

Lecture 18

box2d Physics

Physics in Games

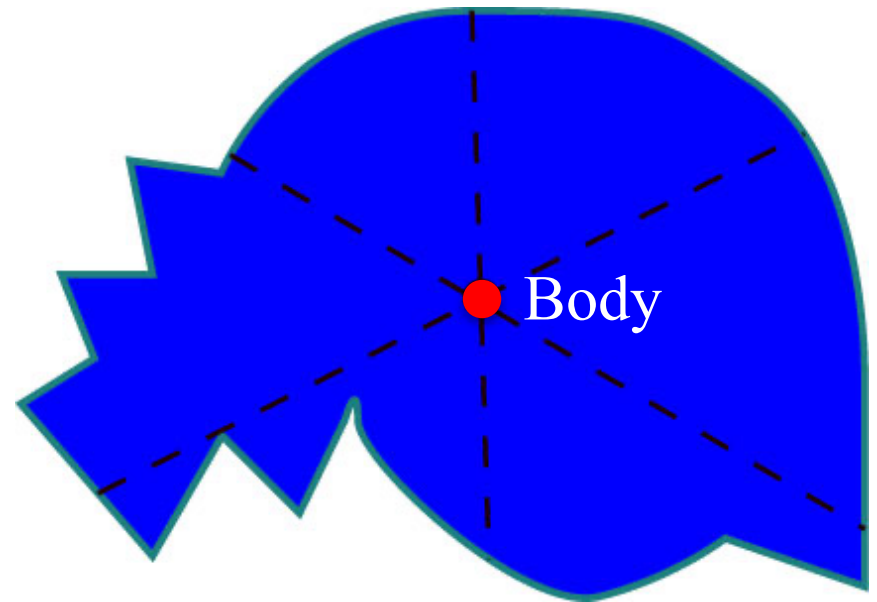
- **Moving** objects about the screen
 - **Kinematics**: Motion ignoring external forces
(Only consider position, velocity, acceleration)
 - **Dynamics**: The effect of forces on the screen
- **Collisions** between objects
 - **Collision Detection**: Did a collision occur?
 - **Collision Resolution**: What do we do?

Physics in Games

- **Moving** objects about the screen
 - **Kinematics**: Motion ignoring forces
(Class **Body**)
 - **Dynamics**: The effect of forces on the screen
- **Collisions** between objects
 - **Collision Detection**: How do we do?
(Class **Fixture**)
 - **Collision Response**: How do we do?

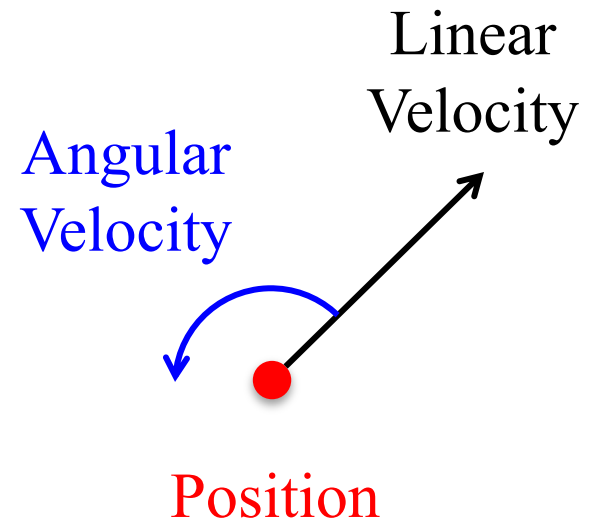
Body in box2d

- 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

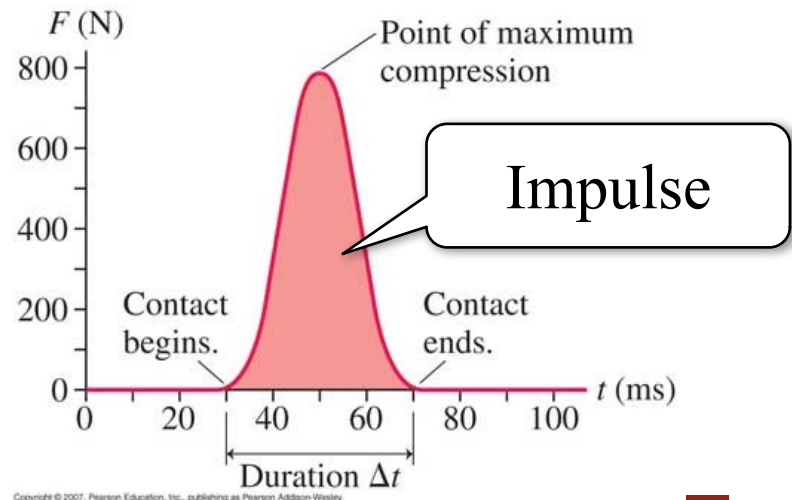
Forces

- 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



Copyright © 2007, Pearson Education, Inc., publishing as Pearson Addison-Wesley

Forces vs. Impulses

Forces

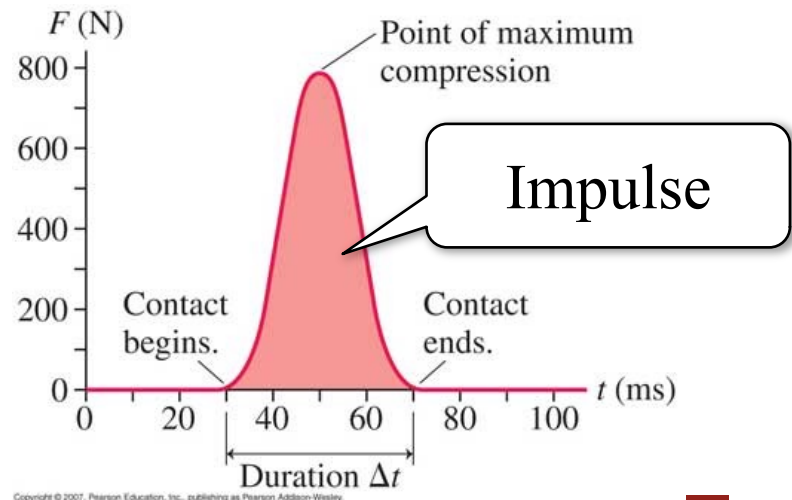
- Instantaneous push
 - To be applied over time
 - Gradually accelerates
 - Momentum if sustained

Impulses

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

Impulse = Force x **1 Sec**

in Box2D



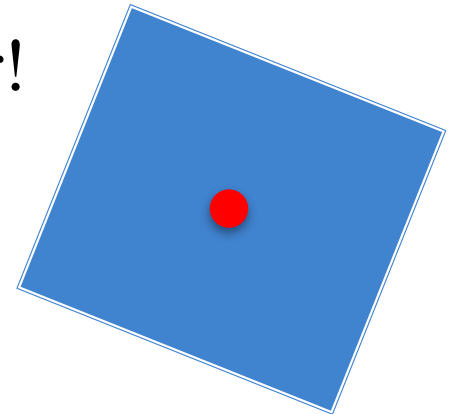
Copyright © 2007, Pearson Education, Inc., publishing as Pearson Addison-Wesley

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

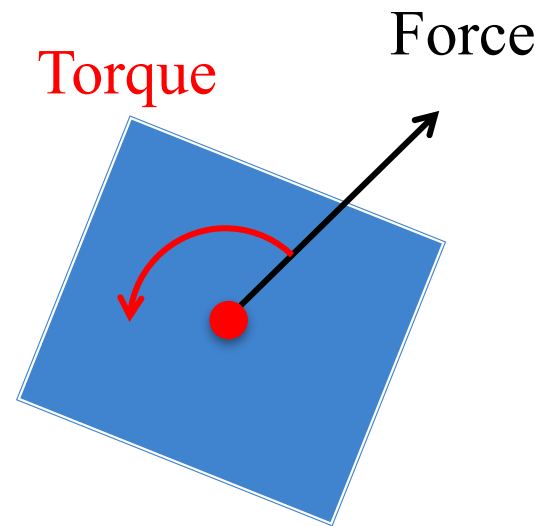
Force and Acceleration

- 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 **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



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)

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

- **Impulses**

- `applyLinearImpulse`
- `applyAngularImpulse`

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

- **Velocity**

- `setLinearVelocity`
- `setAngularVelocity`

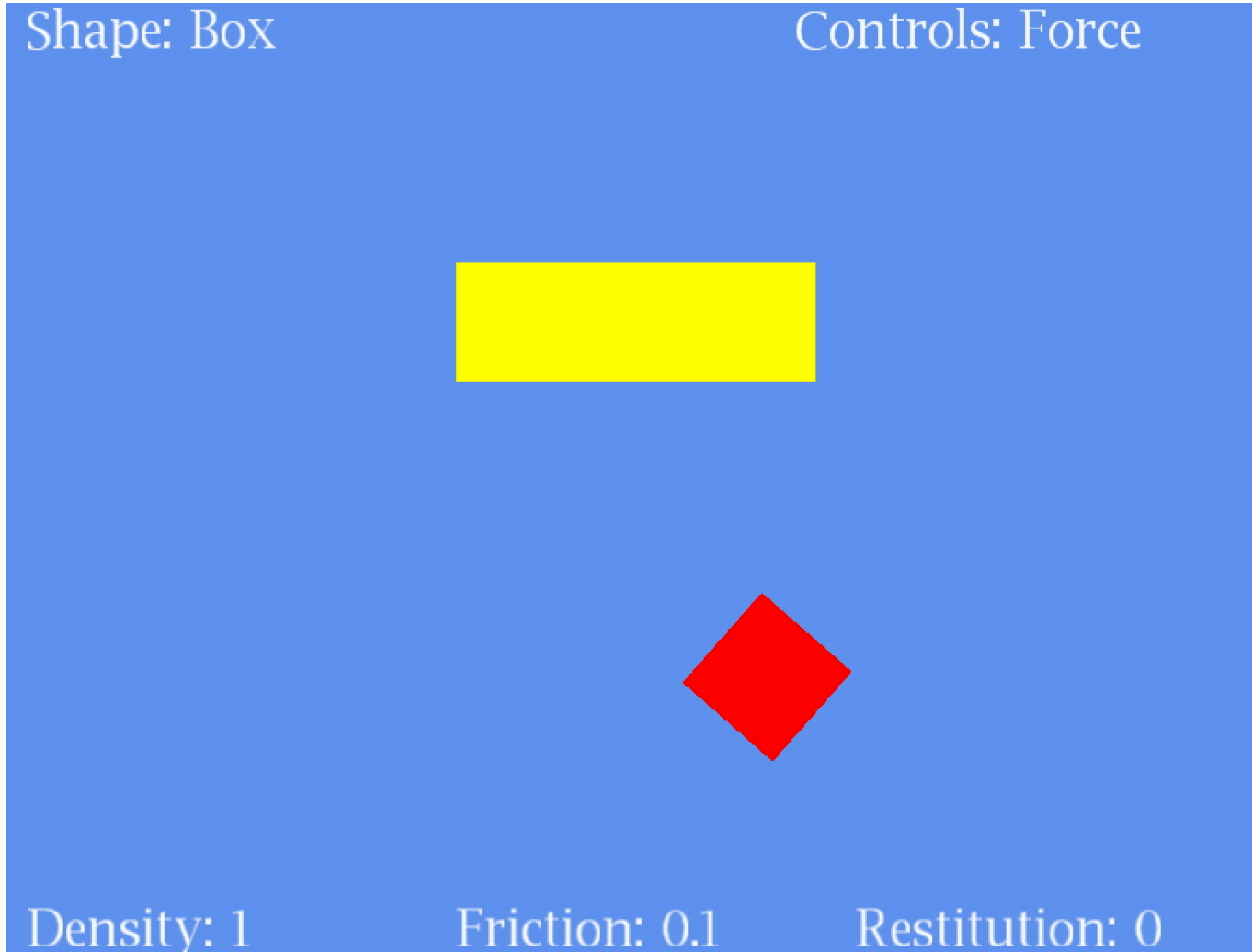
- Bad for joints, complex shapes
- Excellent response to user input
- Very easy to control

- **Translation**

- `setTransform`

- **Completely ignores physics!**
- Very easy to control

Example: box2d Demo



Example: box2d Demo

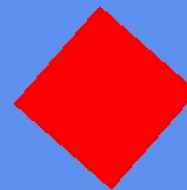
Shape: Box

Controls: Force



Controls:

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



Density: 1

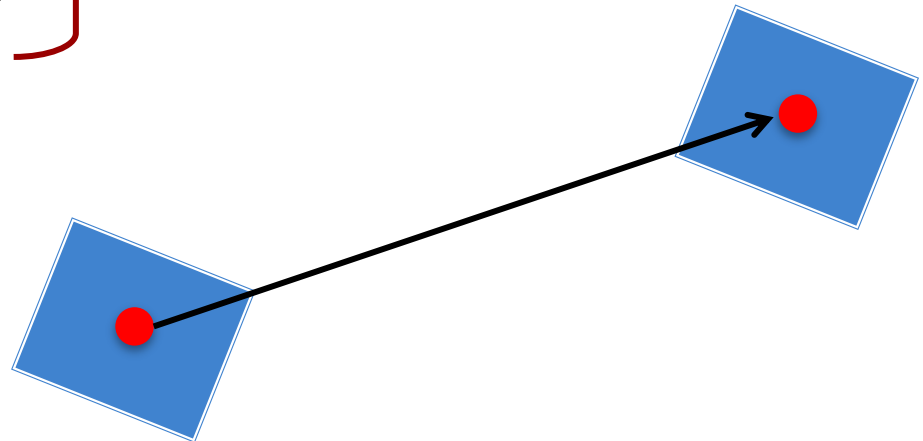
Friction: 0.1

Restitution: 0

Four Ways to Move a Dynamic Body

- **Forces**
 - applyForce (linear)
 - applyTorque (angular)
- **Impulses**
 - applyLinearImpulse
 - applyAngularImpulse
- **Velocity**
 - setLinearVelocity
 - setAngularVelocity
- **Translation**
 - setTransform

Must Cap Velocity

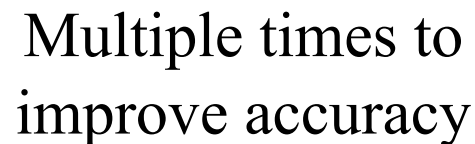


Basic Structure of a Update Loop

```
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

```
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

Basic Structure of a Update Loop

```
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);  
}
```

**Only before
first iteration!**

Multiple times to
improve accuracy

Collision Objects in box2d

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);
```



Normal Allocation



Optimized Allocation

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

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

Also set **friction** and **restitution** parameters

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

Reason for separating **Fixture** & **Body** classes

```
// 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;

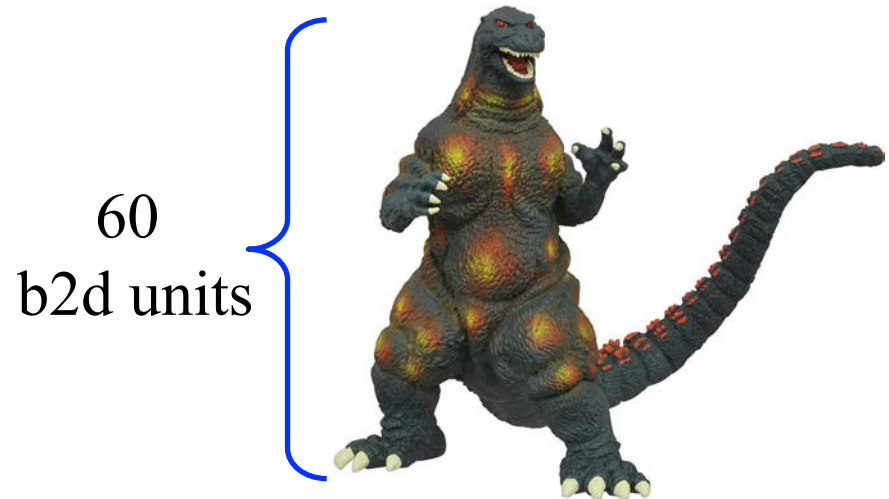
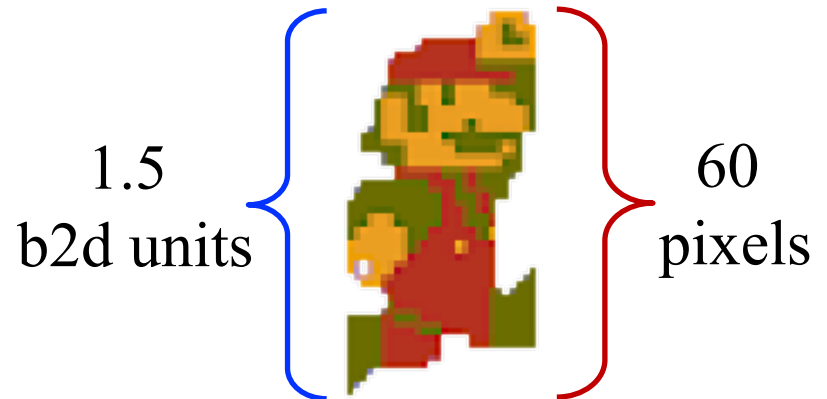
// Attach the two shapes to body
fixdef.shape = shape1;
fixture1 = body1.createFixture(fixdef);
fixdef.shape = shape2;
fixture2 = body1.createFixture(fixdef);
```


Observations on Fixture Parameters

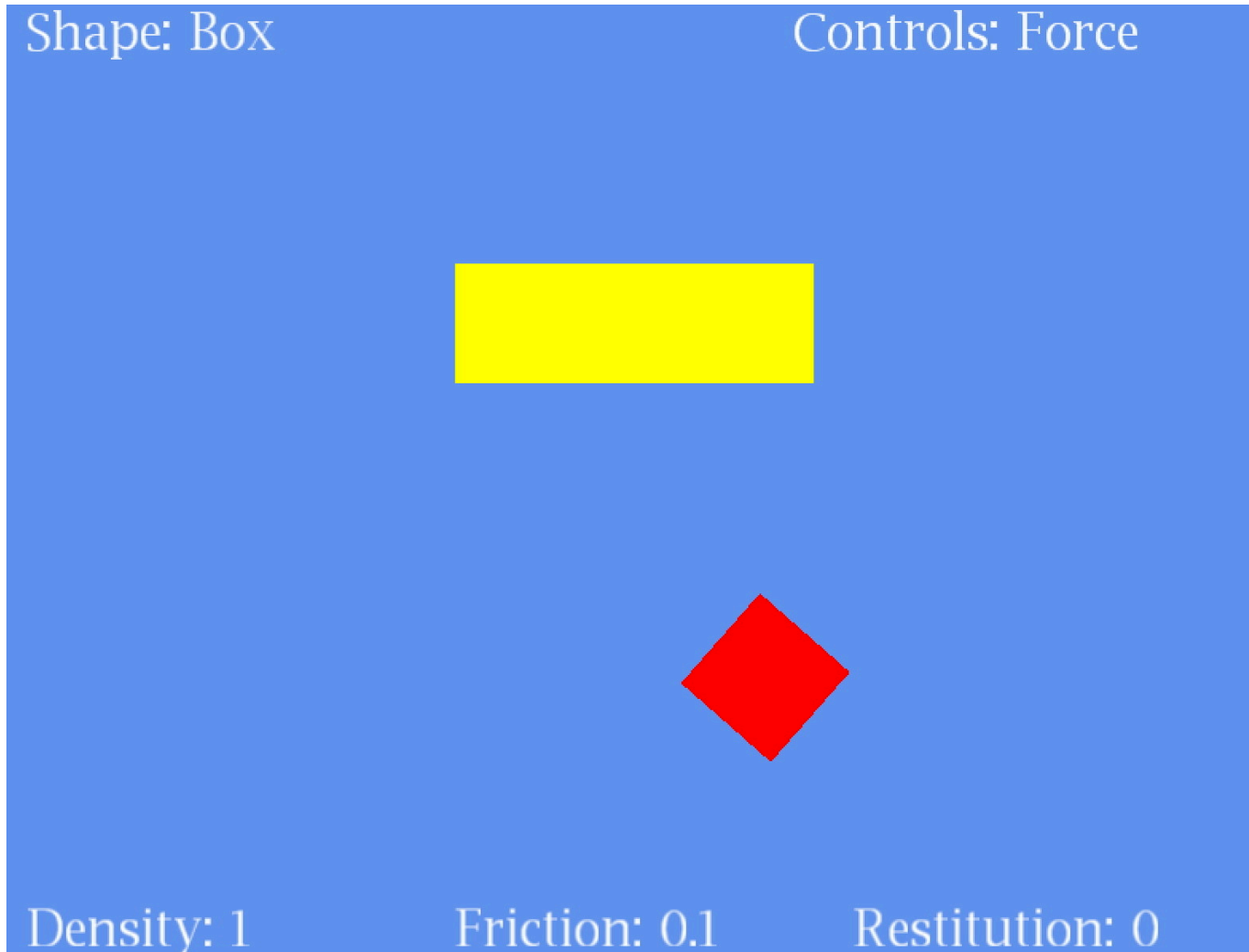
- **Density** can be anything **non-zero**
 - The higher the density the higher the mass
 - Heavier objects are harder to move
- **Friction** should be within **0 to 1**
 - Can be larger, but effects are unpredictable
 - Affects everything, even manual velocity control
- **Restitution** should be within **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



Example: Box2D Demo



Example: Box2D Demo

Shape: Box

Controls: Force

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

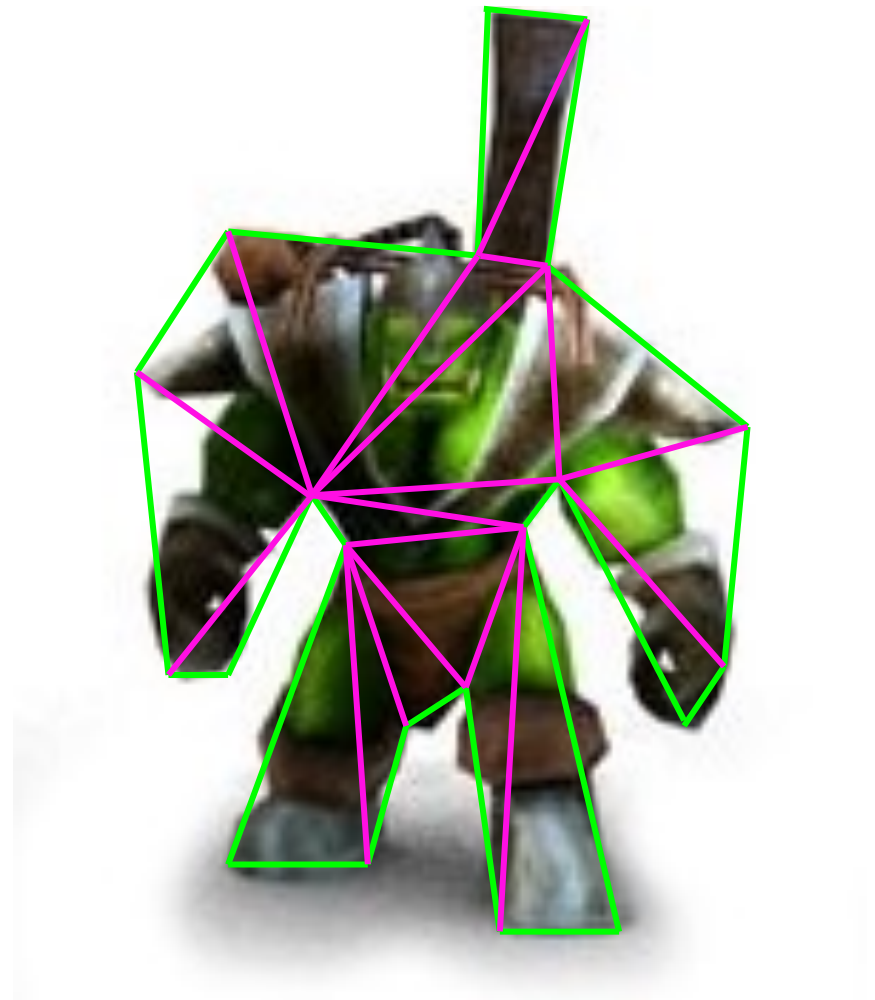
Density: 1

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



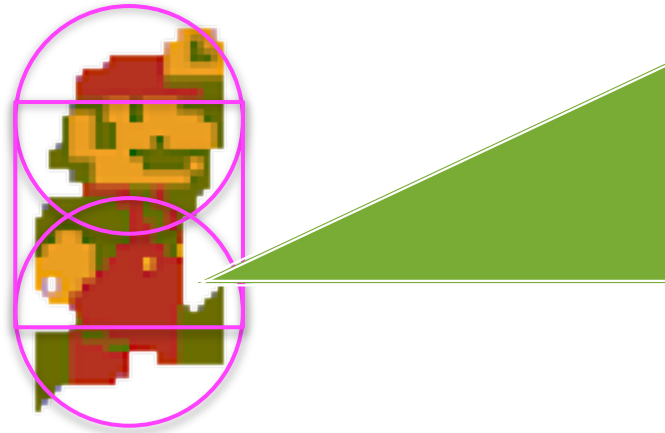
character.shape

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

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:**

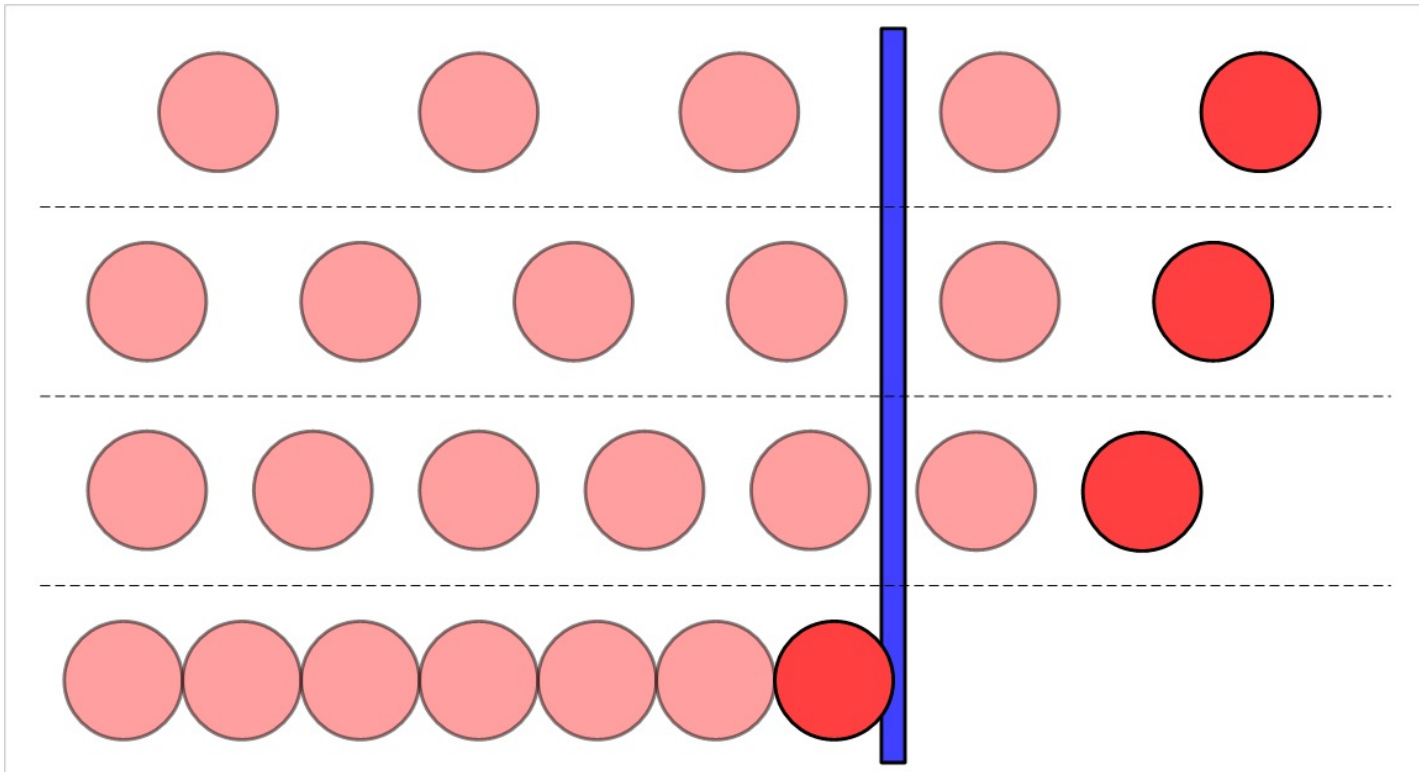
- Fixture A category `001` Fixture B category `10`
- Filtering means is never detected!
- Collision between A and B

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>

Recall: Tunneling

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

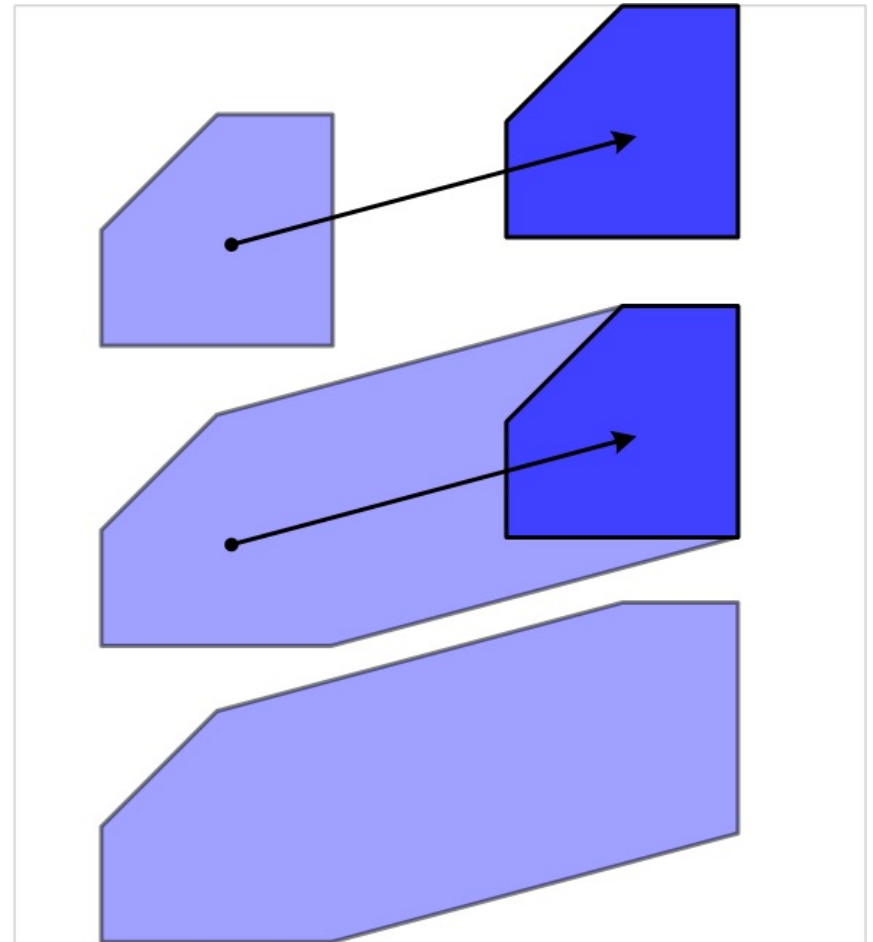


Possible Solutions to Tunnelling

- 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**

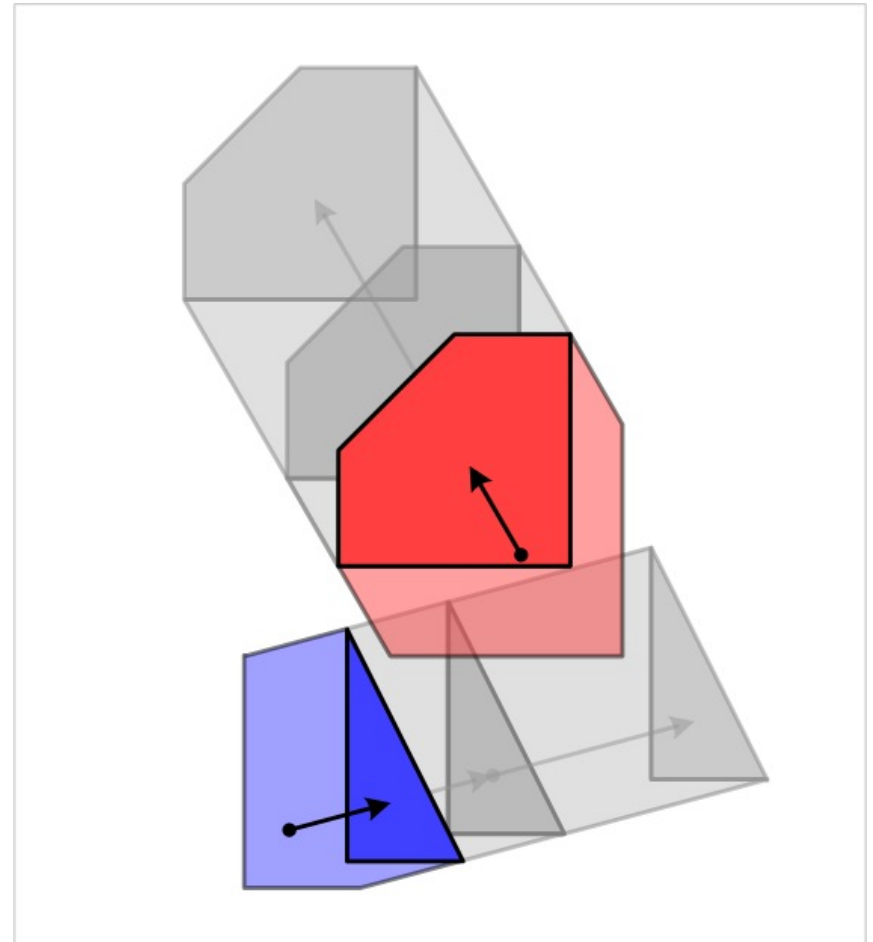
Swept Shapes

- 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



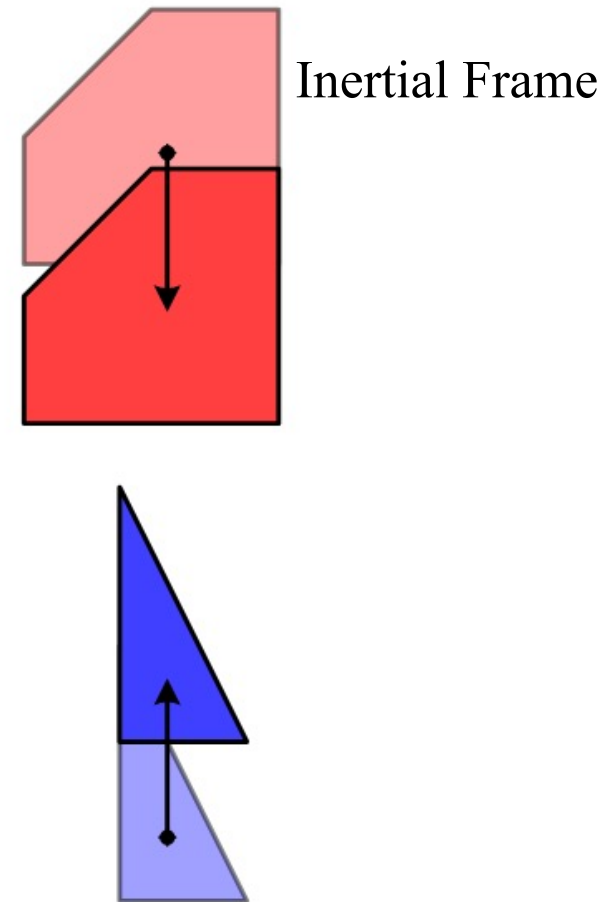
Swept Shapes

- 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



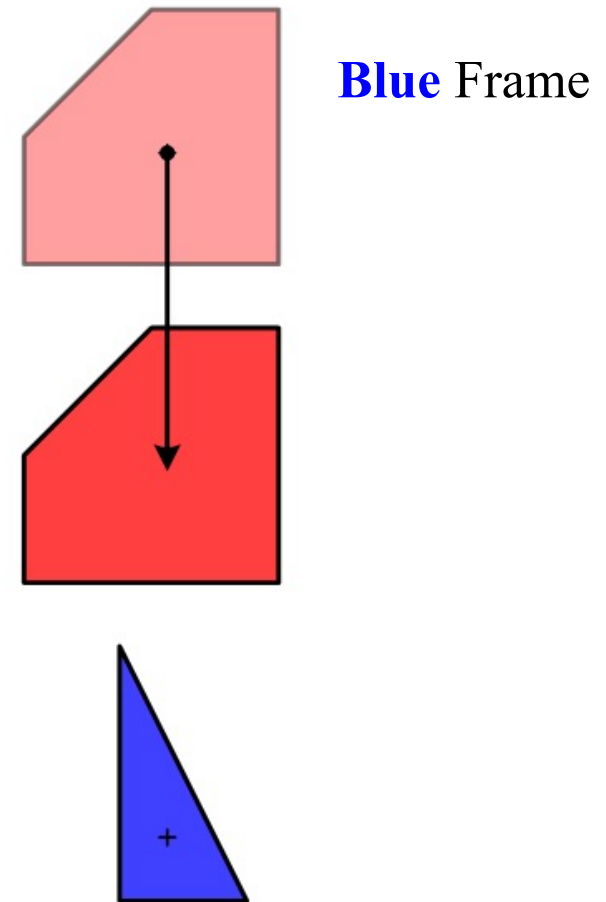
Swept Shapes & Relative Coordinates

- **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



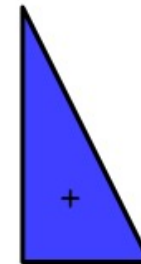
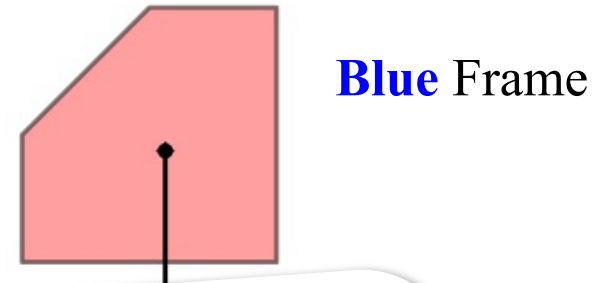
Swept Shapes & Relative Coordinates

- **False positives** happen if:
 - Two objects are moving
 - Swept shapes intersect at different intersection times
- What if only one object is moving?
 - Swept in one direction
 - So no false positives

How “Bullets” are handled

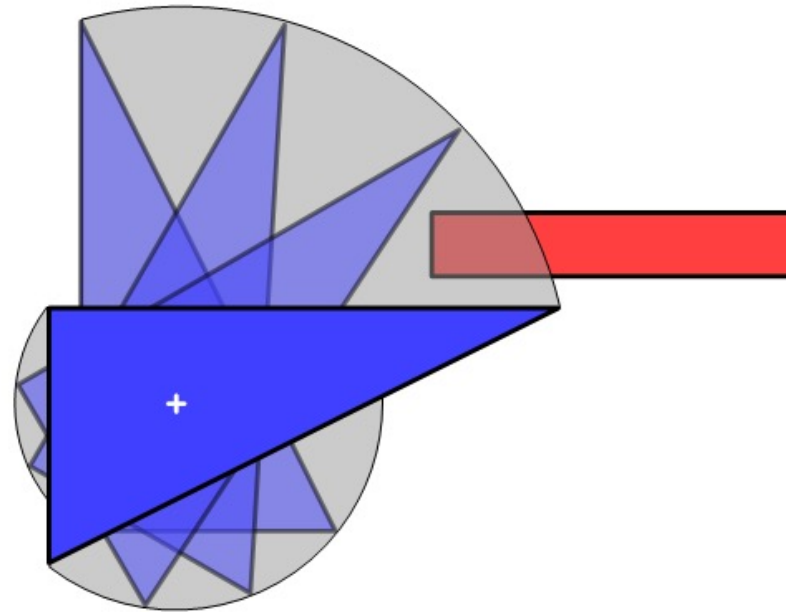
- Change **reference frames**

Expensive!

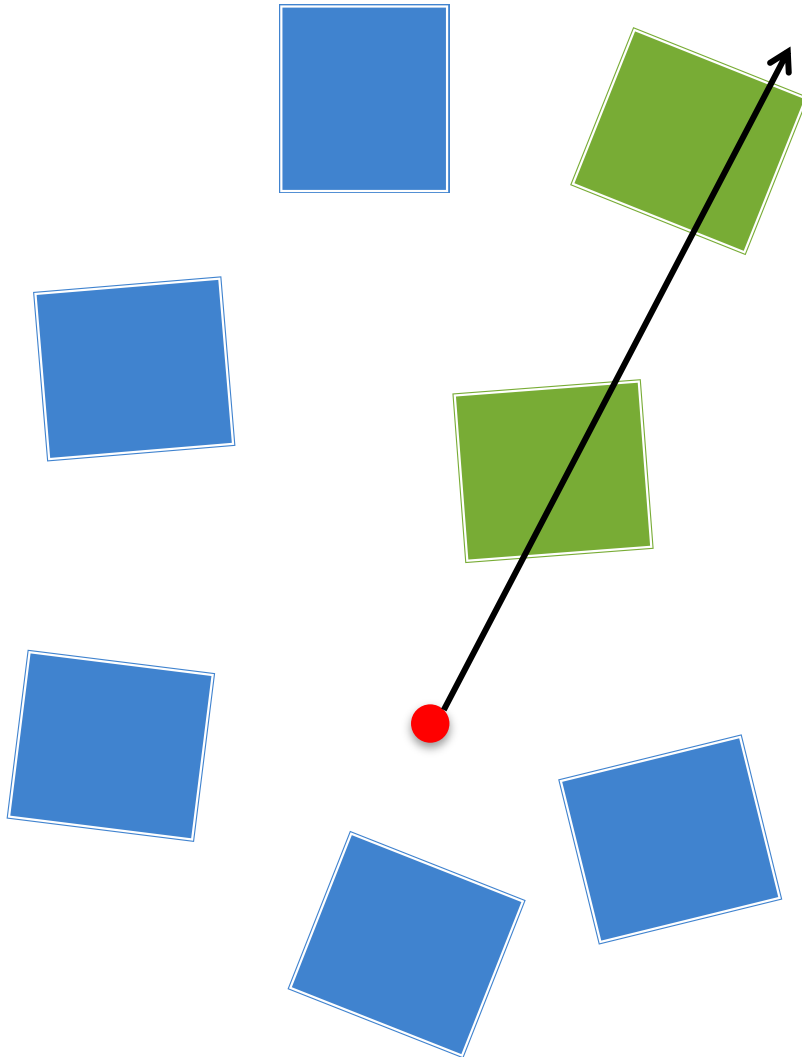


Rotations Suck

- 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

```
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

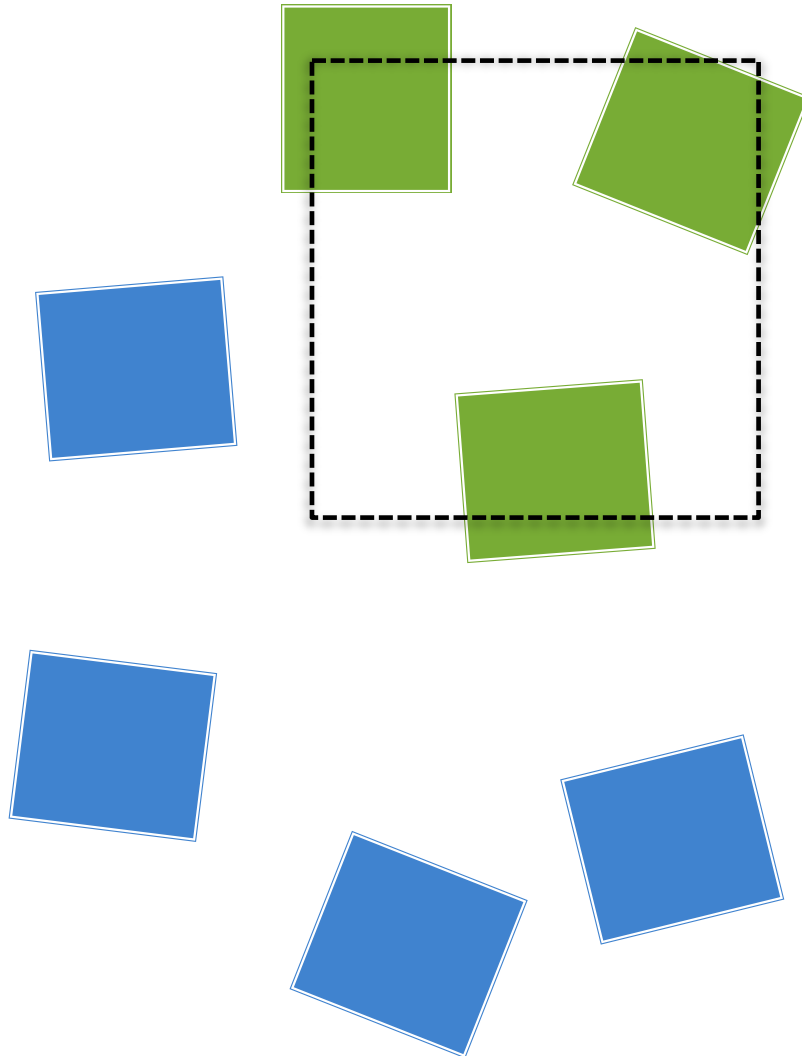
The RayCastCallback Interface

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

Allowed fraction
for future matches

- 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

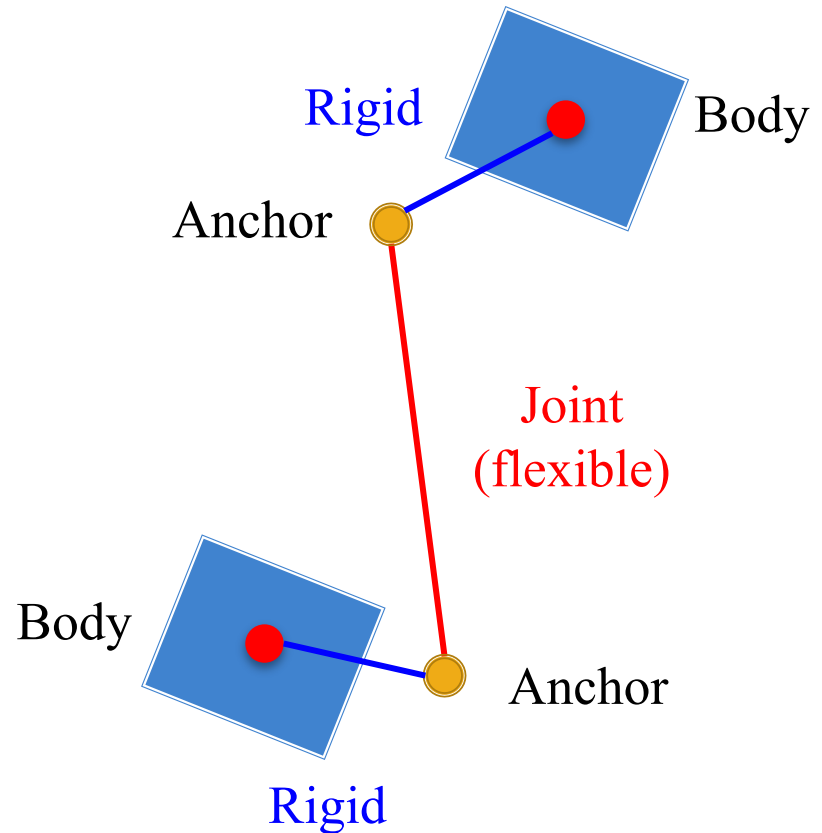
AABB Queries



- **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

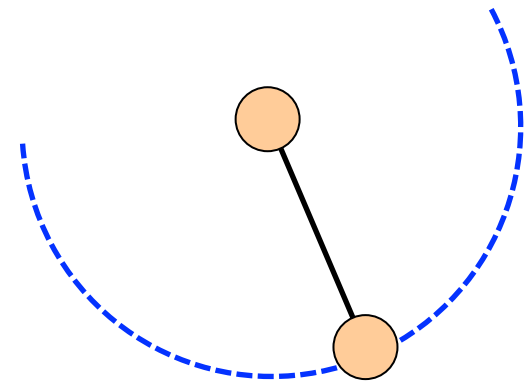
- Joints connect **bodies**
 - Anchors can be offset body
 - Coordinates relative to body
- Are affected by **fixtures**
 - Fixtures prevent collisions
 - Limit relative movement
- Must control with forces
 - Manual velocity might violate constraints
 - Use force or impulse



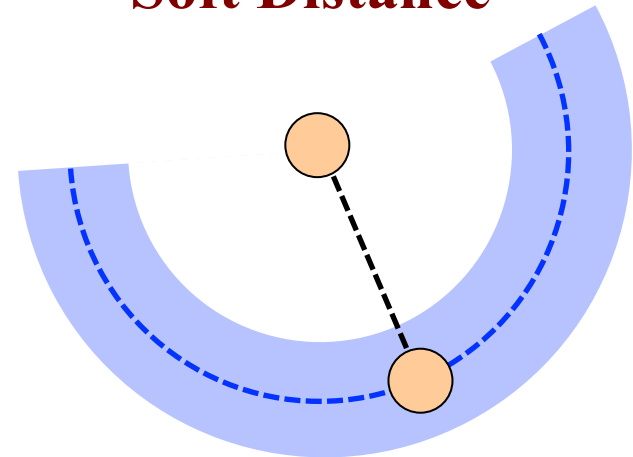
The Distance Joint

- Extremely common joint
 - Separates by a fixed amount
 - Good for ropes/grappling
- Can be **hard** or **soft**
 - **Hard**: **Strong** but very **brittle**
 - **Soft**: **Stretchy** but very **weak**
- Softness set in the **joint def**
 - Damping, frequency values
 - Turns the joint into a **spring**
 - **Damping**: Use <1 to soften
 - **Frequency**: Spring oscillation

Hard Distance



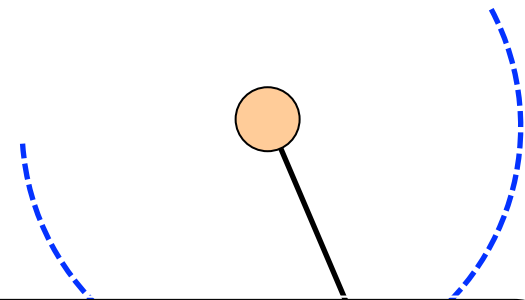
Soft Distance



The Distance Joint

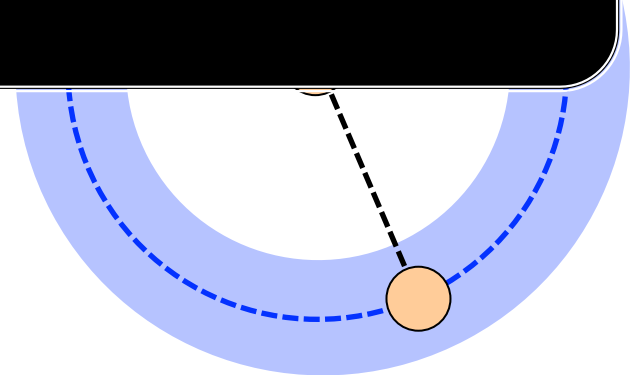
- Extremely common joint
 - Separates by a fixed amount
 - Good for ropes/grappling
- Can be **hard** or **soft**

Hard Distance



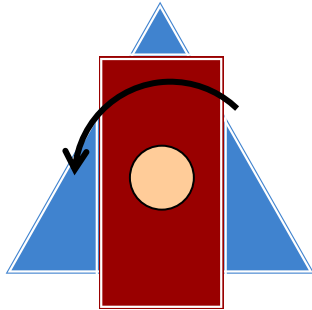
Older versions of box2d have a rope joint.
This is **deprecated** in favor of soft distances.

- Turns the joint into a **spring**
- **Damping**: Use <1 to soften
- **Frequency**: Spring oscillation



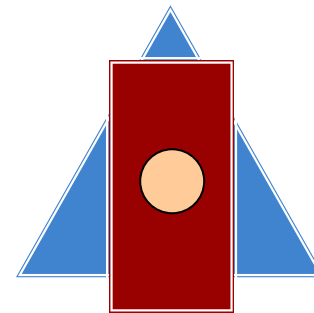
Other Joint Types

Revolute



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

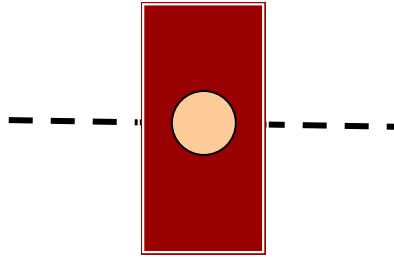
Weld



- Joint binds at one point
- Both translate together
- Both rotate together

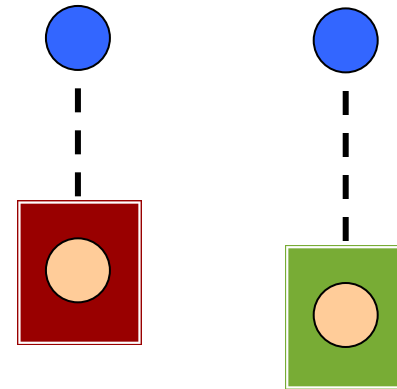
Other Joint Types

Prismatic



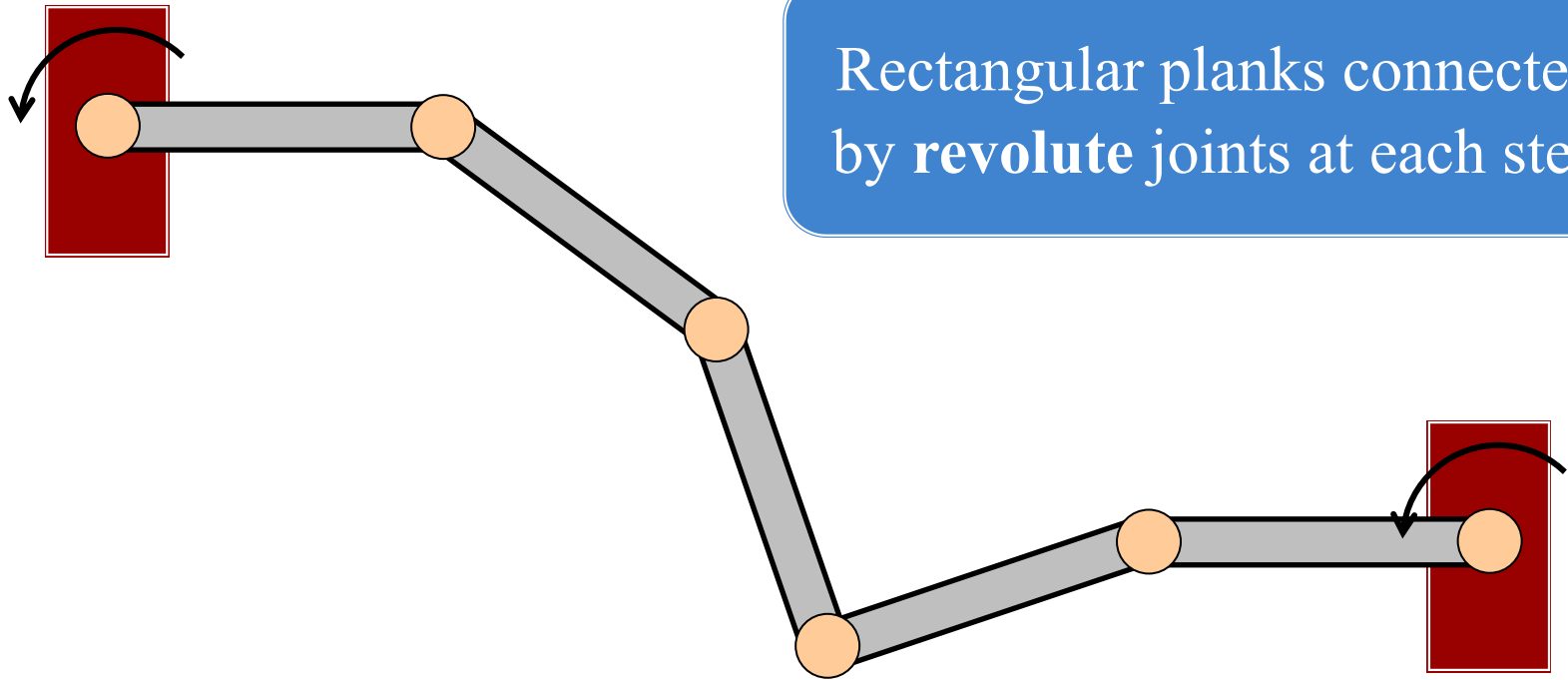
- Joint binds with a “track”
- Both rotate together
- But **translate along track**

Pulley



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

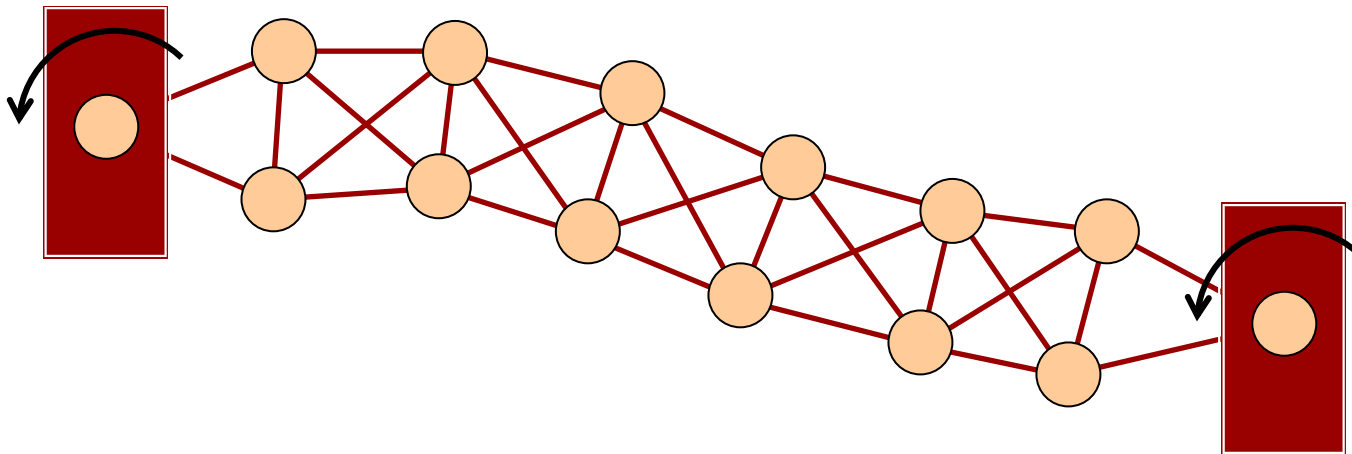
Making a Rope: The Simple Way



Bridge in Lab 4

Making a Rope: The Better Way

Web of springy **distance** joints
with revolute joints at the end



Keeps rope strong but flexible!

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