Matthew Nor

Lucas Christenson

CSE 165 Term Project: Dino Run Game

Dec. 11th, 2021

Introduction:

For our final term project for CSE 165, ‘Intro to Object Oriented Programming’, we were tasked with creating a 2D platformer running through OpenGL, a cross-language, cross-platform programming interface with the ability to render 2D/3D vector graphics. This is achieved through an API used to interact with the user’s GPU (graphics processing unit), to produce rendering results backed by the unit’s hardware. Requirements for the game included two separate modules, one that is controlled by a user and the other controlled by an autonomous user. In addition, the use of the SOIL library was to be implemented, a specific C library commonly used for uploading textures. With these objectives, we wanted to create a game that displayed the knowledge we learned in this class utilizing specific uses of objects and classes.

Motivation:

In our pursuit to create a game, our team drew inspiration from common platformers we were already aware of from our experiences growing up as children. Popular games that reflected the same aspects as were required from the project included notable titles such as ‘Space Invaders’, ‘Flappy Bird’, ‘Super Mario’, and ‘Pacman’. We wanted to create a game that followed a similar theme to these side scrollers, and we landed on a conventional game that many people have played at one time or another, ‘Dino Run’.

Dino Run is a side scroller game similar to ‘Flappy Bird’ and ‘Super Mario’ where the goal of the game is to control your character ( a dinosaur ) across a barren desert jumping over obstacles that come into your path ( cactuses & pterodactyls ). As the game progresses, the speed of which you are traveling increases, requiring the player to time their movements accordingly. There are no levels, and the simple goal of the game is to run as far as possible. The game ends when you collide with an object. We chose this game because it was quite simple in our approach, and the functions of the game’s objects could be implemented straightforwardly. We didn’t want to bite off more than we could chew on this project, as we were already fighting an uphill battle, coming into this project unfamiliar with any background in game development and the programming interfaces we were to use. Challenges to face include learning how to utilize and implement the respective OpenGL interface while including the SOIL library.

Methodology (Design & Implementation)

1. *Libraries*

The use of libraries is essential in developing an OpenGL application. Some necessary implementations such as rendering objects on screen and defining window parameters are functionalities imported by specific C libraries targeted at OpenGL such as GLFW, glad, glm, & KHR.

SOIL: A tiny C library used primarily for the ability to upload textures into OpenGL. Meant to be used as a static library (hence its compact size and public availability).

GLFW: Offers necessary elements required for rendering objects on the screen. It allows for creation of an OpenGL context, defining window parameters, and handling user inputs.

Glad: Used in the retrieval of the location of the functions the developer needs to find the location of the functions and storing them in function pointers for later use.

glm: Stands for ‘OpenGL Mathematics. It is a set of C++ classes and functions that fill in programming gaps for writing basic vector and matrix formulas in OpenGL applications. Specific in our application, this is utilized in performing transformations, a crucial aspect of controlling our character in our platformer.

KHR: A mechanism through Khronos OpenCL exposes multiple separate vendor installable client drivers for OpenCL. Since extensions vary from a multitude of platforms as well as a vast array of different drivers, as developers we cannot expect interfaces for all the extensions we have implemented in our library to be defined in the standard format (ex: gl.h, glx.h, wgl.h.)

1. *CPP Files*

Game.cpp: In this file, most of our games core functionalities are implemented. Firstly, the initialization of the game’s initiation is performed at start. Rendering sprites consists of declaring and initializing float positions for the coordinates of where the textures are to be rendered. Shaders are initialized for the background images we have stored in corresponding to the game’s background, enemy sprites, and user sprites. Positions are labeled and connected to transformers to communicate movement into the frontend. A function for CheckCollision() checks the coordinates attributing to the X & Y axis to return if the user’s character bumped into the enemy, thus resulting in death.

GameObject.cpp: Declares the position, size, and velocity of our sprite ID.

Main.cpp: This is the main file of our game application. Set initially are the window dimensions for the game’s screen, alongside with the main function where the application begins and proceeds towards the intended loop. First the required options for the appreciated libraries are attributed; GLFW creates a window object that we can utilize for its [GLFW] functions. The viewpoint dimensions are defined with the width & height accordingly.

The specific variables for our game's initialization include data(), text(), Pos(), posEnemy(), and background(). Next, the loop is started where we analyze all the corresponding response functions and our renders are set in place.

Shader.cpp: The file initially retrieves the vertex/fragment source code from the system’s filePath. Code is implemented in opening the shaders files, and reading the file’s buffer contents into streams. Shaders are created for the vertex, fragment, and shader programs.

1. Additional Resources

Res: In this subfolder, some basic .fs and .vs files are used to retrieve specific images we want to render on screen, including objects for the background, enemy, and sprite.

Outcome ( Evaluation )

The outcome of our work resulted in a functional game that shows basic object functionality in an OpenGL environment. Although we were formerly lackluster in our initial presentation for the code’s demonstration, we worked to finalize the rest of our application by the end of our due date. We were able to fulfill all the aspects we wanted to be required in our side-scrolling game, including object collision and proper, smooth rendering for objects. Bugs that were attributed to consistent performance in our system were fixed by limiting the alleged frames per second (fps) displayed.

In the future if we had more time to develop our project further, other aspects of the game’s functionality we could include could be multiple stager integration, a score keeping system, and possibly even a boss event triggered after a certain amount of time.

Conclusion :

The final product of our game’s application demonstrates for us the opportunity to learn the OpenGL system with very little prior knowledge in any basic game development up to this point. The use of multiple classes and libraries integrated into our program taught us valuable skills in rendering. Most importantly, the ability to create was given to us, and thus going forward we are able to use this project as a crutch towards any future developments we plan to tackle in game development.

Works Cited

1. “LearnOpenGL - Introduction.” *Learnopengl.com*, learnopengl.com/Introduction.

- Basic introduction to openGL, gives information regarding to getting started as well as basic understanding of rendering, shading, and lighting.

1. “Setting up OpenGL and Creating a Window in C++.” *Www.youtube.com*, www.youtube.com/watch?v=OR4fNpBjmq8. Accessed 13 Dec. 2021.

* Guide to setting up OpenGL and creating your first window.

‌