COS30031 Games Programming
Research Project Report

Daniel Coady 102084174

18/10/2021

Contents

Abstract

Put the abstract here once we actually have one!

1 Introduction

1.1 Background

Game architecture is an art, one that is truly hard to learn and master. In fact, there are many experts and experienced developers who disagree frequently on the topic of how the backbone of games should be structured. This happens for a variety of reasons, but more often than not we will see two aspects of various architectures compared: the usability or maintainability, and the performance.

When we talk about the usability or maintainability of something, we refer to how easy it is to work with. Of interest to us is the ergonomics related to developing with and extending on a given architecture as the needs of a solution expands in scope. Performance on the other hand is far more straightforward– how fast is it? Both aspects are incredibly important when weighing up which architecture is most appropriate to you, your project, and your needs.

1.2 Purpose of This Research

Using the metrics outlined earlier, I aim to compare and contrast four different architectures. This will involve deep dives into every architecture, dissecting how all of them work and what their purposes are. My hope is that through this research presented in this report, the reader may be able to:

- Understand what each architecture is, and how they work
- Understand the purpose of each architecture
- Understand the use case for each architecture
- Come to their own conclusions regarding choices in game architecture

2 Methodology

In order to test the various aspects of each of these architectures, I have had to formulate a series of tests and establish a common testing environment for each of these tests to be run within.

2.1 Tests

All tests will be done with a simple set of entities. These entities will be represented by squares in a graphical window, and they will move with a fixed velocity. When an entity reaches the edge of the window, it will then loop back around to the other side of the window. To add an extra layer of complexity to

the processing of the entities, a random amount of entities will also colour shift while moving. Data will be tracked in the form of average cycles per second and the output provided by valgrind's cachegrind tool.

To ensure we collect as much useful data as possible, I will extend this basis for testing in three key ways:

2.1.1 Static

The test will be run with 250,000 entities in the simulation over the course of one minute. The purpose of this test is to understand at a high level how compiler optimisation level affects the speed of each architecture.

2.1.2 Ramp-up

There will be multiple tests run for each architecture, starting at 100,000 entities and adding 100,000 with each subsequent test until 500,000 entities is reached. Each test will be run over the course of one minute. The purpose of this test is to understand at a high level how a given architecture scales up given n amount of entities.

2.1.3 Dynamic Ramp-up

A single test will be run for each architecture that starts with 0 entities. For each cycle performed, an entity will be added to the architecture until a limit of 100,000 entities is hit. Once this limit has been hit, entities will start being removed from the architecture until there are none left, and the time taken to execute will be measured. The purpose of this test is to understand at a high level how the overhead introduced by the creation and removal of entities from an architecture affect the execution time of an application.

2.2 Environment

In order to test each of these architectures, a common environment for them to run in has been established. This will take form of a simple front-end abstraction I have written on top of the OpenGL 4.3 Core API, using GLFW for windowing and other miscellaneous functionality. All code used for the environment and development of tests will be written in C++, targeting the C++17 standard at a maximum. Something to note here is that I have elected to use OpenGL 4.3 Core. This is because it is the first version of the OpenGL specification to add compute shaders to the core profile, which will become important later on for one of our architectures.

3 Introduction to Tested Architectures

This research has chosen to focus on four key architectures. The rationale behind this is while it might not be exhaustive, it will provide meaningful data points

to inform how different styles and implementations of architecture can affects the usability/maintainability and performance of your game.

3.1 Pure Object Oriented

3.2 Object Oriented Component Pattern

3.3 Entity Component System

3.4 Entity Component System With Compute Shaders

4 Data

4.1 Pure Object Oriented

4.2 Object Oriented Component Pattern

4.3 Entity Component System

4.4 Entity Component System With Compute Shaders

5 Analysis

5.1 Pure Object Oriented

5.2 Object Oriented Component Pattern

5.3 Entity Component System

5.4 Entity Component System With Compute Shaders

6 Conclusion