AN ABSTRACT OF THE THESIS OF

Jessica L. McGregor for the degree of Honors Baccalaureate of Science in Computer Science presented on June 9, 2006. Title: Believable Automotive Flocking Model.

Abstract approved: ______________________________
Mike Bailey

The high cost of manually producing background characters creates a demand for a way to automatically generate plausible behaviors. These background extras need to behave in a manner that is believable such that they do not distract the focus of the audience from the primary action occurring in the scene. Vehicles constrained to a road are the extras that are addressed in this project.

Craig Reynolds's work on automated bird flocking was a large inspiration for the general algorithms used in this project. The order of priorities in determining how to react to those objects around it are extremely similar between birds and drivers, the number one priority being not to run into anything. All of the rest of the preferences for specific types of behaviors have weights, each car containing unique preferences.

With the proper distribution of personality traits, the resulting simulation produces believable traffic. This distribution has proven to be quite interesting. The most believable traffic results from a generally aggressive, impatient set of drivers.

Key Words: Traffic automation, OpenGL, GLUT, GLUI, video games

Corresponding e-mail address: jessica.janik@gmail.com

Believable Automotive Flocking Model

By

Jessica L. McGregor

A PROJECT

Submitted to

Oregon State University

University Honors College

in partial fulfillment of
the requirements for the
degree of

Honors Baccalaureate of Science in Computer Science (Honors Scholar)

Presented June 9, 2006
Commencement June 2006

Honors Baccalaureate of Science in Computer Science project of Jessica L. McGregor presented on June 9, 2006.

APPROVED:

Mentor, representing Computer Science

Committee Member, representing Computer Science

Committee Member, representing Computer Science

Dean, University Honors College

I understand that my project will become part of the permanent collection of Oregon State University, University Honors College. My signature below authorizes release of my project to any reader upon request.

Jessica L. McGregor, Author

ACKNOWLEDGEMENT

First, I would like to express my gratitude to my mentor, Dr. Mike Bailey, for the guidance and support. Without your knowledge, expertise, and encouragement this project would not have been possible.

Next I would like to thank my committee members, Dr. Eugene Zhang and Dr. Ron Metoyer. I am most indebted for your service on my panel, and sincerely appreciate all of the time and effort you have given.

Finally, I would like to express my appreciation to my family, for always supporting me in every endeavor. Mom and Dad, thank you for everything you have done over the years; I wouldn't be here without your support. Chris and Gwendolyn, my most heartfelt gratitude to you for the patience and understanding throughout this whole process, which occasionally meant family time being sacrificed in the interest of completing this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

To my Gwendolyn

1 – INTRODUCTION

In any movie or game production, there are always background "extras" such as people, animals, and various vehicles. These extras serve as an added element of ambiance. They are what they should be, where they should be, and behave in entirely plausible ways. We hardly notice that they are there, but certainly do notice when they are not. The wildebeest stampede in the movie *The Lion King* and the Courescant traffic scenes in *Star Wars Episode I, II, and III* are prime examples of background extras that are an integral of the scene that must be believable enough to be overlooked.

We do not want to waste any valuable production time creating that behavior manually. Rather, we would like to acquire them automatically, since these extras need not act in an absolutely accurate manner. For this reason, we are able to make optimizations in calculating them, rather than being constrained by costly computations.

Additionally, due to the lack of scalability of time and cost, there needs to be an alternative to manually animating background traffic. Generating these background extras automatically maintains consistency between each car in all of the frames, and greatly reduces the production costs incurred for something that is not even the main focus of the scene. This thesis project addresses this issue in a real-time, interactive program.

2 – PREVIOUS WORK

We call this model “automotive flocking”, in deference to Craig Reynolds’s [Reynolds1987] original work on bird flocking behavior. Reynolds was the first to accurately simulate the flocking behavior seen in birds, where each bird makes its own decisions about how and where to fly.

Like bird flocking, in automotive flocking, various vehicles all occupy the same road space. Each must be aware of others around it, and make behavioral decisions based on where the others are and how they are moving. The first priority of both the birds and the vehicles is to avoid collisions. The second priority of each of the birds is to stay with the group, in terms of velocity. The flock has a group speed that each of the birds within it matches fairly closely. The third priority of the birds is flock centering. Each bird attempts to move to the center of the flock, which results in the flock staying together as a group. Since flock centering is the least applicable priority for vehicles, it is not used in determining how each car moves.

The range of behaviors used to describe the participants in the automotive model is quite different than the bird model. Various driving behaviors are defined for each car, such as how much to “push” the speed limit, the tendency to tailgate, the willingness to pass, and how quickly to accelerate.

3 – PROGRAM ORGANIZATION AND DESIGN DECISIONS

3.1 – Object Oriented Design

Due to the nature of the simulation, an object-oriented approach was logical. There are two classes of objects, car and track. The purpose of the track class is to maintain a set of car objects. Any number of tracks can be instantiated, so that multiple roads can be present simultaneously.

In the car class, each car on the track maintains its own separate driving personality parameters, which are used to determine how it behaves in relation to the other vehicles on the road. The car class is set up in a way that allows for subclasses of driver types to be easily implemented, which is a distinct advantage to using an object-oriented design approach.

3.2 – Car Behavior Characteristics

Each car object determines how it moves based on a number of constraints that use its personality variables, as shown in Table 3.1. Such variables include the minimum following distance a given vehicle is comfortable with, its preferred following distance, its preferred speed, its current speed, and its willingness to change lanes. Each vehicle also keeps track of its car type; a unique identifying number; the nearest cars in the four basic directions, left, right, front, and back; and which, if any, direction it is changing lanes.

Table 3.1: Car Class Variables

Variable Name	Type	Description
lane_change_willingness	float	Willingness to change lanes
min_dist	float	Absolute minimum following distance
mph	float	Current speed used to move the car
old_mph	float	Current speed at the last update
preferred_distance	float	Preferred following distance
speed	float	Preferred speed
car_type	int	Index of car model type
id	int	Unique identifying number
cbi, cfi	int	Index of closest back and closest front cars
closest_left, closest_right, closest_front, closest_back	float	Distance to closest car to the left, right, front and back
changing_left, changing_right	bool	Flag indicating whether car is changing lanes in left or right direction
left_to, right_to	float	Desired destination of lane change

Car class functions are used in the determination of how to move at each time step, see Table 3.2. One such function is MoveForward(), which calculates and sets the current speed at which the car is moving, and determines whether to check for a lane change. The function CheckLaneChange() first checks for the nearest cars surrounding it, keeping track of which cars they are and how far they are from the current vehicle. Next it determines which of the bordering lanes are open and then chooses which lane to move into.

Table 3.2: Car Class Functions

Function Signature	Description
void CheckLaneChange()	Checks bordering lanes for availability, choosing the best option abailable.
void FindClosest()	Determines which cars are nearest in front, back, left, and right directions.
void MoveForward(float elapsed_time)	Current speed used to move the car

3.3 – Track Behavior Characteristics

The track maintains properties of its own as well, including an array of car objects; the location of the beginning and ending of the track; the speed limit for the track; what the safe stopping distance is, given the speed limit; the average speed of the vehicles; and the number of vehicles on the track.

As each car checks for where the other cars are and what they are doing, it accesses them through the track's list of cars. The main program also accesses the list to determine where each of the vehicles is on the track. It uses this location to draw each of them in the graphics window update.

The safe stopping distance was originally a car class variable that was calculated based on current speed, but this produced extremely unrealistic behaviors. Since most real drivers appear to leave a consistent amount of space in front of them regardless of their traveling speed, this variable was moved into the track class. In there, the safe stopping distance can also be changed, causing the cars to react and modify their behaviors accordingly, while the program is running. This could be used in the case that something on the road changes, such as fog appearing.

3.4 – Simulation: How to Update at Each Time Step

Any time the CPU is idle, the GLUT framework calls the GLUT idle function, whose decision flowchart is shown in Figure 3.1. Each time the idle update function is called, it first calculates the time that has elapsed since it was last called. This number is passed to the track to be accessed by the cars, who use it to determine how much time has elapsed, and consequently how far forward to move. This is one way to keep the cars moving the same speed on the screen regardless of the frame rate. Without the elapsed time being taken into account, the cars' movement on screen was in accordance with the frame rate, either racing off too quickly or too slowly.

Each vehicle, at each time step, first takes note of the distance to the cars immediately surrounding it in the front, back, left, and right. Next, it decides how to move forward – how fast to go and whether to change lanes.

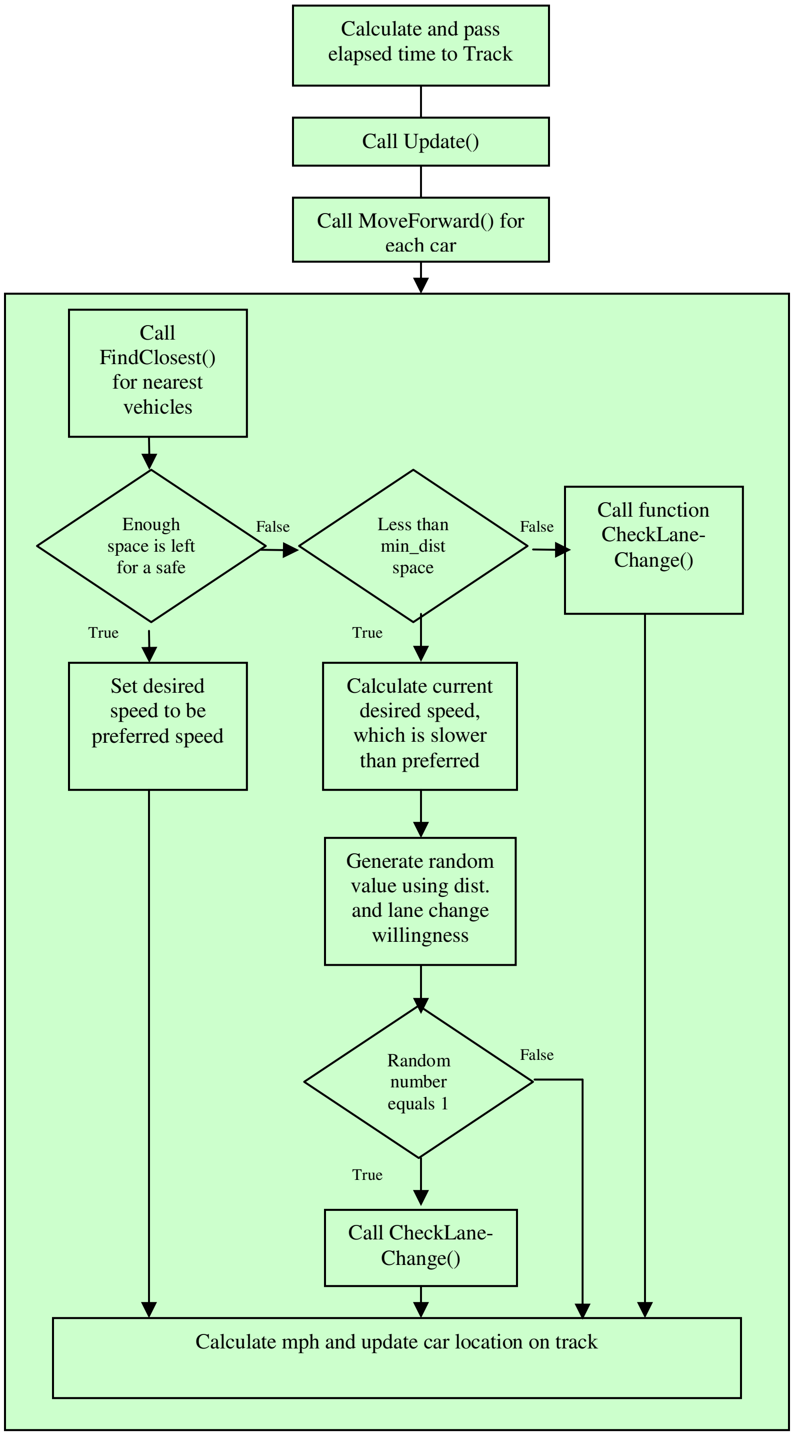

Calculate and pass elapsed time to Track
Call Update()
Call MoveForward() for each car
Call FindClosest() for nearest vehicles
Enough space is left for a safe
False
True
Less than min_dist space
False
True
Call function CheckLane-Change()
Set desired speed to be preferred speed
Calculate current desired speed, which is slower than preferred
Generate random value using dist. and lane change willingness
Random number equals 1
False
True
Call CheckLane-Change()
Calculate mph and update car location on track

Figure 3.1: Idle Update Decision Flowchart

If the current vehicle is leaving at least enough space for a safe stopping distance, the current desired speed is set to be its preferred speed, which is calculated using the expression, `preferred_speed * speed_limit`, where `preferred_speed` is a value close to 1.0 that represents the speed the vehicle would ideally like to maintain. Otherwise, if there is less space between the current and next forward vehicles than the current car's minimum following distance, it checks to change lanes.

If neither of the above is the case, and the distance to the car in front of it is greater than the minimum distance as well as less than the safe stopping distance, the current desired speed, `mph`, is set based on the following expression:

$$\texttt{preferred\_speed * speed\_limit * } \frac{\texttt{closest\_front}}{\texttt{safe\_stopping\_dist * preferred\_distance}}$$

Where `preferred_speed` and `preferred_distance` are scaling factors for the speed and distance ranging from around 0.7 to 1.3, and the speed limit is the velocity in miles per hour. The fraction from the abovementioned speed expression, $\frac{\texttt{closest\_front}}{\texttt{safe\_stopping\_dist * preferred\_distance}}$, scales proportionately with the distance to the car in front, causing the current speed to drop as the gap narrows. Using the product of the distance to the car to the front and the willingness to change lanes as the upper bound, we generate a random number. If that number is 1, then the CheckLaneChange() function is called. This means that the smaller the gap between the cars, the more likely a 1 is to be generated, and a lane change occurs. Using a random number to make the decision to change lanes greatly improves the believability of the simulation, since there is a bit of randomness to when exactly drivers choose to change lanes. Like real drivers, the generated ones, too, become more eager to switch lanes, as they get closer to the car in front of them.

The CheckLaneChange() function checks the lanes on either side of the current vehicle for enough room to change lanes. If there is a car in one of those lanes that is also within a certain distance of the current car's front or back, that lane is ruled out for the current lane change. Other vehicles that are in the process of changing lanes into that same space in the bordering lanes also remove the lane as an option for the current car. Next, the car decides between the lane options. If the left is open and either the traffic in the left lane that is in close proximity of the given car is moving faster or is further up, leaving more free space, then the car sets its destination to be the left lane. Otherwise, if the right lane is open, or there is more space and faster traffic, the car sets its destination to be the right lane.

When one lane is chosen, the elapsed time variable is used to move a fraction of the way to that lane, so as to maintain a constant speed independent of the frame rate. If neither of the lanes is chosen, the car continues on, using the current speed to determine how far the car is going to move in the forward direction. Again, the elapsed time variable is used to make that calculation to keep the cars moving consistently.

Once any lane changing for the given time step is completed, the current speed to be used is calculated by weighing the last time step's current speed for that car, and the current desired speed. Originally, the last current speed was not taken into account at all in the next current speed calculation, but that produced over-exaggerated acceleration and deceleration. After much calibration, the weight for the old speed was determined to look most realistic when weighted at 75 percent if the new desired speed is faster than the old, and 25 percent if it is slower. This produces generally slower accelerations, and faster decelerations, which is quite accurate, given that real drivers don't leave enough space in front of them to brake lightly.

In future additions to this program, an acceleration model might be implemented to make different types of vehicles accelerate according to their type. Semi-trucks accelerate at a much different rate than passenger cars do, so adding unique acceleration models to the various types of vehicles would add to the believability of the simulated traffic.

4 – GRAPHICS

There are many components in the generation of the resulting graphics. These include the geometry, motion, color, surface characteristics, and lighting of all of the objects in the scene. To implement the graphics, we used the OpenGL API, which is an open standard that is widely used.

The geometry of an object is defined by a set of vertices that are connected in a manner defined by the associated connectivity that the developer specifies, such as triangles, quads, triangle strips, etc. [OpenGL2006]. The motion of this geometry is controlled by a set of transformations – translate, rotate, and scale.

The color of the geometry can be defined either per-vertex or by mapping a texture to the geometry. In this project, the color is defined per-vertex for the vehicles, and an image is mapped for the road and ground geometries. Additionally, the lighting, which has ambient, diffuse, and specular properties defined, interacts with the per-vertex definitions, illuminating the features of the cars. Without the lighting, the details of the cars would be lost, and the cars would be flat colored silhouettes.

GLUT is an OpenGL Utility Toolkit that simplifies coding a windowed interface [GLUT2006]. Doing so using the native windowing libraries requires writing substantially more complex, and lengthy, code, which is quite error prone.

4.1 – Scene characteristics

After much experimentation with the positioning of the light source, it was made stationary above the track, at a height of about a thousand times the height of one of the cars, as is shown in Figure 4.1.

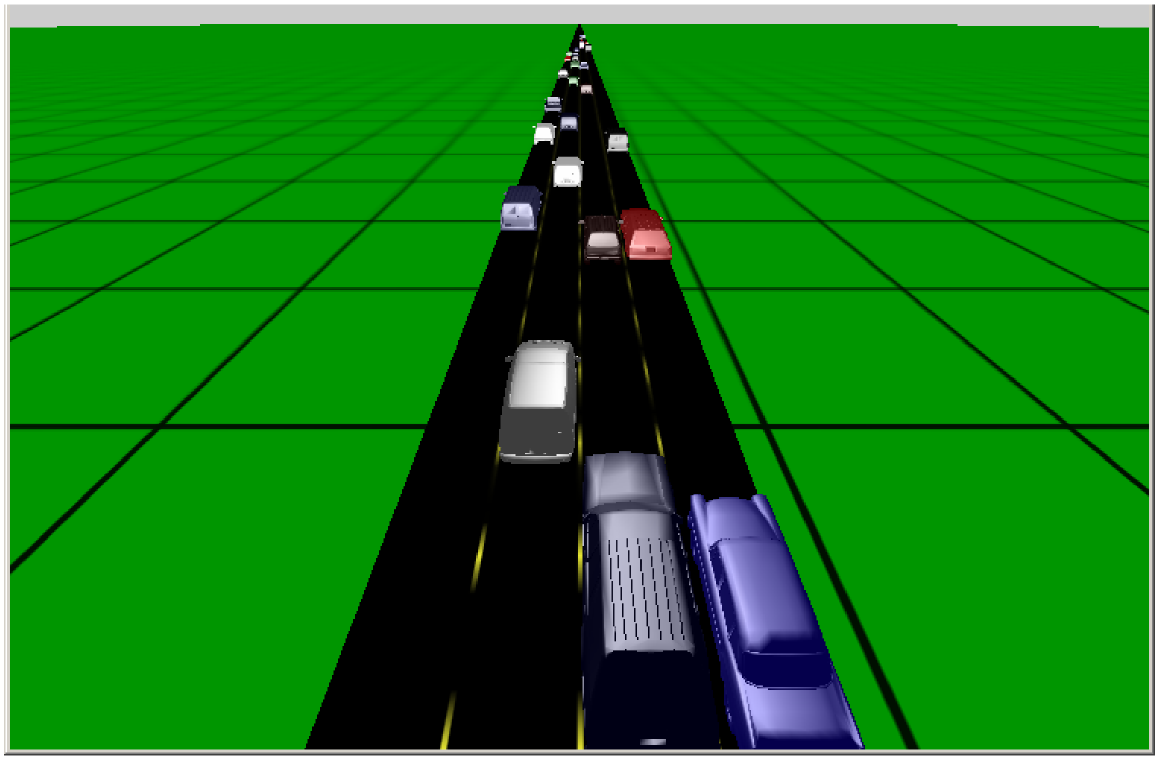


Figure 4.1: Final Light Source Location High Above Scene

Initially, the light source had been designed to move with the cars, but it made the scene look unrealistic. The specular highlights on the cars did not act in a way that a directional light such as the sun would. The next location tested was a fixed point above the track. This location was twenty or so times the height of a vehicle above the track. The result of this location was a set of specular highlights on the cars that made the light appear to be on the horizon. Figure 4.2, below, shows this phenomenon through a series of screen shots. The light appears to be above the car in the left image, and by the right image, it appears to be down lower, behind the car.

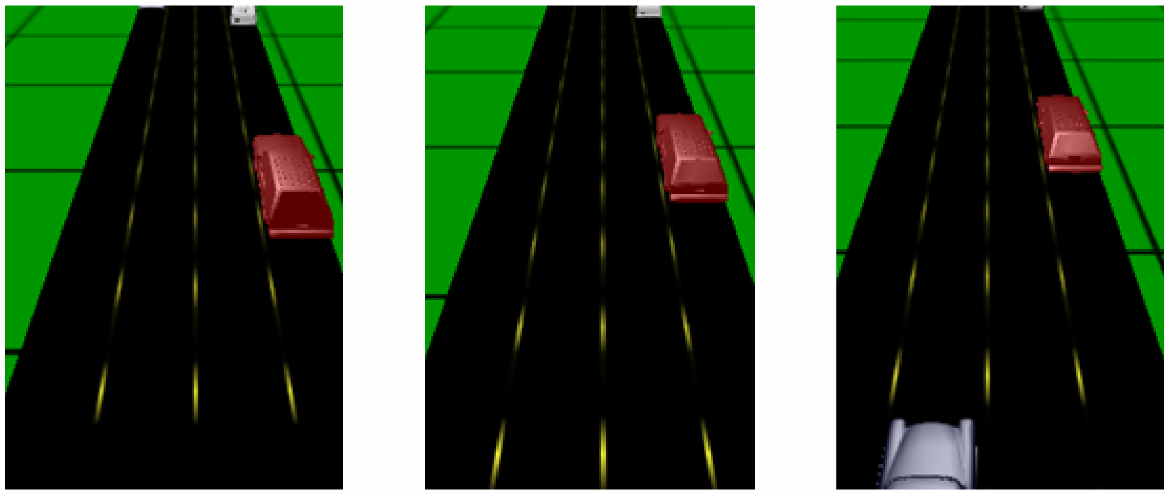


Figure 4.2: Light Source Location Moving with Group

While the sunset effect was not what we were looking for, it was noted as a potential feature to include in future revisions. The light source location could be interpolated between the starting location and one much closer to the track, using the passage of time to determine how quickly to do so. This would give the illusion of time passing, which could improve realism in a long-running simulation.

4.2 – Car physical characteristics

For each of the cars, models from the free collection on the Design Modélisation Image [ChezAlice2006] website were used. The specific models included in this program were common vehicles that are not unlikely to be encountered on any given highway at any point in time, though any model in the .obj file format can be just as easily included.

The colors of the cars are stored in a two dimensional array, in which the first dimension is indexed using a car's unique identification number, and the second holds the red, green, and blue components, indexed 0 through 2. By setting the color in the display lists, it saves us from having to change the states during runtime.

Depending on the type of the car, the scaling factor was different for each. This scaling to make all of the vehicles proportionately accurate to one another, was also done in the initialization of the display lists, and utilized the car type identification stored in each car.

4.3 – Track physical characteristics

To give a better sense of motion, speed, and field of reference, as shown in Figure 4.3, a set of broken yellow stripes and a simple grid pattern were used on the ground. The textures were mip-mapped, and a trilinear anisotropic filtering was applied to remove the texture shimmering in the distance. Without these, the vehicles appear to remain relatively stationary, some occasionally seeming to move backward. This phenomenon occurs as a result of the camera moving at the rate of the average velocity of all of the cars, which makes those vehicles that are traveling at a slower speed than the average appear as though they are moving in reverse, and those that are traveling just at the average appear as though they are not moving at all.

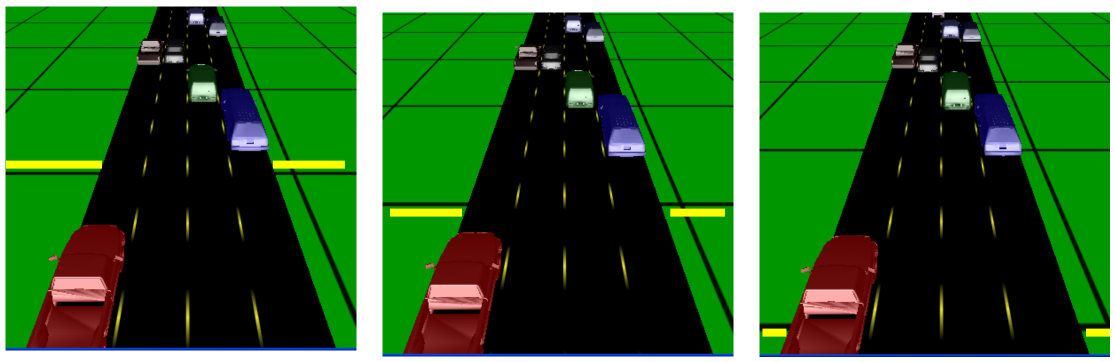


Figure 4.3: Field of Reference of Forward Progress

5 – USER INTERACTION

User interaction, through a pop-up control panel for a given car, was invaluable for debugging and calibration purposes, as well as for scene manipulation. The user interface window contains the camera transformation controls, allowing the user to move the camera forward or backward along the track, and up and down in the height direction. The standard reset, quit, and pause controls are also included in this window. To generate this window, I used the user interface library GLUI [Rademacher2006], to create a graphical user interface. This library provides controls such as buttons, translations, and checkboxes.

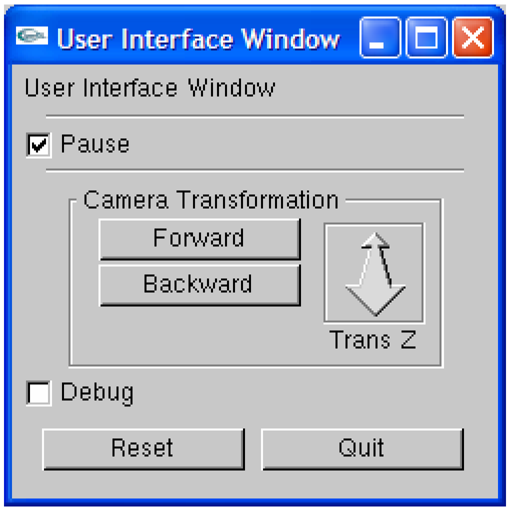
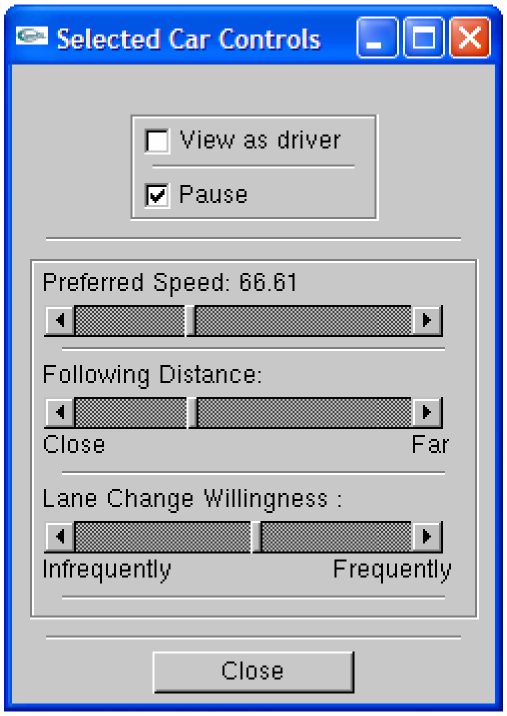


Figure 5.1: User Interface Window

Picking is one way to get information back from OpenGL. In the picking mode, the scene is rendered without drawing anything to the screen. Next, when the scene is drawn, each of the objects that we want to be able to pick is assigned an identifier. When a click occurs, OpenGL returns a list of objects that occupy the space within a certain tolerance of the cursor's location, along with information about the object's z-dimension [Shreiner2005]. The developer is then responsible for deciding how to use that information. In this case, the car with the smallest z-distance in the pick buffer is selected.

Through picking, the user can manipulate a vehicle's preferred speed, preferred following distance, and willingness to change lanes. When the user clicks on a vehicle, that vehicle is picked and a control panel with sliders and checkboxes pops up.



☐ View as driver
☑ Pause

Preferred Speed: 66.61

Following Distance:
Close Far

Lane Change Willingness :
Infrequently Frequently

Figure 5.2: Individual Car Controls

Each slider has a callback function attached to it that updates the property it represents, on the fly, causing the car to instantly take the new value into its calculations. One checkbox in the car control window toggles viewing the scene from the driver's seat of the car and looking down from the sky at the road. The other checkbox toggles between pausing the simulation and running it.

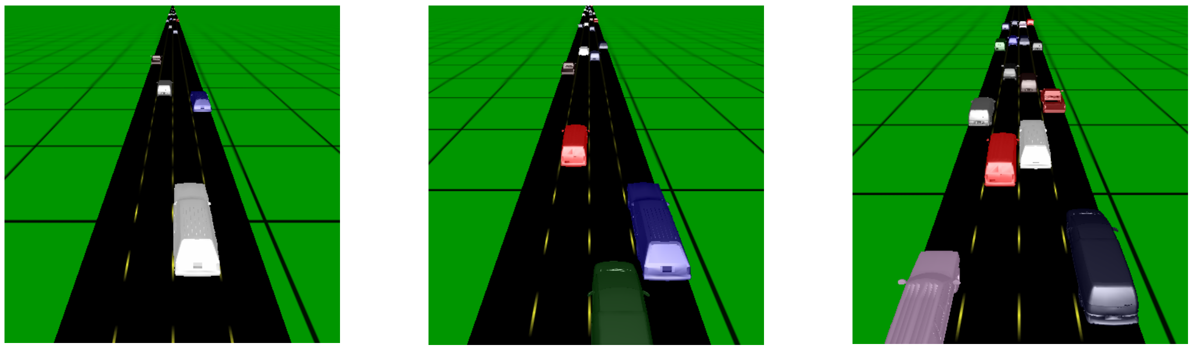
By default, the "camera" is located in the sky looking down at the cars on the road. As the program runs and the cars progress forward, so does the camera, which moves at the average speed of all of the cars. This keeps the camera towards the middle of the pack, since a fixed speed either gets ahead of or behind the pack of cars. When the focus is on the main graphics window, the up and down arrow keys move the camera position forward and backward, respectively. This allows the user to interactively control where the camera is along the road. These same controls can be manipulated using the forward and backward buttons in the user interface window, which also allows the user to move the camera up or down above the track.

6 – RESULTS

The value used for the safe_stopping_distance variable has great influence in the overall behavior of the vehicles in this simulation. As shown in Figure 6.1, when a true safe stopping distance was used, the cars were too far spread out to look believable. Even at half of a real safe stopping distance, the traffic still appeared too far spaced. The final stopping distance, which we found to produce the most believable traffic, ended up being only two to four car lengths long.

Lane changing also ended up looking quite believable. The cars make smooth lane changes, at different rates of acceleration. Some cars that have more aggressive characteristics speed up to change lanes, and those that are more inclined to leave a lot of space when following tend to slow down and change lanes more cautiously. Figure 6.2 shows a red van that has aggressive personality traits, speeding up, passing the gray car to the left, and changing lanes.

Figure 6.1: Stopping Distance Examples



Safe Stopping Distance

Half of Safe Distance

Closest-to-Personally-Observed Distance

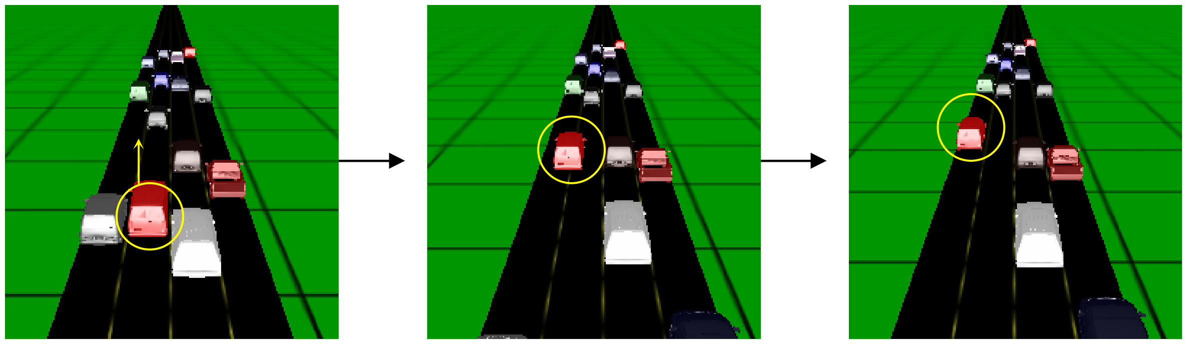


Figure 6.2: Lane Change Example

To get a realistic appearance, we need to be able to display a large number of cars at once. To do so, we need to be aware of where the bottleneck lies in this program. Operations performed per pixel consist of the OpenGL fixed function lighting, which means that they are simple, making them an unlikely candidate for a bottleneck. Additionally, the viewing angle does not allow for much opaque overdraw, reducing the probability that fragment processing is the bottleneck. Figure 6.3 shows that the performance drops with the increasing number of cars. The likely reason for this is the high polygon count of the models.

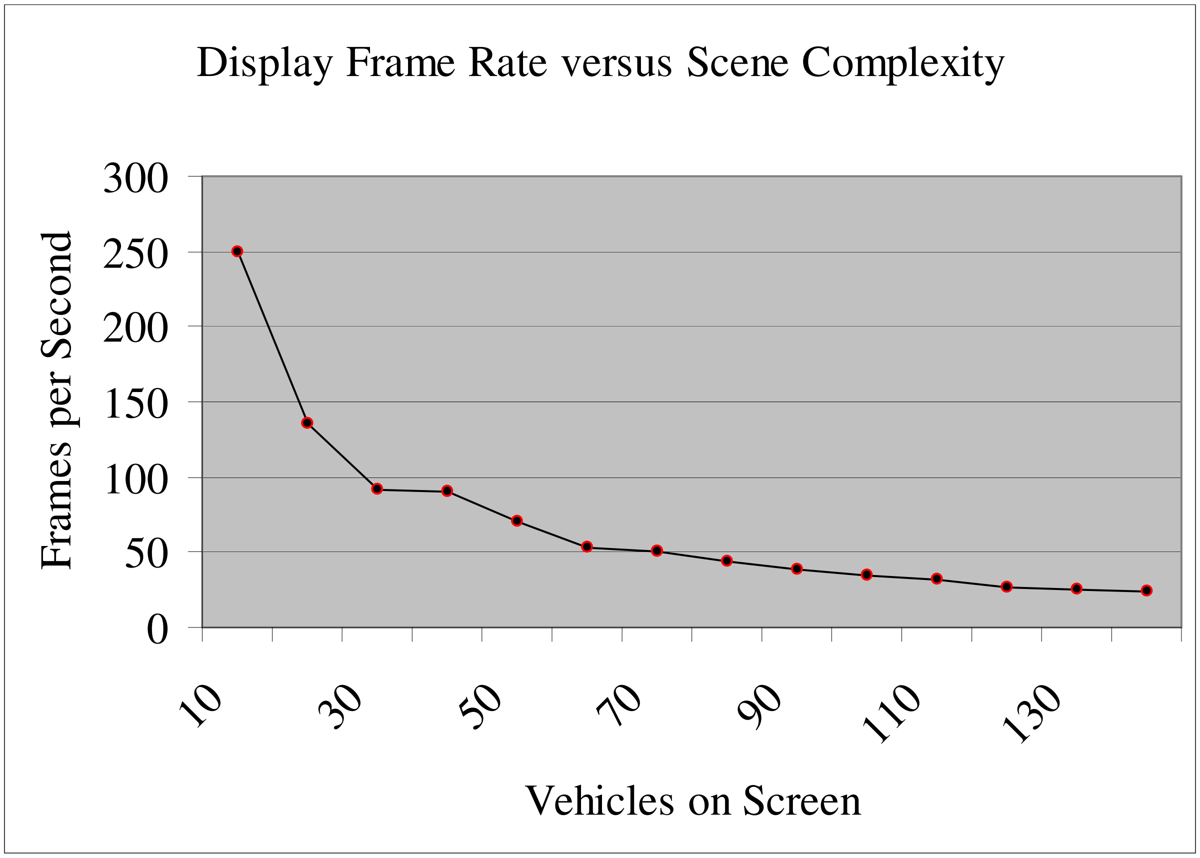


Figure 6.3: Performance with Varying Scene Complexities

7 – CONCLUSIONS AND FUTURE WORK

The traffic generated by this program meets the goal of the project. Qualitatively, the vehicles behave in plausible ways. Traffic flows quite nicely in some stretches of the road, and gets rather congested in others. There is variance in how closely cars follow and how fast they drive. Some vehicles accelerate quickly, others gradually. Different cars change lanes differently as well, some speeding up while changing, some slowing down, and some maintaining a constant speed.

An interesting conclusion that resulted from this project is the fact that to attain the most realistic-looking traffic, the drivers frequently push the rules of the road. This is to be expected, since very few, if any, drivers actually drive right at the speed limit. Even fewer drivers leave enough room in front of them for a safe stopping distance, regardless of the driving conditions. The space required for a safe stop grows with the square of a car's velocity [Oregon2006], which translates into huge distances at highway speeds. For a car traveling at 60 miles per hour, a safe stopping distance is 180 feet. The traffic simulation behaved more like real traffic when a two to five car length gap was left in front of cars, usually around 90 feet.

While the simulation is believable, one improvement that could be made would be adding another personality parameter for proper lane use. Currently all of the vehicles are just as content sitting in the left hand lane as in the right hand lane, so long as they are not being slowed down by another vehicle. The proper-lane-usage variable would be used in the function for lane change checking, to decide which lane to use, and would be used in conjunction with the speed of vehicles behind it.

Based on the results of the experiments calibrating other personality traits, such a parameter would likely be less true-to-life than might be expected. It would seem that perhaps when randomly generating this number, the average to use should be a 0.4 on a scale of 0.0 to 1.0, with a relatively large standard deviation. This would put a good percentage of the population of vehicles in a position where they are oblivious to the notion that the right lanes are for slower traffic and the left lanes are for faster, which seems to be the case more frequently than not.

As a result of working on this project, I have gained a more thorough understanding of the graphics pipeline. I have gained better mastery of implementing lighting, manipulating surface materials, and using display lists to improve performance. The lighting and surface materials are what give the sense of depth and warmth.

It was enlightening to see real human traits reflected in the simulated vehicles. While it is commonly understood that there are many impatient, aggressive drivers on our roads, it did not seem to me that most drivers on the roads were fairly so in real life. In order to produce believable behaviors, it was surprising that such a large portion of the vehicles on the virtual tracks needed to have aggressive and impatient traits.

Through my observations on real roads and experimentation with the virtual tracks, I have been convinced that real-world drivers do not leave enough space in front of them to make a safe emergency stop. Lucky for them, such stops do not seem to need to be made too often, especially since many people appear to only leave enough room to safely stop while driving 30 miles per hour. They need nearly twice that distance when driving at highway speeds [Oregon2006].

The realistic background traffic that we generated using a set of unique cars containing unique personality traits was quite successful. The process through which the program was developed taught me a great deal, both about the graphics pipeline and about calibrating parameters to produce believable behaviors. There are many directions that this project could continue into, adding cross-streets, onramps, or adding additional personality characteristics.

8 – REFERENCES

[ChezAlice2006] "Car 3-D Models." Design Modélisation Image. 3 Apr. 2006 <http://dmi.chez-alice.fr/models0.html>.

[GLUT2006] "GLUT – The OpenGL Utility Toolkit." OpenGL. Silicon Graphics, Inc. 26 May. 2006 <http://opengl.org/resources/libraries/glut/>.

[OpenGL2006] "OpenGL API Documentation Overview." OpenGL. Silicon Graphics, Inc. 4 Jun. 2006 <http://www.opengl.org/documentation/>.

[Oregon2006] "Stopping/Breaking Distance Chart." Oregon State Police: Patrol Division. Oregon.gov. 4 Jun. 2006 <http://www.oregon.gov/OSP/PATROL/stop_brake_distance_chart.shtml>.

[Rademacher2006] Rademacher, Paul. "GLUI User Interface Library." GLUI User Interface Library, v1. 4 Jun. 2006. <http://www.cs.unc.edu/~rademach/glui/>.

[Reynolds1987] C.W. Reynolds. "Flocks, Herds, and Schools: A Distributed Behavioral Model", *Computer Graphics* (Proceedings of SIGGRAPH 1987), Vol. *21, No. 4*, pp. 25-34.

[Shreiner2005] Shreiner, Dave, et al. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 2.* Addison-Wesley Professional, 2005.