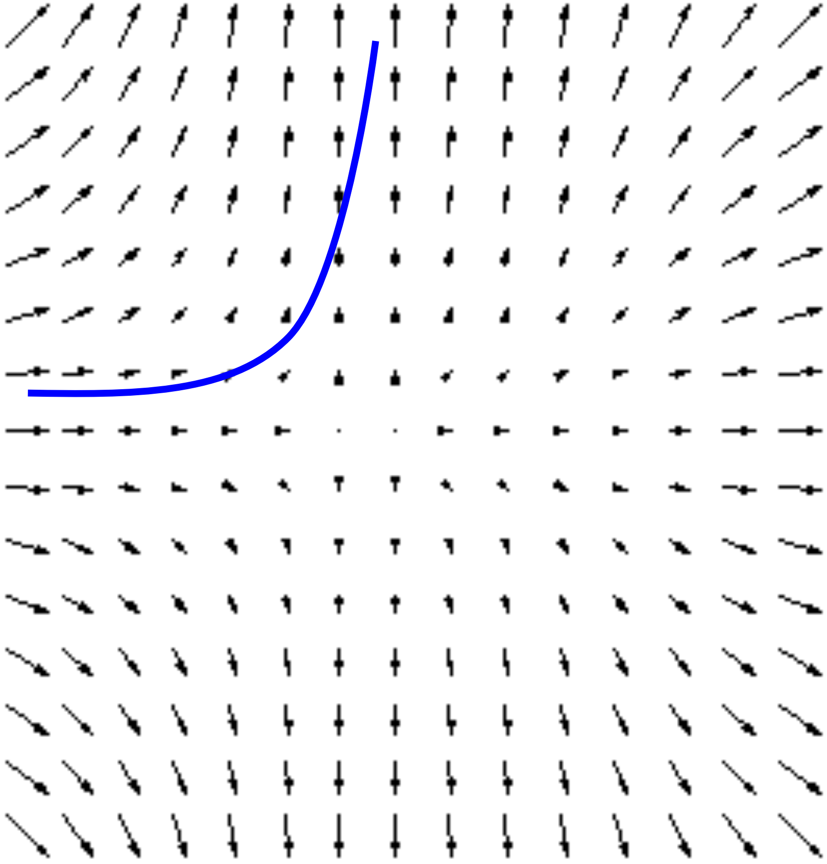
Linear Dynamics

- **Force**: $F(p,t)$
  - p: current position
  - t: current time
- Creates a **vector field**
  - Movement should follow field direction
- Update formulas
  - $a_i = F(p_i, i\Delta t)/m$
  - $v_{i+1} = v_i + a_i \Delta t$
  - $p_{i+1} = p_i + v_i \Delta t$



Linear Dynamics

- **Force**: $F(p,t)$
  - p: current position
  - t: current time
- Creates a **vector field**
  - Movement should follow field direction
- Update formulas
  - $a_i = F(p_i, i\Delta t)/m$
  - $v_{i+1} = v_i + a_i \Delta t$
  - $p_{i+1} = p_i + v_i \Delta t$

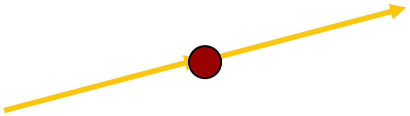


Physics Engines are DE Solvers

- Differential Equation
  - $F(p,t) = m\, a(t)$
  - $F(p,t) = m\, p''(t)$
- Euler's method:
  - $a_i = F(p_i, i\Delta t)/m$
  - $v_{i+1} = v_i + a_i \Delta t$
  - $p_{i+1} = p_i + v_i \Delta t$
- But heavily optimized

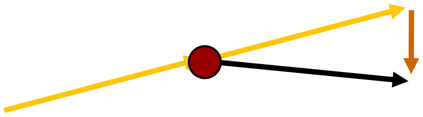
Physics Engines are DE Solvers

- Differential Equation
  - $F(p,t) = m\ a(t)$
  - $F(p,t) = m\ p''(t)$
- Euler's method:
  - $a_i = F(p_i, i\Delta t)/m$
  - $v_{i+1} = v_i + a_i \Delta t$
  - $p_{i+1} = p_i + v_i \Delta t$
- But heavily optimized



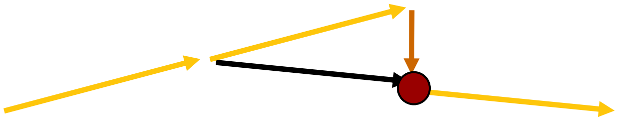
Physics Engines are DE Solvers

- Differential Equation
  - $F(p,t) = m\ a(t)$
  - $F(p,t) = m\ p''(t)$
- Euler's method:
  - $a_i = F(p_i, i\Delta t)/m$
  - $v_{i+1} = v_i + a_i\Delta t$
  - $p_{i+1} = p_i + v_i\Delta t$
- But heavily optimized



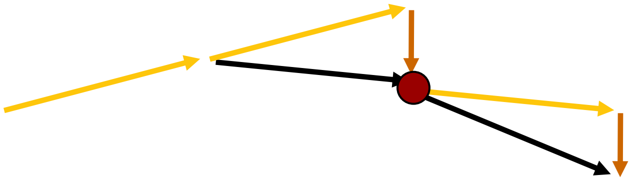
Physics Engines are DE Solvers

- Differential Equation
  - $F(p,t) = m\ a(t)$
  - $F(p,t) = m\ p''(t)$
- Euler's method:
  - $a_i = F(p_i, i\Delta t)/m$
  - $v_{i+1} = v_i + a_i\Delta t$
  - $p_{i+1} = p_i + v_i\Delta t$
- But heavily optimized



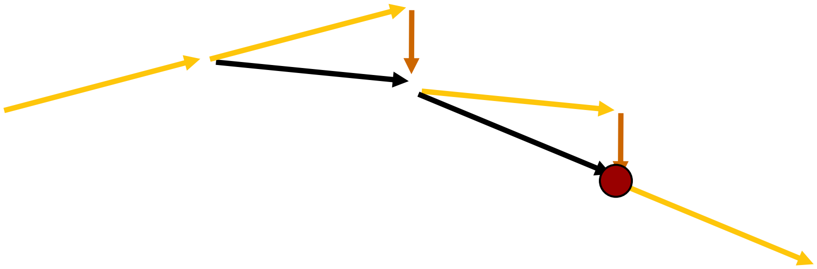
Physics Engines are DE Solvers

- Differential Equation
  - $F(p,t) = m\ a(t)$
  - $F(p,t) = m\ p''(t)$
- Euler's method:
  - $a_i = F(p_i, i\Delta t)/m$
  - $v_{i+1} = v_i + a_i \Delta t$
  - $p_{i+1} = p_i + v_i \Delta t$
- But heavily optimized



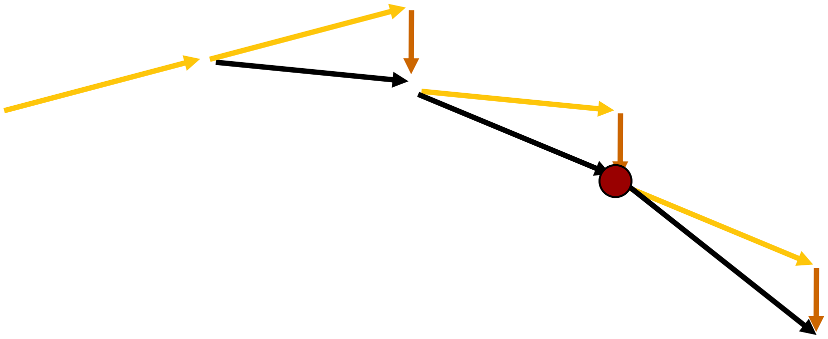
Physics Engines are DE Solvers

- Differential Equation
  - $F(p,t) = m\ a(t)$
  - $F(p,t) = m\ p''(t)$
- Euler's method:
  - $a_i = F(p_i, i\Delta t)/m$
  - $v_{i+1} = v_i + a_i \Delta t$
  - $p_{i+1} = p_i + v_i \Delta t$
- But heavily optimized



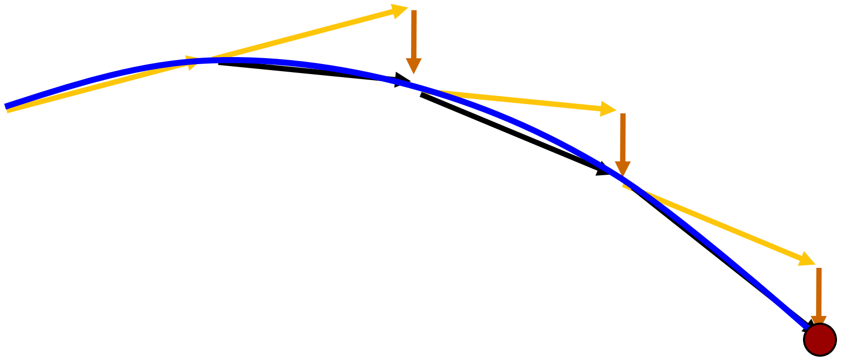
Physics Engines are DE Solvers

- Differential Equation
  - $F(p,t) = m\,a(t)$
  - $F(p,t) = m\,p''(t)$
- Euler's method:
  - $a_i = F(p_i, i\Delta t)/m$
  - $v_{i+1} = v_i + a_i \Delta t$
  - $p_{i+1} = p_i + v_i \Delta t$
- But heavily optimized



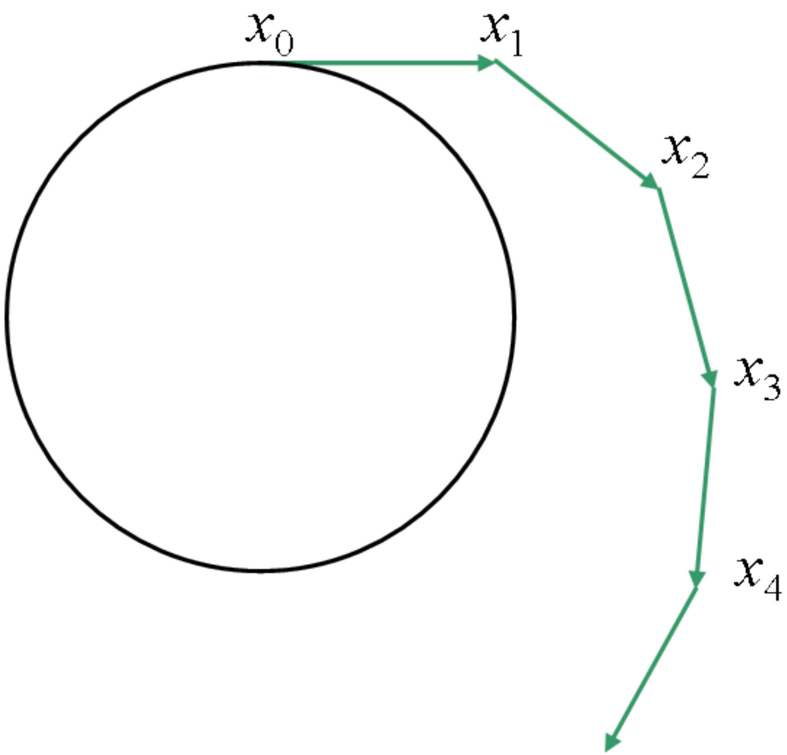
Physics Engines are DE Solvers

- Differential Equation
  - $F(p,t) = m\, a(t)$
  - $F(p,t) = m\, p''(t)$
- Euler's method:
  - $a_i = F(p_i, i\Delta t)/m$
  - $v_{i+1} = v_i + a_i \Delta t$
  - $p_{i+1} = p_i + v_i \Delta t$
- But heavily optimized



Problem with DE Solvers

- **Errors accumulate**
  - Side effect of techniques
  - Stepwise approximations
- Major problem with *orbits*
  - Move along tangent vector
  - Vector takes out of orbit
  - Gets worse over time
- Must *constrain* behavior
  - Keep movement in orbit



Dealing with Error Creep

- Classic solution: reduce the time step Δt
  - Up the frame rate (not necessarily good)
  - Perform more than one step per frame
  - Each Euler step is called an *iteration*
- **Multiple iterations per frame**
  - Let h be the length of the frame
  - Let n be the number of iterations

$$\boxed{\Delta t = h/n}$$

- Typically a parameter in your physics engine

Dealing with Error Creep

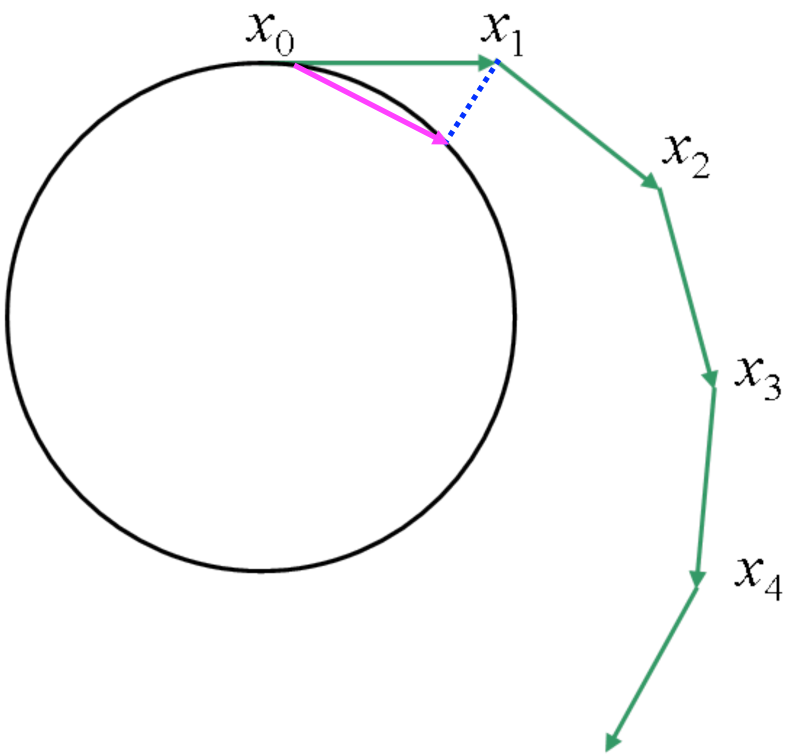
- Classic solution: reduce the time step Δt
  - Up the frame rate (not necessarily good)
  - Perform more than one step per frame
  - Each Euler step i
- **Mu**
  - Let *h* be the length of the frame
  - Let *n* be the number of iterations
- Typically a parameter in your physics engine

Still does not solve orbit problem

$$\Delta t = h/n$$

Problem with DE Solvers

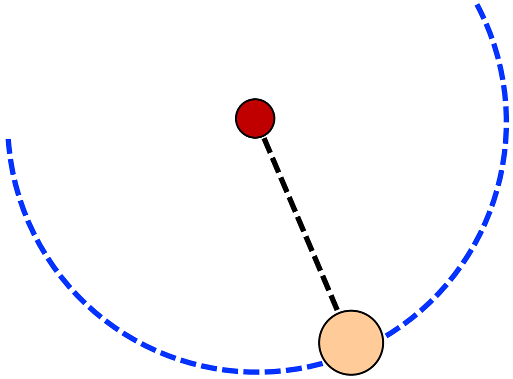
- Errors accumulate
  - Side effect of techniques
  - Stepwise approximations
- Major problem with *orbits*
  - Move along tangent vector
  - Vector takes out of orbit
  - Gets worse over time
- **Must *constrain* behavior**
  - Keep movement in orbit



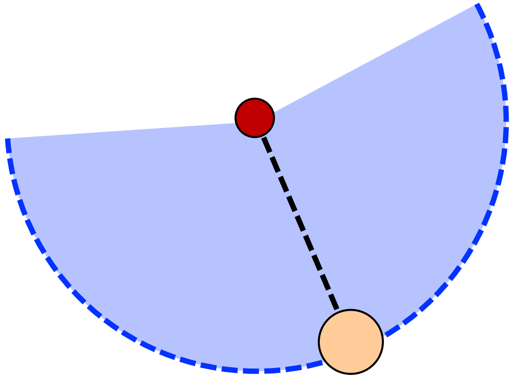
Constraint Solvers

- **Limit** object movement
  - Pos must satisfy constraint
  - Correct position if does not
- **Example:** Distance
  - Hard: Dist must be exact
  - Soft: Dist must be no more
- Other constraints
  - Contact: non-penetration
  - Restitution: bouncing
  - Friction: sliding, sticking

Hard Constraint



Soft Constraint

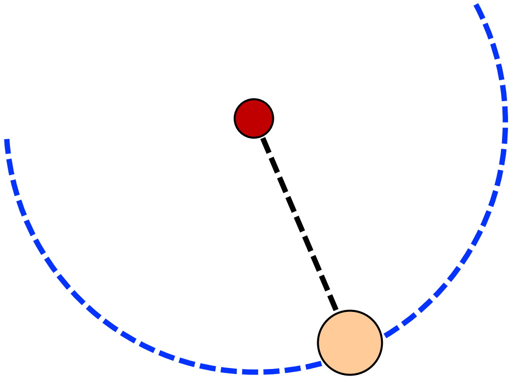


Constraint Solvers

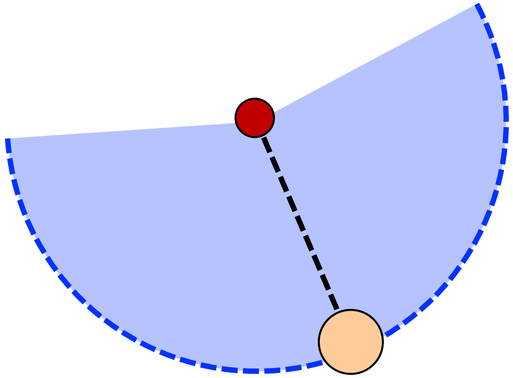
- **Limit** object movement
  - Pos must satisfy constraint
  - Correct position if does not
- **Example:** Distance
  -
  - ore
- Other constraints
  - Contact: non-penetration
  - Restitution: bouncing
  - Friction: sliding, sticking

Focus of Lab 4

Hard Constraint

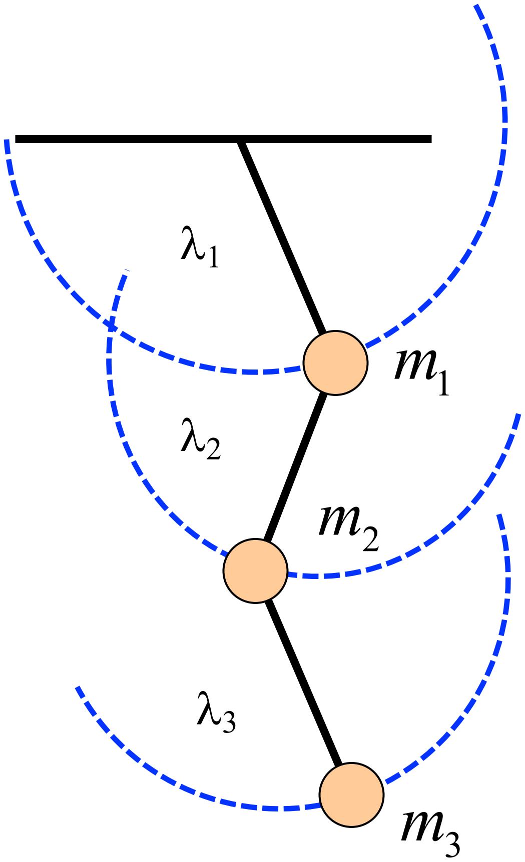


Soft Constraint



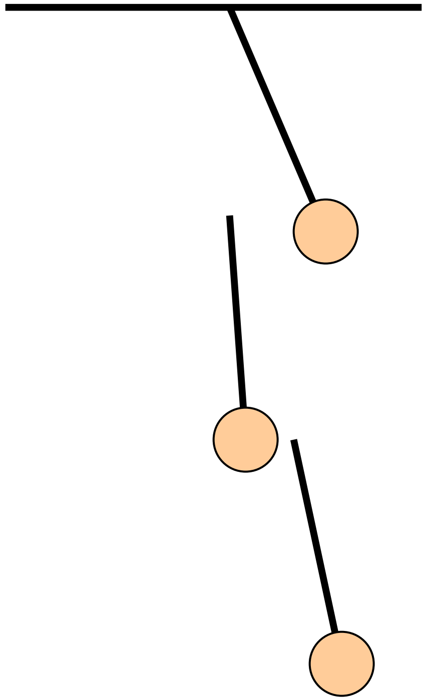
Challenge: Interconnected Constraints

- Not hard if **one** object
  - Just move it and correct
- How about *relationships*?
  - Correct an object
  - But it constrained another
  - So have to correct it and…
- When does this happen?
  - Ropes, chains
  - Box stacking



Challenge: Interconnected Constraints

- Not hard if **one** object
  - Just move it and correct
- How about *relationships*?
  - Correct an object
  - But it constrained another
  - So have to correct it and…
- When does this happen?
  - Ropes, chains
  - Box stacking

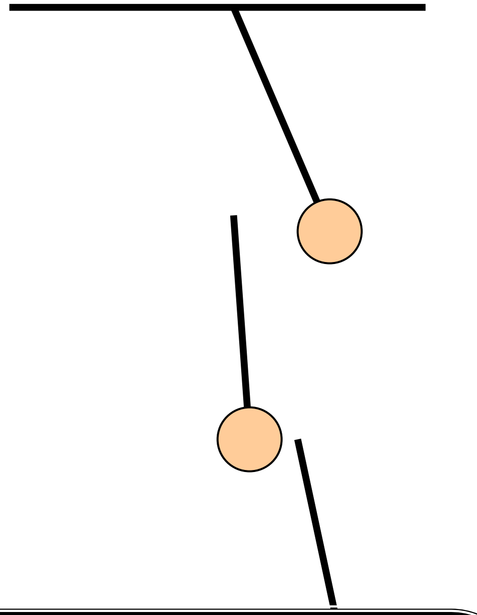


Joints Fall Apart!

Challenge: Interconnected Constraints

- Not hard if **one** object
  - Just move it and correct
- How about *relationships*?
  - Correct an object
  - But it constrained another
  - So have to correct it and…
- When does this happen?
  - Ropes, chains
  - Box stacking

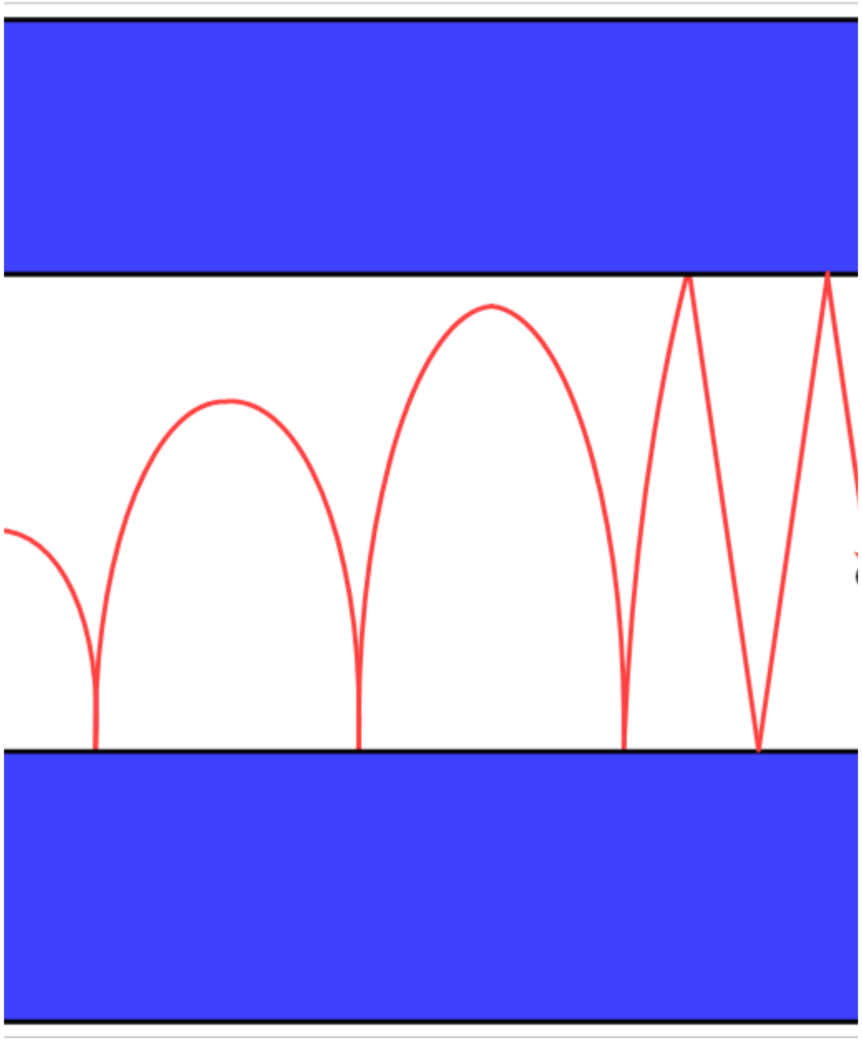
Joints Fall Apart!



Box2d is good, but not perfect

Error Accumulation: Energy

- Want energy conserved
  - Energy loss undesirable
  - Energy gain is **evil**
  - Simulations explode!
- Not always possible
  - Error accumulation!
- Need *ad hoc* solutions
  - Clamping (max values)
  - Manual **dampening**

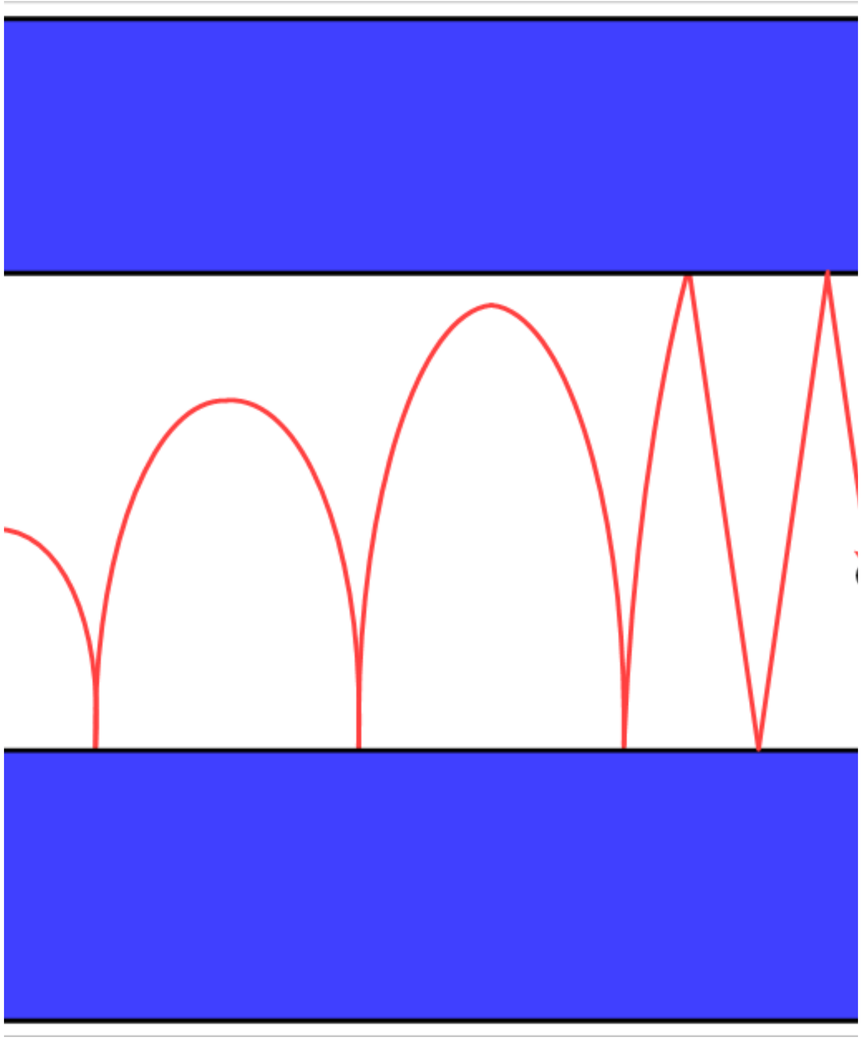


Error Accumulation: Energy

- Want energy conserved
  - Energy loss undesirable
  - Energy gain is **evil**
  - Simulations explode!

High Energy is where joints fail

- Need *ad hoc* solutions
  - Clamping (max values)
  - Manual **dampening**



Kinematics vs. Dynamics

Kinematics

- **Advantages**
  - Very simple to use
  - Non-calculus physics
- **Disadvantages**
  - Only simple physics
  - All bodies are rigid
- Old school games

Dynamics

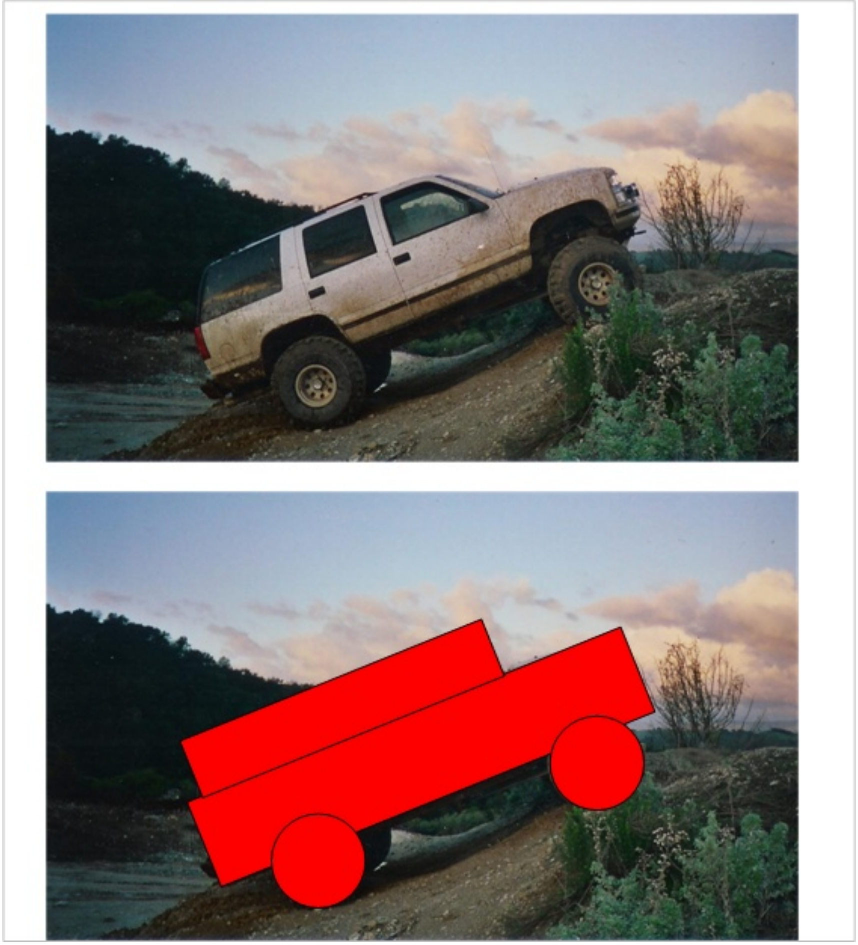
- **Advantages**
  - Complex physics
  - Non-rigid bodies
- **Disadvantages**
  - Beyond scope of course
  - Need a physics engine
- Neo-retro games

Physics in Games

- **Moving** objects about the screen
  - **Kinematics**: Motion ignoring external forces
    (Only consider position, velocity, acceleration)
  - **Dynamics**: The effect of forces on the screen
- **Collisions** between objects
  - **Collision Detection**: Did a collision occur?
  - **Collision Resolution**: What do we do?

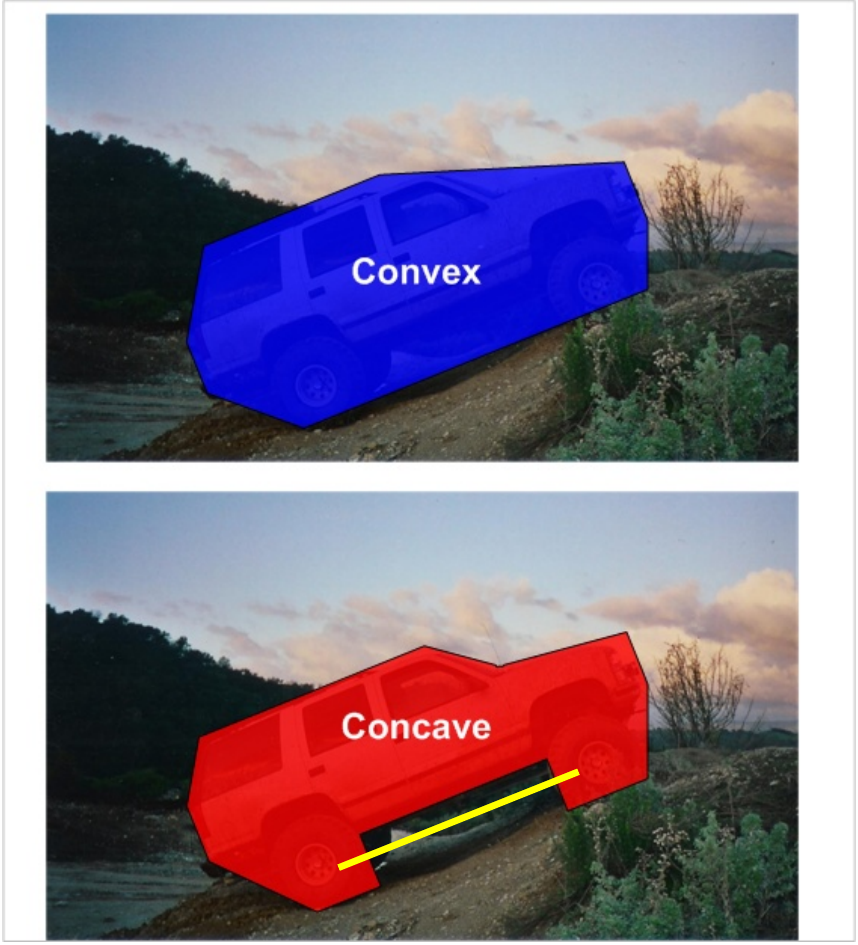
Collisions and Geometry

- Collisions need **geometry**
  - Points are not enough
  - Find *where* objects meet
- Often use **convex** shapes
  - Lines always remain inside
  - If not convex, is *concave*
- What if is not convex?
  - Break into components
  - **Triangles** always convex!



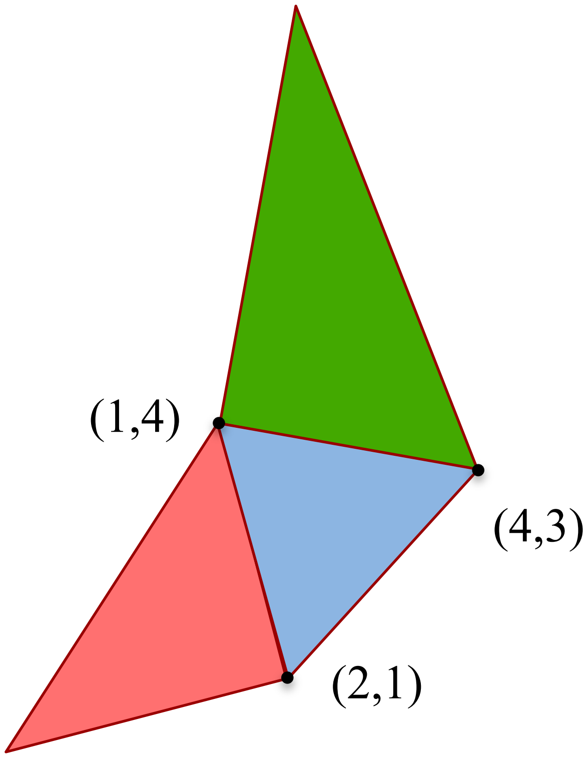
Collisions and Geometry

- Collisions need **geometry**
  - Points are not enough
  - Find *where* objects meet
- Often use **convex** shapes
  - Lines always remain inside
  - If not convex, is *concave*
- What if is not convex?
  - Break into components
  - **Triangles** always convex!



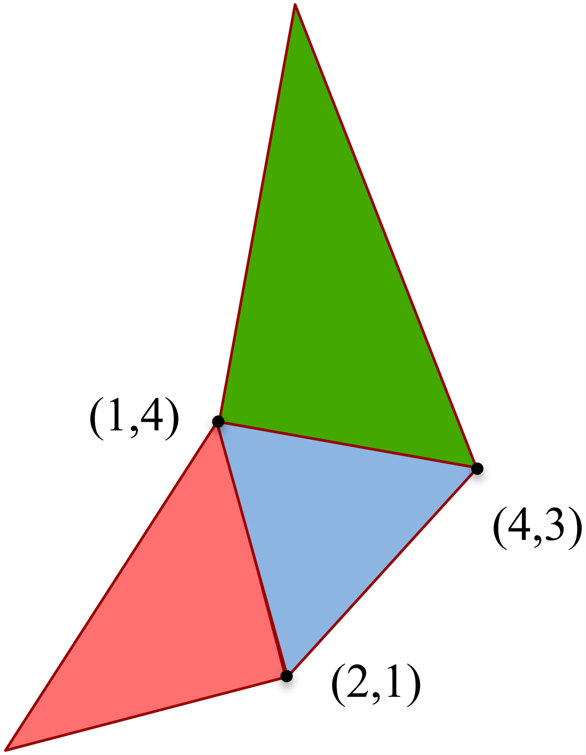
Recall: Triangles in Computer Graphics

- Everything made of **triangles**
  - Mathematically “nice”
  - Hardware support (GPUs)
- Specify with **three vertices**
  - Coordinates of corners
- Composite for complex shapes
  - Array of vertex objects
  - Each 3 vertices = triangle



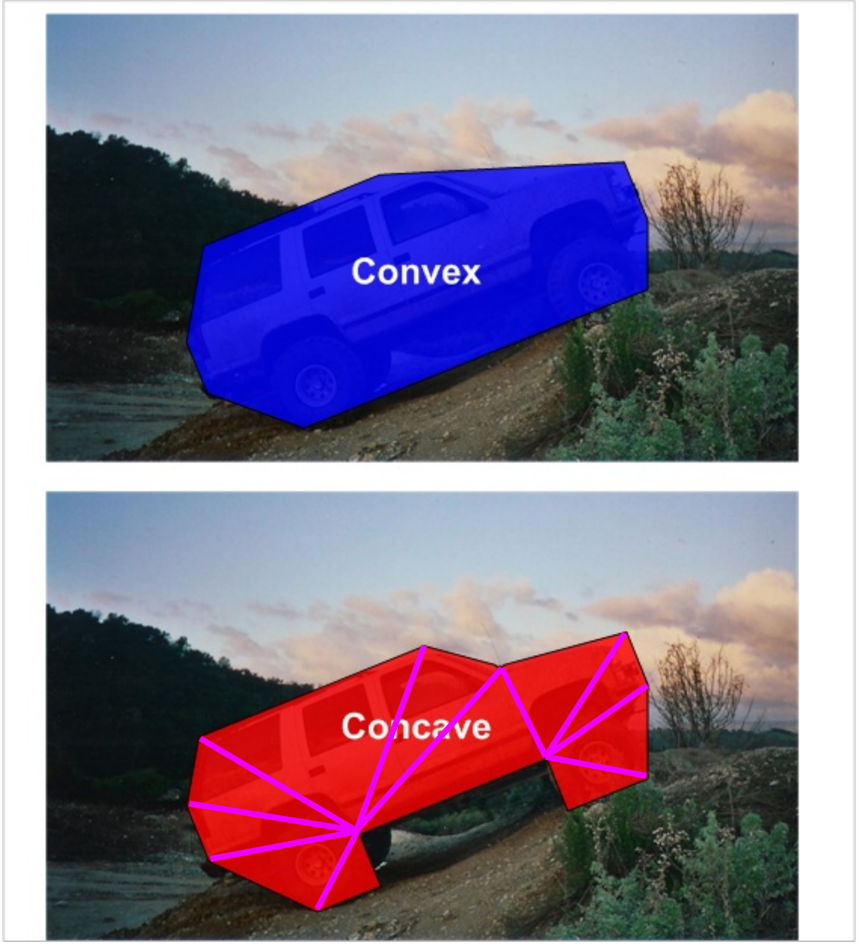
Recall: Triangles in Computer Graphics

- Everything made of **triangles**
  - Guaranteed to be convex
  - Hardware support (GPUs)
- Specify with **three vertices**
  - Coordinates of corners
- Composite for complex shapes
  - Array of vertex objects
  - Each 3 vertices = triangle



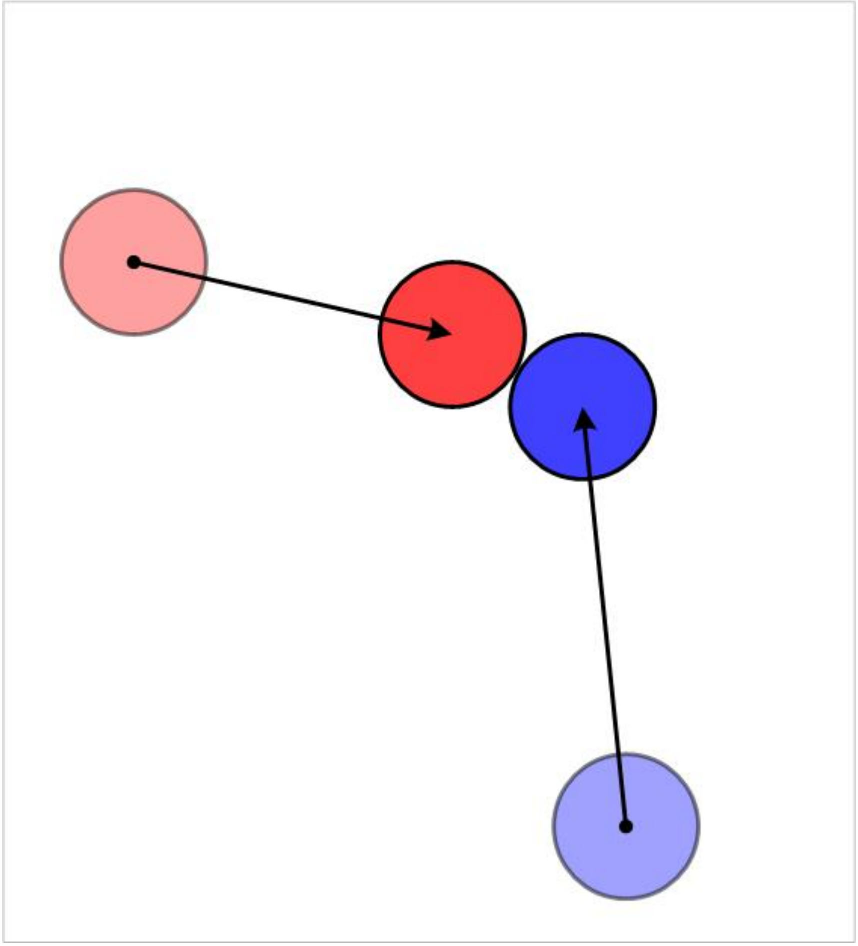
Collisions and Geometry

- Collisions need **geometry**
  - Points are not enough
  - Find *where* objects meet
- Often use **convex** shapes
  - Lines always remain inside
  - If not convex, is *concave*
- What if is not convex?
  - Break into components
  - **Triangles** always convex!



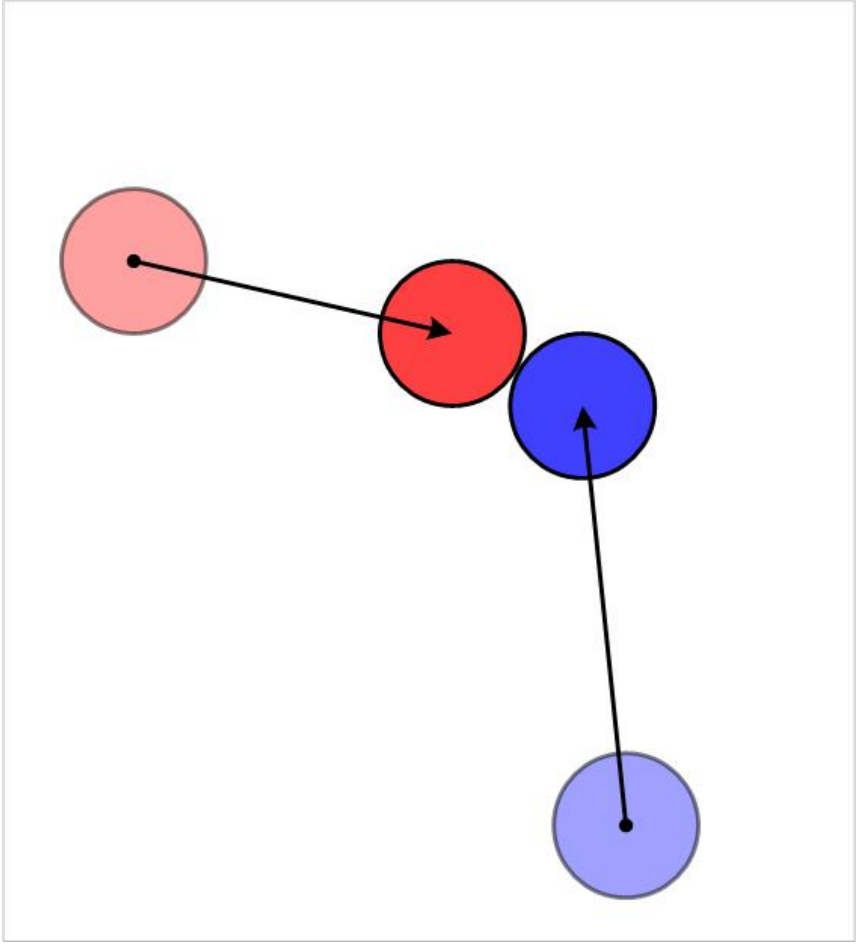
Collision Types

- **Inelastic Collisions**
  - No energy preserved
  - Stop in place ($v = 0$)
  - “Back-out” so no overlap
  - Very easy to implement
- **Elastic Collisions**
  - 100% energy preserved
  - Think billiard balls
  - Classic physics problem



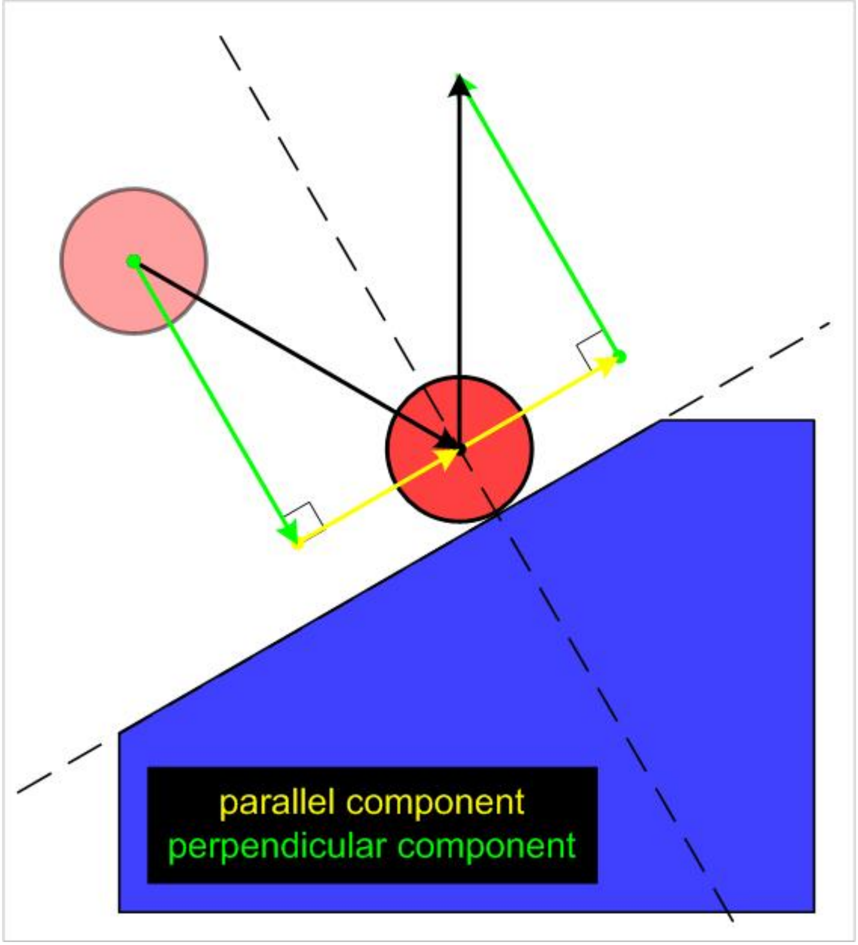
Something In-Between?

- **Partially Elastic**
  - x% energy preserved
  - Different each object
  - Like elastic, but harder
- **Issue**: object "material"
  - What is object made of?
  - **Example**: Rubber? Steel?
- Another parameter!
  - Technical prototype?



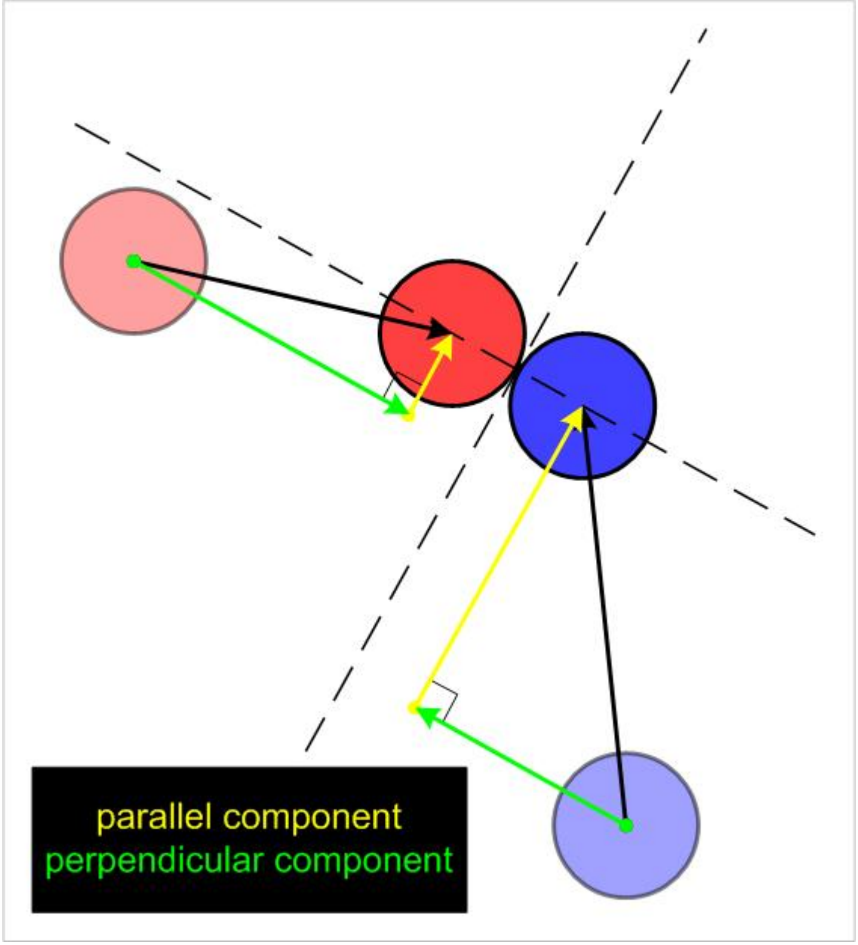
Collision Restitution: Circles

- Single point of contact!
  - Energy transferred at point
  - Not true in complex shapes
- Use **relative coordinates**
  - Point of contact is origin
  - **Perpendicular component**: Line through origin, center
  - **Parallel component**: Axis of collision "surface"
- Reverse object motion on the perpendicular comp



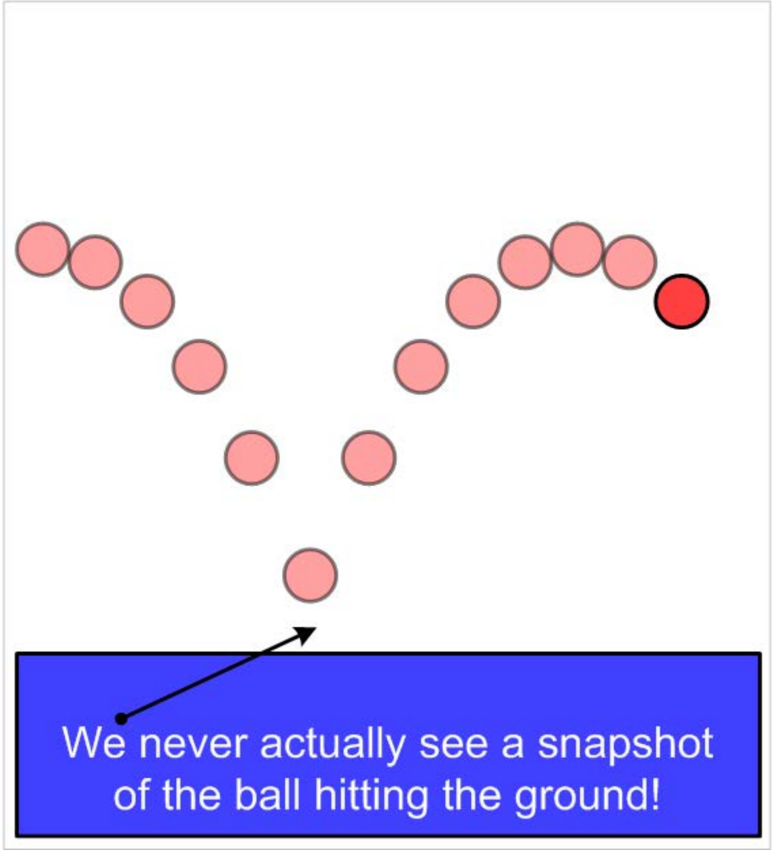
Collision Restitution b: Circles

- Single point of contact!
  - Energy transferred at point
  - Not true in complex shapes
- Use **relative coordinates**
  - Point of contact is origin
  - **Perpendicular component**: Line through origin, center
  - **Parallel component**: Axis of collision “surface”
- **Exchange energy** on the perpendicular comp



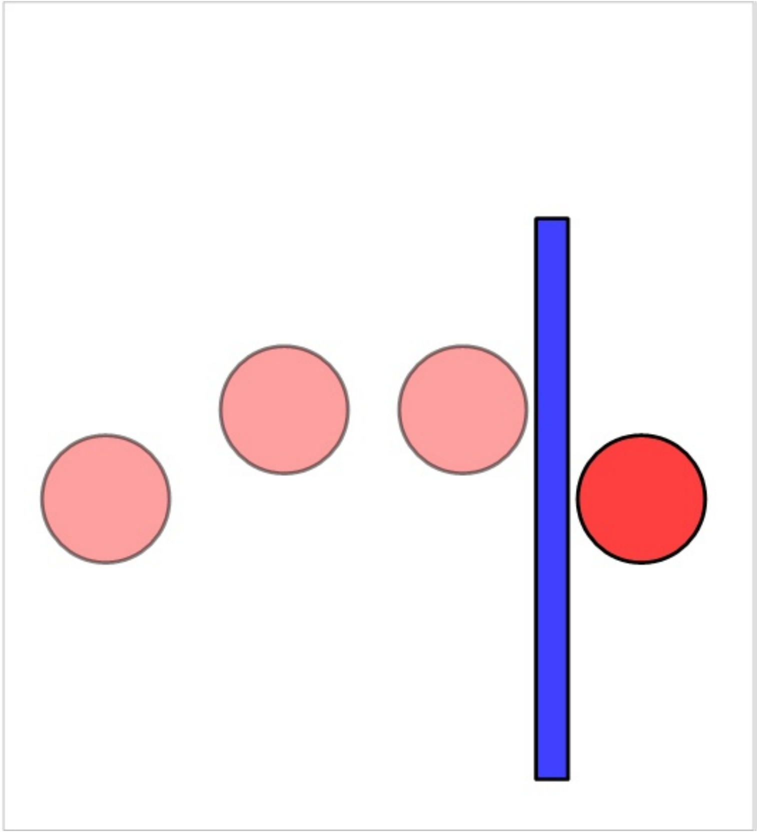
Issues with Collisions: Tunneling

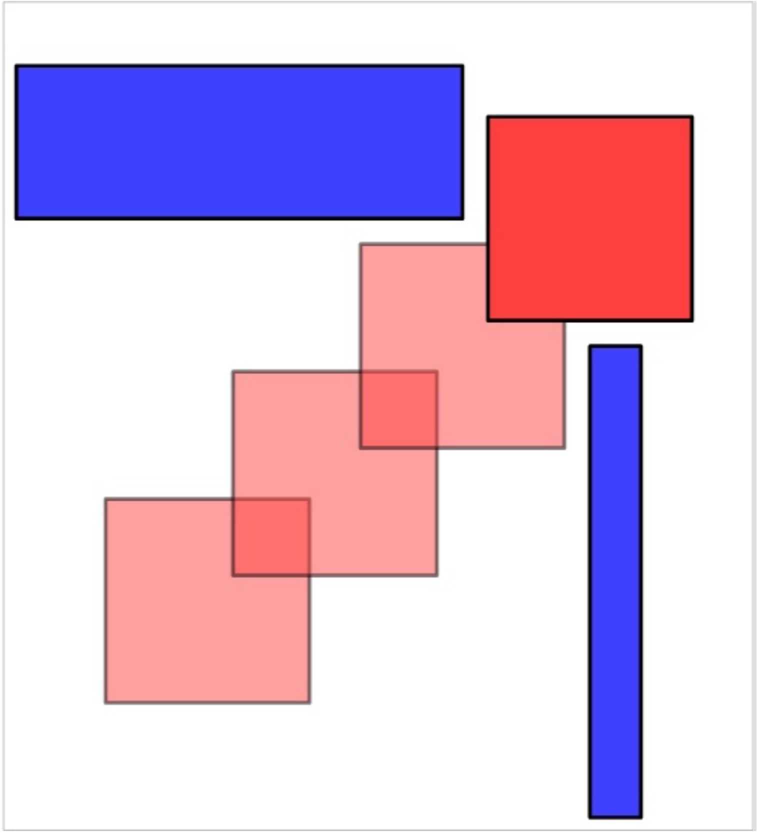
- Games act like **flip-books**
  - Sequence of snapshots
  - Collisions mid-snapshot?
  - Could *miss* the collision
- Example of **false negative**
- This is a **serious** problem
  - Players going where shouldn't
  - Players missing event trigger
  - Cannot ignore tunneling



We never actually see a snapshot of the ball hitting the ground!

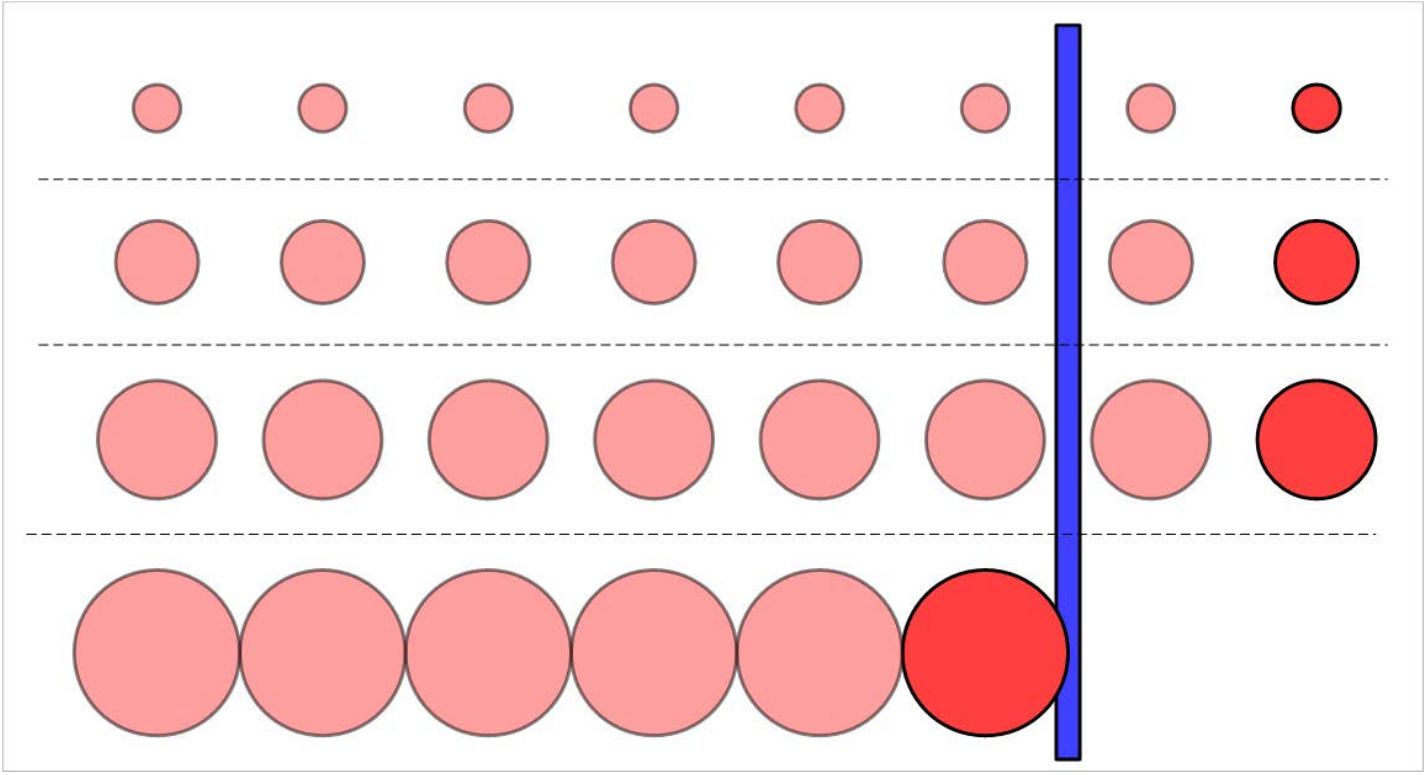
Tunneling





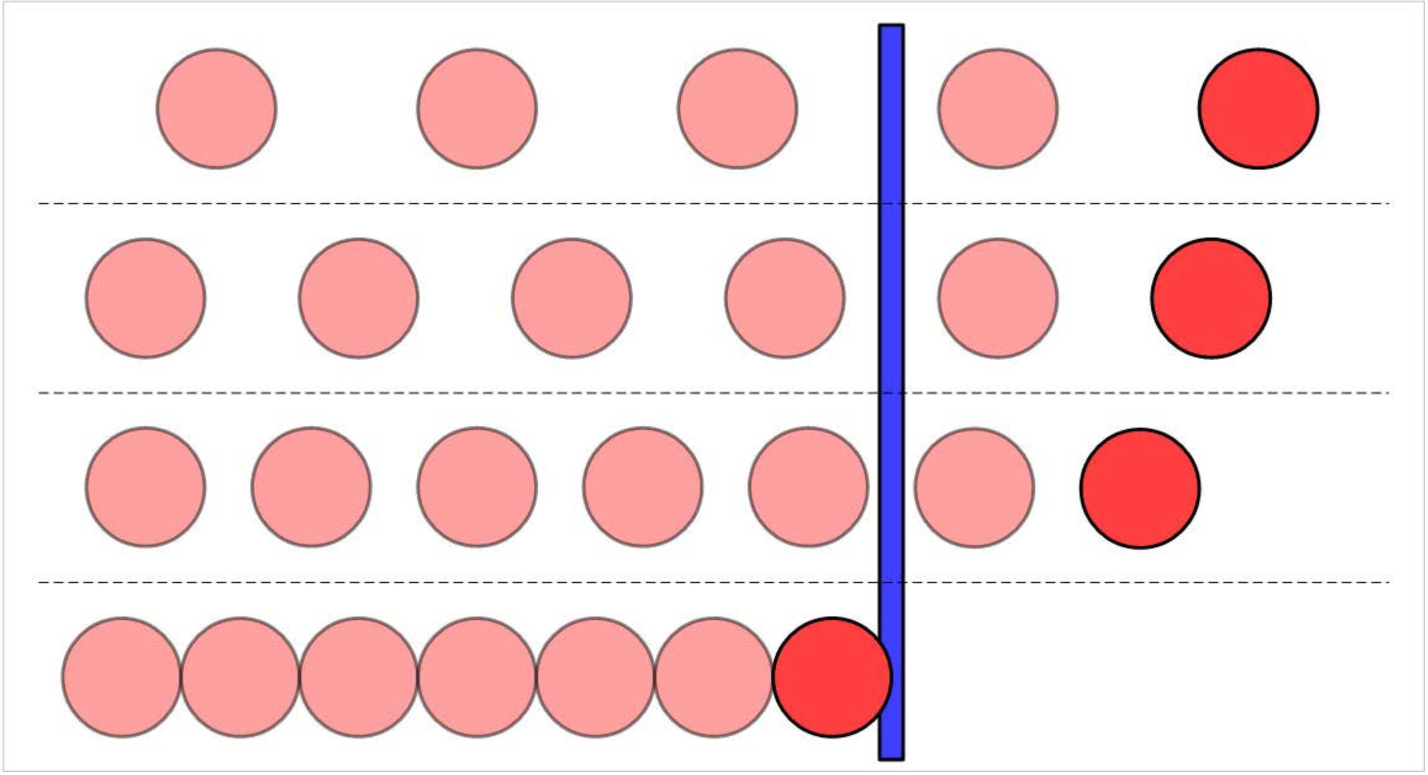
Tunneling: Observations

- Small objects tunnel more easily



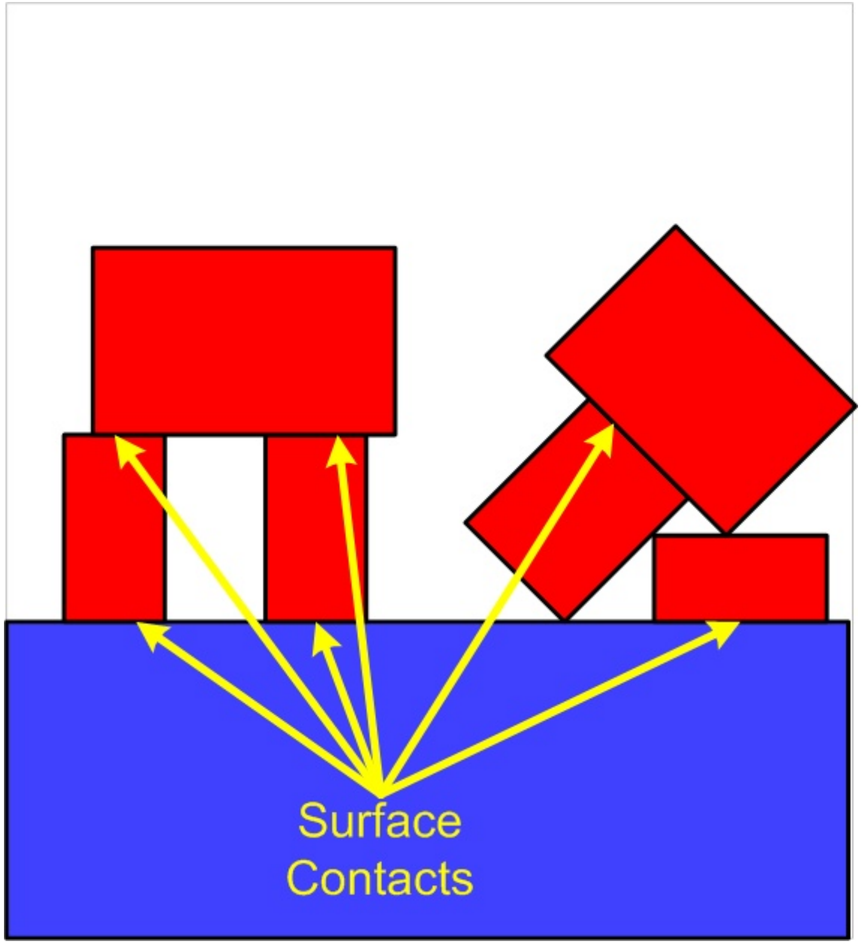
Tunneling: Observations

- Small objects tunnel more easily
- Fast-moving objects tunnel more easily



More Complex Shapes

- Point of contact harder
  - Could just be a point
  - Or it could be an edge
- Model w/ **rigid bodies**
  - Break object into points
  - Connect with constraints
  - Force at point of contact
  - Transfers to other points
- Needs **constraint solver**



Summary

- Object representation depends on goals
  - For **motion**, represent object as a **single point**
  - For **collision**, objects must have **geometry**
- Dynamics is use of forces to move objects
  - Solve **differential equations** for position
  - Need **constraint solvers** to overcome error creep
- Collisions are broken up into two steps
  - **Collision detection** checks for intersections
  - **Collision resolution** is hard if not a circle