Lecture 18

Physics: Overview
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 (Farseer), Chipmunk, Bullet
Approaching the Problem

- Want to start with the **problem description**
  - Squirrel Eiserloh’s *Problem Overview* slides
  - [http://www.essentialmath.com/tutorial.htm](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
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
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?
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?
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?
Physics Overview

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?
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?
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?
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 $\Delta p = v_0\Delta t + \frac{1}{2}a(\Delta t)^2$
**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 $\Delta p = p(t+\Delta t) - p(t)$

**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 $\Delta p = v_0\Delta t + \frac{1}{2}a(\Delta t)^2$
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**

Physics Overview
Linear Dynamics

- **Force**: $F(p,t)$
  - $p$: current position
  - $t$: current time

- Creates a **vector field**
  - Movement should follow field direction

- **Update formulas**
  - $a_i = F(p_i,i\Delta t)/m$
  - $v_{i+1} = v_i + a_i\Delta t$
  - $p_{i+1} = p_i + v_i\Delta t$
• **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 as DE Solvers

- Differential Equation
  - $F(p,t) = ma(t)$
  - $F(p,t) = mp''(t)$

- 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
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
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
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
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
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
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
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
Physics as DE Solvers

• Differential Equation
  • \( F(p,t) = m \ a(t) \)
  • \( F(p,t) = m \ \dot{p}''(t) \)

• 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>

**Physics Overview**
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 \textit{iteration}

- Multiple iterations per frame
  - Let $h$ be the length of the frame
  - Let $n$ be the number of iterations

$\Delta t = \frac{h}{n}$

- 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
Interactions of Objects

- **Particle Systems**
  - Elastic, collisional 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

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

### Springs

- \[ \Delta f = k \Delta x \]
  - Changes in force dependent on changes in compression
  - \( f + \Delta f \)
Particle Systems

- World is a bunch of **particles**
- Particles interact via forces
  - *Constant*: gravity
  - *Position/time dependent*: force fields
  - *Velocity dependent*: drag
  - *N-ary dependent*: springs, collisions
- Force is function $F(p_1, \ldots, p_n, v_1, \ldots, v_n, t)$
  - Handle with this in a physics engine
  - Insert particles & forces and “turn the crank”
Constrained Particle 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 $C(x,y) = 0$
Particle Systems?

- **Idea**: Attach bead to wire with a spring
  - Move the bead normally (maybe off wire)
  - Apply spring force to pull it toward curve

- **Problem**:
  - Weak springs have laggy movement
  - Strong springs have too much energy
Constraint Solvers

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

- Many applications
  - Ropes, chains
  - Box stacking

- Focus of Lab 4 (Box2D)
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
  - Limit applications to joints
  - **Example**: ropes, rag dolls
- Want more? CS 5643
  - Or read about it online
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 require **geometry**
  - Points are no longer enough
  - Must know *where* objects meet

- Often use convex shapes
  - Lines always remain inside
  - If not convex, call it concave
  - Easiest shapes to compute with

- What to do if is not convex?
  - Break into convex components
  - Triangles are always convex!
Collisions and Geometry

- Collisions require geometry
  - Points are no longer enough
  - Must know *where* objects meet

- Often use **convex shapes**
  - Lines always remain inside
  - If not convex, call it concave
  - Easiest shapes to compute with

- What to do if is not convex?
  - Break into convex components
  - Triangles are 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 require geometry
  - Points are no longer enough
  - Must know where objects meet
- Often use convex shapes
  - Lines always remain inside
  - If not convex, call it concave
  - Easiest shapes to compute with
- What to do if is not convex?
  - Break into convex components
  - Triangles are *always convex!*

Physics Overview
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 Resolution: 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 Resolution: 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
More Complex Shapes

- Point of contact harder
  - Could just be a point
  - Or it could be an edge

- Model with **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 the use of forces to move objects
  • **Particle systems**: objects exert a force on one another
  • **Constraint solvers**: restrictions for more rigid behavior

• Collisions are broken up into two steps
  • **Collision detection** checks for intersections
  • **Collision resolution** depends on energy transfer