Physics II

CITM

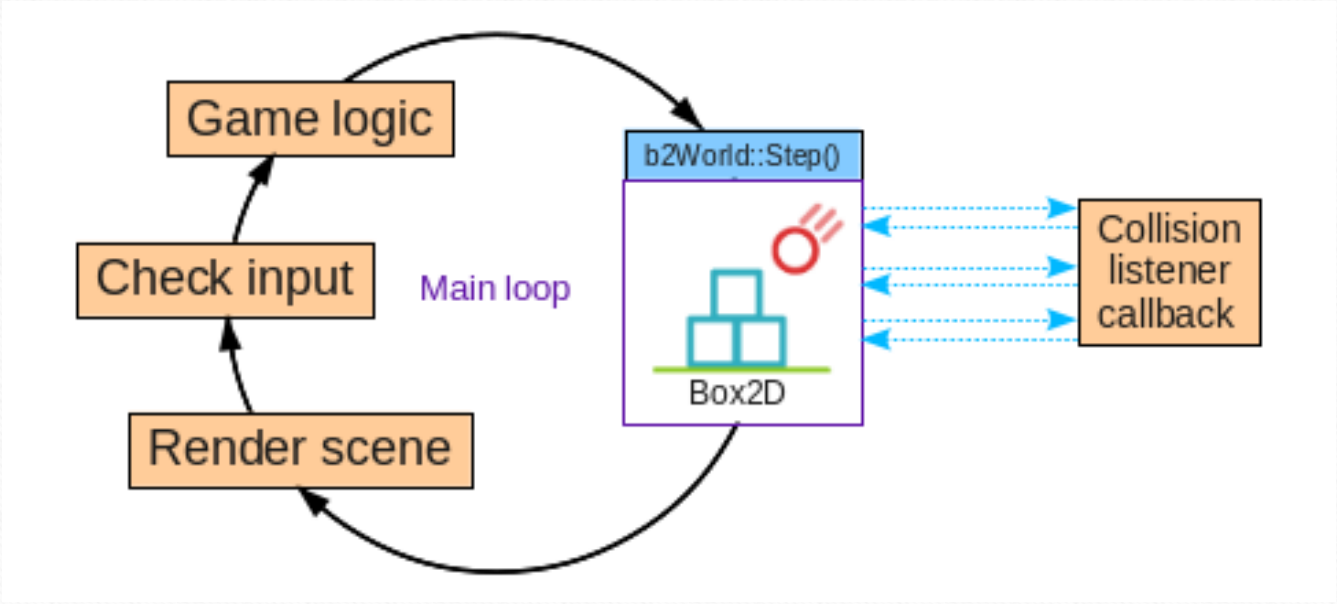
Box2d – Collisions

Collision Module

- It manages the creation of shapes, collision tests and binary functions:
- **Shapes**: circles, polygons and chains
- **Binary functions (pairwise)**: Overlap, Contact, Distance, Time of Impact
- **Tests**: point, raycast

Collision Callbacks

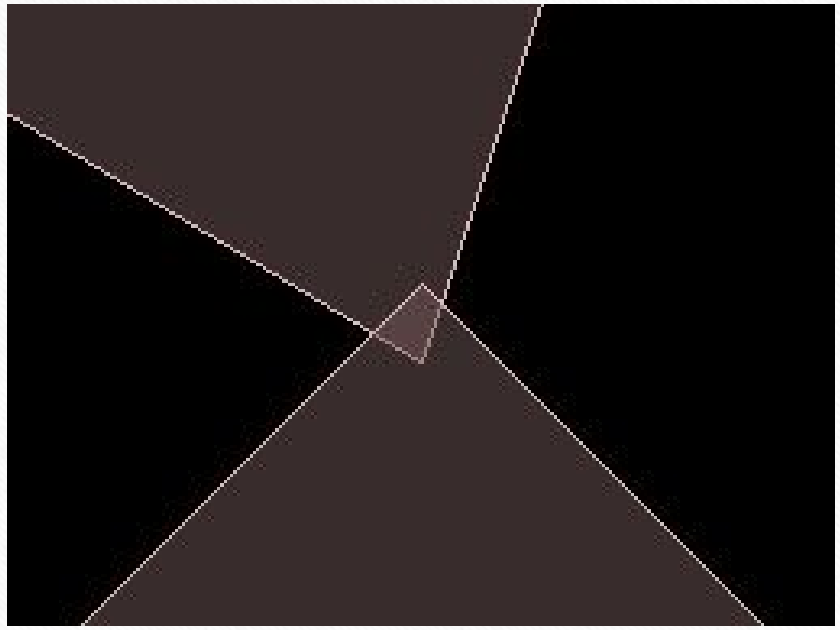
- When Box2D detects a collision, we want to do something (play audio!)
- We "kidnap" (override) the execution of Box2D when a collision happens, do our stuff, then return the execution to Box2D → ***Callback Listeners***.



Overlap

- Just tells if two shapes overlap in any point:

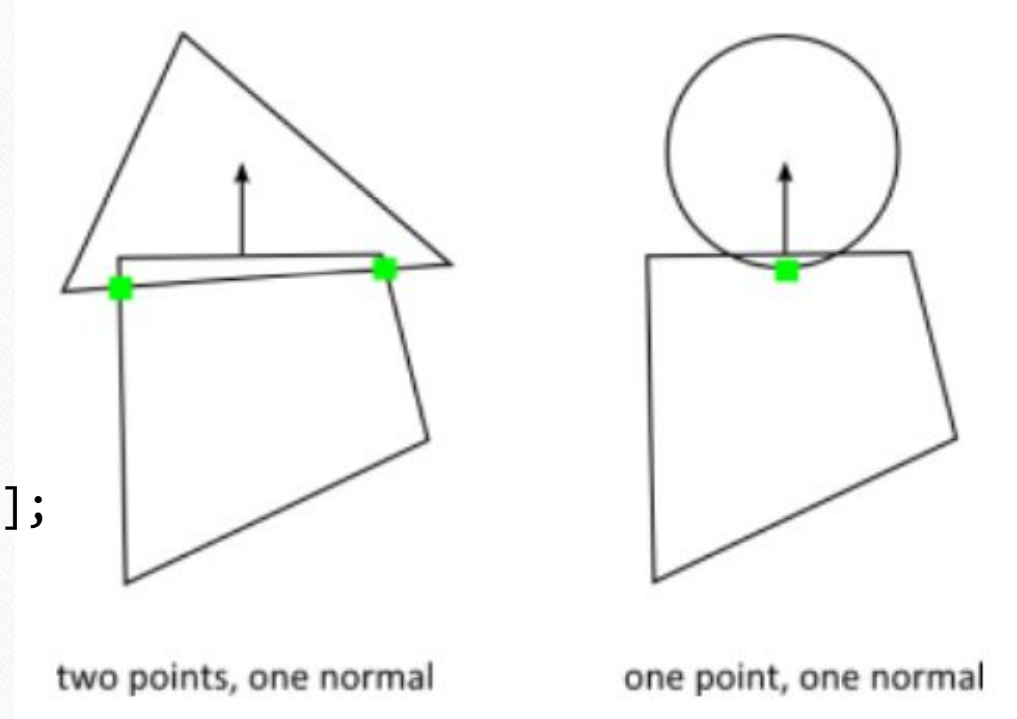
```
bool overlap = b2TestOverlap(
    shapeA, (indexA),
    shapeB, (indexB),
    transformA,
    transformB);

body->GetFixtureList()->GetShape()
body->GetTransform()
```



Contact

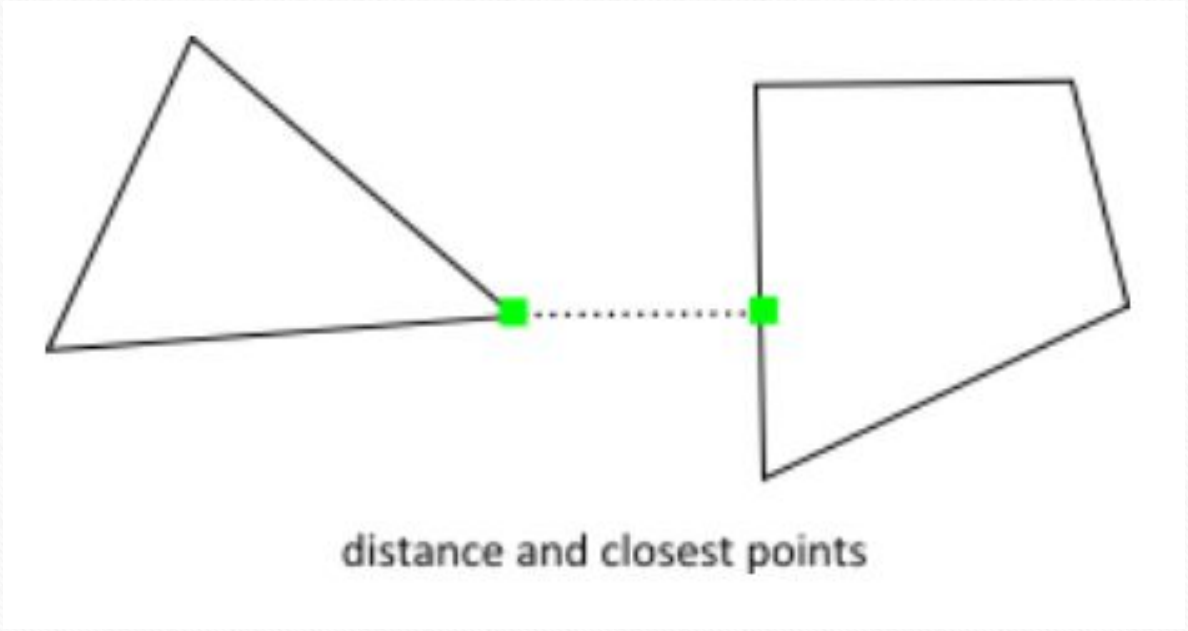
- Given two shapes, returns point in space of contact and normals:

```
b2WorldManifold worldManifold;

worldManifold.Initialize(&manifold,
transformA, shapeA.m_radius,
transformB, shapeB.m_radius);


for (… i < manifold.pointCount…) {
    b2Vec2 point = worldManifold.points[i];
    ...
}
```



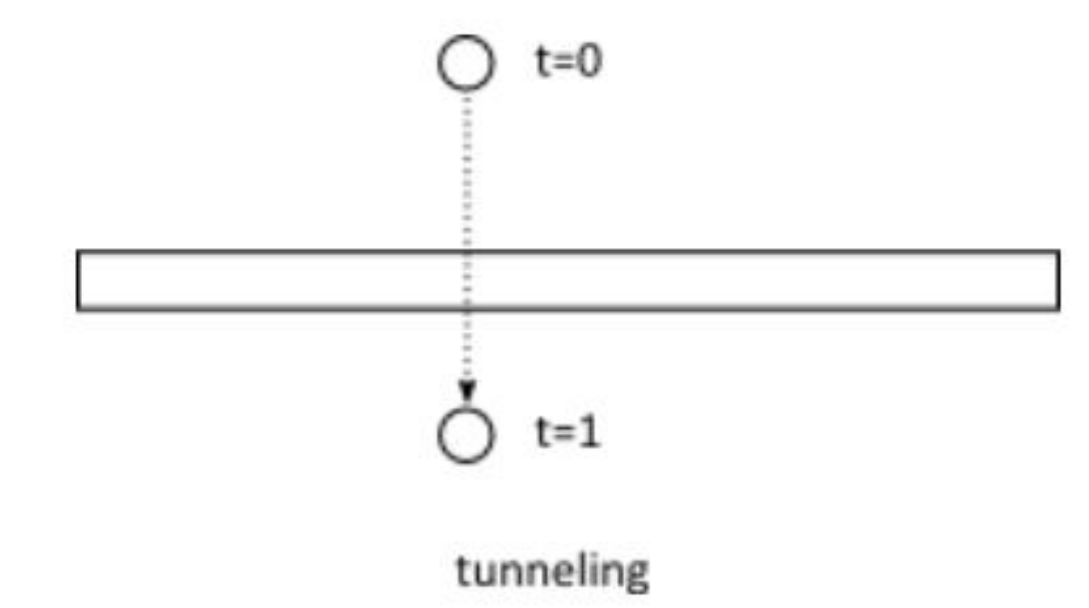
Distance

- Given two shapes, returns distance and closest points (**b2Distance**)
- The function needs both shapes to be converted into a **b2DistanceProxy**



Time of Impact

- Given two shapes, **b2TimeOfImpact** returns seconds (in simulation) to collision (TOI = Time Of Impact)
- Accounts for trans/rotation, but if they are too large it may miss a collision.



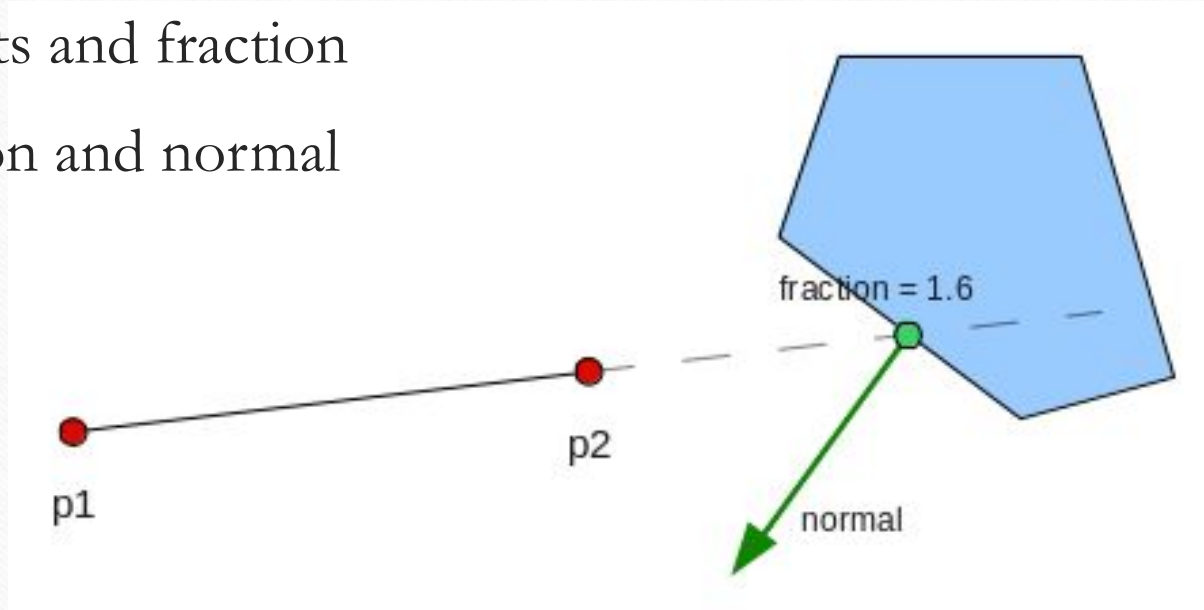
TODO 1

Write the code to return true in case an input point is inside ANY of the shapes (body->GetFixtureList) contained by the body we store in PhysBody.

- We can test if a point is inside a shape
- It is a method (TestPoint) inside **shape** (body → many fixtures → 1 shape)
- Needs to know its current transformation (body→GetTransformation())
- Remember to always transform from pixels to meters!

Raycasting

- Method per shape (b2Fixture::RayCast), it needs the transformation too
- Gets two points and fraction
- Returns fraction and normal



- http://www.iforce2d.net/b2dtut/raycasting

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.

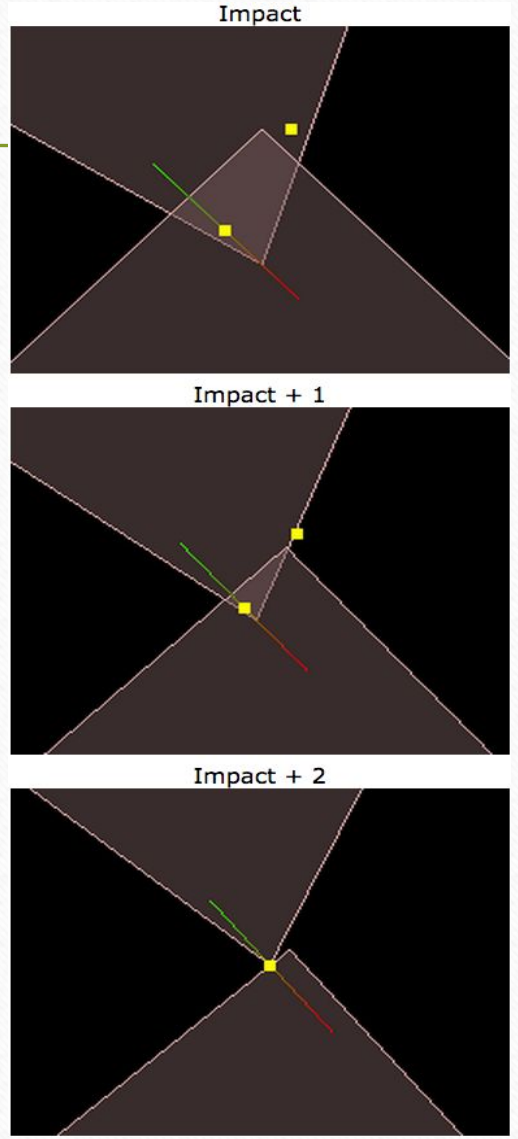
- Code is similar to previous TODO
- Use **output.fraction** to calculate the returning value
- This to-do might need some (fun) math work from you!

Collision Detection

- Box2D reacts to overlapping shapes, pushing them away.
- It does this in several steps:
  - `BeginContact`
  - `PreSolve`
  - `PostSolve`
  - `EndContact`

Called multiple times: one time per solver step

- http://www.iforce2d.net/b2dtut/collision-anatomy



TODO 3

Make module physics inherit from **b2ContactListener** then override the function **void BeginContact**(b2Contact* contact)

- You need to make **ModulePhysics** class a contact listener:
  - world->SetContactListener(b2ContactListener*)
- Just LOG “collision!” and check if it works

TODO 4

Add a pointer to **PhysBody** as *UserData* to the b2body

- Bodies have a void pointer to user data for our convenience (SetUserData)
- Do it in all methods that create a body
- We can use it to store any pointer!
- Use it to store the address of our custom **PhysBody** class
- Now inside collision callback (ModulePhysics::BeginContact), obtain pointers to both **PhysBody*** bodies.

TODO 5

Create a **OnCollision** method (in Module.h) that receives both **PhysBodies**

- We will specialize this method to modules interested in it
- It should receive both **PhysBodies** involved in the collision
- Avoid includes if possible: use forward declarations

TODO 6

Add a pointer to a module that might want to listen to a collision from this body

- **PhysBodies** should know which module (if any) want to listen to collisions
- Make sure the pointer is initialized NULL when the class is constructed

TODO 7 & 8

Now just define collision callback for the circle and play bonus_fx audio

- Make sure to add **ModuleSceneIntro** class as listener to all circles
- Define the collision callback function and just play bonus_fx sound
  - App→audio→PlayFx(bonus_fx);

Homework

- Find out about sensors in Box2D
- Sensors won't use callbacks for collision detection (no "kidnap" execution)
- You need to iterate all contacts and keep those that collide **IsTouching()**
- Then call collision callback as you'd do with any other body.
- I suggest you use **PreUpdate** method for this, just after stepping the world.