MSIM 441/541 & ECE 406/506

Computer Graphics & Visualization

Programming Assignment Two

Transportation Simulation

Jeffrey McAteer

Modeling & Simulation

jmcat001@odu.edu

540-842-6017

Virtual

2022-11-14

**Introduction**

Dear Mr. Jones, as you know I have recently been promoted to Chief Information Officer and wanted to thank you for stepping up to fill the role of Project Lead for our company’s safety simulation program, Traffic Simulation.

The goal of Traffic Simulation is to enable our customers to view their new car designs from multiple angles in realistic environments before they proceed to engineering and development. The simulator needs to support all types of motion that real cars can perform, render views of the vehicle from multiple angles, and allow users to add custom models from their marketing teams to the scene for promotional imagery.

The project is still a little young, but the fundamentals have been completed and are under-budget. The road has recently been modified to conform to US driving standards, stoplights have been added and change signal with 5s green, 1s yellow, and 6s red timings. Cameras have been added to each stoplight then rotated to face oncoming traffic, and a bored engineer has put in a Bench object model that none of our customers asked for. One of the things I’m hoping you will pursue is adding arbitrary models at run-time, but we can discuss that once you’re familiar with the project at the next company planning meeting.

If you want to test it out before meeting your team I’ve had IT install a copy on your PC. After starting it, use the arrow keys or WASD to move the vehicle forward, backward, and rotate it. If you should lose control of the vehicle we have a safety braking system, press ‘b’ to stop immediately. You may also reset the simulation with ‘r’, and ‘q’ exits the simulator. While driving your simulation heading and speed will be displayed in the middle of the screen, while the alternative view angles customers use for marketing will be displayed at the top of the screen.

Good luck with your new team, and welcome to the simulation division!

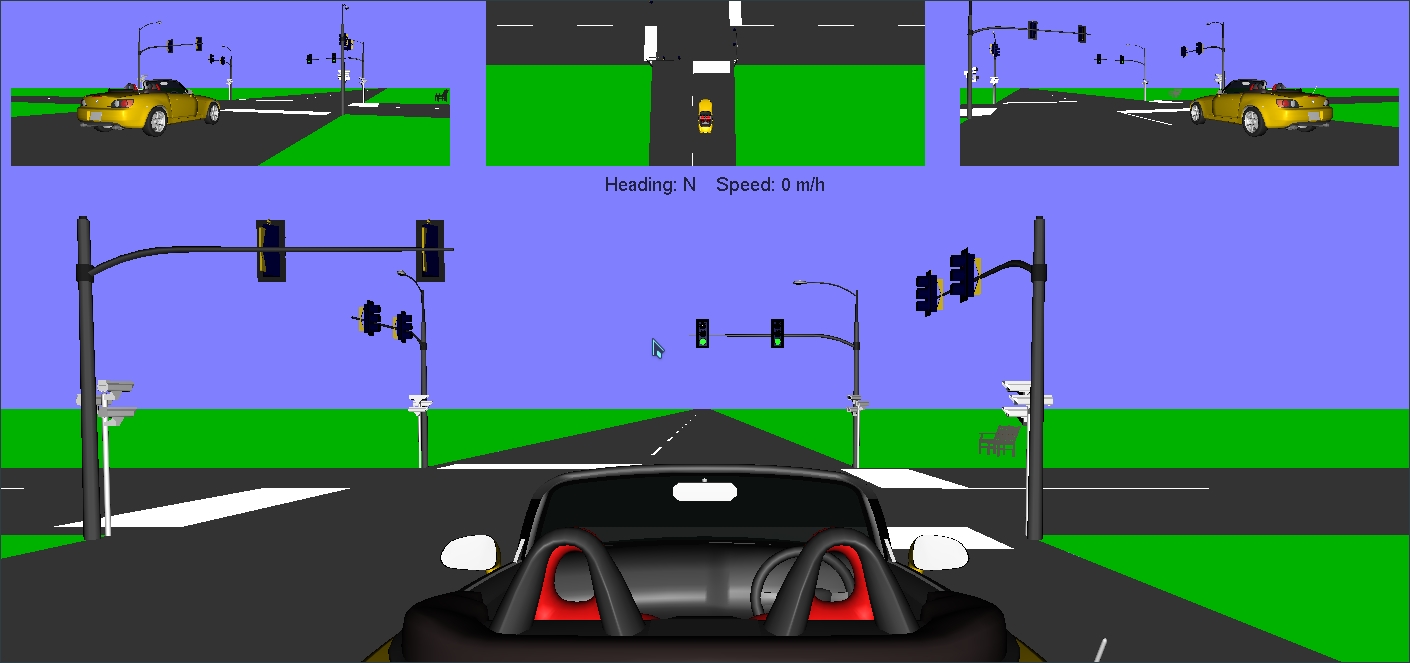
**Program Design**

See the doxygen/ folder in JeffreyMcAteerProject02.zip.

**Results**

**3.1**

The yellow center line was modified to be a different color and shorter, then two for loops were used to generate many of these lines with a length of 9 units and a stride of 20 units (11 units separation between lines). A similar process was done to generate four white stop bars as required for US stoplights.



**3.2**

The traffic pole and camera draw commands were copied and glRotatef was used to rotate the pole about the Y axis in the correct directions for the 3 other placements.

The traffic pole is located at x=0,y=0 pointed in the Z direction along its local axis (these measurements are from Blender, which swaps the Y and Z axis names to match what industry uses).

glTranslatef is be called before glRotatef to rotate the model around the world 0,0,0 coordinate; if glRotatef was called first the pole would be rotated locally and then translated. The end transformations are identical no matter what order glTranslatef and glRotatef are called in.

The Camera model file only extends 0.776471 in it’s coordinate space, and the traffic light extends a maximum of 1.0 units in its coordinate space. If glScalef were not used to make the traffic light 3.65x larger, it would be tiny compared to the road and the camera.

**3.3**

The TrafficLight class was completed and setSignal was implemented to overwrite materials["\_Red\_"], materials["\_Yellow\_"], and materials["\_Green\_"] with either an on or an off texture based on what signal state the light was in. The material names were read from the bottom of src/Models/TrafficLight.mtl. A bug was also fixed in ObjModel::ReadFile which looped infinitely when reading CLRF text files on unix systems (windows transparently converts “\r\n” to “\n” when a file is opened in text mode, so windows users would not have discovered the bug).

**3.4**

The update() function was implemented to integrate the localCarSpeed .y into worldCarSpeed.x and .z by taking the sin and cos of the carDirection and multiplying that by the car’s local .y speed.

Once worldCarSpeed was known, carPosition is updated by multiplying the world car speed by updateInterval / 1000.0, which is a simple riemann sum that works for this simple simulations. If we were doing collision detection we would want a more detailed integration that would not clip through geometries, but this simulation does not use object collision.

Braking is done by setting the localCarSpeed.y = 0, and reset is done by assigning constants to carPosition, localCarSpeed, carDirection, and carHeading.

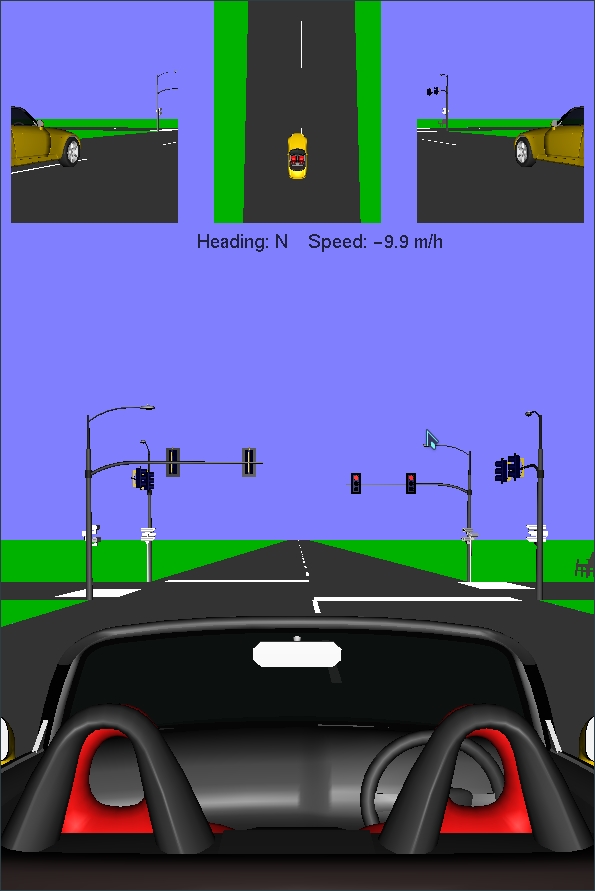
The eye location parameters used for gluLookAt for the center view is determined by the car’s carPosition x,y,z coordinates with a fixed offset, and a Vector3 eyeFocus position is calculated by using the car’s carDirection to compute a location in line with but in front of the car.

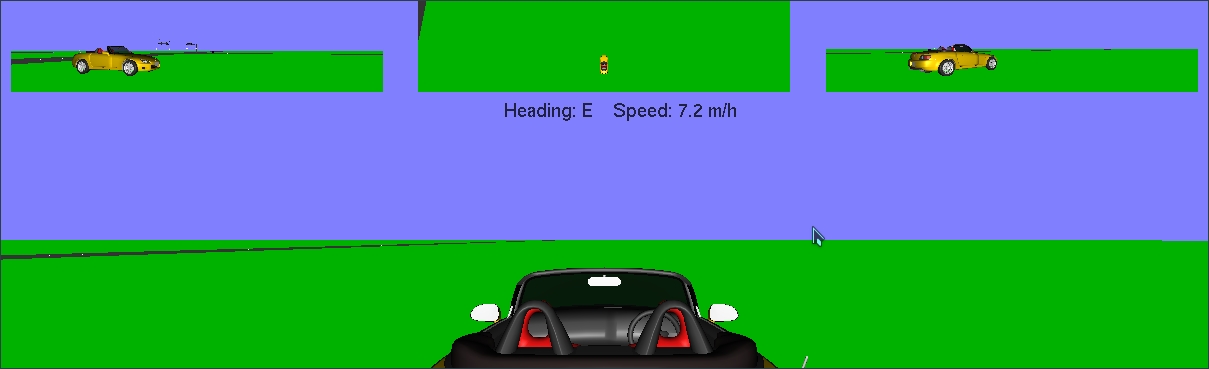
**3.5**

The eye location parameters used for gluLookAt for the top 3 views are determined by the car’s carPosition x,y,z coordinates with a fixed offset.

For the center view, the up vector is determined by taking the sin/cos of carDirection for the x and z values. Y is a constant (0) for the center view up vector.

Width and height values for the small viewports are calculated using the global winWidth and winHeight values, so resizing the main window should transform the entire window smoothly.





**3.6**

Originally my plan was to add a tree or some bushes to the scene, but no freely-available .obj files had less than 10,000 nodes which was very heavy to load. I instead found a bench model to add, but it doesn’t do much for the scene. If time were no factor I’d use Blender to design a model myself.

**Conclusion**

Traffic Simulator bridged the gap for me between simulations and video games; while the program we wrote is simple it shows how movement mechanics and position integration make modern games possible. Multiple camera angles gave me new insight to how GL rendering can scale to support very complex scenes by batching areas to be rendered; there is no rule that the top views must render fully every frame, they could be updated every other frame for a performance benefit. Sometime in the future I may implement the arbitrary model loading, that would be fun.