Awesome Ball Source Code

Version 1.11, April 2012, iOS 5.1, Xcode 4.3.2

Summary

Awesome Ball is a 3-D bouncing ball simulator for iOS devices. It uses the Open Dynamic Engine physics engine to model the physical environment and OpenGL to display the graphics. Through the settings screen, the user is able to customize the ball being bounced as well as the box it is bouncing in. Awesome Ball includes several "Easter eggs" (unadvertised features) that can be unlocked with specific multi-touch gestures while bouncing the ball.

Source Code Groups

The Awesome Ball code can be divided logically into several parts based on the function of the Objective-C classes. These divisions are represented in the Xcode project as "groups."

- 3D Models
  - Contains a header file with the OpenGL and physics specification for the ball.
- ODE
  - Contains the Open Dynamics Engine physics library obtained from www.ode.org.
  - Awesome Ball uses this library under the BSD license.
- OpenGL
  - Contains the 3-D visualization code for the bouncing ball environment as well as the tie-ins to the modeled physics environment.
  - This is the front-end of the app that the user sees.
- Classes
  - These are the main iOS classes that run the Awesome Ball app, including the AppDelegate and the root ViewController.
  - The ball types and saved user settings are also found in this group.
- Game Types
  - Awesome Ball is written such that multiple "games" may be created within the "GameType" class hierarchy.
  - Currently, only the "BasicGame" game is written, which just lets the user bounce a ball around without any goal or structure.
  - The "FreePlay" game type extends BasicGame by adding the Easter eggs.
- Audio
  - Contains the SoundEffect class, which handles playing the sounds of the ball bouncing against the walls of the box.
- SettingsView
  - The SettingsView, as the user sees it, is the flip-side of the app, where the ball and box may be customized.

    - Several classes related to these functions are included here.
- Other Sources
    - Contains non-class source files which help with launching the app as well as some helper functions for the physics environment calculations and the OpenGL views.
- Resources
    - Contains the InterfaceBuilder .xib files for the RootView, SettingsView, and CustomBallView.
    - Contains an “Images” subgroup with all of the images for the balls, walls, icons, and graphics for the SettingsView and splash screen.
    - Contains a “Sounds” subgroup that contains the sound files (.caf format) for the different balls.

Class Descriptions

OpenGL

- GlView
    - GLView is the UIView that displays the OpenGL elements of the app (the ball, wall, etc). When refreshing its view, it uses the GameType object specified to do any extra actions.
- GlViewController
    - GLViewController owns the OpenGL view. It handles the loading and unloading of that view.
- GLBall
    - GLBall simply implements an OpenGL sphere and handles the settings for the appearance of the sphere.
- GLWall
    - GLWall simply implements an OpenGL rectangle and handles the settings for the appearance of the rectangle.
- GLWalls
    - GLWalls is a box consisting of 5 GLWall objects. Since the settings for these walls (position, appearance, etc.) are tied together so tightly, they are grouped in this class.
- TextureLoader
    - TextureLoader handles the textures used in our two EAGLContexts. Textures are cached and can be released all at once or individually. The textures include those used on spheres and walls (and anything else).
- TextureLoaderMapEntry
    - This class exists because an NSMutableDictionary always makes a copy of a key when inserting an entry into a map. This is not always desirable. In this case, we need a map from an EAGLContext to another NSMutableDictionary. Since the EAGLContext class doesn't want to have its instances copied, we need another way. We'll use an NSMutableSet instead of a map.

Game Types

- GameType.h

    - GameType is an interface for all game types. All game types (e.g. FreePlay) conform to this interface.
- GameState
    - GameState holds information about the current state of a game. This could be which level the user is on, his score, etc. This information is to be saved to disk when the app is exited so the state can be restored.
- BasicGame
    - BasicGame is the most basic game type: one ball in a box BasicGame includes handling for the accelerometer, pinch zooming, and swipe spinning
- FreePlay
    - FreePlay extends BasicGame by adding easter eggs
- EasterEggs.h
    - This header file assigns an ID number to each EasterEgg

Classes

- AwesomeBallAppDelegate
    - AwesomeBallAppDelegate loads the Awesome Ball app
- RootViewController
    - RootViewController is the main View Controller. It owns the GLViewController (the OpenGL ball view) and the SettingsViewController, where all the settings are kept.
- BallTypes
    - BallTypes is a wrapper around an NSArray that holds all of the ball types supported. Characteristics of the different balls are stored here. For now, a single custom ball is also supported. This is a singleton object. Many classes will access this object.
- UserDefaults
    - UserDefaults holds settings that are shared between the Awesome Ball classes and/or that are meant to persist across sessions. The calls to [NSUserDefaults standardUserDefaults] are to the standard user settings class provided by Apple. The settings are stored in a file in the App's Documents directory and the file persists across updates to the App.
- ImagePickerController
    - A helper class to load up the Apple-provided UIImagePicker, allowing the user to select the image source location (camera, photo library, etc.)
- ImageManipulation
    - Contains helper methods for cropping, resizing, and duplicating images for use on walls and balls

Audio

- SoundEffect
    - An Apple-provided class, SoundEffect is a simple Objective-C wrapper around Audio Services functions that allow the loading and playing of sound files.

SettingsView

- SettingsViewController

    - SettingsViewController is a beast of a class that handles all of the settings available to the user including custom images, ball type, custom ball settings. Most of the settings are stored using the UserDefaults class, but are set here.
- BallPickerController
    - BallPickerController is a simple class to handle the UIPickerView for the different ball types. It provides the picker with the balls to choose from and handles actions, passing them back to the delegate.
- BallImagePickerController
    - BallImagePickerController is a simple class to handle the UIPickerView for the different ball image types used for wrapping on the custom ball. It provides the picker with the balls to choose from and handles actions, passing them back to the delegate.
- CustomBallViewController
    - CustomBallViewController controls the view presented to the user when customizing a ball. The user can choose a built-in ball image or their own image and can customize some of the ball parameters.
- GLPreView
    - GLPreView is a basic GLView meant for showing off the different ball types or environment types that a user can choose from in a small OpenGL window. GLPreView has its own EAGLContext associated with it, so it has a set of textures separate from the main GLView. This view is meant to be completely unloaded when it is not in use in order to save memory.
- ShakingView
    - Extends UIView to be able to receive shaking events: UIEventSubtypeMotionShake. This is currently only used by the SettingsViewController to trigger an Easter egg that enables high-resolution OpenGL graphics on a retina display device.

Selected Detailed Descriptions

OpenGL/ODE Physics

The most complex code in Awesome Ball is found in the BasicGame class. This class coordinates the drawing of the OpenGL ball and the physics modeling. These two worlds (graphics and physics) are necessarily tied together. The physics environment is all modeled with the ODE library routines, and they are visualized by drawing the OpenGL walls and ball where the physics ODE library calculates them to be at each moment in time.

Physics:
Six infinite planes are created in the physics world to model the box in which the ball bounces. A single sphere is also created in the physics world. The sizes of the walls are controlled by constants at the top of the BasicGame.m file. The size of the ball is controlled by the type of ball selected by the user.

At regular time intervals, a timer callback updates the physics world with the current state of the environment and the forces to be applied to the ball at each time step. These forces come from several sources:

1. Gravity, based on the device's accelerometer readings. A low-pass filter on the accelerometer values is used to isolate the gravity component of the readings. This force is applied to the center of mass of the ball.
2. User-applied force, the opposite of the force applied by the user by shaking the device, also based on the device's accelerometer readings. A high-pass filter on the accelerometer values is used to subtract out the gravity component of the readings.
3. Swiping force, exhibited when the user swipes his/her finger on the screen. This force is applied as a torque on the surface of the ball and is controlled by the length and speed of the swipe.
4. Aerodynamic drag force, which simply models the force against the ball's center of mass that would occur when the ball is moving through air instead of a vacuum. This allows the beach ball and other light balls to feel more realistic.
5. Bouncing force, which occurs when the ball collides with one or more of the box's walls. This can create a force on the center of mass of the ball or a torque on the surface of the ball.

The physics modeling also has a callback routine activated when two physical bodies (a wall and the ball in this case) approach each other. When a collision is detected, this routine handles the consequences by setting the type of interaction between the two bodies. Since we only have one ball moving, the collision is always between a ball and a wall and the bounciness of the collision should always be the same. This "callback" method can be modified to allow bounces or not and to modify the coefficient of friction between the ball and wall, altering how much torque the ball receives, etc.

Graphics:

OpenGL is used to draw a visualization of the five visible walls (the sixth is the iOS device's screen) and the ball. Textures are applied to the walls and ball to create the different wall and ball types.

The user can control the view on the world in two ways:

1. By default, the view ("camera") follows the ball, with a slight delay to smooth the view. The user can choose (in the SettingsView) to fix the camera in a certain location so the view is fixed above the box.
2. The user can zoom the view (or camera) in and out by using a two-finger pinch gesture.

Sounds:

When the ball collides with a wall, a bounce sound effect is played. The volume of the sound effect is scaled to correspond to the force/speed of the bounce.