
Collision
Detection

Tools for collision detection

Unary operators (for a single shape):

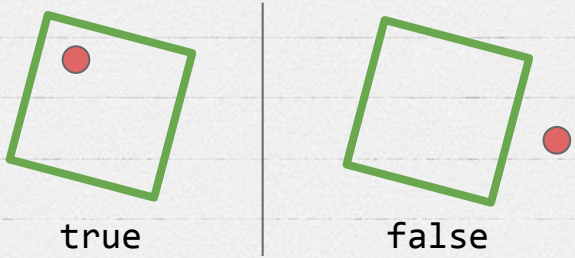
- ✗ Point test (see **TODO 1**)
- ✗ Raycast (see **TODO 2**)

Binary functions (for two shapes):

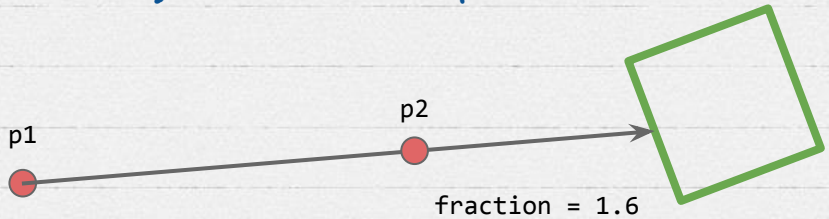
- ✗ Overlap
- ✗ Contact manifolds
- ✗ Distance
- ✗ Time of Impact

Unary operators

Test point: Check if a point is inside a shape



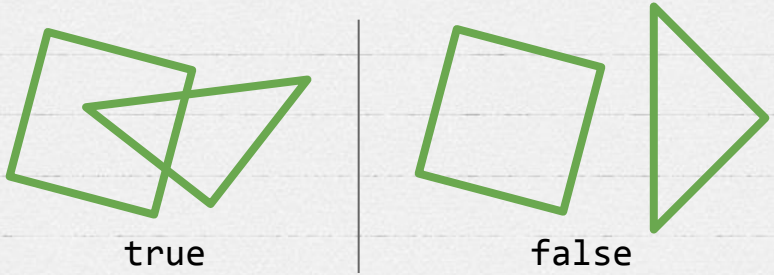
RayCast: Cast a ray between two points



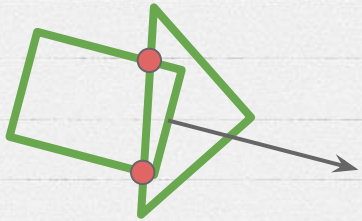
Raycast works with fractions, ie, normalized value of the distance between p1 and p2

Binary functions

Overlap: Check if two shapes are overlapping

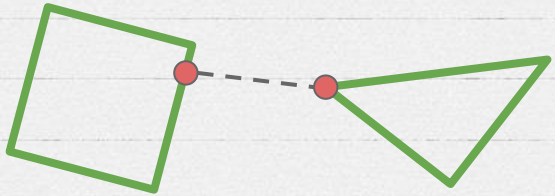


Contact manifolds: Offers information about contact points and normal/s

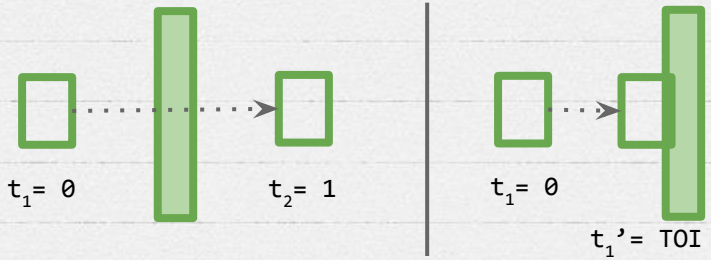


Binary functions

Distance: Returns the shortest distance between two shapes

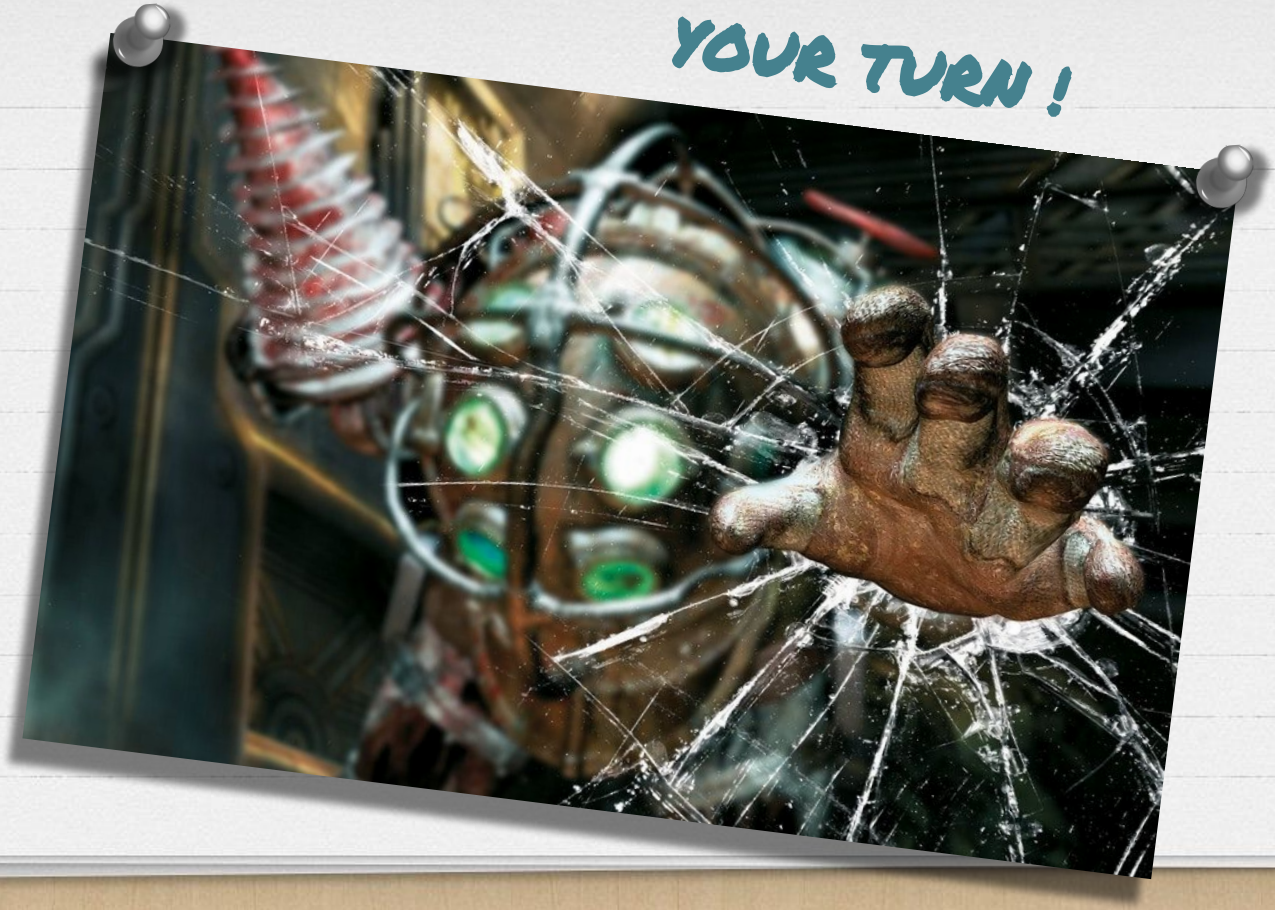


Time of impact: The time that two shapes collide. Used to avoid *tunneling*



Continuous collision detection (CCD) uses TimeOfImpact between static and dynamic bodies to avoid going outside a static level (tunneling)

YOUR TURN !



TODO No 1

```
// TODO 1: Write the code to return true in case the point
// is inside ANY of the shapes contained by this body
```

- ✗ You have to iterate over all the shapes on a body and check whether the point is inside or not
- ✗ You need to pass the body transformation (*body->GetTransform()*) to the test point method
- ✗ Remember the pixels ⇔ meters conversion!

TODO No 2

```
// TODO 2: Write code to test a ray cast between both points provided. If not hit return -1
// if hit, fill normal_x and normal_y and return the distance between x1,y1 and its
colliding point
```

- ✗ Very similar to the previous TODO. It needs shape iteration and body transformation.
- ✗ You have to use the *output.fraction* to calculate the corresponding distance

How to get collision information?

If we could detect the precise moment of collision we would be able to:

- ✗ Trigger an sound effect
- ✗ Trigger an animation
- ✗ Add some points to the final score
- ✗ Add more velocity to the ball
- ✗ ....

How to get collision information?

Check this excellent tutorial about the anatomy of a collision on Box2D!

In order to obtain this information, we have to options. Check the full list of contacts or each body contact list, or ...

```
for (b2Contact* contact = world->GetContactList(); contact; contact = contact->GetNext())
     contact->... //do something with the contact

 for (b2ContactEdge* edge = body->GetContactList(); edge; edge = edge->next)
     edge->contact->... //do something with the contact
```

How to get collision information?

Check this excellent tutorial about the anatomy of a collision on Box2D!

...add contact listeners. We can override the different methods during a collision from the class *b2ContactListener*.

```
void BeginContact(b2Contact* contact);
void EndContact(b2Contact* contact);
void PreSolve(b2Contact* contact, const b2Manifold* oldManifold);
void PostSolve(b2Contact* contact, const b2ContactImpulse* impulse);
```

Road map

3

ModulePhysics needs to be a contact listener (inherit from **b2ContactListener**). Now, we can override:

```
void BeginContact(b2Contact* contact);
```

4

We could extract information of the bodies that produce the collision from b2Contact. But these bodies are Box2D bodies. But we need a pointer to our PhysBody class!

Luckily, b2Body have an extra storage pointer to save whatever we want:

```
b2Body->SetUserData(void *);
b2Body->SetUserData(physBody);
```

5

We will add an *OnCollision* virtual method on the Module class that will contain the two PhysBodies as arguments.

6

On PhysBody class, we will add a listener (the module that will response with its OnCollision)

```
Module *listener;
```

7

On each PhysBody, we call OnCollision on its listener:

```
physBodyA->listener->OnCollision(physBodyA, physBodyB);
```

YOUR TURN (AGAIN)!

TODO No 3

```
// TODO 3: Make module physics inherit from b2ContactListener
// then override void BeginContact(b2Contact* contact)
// You need to make ModulePhysics class a contact listener
```

Remember to add the ModulePhysics as a listener!

```
world->SetContactListener(b2ContactListener*)
```

Override the *BeginContact* method and create a *LOG("Collision")* to check if it works!

TODO No 4

```
// TODO 4: Add a pointer to PhysBody as UserData to the body
```

When *BeginContact* is called, we need to access our *PhysBody* bodies. Luckily, Box2D bodies offer an extra storage for our convenience. We can save the address of any pointer in the UserData, so we will save our PhysBody pointers upon any body creation.

Once done, in the collision callback we can obtain both PhysBodies that take part in the collision.

TODO No 5

```
// TODO 5: Create a OnCollision method that receives both PhysBodies
```

- ✗ Each module will specialize the method according to its purposes
- ✗ The method has the two PhysBody involved in the collision as arguments
- ✗ Avoid includes! Use forward declarations

TODO No 6

```
// TODO 6: Add a pointer to a module that might want to
listen to a collision from this body
```

- ✗ The PhysBody will keep a pointer (or not) to the module that will listen to the collision of this PhysBody
- ✗ Make sure that this pointer is NULL in the constructor and add this module's pointer where you want (in the PhysBody constructor, in a public method, as a public member, ...)

TODO No 7

```
// TODO 7: Call the listeners that are not NULL
```

In *BeginContact()* now we can call the OnCollision() methods for each module acting as a listener. Call it in both PhysBodies if the listeners exist

TODO No 8

```
// TODO 8: Make sure to add yourself as collision callback to the circle you creates
// Now just define collision callback for the circle and play bonus_fx audio
```

- ✗ Add the ModuleSceneIntro module as a listener for each circle created
- ✗ Define the OnCollision method on this module and play the FX included for each circle collision

Homework

Sensors!! You will need for the assignment! They are the same concept as *triggers* in Unity.

Sensors doesn't offer the callback functionality, so you have to iterate all the contacts and keep those that IsTouching( ) method returns true.

And then, you can call collision callback in the same way that we did it before.

Next Week . . .

Joints