MSIM 441/541 & ECE 406/506
Computer Graphics & Visualization

Programming Assignment Three: Transportation Simulation

Thomas J Laverghetta
Tlave002@odu.edu
757-620-7607
Virtual
MSIM411
12/16/20

Introduction

The transportation simulation project is designed to allow players/users to drive a simulated taxi through a virtual world that includes roadways with markings, an intersection with a traffic light, two roadside billboards, and surveillance camera on each corner. This document will go over the design, implementation, and results for this simulation so to illustrate use and how to build upon what has been done.

The report will have three sections: program design, program tasks and results, and conclusion. The program design will discuss the Doxygen documentation generated for this simulation. Task and result are the main section of this document, and will discuss the simulation's design, implementation, and results from implementation. Lastly, conclusion will discuss accomplishments, learning outcomes, difficulties, and future improvements.

Program Design

The design documentation for the programming assignment was made using Doxygen. Doxygen is a tool for generating documentation from annotated C++ sources (classes, enums, variables, functions, etc.), and programming languages (C, Objective-C, Java, etc.). To make documentation using your source code with Doxygen you comment your source code with specified markers and description of the code. The markers give specific meaning to the comments (such as identifying the code is a class, enum, variable, function, etc.). The markers and other documentation on Doxygen can be found on their website (https://www.doxygen.nl/manual/docblocks.html). Using these markers, I commented my code and ran Doxygen wizard tool. This generated HTML design documentation (PA_3_Laverghetta_Thomas.chm). The HTML design documentation illustrates the functions and variables used. The functions and variables will all have short briefings explaining their use within the program and any corresponding dependencies. Doxygen will also illustrate how the functions are connected and where dependencies are.

Tasks

Customized Car

For this task, I replaced the previously used Honda S2000 OBJ model (used in PA2) with taxi OBJ model provided by Dr. Shen. The taxi OBJ along with respective MTL and texture files were provided by Dr. Shen; although, the texture images provided were in JPG format, and object parser only accepts PPM format. Therefore, I had to reformat files.

To reformat JPG to PPM I used GIMP. In GIMP, I would open the JPG file, vertically flip the image (needed because JPG files store the pixels by starting from bottom-left corner then proceeding in row major form, while PPM files start from top-left corner then proceed in row major form) and used GIMP's export capability to export modified JPG to PPM in ASCII format.

I was also tasked with creating a custom license plate for the taxi with the following text: "Thomas MSIM441". To do this, I modified the previous license plate by replacing its text (ANT 1CS) with required text using GIMP.

Billboards

For this task, I implemented two billboards using a modified billboard class. A billboard class template was provided by Dr. Shen (i.e., the template provided basic structure and attributes) which I had to modify to allow for the following functionality: to read in PPM billboard images and create textures for billboards (ReadFile), mutation methods to set billboard positioning (SetSize, SetLocation, SetOrientation), and draw method to draw billboard geometry and pillar (Draw). (functionality can be found in BillBoard.cpp). Once functionality was added, I created two billboard instances too create two billboards in virtual environment. Figure 1 illustrates the two billboards added to the virtual environment.

Figure 1. Billboards in virtual environment

The following are the questions and answers to billboard task questions:

- What is the purpose of the OpenGL command glBindTexture? Why do we need to use this command?
    - glBindTexture initially binds texture objects to texture data, including the image arrays and texture properties. Then when bind (rebind) is used outside of initialization, OpenGL will set texture object as the currently active texture object (any subsequent texture operations will affect the currently bound texture object).
    - I used this command to initially create the texture object mapping (mapping texture name generated with texture type (GL_TEXTURE_2D)). Also, during initialization phase, I use the fact it makes it active to set texture parameters (e.g., linear filtering). Then I used the command within billboard draw method to activate texture object so to display billboards.
- What is the difference between parameters GL_REPEAT & GL_CLAMP in OpenGL command glTexParamteri?

    - Texture coordinates can be outside the range [0,1], and GL_REPEAT and GL_CLAMP are rules which specify textures behaviors outside the range [0,1]. GL_REPEAT repeats the texture image outside [0,1], the default behavior for textures. GL_CLAMP clamps the coordinates between 0 and 1 (i.e., empty space outside [0,1]). For my program, since my textures never go outside the range [0,1], I left them to the default values (GL_REPEAT).
- Why do you need to use glEnable(GL_TEXTURE_2D) and glDisable(GL_TEXTURE_2D) in different places in your code?
    - glEnable and glDisable enable and disable various capabilities, respectively. GL_TEXTURE_2D enables the use of 2D texturing. Therefore, I use glEnable(GL_TEXTURE_2D) and glDisable(GL_TEXTURE_2D) in different places to allow me to conduct texture operations. Specifically, the operations are done within billboard's draw method to allow me to use glTexCoord2f.

Snapshot

For this task, I created a key-binding to take snapshots of current OpenGL window when the 's' key is pressed. To do this, I created a key-binding in Keyboard callback method which when called will allocate memory needed to create snapshot, read in OpenGL window pixels and save it to allocated snapshot image (saved in PPM format), vertically flip image (since in PPM format), and finally, output image to project directory with name "snapshot_<number of snaps>.ppm". Figure 2 shows code to conduct operations.

```
case 's': // taking snapshot
    printf("Starting Snapshot\n"); fflush(stdout);
    snapshotCounter++;
    snapshot.AllocateMemory(winWidth, winHeight);
    glReadPixels(0, 0, winWidth, winHeight, GL_RGB, GL_UNSIGNED_BYTE, snapshot.image);
    snapshot.VerticalFlip();
    snapshot.WriteFile("snapshot_" + to_string(snapshotCounter) + ".ppm", "P3");
    printf("Done Snapshot\n"); fflush(stdout);
    break;
```

Figure 2. Snapshot Key-Binding Code

The following are the questions and answers to snapshot task questions:

- Why is a vertical slip needed before saving the snapshot?
    - Because glReadPixels return pixel data from the frame buffer, starting with the pixel whose lower left corner into snapshot location data. Therefore, causing the outputted image to be vertically flipped. Thus, to make image right-side-up, I vertically flipped image.
- You need to call the function PPM::AllocateMemory before calling PPM::WriteFile. Why?
    - PPM::AllocateMemory allocates the memory needed to save image.
- Explain the destructor of PPMImage does. Why is there a pair of square brackets?

    - PPMImage destructor is deleting image data. The square brackets signify to C++ delete command that image is array; otherwise, C++ delete command would delete image as a non-array variable (delete the first element and not the entire array).
- What are the two types supported by PPM images? Compare the type types (e.g., difference, advantages, disadvantages). How are the two types handled differently in PPM::WriteFile?
    - The two supported types for PPM are binary and ASCII. When comparing the two, binary files tend to be more efficient than ASCII. This is because while the basic unit of information is very straightforward in a plain text file (one byte equals one character), finding the actual data values is often much harder. For example, to find the third data value on the tenth row of a CSV file, the reader software must keep reading bytes until nine end-of-line characters have been found and then two delimiter characters have been found. This means that, with text files, it is usually necessary to read the entire file to find any value. [1]
    - PPM:WriteFile handles binary by using fstream's binary output for binary file format and writing image whole (not concerning with formatting file with element specifications). PPM:WriteFile handles ASCII by outputting data in row major format.

Automatic Snapshot of Traffic Violation

I created functionality that automatically snapshots the car (taxi) when making traffic light violations in north direction (i.e., when taxi is middle of intersection during red light). The functionality I created was implemented in the update function after taxi position and traffic lights have been updated. The program checks if north-south signal is red, taxi is heading north, and taxi is in intersection. If true, the program will take three snapshots while taxi is in intersection and output snapshots with current date and time.

Conclusion and Discussion

Accomplishments

I was able to expand PA2 by replacing Honda S2000 with taxi model, implementing two roadside billboards, implementing key-binding to allow users to take snapshots of OpenGL window, and implement an automatic snapshot when traffic light violation has occurred.

Learning Outcomes

I learned how to use GIMP, how to use textures in OpenGL, and the PPM format.

Difficulties

For the most part, the assignment was straight forward, especially since it just expanded upon PA2.

References

[1] P. Murrell, "Binary file structure," [Online]. Available: http://statmath.wu.ac.at/courses/data-analysis/itdtHTML/node58.html. [Accessed 14 12 2020].