Custom Physics Engine Documentation

By Leonard Andrew Spencer

3/3/2018

Contents

[Introduction 3](#_Toc507849222)

[Dependencies 4](#_Toc507849223)

[Physics Engine 5](#_Toc507849224)

[PhysicsScene 5](#_Toc507849225)

[IFixedUpdater 5](#_Toc507849226)

[ICollisionObserver 5](#_Toc507849227)

[PhysicsObject 5](#_Toc507849228)

[Plane 6](#_Toc507849229)

[RigidBody 6](#_Toc507849230)

[Sphere 6](#_Toc507849231)

[Box 6](#_Toc507849232)

[Joint 6](#_Toc507849233)

[Spring 7](#_Toc507849234)

[Collision 7](#_Toc507849235)

[SoftBody 7](#_Toc507849236)

[Rope 7](#_Toc507849237)

[Post Mortem 8](#_Toc507849238)

# Introduction

\*\*\*\*\*\*TO DO\*\*\*\*\*\*

# Dependencies

This library uses OpenGL Mathematics for vector and matrix mathematics operations.

It also uses the AIE Bootstrap library to render physics objects.

# Physics Engine

Classes in the PhysicsEngine library are in the physics namespace.

## PhysicsScene

class PhysicsScene

A physics scene contains physics objects which interact with each other and manages the simulation.

During each fixed duration physics step the following actions are taken:

* FixedUpdater objects observing the scene have their fixedUpdate method called
* earlyUpdate is called on all physics objects. This is used to apply forces from springs.
* fixedUpdate is called on all physics objects. This is where rigidbodies move.
* Collisions are checked between physics objects. On a hit, the objects inform observers about it and resolve the collision

After all physics steps in an update, the physics objects are drawn. This interpolates their current and previous position by the amount of time left in the update. This means the object is consistently drawn one timestep before the current time, preventing temporal aliasing.

PhysicsObjects and FixedUpdaters are stored as shared pointers, since the scene will probably share responsibility for them with some other object. While FixedUpdaters can be removed at any time, PhysicsObjects should not be removed during a collision. Instead, the object’s kill() method should be called. This will flag it for removal at the end of a fixed update. Also, Clear should never be called within fixedUpdate or OnCollision functions.

## IFixedUpdater

class IFixedUpdater

This purely abstract class is an interface to allow derived classes to be added to a PhysicsScene and have their fixedUpdate function called every physics timestep. These are held by the PhysicsScene as shared pointers.

## ICollisionObserver

class ICollisionObserver

This purely abstract class is an interface allowing derived classes to subscribe to PhysicsObjects and be informed about collisions involving the object. They are held by the PhysicsObject as weak pointers, to avoid cyclical references causing memory leaks.

## PhysicsObject

class PhysicsObject: public std::enable\_shared\_from\_this<PhysicsObject>

This is an abstract base class for all objects in a physics scene. It contains data used by several derived classes, such as elasticity and friction, and provides a common interface for updating, testing and resolving collisions, and drawing objects.

A physics object can have CollisionObserver objects subscribed to it. When BroadcastCollision is called, these objects will be sent the collision object, allowing them to react to the collision. They are stored as weak pointers by the PhysicsObject, to avoid reference cycles between the observer and object.

To remove PhysicsObjects from a scene safely, and let anything else with references to it know it has been removed, the kill and isAlive methods can be used. Calling kill will set the object to dead, and after the next physics update it will be removed from the physics scene. Similarly, if it has been remove from a scene using removeActor, it will be set as dead. Before a killed object is returned to a physics scene, resetAlive must be called, or it will be removed again the next frame.

## Plane

class Plane : public PhysicsObject

This PhysicsObject represents a plane. The plane is one sided (anything behind the plane is considered to be colliding with it) and infinitely long. Planes are always considered static objects.

## RigidBody

class RigidBody : public PhysicsObject

A rigidbody represents a solid, non-deformable object. This class implements shared behaviours which do not depend on the object’s shape.

Rigidbodies can be dynamic, kinematic, or static. Dynamic bodies move, are affected by drag and gravity, and can have forces applied to them both through collisions and with the applyForce or applyImpulse methods. Kinematic bodies move and rotate, but aren’t affected by any forces. Static bodies do not move, except by setting their position directly.

A body is made kinematic by setting its mass as 0 or infinity, and is made static with the setStatic method. Collisions aren’t tested between two static bodies.

Rigidbodies store both their current and past positions and local axes. This is so they can be rendered at a position and orientation interpolated between these values, ensuring smooth movement regardless of update time.

## Sphere

class Sphere : public RigidBody

This is a spherical rigidbody. Collision detection, drawing spheres, and calculating moment of inertia are implemented by this class

## Box

class Box : public RigidBody

This is an oriented box rigidbody. Collision detection, drawing spheres, and calculating moment of inertia are implemented by this class.

Box collision is tested using the separating axis theorem, checking if there’s an axis on which both bodies can be projected without overlapping. If colliding with a sphere, the collision point is along the collision normal from the circle’s centre, half the collision depth from its edge. If colliding with a box, the area where the two colliding edges overlap is found, and the centre of that area is the collision point.

## Joint

class Joint : public PhysicsObject

This is a base class for joints between two rigidbodies. Joints hold references to the bodies at each end, and store the anchor point they attach to in that end’s body’s local coordinates.

## Spring

class Spring : public Joint

This is a spring between two bodies. The spring has a set resting length, tightness, and damping. It applies a force to the attached bodies to bring the distance between their anchor points to the spring’s resting length.

When the spring notices an attached object has been killed, it removes the object. So, if it is returned to the scene, the object will have to be reattached.

## Collision

struct Collision

The Collision struct is used to pass around data about a collision. The checkCollision functions return Collision objects, and they are passed as arguments to the resolveCollision, broadcastCollision and onCollision functions.

The Collision object stores the two objects involved, whether a collision occurred, the collision normal and depth of interpenetration, and the contact point of the collision.

A Collision struct stores the involved objects as raw pointers, since the Collision object only exists long enough for the scene to resolve that collision.

## SoftBody

class SoftBody

A SoftBody is a collection of rigidbodies connected with springs. The SoftBody object holds references to these PhysicsObjects, and provides methods to add them all to a PhysicsScene, kill all its objects, access its rigidbodies, and change the tightness and damping of its springs.

Destroying the SoftBody object doesn’t destroy its contained PhysicsObjects, unless it’s the last remaining reference to them. If there’s no need to modify the SoftBody after adding it to the scene, it can just be destroyed, to save memory space.

## Rope

class Rope : public SoftBody

A rope is a one-dimensional soft body. It does not have shear or bend springs, and provides a getSegments method to more easily access its particles.

# Post Mortem

There are several improvements that could be made to this physics library. First, there are several features that could be implemented. Composite rigidbodies, made from several primitive shapes, would allow more shapes to be created. Also, rigidbodies could have a centre of mass set, rather than being at the centre of the body. Additional joints could be created like hinges or sliders.

Currently high enough tightness and damping on springs can cause numerical instability. Setting limits on the force applied by the spring could avoid this problem.

Softbodies could be created with shapes other than rectangles. For example, a 2D boolean array could be passed in the constructor, indicating which positions should have particles.

It could be useful to let PhysicsObjects have one particular CollisionObserver be set as its “owner”, allowing it to be easily accessed from the PhysicsObject.

Rather than having the PhysicsScene just store PhysicsObjects in a single container, it could hold different types of objects in different containers. This could make collision detection more efficient, as there’s no need to check joints or pairs of planes against each other. Also, there wouldn’t need to be an earlyUpdate method in PhysicsObjects, since joints could just have their update called before other objects.

Rather than having the physics objects directly implement the collision detection, it could be extracted out into a geometry module, which could check for overlaps between objects, line intersection, and so on. This would allow