Virtual Urban Traffic Network Simulator

by

Jason Uh

S.B. Computer Science and Engineering
Massachusetts Institute of Technology, 2010

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
JUN 2 1 2011
LIBRARIES
ARCHIVES

Submitted to the Department of Electrical Engineering
and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2011
[June 2011]

© Massachusetts Institute of Technology 2011. All rights reserved.

Author .....
Department of Electrical Engineering
and Computer Science
May 20, 2011

Certified by .....
Professor Emilio Frazzoli
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .....
Ketan Savla
Research Scientist
Thesis Supervisor

Accepted by .....
Dr. Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

Virtual Urban Traffic Network Simulator

by

Jason Uh

Submitted to the Department of Electrical Engineering
and Computer Science
on May 20, 2011, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In this project, I designed and implemented a virtual urban traffic network simulator. The simulator serves as a testbed for human-subject experiments to determine driver behavior in road networks and also as a platform for testing route-planning algorithms. The simulator was implemented using the C4 game engine and OpenGL. The simulator is capable of producing both 3- and 2-dimensional visualizations of a traffic network. In this thesis, I describe the key components of the simulator, the necessary inputs, and the expected outputs. I verify operation of the simulator through observation of the actual system outputs.

Thesis Supervisor: Professor Emilio Frazzoli
Title: Associate Professor of Aeronautics and Astronautics

Thesis Supervisor: Ketan Savla
Title: Research Scientist

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Purpose and Objectives

Unexpected disturbances in traffic networks present a significant challenge in routing for the drivers navigating the roads. These disturbances could include a number of different phenomena, not limited to vehicular accidents, natural disasters, or even terrorist attacks. In any event, such disturbances that block paths in a traffic network call for some improvisation on the part of the drivers, the traffic network agents. That is, assuming each driver had an origin and intended destination, the challenge for the driver lies in finding a new efficient route to the destination.

This thesis project is a part of an interdisciplinary effort involving the study of algorithms, networking, and urban planning. It is the aim of the overall study to develop mechanisms for the autonomous reconfiguration of cyber-physical systems in the event of unexpected disruptions [3]. In the context of urban traffic networks, the scope of this thesis project, a cyber-physical system would entail a central server maintaining global knowledge of the all drivers' movements and positions, a way to selectively communicate that information to the drivers, and a way to influence the decisions of the drivers as they navigate the network. In this implementation, the Testbed employs a toll-based incentive mechanism for experiments. The server charges the driver a toll based on the congestion level on each road, thereby influencing the decisions of human agents. This mechanism is explained in greater detail in [9].

Because such a system cannot yet be physically tested in the real world, in this project we develop a simulator that we call the Virtual City Testbed. The Testbed can be used to model traffic network agent behavior and to serve as the platform for testing new network reconfiguration and traffic re-routing schemes. The Testbed serves two purposes:

1. *To allow human experiments.* The Testbed can be used as a platform for running human experiments that help us to understand how human drivers actually respond to unexpected disturbances and to different incentive schemes designed to influence their route-planning behavior. Data from these experiments can later be used to program more realistic AI traffic agents.

2. *To allow virtual simulations.* With or without human agents, the Testbed provides a platform for simulating drivers' route-planning behavior in networks. Simulations based on various combinations of incentive schemes and the physical parameters of the network can help us to better understand the influence of the cyber-physical system's server.

The Virtual City Testbed simulates automobile traffic flow. Traffic networks are represented visually as a 3-dimensional city road network. The Testbed accommodates both computer- and human user-controlled agents. This makes the Testbed a platform capable of not only user experiments, but also the deployment of traffic routing protocols in the virtual network.

1.2 Previous Research

There exist a number of virtual traffic simulators and traffic management systems. There has been work to develop so-called advanced traveler information systems (ATIS) or advanced traffic management systems (ATMS) to lessen congestion in road networks [2] and traffic simulators based on these systems. Such simulators include MITSIM (MIcroscopic Traffic SIMulator) [8], TRANSIMS (TRansportation ANaly-

sis SIMulation System) [5], and TransCAD [1]. Each of these systems simulates and forecasts route planning in a traffic network.

These systems have key shortcomings, however. Foremost, none of these systems allow for real-time simulation of traffic flow. In addition, they also assume driver behavior that reduces the flexibility of the simulations. In order to address these limitations, Kochhar and Kozhushnyan designed a prototype testbed as described in [3] and [4]. This project further advances their efforts with a more complete design and implementation. We keep in mind that the ultimate goal of the overall study of which this project is a part is to develop protocols for reconfigurable cyber-physical systems. To this end, this thesis addresses the shortcomings in previous traffic simulation projects for a system that gives three critical advantages over existing systems:

1. *Human agents.* The system allows a combination of computer- and human-controlled agents in the traffic network. Enabling human user-controlled vehicles makes possible the study of human decision-making and route-planning, which in turn can contribute to an algorithm to model human route-planning.

2. *Real-time introduction of disruptions.* The system allows the introduction of disruptions into the road network to simulate agent adaptation in the presence of unplanned disruptions.

3. *3-dimensional graphical user-interface.* The GUI presents the traffic network as a 3D city. This creates a more realistic experience for the user, in turn allowing more realistic models of human behavior.

1.3 Thesis Organization

In this thesis, we present the design of the Virtual City Testbed. In Chapter 2, we describe the internal architecture of the Testbed, including the various simulated traffic network elements. We also present two implementations of the simulator: one designed for human experiments and another intended to be a more lightweight, fully automated standalone simulator. In Chapter 3, we describe the simulator interface in

terms of the inputs of the system. In Chapter 4, we describe the expected outputs of the Testbed and validate its operation through inspection of the actual outputs. In Chapter 5, we highlight areas for future development and summarize the contributions of this project.

Chapter 2

Internal Architecture of the Simulator

2.1 System Overview

The overall Testbed is the composite of many individual simulated components. At the highest level of the simulator architecture is the `TrafficWorld`, which represents the simulated environment as a whole. The `TrafficWorld` has a `RoadSystem` that models the entire traffic network of `Road` and `Intersection` objects. The `RoadSystem` in turn is populated with `Car` objects, which are the basic traffic network agents in our implementation; in other words, `Car` objects represent the drivers in a road network and their paths represent their decisions. Each `Car` is equipped with a `CarController` that makes the path-taking decisions for the `Car`. `TrafficWorld` also has `GlobalPathInfo`, a data structure to represent the `RoadSystem` as a graph. Figure 2-1 presents a high-level picture of this system. Section 2.2 of this chapter outlines each of the critical components of the simulator in greater detail.

We implemented the simulator for use in two different forms. The first is a 3-dimensional implementation running on the C4 game engine, as presented in Section 2.3. This implementation employs a client-server model, modeling human-controlled agents as clients and the simulated world and road network as the server. The second implementation, further described in section 2.4, is a 2-dimensional simulation

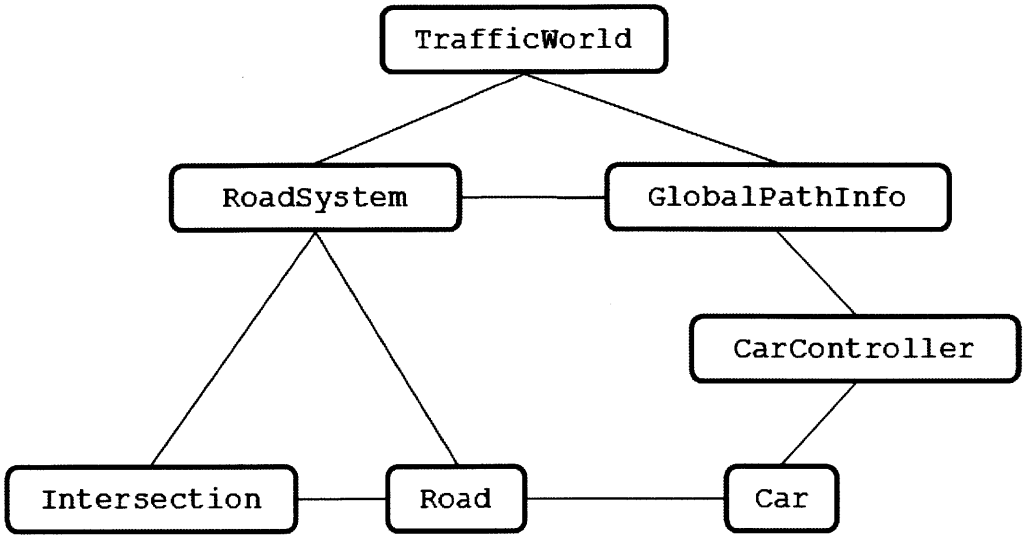

Figure 2-1: A high-level diagram of the key simulator components.

running on OpenGL. This is implementation is meant to be used for simulations in which all agents are AI-controlled.

2.2 System Components

This section describes the critical components that comprise the Testbed system. The following subsections serve not to provide the details of the source code, but rather to describe the high-level interactions between each of these components.

2.2.1 Traffic Game and World

`TrafficGame` and `TrafficWorld` are the highest level components of the Testbed. The `TrafficGame` is responsible for maintaining an instance of the `TrafficWorld` and communicating to it the events that take place during a simulation. The `TrafficGame` handles events including human agent connections, user inputs, and the introduction of disturbances in the form of explosions. The `TrafficGame` also manages the synchronization between server and clients in the experiment-oriented 3-dimensional implementation of the simulation, further presented in Section 2.3.

The `TrafficWorld` is the topmost simulated element within the system. It main-

tains the RoadSystem and populates it with agents in the form of Car objects. At every time step of the simulation, the TrafficWorld will update its state. This world-wide update is accomplished through a call to update that is propagated to all of the simulated elements. That is, upon the world's update, every Car in the network will compute its new position and speed, which in turn gives the RoadSystem a new state of its own.

While the purpose of the TrafficGame is to instantiate the world and to handle the mechanics of a user interface, the TrafficWorld effectively serves as the manager for all of the properties corresponding to the traffic network and its agents. Additional responsibilities of the TrafficWorld include selecting Cars for assignment to human users and maintaining GlobalPathInfo, the graph data structure of the road network.

2.2.2 Road System

The RoadSystem is a model for a physical city. It defines the interface used for constructing a virtual traffic network. Once constructed, a RoadSystem consists of Road and Intersection objects. A Road is defined in turn by the Intersections at its endpoints. Furthermore, as in the real world, a Road is composed of one or more Lanes, each of which has directionality, so that a Road may be either a one-way or two-way street.

As a simulated element, each Road has a live state. At a given time step, a Road has access to theCars in its lanes, which allows live computation of the average speed of all cars on a road. The average speed can then be used as a metric for the level of congestion on a road. Roads also have speed limits that are enforced on its cars.

In order to accommodate disruptions in the road network, each Road has a flag denoted isBlocked, which is a property that indicates whether or not cars can drive on the road. When a disruption takes place, the road is flagged as blocked and is consequently made unavailable for traversal.

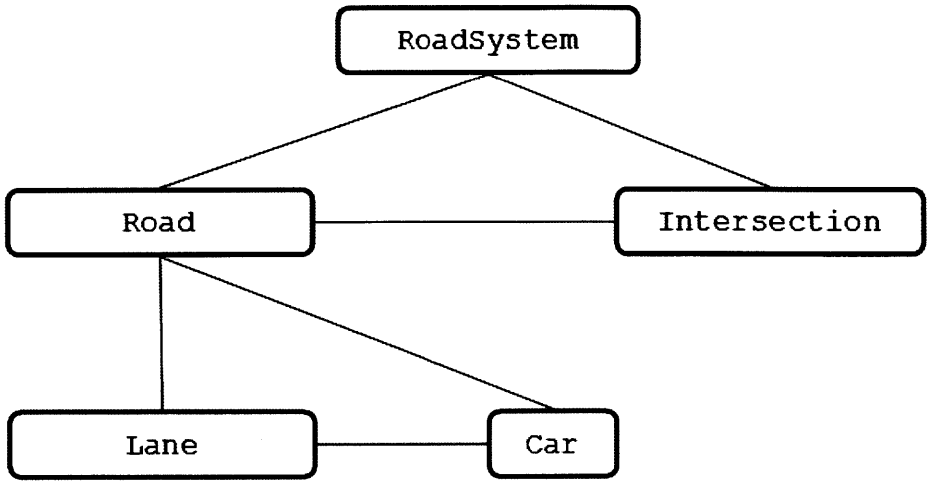

Figure 2-2: The components of a `RoadSystem`.

2.2.3 Cars

Each `Car` object represents an agent navigating through the road network. The state of each car at a given moment is defined by the car's speed, direction, and physical coordinates. Also, the `Car` has knowledge of the road that it is on. `Cars` adhere to the physical constraints of the roads and also respect their speed limits.

2.2.4 Car Controller

The `CarController` is the brain behind each `Car` object. Every `Car` is instantiated in the simulation with a `CarController`. The controller is responsible for the path-taking operations of the car.

Foremost, the `CarController` for every AI-controlled vehicle stores a data structure called the `ODPair`, which represents the car's origin-destination pair. All intersections in the road network qualify as candidate origins and destinations for the cars. An `ODPair` therefore contains identifiers for the origin and destination intersections for the car. Using the graph-based data structure of the road network in `GlobalPathInfo` (described in Section 2.2.5), based on its car's origin and destination, a `CarController` will find a path to the destination.

The second role of the `CarController` is to direct its car at every intersection.

When a car approaches an intersection, it queries its `CarController` for the next road to take. The `CarController` refers to the pre-computed path and informs the car of the next road to turn onto. In this way, the `CarController` guides the car from start to finish.

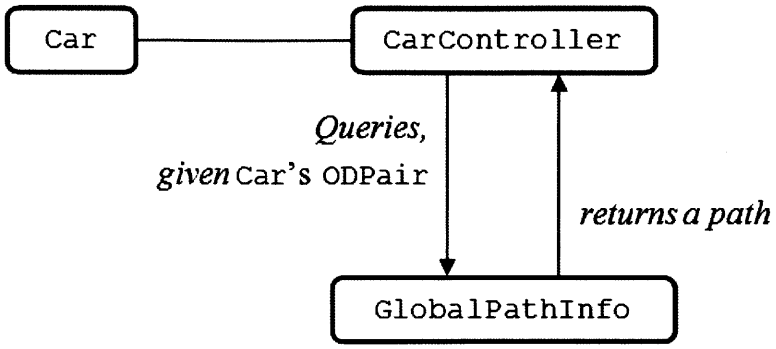

Figure 2-3: The `Car` and `CarController`.

2.2.5 Global Path Information

`GlobalPathInfo` is an object accessible to the `TrafficWorld` that contains a number of data structures that are important to the cars' route-planning.

The `GlobalPathInfo` contains `trafficGraph`, a graph equivalent of the road network represented by `RoadSystem`. Naturally, the edges of the graph are the roads in the network and the nodes are the intersections. When a simulation begins by creating a new instance of `TrafficGame`, `TrafficWorld` is constructed by loading a map of the road network to be rendered. During this load, the network graph is constructed one road at a time. Also, while each road is added to the graph, a separate data structure, a hash map mapping pairs of intersections to roads, is simultaneously constructed. Specifically, in order to refer to the roads in the network by the two intersections at their respective endpoints, each pair of endpoint intersections is mapped to the road that they define.

The `GlobalPathInfo` also computes the routes for all computer-controlled cars. After the construction of the traffic graph and the intersections-to-road hash map, the `Cars` in the network and their `CarControllers` are instantiated, each with an origin

and destination pair, as described in Section 2.2.4. `GlobalPathInfo` will proceed to compute the path for each origin-destination pair. The paths are returned as a sequence of intersections, as a sequence of nodes in a graph specifies a unique path. These paths are saved in another hash map that maps origin-destination pairs to their corresponding paths, allowing later reuse of this information if necessary. Each `CarController` then retrieves the path corresponding to its cars origin-destination pair. Based on the retrieved path, the `CarController` can proceed to lead the car to its destination.

2.2.6 Disturbances

The ability to introduce unexpected disturbances into the network is one of the important features in the design of the simulator. In the Testbed, a disruption is implemented as a blockage in the road network. The location of the disruption is specified with coordinates within the `TrafficWorld`. Visually, a disruption is represented as a small burning explosion on a road surface.

2.2.7 Logger

The `TrafficGame` has a `Logger` object that is responsible for recording live simulation data. We require the logging of live simulation data for a number of reasons. First, it allows later study of the results of the simulations and experiments. Future analysis of the results of human experiments is necessary to model realistic human drivers' behavior. Also, during the experiments themselves, the user is presented with a map of the road network displaying congestion levels in the roads and the corresponding tolls to drive in each road. We call this map the Virtual Positioning System, or VPS. The VPS, whose design is explained in further detail in [9], is implemented as a web application. To feed the application the live data from the Testbed simulation, the `Logger` writes data to a MySQL database, which is then read in real time by the VPS.

At each update of the `TrafficWorld`, the `Logger` is called on every car and road in

the simulation. In a MySQL database, the `Logger` writes the position and speed of the human-controlled car as well as the average speed of the cars on every road. The VPS is then able to use these data to provide a real-time display of the changing congestion levels and corresponding toll changes in the road network. We intentionally do not write all properties of all roads and cars to the MySQL database, as the number and frequency of the database transactions would overload and decrease performance of the Testbed backend. Therefore, as an alternative, all properties relevant to future study but not necessary for the VPS are written to text files that can later be parsed to extract the data. These data include cars' speeds, directions, and coordinates and also the number of cars and average speed on each road.

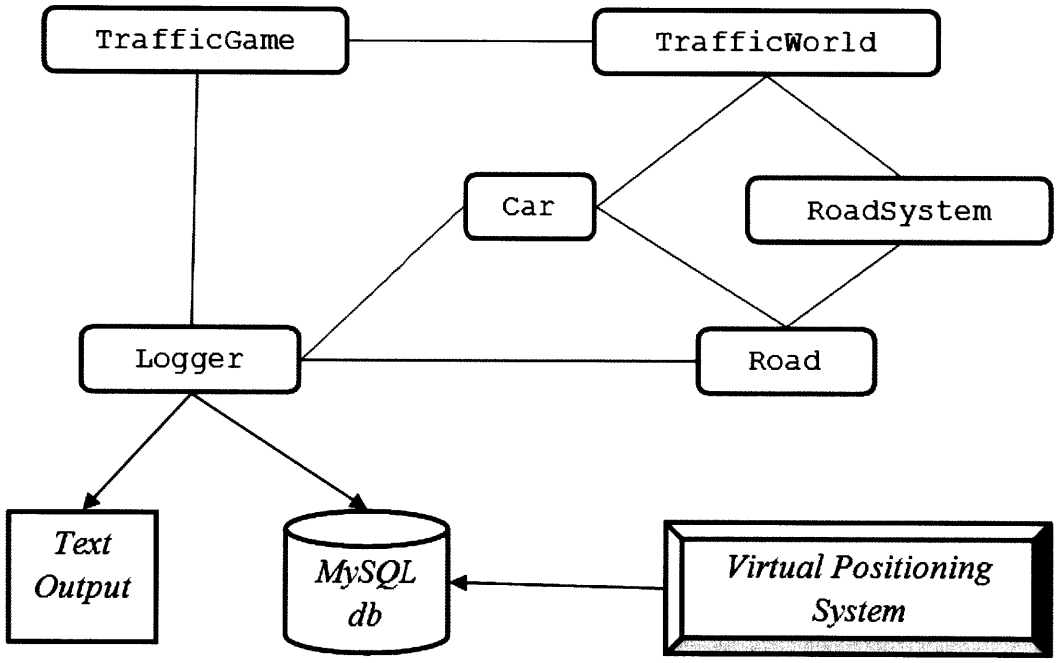

Figure 2-4: The `Logger` and the Virtual Positioning System

2.3 Experiment-Oriented Implementation of Simulator

The first implementation of the simulator is intended for human experiments and is presented as a 3-dimensional graphical interface. The C4 game engine is used for the

graphical rendering.

2.3.1 C4 Game Engine

Building a 3-dimensional graphical user interface for experiments was important for us, as it adds to the realism of the user's experience. That is, for a computer-based exercise, a 3-dimensional view of the car and its surroundings would most closely recreate the real-life driving experience for the user, in turn eliciting more realistic behavior.

To build this interface, we leveraged the C4 game engine from Terathon Software. The C4 engine provided us with support for user input, 3D graphics, physics, as well as networking [7]. In this implementation, C4 serves as the heart of the simulation by creating and running an instance of the `TrafficGame` and `TrafficWorld`. It is C4 that ultimately calls the TrafficWorld, and consequently all of the simulated elements, to update at each time step.

2.3.2 Client-Server Model

The C4 game engine provided a way to establish a network connection between multiple instances of a C4 game. This allowed us to design the simulation with a client-server model. That is, in this implementation, all global objects in the form of `TrafficGame`, `TrafficWorld`, `GlobalPathInfo`, and `RoadSystem` are maintained by a server, and each human driver is implemented as a client. The server and all of the clients are run on separate C4 instances.

In order to keep the server and the clients synchronized, we use a messaging scheme between the server and each client. The `TrafficGame` of the server will send and receive `WorldStateMessage`s to and from the clients, which contain the state of the cars in the network.

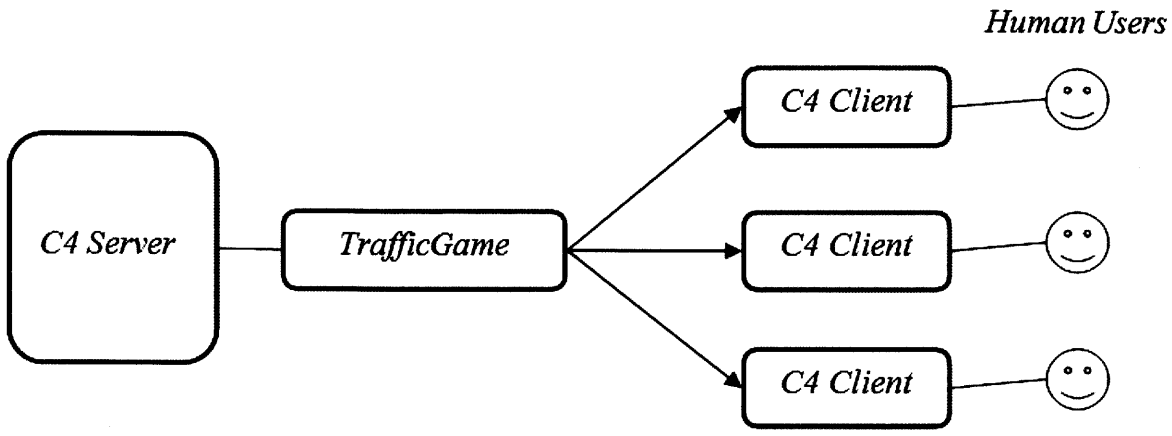

Figure 2-5: The interaction between the C4 server and clients.

2.4 Standalone Simulator Implementation

When running simulations that do not involve human input, the C4 implementation of Section 2.3 is cumbersome. The overhead of the unnecessary rendering of 3D graphics and client-server networking makes the C4 system overkill for simulations of all AI-controlled cars. To address this concern, we developed a 2-dimensional implementation of the simulator using OpenGL to render the visuals. This implementation of the Testbed removes the TrafficGame, which is not necessary for an all-computer agent simulation, and takes the place of the TrafficWorld, instantiating the `RoadSystem`, the `Cars`, and the `GlobalPathInfo`. This allows the simulation of the traffic with the same behavior as in the C4 version, without the extra computing overhead that the 3-dimensional implementation demands.

Chapter 3

Simulator Inputs

While Chapter 2 describes the internal architecture of the Testbed, users of the simulator need only to understand its interface in terms of the necessary inputs and the expected outputs. The following sections outline this interfaces inputs, and Chapter 4 explains the outputs.

3.1 OSM Map

In both the C4 and OpenGL implementations, the road network representations in `RoadSystem` and `GlobalPathInfo` are created at the start of the simulation. The road network is constructed from an `.osm` file, an XML file formatted to the schema of the OpenStreetMap project. These files can either be created from such OSM editors as the Java OSM (JOSM) application, or be downloaded from the OpenStreetMap website [6]. In this way, both user-created and real-life road networks can be simulated and rendered by the Testbed. Table 3.1 shows an example of three nodes from an actual `.osm` file mapping downtown Boston.

3.2 3D Map

For the C4 implementation, the road network is displayed in 3D. This 3D world must be rendered before the simulation is run. The C4 engine requires a `.wld` file, which

Table 3.1: Example OSM data

Nodes in Boston

```
<node id="61339937" lat="42.360354" lon="-71.067371" version="1"
changeset="64956" user="MassGIS Import" uid="15750"
visible="true" timestamp="2007-10-09T00:41:42Z">
    <tag k="created_by" v="JOSM"/>
    <tag k="source" v="massgis_import_v0.1_20071008193615"/>
    <tag k="attribution"
v="Office of Geographic and Environmental Information (MassGIS)"/>
  </node>
```

```
  <node id="61339940" lat="42.360346" lon="-71.067791" version="1"
changeset="64956" user="MassGIS Import" uid="15750"
visible="true" timestamp="2007-10-09T00:41:43Z">
    <tag k="created_by" v="JOSM"/>
    <tag k="attribution"
v="Office of Geographic and Environmental Information (MassGIS)"/>
    <tag k="source" v="massgis_import_v0.1_20071008193615"/>
  </node>
```

```
  <node id="61339946" lat="42.360381" lon="-71.066371" version="1"
changeset="64956" user="MassGIS Import" uid="15750"
visible="true" timestamp="2007-10-09T00:41:43Z">
    <tag k="created_by" v="JOSM"/>
    <tag k="source" v="massgis_import_v0.1_20071008193615"/>
    <tag k="attribution"
v="Office of Geographic and Environmental Information (MassGIS)"/>
  </node>
```

includes the 3D world and any other graphical widgets that are to be displayed at runtime. The graphics, including the road and car textures and any buildings that the user may wish to decorate the world with, can be prepared with the COLLADA software suite. Then, at runtime, C4 takes the prepared textures and 3D objects of

the `.wld` file and displays them live in animated form.

3.3 Simulator Parameters

We can tune various physical parameters of the system prior to simulation. In the source code of `TrafficWorld`, we can adjust the number of cars in the system. There, we can also set the cars' initial locations. The default behavior is to spawn the cars at random roads throughout the system to distribute them evenly in the network.

We can also set the origins and destinations of the cars. The default behavior with respect to origin-destination pairs is to set them randomly throughout the network, but the `ODPair` for any car may be programmatically set to any pair of intersections in the road system.

The speed limits of the roads may also be set programmatically.

3.4 Route-Planning Algorithm

The implementation of `GlobalPathInfo` will determine the route-planning behavior of the AI-controlled agents. In particular, it is the `GlobalPathInfo` object in the world that computes the paths for cars, given origin-destination pairs. This is accomplished through the `GlobalPathInfo`'s `computePaths()` method.

The algorithm used to compute the paths will determine the routes that the agents take. The default algorithm is an implementation of k-shortest paths based on Dijkstra's shortest path graph search algorithm. However, by altering the contents of `GlobalPathInfo`'s `computePaths()` method, we can modify the algorithm. We also note that, as currently implemented, the default behavior for a given cars origin-destination pair is to compute the k-shortest paths and then to select one of the k paths randomly. The motivation behind this design decision was to avoid congestion; if a certain origin-destination pair is particularly common (as may very well be in the real world), all cars with that `ODPair` would attempt to take the same shortest path if it were not for this randomization.

Also, by tuning `GlobalPathInfo`'s `loadGraph()` method, we can change the `trafficGraph` data structure representing the road network. As the graph is constructed during the execution of this method, the weights are assigned to the edges (i.e., roads) in the graph. The default behavior is to set the weight to the inverse of the average speed on a road, which effectively makes average speed the metric for measuring congestion in the road network. If this is not desirable, however, any other metric can be used for the weight. For example, we could set the weight of a road to the number of cars, its length, or even an estimated amount of time to traverse the road (which can be computed given the length and average speed).

Chapter 4

Outputs and Validation

Depending on whether we run the C4 or OpenGL implementation of the simulator, we will see different graphical outputs. Ultimately, the output that we obtain from the Testbed is the movement of the agents throughout the network, given the inputs discussed in Chapter 3: the road network topology, the simulation's initial conditions and physical parameters, and the path selection algorithm. However, in terms of the operation of the Testbed, we define the output to be not only the movement of the cars, but also the implementations' abilities to maintain and simulate the state of the road network.

It follows that checking the operation of the Testbed entails inspection of this output. The following sections describe the various forms of the output we expect from the Testbed and present the actual output from the implementation.

4.1 C4 Implementation's Server-side State

The server-side state of the C4 implementation should hold global information about the road network and all of its cars. In particular, the server should update the speed and position of all AI-controlled cars at each time step of the simulation. We are able to confirm this operation through the C4 engine's server instance. Figure 4-1 shows the C4 visual output for the server, demonstrating its maintenance of the global road network and all of its cars.

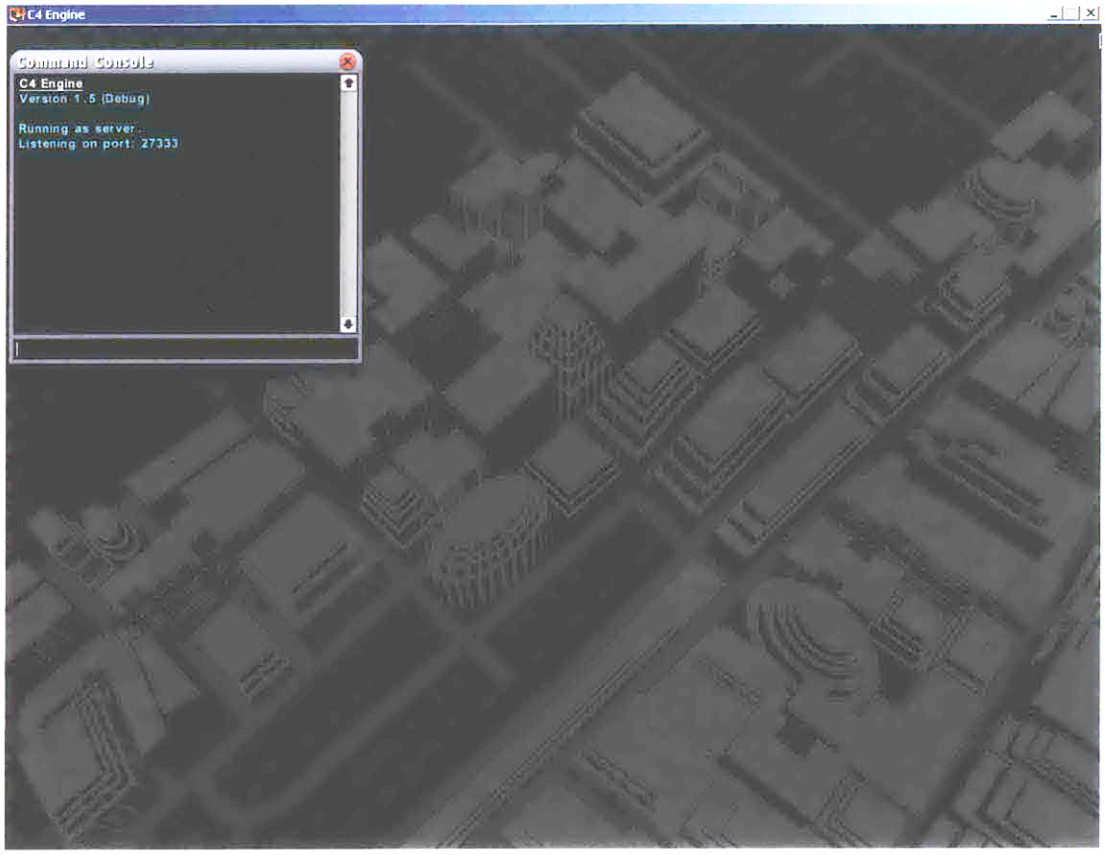

Figure 4-1: C4 server-side state.

4.2 C4 Implementation's Client-side Interface

While the C4 server should maintain the global information of the road system, the world's state as stored in the server should be propagated to the clients so that the state of the road network is kept consistent between the server and the clients. Additionally, the client interface should allow human control of a car in the network. We are able to confirm this operation through the C4 engine's client instance. Figure 4-2 shows the C4 visual output for a human-controlled client, showing the interface upon the user's assuming control of a car in the network. We note that the camera angle shifts to the perspective of the user-controlled car.

Figure 4-2: C4 client-side driving interface.

4.3 Virtual Positioning System

At any given time in during a simulation using the C4 implementation, the Virtual Positioning System should show the position of the user-controlled car and the congestion levels of the roads in the network. Specifically, the congestion levels of the roads should be reflected in tolls corresponding to the average speed of a road segment; that is, roads with low speeds - indicating high congestion - should charge a higher toll for access, in order to discourage more drivers from entering it and adding to the congestion. We are able to confirm this operation. Figure 4-3 shows the visual output from a browser for the VPS web application. The VPS is able to read data from the MySQL database and thus tracks the user-controlled car and also dynamically displays the toll levels for each road.

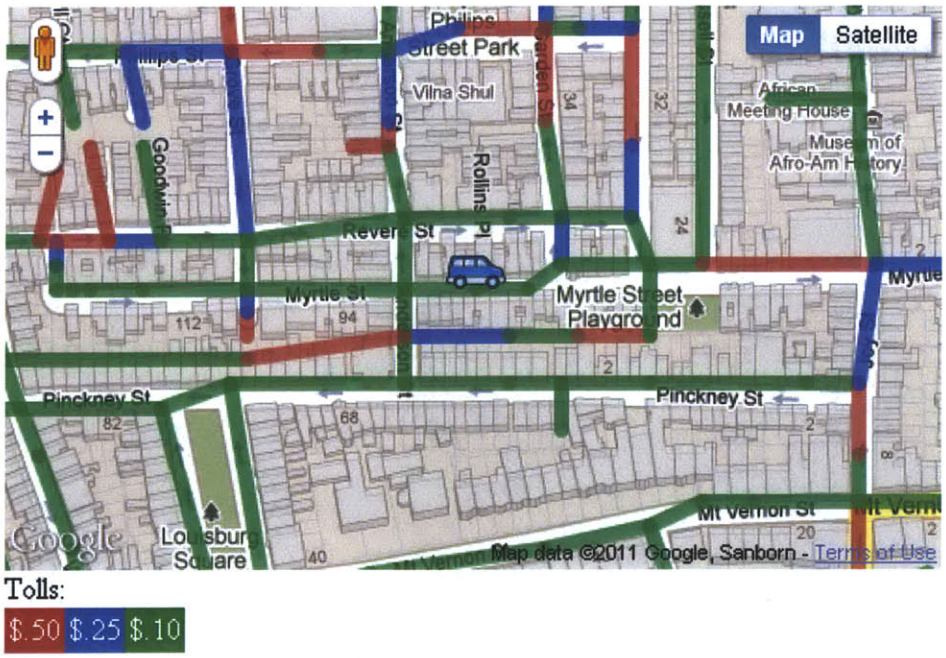

Figure 4-3: Virtual Positioning System.

4.4 OpenGL Implementation's 2-dimensional Output

The OpenGL implementation should simulate car behavior in the same way that the C4 implementation does, only without the overhead of networking and 3D-graphic processing. The 2-dimensional interface is especially an effective way to display the shortest paths computed for cars, as the simple graph depiction makes it easy to view the network as a whole. We are able to confirm this operation by running the OpenGL implementation. Figure 4-4 shows the shortest path computed and selected for a particular car. We are able to observe that the car does indeed follow the drawn path.

4.5 Traffic Logs

Sections 4.1 to 4.4 discuss the outputs of the simulator in terms of visual output. However, for future analysis, the logged states of the cars and roads are the most

Figure 4-4: OpenGL's 2-dimensional graph output of a selected path.

important. The `Logger` writes to the MySQL database what the VPS needs to make the visual output described in section 4.3. This includes the average speeds of cars on each road, as well as the coordinates of the user-controlled car. Also, for future study, the `Logger` writes parameters of all cars and all roads to a text file that can later be parsed to extract the data. We can confirm the operation of the `Logger` by inspecting both the MySQL database tables and the output text files. Table 4.1 shows sample entries from the MySQL table recording road data. Table 4.2 shows entries from the table recording the human-controlled car data; we observe `Car 0`'s movement over time.

Table 4.1: Sample entries from MySQL table recording road data

roadID	time	numCars	avgSpeed
0	2011-03-21 19:43:42	1	9
1	2011-03-21 19:43:42	1	27
2	2011-03-21 19:43:42	2	13.5
3	2011-03-21 19:43:42	1	22.5

Table 4.2: Sample entries from MySQL table recording car data

carID	roadID	time	utmX	utmY	speed
0	0	2011-05-17 21:58:01	329793	4691420	0
0	0	2011-05-17 21:58:03	329804	4691420	24.156
0	5	2011-05-17 21:58:07	329812	4691400	16.848
0	5	2011-05-17 21:58:08	329812	4691390	10.548

Chapter 5

Conclusion

5.1 Future Work

The natural step for the virtual urban traffic network simulation project is to conduct human experiments using the Testbed presented in this thesis. The experiment participants' path-taking decisions would be recorded and analyzed to gain a better understanding about human decision-making in traffic networks, particularly in the event of unexpected disruptions in the network. Running these experiments and gauging participants' feedback may also bring to attention areas for improvement in the user interface of the Testbed.

In addition to user testing, for the future development of the Testbed, we would like to experiment with the design of the Logger. In the current implementation, the `Logger` writes the data needed by the VPS to a MySQL database and data for all roads and cars in a text file. There are two alternative implementations of the logging mechanism that we could explore. First, the server could be split into two separate instances, one that is dedicated to directly serving the clients, and another for writing the data for all cars and roads to a MySQL database. The reason the current implementation writes this data to a text file is that writing global data to a MySQL requires too many transactions, drastically slowing down the performance of the server. If we pursue this distributed implementation, however, one server instance could be dedicated to logging, which would free up the parallel server instance for sim-

ply serving the clients, thereby minimizing the loss in performance. We would need to take care to make sure the two servers are synchronized using a world state messaging scheme, similar to that occurring between clients and the server, as described in Section 2.3.2.

A second alternative `Logger` design would change the nature of the data that are logged. The current implementation directly logs parameters of the road and car states - that is, coordinates, speeds, and speeds. However, we could choose to log instead the *decisions* of all the agents at each time step. If the `Logger` were to write each car's decisions - speed adjustments and which roads it chose at intersections - and the time that the car made those decisions, then theoretically, the entire simulation could be recreated from a log of those decisions alone. This implementation may reduce the amount of data to be logged, potentially obviating the need for a distributed server, and may also allow more accurate recreations of simulations.

5.2 Summary

We have presented the architecture and implementations of a virtual urban traffic network simulator. The simulator serves as a testbed on which we can conduct both human experiments and fully automated simulations. By design, the Testbed allows the tuning of various physical parameters of the road system and its initial conditions. We have also demonstrated the operation of system in terms of visual and numerical output corresponding to the cars movements. We have also implemented an extensible and customizable framework for the automated route-planning of cars in the traffic network.

Appendix A

Managing Virtual City Testbed Inputs

A.1 C4 Implementation

A.1.1 Server

The server of the C4 implementation resides in a directory called `c4_server/`. The following are the key files for the server:

- `c4_server/C4.exe`

  The executable for starting the C4 engine and server.

- `c4_server/TrafficGame.dll`

  This is the dynamic link library containing the implementation of the Testbed and all of the simulation's components.

- `c4_server/Data/Engine/variables.cfg`

  This is where the server's port for the simulation is defined. This is the port that the server will open up for client connections.

- `c4_server/VirtualCity/world/<somewhere>.wld`

  The contents of this file represents the 3D graphical representation of the road network.

By running `c4_server/C4.exe`, an instance of the server will begin.

A.1.2 Client

The client of the C4 implementation resides in a directory called `c4_client/`. The following are key files for the client:

- `c4_client/C4.exe`

  The executable for starting the C4 engine and client.

- `c4_client/TrafficGame.dll`

  This is the dynamic link library containing the implementation of the Testbed and all of the simulation's components. This should be a copy of the `TrafficGame.dll` file for the server.

- `c4_client/Data/Engine/variables.cfg`

  This is where the IP address of the server is defined.

- `c4_client/VirtualCity/world/<somewhere>.wld`

  The contents of this file represents the 3D graphical representation of the road network.

By running `c4_client/C4.exe`, an instance of the client will begin. By typing `join` in the client's Command Console, the client will be assigned a car in the road network.

A.2 OpenGL Implementation

To run the OpenGL Implementation, we start the `TrafficVis.exe` executable. This should open an OpenGL window and display a 2-dimensional simulation of the traffic network.

To edit this implementation, we simply tune the parameters in the `main.cpp` file of the TrafficVis C++ project.

A.3 Physical Road Network Parameters

A.3.1 Number of Cars

The number of cars is specified in the
`<solutionDirectory>/TrafficGame/traffic_world.h` file, where
`<solutionDirectory>` is the top-level directory of the C++ solution for the Testbed.
The number is defined by the variable `NUM_CARS`.

A.3.2 Road System

The road system is created by loading an `.osm` file whose path is specified to the `RoadSystem::createFromFile()` method. This method is called in the
`<solutionDirectory>/TrafficGame/traffic_world.cpp` file.

A.4 Route-Planning

We want to alter the `global_path_info.cpp` file to tweak the path-finding functionality of the Testbed. This file should be found in `<solutionDirectory>/TrafficLib/`.

A.4.1 Graph Edge Weights

To change the edge weight metric for the road system, we want to change how the graph is constructed. The graph is constructed in `GlobalPathInfo`'s `loadGraph()` method. This method calls an `addEdge()` on every road in the road system. The third argument of `addEdge()` (i.e., in `addEdge(firstArg, secondArg, thirdArg)`) is the value used to set the edge weight. In the current implementation, this is based on the average speed of the cars on the road. However, this may to set to anything. For example, it may be the number of cars on the road, the length of the road, or even a constant number to give all edges equal weight.

A.4.2 Path-Computing Algorithm

To edit the algorithm that finds a path between an origin and destination pair, we want to change the contents of `GlobalPathInfo`'s `computePaths()` method.

Bibliography

[1] Caliper Corporation. Transcad, 2010, `http://www.caliper.com/tcovu.htm`.

[2] A. Polydoropoulou H. Koutsopoulos and M. Ben-Akiva. Travel simulators for data collection on driver behavior. *Transportation Research*, pages 143–159, 1995.

[3] Amrik Kochhar. Simulation and verification of autonomous route planning behavior. Master's thesis, Massachusetts Institute of Technology, 2010.

[4] Oleg Kozhushnyan. Virtual city testbed. Master's thesis, Massachusetts Institute of Technology, 2010.

[5] D. Anson K. Nagel L. Smith, R. Beckman and M. Williams. Transims: Transportation analysis and simulation system. *Los Alamos National Lab*, 1995.

[6] OSM Project. Open street map, 2011, `http://www.openstreetmap.org/`.

[7] Terathon Software. C4, 2011, `http://www.terathon.com/c4engine/index.php`.

[8] Q. Yang. *A Simulation Laboratory for Evaluating Dynamic Trac Management Systems.* PhD thesis, Fanstord University, 1988.

[9] Boyuan Zhu. Traffic condition tracking in virtual city testbed. Master's thesis, Massachusetts Institute of Technology, 2011.