Physics

INTRODUCTION, POINT MASSES

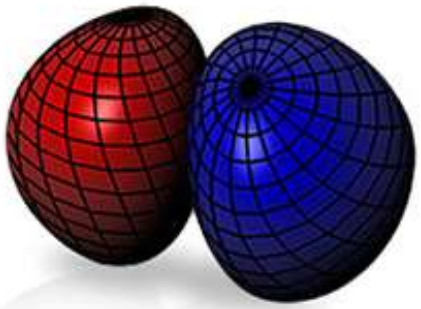
Introduction(1 of 2)

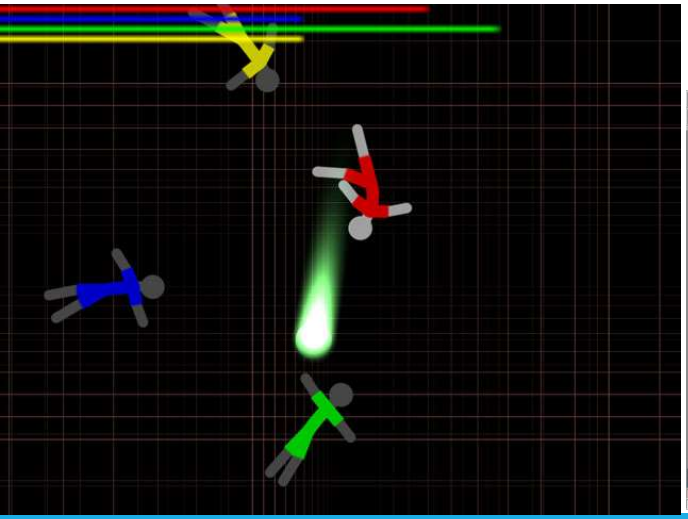
- Physics deals with motions of objects in virtual scene
  - And object interactions during collisions
- Physics increasingly (but only recently, last 3 years?) important for games
  - Similar to advanced AI, advanced graphics
- Enabled by more processing
  - Used to need it all for more core Gameplay (graphics, I/O, AI)
  - Now have additional processing for more
    - Duo-core processors
    - Physics hardware (Ageia's Physx) and general GPU (instead of graphics)
    - Physics libraries (Havok FX) that are optimized

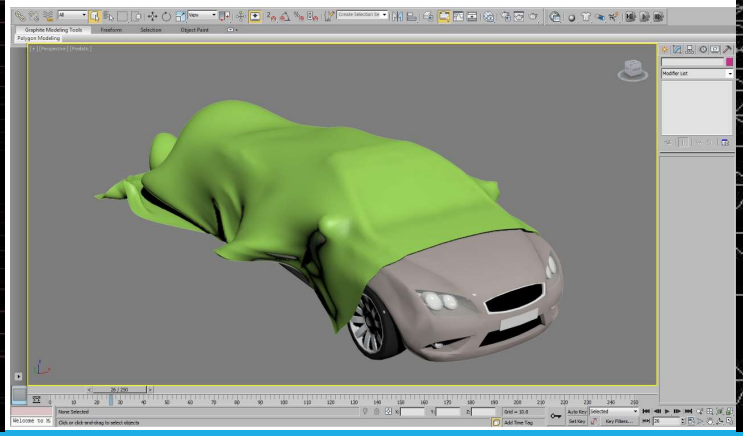
Introduction(2 of 2)

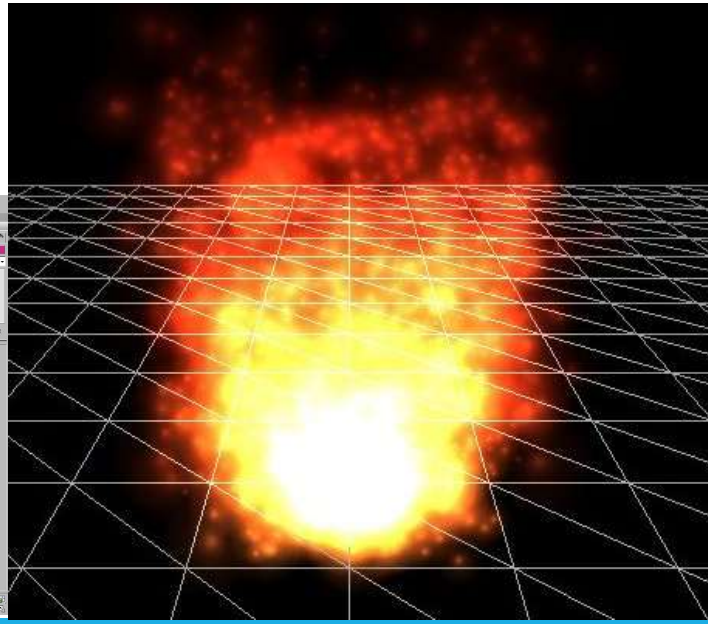
- Potential
  - New gameplay elements
  - Realism (ie- gravity, water resistance, etc.)
  - Particle effects
  - Improved collision detection
  - Rag doll physics
  - Realistic motion


ASSASSINS
CREED.COM





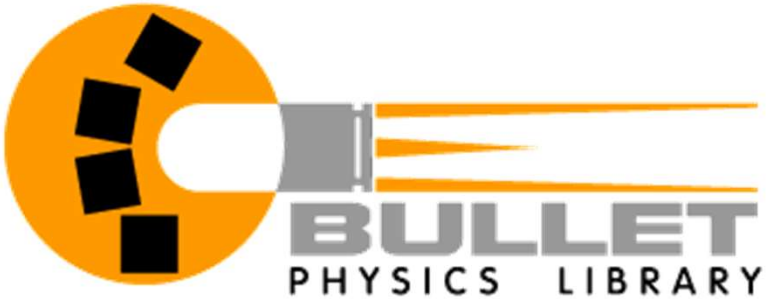




Physics Engine – Build or Buy?

- Physics engine can be part of a game engine
- License middleware physics engine
  - Complete solution from day 1
  - Proven, robust code base (in theory)
  - Features are always a tradeoff
- Build physics engine in-house
  - Choose only the features you need
  - Opportunity for more game-specific optimizations
  - Greater opportunity to innovate
  - Cost can be easily be much greater

Famous Physics Engine



Newtonian Physics (1 of 3)

- Sir Isaac Newton (around 1700) described three laws, as basis for *classical mechanics*:
  1. A body will remain at rest or continue to move in a straight line at a constant velocity unless acted upon by another force
     - (So, Atari *Breakout* had realistic physics! ☺)
  2. The acceleration of a body is proportional to the resultant force acting on the body and is in the same direction as the resultant force.
  3. For every action, there is an equal and opposite reaction
- More recent physics show laws break down when trying to describe universe (Einstein), but good for computer games

Newtonian Physics (2 of 3)

- Generally, object does not come to a stop naturally, but forces must bring it to stop
  - Force can be friction (ie- ground)
  - Force can be drag (ie- air or fluid)
- Forces: gravitational, electromagnetic, weak nuclear, strong nuclear
  - But gravitational most common in games (and most well-known)
- From dynamics:
  - Force = mass x acceleration ($F=ma$)
- In games, forces often known, so need to calculate acceleration

$$a = F/m$$

- Acceleration used to update velocity and velocity used to update objects position:
  - $x = x + (v + a * t) * t$ (t is the delta time)
  - Can do for (x, y, z) positions
  - (speed is just magnitude, or size, of velocity vector)
- So, if add up all forces on object and divide by mass to get acceleration

Newtonian Physics (3 of 3)

- *Kinematics* is study of motion of bodies and forces acting upon bodies
- Three bodies:
  - *Point masses* – no angles, so only linear motion (considered infinitely small)
    - Particle effects
  - *Rigid bodies* – shapes to not change, so deals with angular (orientation) and linear motion
    - Characters and dynamic game objects
  - *Soft bodies* – have position and orientation and can change shape (ie- cloth, liquids)
    - Starting to be possible in real-time

Topics

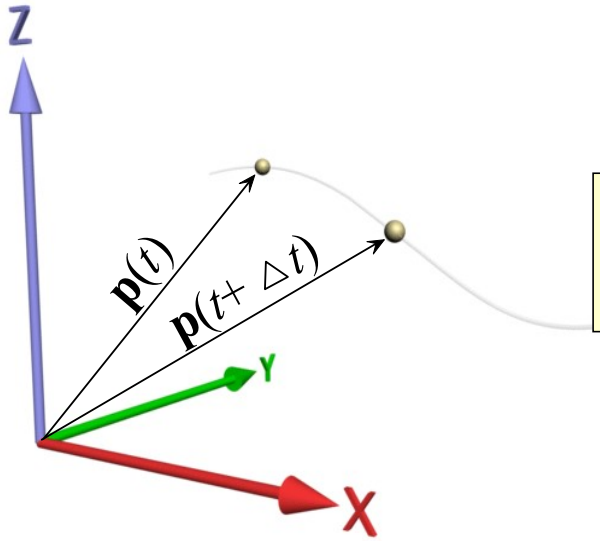
- Introduction
- Point Masses
  - Projectile motion
  - Collision response
- Rigid-Bodies
  - Numerical simulation
  - Controlling truncation error
  - Generalized translation motion
- Soft Body Dynamic System
- Collision Detection

Point-Mass (Particle) Physics

- What is a Particle?
  - A sphere of finite radius with a perfectly smooth, frictionless surface
  - Experiences no rotational motion
- Particle kinematics
  - Defines the basic properties of particle motion
  - Position, Velocity, Acceleration

Particle Kinematics - Position

- Location of Particle in World Space (units are meters (m))
- Changes over time when object moves



$$\mathbf{p} = \langle p_x, p_y, p_z \rangle$$

Particle Kinematics - Velocity and Acceleration

- Average velocity (units: meters/sec):
  - $[p(t+\Delta t) - p(t)] / \Delta t$
  - But velocity may change in time Δt
- Instantaneous velocity is derivative of position:

$$\mathbf{V}(t) = \lim_{\Delta t \to 0} \frac{\mathbf{p}(t + \Delta t) - \mathbf{p}(t)}{\Delta t} = \frac{d}{dt}\mathbf{p}(t)$$

(Position is the integral of velocity over time)

- Acceleration (units: m/s^2)
  - First time derivative of velocity
  - Second time derivative of position

$$\mathbf{a}(t) = \frac{d}{dt}\mathbf{V}(t) = \frac{d^2}{dt^2}\mathbf{p}(t)$$

Newton's 2nd Law of Motion

- Paraphrased – "An object's change in velocity is proportional to an applied force"
- The Classic Equation:

$$\mathbf{F}(t) = m\mathbf{a}(t)$$

  - m = mass (units: kilograms, kg)
  - $\mathbf{F}(t)$ = force (units: Newtons)

What is Physics Simulation?

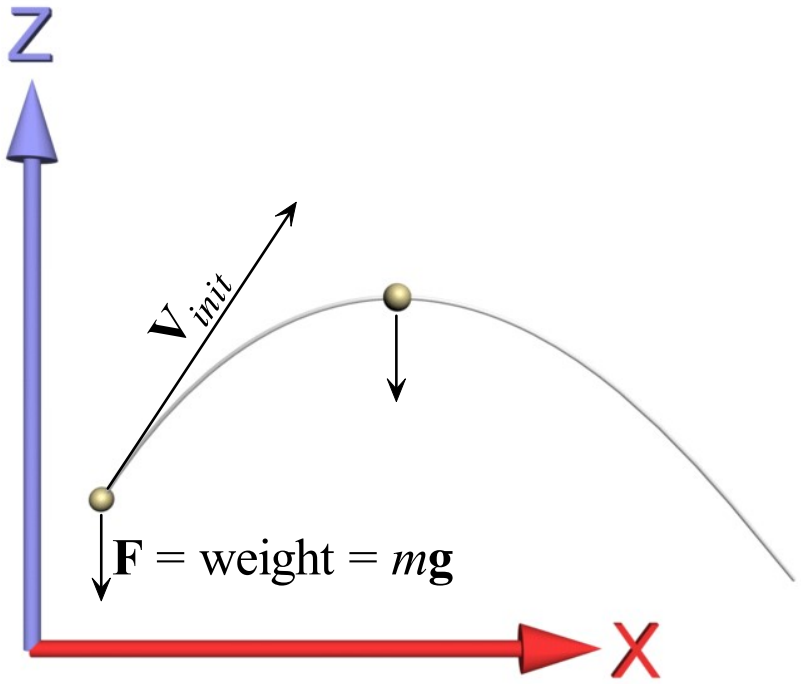
- The Cycle of Motion:
  - Force, $\mathbf{F}(t)$, causes acceleration
  - Acceleration, $\mathbf{a}(t)$, causes a change in velocity
  - Velocity, $\mathbf{V}(t)$ causes a change in position

- Physics Simulation:
  - Solving variations of the above equations over time
  - Use to get positions of objects
  - Render objects on screen
  - Repeat to emulate the cycle of motion

Topics

- Introduction
- Point Masses
  - Projectile motion
  - Collision response
- Rigid-Bodies
  - Numerical simulation
  - Controlling truncation error
  - Generalized translation motion
- Soft Body Dynamic System
- Collision Detection

Example: 3D Projectile Motion (1 of 3)



- Example
- https://phet.colorado.edu/sims/projectile-motion/projectile-motion_en.html

Example: 3D Projectile Motion (1 of 3)

- Constant Force (ie- gravity)
  - Force is *weight* of the projectile, $\mathbf{W} = m\mathbf{g}$
  - $\mathbf{g}$ is constant acceleration due to gravity
    - On earth, gravity (g) is 9.81 m/s^2
- With constant force, acceleration is constant
- Easy to integrate to get closed form
- Closed-form "Projectile Equations of Motion":

$$\mathbf{V}(t) = \mathbf{V}_{init} + \mathbf{g}\left(t - t_{init}\right)$$

$$\mathbf{p}(t) = \mathbf{p}_{init} + \mathbf{V}_{init}\left(t - t_{init}\right) + \frac{1}{2}\mathbf{g}\left(t - t_{init}\right)^2$$

  - These closed-form equations are valid, and *exact**, for any time, t, in seconds, greater than or equal to t_{init} (Note, requires constant force)

Example: 3D Projectile Motion (2 of 3)

- For simulation:
  - Begins at time t_{init}
  - Initial velocity, $\mathbf{V}_{init}$ and position, $\mathbf{p}_{init}$, at time t_{init}, are known
  - Can find later values (at time t) based on initial values
- On Earth:
  - If we choose positive Z to be straight up (away from center of Earth), g_{Earth} = 9.81 m/s^2:

$$\mathbf{g}_{Earth} = -g_{Earth}\hat{k} = \langle 0.0, 0.0, -9.81 \rangle \text{ m/s}^2$$

Note: the Moon's gravity is about 1/6th that of Earth

Pseudo-code for Simulating Projectile Motion

```
void main() {
// Initialize variables
    Vector v_init(10.0, 0.0, 10.0);
    Vector p_init(0.0, 0.0, 100.0), p = p_init;
    Vector g(0.0, 0.0, -9.81); // earth
    float t_init = 10.0; // launch at time 10 seconds

// The game sim/rendering loop
    while (1) {
        float t = getCurrentGameTime(); // could use system clock
        if (t > t_init) {
        float t_delta = t - t_init;
        p = p_init + (v_init * t_delta); // velocity
        p = p + 0.5 * g * (t_delta * t_delta); // acceleration
        }
    renderParticle(p); // render particle at location p

    }
}
```

Topics

- Introduction
- Point Masses
  - Projectile motion
  - Collision response
- Rigid-Bodies
  - Numerical simulation
  - Controlling truncation error
  - Generalized translation motion
- Soft Body Dynamic System
- Collision Detection

Frictionless Collision Response (1 of 4)

- *Linear momentum* – is the mass times the velocity

$$\text{momentum} = mV$$

  - (units are kilogram-meters per second)
- Related to the force being applied
  - 1st time derivative of linear momentum is equal to net force applied to object

$$d/dt\,(mV(t)) = F(t)$$

- Most objects have constant mass, so:

$$d/dt\,(mV(t)) = m\ d/dt\,(V(t))$$

  - Called the *Newtonian Equation of Motion*
    - Since when integrated over time it determines the motion of an object

Frictionless Collision Response (2 of 4)

- Consider two colliding particles
- For the duration of the collision, both particles exert force on each other
  - Normally, collision duration is very short, yet change in velocity is dramatic (ex- pool balls)
- Integrate previous equation over duration of collision

$$m_1 V_1^+ = m_1 V_1^- + \Lambda \qquad \text{(equation 1)}$$

- $m_1 V_1^-$ is linear momentum of particle 1 just before collision
- $m_1 V_1^+$ is the linear momentum just after collision
- Λ is the linear impulse
  - Integral of collision force over duration of collision

Frictionless Collision Response (3 of 4)

- Newton's third law of motion says for every action, there is an equal and opposite reaction
  - So, particle 2 is the same magnitude, but opposite in direction (so, $-1*\Lambda$)
- Can solve these equations if know Λ
- Without friction, impulse force acts completely along unit surface normal vector at point of contact

$$\Lambda = \Lambda_s \mathrm{n} \qquad \text{(equation 2)}$$

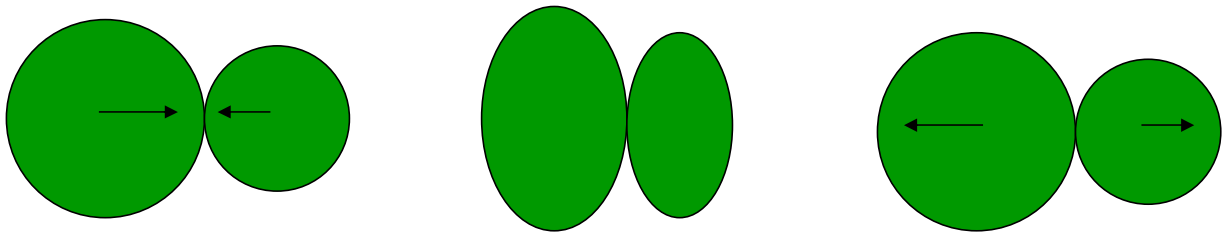
  - n is the unit surface normal vector (see collision detection for point of contact
  - Λ_s is the scalar value of the impulse
    - (In physics, *scalar* is simple physical quantity that does not depend on direction)
- So, have 2 equations with three unknowns (V_1^+, V_2^+, Λ_s).
  - Need third equation to solve for all

Frictionless Collision Response (4 of 4)

- Third equation is approximation of material response to colliding objects

$$(V_1^+ - V_2^+)\, n = -\varepsilon\, (V_1^- - V_2^-)\, n \qquad \text{(equation 3)}$$

  - Note, in general, can collide at angle
  - ε is coefficient of restitution
    - Related to conservation or loss of kinetic energy
    - ε is 1, totally elastic, so objects rebound fully
    - ε is 0, totally plastic, objects no restitution, maximum loss of energy
    - In real life, depends upon materials
      - Ex: tennis ball on raquet, ε is 0.85 and deflated basketball with court ε is 0)
      - (Next slides have details)



Period of deformation Period of restitution

Coefficient of Restitution (1 of 6)

- A measure of the elasticity of the collision
  - How much of the kinetic energy of the colliding objects before collision remains as kinetic energy after collision
- Links:
  - Basic Overview
  - Wiki
  - The Physics Factbook
  - Physics of Baseball and Softball Bats
  - Measurements of Sports Balls

Coefficient of Restitution (2 of 6)

- Defined as the ratio of the differences in velocities before and after collision

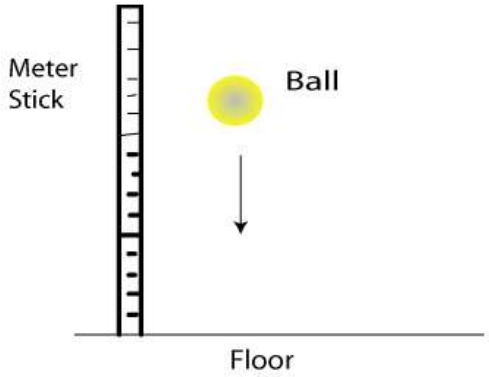
$$\varepsilon = (V_1^+ - V_2^+) \ / (V_1^- - V_2^-)$$

- For an object hitting an immovable object (ie- the floor)

$$\varepsilon = \text{sqrt}(h/H)$$

  - Where h is bounce height, H is drop height

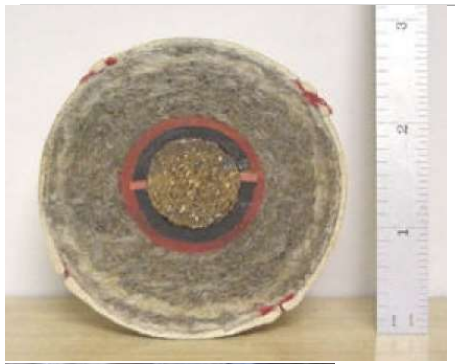
Coefficient of Restitution (3 of 6)



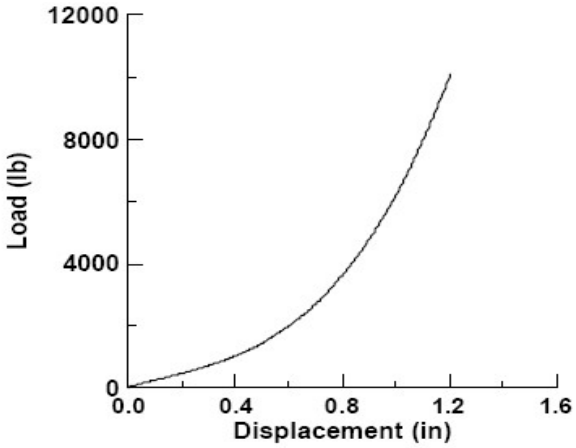
object	H (cm)	h_1 (cm)	h_2 (cm)	h_3 (cm)	h_4 (cm)	h_5 (cm)	h_{ave} (cm)	c.o.r.
range golf ball	92	67	66	68	68	70	67.8	0.858
tennis ball	92	47	46	45	48	47	46.6	0.712
billiard ball	92	60	55	61	59	62	59.4	0.804
hand ball	92	51	51	52	53	53	52.0	0.752
wooden ball	92	31	38	36	32	30	33.4	0.603
steel ball bearing	92	32	33	34	32	33	32.8	0.597
glass marble	92	37	40	43	39	40	39.8	0.658
ball of rubber bands	92	62	63	64	62	64	63.0	0.828
hollow, hard plastic ball	92	47	44	43	42	42	43.6	0.688

- Drop ball from fixed height (92 cm)
- Record bounce
- Repeat 5 times and average)
- Various balls

Coefficient of Restitution (4 of 6)



More force needed to compress, sort of like a spring



- Layers:
  - Cork and rubber (like a superball)
  - Tightly round yarn
  - Thin tweed
  - Leather
- (Softball simpler – just cork and rubber with leather)

Spring would be straight line:

$F = k\,x$

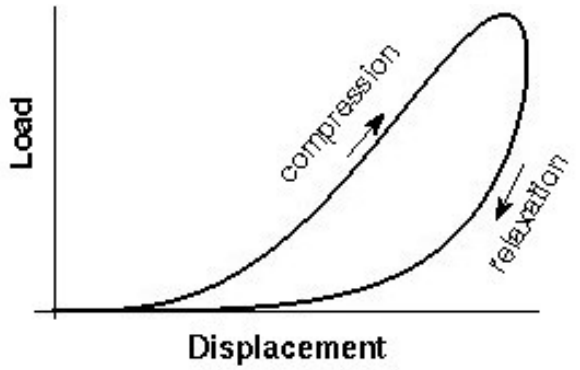
But is:

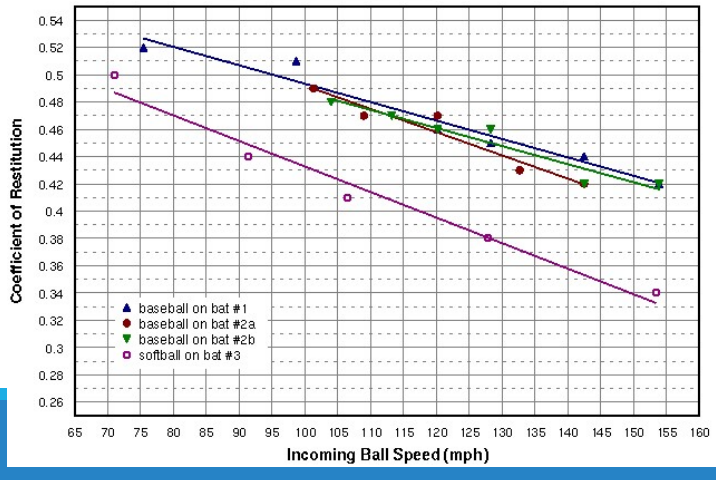
$F = k\,x^{p}$

Coefficient of Restitution (5 of 6)

- Plus, force-compression curve not symmetric
  - Takes more time to expand than compress
  - Meaning, for $F = kx^p$, p different during relaxation
- Area inside curve is energy that is lost to internal friction

- Coefficient of restitution depends upon speed
  - Makes it even more complicated





Coefficient of Restitution (6 of 6)

- Last notes …
- Technically
  - COR a property of a collision, not necessarily an object
    - 5 different types of objects → 10 (5 choose 2 = 10) different CORs
  - May be energy lost to internal friction (baseball)
  - May depend upon speed
  - All that can get complicated!
- But, for properties not available, can estimate
  - (ie- rock off of helmet, dodge ball off wall)
  - Playtest until looks “right”

Putting It All Together

- Have 3 equations (equation 1, 1+ and 4) and 3 unknowns (V_1^+, V_2^+, Λ_s)
- Can then compute the linear impulse

$$\Lambda = -\left(\frac{m_1 m_2 (1+\varepsilon)\,(V_1^- - V_2^-)\,n}{m_1 + m_2} \right) n \quad \text{(equation 4)}$$

- Can then apply Λ to previous equations:
  - Equation 1 to get V_1^+ (and similarly V_2^+)
- … and divide by m_1 (or m_2) to get after-collision velocities

The Story So Far

- Visited basic concepts in kinematics and Newtonian physics
- Generalized for 3 dimensions
- Ready to be used in some games!

- Show Pseudo code next
  - Simulating N Spherical Particles under Gravity with no Friction

Psuedocode (1 of 5)

```
void main() {
// initialize variables
vector v_init[N] = initial velocities;
vector p_init[N] = initial positions;
vector g(0.0, 0.0, -9.81); // earth
float mass[N] = particle masses;
float time_init[N] = start times;
float eps = coefficient of restitution;
```

Psuedocode (2 of 5)

```
// main game simulation loop
while (1) {
    float t = getCurrentGameTime();
    detect collisions(t_collide is time);
    for each colliding pair(i, j) {

    // calc position and velocity of i
    float telapsed = t_collide - time_init[i];
    pi = p_init[i] + (V_init[i] * telapsed); // velocity
    pi = pi + 0.5*g*(telapsed*telapsed);     // accel

    // calc position and velocity of j
    float telapsed = tcollide - time_init[j];
    pj = p_init[j] + (V_init[j] * telapsed); // velocity
    pj = pj + 0.5*g*(telapsed*telapsed);     // accel
```

Psuedocode (3 of 5)

```
// for spherical particles, surface
// normal is just vector joining middle
normal = Normalize(pj – pi);

// compute impulse (equation 4)
impulse = normal;
impulse *= -(1 + eps)*mass[i] * mass[j];
impulse *= normal.DotProduct(vi - vj); //Vi1Vj1+Vi2Vj2+Vi3Vj3
impulse /= (mass[i] + mass[j]);
```

Psuedocode (4 of 5)

```
// Restart particles i and j after collision (eq 1)
// Since collision is instant, after-collisions
// positions are the same as before
V_init[i] += impulse / mass[i];
V_init[j] -= impulse / mass[j];  // equal and opposite
p_init[i] = pi;
p_init[j] = pj;

// reset start times since new init V
time_init[i] = t_collide;
time_init[j] = t_collide;
} // end of for each
```

Psuedocode (5 of 5)

```
// Update and render particles
for k = 0; k<N; k++){
float tm = t – time_init[k];
p = p_init[k] + V_init[k] + tm; //velocity
p = p + 0.5*g*(tm*tm); // acceleration

render particle k at location p;
}
```

Topics

- Introduction
- Point Masses
  - Projectile motion
  - Collision response
- Rigid-Bodies (next session)
  - Numerical simulation
  - Controlling truncation error
  - Generalized translation motion
- Soft Body Dynamic System
- Collision Detection