

Multi-Platform Game Development

Abbey Sparrow

December 3, 2001

1 Abstract

Game engine development represents the peak of software development efficiency; requiring real-time 2D or 3D visualization, Artificial Intelligence, Physics modeling, Networking, and it continues to be one of the few desktop computer applications that are greatly enhanced by optimization. Innovative Engines are frequently licensed, providing an incentive for developers to create new and more advanced features. Multi-Platform development further promotes widespread adoption and universal appeal, increasing the audience, not only for the developer's game, but for any licensees, as well. The selection of development tools and technologies is critical to universal compatibility. Testing, while crucial, must be constrained to a very tight 16 month development cycle in order to ensure the engine's technology is as advanced as possible and the game content follows current trends. This focus allows the most return on investment allowing for an even greater research investment during the next development cycle.

2 Introduction

The modern computer gaming industry is an evolutionary extension of more traditional pen and paper role-playing games popularized in the late 70s and early 80s and the even older table top combat strategy gaming genre. In fact, many old pen and paper games have Video games based on the plot and, more often than not, the internal mechanics of the game(including Advanced Dungeon and Dragons, Battletech and Shadowrun). These games were based on statistical modeling of the game world, giving player a set of abilities and attributes, which affected their actions but some range was left to represent random variation. This was incorporated with a die roll. Many generations of these games progressed becoming more refined, distinctive, complex and balanced in terms of gameplay.

Meanwhile Atari began making videogames and capitalized on the pin-ball craze to expose people to standup video games which were single screen and used points to rank players in relation to each other rather than a plot that could be played to completion. This type of game is based mostly on repetitive motion and hand-eye coordination, but can also rely on pattern matching. As college students who had experience with traditional role-playing games became exposed to more advanced machinery, text based games like Zork emerged. These games had a linear storyline and were really little more than text parsers using primitive commands to navigate around the game world. Later games such as bard's tale coupled the two genres with a text based story and input, but showing a graphical viewport into the game world.

Early arcade games other characters moved according to a set of static preprogrammed rules, making the game focused on reaction time and path prediction. But the advent of first person combat games originating with Wolfenstein 3-D also saw the beginning of limited AIs allowing different units to display a personality of sorts, and coordinate tactics to the users style of play. this dramatically raised the replay value of the games.

Another technology common in campuses was networking. This allowed student to write games where players on different computers could combat each other remotely, the most prolific of which is the tank assault game Bolo which allowed users to create maps and script AIs. The user's ability to create custom maps is an essential feature in a network engine, it gives the game longevity and adds value to the game without adding a burden to the developer.

All of these influences have combined to create four main genres of games, each with ver similar requirements, but very different interfaces. These are: Real-Time Strategy, First Person Shooters, 3rd person adventure, and most recently Massively Multiplayer Online Games. Due to the great similarities in requirements for most games many of the more feature-rich and open engines are licensed by third parties so that they can insert their own graphics, sound and plot resources to reduces the development cost and personnel requirements.

The varied roots of video game development have given them a robust and challenging feature set, constrained not only by the speed of the gaming market, but the real-time performance as well. Game engine research requires strict schedules, dedicated and innovative staff, and thorough re-

search and testing. But, in a market which has eclipsed the motion picture industry, the return is substantial.

3 Merging Markets

As computer hardware became more powerful and had increasingly more memory, it became possible to do real-time 3D visualizations in software, this caused an explosion of new ideas as older game paradigms were adapted. As Jez San said: "They did a pretty good job rendering 3D graphics on the fly using software algorithms but now the new 3D games consoles have arrived in abundance with more to follow. These do fast texture mapping and shading in hardware allowing incredible 3D games to cover both food groups(look great, play well) but the console companies are trying to keep them proprietary and incompatible."(Veeder) Thus, the convergence of the two major game venues (the computer gaming market and the console and arcade market) which each have many proprietary and incompatible systems has created an environment where the developer must either port the game to an ever increasing number of platforms or tie thier business to a single platform.

If the developer ties themselves to a single platform the chance of of being hurt by the consoles rate of adoption is much greater than if the developer can release on many different platforms. Conversely, making a game where many ports are developed in parallel would be a terrible financial burden in terms of development resources, and delayed release of port would negatively impact their sales, because of an increased time to market. Paul D. Lewis

said: "Film delivery might be 4 years or more, where game development cycles need to be 16 months or less to be cost effective and stay in tune with audience trends." (Veeder)

4 The Middle Man

The most commonly used solution is to use a cross-platform graphics library in conjunction with an in-house API covering all the basic needs of a game including AI, Interfaces, Turn sequencing and Object and environment representation. This allows the developer to implement the API only once for a given platform and until the API itself changes, all games developed with it will be able to run on any platform the API is implemented on. New systems are simply a matter of implementing the API for it. With this kind of approach programmers making the game need no familiarity with the proprietary API, which not only reduces the need for training, but also allow the developer to ramp up into production quickly since "many game designs take about a month to work out, with the development of scripts, characters, props, backdrops and game rules occurring at this time." (Haykin) This design philosophy allows a minimum of work to produce a maximum amount of product and much of the work is reuseable for subsequent games. Eric Lengyel says "When programming for multiple platforms, it is usually desirable to hide code that is dependant on a particular operating system by using layered design. Platform-specific code should be safely tucked away at the lowest level possible, and code inhabiting the higher levels should require no knowledge of how the lower levels are implemented. This black box

approach should be used to encapsulate major subsystems of a game engine as well as any miscellaneous functions and data types which may depend on the underlying operating system.”(Lengyel)

5 Conclusion

The widely forked and ultimately convergent path of videogame development history has produced an amazing array of devices which a developer may develop for. Using standard techniques a developer can harness the various markets of any of the platform rather than being constrained by API availability or built-in compatibility options (of which few exist). This presents the lowest cost and highest value, not only to the developer, but also to the consumer.

6 References

- 6.1 Haykin, Randy *Multimedia Demystified* Apple Computer /New Media, 1994.
- 6.2 Lengyel, Eric *Simultaneous Cross-Platform Game Development* Gamasutra.com, 2000.
- 6.3 Swadley, Richard *Tricks of the Game Programming Gurus* Sams publishing, 1994.
- 6.4 Veeder, Jane *Video Game Industry Overview:Technology, Markets, Content, Future* ACM Computer Graphic Proceedings, August 1995.
- 6.5 Veeder, Jane *New Developments in Animation Production for Video Games* ACM Computer Graphic Proceedings, August 1995.