Lecture 18

Box2D Physics
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
  - To be applied over time
  - Gradually accelerates
  - Momentum if sustained

Impulses
- Push with duration
  - To be applied in one frame
  - Quickly accelerates
  - Immediate momentum

Impulse = Force x Time

Diagram showing impulse over time with contact beginning and ending.
Forces vs. Impulses

Forces
- Instantaneous push
  - To be applied over time
  - Gradually accelerates
  - Momentum if sustained

Impulses
- Push with duration
  - To be applied in one frame
  - Quickly accelerates
  - Immediate momentum

Impulse = Force $\times$ 1 Sec

in Box2D
Force and Acceleration

• What do we need to compute motion?
  • \( \Delta p = v\Delta t = v_0\Delta t + \frac{1}{2}a(\Delta t)^2 = v_0\Delta t + \frac{1}{2}(F/m)(\Delta t)^2 \)
  • So depends on Force, current velocity and mass

• Where does that mass come from?
  • Class Body has a getter, but no setter!
  • It comes from the Fixture class
  • Fixture gives volume to body

• Will revisit this later with collisions
Force and Acceleration

- What do we need to compute motion?
  - \( \Delta p = v \Delta t = v_0 \Delta t + \frac{1}{2}a(\Delta t)^2 = v_0 \Delta t + \frac{1}{2}(F/m)(\Delta t)^2 \)
  - So depends on Force, current velocity and \textbf{mass}

- Where does that mass come from?
  - Class Body has a getter, but no setter!
  - It comes from the \textbf{Fixture} class
  - Fixture gives \textit{volume} to body

- Will revisit this later with collisions
Four Ways to Move a Dynamic Body

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

- **Impulses**
  - applyLinearImpulse
  - applyAngularImpulse

- **Velocity**
  - setLinearVelocity
  - setAngularVelocity

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

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

- **Impulses**
  - applyLinearImpulse
  - applyAngularImpulse

- **Velocity**
  - setLinearVelocity
  - setAngularVelocity

- **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

Shape: Box

Controls: Force

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

Density: 1  Friction: 0.1  Restitution: 0

Collisions
Four Ways to Move a Dynamic Body

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

- **Impulses**
  - applyLinearImpulse
  - applyAngularImpulse

- **Velocity**
  - setLinearVelocity
  - setAngularVelocity

- **Translation**
  - setTransform

**Must Cap Velocity**
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);
}
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

## Shape
- Stores the object geometry
  - Boxes, circles 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 \( \text{area} \times \text{density} \)

## 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

// 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);
Making a Box2D Physics Object

Other shapes possible

Also set friction and restitution parameters

Reason for separating Fixture & Body classes

// 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);
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
A Word on Units

- Size is **not** in pixels
  - 1 Box2D unit = 1 meter
  - Also 1 density = 1 kg/m²
  - Drawing scale in Lab 4

- This is **rescalable**
  - Could say 1 unit = 10 m
  - But must be consistent

- Box2d likes units *near* 1
  - Best if objects same size
  - Adjust scale so 1 default

1.5 B2d units
60 pixels

60 B2d units
60 pixels
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

**Shape:** Box

**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

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
Collision is About Fixtures!

- Capsule obstacle is two circles and rectangle
  - Allows smooth motion while walking
  - Feet do not get hung up on surfaces
- But may register **multiple collisions**!
Collision Filtering

• FixtureDef has a Filter attribute
  • categoryBits: Defines what can collide with it
  • maskBits: Defines what it can collide with
  • groupIndex: Collision group (overrides bits)

• Example:
  • Fixture A category x001, Fixture B category x010
  • Mask x101 or x001 only collides with A
  • Mask x011 collides with both A and B
Collision Filtering

- FixtureDef has a Filter attribute
  - categoryBits: Defines what can collide with it
  - maskBits: Defines what it can collide with
  - groupIndex: Collision group (overrides bits)

- Example:
  - Fixture A category: 001, Fixture B category: 010
  - Mask 101 or 001 only collides with A
  - Mask 011 collides with both A and B

Filtering means is never detected!
How about Sort-of-Filtering?

- Want a non-sensor object where
  - We always detect the collision
  - But sometimes ignore the restitution

- Method `beginContact` has a `Contact` parameter
  - Manages the physics while it resolves collision
  - Can call the method `contact.isEnabled(false)`
  - Turns off collision; `endContact` is never called

- See tutorials for “anatomy of a collision”
  - [https://www.iforce2d.net/b2dtut/collision-anatomy](https://www.iforce2d.net/b2dtut/collision-anatomy)
Recall: Tunneling

- 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
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**
  - Expensive!
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
More Collisions: RayCasting

- Method `rayCast` in world
  - Give it start, end of ray
  - Also a `RayCastCallback`
  - Executed when call step

- Invoked on **all collisions**
  - Not just the first on
  - Does not return in order!
  - This is for optimization

- Sight-cones = many rays
The RayCastCallback Interface

```c
float reportRayFixture(Fixture fixture, // Fixture found
    Vector2 point,     // Collision point
    Vector2 nom,      // Collision normal
    float fraction    // Fraction of ray
)
```

- Fraction is how far along ray (0 = start, 1 = end)
  - First collision is one with lowest fraction
  - But be prepared for larger fractions first

- Return value is optimization to limit search
  - Ignores collisions with fraction later than return
The RayCastCallback Interface

```c
float reportRayFixture(Fixture fixture, // Fixture found
                      Vector2 point,    // Collision point
                      Vector2 nom,      // Collision normal
                      float fraction    // Fraction of ray
                    )
```

- Fraction is how far along ray (0 = start, 1 = end)
  - First collision is one with lowest fraction
  - But be prepared for larger fractions first

- Return value is optimization to limit search
  - Ignores collisions with fraction later than return
AABB Queries

- **Bounding Box** queries
  - Find all fixtures in box
  - Must be *axis aligned*
  - Rotation not allowed

- Similar to raycasting
  - Provide callback listener
  - Call step method in world
  - Prepare for many matches

- **Application**: selection
  - See Ragdoll Demo
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

Collisions
Sample Joint Types

Distance Joint

- Hard constraint
- Strong but very brittle
- Primary chain/rope joint

Rope Joint

- Soft constraint
- Stretchy but very weak
- More for reinforcement
Sample Joint Types

**Revolute**
- Joint binds at one point
- Both translate together
- But rotate **independently**

**Weld**
- Joint binds at one point
- Both translate together
- Both rotate together
Sample Joint Types

**Prismatic**
- Joint binds with a “track”
- Both rotate together
- But *translate along track*

**Pulley**
- Joint binds through portals
- Pulling one raises the other
- *Distance* w/ “teleportation”
Summary

• Box2d 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

• Collisions are managed by callback functions
  • Invoked once you call the world step method
  • Collisions are processed per fixture, not per body