ABSTRACT

For game developers, one of the stickiest tasks has always been porting their creations to new devices. Not only is this process costly and time-consuming, but it also rarely allows for interoperability between platforms. If a developer builds a game for Windows, it would be a major ordeal to get it running on the iPhone or Linux. And even then, the cross-platform experience is lackluster and does not take advantage of the potential for a unified game experience. The emergence of mobile platforms and the worldwide use of such devices has facilitated the need for a more uniform hardware and software interface. Through the use of a modular design as well as a strict adherence to cross-platform support, our project attempts to bridge the gap between computer configurations and create a truly unified game development experience. In the end, we wish to develop a set of tools that abstracts away a predefined hardware or operating system interface and allows game developers to compile and play games on any supported platform.

1. INTRODUCTION

1.1 Motivation

Since the emergence of the video game console, developers have been segregated to individual platforms for game development. Specialized and limited hardware support forced this dichotomy long ago. But now, in an age of super fast mobile devices, advanced cross-platform compilers, and broadband internet connections, the dichotomy is an unnecessary result of accumulated hardware and software specialization.

Specialization comes in many forms: platform specific graphics and audio libraries, differences in native compilers and networking protocols, data serialization incompatibility, native resolution differences, user input and output methods, and many others. These specializations have allowed software to grow and mature, but at the cost of exclusivity to the platform. Programming a game on one platform leads to time consuming and costly rewriting if the developers wish to switch or support multiple hardware configurations. A better solution is needed.

1.2 What is a Game Engine?

Generally, a game engine is a software system that aids in the programming, design, and production of a video game. Such a system could include support for rendering 2/3D objects, producing audio, handling in-game physics and collisions, doing mathematical calculations, scripting, networking, artificial intelligence, memory management, scene management, localization support, and much more [2]. A game engine is typically designed to reduce the overhead of game creation by abstracting these systems into a set of tools that is invariant from one game to another. Thus, code is reused and time is saved in the creation of a new game. However, most game engines specialize to one platform, and a game written using one of these engines still faces an uphill battle to get it running on other platforms.

A cross-platform game engine is simply a game engine with another layer of abstraction. It solves the hardware and software specialization problem and allows games to be run on any supported platform. Typically these game engines are only available to large software companies with teams of experienced game developers. Additionally, they are closed source, require strict adherence to their published API, and have a number of other drawbacks. Thus, it is nearly impossible for small game studios and independent developers to extend their games beyond one or two platforms.

1.3 Benefits of Cross-Platform Games

By creating a development environment that is invariant under hardware changes, game developers can focus more attention to game mechanics and content rather than platform eccentricities and porting efforts. In addition, it brings the possibility of game interoperability between platforms. That is to say, a player on one platform can interact and play with players on separate platforms. Players will also be able to experience the same game across multiple hardware configurations. This, in essence, is the focus of the Tec Game Engine.

Figure 1: Benefits
1.4 Supported Platforms

Tec currently supports Windows, Linux, Mac, and iPhone hardware. This is not to say that these platforms are the only possible platforms we support. Tec is extendable to almost any hardware configuration and operating system. The Tec Engine is written in highly portable C++ and uses a system of conditional native code compilation to support features across all platforms. In addition, the Tec Engine comes packaged with a server program that allows data synchronization across networking enabled platforms.

Figure 2: Supported Platforms

2. RELATED WORK

2.1 Other Game Engines

For the list of available game engines with some details of their functionality, the reader can refer to the [3] Wiki page "List of game engines". As we were researching the related work in this area, we could not find an open-source game engine that runs on the PC and can be ported to mobile platforms. There exists cocos2d [1], which is a framework for building 2D games, demos, and other graphical/interactive applications for iPhone. Even though it is a fully-fledged 2D game engine for the iPhone, it does not address the issue of portability. That is, game developers who usually create games for such platforms as Windows, Mac OS X, and Linux, have to learn the cocos2d game engine API for the iPhone SDK. This is not quite what developers want to do, especially when there are time constraints induced on them. The bulk of available game engines [3], however, are cross-platform across the three convenient OS configurations. Though, all of those are different among each other in some ways: primary programming language, scripting support (built in), the way they achieve portability (which tools used), etc.

C++ is a preferred choice of programming language for such projects since it is extremely fast and at the same time is an object-oriented language unlike C. We will conform to the majority and program our engine in C++. However, we will be implementing scripting functionality with LUA since it is extremely lightweight, game-friendly, and portable to many different platforms. (Some other game engines use Python as a scripting language, but it is not as efficient when dealing with small game logic scripts) In addition to the stated advantages, there exists a variety of open source libraries that can easily bind C++ and LUA codes.

Another feature is that we will be using open-source Simple DirectMedia Layer which is a cross-platform multimedia library designed to provide low level access to audio, keyboard, mouse, joystick, 3D hardware via OpenGL, and 2D video framebuffer. This would enable our engine to deal with different hardware configurations. It should be noted that SDL only works across Windows, Mac OS X, and Linux. Mobile and other support will require a separate implementation.

2.2 Differentiation

Some game engines have tried to solve the cross-platform specialization problem. Unity, Torque, and Unreal each have their own cross-platform engine. They take the concept of hardware abstraction and are able to create, to some extent, hardware independent games.

However, none achieves the main goal of Tec: Interoperability. Despite supporting compilation on multiple platforms, these engines lack any sort of interplay from one system to another. Games programmed using these engines can only interact with games on the same platform. They do not see the value of players from one platform interacting with players from another. Additionally, they do not support a unified game experience where players can start a game on one platform, save the game to a centralized server, and continue playing the same game from another platform. In addition, all of the engines are closed source projects run by major companies. Tec, on the other hand, is a freely available, open-source project with interoperability as one of its main tenants.

3. SYSTEM MODEL

3.1 Modularity

Following a convenient modular approach to programming, the engine is comprised of separate but interoperating
components. To provide an easy and understandable handle of the entire program, it has a singular point of access, GameController, a singleton class containing the main loop and which glues the game managers together for their interoperability. Manager classes (game managers) serve as a layer of platform independence. They manage different modules (GL rendering, Entity system, input, levels, etc.) and can communicate among each other through the master manager which is, in our case, the GameController.

3.2 Libraries

Tec incorporates a number of third party libraries to help extend its functionality.

Libraries:
- **Box2D**: A fully functional 2D physics library that ensures a precise physical simulation of the entities. It also handles all collision detection between objects.
- **OPENAL**: Provides 3D audio support and OGG Vorbis enables playback of ogg audio files.
- **TILED**: Tiled is a cross-platform level editor that we have integrated into the Tec Game Engine. Tec handles saving and loading of these maps as well as some extended functionality.
- **SOIL IMAGES**: A library that is able to load most common image files and render with alpha transparency. (jpg, bmp, png, tga, etc...) SOIL has also been ported to the iPhone.
- **TinyXML**: A lightweight XML file loading and saving library.
- **Zlib**: A highly portable compression and decompression library. Zlib is used to compress map data as well as network packets.
- **ENet**: A networking library that acts as a wrapper around native sockets.
- **LUA**: Users can call Lua functions from C++ and C++ functions can also be called from Lua.

3.3 Cross-Platform Features

To help make programming across platforms a bit easier, we have written a number of classes.

Cross-Platform Features:
- **UniversalWindow**: An abstract class that facilitates the creation of windows across multiple platforms. SDL handles this for Windows, Mac, and Linux through the SDLWindow Class. A custom implementation is written for mobile devices.
- **InputController**: An abstract class that allows polling of user input across platforms. SDL handles this for Windows, Mac, and Linux through the SDLInputController Class. A custom implementation is written for mobile devices.
- **Timer**: Basic timer class to keep track of game events. Our implementation uses native OS time functions to
keep track of time. This is supported differently for each platform.

- **RenderManager**: Tec takes advantage of the portability of OpenGL for its own rendering system that is used across all supported platforms.

### 3.4 Component Interoperability

When laying out the model and incorporating the third party utility libraries, we were much concerned about the interoperability of our components and also how well those libraries integrate into our system. This allowed us to create a robust cross platform game engine with a modular approach. Managers (SceneManager, RenderManager, etc.) which manage separate modules/components abstract out physical properties of a particular platform and allow for easy exchange and use of information among them.

### 3.5 Hardware Interoperability

Because the games created with Tec can be compiled and run on different platforms, and due to the interoperability of engine components controlled by Manager classes, we have envisioned the cloud/server model.

A user playing a game on his platform of choice can save the current game state, which gets stored on the server; when he starts playing the game again, he can restore it to the point when he last saved on the server. Play can be continued from one supported device to another for a unified game experience across devices!

In addition, the cloud/server model allows all interaction between game devices to happen on a centralized game server. Clients authenticate with a username / password system and play can be tracked, verified, and authenticated.

### 4. System Implementation

Tec is comprised of a number of managers and classes to facilitate modular development and interoperability.

#### 4.1 GameController

GameController is a master class which hosts all the secondary controllers and contains the main loop of the engine. It is a singleton class, meaning that there is only one instance of it at all times. It contains references to all the managers. The instance of it can be called anywhere thus providing global access to manager classes and its data.

#### 4.2 Second Level Managers and Controllers

Unlike GameController, each of them (RenderManager, SceneManager, UniversalWindow, LevelManager, etc.) is responsible only for one particular module and do not have direct access to each. All interoperability between these classes goes through the GameController.

Since we aimed to create a cross platform engine, the manager classes which control system modules, such as video, input, audio, were abstracted to top level classes and during the initialization of the system they create proper instances of each manager based on the underlying platform.

#### 4.2.1 Render Manager

The Render Manager sets up the OpenGL context for rendering, updates the current camera viewing the scene, and to actually renders sprites, entities, and fonts. After completely rendering a scene, it swaps buffers and starts again.

#### 4.2.2 Scene Manager

The Scene Manager is the general storage class for the game. All drawable objects are held in the scene manager. There is built-in functionality for accessing, finding, adding to, deleting, and merging these objects. In addition, all cameras are stored here.

#### 4.2.3 Level Manager

The Level Manager provides basic level functionality where a user can create list of levels and have each level support multiple maps. In addition, a user can save current game state in the xml file and load an xml file containing a snapshot of the game, which will restore the game to the saved game state.

#### 4.2.4 Universal Window

Universal Window is an abstract class, so it is implemented in a number of ways. However, each implementa-
When programming, we use modular approach: start from simple and basic details and end up with the complex subsystem. At each step after testing for correctness we ensure that the code can be run on other platforms (Mac, Linux, iPhone). During development, each new feature may introduce new problems and bugs, or reveal some existing ones. So, when implementing new features, the commitment of the code happens only if the integrity of the system is preserved and the exposed problems are fixed. This ensures a gradual and stable build up of the system.

### Development Cycle

<table>
<thead>
<tr>
<th>Idea for Feature</th>
<th>Not Needed</th>
<th>Research Feature</th>
<th>Program on PC</th>
<th>Test</th>
<th>Port to Linux</th>
<th>Port to iPhone</th>
<th>Port to Mac</th>
<th>Feature Complete</th>
<th>Deploy</th>
</tr>
</thead>
</table>

---

5. **MAKING A GAME**

In addition to creating the Tec Game Engine, it was also necessary to test, document, and demo the project for future developers.

#### 5.1 Web Presence

To manage our code as well as some project management tasks, a Google Code page was created for Tec:

http://code.google.com/p/tec-engine

As the project progressed, Tec was opened to the public, and a website was created for pushing out updates and sharing information to Tec users:

http://www.liquidmetalgames.com

On the site, there are a number of videos showing the capabilities of the Tec Game Engine, as well as a number of other updates.

Over 2,000 page views have been logged since the websites launched in March of 2010.

#### 5.2 Game Testing and Research

A large amount of game research was necessary in order to determine which features should be included in the Tec Engine API. In addition, modern 2D games have a number of difficult technical challenges. Recognizing these challenges and implementing a natural, easy to use, and fast development environment are some of the Tec’s biggest goals.

Some games specifically evaluated for features were: Dragon Warrior (1989, NES), Super Mario Brothers (1986, NES), and others.
Braid (2008, XBLA), World of Goo (2008, PC), Zelda I and II (1986 and 1988, NES), Raptor: Call of The Shadows (1994, PC), and Scribblenauts (2009, DS). These games spanned multiple decades, and most shared a core set of features. Things like collision detection, movable characters, background images, multiple viewing perspectives, top-notch audio support, artificial intelligence, paths, timing, and image processing were prevalent in some degree to each title. Some titles also supported multiplayer, networking, physics, and multi-platform support. As the Gaming Industry matured, the games tended to be more feature-rich.

One thing they did not share, however, was a cross-platform release. Every title tested and researched originally debuted on only one or two platforms. Easy cross-platform development, cross-platform interaction, and a unified development API are necessary features for the future of Game Development.

5.3 Game Demonstration

A game demo was made in order to showcase some of the functionality of the Tec Game Engine.

The first picture shows the game running on Mac OS X (built via XCode and g++), and the second picture is on Windows (built via MSVC). This demo is the starting point for developers getting their feet wet with the Tec Game Engine. It includes a number of default functions for game developers and compiles out of the box on all platforms! Some key features that are immediately noticeable: loading of game maps, movable actor class, invisible ground entities, and camera rotation.

5.4 Student Game Programming Competition

As another test to the Tec Engine, a student run game programming competition was held during the month of April 2010. The feedback from the competition helped development of the engine and managed to expose a number of previously unseen bugs.

5.5 In-house Game Development

To showcase the functionality of the Tec Engine, Sky Island Dive was created. It is a collaborative effort between Tec developers and a number of other game developers. The game is currently in development for a simultaneous release on iPhone, iPad, Windows, Mac OS X, and Linux in early May 2010.
6. CONCLUSION

The Tec Game Engine is a fully featured, working, documented, and tested cross-platform video game engine. It is capable of native compilation on a number of platforms and can be extended well into the future. Tec has incorporated nearly every feature originally planned for it, and has already exceeded its original specification. In addition, Tec is currently being used to develop a number of games for commercial release, and is, in addition, a successful, open-source project. The Tec Game Engine has succeeded in becoming a natively interoperable, cross-platform game engine.

7. FUTURE WORK

Development of the Tec Engine has no intention of stopping anytime soon. Here are some of the features aimed at a later release of the Tec Engine.

- **Out of the Box Multiplayer Support**: Based off the Cloud/Server Model for game saving and loading.
- **AI Algorithms**: For entity pathfinding and game mechanics.
- **Amazon Web Services Integration**: To have a more scalable server interface.
- **Android Testing**: Hardware support and testing for Android devices.
- **Social Network Integration**: Publish achievements and reward to web sites and leader boards.
- **Web Based Version of Tec**: This will allow the game to run in the cloud and have all interactions take place in the web browser.
- **DirectX Rendering on Microsoft Platforms**: Support DirectX for future compatibility.
- **Windows Mobile 7 Port**: Allow another mobile device into the Tec family.
- **Xbox, Wii, PlayStation, and Nintendo DS Ports**: Distant goals for the future.

8. REFERENCES