Looking Back on the Journey - a Reflection

Nathan Hallam

GAM 415

Southern New Hampshire University

Adam Moore

23rd June, 2024

The past few weeks were interesting to say the least. From learning about the basics of graphical APIs and how to code with C++ in Unreal Engine 5 there was never a shortage of work to be done. While I still have much to learn going forward I have learned a decent amount in comparison to where I was at the beginning of the course. I have learned to use dynamic material instances in multiple scenarios from changing colors of objects to the particles in the niagara system. I have learned to create a decal that is randomized based on a sprite sheet whenever a projectile collides with an object. I have learned to procedural generate a mesh and from that mesh a modifiable terrain. Lastly I have learned to use screen capture components to create ‘windows’ between one point and another on the map - a portal. While some of these lessons were more interesting than others I do feel that each has its uses in a variety of games across many genres and by learning about each component I was able to expand my toolset to make a section of a game utilizing these toolsets.

One of the overarching tools that I learned in this course and others was Unreal Engine 5, its blueprint system and how to program using C++. Prior experience using Unity has pushed me towards more code oriented development rather than visual scripting but each has their use cases. Blueprints are basic and fast to prototype, everything is done visually. This means that someone who is not well versed in programming can go into a blueprint and visually script an actor to perform an action as long as they have a basic understanding of how to perform the action. On the other hand coding in C++ gives more granular control over the script as compared to visual scripting in blueprint. Knowledge of programming in C++ is required but a programmer can get more out of the game engine from using C++ as compared to the limitations of a blueprint such as execution speed.

Graphical APIs have become a necessary part of rendering images to the screen and this comes down to what graphics APIs were made to do. In short the graphics API is a middle man between the application and the graphics card’s drivers. Additionally the API provides functions, contained in libraries, to programmers to allow them to easily access computer hardware of various types and on various operating systems making porting between these different operating systems easier - granted the API supports said operating system. In regards to game engines and the Unreal Engine in particular the graphics API can be used for a variety of actions. Libraries, such as the math library, gave access to needed functions to perform vector math, while the creation of a dynamic material instance allowed for an optimized way of creating multiple materials off of a single template. By using a template to make a material a baseline is set for the information that will be used by the material, and an instance of said material template can then be made with its own properties that derive from the template. This works similarly to other blueprints in Unreal where there is a ‘template’ and instances of it that can be modified. This also ties into how much memory is being used in the backend. The template of the material only uses one set of code, a single script, that the instances of the material all interact with. Since the material instances only use one set of code the amount of storage and memory being used is reduced overall. One real example is the storage requirements for modern games needing hundreds of gigabytes to store the 4k textures as compared to older titles that were limited by storage sizes and had to optimize or recycle old textures already in the game files.

Experience in anything we do allows for better decisions made later as we have a wealthier knowledge base to work off of. In Unreal Engine developers can decide to use C++, Blueprint or a mix of both. When to decide to use one over the other depends on the issue at hand in my opinion. I feel that blueprints can be used to prototype faster than C++ but C++ is better overall due to the increased performance compared to blueprint. Unreal Engine extends this further with the multitude of tools built into the engine that further help to prototype a game, such as the 3d model editor. Ultimately decisions should be made based on the needs of the project but are still limited on the knowledge of the individual and team.

As games have grown over the past few years APIs have begun to adapt to the increasing and everchanging need for games to run on a variety of hardware combinations. This has caused some APIs to become obsolete while others have become standardized across multiple operating systems. OpenGL is one such API that is being phased out in favor of Vulkan. Despite this OpenGL is still used for its support of backwards compatibility with older hardware but due to issues with supporting old hardware while adding new features OpenGL was replaced by Vulkan. Because of this Vulkan became a widely used and supported API that boasts little overhead for developers while supporting multiple operating systems. Overtime Vulkan has become more of the standard API to use over DirectX as it is supported on multiple systems as compared to DirectX which is only supported on Microsoft systems. In regards to trends in today's industry the implementation of high end realistic physics is overshadowed by AAA stagnation and technical limitations. For a singleplayer game like Teardown or Kerbal Space Program the implementation of realistic physics is based on the capabilities of the hardware running the calculations. In contrast a game like Battlefield has destructible environments to some degree but the majority of the destruction is masked by particle systems leaving little in terms of physical collisions. This is because the network is a bottleneck for physics calculations that would otherwise get swarmed by collision packets if every single segment of a wall were to break off and collide with one another. Instead, as previously stated, the destruction is masked by particle systems to give the illusion of the environment being destroyed while allowing each client to render the scene on their own without needing physics packets to be sent for every object over the network. However due to this paradigm there are little to no games on the network that have intense physics simulations that are currently reserved for single player offline games.

Overall the use of graphical APIs has allowed for greater performance across multiple systems thanks to clever implementations of functions and the reduction of bottlenecks between hardware and software. While this course only provided an introduction to low-level APIs in the Unreal Engine I hope to experience more from 3d graphics and better learn to utilize the available tools given by graphical APIs like Vulkan.

References

ITFreeTraining. (2022, December 24). *Graphics API*. YouTube. https://youtu.be/ppGasKJ\_-Wk?si=ACH5OXE0dFIoT0vo

*Unreal Engine C++ vs. Blueprint: Which Should I Choose? - techarthub*. (n.d.). Retrieved June 21, 2024, from https://techarthub.com/unreal-engine-c-vs-blueprint-which-should-i-choose/#:~:text=C%2B%2B%20functionality%20is%20usually%20much%20quicker%20than%20its