Farseer Physice Engine Manual

## Introduction

The Farseer Physics Engine is an easy to use 2D physics engine designed specifically for Microsoft's XNA platform. The Farseer Physics Engine focuses on simplicity, useful features, and enabling the creation of fun, dynamic games.

## Overview

Getting right to the nut, the Farseer Physics Engine is designed to control the position and rotation of game entities over time.

In the real world, things move and spin due to applied forces and torques. In Farseer, the same is true. Objects called "Bodies" represent the real world things.  As forces and torques are applied, the bodies react according to the laws of 2D physics. The position and rotation of these bodies are then used to update game entities.

In the very simplest outline it works like this:

1. Create "Body" object
2. Add Body object to simulator.
3. Begin Game Loop
   1. Apply forces and torques to Body.
   2. Update the simulator
   3. Set game entity(sprite or whatever) position and rotation equal to Body position and rotation.
   4. Draw game entity
4. End Game Loop

Bodies, by design, have no geometry in the 2D world and therefore have no concept of collisions.

For collision, Farseer has the "Geometry" object. Geometry objects are represented as 2D polygons and can be either concave or convex. They are defined by a set of vertices.  One or more Geometries are attached to a Body in order to give the Body geometrical awareness. This allows the Body to participate in collisions with other Bodies(actually other Geometries attached to other Bodies, but you get the picture.)

## Dynamics

### Body

Bodies are the core physics objects in Farseer. Forces, torques, and impulses are applied to bodies and the bodies react by moving realistically. Bodies do not contain any form of collision geometry by themselves. You will need to create a Geom for collision detection. To create a body you usually use BodyFactory.

You create a body like this:

int mass = 1;

float width = 128;

float height = 128;

Body rectangleBody = new Body();

rectangleBody.Mass = mass;

rectangleBody.MomentOfInertia = mass \* (width \* width + height \* height) / 12;

As you can see, calculating MOI (Moment Of Inertia) for a rectangle is done like this:

D:\Projects\TFS\FarseerPhysics\Documentation\ManualImages\MOIRectangle.png

Instead of remembering MOI for all shapes, Farseer can calculate the MOI for you. All you have to do is use the BodyFactory class described below.

### Body Factory

You create a body with the factory like this:  
Body rect = BodyFactory.Instance.CreateRectangleBody(PhysicsSimulator, 128, 128, 1);  
  
This body has a size of 128 width, 128 height and a mass of 1.

The MOI (Moment Of Inertia) is calculated for you.

## Joints

### Revolute Joint

### Angle Joint

### Joint Factory

### Linear Spring

### Angle Spring

## Collision

CollidesWith - Default is CollisionCategories.All - Defines the categories that the geom will collide with  
CollisionCategories - Default is CollisionCategories.All - Defines the categories that the geom is a member of.  
CollisionGroup - Default is 0 - Defines the collision group that the geom is a member of.  
  
CollisionGroup is the easiest to use. All geoms in the same group does not collide. GeomA of group 10 and GeomB of group 10 will not collide.  
CollidesWith and CollisionCategories are the more advanced way of making collision groups. But works in kind of the same way.  
  
The enum CollisionCategories is defined with the Flags attribute (info [here](http://msdn.microsoft.com/en-us/library/system.flagsattribute.aspx)) that enables you to use bitwise addition or substraction of categories.   
Example (taken from demo 5 in the samples):  
  
\_agent = new Agent(ScreenManager.ScreenCenter);  
\_agent.CollisionCategory = CollisionCategories.Cat5;  
\_agent.CollidesWith = CollisionCategories.All & ~CollisionCategories.Cat4;  
//collide with all but Cat4 (black)

\_blackCircles3 = new Circles(startPosition, endPosition, 10, 9, new Color(0, 0, 0, 200), Color.Black);  
\_blackCircles3.CollisionCategories = CollisionCategories.Cat4;  
\_blackCircles3.CollidesWith = CollisionCategories.All & ~CollisionCategories.Cat5;  
//Collide with all but Cat5

### Geometry

### Grid

### Vertices

### Collision Groups

### Collision Layers

### Collision Events

## Example Code Snippets