Physics II

CITM

Bullet Physics – Collision Detection

Recap

- On our first class, we created **Primitives**, and rendered them.
- On our second class, we created **RigidBodies**, and "debug rendered".

- Today:
  - We'll fix the memory leaks.
  - We'll link RigidBodies and Primitives.
  - We'll have modules react to collisions.

A few changes

The handout is looking a little different to what it did before:

- We're calculating the "**local Inertia**" upon creating PhysBody
- Adding inertia, will allow bodies to turn and roll

```
btVector3 localInertia(0, 0, 0);
if(mass != 0.f)
	Shape->calculateLocalInertia(mass, localInertia);
```

A few changes

The handout is looking a little different to what it did before:

- A wild **PhysBody3D** class has appeared!
  - Basic interface between Bullet3D and the rest of the code
  - btRigidBodies now hold a pointer to their "PhysBody3D"
  - Will help us manage memory, and void leaks

```
btRigidBody* body = new btRigidBody();
body->SetUserPointer(this) //this PhysBody3D
```

A few changes

The handout is looking a little different to what it did before:

- A wild **PhysBody3D** class has appeared!
  - The "RigidBody" creation has been moved into PhysBody3D
  - Instead of sending "radius" as a value, we pass a "Sphere" it's going to fit into

```
void CreateBody(float radius, float mass);
```

→

```
void CreateBody(const Sphere& sphere, float mass);
```

TODO 1

Store all “new” created values...

- Memory leaks! Every **new** needs a matching **delete**.
- Let’s store all the “new” created variables, so we can destroy them later!

- Not for the faint of heart… but a good thinking exercice: Collision shapes could be re-used between equal bodies. How could that be done?

TODO 2

…and delete them!

- Now, let's **delete** all the values we've stored!
- **Make sure there's something to delete**! If no "new" was called, calling "delete" will yield an exception → AVOID.
- How do we differentiate between a pointer with or without content?

TODO 3

Create a “new” sphere, and add it to the “primitives” DynArray.

- Before, when we pressed “1” we used to create a “PhysBody” that we could only render in “debug”.
- Now, let’s instead create a **Sphere primitive** which should be always visible. We’ll later link primitives and PhysBodies.

- Add the sphere to the “Primitives” array to store them.

TODO 4

Add a PhysBody to the primitive.

- We want to make sure every primitive has its own physics body linked to it.

TODO 4

Initialize the PhysBody to be a Sphere.

- When we create the Sphere, call "**InitBody**(...)" in order to actually create the Physics body.

TODO 5

On the primitive update, make it match the

Physics body, so the render moves around!

- Right now, every time we create a Primitive, we're also creating a Physics body. However, the body will fall and leave the Primitive behind!
- Let's make the primitive match the Physics body.

TODO 5

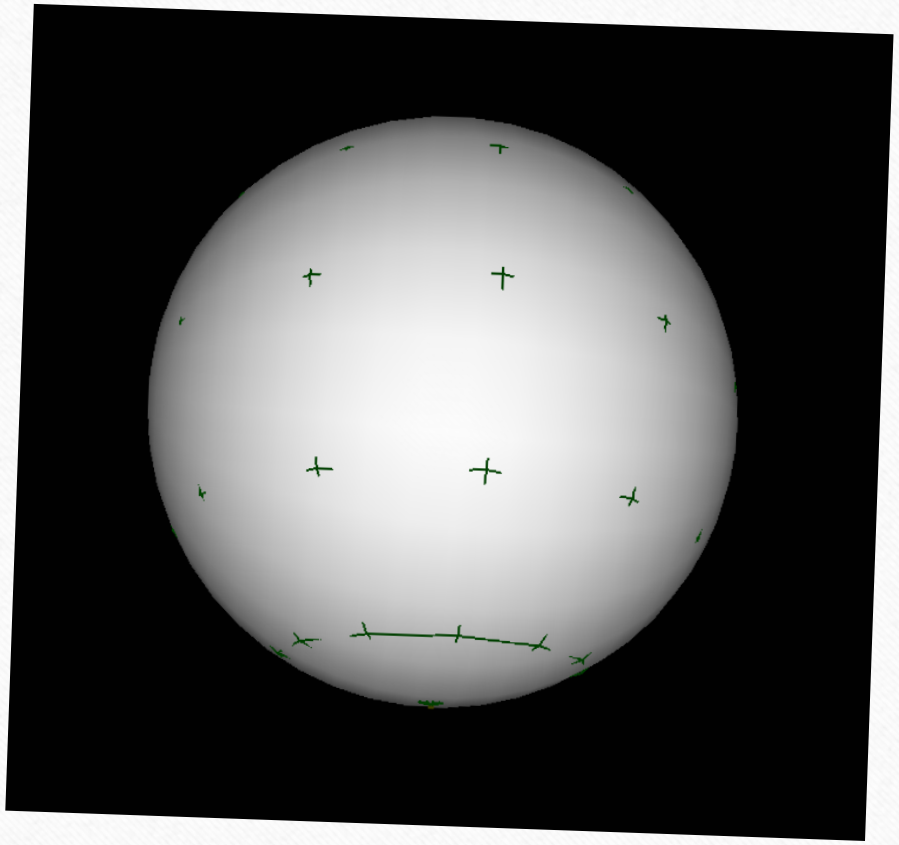
Complete PhysBody3D functions!

- In order to do so, we'll need to finish the PhysBody3D functions!
- Look what functions "btRigidBody" provides to you.

- At the end, add "body->activate();"...

TODO 5

Now, both entities are synchronized!



TODO 6

When we move the primitive, we want to move the Physics body too!

- Inside Primitive::SetPosition(...)
- If we don't update the Physics body position, as soon as we call "Primitive::Update()" the primitive will return to its original position.

- Let's make sure that doesn't happen, and make position change in both the "render" world and the "physics" world.

TODO 7

Create virtual method "On Collision", that receives the two colliding PhysBodies.

- Create a virtual void function, that receives two **PhysBody3D***
- This way, any module can implement its own method to handle collisions.

TODO 8

Detect collisions!

- If a PhysBody3D has any modules added to “CollisionListeners”, we want to call that module collision handling function, sending the colliding bodies.

TODO 9

Create an "OnCollision" method specific for the Scene...

- Create a specific “OnCollision” method for **ModuleSceneIntro**, so it overwrites the base Module function.

TODO 9

…and change the color of the colliding bodies, so we can visualize it working!

- Do something when two bodies collide, so we can check if the code is working.
- The easy route: change the primitive color whenever it collides.

Homework

- We can create spheres. Now create boxes and cylinders with their corresponding physics bodies.

- Extra: When pressing ‘1’, throw spheres in the direction that camera is looking at.