Vehicle Traffic Simulation on a full scale 2D model City and road network

**ABSTRACT**

Computer graphics has recently become steadily more applicable for use in technological devices and entertainment. Along with that development, virtual crowd simulations have become increasingly popular and necessary to successfully create realistic movies, video games, and training simulations. An important component of designing a realistic population simulator is to incorporate automobiles, traffic, road networks, and the associated rules which allow those vehicles to properly interact with pedestrians and other vehicles. Many traffic simulators currently exist independently of a crowd simulator with human agents. The goal of this project is to design a traffic simulation system that can be at later stage if time permits could be incorporated into an existing crowd simulation system, full scale 3D model city, and road network. Currently in this project we are working only the traffic simulation system which is on a full scale 2D model city and road network. By building a traffic simulator, the simulation environments gain an additional level of detail and realism, and are more likely to be incorporated into future devices that require accurate and intelligent automated backgrounds.

Traffic simulation has become an important development, helping city planners design roads that do not congest traffic or cause accidents. In the development of the simulation, we had to make sure that all of the cars correctly obey the laws of physics, such as acceleration and braking. Every car has some personality traits mimicking those of real drivers, such as how aggressive the car is, how attentive the car is to its surroundings, and how law-abiding the car is to traffic rules. Simulation of these individualized agents is commonly referred to as a microscopic model of traffic. On the opposite end of the spectrum is the macroscopic model of simulation. It focuses on a large-scale system where traffic can be simulated in real time. The model correctly handles lane changes, merges, and also changes in driving behavior due to changes in speed limit. This method demonstrates how a macroscopic model can be applied so that animating a great number of vehicles in a large-scale traffic network runs in an interactive rate. All of the cars in the simulation are described by a single computational cell that dictates each car’s actions.

The traffic is done by creating roadmaps from the virtual city data and each car follows the same behavioral pattern with some randomness. The simulation is designed to handle large numbers of vehicles at once in a realistic manner without accounting for more intricate details. The cars all react to situations in the same way, with no differentiation between any two cars in the simulation. Possible expansion on our simulation would be to add the implementation of different vehicles. Examples would be the difference between buses that follow a set route and pick up pedestrians and private cars that drive through the city and eventually park.

As the simulation of traffic becomes more complex, processing speed becomes an issue. To counter this issue, we need to work on the traffic simulation that uses parallel multi-processing. The use of a macroscopic continuum traffic model which is based on traffic density, volume, and speed to simulate freeway traffic, which is very useful in applications such as highway safety and driver guidance in highly dense traffic. Because the system maps the simulation to a parallel computer architecture, it is capable of simulating highway traffic flow in real time. The use of parallel multi-processing greatly increases the simulation speed.

**SYSTEM MODEL**

This designed simulator reconstructs the movements of the car while minimizing lane changes, keeping a safe distance from other cars, and smoothing the overall traffic. The whole simulation works in real time, allowing the user to see virtualized traffic of the real world. This designed traffic simulation system includes vehicle dynamics, direct interaction with a road network, and incorporation of traffic rules. The overall benefit of designing this system will be to add a necessary component of detail and realism to an already well established, functional, and realistic human simulator. The combined simulation can then be used as a highly realistic backdrop for any type of virtual world. The primary function of the system is to take as input a road network, generate dynamic vehicles based on that network data, and display the simulation in an accurate way.

Once the road network is received by the simulation, the system generates an internal data structure to store the road data. Simulated vehicles are then created specifically mapped to exact locations obtained from the road data structure. When the simulation begins, predetermined algorithms determine the actions taken by each vehicle along the roads they have been assigned to. Finally, the positions of each vehicle are smoothly animated in a designated display, which includes making turns, changing lanes according to the priority algorithm, accelerating and decelerating to account for additional cars and avoid collisions, and stopping at traffic lights and also accounting the minimum time and distance required to reach the destination by the use of the shortest path algorithm.

The most important piece of the system is the simulation codebase. The simulation code has at its core a road network data structure, which monitors every vehicle’s current position on the streets. There is also a global list of all vehicles currently being simulated. Each vehicle stores its currents position, direction, shortest path graph, current lane and velocity. The simulation can be updated, a process which, when executed, moves all of the vehicles one step forward in time. After the simulation connects to a display, the display can easily access both the road network data structure and the list of vehicles. From these lists, the current location coordinates in the data matrix can be retrieved for each street and vehicle, which can subsequently be used to draw the objects on the screen. When a display update occurs, a call is made to the simulation to update all of the vehicle positions. This update process takes into account for each vehicle whether it is turning, stopped at a traffic light, or moving among other vehicles, and then calculates the appropriate new location based on those factors. Once all of the vehicles have been updated by this process, the display can simply iterate over the vehicle list and redraw them in their new positions. Because the simulation update is called whenever the display updates, the system animates smoothly.

The initial step is to retrieve the road network data from the data matrix sent using networking and the corresponding data structure is then constructed. This data structure is then provided to the simulation codebase, where vehicles are instantiated and generated. Because vehicles are generated directly on specific roads in the network data structure, vehicle positions are computed relative to the roads they are traveling on. The vehicle position data is then sent to the display, and the vehicles are drawn. Next, the system enters an animation loop where the display sends requests to the simulation to update the positions of all vehicles. After the update, the display iterates over the list of vehicles and redraws them at their current positions. This process continues until the simulation is closed. The system is currently implemented only in 2D environment and can be implemented in 3D environment using the Unity 3D game Engine, which requires a huge amount of time and effort.

**SYSTEM IMPLEMENTATION**

The traffic simulation system code is written in Python. The 2D system display is generated by using the simple python draw functions.

* **SIMULATION**

The key components of designing the simulation codebase can be divided into generating the road network data structure, creating the vehicles, controlling vehicle movement, implementing and controlling traffic lights, and preventing collisions.

1. ROAD NETWORK

Each pair of vertices that define a street is placed inside a street object. Every street object contains references to all of the streets connected to each of its two vertices. Streets are also broken down further into two edge objects, which represent the directions that a vehicle could travel on that street, three for each “lane”. One edge will travel from V0 to V1 while the other edge will travel from V1 to V0. So basically, each street is divided into six lanes from which three are used to travel from V0 toV1 while the other three are used to travel from V1 to V0. This implementation was used so that vehicles could be traveling specifically on one lane of an edge, instead of a street in general. This design allows for effective replication of two-way streets with different lanes, prevents vehicles from overlapping one another in the simulation, simplifies checking for collisions, and reduces the amount of searching done to discover which vehicles are close in proximity.

In its current state the simulation code does not account for any highways or one-way streets; every road in the simulation is a simple two-way street with 3 lanes of each. Generating random highways throughout the road network would not be realistic, and would also have a high-performance cost, so the design decision was made to make every road function identically at this time.

1. VEHICLES

After the street network is formed, vehicle objects are created and assigned to edges based on the data of the video feed which gives the information about the entry of the vehicles in the city using simple Digital Video Processing, along with their respective streets. Because the road network is implemented as a list of streets, each containing two edges, the street can be chosen by selecting an arbitrary list index. Every new vehicle is added to a global vehicle list. Each vehicle also contains a reference to the edge and street as well as the lane that it is currently driving on, and vice versa. In this way, there does not need to exist a global map of which streets contain which vehicles, which would require an immense amount of memory to store, and an infeasible amount of time to search through and update. After a vehicle is designated to a particular street, it is given a direction as well as the destination co-ordinate which is all retrieved from an array which is sent using networking from the output of the simple Digital Video Processing file. A small offset in the direction perpendicular to the edge is used in order to separate vehicles on the same street and give the illusion of vehicles driving in distinct lanes which in our case depicts 3 lanes in a single edge. This also allows all vehicles to properly obey the rule of driving on one side of a road. Because these computations take place whenever a vehicle is instantiated on a given street, at its moment of its creation it immediately has a calculated position and direction. Base speed is provided for all vehicles by a global time step constant, retrieved from the user interface or game engine, which can be also be modified on a per vehicle basis to simulate acceleration and deceleration.

1. MOVEMENT

Every vehicle has a current position, direction (given by the edge it is currently associated with), and speed. When an update call is made by the display function, the vehicle’s updated position in the given direction is computed as a modifier of its speed and a time step constant. If the new position remains within the vertex boundaries of the current edge, the position is updated and returned. However, if the computed position is a location beyond the edge’s end vertex, the vehicle must choose to move on to a new street. In this way, the vehicles’ positions are always interpolations between the two street vertices, and they never leave the confines of the street edges. To simplify this procedure, as soon as a vehicle is placed on a street it checks which possible streets it could move to after overstepping its current street and selects one based on the shortest path weighted matrix which is based on the distance weightage and a special weightage of number of vehicles on each street. Because of this, whenever a vehicle’s new position is seen to extend beyond the current street, the remaining distance is calculated and that distance is simply transferred to the new edge and traveled in that new direction. Once the vehicle switches to a new edge, its direction is also automatically updated to reflect the orientation of the new edge’s vertices. Moreover, the vehicles can switch the lanes among the street based on the traffic ahead of them and the traffic lighting as well as the path weighted matrix.

1. TRAFFIC LIGHTS

After the street data structure is created, traffic lights are generated for the road network at each of the junction where there are more than two streets joining at a single vertex. The complexity of creating the traffic lights for the intersections is taken care of by introducing different traffic light signals for each of the lanes on a single street. So, if there is an intersection of 4 streets at a junction then there are altogether 24 lanes where vehicles can travel out of which 12 are the incoming lanes and the other 12 are the outgoing lanes. So, we have to implement 12 different traffic signals at this intersection, one for each of the incoming lane. At each of the lane’s vertices, a traffic light is created located at that vertex’s location. The light is then appended to a global traffic light list. Each traffic light data structure contains its position, two light signals, and three edge lists corresponding to the lanes to reach after obeying to the light’s signals. Each light signal is either red, indicating that vehicles should stop, or green, indicating that vehicles should proceed through the intersection.

The traffic light signals are given a green light based on the number of vehicles in each lane of a street. In this way, the lane with most number of vehicles are allowed with higher priority to move and turn and lanes with less number of vehicles are asked to stop and wait for their opportunity. At an intersection, all 12 traffic signals for 12 lanes with combinations of 36 traffic signals for different directions work depending on each other so that simultaneously two or more lanes can avail the green traffic light based on which street and direction they want to go or turn into.

Each edge that ends in a four-way intersection contains a reference to a traffic light. The vehicles traveling along an edge check to see if a traffic light exists at the end of the edge. When they reach the end of the edge, the light signal for the edge is checked. If the light signal is green, then the vehicle’s motion path is unaffected and it continues as normal. However, if the light is red, the vehicle’s path is blocked at the end of the edge by the light signal. Therefore, the vehicle reacts accordingly and decelerates as it approaches the intersection.

1. COLLISION PREVENTION

A key component of the vehicle simulation is the prevention of collisions. The simplest collisions are prevented between vehicles moving along the same street edge. Each vehicle has a certain amount of space in front of it “reserved”, generally computed to be proportional to the vehicle’s current speed. When two vehicles are traveling along the same edge where the following vehicle is moving faster than the front vehicle, the former will smoothly decelerate until it reaches a location where the front vehicle does not enter its reserved space o it may continue with the same acceleration and can overtake the vehicle ahead of it by switching lanes based on checking whether the side lane is empty or not. In this way, a pre-determined constant following distance is always enforced between two vehicles through their reserved space or one vehicle could overtake another vehicle based on whether the other lanes are free to move upon.

When a vehicle approaches the end of an edge, it checks to see if any vehicles are already present on the street it will move on to after leaving the current street. This check is simple because every street contains a list of the vehicles currently on it. The vehicle seeking to move on to the street compares its position with that of the vehicle closest to it on the next street, if one exists. If the distance between them is greater than the reserved space amount, then it continues movement as normal. If, however, the distance is less than the amount of reserved space, the vehicle waits and does not move forwards until another update is called and the distance between them is increased. In this way, the constant amount of reserved space is still preserved between two vehicles during turns and collisions are avoided.

Preventing collisions at intersections between vehicles traveling in different directions is a related but more complicated problem. Each vehicle has its designated amount of reserved space as described above. When a vehicle is moving along a road and reaches a point where its reserved space is greater than the remaining distance to be traveled on that edge, the edge that the vehicle will travel to next is flagged as occupied. This flag persists until the vehicle crosses onto that edge. With this flag implementation, whenever a vehicle approaches an intersection it checks the flag of the edge it will travel to next. If the edge is flagged, it is an indication that another vehicle is approaching and going to turn onto that street first. The vehicle stops before the crossing and waits for the flag to disappear, which indicates that no more vehicles are approaching, before continuing movement by checking reserved space as indicated earlier.

But we have implemented the traffic signal in such a way in this simulator that it would automatically prevent collisions during turning at intersections. Traffic lights control vehicles for each direction of a given street by restricting movement to only one side of the traffic light at any given time. In this way, collisions between vehicles located at a four-way intersection are impossible when combined with the edge flagging system described above. One factor that is taken into account with the addition of traffic lights to the system along with the edge flags is the potential negative effect of creating a mini-deadlock. It is possible that one vehicle can obtain a flag for the next edge it wants to move to, but will then be stopped at a red light. Since the flag is taken, although unused, other vehicles that have a green light and want to move to the same edge are unable to move because it remains flagged. This causes a temporary lock on some vehicles for one traffic light cycle until the vehicle with the edge flag receives a green signal from the traffic light, continues its travel, and ultimately releases the edge flag. This problem can be resolved by changing the edge flagging system so that vehicles will release their flags when stopped at red lights, which will allow other vehicles to continue moving uninhibited. The edge flagging system and the traffic lights can now work simultaneously and synergistically to prevent collisions at four-way intersections. Even on intersections that are not four-way, the edge flag implementation can prove effective for preventing collisions.

* **2D DISPLAY**

1. ROADS

Each pair of vertices that define a street is placed inside a street object. Every street object contains references to all of the streets connected to each of its two vertices. Streets are also broken down further into two edge objects, which represent the directions that a vehicle could travel on that street, three for each “lane”. One edge will travel from V0 to V1 while the other edge will travel from V1 to V0. So basically, each street is divided into six lanes from which three are used to travel from V0 toV1 while the other three are used to travel from V1 to V0. This implementation was used so that vehicles could be traveling specifically on one lane of an edge, instead of a street in general. This design allows for effective replication of two-way streets with different lanes, prevents vehicles from overlapping one another in the simulation, simplifies checking for collisions, and reduces the amount of searching done to discover which vehicles are close in proximity.

In its current state the simulation code does not account for any highways or one-way streets; every road in the simulation is a simple two-way street with 3 lanes of each. Generating random highways throughout the road network would not be realistic, and would also have a high-performance cost, so the design decision was made to make every road function identically at this time.

1. VEHICLES

Vehicles are drawn in the 2D display as simple cubes scaled to fit the road network dimensions. In the display, the vehicles are BLUE & GREEN boxes with green fronts. Whenever the display sends a call to the simulation to update the vehicle positions, it also makes a call to redraw the display. To update the positions of the vehicles each iteration, the display makes a call to retrieve the list of all vehicles from the simulation. The display then iterates over the entire list, calling the Draw () method on each vehicle. Because each vehicle contains the computations internally for where and how to draw itself, the process is extremely simplified and efficient. Vehicles also orient themselves at an angle specific to the street edge they are currently traveling on. This was accomplished by taking the dot product of the street edge vector and the X axis to compute an angle of rotation, which is then applied to every vehicle before it is drawn.

1. TURNING ANIMATIONS

When any vehicle approaches an intersection with another street it enters into a special turning animation, separate from the normal functions that update location. The angle between the two streets is calculated, and from that angle a turning circle is defined. The vehicles then travel along the arc of the circle in order to turn smoothly. In order to compute and execute this turn properly, many steps must be taken. First, the intersection between the vehicle’s current and next edge, not street, is calculated. The distance from the start point of the turn and the end point of the turn to this intersection point is a fixed constant. Once the intersection point is found, the turn start and end points can be easily computed by using this distance. The lines from the start and end points to the edge intersection point form vectors that are both tangent to the turning circle. Computing these vectors is simple, and once obtained it is trivial to find the parallel vector in the direction of the turning circle’s center. At this point, the intersection point between these two vectors is found, which is exactly the center of the circle. By then calculating the distance from the start and end positions to the center, the circle’s radius r is discovered. Now that the turning circle radius r is computed, the incremental position steps around the arc of the circle for the turn can be calculated. The total turn angle θ is found by taking the dot product of the street edges. The amount of the angle travelled per time increment is a function of the vehicle’s current speed, the overall time step, and the radius of the circle. The new X and the new Y positions can be determined from some definite formulas. The position of each turning vehicle will be updated in this manner until the current angle is greater than the total angle θ. At this point, the vehicle’s current edge and street are set appropriately, all other values are reset, and movement along the next edge continues normally.

1. TRAFFIC LIGHTS

Still Not Implemented….