Lecture 19

Physics Engines
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?
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?

Class **Body**

Class **Fixture**
Body in Box2D

- Represents a single point
  - Center of the object’s mass
  - Object must move as unit

- Properties in class Body
  - Position
  - Linear Velocity
  - Angular Velocity
  - Body Type

- There are 3 body types
  - **Static**: Does not move
  - **Kinematic**: Moves w/o force
  - **Dynamic**: Obeys forces
Body in Box2D

- Represents a single point
  - Center of the object’s mass
  - Object must move as unit

- Properties in class Body
  - **Position**
  - **Linear Velocity**
  - **Angular Velocity**
  - **Body Type**

- There are 3 body types
  - **Static**: Does not move
  - **Kinematic**: Moves w/o force
  - **Dynamic**: Obeys forces

Collisions
Body in Box2D

- Represents a single point
  - Center of the object’s mass
  - Object must move as unit

- Properties in class Body
  - Position
  - Linear Velocity
  - Angular Velocity
  - Body Type

- There are 3 body types
  - **Static**: Does not move
  - **Kinematic**: Moves w/o force
  - **Dynamic**: Obeys forces

- Kinematic is rarely useful
  - Limited collision detection
  - Only collides w/ dynamics
  - Does not bounce or react

**Application**: Bullets
- Light, fast-moving objects
- Should not bounce

Looks like last lecture
Forces vs. Impulses

**Forces**
- Instantaneous push
  - Like a slow push
- Gradually accelerates
- Momentum if sustained

**Impulses**
- Push with duration
  - Like a car crash
- Quickly accelerates
- Immediate momentum

**Impulse = Force x Time**
Forces vs. Impulses

**Forces**
- Instantaneous push
  - Like a slow push
  - Gradually accelerates
  - Momentum if sustained

**Impulses**
- Push with duration
  - Like a car crash
  - Quickly accelerates
  - Immediate momentum

Impulse = Force x 1 Sec

in Box2D/Farseer

Collisions
Four Ways to Move a Dynamic Body

- **Forces**
  - ApplyForce (linear)
  - ApplyTorque (angular)

- **Impulses**
  - ApplyLinearImpulse
  - ApplyAngularImpulse

- **Velocity**
  - LinearVelocity
  - AngularVelocity

- **Translation**
  - SetTransform
Four Ways to Move a Dynamic Body

- **Forces**
  - ApplyForce (linear)
  - ApplyTorque (angular)

- **Impulses**
  - ApplyLinearImpulse
  - ApplyAngularImpulse

- **Velocity**
  - LinearVelocity
  - AngularVelocity

- **Translation**
  - SetTransform

- **Collisions**
  - Great for joints, complex shapes
  - Laggy response to user input
  - A bit hard to control

  - Great for joints, complex shapes
  - Good response to user input
  - Extremely hard to control

  - Bad for joints, complex shapes
  - Excellent response to user input
  - Very easy to control

  - Completely ignores physics!
  - Very easy to control
Example: **Box2D Demo**

Shape: Box  Controls: Force

Density: 1  Friction: 0.1  Restitution: 0

Collisions
Example: **Box2D Demo**

**Controls:**
- WASD for linear force
- Left-right arrows to rotate
- 9 or 0 to change controls

**Shape:** Box
**Controls:** Force

**Density:** 1
**Friction:** 0.1
**Restitution:** 0
Four Ways to Move a Dynamic Body

- **Forces**
  - ApplyForce (linear)
  - ApplyTorque (angular)

- **Impulses**
  - ApplyLinearImpulse
  - ApplyAngularImpulse

- **Velocity**
  - LinearVelocity
  - AngularVelocity

- **Translation**
  - SetTransform

---

Must Cap Velocity
public void Update(float dt) {
    // Apply movement to relevant bodies
    if (body above or equal to max velocity) {
        body.LinearVelocity = maximum velocity
    } else {
        body.ApplyForce(force)
        body.ApplyTorque(torque)
    }
    // Use physics engine to update positions
    world.step(dt);
}
Basic Structure of a Update Loop

```java
public void Update(float dt) {
    // Apply movement to relevant bodies
    if (body above or equal to max velocity) {
        body.LinearVelocity = maximum velocity
    } else {
        body.ApplyForce(force)
        body.ApplyTorque(torque)
    }

    // Use physics engine to update positions
    world.step(dt, num_iterations);
}
```

Multiple times to improve accuracy
public void Update(float dt) {

    // Apply movement to relevant bodies
    if (body above or equal to max velocity) {
        body.LinearVelocity = maximum velocity
    } else {
        body.ApplyForce(force)
        body.ApplyTorque(torque)
    }

    // Use physics engine to update positions
    world.step(dt, num_iterations);
}

Only before first iteration!

Multiple times to improve accuracy
ForceControllers

- Special object in Lab 4
  - Added to World object
  - Implements Update()
  - Applies forces/velocity
- Must for multiple iterations
  - World uses it as a callback
  - Executed every iteration
- Good idea in general
  - Cleanly separates controls
  - Might need iterations later
# Collision Objects in Box 2D

## Shape
- Stores the object geometry
  - Boxes, ellipses or polygons
  - **Must be convex!**
- Has own coordinate space
  - Associated body is origin
  - Unaffected if body moved
  - Cannot be resized later
- Also stores object **density**
  - Mass is area x density

## Fixture
- Attaches a shape to a body
- Fixture has only one body
- Bodies have many fixtures
- Cannot change the shape
  - Must destroy old fixture
  - Must make a new fixture
- Has other properties
  - **Friction**: stickiness
  - **Restitution**: bounciness
Making a Box2D Physics Object

// Make body and place it
body = BodyFactory.CreateBody (world);
body.BodyType = type;
body.Position = position;

// Create a shape
PolygonShape shape = MyFunction(dimensions, density)

// Attach with fixture and set properties
fixture = body.CreateFixture(shape);
fixture.Friction = friction;
fixture.Restitution = restitution;
Making a Box2D Physics Object

// Make body and place it
body = BodyFactory.CreateBody (world);
body.BodyType = type;
body.Position = position;

// Create a shape
PolygonShape shape = MyFunction(dimensions, density);

// Attach with fixture and set properties
fixture = body.CreateFixture(shape);
fixture.Friction = friction;
fixture.Restitution = restitution;
Observations on the Parameters

- **Density** can be anything **non-zero**
  - The higher the density the higher the mass
  - Heavier objects are harder to move

- **Friction** should be within **0 to 1**
  - Can be larger, but effects are unpredictable
  - Affects everything, even manual velocity control

- **Restitution** should be within **0 to 1**
  - A value of 0 means no bounciness at all
  - Unpredictable with manual velocity control
Example: **Box2D Demo**

**Shape:** Box  
**Controls:** Force

**Density:** 1  
**Friction:** 0.1  
**Restitution:** 0

Collisions
Example: Box2D Demo

Controls:

- 1 or 2 to change density
- 3 or 4 to change friction
- 5 or 6 to change restitution
- 7 or 8 to change shape

Density: 1  Friction: 0.1  Restitution: 0
How Do We Find the Shape?

- Do not try to learn boundary
  - Image recognition is hard
  - Hull will have **many** sides

- Have **artists** draw the shape
  - Cover shape with triangles
  - But can ignore interiors
  - Keep # sides small!

- Store shape in another file
  - Do not ruin the art!
  - Need coordinates as data
Data-Driven Design

character.jpg
collisions

character.shape

120,2
130,4
125,50
150,65
160,100
150,110
125,80
140,200
130,200
120,110
...

Collisions
Customized Collisions: ContactHandlers

- Special methods attached to world object
  - Return any two **fixtures** that collide
  - Allow you to **override** collision behavior
  - Or you can **augment** collision behavior

- World manages two lists of handlers
  - **BeginContact List:** When objects first collide
  - **EndContact List:** When objects no longer collide

- **Example:** Color changing in Box2D demo
Issues with Collisions: Tunneling

- Collisions in midstep can lead to tunneling
  - Objects that “pass through” each other
    - Not colliding at start or end of simulation
    - But they collided somewhere in between
  - This is an example of a false negative

- This is a serious problem; cannot ignore
  - Players getting places they shouldn’t
  - Players missing an event trigger boundary
Tunneling
Tunneling: Observations

- Small objects tunnel more easily
Tunneling: Observations

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

Collisions
Possible Solutions to Tunnelling

- Minimum size requirement?
  - Fast objects still tunnel

- Maximum speed limit?
  - Speed limit is a function of object size
  - So small & fast objects (bullets) not allowed

- Smaller time step?
  - Essentially the same as a speed limit

- All of these solutions are inadequate
Swept Shapes

- Bounds contain motion
  - “Cylinder” w/ shape at ends
  - Object always in bounds
  - Convex if shape is convex
- New collision checking
  - Put shapes at start and end
  - Create swept shape for pair
  - Check for collisions
- Can have **false positives**
  - Swept shape ignores time
Swept Shapes

• Bounds contain motion
  • “Cylinder” w/ shape at ends
  • Object always in bounds
  • Convex if shape is convex

• New collision checking
  • Put shapes at start and end
  • Create swept shape for pair
  • Check for collisions

• Can have false positives
  • Swept shape ignores time
Swept Shapes & Relative Coordinates

- **False positives** happen if:
  - Two objects are moving
  - Swept shapes intersect at different intersection times

- What if only one moving?
  - Swept intersects stationary
  - So no false positives

- Change **reference frames**
  - Keep one shape still
  - Move other in new coords
Swept Shapes & Relative Coordinates

- **False positives** happen if:
  - Two objects are moving
  - Swept shapes intersect at different intersection times

- What if only one moving?
  - Swept intersects stationary
  - So no false positives

- Change **reference frames**
  - Keep one shape still
  - Move other in new coords
Swept Shapes & Relative Coordinates

- **False positives** happen if:
  - Two objects are moving
  - Swept shapes intersect at different intersection times

- What if only one moving?
  - Swept intersects stationary
  - So no false positives

- Change **reference frames**
  - Keep one shape still
  - Move other in new coords

What Physics Engines Do

Blue Frame
Rotations Suck

- Relative coordinates no help
  - Cannot use swept shapes
  - Actual solution is hard!
- But not so bad…
  - Angular tunneling looks ok
  - Speed limits are feasible
  - Do linear approximations
- Many physics systems **never** handle this well
Some Words on Joints

- Joints connect **bodies**
  - Anchors can be offset body
  - Coordinates relative to body
- Are affected by **fixtures**
  - Fixtures prevent collisions
  - Limit relative movement
- Must control with forces
  - Manual velocity might violate constraints
  - Use force or impulse
Sample Joint Types

Distance (soft)
Rope (hard)

Revolute

Weld (rigid)

Prismatic

Pulley
Summary

- Physics engines support motion and collisions
  - **Body** class provides the motion
  - **Fixture, Shape** classes are for collisions

- Multiple ways to control a physics object
  - Can **apply forces** or manually **control velocity**
  - Joint constraints work best with forces

- Physics engines do not solve all your problems
  - You have manually compute your shapes
  - May need to tune parameters to prevent tunneling