Lecture 17

Physics and Motion
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: Farseer, Box2D, Bullet
The Compromise

- Best can do is give the **problem description**
  - Squirrel Eiserloh’s *Problem Overview* slides
  - [http://www.essentialmath.com/tutorial.htm](http://www.essentialmath.com/tutorial.htm)

- Hopefully allow you to read engine APIs
  - Understand the limitations of physics engines
  - Learn where to go for other solutions

- Will go more in depth on a few select topics
  - **Example:** Collisions and restitution
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?
Take Away for Today

• Difference between **kinematics** and **dynamics**
  • How do each of them work?
  • When is one more appropriate than another?

• What dynamics do physics engines provide?
  • What is a **particle system**? When is it used?
  • What is a **constraint solver**? When is it used?

• What problems do physics engines have?
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
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 $\Delta v = a\Delta t$)
  - $p(t+\Delta t) = p(t) + v(t)\Delta t + \frac{1}{2}a(\Delta t)^2$
  - Or $\Delta p = v_0\Delta t + \frac{1}{2}a(\Delta t)^2$
**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 \( \Delta v = a\Delta t \))
  - \( p(t+\Delta t) = p(t) + v(t)\Delta t + \frac{1}{2}a(\Delta t)^2 \)
  - Or \( \Delta p = v_0\Delta t + \frac{1}{2}a(\Delta t)^2 \)
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 you calculus
- Smooth = smaller steps
  - More frames a second?
Linear Dynamics

- **Forces** affect movement
  - Springs, joints, connections
  - Gravity, repulsion
- Get velocity from forces
  - Compute current force \( F \)
  - \( F \) constant entire frame
- Formulas:
  \[
  \Delta a = \frac{F}{m}
  \]
  \[
  \Delta v = \frac{F \Delta t}{m}
  \]
  \[
  \Delta p = \frac{F (\Delta t)^2}{m}
  \]
- Again, piecewise **linear**
Linear Dynamics

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

- **Differential Equation**
  - $F(p,t) = m \ a(t)$
  - $F(p,t) = m \ p''(t)$

- **Euler’s method:**
  - $a_i = F(p_i, i\Delta t)/m$
  - $v_{i+1} = v_i + a_i \Delta t$
  - $p_{i+1} = p_i + v_i \Delta t$

- **Other techniques exist**
  - **Example:** Runga-Kutta

Made for accuracy
Not for speed
## Kinematics vs. Dynamics

<table>
<thead>
<tr>
<th>Kinematics</th>
<th>Dynamics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advantages</strong></td>
<td><strong>Advantages</strong></td>
</tr>
<tr>
<td>• Very simple to use</td>
<td>• Complex physics</td>
</tr>
<tr>
<td>• Non-calculus physics</td>
<td>• Non-rigid bodies</td>
</tr>
<tr>
<td><strong>Disadvantages</strong></td>
<td><strong>Disadvantages</strong></td>
</tr>
<tr>
<td>• Only simple physics</td>
<td>• Beyond scope of course</td>
</tr>
<tr>
<td>• All bodies are rigid</td>
<td>• Need a physics engine</td>
</tr>
<tr>
<td>• Old school games</td>
<td>• Neo-retro games</td>
</tr>
</tbody>
</table>

**Advantages**
- Very simple to use
- Non-calculus physics

**Disadvantages**
- Only simple physics
- All bodies are rigid
- Old school games
Issues with Game Physics

Flipbook Syndrome

- Things typically happen in-between snapshots
- Curved trajectories are actually piecewise linear
- Terms assumed constant throughout the frame
- Errors accumulate

We never actually see a snapshot of the ball hitting the ground!
Issues with Game Physics

Flipbook Syndrome

• Things typically happen in-between snapshots

• Curved trajectories are actually piecewise linear

• Terms assumed constant throughout the frame

• Errors accumulate
Issues with Game Physics

Flipbook Syndrome

- Things typically happen in-between snapshots
- Curved trajectories are actually piecewise linear
- **Terms assumed constant** throughout the frame
- Errors accumulate
Issues with Game Physics

Flipbook Syndrome

- Things typically happen in-between snapshots
- Curved trajectories are actually piecewise linear
- Terms assumed constant throughout the frame
- Errors accumulate
Issues with Game Physics

- Want energy conserved
  - Energy loss undesirable
  - Energy gain is evil
  - Simulations explode!
- Not always possible
  - Error accumulation
  - Visible artifact of Euler
- Requires ad hoc solutions
  - Clamping (max values)
  - Manual dampening
Dealing with Error Creep

- Classic solution: reduce the time step $\Delta t$
  - 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
  
  $\Delta t = h/n$

- Typically a parameter in your physics engine
Interactions of Objects

- **Collisions**
  - Typically assume elastic
  - 100% energy conserved
  - Think billiard balls

- **Springs**
  - Exerts a force on object
  - If too stretched, pulls back
  - If compressed, pushes out
  - Complex if ends not fixed
  - Repulsive, **attractive** forces

Collisions

Springs

\[ \Delta f = k \Delta x \]
Interactions of Objects

- **Collisions**
  - Typically assume elastic
  - 100% energy conserved
  - Think billiard balls

- **Springs**
  - Exerts a force on object
  - If too stretched, pulls back
  - If compressed, pushes out
  - Complex if ends not fixed
  - Repulsive, **attractive** forces

Collisions

Springs

\[ \Delta f = k \Delta x \]

Next Time
Particle Systems

- World is a bunch of **particles**
  - Particles interact via forces

- Examples of relevant forces
  - **Constant**: gravity
  - **Position/time dependent**: force fields
  - **Velocity dependent**: drag
  - **N-ary dependent**: springs, collisions

- Force is function \( F(p_1, \ldots, p_n, v_1, \ldots, v_n, t) \)
  - Compute from interaction with other particles
Particle Systems

- At the start of each iteration
  - Compute forces between particle pairs
  - Sum up all of the forces acting on a particle
  - From these forces compute the acceleration
  - Compute velocity and position as before

- **Important detail**: delay particle updates
  - Do not move a particle before it is summed
  - This can cause very subtle errors (why?)
Recall: Processing NPCs

- **Naïve solution:** sequentially

- **Problem:** NPCs react too fast!
  - Each reads the actions of previous NPCs
  - Even before drawn on screen!

- **Idea:** only react to what can see
  - Choose actions, but don’t perform
  - Once all chosen, then perform

Another reason to abstract actions
Physics Engine Support

• Most engines support particle systems
  • Specify the collection of bodies
  • Specify the forces that can act on the body
  • Specify the time step and number of iterations
  • Positions are updated automatically for you

• This is a black box implementation
  • Little control over movement
  • Custom methods beyond scope of course
Constrained Behavior

• Suppose we have a bead on a wire
  • The bead can slide freely along wire
  • It can never come off, however hard we pull.
  • How does the bead move under applied forces?

• Usually a curve given by function $C(x,y) = 0$

![Diagram showing a bead on a wire with vectors n and v]
Particle Systems?

- Why don’t we attach to wire with spring?
  - Move the bead normally (maybe off wire)
  - Apply spring force as if fixed to curve

- This is not going to work
  - **Weak springs**: Wobbles like goopy jello
  - **Strong springs**: To Infinity and Beyond!
Constraint Solvers

- **Limit** object movement
  - **Joints**: distance constraint
  - **Contact**: non-penetration
  - **Restitution**: bouncing
  - **Friction**: sliding, sticking

- Many applications
  - Ropes, chains
  - Box stacking

- Focus of XNA Lab 3
Implementing Constraints

- Very difficult to implement
  - **Errors**: joints to fall apart
  - Called *position drift*
  - Too hard for this course

- Use a physics engine!
  - Box2D supports constraints
  - But I would limit to joints
  - **Example**: ropes, rag dolls

- Want more? CS 5643
  - Or read about it online
Final Word on Motion

• Constraints/particle systems are finicky
  • Performance/appearance depends on parameters
  • Expect to do a lot of tuning to get it just right
  • Some of you have already seen in simple systems

• **Solution**: prototype, prototype, prototype
  • Make a knob to change everything
  • Should be able to change knobs real-time
  • Recompiling or restarting not acceptable
Summary

- For motion, model game objects as points
  - Often a single point to represent the object
  - A rigid body is made of several points

- Dynamics is the use of forces to move objects
  - Compute acceleration from the sum of forces
  - Use Euler’s method to compute velocity, position

- Objects can interact in complex ways
  - **Particle systems**: objects exert a force on one another
  - **Constraint solvers**: restrictions for more rigid behavior