Tacky Golf
Senior Project Write-Up
By Robert Crosby

Abstract

This project implements a simple miniature golf game in 3d for the iPhone. Using a modular approach the game engine was written in several modules to handle its various functions and for cross platform potability. Four major modules are described in detail about their design choices and how they work. Ultimately the game and its engine are intended to be sold on the Apple App store and to be used for future games and applications on the iPhone.

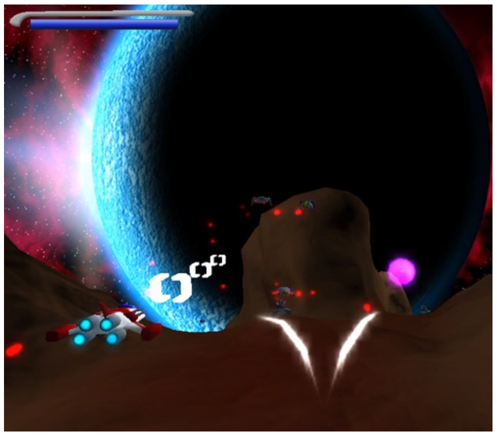
Introduction

Mobile platforms such as ios and android have recently presented a number of exciting opportunities for media rich applications. The graphics hardware running these platforms has evolved to the point where features once available only to desktop computers is now capable of running on a cell phone. OpenGL ES 2.0 is the latest open standard at this time for full function 2d and 3d graphics on embedded systems. On ios many of the 2d drawing libraries use OpenGL directly. Most games on the iPhone are written to use these 2d libraries, but for 3d the game would need to be written to work directly with OpenGL. There are a number of 3d engines available for mobile platforms, but are usually available at a cost and may have features that a simple 3d game may not need. Ultimately the best solution is to build a simple engine tailored for the game.

Previous Work

In the beginning, the goal was to create a simple 3d game on the iPhone. The game to be created was a simple miniature golf game called “Tacky Golf” that is similar to the mini putt games available on addictinggames.com, but with a 3d environment and controls that will work with the touch screen on the iPhone. The game was first built upon a simple asteroids game, written in c++ from an example in a book by Philip Rideout, which used modular design to support older versions of OpenGl ES. The backwards compatibility was dropped since most devices at this time are capable of running OpenGL ES 2.0. But the modular design was still retained for abstracting the core game logic from system specific code interfacing with the iPhone. This modular design was improved upon by several iterations of rewriting to meet the needs of the golf game.

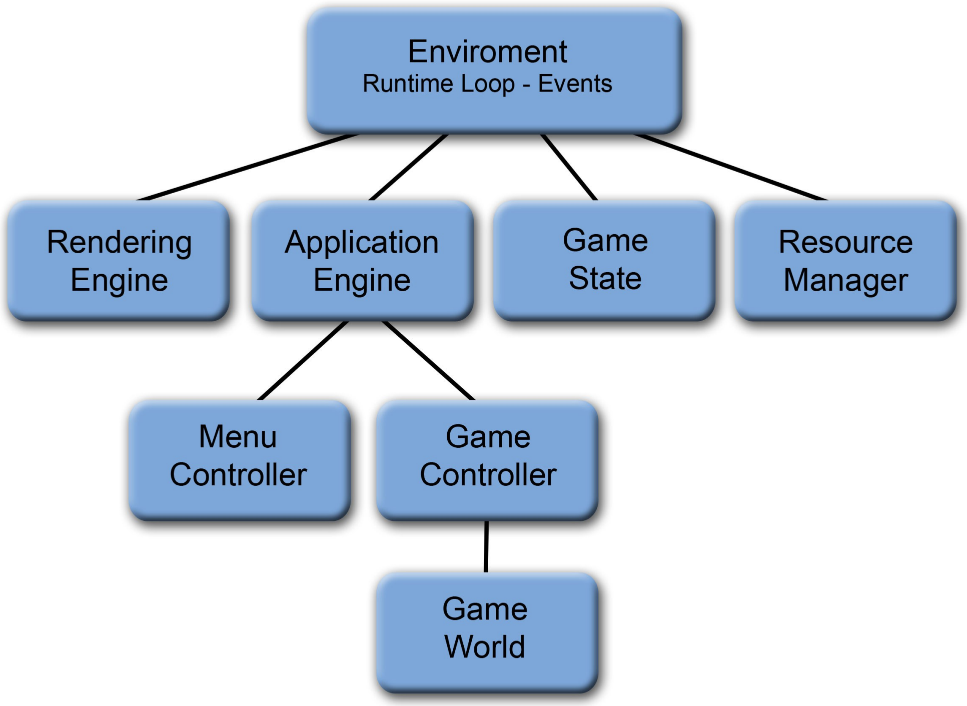
The work on the golf game was later applied to a different game called *Star Republic*, a space shooter similar to *Star Fox*. *Star Republic* was written to use the Simple DirectMedia Layer (SDL) platform on the desktop. SDL provided a similar interface to that of the iPhone, where both provided a main runtime loop, an event system for user input, and an OpenGL context. Not much was needed to convert the system specific modules of the golf game to get an engine working on SDL. The modular nature of the engine proved very useful for the group development of *Star Republic*. This allowed group members the ability to work on individual modules of the engine with little or no impact to other modules. The lessons learned in development of *Star Republic* were applied to the current engine for *Tacky Golf*.


A screen shot from Star Republic

Algorithm

The modular approach to the design of the game involved four main modules, each with specific functions. These modules, as well as other parts of the game, are defined as abstract classes and structs in an Interfaces file. This Interfaces file provides a general blueprint, from which the the rest of the program was built on top. Each module is an individual object that extends an abstract class, defined in the Interfaces file.

The environment where the four main modules reside, is provided by a standard OpenGl template available on Xcode. The template sets up an OpenGl context and provides a run time loop with events. At the start of the game the environment instantiates the four modules and then it provides the Application Engine module with events throughout the duration of the program. All four objects remain for the life of the program.



The general layout of the major parent child relationships of the main modules and objects used in Tacky Golf

Rendering Engine

The first of these modules is the “Rendering Engine.” This module is responsible for all drawing and management of graphical assets. All OpenGL calls in the game are called within the “Rendering Engine.” By keeping all the drawing functionality centralized to this module, it makes it easy to adapt the entire game to different drawing systems other than OpenGL. When the engine for *Tacky Golf* was adapted to run on SDL for *Star Republic*, the Rendering Engine was adapted to run the desktop version of OpenGL from the embedded version of OpenGL ES. The reverse was done, once again from *Star Republic* back to *Tacky Golf*. This adaptive nature can allow the game to be played on a number of different drawing systems offered by different platforms.

Since all drawing operations are performed by the Rendering Engine there needs to be a common interface for the rest of the modules with which to communicate. This is done by a suit of objects defined in the main Interfaces file that can be extended by the Rendering Engine or other modules. The first of these objects is the Camera. The Camera object is used to define view information used by the Rendering Engine. In *Tacky Golf* the Rendering Engine takes only one Camera at a time, but the Camera object can be modified by other modules to update view information for the Rendering Engine.

Another major object used to interface with the Rendering Engine is the Object class. This class defines items visible in the game and can be extended for use in other modules. To the Rendering Engine the Object class is meant to be a container for Mesh objects. The Mesh class defines a single triangular mesh that can be rendered by the Rendering Engine. A large amount of information is contained within the Mesh class such as textures, shading, lighting, location, and rotation of a mesh. The Mesh class is not intended to be extended as much as the Object class by other modules since many objects in the game are composed of multiple meshes that can have different mesh settings.

The TextFeild extension of an Object offers a good example of the relationship between Objects and Meshes. The TextFeild class contains a Mesh object that defines a simple plane with a texture atlas, with letters and numbers for displaying text. Before this class can be displayed it needs to be added to the Rendering Engine, so that it can load the necessary assets for displaying the mesh. After being added, the TextFeild can duplicate the single Mesh object so that each duplicate mesh will display a single plane with a letter or number together forming text for the game. When the TextFeild is no longer needed, it can be removed from the Rendering Engine allowing resources in the engine to be freed up. Both the Object and Mesh classes provide everything needed to display meshes to the screen, which at this time is the only way to display anything on the screen.

In many other engines, each object has the ability to render itself with its own rendering method, but in *Tacky Golf* this is not the case since objects outside of the Rendering Engine module cannot contain any drawing code. Instead Objects are passed to the Rendering Engine as lists in each iterative render cycle. A major benefit to this is the ability to order Objects in the list. This allows for more drawing control for transparency.

Resource Manager

The second major module is the Resource Manager, which is responsible for loading and saving files to the file system of the iPhone. The main purpose of this module is to abstract system specific calls for file management. Since several file operations cannot be done without some Objective C code, this module acts as a boiler plate to keep Objective C out of the other modules and to centralize all file system specific calls. In *Star Republic,* this file was completely rewritten for managing files on a desktop system and changed again for *Tacky Golf*.

Application Engine

The third and largest of the modules is the Application Engine. This module is composed of a number of objects that define the logic and control of the game. Everything in the Application Engine module is written to avoid any system specific code, which should be handled by other modules. At the top of this module is the main Application Engine object, which represents the entire module as a single object that interacts with other modules and the runtime environment of the iPhone. This main object contains several Controller objects that it passes user and system events to. Each Controller object is capable of controlling the entire game, and only one can be active at any time while the game is running. The main Application

Engine object provides the ability to switch between Controller objects, where each controller can control a major part of the program. In *Tacky Golf* one controller is used for the main menu and a second controller is used while in game.

The Controller for the main menu allows for a basic user interface with graphics and buttons. This object receives events from the main Application Engine object and then can either act on them or pass information to it's child objects such as buttons. Button objects can be defined in the Controller with callbacks to methods of other objects in the Application Engine module. When a button determines it has been pressed, it can execute the callback to perform actions. The Controller is responsible for creating lists of Objects to pass to the Rendering Engine module for drawing. The controller generates these lists by adding its own drawable objects and by recursively calling its children for lists of their drawable objects.

When moving from the menu to the game, the main Application Engine object switches the active Controller to one that runs the game part of the application. The Game Controller extends the previous Controller to add more user controls for controlling the game, but to keep the menu and button controls for use in game menus. This object's primary function is to control the game, but the concurrent game state is managed by another child object called the Game World. The Game Controller works closely with the Game World by querying game information and processing user events to tell the Game World what should be happening in the game. Using information from the Game World, the Game Controller can determine if the user is either interacting with either the ball or the environment and can tell the Game World what to do.

The Game Controller provides the user with two ways to hit the ball. The first way is a quick swipe under a certain period of time from the location of the ball. The vector from the start and end touch locations is passed to the Game World as a ball hit. The second way to hit the ball is by touching the ball and dragging in the opposite direction of the intended hit direction. After a certain period of time a potential path is displayed and when the user lifts up the vector is passed to the Game World for a ball hit. Originally the dragging method for hitting the ball was only available on the game. During play testing some users had a hard time figuring out the touch and hold mechanic and where quickly swiping the screen before they could see the path in the opposite direction. The quick swipe method is not as accurate as the dragging method but it does make it easier for a user to discover the path when they hold down for too long. When the ball is hit the Game Controller sends the Game World a vector for the direction of the ball. The Game World would then determine the physics of the ball for each frame of the game and move it until the physics determines that the ball has stopped. As the ball is in motion the Game Controller locks out any user interaction with the ball.

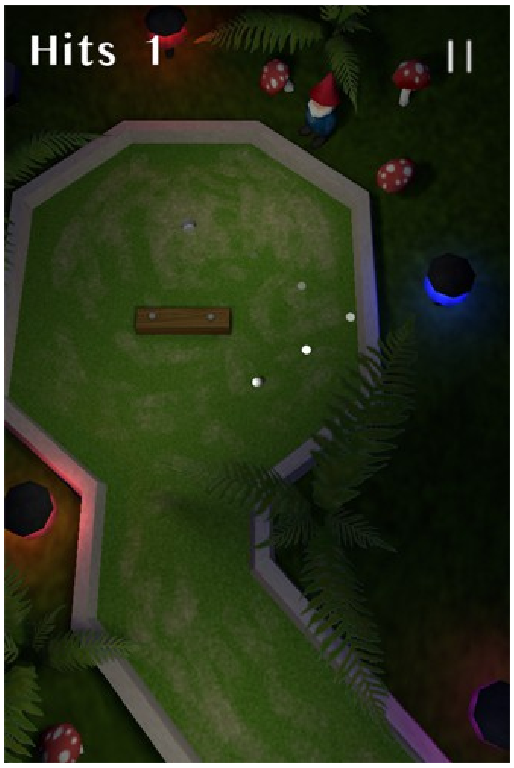
The Game Controller also provides some simple controls for changing the view of the course. By touching and dragging anywhere other than the ball the Game Controller moves the view in that direction. A two finger pinch can also be used to zoom in and out the view as well. These controls are usually intuitive to most users since many things on the iPhone are controlled by dragging and pinching such as web browsing or viewing images.

Game State

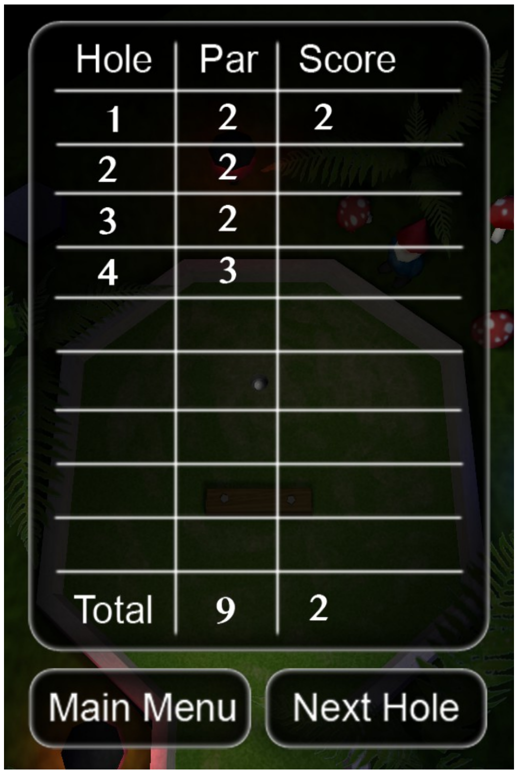
Game State is the last major module used by the game. This module keeps track of the state information of the entire game such as the current course being played and the number of ball hits to complete previous courses. Unlike the Game World class in the Application Engine this module is not intended to handle the details of the active game but is used to set up the game and to recored results. The Application Engine queries the Game State when ever it needs to know what course to play next and for displaying statistics of previously played courses. This module is still under development and several features still need to be implemented such as recording information to a file when the application is closed.

Results

In the end a functional miniature golf game was created that runs smoothly on the iPhone. Four courses were created to demonstrate that the game can switch between courses and record player progress as they played the game. A simple mesh viewer was also created by rewriting the Application Engine module to demonstrate the lighting capabilities of the engine. The mesh viewer used per pixel lighting to display normal mapped mesh that got different frame rates on different devices. The iPhone 4 displayed the mesh at 30 frames per second but an older third generation iPod touch displayed it at about 14 frames per second. This was not an issue for the golf game since most meshes had lighting pre-baked and did not need to perform much lighting.



A screen shot of the *Tacky Golf* game screen.



Hole	Par	Score
1	2	2
2	2	
3	2	
4	3	
Total	9	2

A screen shot of the scores view after finishing a game.

A screen shot of a model viewer using the same engine as Tacky Golf demonstrating per pixel lighting and normal mapping

Future Work

A number of features and items need to be added before this game can be considered done and ready for the app store. First there needs to be more courses with engaging features. There should be at least nine courses for a short game of golf. A major feature this game is lacking is sound. The requirements would be modest since there would be a few sounds needed for sound effects such as a hit noise when the ball is hit. Sound was implemented in *Star Republic* as another module and the same would be done for *Tacky Golf*. Expanding on the physics for courses with slopes and hazards would also be a must. A previous version of the game did support slopes but was extremely buggy. Any problems before implementing the advanced physics were amplified along with many other problems. Last the Game State module should be updated to save game state when the player leaves the app and returns.

References

Mini Putt from addicting games
http://www.mini-putt.org

iPhone 3D Programming book by Philip Rideout
http://ofps.oreilly.com/titles/9780596804824/