Modular Architecture for Computer Game Design

Teddy McNeill
Thomas Jefferson High School for Science and Technology
Alexandria, Virginia

March 27, 2007

Abstract

Common current game architectures limit program flexibility and modularity. With the advent of middleware and the increasing complexity of games, this is no longer acceptable. In this project I have attempted to design and implement (using C++ and OpenGL) a highly modular, Data-centered architecture based on the "System of Systems" approach. The final implementation was not required have any significant complexity within each system (e.g. graphics, AI, etc.) but rather had to demonstrate the successful interaction of independent systems.

1 Introduction

Current game architectures are designed with Object-Oriented programming in mind. This often involves giving the game entity objects the functionality to do all the drawing, calculation, etc. involved in their use. While this method conforms to Object-Oriented programming practices, it has major drawbacks. Specifically, this technique limits the recycling of code and implementation of middleware (or the Component-Off-The-Shelf [COTS] approach). For example, if a sequel to a game were to be created that switched from 2D to 3D graphics, it should not be necessary to completely remake the game; however, that is what is often required.

This project will attempt to implement an architecture for games that allows efficient reuse of code and accomodates the COTS approach. This primarily will be based on the separation of data and calculations. The suggested architecture will use a data-centered, System-of-Systems organizational structure.

The widespread use of this architecture would allow game developers to make extensive use of middleware during the creation of almost any game. It would also reduce the time and work required to produce games with similar elements. While the direct application of this research is to games, the architecture and COTS approach can be applied to nearly any form of software development; the reusal of code and friendliness towards middleware could expedite the development of all programs to some degree.

2 Background

Jeff Plummer, in his paper A Flexible and Expandable Architecture for Computer Games, provided the inspiration for this research. Plummer attempted to solve the problem of game developers having to rewrite large sections of game code that are designed to create very similar outputs. The proposed solution was the System-of-Systems approach.

The System-of-Systems approach is data-centered, meaning that all classes are passed a pointer to a class containing all the data that represents the game world and the entities within it. All classes exist to operate on this data. This is as opposed to more common systems that would allow objects (entities) to operate on their own data.

The System-of-Systems approach, as its name indicates, considers a game no more than a group of interacting systems. Each of these systems (such as AI, Graphics, Collision Detection, Game Logic) can be represented as a class. Each of these classes would act upon the central data.

What this approach provides for is the total separation and independence of the systems. That is, within a correctly designed architecture, the Physics class need not know what it is acting on to operate properly. Thus, with the correct interfacing of classes, a middleware physics system could be added with little effort. The same could be done with nearly any system excepting Game Logic.

3 Early Development

3.1 Requirements

The requirements for the implementation were fairly skeletal. It was necessary to include several systems in order to examine their interaction and interfacing and in order to provide proof of concept. However, it was not necessary that any of these systems exhibit any complexity within themselves. For instance, the graphics system could only display cubes and the architecture could still be contructively examined.

The only other requirement was that the creation actually constitute a 3D game (rather than some other non-game type of program) in order to ensure relevance to the problem being addressed.

3.2 History

Due to the author's lack of experience with the tools at hand (C++ and OpenGL), the first section of development was spent on prototyping the necessary functionality for a game. While the systems involved did not need to be complex, this required research and experimentation.

The second section of development began with organizing and cleaning up the code that had already been written and designing the specific new architecture. This design process experienced early difficuly due to the limitations of the tools being used. Specifically, the use of the GL Utility Toolkit (GLUT) required that the input, output, and initiation of the program occur in the same class. After this was taken into account, the design was completed. It was decided that the final architecture would make use of four classes: a main I/O class, a Data class, a Physics/Collision Detection class, and a Game Logic class.

After this design was completed, variables representing game entities were removed from the prototype code and added into the new Data class. It was ensured that functionality remain the same during this process. After this process was completed and the new design was implemented, minor modifications to game mechanics were implemented in preparation for the final AI system addition.

4 Results and Discussion

The purpose of this project was to implement a modular architecture and examine its development and benefits. The completion of such an architecture could provide a base for future development of complex games requiring middleware or reusal of code.

As the research has not yet been completed, and considering the state of the current results, any conclusions or reports of results would be premature.