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?
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
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

Collisions
Four Ways to Move a Dynamic Body

- **Forces**
  - applyForce (linear)
  - applyTorque (angular)

- **Impulses**
  - applyLinearImpulse
  - applyAngularImpulse

- **Velocity**
  - getLinearVelocity
  - getAngularVelocity

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

- **Forces**
  - applyForce (linear)
  - applyTorque (angular)

- **Impulses**
  - applyLinearImpulse
  - applyAngularImpulse

- **Velocity**
  - getLinearVelocity
  - getAngularVelocity

- **Translation**
  - setTransform

---

- 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
Four Ways to Move a Dynamic Body

- **Forces**
  - `applyForce` (linear)
  - `applyTorque` (angular)

- **Impulses**
  - `applyLinearImpulse`
  - `applyAngularImpulse`

- **Velocity**
  - `getLinearVelocity`
  - `getAngularVelocity`

- **Translation**
  - `setTransform`

---

*Must Cap Velocity*
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.setLinearVelocity(maximum velocity);
    } else {
        body.applyForce(force)
        body.applyTorque(torque)
    }

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

Collisions
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.setLinearVelocity(maximum velocity);
    } else {
        body.applyForce(force)
        body.applyTorque(torque)
    }
    // Use physics engine to update positions
    world.step(dt, vel_iterations, pos_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.setLinearVelocity(maximum velocity);
    } else {
        body.applyForce(force)
        body.applyTorque(torque)
    }
    // Use physics engine to update positions
    world.step(dt, vel_iterations, pos_iterations);
}
## Collision Objects in Box 2D

<table>
<thead>
<tr>
<th>Shape</th>
<th>Fixture</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Stores the object geometry</td>
<td>• Attaches a shape to a body</td>
</tr>
<tr>
<td>• Boxes, circles or polygons</td>
<td>• Fixture has only one body</td>
</tr>
<tr>
<td>• <strong>Must be convex!</strong></td>
<td>• Bodies have many fixtures</td>
</tr>
<tr>
<td>• Has own coordinate space</td>
<td>• Cannot change the shape</td>
</tr>
<tr>
<td>• Associated body is origin</td>
<td>• Must destroy old fixture</td>
</tr>
<tr>
<td>• Unaffected if body moved</td>
<td>• Must make a new fixture</td>
</tr>
<tr>
<td>• Cannot be resized later</td>
<td>• Has other properties</td>
</tr>
<tr>
<td>• Also stores object <strong>density</strong></td>
<td>• <strong>Friction</strong>: stickiness</td>
</tr>
<tr>
<td>• Mass is <strong>area x density</strong></td>
<td>• <strong>Restitution</strong>: bounciness</td>
</tr>
</tbody>
</table>
Making a Box2D Physics Object

// Create a body definition
// (this can be reused)
bodydef = new BodyDef();
bodydef.type = type;
bodydef.position.set(position);
bodydef.angle = angle;

// Allocate the body
body1 = world.createBody(bodydef);

// Another?
bodydef.position.set(position2);
body2 = world.createBody(bodydef);
Making a Box2D Physics Object

// Create a body definition
// (this can be reused)
bodydef = new BodyDef();
bodydef.type = type;
bodydef.position.set(position);
bodydef.angle = angle;

// Allocate the body
body1 = world.createBody(bodydef);

// Another?
bodydef.position.set(position2);
body2 = world.createBody(bodydef);
Making a Box2D Physics Object

// Create two triangles as shapes
shape1 = new PolygonShape();
shape2 = new PolygonShape();
shape1.set(verts1); shape2.set(verts2);

// Create a fixture definition
fixdef = new FixtureDef();
fixdef.density = density;

// Attach the two shapes to body
fixdef.shape = shape1;
fixture1 = body1.createFixture(fixdef);
fixdef.shape = shape2;
fixture2 = body1.createFixture(fixdef);

Collisions
Making a Box2D Physics Object

Other shapes possible

- Create two triangles as shapes
  ```java
  shape1 = new PolygonShape();
  shape2 = new PolygonShape();
  shape1.set(verts1); shape2.set(verts2);
  ```

- Create a fixture definition
  ```java
  fixdef = new FixtureDef();
  fixdef.density = density;
  ```

- Attach the two shapes to body
  ```java
  fixture1 = body1.createFixture(fixdef);
  fixture2 = body1.createFixture(fixdef);
  ```

Also set friction and restitution parameters

Reason for separating Fixture & Body classes

21 Collisions
Making a Box2D Physics Object

// Create a body definition
// (this can be reused)
bodydef = new BodyDef();
bodydef.type = type;
bodydef.position.set(position);
bodydef.angle = angle;

// Allocate the body
body1 = world.createBody(bodydef);

// Another?
bodydef.position.set(position2);
body2 = world.createBody(bodydef);

// Create two triangles as shapes
shape1 = new PolygonShape();
shape2 = new PolygonShape();
shape1.set(verts1); shape2.set(verts2);

// Create a fixture definition
fixdef = new FixtureDef();
fixdef.density = density;

// Attach the two shapes to body
fixdef.shape = shape1;
fixture1 = body1.createFixture(fixdef);
fixdef.shape = shape2;
fixture2 = body1.createFixture(fixdef);
Observations on Fixture 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**

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

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

**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
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

character.shape

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

Collisions
Custom Collisions: ContactListeners

- Special listener attached to world object
  - Reacts to any two fixtures that collide
  - Allow you to override collision behavior
  - Or you can augment collision behavior

- Two primary methods in interface
  - `beginContact`: When objects first collide
  - `endContact`: 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
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

Collisions
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
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

How “Bullets” are handled
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