MODELING VIRTUAL ENVIRONMENTS FILLED
WITH AUTONOMOUS TRAFFIC

A dissertation presented

by

Zhishuai Yin

to
The Department of Mechanical and Industrial Engineering

In partial fulfillment of the requirements for the degree of
Doctor of Philosophy

in the field of
Interdisciplinary Engineering

Northeastern University
Boston, Massachusetts

October, 2012
**Table of Contents**

1 Introduction ............................................................................................................................................... 1

1.1 Introduction of Virtual Environments and Driving Simulations ......................................................... 1

1.2 Motivations .......................................................................................................................................... 4

1.3 Research Objectives and Research Problems ...................................................................................... 6

1.3.1 Research Objectives ....................................................................................................................... 6

1.3.2 Research Problems ............................................................................................................................ 7

1.4 Organization of the Thesis .................................................................................................................... 10

2 Related work .......................................................................................................................................... 12

2.1 3D Modeling of Virtual Environments ............................................................................................... 12

2.1.1 Road Network Modeling ................................................................................................................ 13

2.2 Representation of Virtual Environments ............................................................................................. 23

2.3 Driving Behaviors ............................................................................................................................... 29

2.3.1 Car-following Models .................................................................................................................... 29

2.3.2 Lane-Changing Models .................................................................................................................. 32

2.3.3 Intersection Behavior Models ....................................................................................................... 36

3 Geometric Modeling ............................................................................................................................. 38

3.1 Preparing the geographical dataset .................................................................................................... 39

3.2 Extracting 2-D road centerlines from aerial maps ............................................................................... 40

3.2.1 Image preprocessing ....................................................................................................................... 43

3.2.2 Roadways Thinning ......................................................................................................................... 46

3.2.3 Intersection Detection .................................................................................................................... 47

3.2.4 Roadway Segments Extraction ...................................................................................................... 49

3.2.5 Curve Fitting .................................................................................................................................... 50

3.3 Roadway Elevations .......................................................................................................................... 54

3.4 Converting 3-D road centerlines to 3-D geometric roadway models .................................................. 59

4 Topological Modeling ............................................................................................................................ 62
LIST OF FIGURES

Figure 2.1 A semi-automatic extraction scheme (from (Gruen & Li, 1997)) ........................................... 16
Figure 2.2 a-c. Extraction of salient roads. a Extraction of line. b Optimization of ribbons boundaries. c Selection of ribbon parts with constant width. Only the ribbon defiled by thick lines is considered as a salient road (from (Laptev, H., Lindeberg, Eckstein, Steger, & Baumgartner, 2000)) ........................................... 18
Figure 2.3a-d. Extraction of non-salient roads. a Ends of extracted salient roads. b Extraction of optimal path. c Verification by optimization of width. d Acceptance of a hypothesis with low variation of width (from (Laptev, H., Lindeberg, Eckstein, Steger, & Baumgartner, 2000)) ........................................... 18
Figure 2.4a-d. Extraction of crossings. a selection of possible crossings. b Approximation of the crossing outline. c Verification of connections to adjacent roads. d Construction and connection of the crossing (from (Laptev, H., Lindeberg, Eckstein, Steger, & Baumgartner, 2000)) ........................................... 19
Figure 2.5 Road extraction flow (from (Wiedemann & Wessel, 2003)) ........................................... 20
Figure 2.6 Flow of the filtering algorithm (from (Li & Briggs, 2009)) ........................................... 23
Figure 2.7 Representation of road networks in (a) conventional geospatial data and (b) as a road network graph (from (Srivastava, 2010)) ........................................... 24
Figure 2.8 Two-dimensional and Graph Representations of Road Network (from (Christian, Jan, B.P., & Timko, 2003)) ........................................... 25
Figure 2.9 A Road Geometry configuration example and its lowest topologic level (from (Donikian, 1997)) ........................................... 28
Figure 2.10 The hierarchical structure describing urban concepts (from (Donikian, 1997)) ........................................... 28
Figure 2.11 The lane-changing model structure (from (Ahmed, 1999)) ........................................... 34
Figure 3.1 Flow of extracting centerlines of roadways from an aerial Google Map image ........................................... 43
Figure 3.2 Original aerial Google Map Image ........................................... 43
Figure 3.3 Gray scale result of the original aerial Google Map image ........................................... 44
Figure 3.4 Binary Images of three types of roadways ........................................... 45
Figure 3.5 Thinned Images of three types of roadways ........................................... 47
Figure 3.6 Detected Intersections ........................................... 49
Figure 3.7 Extracted road segments ........................................... 50
Figure 3.8 Clothoid ........................................... 52
Figure 3.9 Clothoid fitting (from (McGrae & Singh, 2008)) ........................................... 53
Figure 3.10 Curve fitting result of the highway ........................................... 54
Figure 3.11 3-D plot of a geographic area in Belmont, MA ........................................... 55
Figure 4.28 Two curves that approximate the lane change swerve ................................................................. 88
Figure 4.29 endpoints of “V05” .................................................................................................................. 90
Figure 4.30 Start and End topological nodes of autonomous vehicle “A01” .................................................. 91
Figure 4.31 Move onto the next pair of endpoints .......................................................................................... 93
Figure 4.32 vehicle hits an endpoint ............................................................................................................. 94
Figure 5.1 Architecture of the driver behavior framework (from (Al-Shihabi & Mourant, 2003)).................. 97
Figure 5.2 Refer to vehicles on roadways as elements of 2D arrays .......................................................... 99
Figure 5.3 Operations on Arrays in a lane-change scenario ........................................................................ 101
Figure 5.4 The collider of v06 enters the collider of lane 1 ........................................................................ 102
Figure 5.5 A simulated vehicle and its attached transform ............................................................................ 103
Figure 5.6 Autonomous vehicle and obstacle represented as elements of Arrays ....................................... 104
Figure 5.7 Vehicles communicate with each other at an intersection ......................................................... 106
Figure 5.8 A following scenario .................................................................................................................. 109
Figure 5.9 Communication in a lane change scenario .................................................................................. 110
Figure 5.10 Communication in an intersection scenario ............................................................................. 111
Figure 5.11 Intersection Model .................................................................................................................. 113
Figure 5.12 An example of the decision-making tree at the global level and the local level ................. 116
ABSTRACT

A primary goal of modeling virtual environments for driving simulations is to reproduce the real world as completely as possible. Designers have to accurately represent the geometric information about real roadways and reproduce the real traffic as if autonomous agents were controlled by human beings. Another goal that is equally important is to build the virtual environment with efficiency. Both workloads and time consumed are expected to be minimized.

This thesis introduces improved methods to build large-scale driving simulations. The process of modeling the virtual environment is categorized into two levels: geometric and topological. Traditionally, most driving simulations model the virtual environment on the geometric level only. As a consequence, autonomous objects ( vehicles, pedestrians, etc.) are blind to their surroundings and therefore lack intelligence. Furthermore, designers have to manually guide all autonomous objects through the road network and hence the workload is increased tremendously.

At the geometric level, creating real-scene based road networks, which is a key contributor to the reality of driving simulations and an important factor in determining the resemblance of virtual driving scenarios to the real world, is still considered as a labor intensive task. A lot of challenges are posed by the requirement that designers have to obtain the accurate data of the geometric shape of roads and transform the data into 3D models in virtual environments. The data source, the approach utilized to extract the data, and the algorithm that processes the data so that it could be used by 3D modeling tools, are all issues to be addressed at the geometrical modeling level. This thesis proposes a new methodology for modeling road networks at the geometric level, with the purpose to reproduce real-world road networks with an appropriate level of accuracy as well as a low level of workload. After evaluating existing data sources and road network extraction approaches, coarse scale aerial images are selected as data sources of road networks on the 2D x-y plane. An image processing model is proposed to perform road categorization, road segmentation, and road segment centerline extraction on aerial images. The third component:
elevation, which cannot be extracted from aerial images, is obtained from another data source: GIS, a MASS public geospatial database.

At last, a curve fitting process is applied to approximate smooth curves to the extracted 3-D raw centerlines of road segments.

At the topological level, this thesis introduces topological structures to store all information about the virtual environment. Two inter-connected topological structures are proposed: 1) global topological structure and 2) local topological structure. Each component of the road network such as road segments, intersections, bridges, etc. is represented as a node in the global topological structure. Connectivity and traffic information is stored at the global level. For each global topological node, a local topological structure is associated. Each component of the global node is represented as a node in the local topological structure. Information about the local driving environment such as signs, obstacles, etc. is stored at the local level topological structure. The idea of having topological structures at different levels enables autonomous objects to plan and follow its global route in the road network with ability to react to its instantaneous local driving conditions.

The issue of implementing autonomous traffic is then addressed. A combination of a perception model, a communication model and a decision-making model is implemented to achieve the goal of modeling autonomous objects that behave intelligently. All models are attached to autonomous objects that represent people and vehicles in a virtual environment. This enables such an autonomous object to be an independent entity that is self-motivated and self-controlled. The perception model enables autonomous agents to perceive their instant driving situation. Autonomous agents communicate with other autonomous objects via their communication model and the decision-making model relies on the perception model and the communication model to investigate possible outcomes before making decisions. The communication model defines senders, receivers, contents, and channels (media through which content is transferred) in real-time to gather desired information from specified objects. The
decision-making model is divided into two levels, the global level and the local level. These work, respectively, with environmental information perceived by the perception model and traffic information received by the communications model. A group of logic rules are formulated as decision trees to model the process of making decisions on the basis of real-time activities.

A database is used to store both geometric and topological information about the virtual environment. All information is then loaded into in-memory arrays for real-time access and update. Triggers and Colliders, which are attached to all simulated objects, are adopted in this project to update dynamic arrays in all events. Also, triggers and colliders play key roles in designing repeatable driving scenarios.

This research models a geographical area in the metro Boston, MA area to demonstrate all techniques proposed in this thesis.
Acknowledgements

First of all, I would like to take this opportunity to express my deepest gratitude and appreciation to my advisor, Prof. Ronald Mourant, for his great supervision and patient guidance, from the date that I started applying to the PhD program in Interdisciplinary Engineering, all the way to completion of the degree, on study and research. His excellent mentorship and great personality had made my school life as a graduate student not only substantial, but also joyful. I am truly grateful to have had the opportunity to work him.

I must also thank my committee members, Prof. Yingzi Lin and Prof. Sagar Kamathi, for their wonderful instructions over the years, and their insightful suggestions on my thesis research. Both Prof.Lin and Prof.Kamathi had provided me gracious help and precious opportunities to explore a broad range of cutting-edge subjects in Industrial Engineering.

My thanks have to as well go to all my colleagues in the Virtual Environment Laboratory and all my friends throughout the years. They’ve not only offered me generous help in study and research, but also in my personal and career life. I could have not been enjoying my school life so much without them.

Finally, I dedicate this dissertation to my beloved parents, for their absolute selfless love through all these years, for their support whenever and wherever. I wouldn’t have achieved any of these without them.
1 Introduction

1.1 Introduction of Virtual Environments and Driving Simulations

Virtual environments, also referred to as “Virtual Reality”, “Artificial Reality”, are computer-simulated environments that can simulate physical presence in places both in the real-world and in imaginary worlds.

As defined by (Burdea & Coiffet, 1994), virtual reality (VR) is “a high-end user interface that involves real-time simulation and interaction through multiple sensorial channels (vision, sound, touch, smell, taste)”. Although some simulations include sensory information such as sound and tactile, most current VR environments are primarily visual experience. A high-fidelity visual environment plays a major role in creating a lifelike experience to users. Also as the same study describes, there are three “Is” of a VR: interaction, imagination & immersion. “A VR simulation is interactive so that users can navigate and interact in the simulation. Designers may use imagination to create both imaginary and realistic environments. And last but not least, the simulation should be immersive so that users are drawn into the virtual environment”.

Basically, VR systems consist of two key elements: 1) the VR engine and 2) I/O devices (Grigore, Paul, & Philippe, 1996). The VR engine is a combination of computer hardware and software. The hardware and software are combined to create the dynamic computer generated image which gets updated multiple times per second. The frequency at which the image gets updated is defined as frame rate. I/O devices are devices that users use to interact with the virtual world. A typical output device is the video display. Input devices include keyboard, mouse, joysticks, head-mounted equipment, haptic gloves, etc.

As Virtual Environments enable users to interact with a simulated world, it’s widely applied in areas such as: entertainment, design, manufacturing, health care, hazardous operations, and training, etc. When applied to transportation, the common professional application of Virtual Environments is driving simulation.
Driving simulations are becoming increasingly important tools for various uses. They are being used for driver training and proved to be excellent effective and practical tools. Utilizing driving simulations allow researchers to study and monitor driver performance, behaviors for cases where it will be unsafe or maybe illegal to place drivers in the real world. New driver assistance systems can also be tested with the help of driving simulators.

Existing driving simulators can range widely in capability and complexity. Simple driving simulators can be merely a desktop PC with a game steering set. To build immersive VRs, a lot of efforts have been made in improving I/O devices.

Since VRs are being widely used in training, experimenting, etc. a lot more input devices are developed in addition to standard input devices. Basically, these new input devices allow users to interact with the virtual environment in more ways. For example, haptic systems include tactile information so that users can actually feel and move 3D objects in virtual environments. It obviously helps enhancing the degree of reality of VRs.

Regarding output devices, instead of displaying images on a computer screen which provides a very unrealistic visual experience to users, displays that produces a large field of view that appears surround the user are becoming popular. Almost all highly sophisticated driving simulators nowadays employ 360 degree video displays. Some examples of highly sophisticated simulators are the National Advanced Driving Simulator located in Iowa and the Toyota Driving Simulator (Murano, Yonekawa, Aga, & Nagiri, 2009). They both have an actual car on a platform inside a six-axis dome which also serves as a 360-degree video screen.

These two sophisticated simulators both made great improvements on one major component of the VR system: I/O devices. The six-axis dome as a 360-degree stereoscopic display enhances the perception of depth and the sense of presence (Lin, Henry, Donald, Habib, & Furness, 2002). The actual car mounted
on the platform is an input device that provides users much more realistic driving experience as compared to game steering sets.

However, to construct an immersive and interactive driving simulator, efforts can be not only made on I/O devices, but also on the other major component of the VR system: the VR engine, which is a combination of computer hardware and software.

Recently, great technological advancements have been achieved in computer hardware fields, such as computer processing power, graphics. In the past decades, graphics cards have undergone a tremendous development. Take NVIDIA graphics cards as examples. At 1995, the first release, NVIDIA NV1 has a maximum memory of 4MB and a maximum bandwidth of 0.6GB/s. While at 2008, the released GeForce 9600GT has a memory of 512 MB and a maximum bandwidth of 57.6GB/s.

The great development of Graphics cards helped significantly in constructions of high-fidelity VRs. With graphics cards, designers were capable of developing more realistic images that are rendered with higher degrees of details.

Other than hardware, software is also indispensable to any VR. Without graphics content, the computer hardware has nothing to transfer to a signal that will be used by the display medium.

A great amount of works have been done in this area in the past decades as the power of the hardware grows. “In the 1980s, it was challenging to develop software that could function with a broad range of graphics hardware. Developers had to write custom interfaces and drivers for each piece of hardware. This was expensive and resulted in many duplicated efforts.” (OpenGL) . Graphics APIs became available in the 1990s as standard interfaces for developers to use to produce 2D or 3D computer graphics. These APIs hide complexities of working with computer hardware and provide standard specifications so that graphics applications can be shared and distributed easily. Some examples of Graphics APIs are OpenGL, Direct3D, Mesa 3D, etc. Most of them have been developing for more than 10 years. Take
OpenGL for example, over the past 20 years, a great number of features have been added to this API. Those features either made the API more convenient to use, or introduced new functions.

Made use of the existing graphics APIs, a few integrated development tools were later developed in order to further facilitate and improve the development of 3D virtual applications. It saves designers a lot of time and becomes the primary method of development. Tremendous amount of efforts can be saved with the implementation of Integrated Development Tools. For example, with OpenGL, a designer has to dedicate the exact steps required to create a scene. Usually, multiple lines of code are written to merely create a 3D cube in the virtual environment. With Unity 3D, an integrated development tool, however, designers would just have to make a few clicks.

1.2 Motivations

Developments in both computer hardware and software related fields have certainly increased the reality of virtual environments, and lowered the workloads of developers over time. However, in practice, it’s not possible for a developer to overcome technical limitations on processing power, communication bandwidth in order to create a more realistic virtual reality experience. It also requires works of a whole team to improve existing graphics APIs or integrated development tools.

Though, it doesn’t mean that developers play less important roles as compared to decades ago. Nowadays most current virtual environments are primarily visual experiences, visual environments generated by the software are essential to driving simulators. A driving simulator without computer-generated scenes displayed on displays is not usable. As introduced in 1.1, existing driving simulators range broadly in complexity. In practice, a great number of driving simulators fall somewhere between these two extremes and provide drivers adequate immersive and realistic driving experiences with affordable expenses (Mourant & Yin, A Turning Cabin Simulator to Reduce Simulator Sickness, 2010). Compared to I/O devices (displays, platforms, etc.), life-like visual environments enhance the feeling of immersion at an impressive rate with much less costs, and are greatly developer-dependent.
Using existing APIs and integrated development tools, developers produce a variety of visual environments that differ greatly with respect to two key criteria: efficiency and reality.

A major advantage of using a virtual environment is that the designer may update it often to meet changing needs (Oza, 2006). A key feature of driving simulations is therefore the flexibility. Users can be exposed to different scenarios for different research purposes. The ability of designers to modify and update the virtual environment with minimal efforts is highly desired. Efficiency of developing and updating visual environments is therefore a great concern.

Graphics APIs and integrated development tools have already helped tremendously at the technical level. Developers no longer have to spend time on creating interfaces to work with graphics hardware. What really makes difference now is the algorithm developed by designers.

Another big concern of both developers and users is the reality of visual environments. The notion of presence is considered to be essential to virtual environments. According to researches done previously, a great number of key facts could be contributing to the degree of reality of the visual environment. And among them, an emerging fact is Intelligent Virtual Agents (e.g. autonomous vehicles). Realistic behavior exhibited by Intelligent Virtual Agents helps enhancing the feeling of user presence greatly, flexible and dynamic interactions between agents and their environments as well as among themselves play an important role in building a believable, life-like virtual environment (Arnellos, Vosinakis, George, & Darzentas, 2008). In the case of driving simulators, it’s especially necessary to have intelligent virtual agents to exhibit life-like behaviors to make users immerse in the driving environment.

Developers can be tied up to detailed works when directing virtual agents in virtual environments, especially when the number of virtual agents grows. Therefore, most current research in virtual environments has required autonomy in virtual agents. Presence of autonomous agents is desired in virtual environments that are immersive and interactive.
Two types of virtual environments: geo-typical and geo-specific are often seen in driving simulators. A geo-typical virtual environment is imaginary based, while a geo-specific virtual environment is real-scene based (Guo, 2005). To build a geo-typical virtual environment, designers create typical and imaginary environments. To build a geo-specific virtual environment, however, designers have to reproduce reality, in another word, re-create the real-scene as completely as possible in virtual environments.

Modeling a geo-specific environment requires much more work as compared to a geo-typical environment. It’s preferable to present geo-specific environments in driving simulators, so that simulators are more realistic and immersive because they closely resemble the real-world scene with which users will be interacting in reality. Consequently, results obtained in driving simulators with geo-specific virtual environments are more convincing.

Ideally, designers should be able to reproduce any objects in the real-scene. However, this is not the real goal of geo-specific modeling for following reasons: 1) reproducing all details of a real-scene is extremely time-consuming and labor intensive. A lot of manual works are required no matter what techniques designers may adopt; 2) reproducing all details of a real-scene is often not necessary, especially when the virtual environment is presented in mid-level driving simulators, which try to provide drivers adequate immersive and realistic driving experiences with affordable expenses and efforts. For example, it’s not necessary to capture road surface details in virtual environments presented in static based driving simulators.

1.3 Research Objectives and Research Problems

1.3.1 Research Objectives

This study intends to propose algorithms to address the issues involved in building virtual environments discussed in the above sections. The research will enable developers to reproduce high-fidelity geo-specific environments accurately and efficiently in driving simulations.
Three specific objectives of this study are listed as follows:

**Objective 1**: To efficiently construct a geo-specific driving environment with a high level of details. Various types of roadways that are commonly seen in real life driving will be modeled to demonstrate techniques proposed in this research. The 3-D geometric information about the shape of roadways is represented concisely. The process of modeling roadways will be automated as much as possible to lower designers’ workload.

**Objective 2**: Fill the virtual environment with autonomous traffic. The goal is to have simulated objects behave as humans, or in the case of simulated vehicles, as if humans controlled them. In this study, autonomous vehicles will be modeled so that they’re capable of perceiving the surrounded environment and communicating with other vehicles. Consequently, vehicles are capable of reaching right decisions and behave autonomously based on the instantaneous information they perceived and received.

**Objective 3**: To construct a virtual application with high efficiency and scalability. Reconfiguring and updating the virtual environment will be facilitated. The ability of the application to handle a growing number of dynamic objects and form orderly virtual traffic without requiring much extra work from designers will be considered as a primary goal.

### 1.3.2 Research Problems

The following six problems, associated with objectives stated above, will be addressed in this research. All proposed solutions will be demonstrated in driving simulations.

**Problem 1**: How to blend geo-standard and geo-specific?

Ideally, developers would try to reproduce every single detail of the real-world when reproducing geo-specific environment since that’ll provide the most believable virtual environment to users. And it’s only natural to conclude that the sense of presence would hence be enhanced to a much higher level. It is,
however, neither preferable nor necessary in this research due to two factors: 1) main purpose of the driving simulation built in this study, and 2) efficiency. Moderate driving simulations are developed to provide users realistic experiences with affordable costs. Trying to capture every detail and reproduce it in virtual environments is tedious and costly.

Some details of the environment are contributing to drivers’ experiences, but don’t have to be exactly the same as in the real-world. In another word, can be geo-standard. For example, road medians, curbs are components of the road network. However, it’s worth standardizing these details when modeling a real-world driving scene because it significantly decreases the time spent.

With geo-standard modeling and geo-specific modeling blended, developers can reproduce real-world driving scenes with affordable costs without compromising much on the degree of reality.

**Problem 2:** Geometric data acquisition of real-world road networks. Furthermore, how to guarantee both the efficiency of the data preparation process and the accuracy of the data?

Road network is an essential component of virtual environments for driving simulations. And most part of the road network is geo-specific. Developers thus have to define data sources and obtain data of the real-scene road network to be modeled before trying to reproduce it in virtual environments.

The process of geo-specific data preparation can significantly affect the efficiency of developing virtual environments, and it also plays an important role in providing realistic driving experience to users. Developers therefore have to come up with an approach to acquire the data of the road network with an appropriate amount of work, as well as a good degree of accuracy.

**Problem 3:** Geometric data reproduction in virtual environments

After obtaining geometric data of the road network, the problem of converting the data into 3D geometric models then emerges. The raw geometric data acquired should be processed so that it could be used by
3D modeling tools. This process has to be automated since the workload would grow dramatically as the scale of the road network grows if it’s heavily relied on human interferences.

**Problem 4: Storing, managing and retrieving data**

In order to enable autonomy in simulated objects, the ability of the application to provide comprehensive information about the environment has to be enabled. As the complexity of the virtual environment grows, a large amount of data will be needed. In order to ensure a high scalability, the issue of storing, managing and retrieving data in an efficient manner will be addressed.

**Problem 5: How to model the process of perceiving, communicating and decision making?**

For a simulated object to perceive, it needs to possess the ability to “see” the surrounding environment. The prerequisite of the perception is that the simulated object is aware of its current location. The scope of the “surrounding environment” has to be defined dynamically as the object moves around in the environment.

Modeling the communication process between simulated objects requires that the simulated object understands its current driving scenario. It has to filter simulated objects whose behaviors concern it from all other simulated objects presented in the environment. It needs to not only be aware of its own behavior at the next stage, but also be able to estimate possible actions of other simulated objects in the scenario.

The decision-making model takes information both perceived by the perception model and received by the communication as the input. The model needs to simulate the process of logical reasoning and reaches right decisions in real-time with respect to the instantaneous driving condition with no perceptible delay.

**Problem 6: How to balance realism and reproducibility?**

With the geo-specific virtual environment built, it’s key to fill it with traffic to increase the sense of presence. Reproducing types of driving situations is not only necessary in enhancing the driver’s
experience, but also is important in training drivers and testing driver assistance systems. Two characteristics: realism and reproducibility are often required when implementing traffic to the virtual environment. However, realism and reproducibility can be counterpart. In order to ensure high reproducibility, the behavior of the surrounding road users is often strictly controlled. Due to both the complexity of the scenario programming and the programming effort required, it’s quite difficult to create scenario situation with high complexity if strictly controls are applied on vehicles.

On another side, if vehicles are given a high level autonomy, it’s difficult to ensure reproducibility amongst participants. As a consequence, participants may run into different driving situations. This is undesired when driving simulators are used as platforms of experiments.

Therefore, designers have to face the trade-off between realism and reproducibility when introducing virtual traffic to the virtual environment. An approach to solve this problem is to combine autonomous and controlled simulated vehicles

1.4 Organization of the Thesis

The thesis is organized as follows:

Chapter 1 introduces virtual environments, driving simulators and key components of driving simulators. It then states the objectives of this study. Three problems associated with the objectives are also presented.

Chapter 2 is a literature review of relevant topics. It first reviews the previous works done in the area of extracting roadway centerlines from aerial images. Then studies done in topological modeling are covered. Achievements and disadvantages of those studies are discussed. Last, this chapter reviews the studies on driving behavior models. This chapter aims to provide a good understanding of the efforts made to model interactive, immersive virtual environments.
Chapter 3 discusses the process of geometric modeling of virtual environments. A new algorithm that prepares the geographical dataset with both efficiency and accuracy taken into account is introduced. Steps involved in converting data into 3-D geometric models are then described in details.

Chapter 4 discusses topological modeling and introduces two topological structures: global and local. A database that stores all geometric and topological information about the virtual environment is created. Furthermore, route planning for autonomous vehicles is introduced as a major effort to lower workloads of developers. A key object: trigger is discussed regarding its usage in scenario design.

Chapter 5 gives a comprehensive description of the perception, communication and decision making models. Autonomy is enabled in simulated vehicles with the implement of these three models. The decision model is built into three typical driving behavior models.

Chapter 6 summarizes the contributions of this study and proposes the future work.
2 Related work

This chapter reviews the related work done previously that provides a foundation for our work.

As introduced in Chapter 1, the research described in this thesis focus on two issues: 1) propose an approach to geographically visualize geo-specific virtual environments efficiently and 2) fill the virtual environment with autonomous traffic to provide users a sense of presence and more immersive driving experiences.

In the past decades, the demand for more realistic driving simulations in a broad range of fields has expedited a great number of researches on environmental simulation. Thanks to the great advancement in graphics card and computer processing power, developers are capable of developing large-scale virtual environments. As a consequence, it requires more and more efforts from developers to develop and update virtual environments. An abundance of studies were therefore conveyed to build virtual environments with less tedious manual works.

Another great concern of developers these days are autonomy of virtual agents. And that requires a more comprehensive representation of the virtual environment.

2.1 3D Modeling of Virtual Environments

3D object modeling is the basis of virtual environment construction (You & Neumann, 1998). In the case of developing driving simulations, the 3D object modeling process produces 3D models that formulate the environment in which users navigate, and influences the driving experience greatly. Existing 3D modeling systems make it possible to generate realistic 3D models of virtual environments, and rapid progress in graphic cards allows extensive use of textures (Thomas & Donikian, 2000). With the purpose of lowering the amount of manual works significantly, researches (Evans, 1995) (Sun, Yu, Baciu, & Green, 2002) (Willemsen, Kearney, & Wang, 2006) were done to automate the process of 3D model generation. However, these studies were more focused on visual effects, instead of the geometric fidelity,
of the generated models. As a consequence, the problem of efficiently modeling geo-specific environments was not addressed in these researches.

Various types of 3D models such as roads, buildings, vegetation, terrain, etc. are typically seen in virtual environments of driving simulations. This study doesn’t try to cover the modeling process of all types of 3D models, but will only focus on the most important object: the road network.

Previous work related with road network modeling will be reviewed in the following section.

2.1.1 Road Network Modeling

Road networks are critical infrastructures in human societies and are also essential to virtual environments of driving simulations. The road network not only provides the surfaces for users and dynamic agents to navigate on, but also has enormous impact on the implementation of autonomous traffic (Wang, Kearney, Cremer, & Willemse, 2005), which will be described in Chapter 5.

Previous studies in the area of modeling road networks can be generally categorized into two sub-groups: 1) Data acquisition of the road network and 2) Reproduce the road network based on the geographic data obtained.

2.1.1.1 Data Acquisition of the Road Network

In order to reproduce a real road network in virtual environments, developers first have to obtain the realistic data that stores the information of real roads to ensure the geometric fidelity. In recent years, mostly motivated by urban planning, traffic management and vehicle navigation, extraction of road networks has become a popular research topic that received considerable attention.

A number of data sources were adopted when extracting the acquisition of road networks for Geographic Information System (GIS).

A Geographic Information System (GIS) is a computer-based system that enables capture, administration, analysis and visualization of geographic information (Haunert, Brenner, & Neidhart, 2005). Vast
geographic data are captured via a variety of methods such as remotely sensing with camera, digital cameras and LIDAR.

As one kind of GIS data, navigable road networks GIS data store geometrical information of real roads. A great number of GIS providers, both public and commercial, are available for developers to use. As a consequence, the acquisition of data that can be used to create simulations emerged as a research field that draws a lot of attentions in the past years.

Usually, it is a good deal of work to find the data that developers need. And also, datasets from different sources do not always fit together since regulations with which providers comply vary greatly from place to place. Integration of datasets from different sources poses great hardship on developers.

With GIS software provided either publicly or privately, a conventional road extraction method is manually digitizing. This method is tedious and time consuming, considering the rapid development of road networks nowadays. Therefore, most researches and studies in the area of extracting road networks aimed to automate the extraction process.

Due to the great advancement in digital photogrammetry in the past years, photogrammetry had become an active method for GIS data acquisition. Aerial images therefore became a standard data sources for GIS. An abundance of researches were done to extract geographic data of road networks from high resolution aerial images.

Certainly, road networks can be manually extracted from images. Although a lot of human efforts are required, manually plotted results are still often used as reference data when evaluating the obtained results of semi-automatic and automatic road extraction approaches.

According to the extensive survey done by (Mena, 2003), road extraction works were classified into eight categories according to the extraction technique applied. And they are: 1) Road tracking methods; 2)
Morphology and filtrate; 3) Dynamic programming and snakes; 4) Segmentation and classification; 5) Multi-scale and multi-resolution; 6) Stereoscopic analysis; 7) Multi-temporal analysis and 8) Others.

Although the goal of all researches were to fully automate road extraction, in practical, existing approaches proposed could generally be divided into two categories: semi-automatic and automatic.

Semi-automatic methods are methods that require human interaction as complements, at some point of the extraction process. One set of semi-automatic methods introduced manual works at the beginning of the extraction by giving points, directions, or road seeds. Another set of semi-automatic methods bring in human works at the end of the extraction process by editing the extracted results.

The first set of semi-automatic methods normally starts road extraction with supervisions from developers. Developers provide some information that is hard to detect automatically and is of use to extraction strategies. The type of information that developers provide is dependent on the extraction strategies adopted. A variety of strategies was covered in existing semi-automatic approaches and can be classified into five categories (Kim, Park, Kim, Jeong, & Kim, 2004): perceptual grouping, scale-space, neural network and classification, “Snakes”/energy-minimizing, and profile matching.

Profile matching approaches work by defining a template from one point on the feature of interest and find the rest of the feature by matching. In order to apply profile matching, the features of interest must have similar brightness patterns. (Vosselman & Knecht, 1995) proposed a profile matching approach that relies heavily on human interaction. They initialized a road tracer by measuring two points that indicate a short road segment. Grey value cross sections are computed at intervals of one pixel between the two points. The model road profile is taken as the average of these cross sections. This model profile is used as the template in the profile matching.

(Kim, Park, Kim, Jeong, & Kim, 2004) designed a new profile matching algorithm to extract road centerlines based on two assumptions: 1) road centerlines are appeared as curvilinear features and have distinctive brightness patterns. As a result, profile matching can be applied along the road centerlines; 2)
geometric transformation between one point and the other on a road centerline can be modeled as similarity transformation. And hence, a rectangular window centered on a point of a road centerline can be transformed by translating and rotating to define the window at another point of the road centerline.

In the study of (Gruen & Li, 1997), a semi-automatic method that combines a dynamic programming strategy and a “Snakes” strategy was proposed. A new kind of Snake class, the LSB-Snakes (Least Squares B-splink Snakes), was developed. The extraction scheme is presented in Figure 2.1.

Figure 2.1 A semi-automatic extraction scheme (from (Gruen & Li, 1997))

Human works are performed at the step of “Feature Identification & Classification”. Human operators identifies the object from the pre-processed digital image, selects the particular class this object belongs to, and provides a very few coarsely distributed seed points. With these seed points as an approximation of
the position and shape, the linear feature will be extracted automatically through either the dynamic programming approach or the LSB-Snakes approach.

A detailed description of abstracting roads with the concept of scale-space is provided by (Mayer & Steger, 1998). The scale-space behavior of roads was utilized. Basically, the appearance of roads in digital imagery depends on the sensor’s spectral sensitivity and its resolution. Images with various scale exhibit different characteristics of roads. In coarse scale images, roads appear as lines establishing a network. At coarse scales, finer scale substructures of the road, such as cars on the road, road markings, and disturbances are eliminated. While in finer scale images, roads are depicted as bright elongated areas with almost constant width and bounded curvature. Therefore, roadways can be extracted by taking advantage of the scale-space behavior or roads. (Laptev, H., Lindeberg, Eckstein, Steger, & Baumgartner, 2000) presented a research that aimed to extract roads from aerial images automatically with scale-space and snakes. Both coarse scale images and finer scale images are utilized. Coarse scales give global information which is suited for initial road detection, fine scales add detailed information which can be used to verify and complete the road network. Three terms: salient roads, non-salient roads and crossings were defined in this research for the extraction of the road network. Salient roads are roads with distinct appearances in images. A line at a coarse scale initializes the centerline of a ribbon snake at a finer scale. And the extraction of salient roads is restricted to homogenous rural areas. Non-salient roads are parts of the road network that are more or less disturbed in the image by shadows or partial occlusions. Since all roads are connected into a global network, extraction of non-salient roads are done by connecting adjacent ends of salient roads as start and endpoints of non-salient roads. An optimization was done afterwards with the “ziplock” method (Neuenschwander, Fua, Iverson, Székely, & Kübler, 1997). Crossings correspond to junctions of lines at coarse scales. To extract crossings, a closed snake around each junction was expanded and the connections between the outline of the crossing and its adjacent roads were checked.
Figure 2.2 a-c. Extraction of salient roads. a Extraction of line. b Optimization of ribbons boundaries. c Selection of ribbon parts with constant width. Only the ribbon defiled by thick lines is considered as a salient road (from Laptev, H., Lindeberg, Eckstein, Steger, & Baumgartner, 2000))

Figure 2.3a-d. Extraction of non-salient roads. a Ends of extracted salient roads. b Extraction of optimal path. c Verification by optimization of width. d Acceptance of a hypothesis with low variation of width (from Laptev, H., Lindeberg, Eckstein, Steger, & Baumgartner, 2000))
Extractions of salient roads, non-salient roads, and crossings are illustrated in Figure 2.2, Figure 2.3 and Figure 2.4.

As compared to semi-automatic extraction, fully automatic extraction detects road seeds automatically and links them to complete the road network. This is claimed to be preferable than semi-automatic extractions which still require lots of human efforts and time. However, most of the proposed methods have made several hypotheses, such as:

a) Road width is constant or changes slowly
b) Road direction changes slowly
c) Constant intensity
d) Intensity between road and background is large so that roads are easily identified
e) Road edges are straight and smooth

Some of these assumptions are not always true and as a result, these methods are quite sensitive to disturbances such as cars, buildings, trees, etc.
At Technische Universität München (TUM), (Wiedemann, Heipke, Mayer, & Jamet, 1998) and (Wiedemann & Hinz, 1999) presented an automatic roads extraction approach for roads in rural areas from optical imagery. Multiple image channels were used and the road segments extracted from these image channels are fused, road junctions are introduced and a weighted graph of road segments is constructed. And finally, a road network is extracted connecting road seeds by optimal paths through the weighted graph. In 2003, they extended their previous works to investigate the potential of the TUM approach for automatic extraction of roads from airborne SAR imagery (Wiedemann & Wessel, 2003). The extraction strategy they employed consists of a few steps, which are shown in Figure 2.5.

![Figure 2.5 Road extraction flow (from (Wiedemann & Wessel, 2003))](image)

The extraction results were evaluated based on a comparison with reference data. Both the completeness which indicates the percentage of the actually present road network, and correctness which describes the probability of an extracted linear piece to be indeed a road are calculated. The reference data were
separated into three classes: highways, main roads, and secondary roads. Evaluation results suggested that satisfying results can be achieved for main roads. However, for highways and secondary roads, neither the completeness nor the correctness is good enough.

To extract complex road networks from urban areas, (Bong, Lai, & Joseph, 2009) proposed an algorithm based on hybrid simple color space segmentation and edge detection (Hybrid SCSS-EDGE). Satellite images from Google Earth were considered due to its wide availability and less cost.

In their study, two more color models namely $YC_bC_r$ and HSV were implemented other than the basic RGB model. $YC_bC_r$ model consists of $Y$ (represents luminance or a measurement unit of the energy amount that an observer perceives from a light source), $C_b$ (represents the difference between the blue component and a reference value) and $C_r$ (represents the difference between the red component and a reference value). HSV model consists of Hue (a color attribute that describes a pure color like yellow, orange, or red), Saturation (a measurement of the degree to which a pure color is diluted by white light) and Value (intensity or average value of R, G, B at specific location). Both color spaces can be obtained by transforming the RGB color space. Characteristics of five categories of elements found in a high-resolution color satellite image, namely: bushes or trees, roads, buildings, sandy region and water regions, were studied and examined. Results suggested that by applying some appropriate thresholds of Luminance, Hue, Saturation and Intensity, Bushes/trees, sandy region, and buildings can be eliminated. Furthermore, the water regions and other small regions can be separated from road network using segmentation, and the broken road segments due to some undesired elimination can be recovered using edge detection. Table 2.1 presents the values of the five elements in of $YC_bC_r$ and HSV color spaces.
Table 2.1 Color components of the five elements

(from (Bong, Lai, & Joseph, 2009))

<table>
<thead>
<tr>
<th></th>
<th>Bushes or trees</th>
<th>Roads</th>
<th>Buildings</th>
<th>Sandy regions</th>
<th>Water regions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Luminance</td>
<td>25 ~ 100</td>
<td>110 ~ 160</td>
<td>50 ~ 150</td>
<td>100 ~ 200</td>
<td>50 ~ 80</td>
</tr>
<tr>
<td>Hue</td>
<td>0.05 ~ 0.3</td>
<td>0.05 ~ 0.2</td>
<td>0 ~ 0.1</td>
<td>0.05 ~ 0.1</td>
<td>0.1 ~ 0.45</td>
</tr>
<tr>
<td>Saturation</td>
<td>0.05 ~ 0.4</td>
<td>0.1 ~ 0.25</td>
<td>0.1 ~ 0.5</td>
<td>0.3 ~ 0.5</td>
<td>0.05 ~ 0.2</td>
</tr>
<tr>
<td>Intensity</td>
<td>0.05 ~ 0.5</td>
<td>0.15 ~ 0.8</td>
<td>0.2 ~ 0.6</td>
<td>0.4 ~ 1</td>
<td>0.2 ~ 0.3</td>
</tr>
</tbody>
</table>

This methodology is a full automatic approach that is capable of extracting road regions in both rural and urban areas in a very fast way. The drawback, however, of this approach is its accuracy. And in the case of building driving simulations, the results of this methodology are not quite usable.

(Li & Briggs, 2009) proposed an automatic extraction approach that is composed of two stages. The first stage deals with roads that contain relatively less noise and hence more identifiable. The second stage extracts roads that are less visible or heavily impacted by surrounding objects and hence harder to identify. Each stage consists of three major steps: filtering, segmentation, and grouping and optimization. Different from those made in previous works, two assumptions were derived: 1) Visual constraint: majority of the pixels from the same road have similar spectrum that is distinguishable from most of the surrounding areas; 2) Geometric constraint: a road is a region that’s relatively long and narrow, compared with other objects in the image. A road region is not required to have constant intensity and a road edge is not required to be smooth. A new method which involves two key concepts: reference circle and central pixel was proposed to separate road regions from the rest of the image. For a pixel P, its reference circle $C(P)$ is the largest circle centered at P that doesn’t contain edge points. And a pixel is a central pixel if it has...
the maximum reference circle among its neighboring pixels in a 3x3 square. The flow of the filtering algorithm is illustrated in Figure 2.6.

![Flow of the filtering algorithm](image)

**Figure 2.6 Flow of the filtering algorithm (from (Li & Briggs, 2009))**

### 2.2 Representation of Virtual Environments

Researches reviewed in 2.1 were helpful regarding the modeling of physical structures of virtual environments.

However, they’re not sufficient due to the lack of animation and interaction. Being uninhabited, virtual environments cannot provide the visitor with real life feeling (modeling virtual cities dedicated). Being not interactive not only decreases the degree of reality of the virtual environment, but also prevents virtual environments from being used by researchers to do tests, which often require interactions between users and other virtual agents.

Another objective of this research is to populate the virtual environment with autonomous agents like live beings. As an agent-based simulation, the focus is on modeling the decision making processes and
detailed motion control of autonomous vehicles (Willemsen P. J., 2000). A pre-requirement of enabling autonomy in virtual agents, therefore, is to design a robust and explicit representation of the physical environment, so that virtual agents are provided rich and detailed information about the environment. And moreover, these information should be accessible easily by virtual agents to support high-frequency and real-time information queries.

With aerial imagery as the source network data in this study, the extracted networks from aerial images don’t really provide information that is beyond the geometric level. Therefore, they’re inappropriate for navigational tasks, and when applied to driving simulations, they don’t provide enough information about the virtual environment for virtual agents to perceive. Topological connectivity between road network elements, and semantic information about road network elements are all missing so far.

Road networks are normally represented as road network graphs. An example of a network graph is shown in Figure 2.7 (Srivastava, 2010). In this network graph, each lane is represented as an arrowed edge, where the arrow indicates the traffic direction. Intersections are not presented as separated elements in the graph. However, topological connectivity between road network elements is faithfully presented. Interconnections between lanes in an intersection are therefore explicitly given.

![Figure 2.7 Representation of road networks in (a) conventional geospatial data and (b) as a road network graph](from (Srivastava, 2010))
Some other researches represent road network as graphs as well, but in a different way. Edges represent roads with one or more lanes and nodes represent intersections. Traffic direction of each lane is not represented in the graph structure. However, lanes are preserved to store information about lane adjacency orderings. A study done by (Christian, Jan, B.P., & Timko, 2003) represented a real-world road network as a graph (shown in Figure 2.8) with the purpose to present a data model that supports nearest neighbor search. Each intersection and terminal vertex is transformed to a node, which has a unique id assigned. Their graph representation however doesn’t consider directional attributes and traffic regulations, which means the graph only provides partial information about the road network.

![Figure 2.8 Two-dimensional and Graph Representations of Road Network (from (Christian, Jan, B.P., & Timko, 2003))](image)

In a research that studies tactile behavior in high driving situation, roads are represented as a connected set of Road segments (Sukthankar, 1997). In this representation, intersections are not represented as standalone elements. Road segments connect to others on a lane-by-lane basis.

These three researches reviewed above all have demerits when representing intersections. Either intersections are eliminated from the representation graph or the topology for interconnections between lanes is not shown. For virtual agents to traverse through intersection autonomously, both the geometry
and topology of interconnections of lanes must be represented and provided to virtual agents. At the same time, it’s not wise to not represent intersections as separate elements since some semantic information of the road network is lost in that way.

As a result, intersections are treated as special elements by a number of studies. Topology connectivity between intersections and roadways are preserved. And in order to also represent the connectivity between incoming and outgoing lanes within an intersection, extra components were usually introduced.

Pathways that connect incoming to outgoing lanes, for example, were implemented to guide traffic explicitly through intersections. A benefit of this representation is that routes through intersections can be modeled with great flexibilities. Agents are allowed to follow non-standard routes, which add more reality to agents’ behaviors. (Willemsen, Kerney, & Wang, 2003) introduced a representation called a path which melds road and intersection segments into a single, continuous ribbon. In that case, interconnections between lanes in an intersection are no longer needed.

However, implementation of pathways are not ideal for two primary reasons: 1) developers would have to devote a lot of efforts to model these pathways one by one and 2) hence agents are not behaving autonomously when coming to intersections due to complete control over all pathways by developers.

To eliminate the cost for representing the connection between lanes, (Bayarri, Fernandez, Perez, & Rodriguez, 1995) and (Bailey, Jamson, Parkes, & Wright, 1999) automated the generation of lane interconnections.

(Wang, Kearney, Cremer, & Willemse, 2005) introduced a representation called a route to enable wayfinding. The route doesn’t provide a moment-to-moment guide for steering a virtual agent, instead, it is a strategic goal of the agent. Agents are allowed to make decisions and form the actual path that they follow in real-time with regarding to real-time situation. This approach lowered workloads of developers as well gave virtual agents higher degrees of autonomy.
Researches done to model intersection as special elements of the road network helped to provide more complete information about the virtual environment to virtual agents. Traffic directions and traffic regulations not only on roadways, but also at intersection, are taken into account in their studies.

A major disadvantage of these researches is that no representation that stores geometric, topological, and semantic information universally was proposed. Consequently, virtual agents were not making decisions and behaving autonomously based on information they perceived about the driving environment. For example, at intersections, it’s the developer not the virtual agent knows which outgoing lane it should be traveling to. The pathway or route is defined by developers. Furthermore, these researches didn’t enable the communication amongst virtual agents. They succeeded in guiding agents through the environment without having to give specific guidance, but they failed in having agents behave like live-beings still.

Donikian, S. proposed in his studies done in late 1990s (Donikian, 1997) a Virtual Urban Environment Modeling System (VUEMS) which has three levels of representation of virtual environments: geometric, topological and semantic, with the purpose to add autonomous agents in the simulation. A set of eleven different classes, each specimen possessing its own geometry, was used to represent the road network at the geometric level. Topologic information was managed at two different levels: lowest and highest. The lowest topologic level links with the Geometry. At this level, the topological information is represented as a graph that interconnects all the road sections of the road network. Each elementary section is a node in the graph. Each node has number of connections that equals to number of In/Out points. Figure 2.9 shows a road network and its corresponding topological information.

The highest topologic level links with the semantic. This level provides a list of outstanding road elements forming the path to be used to travel from the current location to the desired one. Road elements can be streets, street segments, crossings, etc. The graph associated to this level is automatically computed by using the lowest topologic graph and the hierarchical semantic structure. Last, a hierarchical structure is defined above the geometric level to permit to describe conceptual objects such as a street or a square (as shown in Figure 2.10).
Figure 2.9 A Road Geometry configuration example and its lowest topologic level (from Donikian, 1997)

Figure 2.10 The hierarchical structure describing urban concepts (from Donikian, 1997)
The concept of multilevel modeling of virtual environments helped tremendously in standardizing the construction of virtual environments as well providing abundant information to virtual agents to behave autonomously.

2.3 Driving Behaviors

Studies reviewed in 2.2 all aimed to represent the road network more comprehensively so that virtual agents are provided enough information to make decisions and take actions like live-beings. With the information of the driving environment obtained however, virtual agents are not capable of reaching decisions and hence taking proper actions with the absence of driving behavior models.

Driving behavior models describe drivers’ decisions regarding their vehicle movement under different traffic conditions. These models include speed/acceleration, which describe the movement of the vehicle in the longitudinal direction, and lane changing models, which describe drivers’ lane selection and gap acceptance behaviors.

To model driving behaviors in driving simulations, developers have to closely observe driving behaviors of drivers in real-world, so that virtual agents could demonstrate similar behaviors under same circumstances.

A great number of driving behavior models were proposed to capture the complexity of human decision-making process. And these models can be classified into three categories: 1) Acceleration models; 2) Lane changing models and 3) Intersection models.

2.3.1 Car-following Models

Car-following behavior describes the processes by which drivers follow each other in the traffic stream. Brackstone and McDonald (Brackstone & McDonald, 1999) gave a historical review on car-following models formulated and classified them into five categories.
2.3.1.1  **GM Model (GHR Model)**

The GM model was proposed at the General Motors research labs in late 1950s.

The first prototype of this model was linear (Chandler, Herman, & Montroll, 1958) and relative simple. It’s based on the equation given as in equation Equation 2.1.

\[
a_n(t) = \alpha \Delta V_{\text{front}}^n (t - \tau_n)
\]

**Equation 2.1**

Where \(a_n(t)\) is the acceleration applied by driver \(n\) at time \(t\). \(\Delta V_{\text{front}}^n\) is the relative speed of the leading vehicle measure at time \(t - \tau_n\). \(\tau_n\) is the reaction time for driver \(n\). \(\alpha\) is the constant sensitivity parameter. This model ignores the dependency of the response to the spacing between the vehicles. A non-linear model, represented as in equation Equation 2.2, was therefore developed (Gazis, Herman, & Potts, 1959) to take the spacing into account.

\[
a_n(t) = \alpha \frac{\Delta V_{\text{front}}^n (t - \tau_n)}{\Delta X_{\text{front}}^n (t - \tau_n)}
\]

**Equation 2.2**

Where \(\Delta X_{\text{front}}^n (t - \tau_n)\) is the spacing between the following vehicle and its leading vehicle measured at time \(t - \tau_n\).

In 1961, Gazis, et al. added non-linearity to the sensitivity of the response to the spacing and the following vehicles’ speed and drew the most general form of the GM model.

\[
a_n(t) = \alpha \frac{V_n(t)^\beta}{\Delta X_{\text{front}}^n (t - \tau_n)} \Delta V_{\text{front}}^n (t - \tau_n)
\]

**Equation 2.3**

Where \(V_n(t)\) is the speed of the following vehicle’s speed measured at time \(t\). \(\alpha, \beta\) and \(\gamma\) are parameters.
2.3.1.2 The CA model

This model was presented by (Kometani & Sasaki, 1959) and describes the safe following distance, which is required to avoid collision with the leading vehicle, as a function of speeds of the following and leading vehicles, and the driver’s reaction time. The equation is given in equation Equation 2.4

\[ \Delta x(t - T) = \alpha v_{n-1}^2 (t - T) + \beta_1 v_n^2(t) + \beta v_n(t) + b_0 \]

Equation 2.4

Where \( v_n \) is the speed of \( n^{th} \) vehicle. \( v_{n-1} \) is the speed of \((n - 1)^{th}\) vehicle. \( \Delta x \) is the relative distance between vehicle \( n \) and \((n - 1)\). \( T \) is the reaction time of the driver. \( \alpha, \beta, \beta_1 \) and \( b_0 \) are calibration parameters.

Gipps proposed the Gipps’ model, a development of the CA model, in 1981 (Gipps, A behavioural car following model for computer simulation, 1981). The Gipps’ model considered several factors that the earlier CA model neglected. First, drivers will allow an additional “safety” reaction time equal to \( T/2 \). Second, the kinetic parameters in equation Equation 2.4 are related to braking rates of \(-1/2b_n\), \( b_n \) is the maximum braking rate the driver of the \( n^{th} \)vehicle wished to use, and \(-1/2b^*\), \( b^* \) is the maximum braking rate the \((n - 1)^{th}\) vehicle that the \( n^{th} \) driver believes is likely to be used.

2.3.1.3 Linear Model

This model, proposed by Helly in 1959 (Helly, 1959), relates the acceleration of the following vehicle to desired following distance, speed of the leading vehicle, relative distance and relative speed between following and leading vehicles, and also driver’s reaction time. The model is based on the equation given in equation Equation 2.5.

\[ a_n(t) = C_1 \Delta v(t - T) + C_2 (\Delta x(t - T) - D_n(t)), \quad D_n(t) = \alpha + \beta v(t - T) + \gamma a_n(t - T) \]

Equation 2.5
Where $a_n(t)$ is the acceleration of vehicle $n$ implemented at time $t$. $D_n(t)$ is the desired following distance at time $t$. $v$ is the speed of $n^{th}$ vehicle. $\Delta v$ is the relative speed between vehicle $n$ and $n - 1$. $\Delta x$ is the relative distance between vehicle $n$ and $n - 1$. $T$ is the driver reaction time and $\alpha, \beta, \gamma, C_1$ and $C_2$ are calibration constants.

### 2.3.1.4 Other Car-following Models

Other models have also been proposed and studied to model drivers’ following behavior, such as Psychophysical Model and Fuzzy-Logic-Based model. These two kinds of models are relatively complex and are unlikely being used in the case of driving simulations.

### 2.3.2 Lane-Changing Models

Lane changing behavior is quite commonly seen in daily driving and hence is required to be produced in driving simulations often. It can be influenced by a numerous factors and the need in development of lane changing models has emerged.

As pointed out by (Toledo, Koutsopoulos, & Ben-Akiva, 2003), lane-changing behavior is usually modeled in two steps: 1) the decision to consider a lane-change and 2) the decision to execute the lane-change.

The first steps models the decision-making process on whether drivers should consider a lane-change. Depends on numerous factors, drivers can choose to either stay in the current lane, or change to the target lane. The process of selecting the target lane has been studied in the past decades.

The first lane-changing model, which is still applied in microscopic traffic simulations, was proposed by Gipps in 1986 (Gipps, 1986). The model covers a variety of scenarios in urban driving simulations, in which traffic signals, transit lanes, obstructions and presence of heavy vehicles affect drivers’ selection of lanes. Gipps’ model is more of a model that decides whether to consider a lane-change. Two considerations decide whether the vehicle should consider a lane-change: 1) whether the vehicle is
traveling at the desired speed and 2) whether the vehicle is in the right lane to perform upcoming turning maneuvers.

To govern the decision-making process on considering a lane-change, Gipps defines three zones: when the upcoming turn is far way, drivers concentrate on attaining the desired speed. In the middle zone, drivers only consider switching to the turning lanes and adjacent lanes. In the last zone, drivers focus primarily on changing to the correct lane and ignore speed considerations when the turn is close.

In another well-known model designed for microscopic traffic simulation: CORSIM (Halati, Lieu, & Walker, 1997), lane changes are classified as either mandatory or discretionary. Mandatory lane-changes (MLC) are conducted when the driver must switch to another lane. Discretionary lane-changes (DLC) are performed to improve driving conditions. CORSIMS is a development of Gipps’ model when considering lane-changes.

Ahmed in late 1990s (Ahmed, 1999)(Ahmed, Ben-Akiva, Koutsopoulos, & Mishalani, 1996) developed a lane-changing model that covers both MLC and DLC situations, by means of implementing a framework that models three lane-changing steps: 1) decision to consider a lane-change, 2) choice of a target lane, and 3) acceptance of gaps in the target lane. Basically it was developed from previous models and construct similar logical rules into the framework to reach proper decisions under different traffic situations.

The framework proposed by their study is presented in Figure 2.11.
A lot of other studies adopted the classification of DLC and MLC and models the lane-changing behavior under these two situations.

In the study done by (Wang, Kearney, Cremer, & Willemse, 2005), a lane-changing behavior was modeled as two steps: 1) lane changing decision making, and 2) lane changing action. At the first step, both DLC and MLC decisions were discussed. In DCL situations, driving agents evaluate the driving condition on the current lane with the speed of the traffic on the current lane. And they believe that traffic speed is limited by the slowest moving vehicle on the lane. Therefore, the model checks the speed of all lead vehicles on the current lane with a certain range, which is defined as expressed in equation Equation 2.6

\[ S_r = \max (S_2, v \times k_r) \]

Equation 2.6
Where $k_r$ is a constant, $v$ is the current speed of the vehicle, and $S_2$ is the minimum range of distance. The speed of the traffic on the current lane is then defined to be the speed of the speed of the slowest leading vehicle within the defined distance. The model does a simple comparison between the speed on the current lane and the target lane to decide whether the target lane has a better traffic condition. The MLC decisions were primarily motivated by the desire to follow routes. It didn’t cover more complex traffic situations, such as obstructions, heavy vehicles, etc. Moreover, when a DLC and a MLC decision conflict, both decisions are fused into a single lane change decision with the rules that a decision to change lane has higher priority over a decision to not change lane, and that an MLC decision has a preemptive priority over a DLC. The model proposed in their study simplifies the traffic situations and logical rules, and can be easily adopted in some driving simulations where low density of traffic present.

An integrated lane-changing model was developed by (Toledo, Koutsopoulos, & Ben-Akiva, 2003) to overcome limitations of classifying lane-changes as mandatory or discretionary. They claim such a classification prohibits tradeoffs between these two considerations. Also, determination of conditions that trigger MLC is often defined by simple rules in most cases. The integrated model enables joint evaluation of mandatory and discretionary considerations. Their results suggested that drivers’ lane selection is affected by path-plan variables and traffic conditions in their neighborhood.

In most lane-changing models, it’s important to evaluate the available gap in the target lane to execute a safe lane-change. To determine whether the gap between the lead and lag vehicles in the target lane is sufficient, drivers have to assess the positions and speeds of them.

The decision reached by gap acceptance models is binary: accept the available gap and execute a lane-change right away or reject the available gap and stays in the current lane.

Critical gaps are implemented in all gap acceptance models to evaluate whether the available gap in the target lane is sufficient. How to define critical gaps though, varies from model to model. In CORSIM, critical gaps are defined by risk factors. The risk factor is defined by the deceleration a driver will have to
apply if the lead vehicle brakes to a stop. For every lane-change, the model calculates the risk factor to the subject vehicle with respect to the lead vehicle in the target lane, and also the risk factor to the lag vehicle in the target lane with respect to the subject vehicle. Under different circumstances with different degrees of urgency, the risk factor is being compared to an appropriate risk factor, and therefore defines whether it’s safe to execute a lane-change.

In the lane-changing model proposed by Ahmed, both the lead gap and the lag gap must be acceptable in order to execute the lane-change.

2.3.3 Intersection Behavior Models

As compared to car-following and lane-changing models, driving behavior at intersections was not studied that intensively.

The intersection behavior model provided by Wang, et al. in the same study introduced in 2.3.2, a simple but effective intersection model was introduced. Their intersection behavior model works as a gate that prevents vehicles from entering the intersection until it is their turn to cross. To determine whether a vehicle has the right to cross the intersection, two factors are included: 1) traffic signal, and 2) right of way. Corridors are applied at intersections to guide vehicles through. Each corridor that crosses an intersection is labeled with right-of-way information that describes the priorities of traffic crossing this corridor with respect to traffic crossing other corridors. Based on the vehicle’s current position and speed, the model estimates the time $t_1$ when the vehicle will enter the intersection and the time $t_2$ when the vehicle will leave the intersection. Only if no vehicle having the right of way will traverse the corridor between time $t_1$ and $t_2$, then the vehicle is allowed by the model to proceed through the intersection. Otherwise, the vehicle will stop and yield.

Intersection behavior models can be really complex when human factors are introduced. To model intersection behaviors, similarities in drivers’ perception and actions need to be studied. A model is only possible if there are something consistent behaviors among all drivers. (Bjelkemyr, 2009) did a study to
try to model how drivers perceive and act at intersections. Two hypotheses were proposed as the baseline of the model: 1) the gaze and/or head movement anticipates the vehicle heading and 2) the vehicle heading and gaze and/or head movements are sufficient to characterize the actions of the driver-vehicle system in the intersection. The model built actually found some consistency across cases with the same sequences of road types in drivers’ actions.

The study of (Zou & Levinson, 2006) derived an intersection behavior model, which takes a great number of human factors into account, with Markov dynamic theory. They acknowledge that driver perceive much information from their own vehicles, other vehicles, traffic facilities and environment and generate responses through a decision-making process that’s not tractable. The understanding of a driver on his own vehicle and driving environment varies greatly from person to person, and vehicle dynamics vary from one to another. These factors all contributed to the complexity of modeling drivers’ behaviors at intersection. However, they then pointed out that drivers’ behavior is predicable to some extent. Psychological and physiological limitations of drivers and drivers’ consistent preferences under certain circumstances both helped to build an intersection behavior model. To simplify the problem though, only conflicts related to crossing and left-turn traffic was studied. Other intersection behaviors such as Right-turn, U-turn and lane-changing behaviors in intersections were not considered. This model surely observes intersection behaviors with more understandings. It is capable of modeling the variety of driving behaviors at intersections, which helps to increase the reality of driving scenes when applied to driving simulations. However, the complexity of the model is high considering many efforts were spent to study just one kind of intersection conflict. Also, when applied in driving simulations, the variety of virtual vehicles’ behaviors at intersections is probably not worth taking much time from developers in building such an intersection behavior model.
3 Geometric Modeling

To construct a virtual driving environment, a number of tasks are involved, including modeling roadways, generating terrain, filling the road network with autonomous traffic, adding static environment objects (signage, vegetation, buildings, etc.) and so on. Among these tasks, modeling roadways is the fundamental step for designers to work on. None of the other tasks could be performed with the absence of the road network. Designers would have no idea about where to place those static environmental objects. It’s also impossible to generate a terrain that conforms to roadways without knowing the 3-D structure of roadways. And it’s obviously not possible to fill the driving environment with autonomous traffic without having roadways built. Therefore, it’s necessary for designers to come up with a methodology that produces the road network with the following concerns taken care of: 1) real-world roadways should be reconstructed in virtual environments efficiently with an appropriate level of details; 2) reconfiguring and updating the road network should be facilitated; 3) autonomous traffic should be filled in to the road network easily.

With the above issues taken into account, in this study, the process of modeling roadways is further divided into two steps: 1) geometric modeling and 2) topological modeling. The goal of geometric modeling is to generate concise and accurate geometric representations of roadways in virtual environments. The output of this step is a three-dimensional road network. Details of geometric modeling are described in this chapter and those of topological modeling will be discussed in Chapter 4.

To reconstruct real-world roadways which have large coverage and complex geometry, developers first have to come up with a dataset that contains the geographical data of the road network. Geographical data consist of the longitude-latitude data and the altitude data. In another word, designers have to get the longitude, latitude and altitude data of the roadways first before reconstructing them concisely in virtual environments.
After getting the geographical data of the roadways, designers then proceed to convert the 3-D raw data into 3-D geometric models.

Efficiency and flexibility of the Geometric Modeling process depends greatly on how the geographical dataset is prepared by designers and what approaches are adopted to convert the dataset into geometric models.

This chapter introduces a methodology that prepares the geographical dataset of a road network with much less manual works and a much greater efficiency. Furthermore, the process of converting the large amount of geographical data into 3-D geometric models which are smooth in slope and elevation is also described in details in this chapter.

3.1 Preparing the geographical dataset

The first issue to be addressed in preparing the dataset of the roadways is the sources of road data to use. In another word, what data should be contained in the dataset to represent a roadway? Nowadays, in most roadway geometry data sources, public or commercial, roadway segments are depicted as centerline of the travel way. In some cases, linear edges indicating edges of curb and pavements, and polygonal extent of road areas are also used as the data source to represent roadways. This project adopts centerlines as the data sources in representing roadways to keep consistent with standards of most road databases. And also, centerlines can be used to encode various road characteristics, such as road name, road width, number of travel lanes, etc.

Now that centerlines are chosen to represent roadways, the question becomes, how could we get the centerlines of real-world roadways?

As indicated in the report provided by New Jersey Office of Information Technology in 2008 (New Jersey Office of Information Technology, 2008), a great number of road centerline data sets were created and
used throughout states in the U.S. However, it’s not preferable to use these existing centerline data sets in this project because:

1) A wide variety of centerline data sets are available for each state, and it’s difficult to choose the most appropriate one; The data contained in these data sets varies greatly in coverage, completeness, accuracy, attributes, etc. It’s almost impossible for designers to develop a universal methodology that reads in centerline data from different datasets. For example, a road centerline dataset in Massachusetts is unlikely to have the same sets of attributes, the same degree of accuracy as a dataset in New York.

2) Existing data sets are often too large to use in our case. Selecting the data that represents the road network that we wish to reproduce in the virtual environment from a large data set could be time consuming.

3) Increased costs since some data sets are commercial or even proprietary.

Due to the disadvantages listed above, we choose to develop a road centerline data set ourselves. A universal and low-cost methodology that prepares the road centerline data set is introduced below in section 3.2 and 3.3. By universal, it means that the methodology is applicable for different geographic areas in the U.S.

The proposed methodology consists of two steps: 1) get the longitude-latitude data of road centerlines, and 2) get the altitude value of each pair of (longitude, latitude) data.

To get the longitude-latitude data, aerial maps are used. Aerial maps are widely available and contain valuable information, including the road network.

3.2 Extracting 2-D road centerlines from aerial maps

Road network extraction from aerial and satellite maps is widely studied and applied as reviewed in Chapter 2.1. It’s true that extracting road centerlines from aerial maps can be manually done. Designers can sample points along the centerline of the roadway with small intervals. By connecting the sequence of
these points, the centerline of the roadway could be plotted. Though, it’s not at all preferable not only due to the amount of efforts it takes, but also the inflexibility it possesses. In that way, to build a road network in virtual environments, designers have to work on every single road separately. Also, any further modifications to the virtual road network would require a lot more works. For example, if the centerline of a roadway was manually extracted, designers would have to resample all points along the road centerline even if the geometric shape of the roadway is slightly changed. As a consequence, the cost of the maintenance of the road centerline dataset is enormously high. Without an approach that extracts centerlines of roadways efficiently, the preparation of the dataset could be extremely time-consuming. As a conclusion, the goal of this step should be automating the process of extracting the road network from aerial maps.

As reviewed previously in Chapter 2, a great number of algorithms were developed to extract road networks from aerial images semi-automatically or automatically. It was acknowledged that high-resolution satellite images provided by, for example Google, pose great challenges for automatic feature extraction because of the inherent complexities. High-resolution images capture all details in that area including buildings, trees, vehicles, etc. Objects may interfere with each other. Also, satellite images are affected by weather and light conditions. Therefore, most of the existing approaches are easily disturbed by objects such as vehicles, shadows, etc. As a consequence, results provided by these methods are not always reliable.

In this study, the purpose of extracting the road network is not for real-life uses, such as road map construction, GIS activities, etc. Instead, the goal of this project is to reconstruct the road network in virtual environments with a certain level of details. In another word, it’s not necessary to capture and reproduce all details of real-world roadways from high-resolution aerial images.
Therefore, to avoid challenges in extracting road map completely from satellite images and also because the centerlines of roadways are of primary concern in our case, Google Maps that only displays the street map is chosen as the image resource which has eliminated disturbances already.

As described in scale-space extraction strategies, images with various scales exhibit different characteristics of roads. In coarse scale images, roads appear as lines establishing a network. At coarse scales, finer scale substructures of the road, such as cars on the road, road markings, and disturbances are eliminated. While in finer scale images, roads are depicted as bright elongated areas with almost constant width and bounded curvature.

The image resource adopted in this study combines advantages of both scales. Roads are depicted as elongated areas with almost constant width and constant intensities, as in finer scale images. Meanwhile all disturbances are eliminated as in coarse scale images.

Our approach consists of several image processing steps, takes Google Map images as inputs of the program developed on the MATLAB platform and produces extracted centerlines of road segments as outputs.

Many types of roads exist around the world. Different types of roads have different characteristics which we should take into account when reproducing them. For example, when reconstructing highways in virtual environments, we would not expect to see many intersections as when reproducing local streets. Therefore, it’s natural to think of classifying roadways into several sub-categories according to road types before extracting centerlines.

In this study, we make use of the road colors that Google Maps employed to indicate types of roads. Orange indicates interstate highway, Yellow indicates state highways and county parkways, and White indicates local and private streets. When a Google Map image is presented as an input to our program, roadways are first categorized into three sub-groups: interstate highways, state highways, and local streets, based on the colors of roadways in the image.
The flow of extracting centerlines of roadways from aerial Google Map images is illustrated in Figure 2.1:

![Flow of extracting centerlines of roadways from an aerial Google Map image](image)

Figure 3.1 Flow of extracting centerlines of roadways from an aerial Google Map image

Each step is described in details as follows:

### 3.2.1 Image preprocessing

The purpose of image preprocessing is to extract desired features (the road network) from the aerial image and categorize them into three sub-categories as introduced above. This step prepares the aerial image for further steps.

The original aerial RGB image (see Figure 3.2) is first converted to gray scale (see Figure 3.3). Roadways are then extracted and divided into sub-categories on the basis of intensities of pixels. Each sub-group of extracted roadways is stored as a separate gray scale image. The program then performs operations to convert each gray scale image into a binary image (Figure 3.4). In the binary image, only pixels that represent roadways are assigned the value 1 (white), all other pixels are assigned the value 0 (black). As a result, only desired features are extracted from the aerial image.

![Original aerial Google Map Image](image)

Figure 3.2 Original aerial Google Map Image
Binary images we obtained so far are not ready to be used as inputs of the extraction process. Although Google Map eliminates disturbances such as environmental objects, it does introduce some other undesired features such as label texts which are labeled right on top of roadways. Label texts are considered as noises in our case since they introduce disturbances when performing the next step: the thinning process. Texts labeled on roadways are some 0 pixels which appear like holes on binary shapes. Converting the original RGB image into a binary image does not remove these noises.
As described, only pixels that represent roadway surfaces are assigned value 1 in the binary image. All other pixels, including pixels that represent labels are assigned value 0. Therefore, in the binary image, text pixels are just some 0 pixels surrounded by many other 1 pixels. Let’s select a 3-by-3 square with a text pixel as the center. It’s obvious that most of its neighbors are 1 pixels. Therefore, by performing a morphological operation, those noises can be detected and eliminated by converting them to 1. In that way, text pixels are converted to roadway pixels and hence noises are removed.
The morphological operation applied in this study is described as: set a pixel to 1 if 5 or more of its neighbors in a 3-by-3 square are 1s. This majority morphological operation can be applied for a few iterations to achieve the best noise-reduction result.

3.2.2 Roadways Thinning
The outputs of the image preprocessing step are binary images of three types of roadways. Since the goal is to extract roadway centerlines which are line data that represent the geographic centerline of roads, it’s necessary to convert binary shapes which represent roadways to 1-pixel wide lines which represent road centerlines. A thinning process is applied to the binary image with the following properties taken into account: 1) end points are not removed, 2) connectivity is preserved, and 3) no excessive erosion of the region is caused. MATLAB does provide a perfect function to thin binary objects, the algorithm behind is not going to be described in this paper.

All roadways are reduced to 1-pixel wide lines after the thinning process is applied, as shown in Figure 3.5. So far, we’ve extracted the road network from each binary image.

![Tinned image of interstate highway](image)

a. Tinned image of interstate highway
3.2.3 Intersection Detection

Local streets or state highways often meet or cross with others. A local street is divided into several segments by intersections. Semantically, this street is considered as a single identity because all road segments bear the same street name. However, geometrically and topologically, those road segments are not connected with each other directly. Instead they’re jointed at intersections. Trying to extract a semantic street from an aerial image is quite difficult since it’s hard to train computer programs to learn which road segments have the same street name in the real world. In another word, the aerial image
contains no interpretable information at the semantic level. Although labels are placed on the map to indicate the names of roadways, it’s difficult for computer programs to learn that information. In relative terms, it’s much easier for computer programs to extract the centerline of each roadway segment.

As a result, this study considers each intersection as a break point. Instead of trying to extract a semantic continuous road, the program tries to extract all road segments which break at intersections or terminate at roadway ends.

Roadways are connected to each other at intersections. In the thinned binary images, intersections are shrunk to single pixels. As a result, in the thinned images, lines that represent centerlines of road segments are intersected with each other. Each road segment is represented by a 1-pixel wide line which is formed by a series of connected pixels. In order to extract the centerline of each road segment, the program has to select the series of connected pixels from all other pixels. However, several road segments are connected to one intersection, and consequently, several lines are connected to one intersection pixel in the thinned image. When performing the operation which returns connected components in the binary image, the entire road network instead of a single road segment will be returned.

Therefore, to extract the centerline of each road segment, the program has to first detect intersections from the binary image. Since intersections are road junctions where two or more roads meet, at least three road segments are connected to an intersection. After thinning, an intersection is reduced to a single pixel which has 3 or more pixels in its 3-by-3 neighborhood that are 1s. Conversely, in the thinned road network, a pixel indicates an intersection if 3 or more pixels in its 3-by-3 neighborhood are 1s. Therefore, to detect intersections, the program detects all pixels which have 3 or more 1 pixels in its 3-by-3 neighborhood. As illustrated in Figure 3.6, each intersection is detected and marked with a green “+” symbol. With intersections detected, the program breaks the connections between 1-pixel wide lines by setting the intersection pixels to 0. In this way, the road network is broken at intersections, and each road
segment can be separated out from the road network by returning all connected components in the binary images.

Figure 3.6 Detected Intersections

3.2.4 Roadway Segments Extraction
After intersection detection, road segments are not connected with each other anymore. Each road segment is separated from the road network. Therefore, by performing an operation that returns connected components in the binary image, all road segments are extracted. Matlab provides a function that returns all connected components with a custom connectivity. By adjusting the connectivity value, road segments can be all detected.
Figure 3.7 shows the result of extracting each road segment from the local street network.

![Extracted road segments](image)

**Figure 3.7 Extracted road segments**

### 3.2.5 Curve Fitting

The centerline of a road segment is extracted from the binary image as a series of connected pixels, or connected points. The series of connected pixels are not ready for 3-D modeling because they’re some scattered data points. These points are merely results of image processing and hence by connecting these points, the smoothness of the produced line is not guaranteed. Therefore, a curve fitting process is needed to construct a smooth curve to approximate the series of points.

Many prior works have been done in the area of approximating road centerlines as curves. (Oza, 2006) proposed a method to model the centerline of a road as a Catmull-Rom spline, which is defined by a series of control points. Roadways appeared smooth and natural as a result. More importantly, the layout of the roadway system became an integral. Hence, by moving a control point to a new desired position, a road is changed consequently. Reconfiguring virtual environments is therefore simplified and fastened tremendously.

However, such a technique shows its weakness when trying to reproduce real-world roadways when building geo-specific virtual environments. Given a curved road in the old urban areas to be simulated in the virtual environment, specifying the series of control points that defines a spline to approximate the centerline of this curved road can be exhausting. Although the Catmull-Rom spline is guaranteed to pass
through all of the control points, it’s hard to have a control on the exact shape of its interpolated segments unless the curve could be mathematically represented as a normal function. As the complexity of the curve increases, more control points are needed or a number of simple sub-splines have to be combined as to generate an assembled spline to fit the curve. An increasing number of control points simply cause increasing difficulties in finding the correct series of control points. An increasing number of sub-splines pose a hardship on the designer to divide the curve into a sequence of connected sub-splines appropriately. In a conclusion, it’s quite difficult to approximate a complicate curve with a small error tolerance using Catmull-Rom splines without making improvements.

(Willemsen, Kerney, & Wang, 2003) represented roads as ribbons and modeled 3-D centerlines as segments, which are hierarchical sets of straight lines, circular curves and spirals. Parameterized spline curves were chosen to be the mathematic representation for ribbon axis (centerlines) and a simple approach was developed to parameterize spline curves by approximate arc-length. Point $p$ is represented in Cartesian coordinates as $C_p = (X_p, Y_p, Z_p)$ and represented in ribbon coordinates as $R_p = (D_p, O_p, L_p)$.

Constructing splines from straight lines, circular arcs, and spirals is an efficient approach to facilitate modeling virtual roadways. It’s efficient because it follows transportation guidelines. Centerlines of modern roads in the real world consist of straight lines, arcs, and clothoids as transition curves. A clothoid is a spiral whose curvature is a linear function of its arc length, as shown in Figure 3.8. It’s widely used in highway and railway designs nowadays because it provides a desirable smooth transition between straight lines and circular arc. Also, drivers feel more comfortable when driving on a clothoid because of a constant rate of angular acceleration.
As long as splines are divided into segments (straight lines, arcs & spirals), designers are more capable of modeling longer and more complex roadways.

However, roads, especially those in old urban areas, do not always comply with the design standards. It’s impossible to divide the centerlines of those roads into standard segments as we stated before. Rather, an approximation approach has to be implemented to approximate centerlines. A sequence of standard segments has to be combined to fit centerlines of roads with a small tolerance of error.

(Pavlidis, 1983) proposed to fit curves with conic splines. Though the result is visually smooth, it’s cannot be represented systematically in the database. Also, they have a relatively poor control over the fitting result.

(McGrae & Singh, 2008) introduced an algorithm to fit a sequence of $G^2$ clothoid segments to polyline stroke data. They first computed the discrete curvature of the stroke as a function of arc-length. After that, with control over the tradeoff between fitting error and the number of linear pieces, they fit a piecewise approximation to the curvature function. Each linear piece defined a straight line, circular curve or clothoid segment. These segments are then assembled into a single composite curve. The last step was to apply a single 2D rigid transform that aligns this composite curve with the stroke to minimize the error of
the stroke from the transformed curve. The error was formulated as a weighted least square optimization problem, so that the transformation can be conducted efficiently. Figure 3.9 shows the steps of fitting a stroke with a sequence of clothoid segments.

![Clothoid fitting](from (McGrae & Singh, 2008))

The fitting result appears good. However, the process of fitting the centerline with a sequence of clothoid segments is time-consuming.

Linear curve fitting is adopted in this study. The produced fitting line is a multivariate function as shown below:

\[ p(x) = p_1x^n + p_2x^{n-1} + \cdots + p_nx + p_{n+1} \]

The reason why linear curve fitting can be used to fit curves to the road segment data is that each segment extracted in this study is relatively short and hence has a simple geometric shape.

Figure 3.10 shows the fitted curve of the highway centerline in Figure 3.5(a). The fitted curve: the blue curve goes smoothly through the points.
3.3 Roadway Elevations

The roadway centerlines we got so far are restricted to the 2D plane and hence flat. The elevation data of roadways, which is indispensible to reproductions of real roadways in virtual environments, is still missing. Elevation data of a geographic location is not contained in the Google Map images and we have to get it from other resources.

There are a great number of resources available for use out there. In this project, we're trying to demonstrate our techniques by reproducing roadways in the State of Massachusetts. Therefore, the database resource we use is “MassGIS”. “MassGIS” is a free statewide resource for geospatial technology and data. The datum for the MassGIS database is North Amercian Datum 1983 (NAD83). The data are registered to the Massachusetts State Plane Coordinate System, Mainland Zone. Units are meters.

Digital Terrain Model (DTM) files are available at http://www.mass.gov/mgis/dtm.htm. A DTM file contains five columns of data: 1) “X” and “Y” coordinates, which are horizontal coordinates in the NAD83 Massachusetts State Plane coordinate system; 2) “Z” elevation values in meters (NAVD88); 3) a
keyword used by the Arc/INFO software for the production of triangulated irregular networks; and 4) a numeric value that indicates the type of point.

DTM files are entitled by the Orthophoto Quad Index which is a 4000 by 4000 meter grid in MA State Plane coordinates.

To get the elevations of an area in MA, one can first locate the area by Orthophoto Quad Index and then download the corresponding DTM file from ftp://data.massgis.state.ma.us/pub/dtm/ . Figure 1.10 shows the 3-D plot of an area in Belmont, MA with the elevation data obtained from Mass GIS.

![3-D plot of a geographic area in Belmont, MA](image)

Figure 3.11 3-D plot of a geographic area in Belmont, MA

Now that elevation data is available in an area, to get elevation data of a roadway in this area, a few more steps are needed.

Step 1 It’s important to realize that we can get the elevation value of a point from the DTM files only if we know the “X” and “Y” coordinates of that point in the NAD83 Massachusetts State Plane coordinate system.
With the 2-D roadway centerlines extracted in section 3.2, we have only got the pixel coordinates of centerlines. The pixel coordinates are of no use when we try to obtain the corresponding elevation value from the DTM files. It’s natural to think that pixel coordinates have to be converted to NAD83 coordinates first.

It’s not possible to convert pixel coordinates to NAD83 coordinates directly. However, it’s fortunate that the internal coordinate system of Google Map is geographic coordinates (longitude/latitude). And it’s easy to convert geographic coordinates to NAD83 coordinates.

Therefore, pixel coordinates have to be converted to geographic coordinates first, in order to be further converted to NAD83 coordinates.

Back to the beginning of the centerline extraction process, we have to download Google Map Images with specific geographic coordinates so that we know exactly which area the downloaded Google Map image covers in the real-world.

Since Google Map is not NAD83 coordinates based, we cannot download a Google Map image by specifying the NAD83 coordinates. Instead, we download Google Map images by specifying the Longitude/Latitude coordinates of the left top corner and the right bottom corner of the area that we want to capture as an image.

As a result, in the captured Google Map image, the mapping of pixel coordinates to longitude/latitude coordinates is as given in:

\[
\begin{align*}
\text{long}_x &= (\text{long}_r - \text{long}_l) \times p_x \\
\text{long}_y &= (\text{lat}_r - \text{lat}_l) \times p_y
\end{align*}
\]

Equation 3.1

And the next step is just to convert longitude/latitude coordinates to NAD83 coordinates. The mapping of pixel coordinates to NAD83 coordinates is illustrated as in equation:
\[
\begin{align*}
N_x &= Long_l + p_x * (Long_r - Long_l)/(w) \\
N_y &= Lat_t + p_y * (Lat_b - Lat_t)/(h)
\end{align*}
\]

Equation 3.2

So far, we’ve got the horizontal “X” and “Y” coordinates of a roadway centerline in the NAD83 Massachusetts State Plane coordinate system (see Figure 3.12).

Figure 3.12 Roadway centerline in the NAD83 MA State Plane coordinate System

Step 2 Load the DTM file of the corresponding area and obtain the elevation value of any point in the captured area by generating the 3D surface with interpolation, and returning \( z \) values of any given \( x, y \) coordinates.

Figure 3.13 shows the interpolated 3D surface of an area along Route 2 using DTM files and Figure 3.14 shows the produced 3-D centerline of the highway which used to be represented by the 2-D centerline in Figure 3.12.
Figure 3.13 Interpolated 3d surface of the area along Route 2 in MA

Figure 3.14 3-D centerline of Route 2 in MA
3.4 Converting 3-D road centerlines to 3-D geometric roadway models

With the 3-D centerlines produced by far, we’re ready to model 3-D roadways in Cheetah 3D with scripts. Given the 3-D centerline and the width of a roadway, Cheetah 3D can produce a 3-D roadway model easily. However, constructing a roadway as an integral 3-D model is undesired in this study for following reasons:

1) Constructing and updating the roadways could be time-consuming. It requires a lot works when texturing 3-D roadways. A new texture has to be prepared whenever the roadway surface is different. For example, we have to prepare a texture for a roadway with four lanes and another for a roadway with six lanes.

2) It poses hardship on designers when modeling the road network topologically, which will be described in Chapter 2. Since a roadway is constructed as a whole, it’s difficult for the program to tell which lane an autonomous vehicles is traveling on. We surely can tell which lane the vehicle is in by calculating the deviation of the vehicle to the roadway centerline in real-time, but it’s costive and hence not preferable.

To create a designer-friendly virtual environment and to enable autonomous traffics in the virtual environment, this project constructs roadways with lanes. Each roadway is a combination of a series of parallel aligned lanes. Here, a lane can be either a carriage way or a lane marking line. That’s to say, for example, a double yellow line is considered as a lane in this study and is constructed separately.

The key for generating the centerline of a lane is to create a line that’s parallel to the roadway centerline with a distance $r$. By using various $r$ values, several lanes are laid side by side to create a roadway, which is done pretty much automatically by Cheetah 3D since objects are placed according to their global coordinates.
Texturing becomes easy since each roadway is divided into multiple lanes. All we need is to apply different textures on different types of lane elements. Figure 3.15 and Figure 3.16 shows the generated 3D model of Route 2 in this study.

Figure 3.15 Top view of the 3D model of route 2

Figure 3.16 3D view of the 3D model of route 2

So far, we’ve had the ability to reproduce real-world road networks in virtual environments with affordable efforts and high reality.

However, it’s far less than enough to have only some static 3-D objects: roadways, buildings, trees, etc. presented in the virtual environment. Autonomous traffic which provides interactions the users will experience is essential to a realistic and immersive virtual environment.
With the geometric modeling only, it’s only possible to manually guide a vehicle through the environment by having a script that describes every single movement of the vehicle. As the number of vehicles increase, much more works are needed for designers in order to maintain the traffic in the virtual environment.

It’s highly undesired and hence this study proposes another level of modeling, as an important step to implement autonomous traffic into the virtual environment. This further level of modeling is named as Topological Modeling in this study.
4 Topological Modeling

Virtual traffic formed by simulated autonomous vehicles plays an important role in giving a sense of realism as well as designing scenarios. Unexpected variations are introduced so that scenarios are not predefined and ideal (Edward, Lourdeaux, Barthès, Lenne, & Burkhardt, 2008). Together with the 3D roadway network modeled in Chapter 3, intelligent vehicles that behave autonomously produce an interactive and immersive virtual driving simulation.

Autonomous in this research is defined as follows: 1) vehicles are capable of perceiving the environment (Thomas & Donikian, 2000); 2) vehicles communicate with other virtual agents in the environment (other vehicles, human, etc.); and 3) vehicles make decisions and take actions with regard to the instant driving conditions.

Traditionally, interactions between autonomous vehicles and the environment were relatively simple. Autonomous vehicles have minimal capabilities of avoiding environmental obstacles. Vehicles demonstrate no abilities of obeying traffic rules or autonomously planning routes.

Communications among autonomous vehicles are critical to high complexity virtual traffic and are missing in most driving simulations as well. Vehicles have to be aware of others’ behaviors, in order to make decisions and take actions accordingly. An intersection is a good example of the scenarios where vehicles need to communicate with each other before taking any actions to avoid collisions.

In order to model virtual vehicles as live beings, in another word, with the ability to perceive and communicate, this research implements a second level of road network modeling in addition to geometric modeling described in Chapter 3, namely topological modeling. This level of modeling aims to provide virtual agents with abundant information so that autonomy is enabled.

4.1 Topological Structures

Geometric modeling defines the spatial structure of roadways/intersections, while Topological Modeling defines the connectivity of roadways/intersections, the traffic rules applied, environmental objects, etc.
As reviewed in section 2.2, in the study done by Donikian, the topological information is represented as a graph that interconnects all the road sections of the road network. Each roads section is a node in the graph. Each node has number of connections that equals to number of In/Out points. Error! Reference source not found. shows a road network and its corresponding topological information. (Seo & Choi, 1998) represented a road network as a graph $G = (N, L)$, where $N$ is a set of nodes ($v_1, v_2, v_3, \ldots, v_n$) and $L$ is a set of links ($e_1, e_2, e_3, \ldots, e_m$). The chain from the start point $o$ to the end point $d$ is a sequence of nodes and links ($o = v_{o1}, v_{o2}, v_{o3}+l, v_{o4}+l, \ldots, v_{od} = d$).

A typical city road network is then represented as the grid shown in Figure 4.1.

![Figure 4.1 Representation of road network (from (Seo & Choi, 1998))](image)

Each node in Figure 4.1 represents an intersection and each link represents a section of roads.

(Thomas & Donikian, 2000) provided a topological structure of the city to navigate dynamic objects from one street to another crossing an intersection (see Figure 4.2). This graph is a strongly connected planar graph. In Figure 4.2, a crossroad composed by four buildings (B1, B2, B3, B4), two parks (P1, P2) and the road network is presented. Edges link adjacent spaces and contain both geometrical and symbolic information. In addition to the topological structure of the road network, they also provided topological structures at a more detailed level to guide dynamic objects specifically from one point to another in the town. Figure 4.3 shows an example of a road section composed of two side-walks ($\square \square$, $\square \square$), a car lane ($\square$) and a pedestrian crossing ($\square$). In the corresponding topological graph, there are two paths to go from topological node A to topological node B.
The idea of having topological structures at different levels enables designers to plan routes for dynamic objects very specifically. For example, it’s possible for designers to create a lane change scenario easily. However, in their study, there is no inter-connection between topological structures at two different levels. For example, there is no connection between the topological structures shown in Figure 4.2 and Figure 4.3. And therefore, it’s not possible to plan routes for dynamic objects globally and specifically at the same time.

In this project, we propose two inter-connected levels of topological structure: 1) global topological structure and 2) local topological structure.

Figure 4.2 Geometry and Topology Graph of a Road Network (from (Thomas & Donikian, 2000))
4.1.1 Global Topological Structure

The global topological structure is actually a topological representation of the 3D road network constructed in Chapter 3. Nodes of the global topological structure are components of the road network: road segments, intersections, ramps, bridges, etc. As shown in Figure 4.5, each component in the geometrical road network shown in Figure 4.4 is represented as a node in the topological structure, and is
given a unique name and unique ID. Arrowed edges link adjacent nodes and represent the connectivity/traffic information.

Figure 4.5 Topological representation of the 3D road network shown in figure 4.5

4.1.2 Local Topological Structure

With the global topological structure, dynamic agents are provided information about which road/intersection of the road network they are currently on. However, this is not enough for agents to behave autonomously. They have to get much more knowledge about the driving environment in order to take the correction actions. For example, a simulated vehicle has to at least know which lane it is currently in, before making a lane change.
Therefore, for each global topological node, there is a local topological structure associated. For example, a roadway node in the global topological structure has a local topological structure which is composed of nodes that represents lanes, sidewalks, pedestrian crossing, etc.

Figure 4.6 shows the geometrical and topological structure of the road segment “massave_00”, which is a node in the global topological structure presented in Figure 4.5.

In this study, a combination of the global and local topological node is used to locate a dynamic agent in the road network. For example, vehicle “01” is currently in lane “2” of “Beacon_00”.

Now that simulated vehicles are provided their locations in the road network, there is still more environmental information for them to perceive to conduct even a simple activity. For example, when approaching an intersection, it’s necessary for a vehicle to read the traffic signage and understand its semantic information. Otherwise, the vehicle may cross the intersection even with a red traffic light on. Another example of simple activities is obstacle avoidance. Vehicles may crash into obstacles on their ways if they were blind to the environment. As in real-life driving, drivers only pay their attentions to
environmental objects that are driving-related (DR). A public phone or a billboard is normally of no concern to a driver, and hence is non-driving-related (NDR). Furthermore, even if it’s a DR object, not all drivers have to perceive it in order to drive safely. A good example would be the traffic light. A traffic light normally only regulates objects on the road segment whose traffic direction faces the traffic light. For example, as show in Figure 4.7, vehicle A which is driving toward Traffic light group1 has to check its status continuously to understand the current traffic rule applied on the intersection. While vehicle B and vehicle C only needs to check the status of Traffic light group 2 instead.

![Figure 4.7 Traffic lights and traffic](image)

As a result, it’s only necessary to associate a DR environmental object to some roadways/lanes/intersections in the road network. Topologically, it’s to associate static environmental objects to certain topological nodes, at both global and local levels.

In this study, static objects that are associated with topological nodes are listed as below:

1) Traffic lights. When a vehicle travels on a road segment, is there any traffic light that it should obey? If there is, which traffic light?

2) Traffic signs. Is there any stop signs, yield signs, etc. on the sides of the road segment? If there is, associate these traffic signs with the topological node.
3) Lane Markings. Is there any lane markings indicating any traffic rules?

4) Environmental obstacles. Is there any obstacle on the road segment? Such as roadwork signs, etc.

After associating environmental objects with topological nodes, vehicles that travel on the geometrical representation of topological nodes will start perceiving associated environmental information.

So far, we’ve got both geometrical and topological structures of the road network. There are also environmental objects associated with the road network both geometrically and topologically. Geometric info about the road network, topological connectivity and the association of static objects and topological nodes all have to be stored somewhere and be ready for use by vehicles whenever requested in an efficient manner. As the scale of the virtual environment grow, the amount of data to store increases dramatically. Therefore, it’s key to come up with an approach to store, maintain and update environmental data efficiently and systematically.

A relational database is proposed in this study to store geometric and topological information of all objects presented in the virtual environment. There are a number of advantages of using a database to manage data. In this study, the relational database is preferred mainly due to three merits: 1) systematic storage, 2) table joins and 3) cater for future requirements. The ability to join tables makes it possible for the application to associate objects with topological nodes. The fact that the database caters for future requirements is important in terms of high scalability of the virtual application.

4.1.3 Store Data into the Database

As stated in 4.1.2, both global and local topological structures are stored into a database. Roadways, Intersections, Ramps, etc. are global topological nodes. Each roadway has its local topological structure. Lanes, sidewalks, isolated belts, etc. are nodes of the local topological structure. Each DR environmental object is associated with a topological node (global or local).

The database also stores all NDR environmental objects that are presented in the virtual environment, such as Vegetation, Buildings, etc. Figure 4.8 shows the structure of the Database.
As illustrated in Figure 4.8, the database creates 10 tables listed below to store everything presented in the virtual environment. These 10 tables are grouped into 4 categories.

**a. Road Network**

1) roadways

The “roadways” table stores all road segments in the Road Network. Each table record is assigned a unique ID. As shown in Figure 4.9, other than key, the “roadways” table has 11 attributes, including “roadtype” which defines the types of road segments, (xrot, yrot, zrot) which are the eulerAngles rotations of the road segment around the x, y, z axes, (xscale, yscale, zscale) which are the scales of the road segment along the x, y, z axes, (xpos, ypos, zpos) which are the x, y, z coordinates of the geometric center of the road segment in the global Cartesian coordinate system, and the name of the road segment which is unique as well.
2) intersections

The “intersections” table stores all intersections in the Road Network (see Figure 4.10). Each intersection has a unique ID and a unique name. Other than the ID and the name, the table stores (xpos, ypos, zpos), (xrot, yrot, zrot) and (xscale, yscale, zscale) of each intersection.

3) bridges

The “Bridges” table stores all bridges in the Road Network (see Figure 4.10). Each intersection has a unique ID and a unique name. Other than the ID and the name, the table contains (xposition, yposition, zposition) of the geometric centerline of each intersection.

4) lanes (including sidewalks, carriageways, isolated belts, etc.)

The “lanes” table stores all lanes of all road segments in the road network. Five attributes are included in the table: 1) “id”, an uniqued id assigned to that lane 2) roadid, which is a foreign key that refers to the “id” column of the “roadways” table. This indicates which road segment that “lane” belongs to; 3) lanetype, which defines the type of the lane, e.g. roadlane, carriageway,
sidewalk, parklane, etc. 4) width, which is the width of the lane in meters; 5) laneDev, which is the deviation of the lane centerline to the road segment centerline; 6) trafficDirection, which defines the direction of the traffic in that lane.

![Table: lanes](image)

**Figure 4.11 “lanes” table**

b. DR Environmental Objects

5) trafficlights

The “trafficlights” table (see Figure 4.12) stores all traffic lights presented in the virtual environment. As discussed above, each traffic light is associated with a topological node, which in our case is the topological representation of an intersection. And the “trafficlights” table stores that information by adding an attribute “intersection”. This attribute specifies which intersection the traffic light is associated with. Furthermore, even the traffic light is associated to an intersection, it has to be specified that which traffic direction the traffic light is regulating. This is done by adding the column “yrot” which is the rotation of the traffic light along the y axis. Only vehicles that approach the associated intersection and face the traffic light are being regulated.
6) Traffic signs

The “trafficsigns” table (see Figure 4.13) stores all traffic signs presented in the virtual environment. Since traffic signs are driving related, they’re associated with road segments. This is done by adding the column “roadid”. Also, there is another column “signtype” which indicates the type of the traffic sign. The record shown in Figure 4.13 is a stop sign which is associated with the road segment “Beacon_00”.

7) Lane markings

It’s obvious that lane markings are associated with a lane.

8) Obstacles

Each obstacle has to be associated with not only a road segment but also a lane because only vehicles in that lane would have to avoid the obstacle. In another word, each obstacle has to be associated with not only a global topological node but also a local topological node. In the “obstacles” table, two attributes “roadname” and “lane” are added to specify the associated road segment and lane.
c. NDR Environmental Objects

The database also stores all NDR environmental objects. A “vegetation” table is created to store positions of all vegetation, and a table “building” is created to store positions of all buildings.

These NDR objects are not affecting traffic and hence are not associated with any topological nodes.

9) Vegetation

10) Buildings

4.1.4 Database Preparation, Update

To become designer-friendly, the preparation of the database must not be laborious and more importantly, modifications to the database should be facilitated. Designers may have to update and reconfigure the
virtual environment quite often to meet changing needs. Consequently, the database needs to be updated often as well without requiring many human efforts. To achieve the goal of building a designer-friendly virtual environment, the preparation of the database is automated with a JavaScript program running in Unity 3D, the ID platform in this project.

The process of preparing and modifying the database tables in this research is described in details as follows:

1) Generate a tree structure for each category of objects.

In Unity, all objects are GameObjects. A few GameObjects can be grouped into a parent GameObject. As shown in Figure 4.17, all road segments of “Beacon Street” are grouped into a parent GameObject named “Beacon”. Furthermore, all roadways are grouped into a root GameObject named “roadways”. Similarly, all GameObjects in the virtual environment (road segments, intersections, vehicles, traffic signage, trees, obstacles, buildings, etc.) can be grouped into some parent GameObjects with respect to their categories. A tree structure is the perfect way of representing the hierarchical nature of these GameObjects. An example of the tree structure of roadways is given in Figure 4.18. The GameObject “roadways” is the root GameObject.

![Tree structure example](image)

Figure 4.17 Objects in the same class are grouped together under a parent Object
Every GameObject in Unity has a “transform” attached. Transform is used in Unity to store and manipulate the position, rotation and scale of the object. And every transform can have a parent. Therefore, transforms of all child GameObjects become children of the parent GameObject’s transform.

2) A JavaScript program is written to automate the process of creating new tables, inserting new records to tables, update existing records and delete records from tables. The program takes the root transforms as the input and steps through its children transforms. A breadth first traversal is performed over each tree. However, instead of explores as far as possible along each branch, the breadth first traversal only explores till it reaches the specified level. The code of stepping through the roadway tree structure and add new records to the “roadways” table is given as below:
Adding an input to the script in Unity is quite simple. Just drag the parent GameObject over to the reference of the variable in the Inspector, as shown in.

3) An Object can be a child of another Object and a parent of some other Objects at the same time.

For example, as shown in Figure 4.19, Object “Beacon_00” is a child of Object “Beacon”, and meanwhile, is the parent of a group of child objects: “parklane”, “roadlane”, etc. This is understandable because roadways are composed of lanes, sidewalks, etc. Therefore, when adding all lanes of all roadways to the database, designers just have to embed another loop inside the loop given in section 1).
4) Inserting records into the database is a one-time task. When the construction of the virtual environment is done, run the JavaScript once to store information of the virtual environment into the database. When updating the database after designers made changes to the virtual environment, e.g. added a new roadway, designers just have to select the “Update Roadway”
option as illustrated in Figure 4.19 and run the Javascript once to update the “roadways” table in the database.

With the process introduced above, preparing and updating the database is simplified tremendously and requires little manual works. Designers are free to make any changes to the virtual environment without spending time updating the database after.

Now that both topological and geometrical information about the environment is stored into the database, the next step is to load the information from the database and provide it for dynamics agents’ uses. The Unity program typically has to render the virtual environment at least 30 times per second in order for users to not see artifacts. Therefore, having the Unity program read information from the database in real-time is highly undesired. It could create a high level of load on the Unity program as well as the database, and hence lower the rendering frame rate significantly. In this study, the Unity program loads information from the database into in-memory arrays only once when the program initializes. When the Unity program runs, it will only work with these in-memory arrays in real-time. More about in-memory arrays will be discussed later in this chapter.

4.2 Route Planning

With both geometric and topological information of the virtual environment stored in the database, the next step is to implement autonomous traffic.

In order for autonomous vehicles to travel autonomously in the virtual environment, route planning is the very first step involved.

Planning routes of autonomous vehicles can be time-consuming if no dynamic route planning method is proposed, especially when the number of autonomous vehicles in the environment increases. A traditional approach to plan routes for autonomous vehicles is to manually script out each segment of a vehicles’ route beforehand and have the vehicle follow the script at runtime. Designers have to elaborate every single move of the vehicle so that it behaves as desired. This intuitive approach does not only eliminate the possibility of enabling autonomy in simulated vehicles, but also introduces great hardships
to designers. Any modifications to the geometrical structures of the underlying road network require manual updates on vehicles’ script as well. As the density of simulated traffic increases, a great amount of time is needed to create and update scripts of simulated agents. It is also a very difficult task to map out all the appropriate turns, lane changes, speed acceleration and decelerations, etc. for vehicles to traverse a planned route, when the scale the environment becomes large.

To avoid these hardships, [Oza,A., 2006] proposed to model roadways with Catmull-Rom splines and associated each autonomous vehicle with them. By giving starting and ending control nodes, an autonomous vehicle will navigate on its own. However, this resulted in difficulties when road intersected and when an autonomous vehicle was required to follow a path where no road spline exists. For example, it’s difficult to have an autonomous vehicle make a lane change.

Based on Achal’s works, [Yin, Z. 2008] introduced another layer of splines which is superimposed to the road spline layer to guide autonomous vehicles through the environment. Because of the implementation of these additional splines, intersections are no longer a problem. Also, by imposing splines, autonomous vehicles can follow complicated paths such as lane changes on the road.

However, both methods require users to manually plant control nodes to generate the spline for autonomous vehicles to follow. As the spline becomes longer and more irregular, it could be almost impossible for programmers to find the sequence of control nodes to generate the spline exactly as expected. Even if possible, it requires a huge amount of time and efforts.

This research proposes a much more robust way of planning routes for autonomous vehicles. Instead of elaborately script out each leg of a vehicle’s route or specifying a sequence of points for the vehicle to follow, only a sequence of topological nodes is assigned to form the route that the vehicle is going to travel on. When designers add a new autonomous vehicle to the virtual environment, all they have to do is to specify the sequence of topological nodes that the vehicle will travel through.

For example, a route of a vehicle could be defined as {Beacon_01, Beacon_02, massave_00}. This means the vehicle will start from some point on road segment “Beacon_01”, travel through “Beacon_02” and
arrive at some point on “massave_00”. Till now, local details are still missing and are indispensable to the generation of the specific route that the vehicle is going to traverse.

Other than global topological nodes (names of roadways and intersections), to guide an autonomous vehicle from point A to point B in the virtual environment, the route has to also include local topological nodes to indicate which lane the autonomous vehicle is in. Furthermore, to specify the exact point that a vehicle is at in that lane, a field “Partition” is used. “Partition” is the portion of distance between two endpoints of the lane. Last, a target velocity that the vehicle travels at is associated with each group of (global nodes, local nodes, partition).

The combination of global nodes (GN), local nodes (LN), partition (P) locates vehicles accurately and defines the route of autonomous vehicles specifically. Routes of all autonomous vehicles are stored into the table “vehiclepath” in the database introduced in this study. To reconfigure the route of an autonomous vehicle, designers only have to update its records in the database table.

Figure 4.20 shows the planned route for vehicle “V05”. Each autonomous vehicle has a unique ID assigned.

As a result, the route of “V05” is formed as

![Figure 4.21 Global Route of V05 represented by pairs of topological nodes](image-url)
4.2.1 Endpoints & Paths

Each record in the “vehiclepath” table actually defines an endpoint along the path of the vehicle. With these records, the designer only specifies a few endpoints that the vehicle has to travel through. However, how the vehicle is going to travel from one endpoint to the next endpoint is undefined yet. The program has to do the work for designers.

So far, endpoints are represented by pairs of \{GN, LN, P\}, which are not interpretable when placing an object in the virtual environment. Instead, \{x, y, z\} coordinates in the global Cartesian coordinates system have to be given. The first step there is to compute the Cartesian coordinates of the endpoints. In another word, to convert \{GN, LN, P\} to \{x, y, z\}.

As specified in the “vehiclepath” table, the route of autonomous vehicle “V05” is defined by 4 endpoints and hence is grouped by 3 paths. Take the first path as an example. The designer specifies that “V05” starts from the first endpoint \{roadway:Beacon_00, lane:0, partition:0.3\}, travels at 20mph till it reaches the second endpoint \{roadway:Beacon_00, lane:0, partition:1.0\}.

Converting \{GN, LN,P\} to \{x,y,z\} is quite simple since all geometrical data about road segments and lanes are available. As illustrated in Figure 4.22, the endpoints of this path segment is computed based on the lane number, widths of lanes, rotation of the roadway, position of the roadway and the value in the “Partition” column, which represents the portion of distance between two roadway ends. All information needed to compute the global Cartesian coordinates of endpoints is stored in the database and has been loaded into in-memory arrays when the program initialized.
Figure 4.22 endpoints of path “1”

Now that endpoints of path “1” are computed, the next step is to navigate the vehicle from endpoint 1 to endpoint 2. Two cases are considered: 1) “Beacon_00” is a straight road segment and 2) “Beacon_00” is a curved road segment.

If “Beacon_00” is a straight roadway, only two endpoints are needed to define the straight path segment, one at each end of the segment.

The path segment is then generated by increasing the interpolation step \( t \), which is the portion of the distance between two path segment ends and is clamped to \([0...1]\), as illustrated in Figure 4.23.

Figure 4.23 Interpolation between two endpoints along a straight path segment

When “Beacon_00” is a curved roadway, the first path segment of “V01” is curved as well. As described in Chapter 3, the centerline of a curved roadway is approximated by two polynomial functions

\[
\begin{align*}
\vec{p}_0 &= \vec{p}_0(t) \\
\vec{p}_1 &= \vec{p}_1(t)
\end{align*}
\]

where \( z \) is the elevation. Therefore, a curved path segment can still be computed by increasing the interpolation step \( t \).
However, it’s actually not a robust way of navigating autonomous vehicles along curved roadways because of the following two reasons:

1) The polynomial functions are functions of variable $\phi$ instead of $t$. With $t$ increasing at a constant rate during each iteration, the program has to compute the corresponding value of $\phi$ in real-time, and pass the value of variable $\phi$ to $\phi(\phi)$ and $\phi(\phi)$, in order to get the Cartesian coordinates of point “Pt” in three dimensions. Too many computations are involved, which would affect the performance of the program and pose difficulties for designers, especially when the degree of the polynomial function is high.

2) The program has to import all polynomial functions of all curved roadways beforehand. It not only creates additional works for designers when preparing the database, but also increases the load of the program.

Hence, in this research, an alternative way is proposed to generate the series of endpoints for a curved path segment.
Before describing the approach, it's worth mentioning that no matter how smooth a 3-D curved roadway model appears to be, it's a sequence of connected straight segments. Figure 4.25 shows an example of a curved roadway.

![Figure 4.25 A 3-D curved roadway](image)

Similarly, a curved path segment can be approximated by a sequence of connected straight path segments. Two endpoints $\square[\square]$ and $\square[\square + l]$ define a straight sub-path $\square[\square]$. The curve path segment is then formed by simply connecting the sequence of $\square[\square]$.

The roadway is rendered as a triangle strip. There are pairs of points that represent the left and right coordinates along the boundary of the roadway. Therefore, the series of endpoints of a curved path segment could simply be computed as

$$\square[\square] = \square[\square] + (\square[\square] - \square[\square]) \ast (\square[\square] + 0.5)$$

Equation 4.1
Where: \( P_0 \) is the point on the left boundary, \( P_1 \) is the point on the right boundary, \( \delta \) is the lane deviation, computed based on the lane number specified in the database, and \( \theta \) is the width of the roadway. \( \theta = 0 \) equals to 0 along the roadway centerline.

Figure 3.x displays the generated curve path segment, which consists of a sequence of connected straight sub-paths. Since the curved path segment is divided into a sequence of straight sub-paths, navigating autonomous vehicles along the curve path segment is not significantly different from navigating autonomous vehicles along the straight path segment. Instead of having \( t = 0.0 \) at \( P_0 \) and \( t = 1.0 \) at \( P_1 \), we have \( t = 0.0 \) at \( \theta \) and \( t = 1.0 \) at \( \theta + \delta \) as illustrated in figure 3.x.

Endpoints of all autonomous vehicles in the virtual environment are stored into a 2-D dynamic array “endpoint[][]”. The first dimension of the array is the autonomous vehicle ID, the second dimension of the array is the sequence of endpoints of that autonomous vehicle. For example, the first endpoint of autonomous vehicle “V08” is stored as endpoint[8][0]; The sequence of endpoints of an autonomous vehicles is pushed into the array in order along the travel direction of the autonomous vehicle.

Notice that the speed of movement is actually determined by the speed of interpolation between two endpoints. The \( t \) step size during each iteration (each frame) between endpoint[m][n] and endpoint[m][n+1] actually defines the new position of the autonomous vehicle when the next frame is rendered and therefore defines how fast the autonomous vehicle appears to move. The rendering frame rate is not a constant across platforms. To make sure the autonomous vehicle travel at the speed specified in the database independently from the rendering frame rate, \( t \) is computed as in equation 4.1 during each iteration.

\[
\theta_{\text{new}} = \theta_{\text{old}} \times \frac{\text{desiredSpeed}}{\text{renderingFrameRate}}
\]

Equation 4.2
Where: \( V \) is the desired velocity specified in the database, \( T \) is the time took in seconds to finish the last frame and \( D \) is the distance between endpoint[m][n] and endpoint[m][n+1].

4.2.2 Lane change Points & Paths

A case that’s not covered in 4.2.1 is when two endpoints of “V05” are not in the same lane. Let’s change the second endpoint of “V05” in the database slightly to \{roadway:Beacon_00, lane:1, partition:1.0\} “V05” is then guided to start from the endpoint \{roadway:Beacon_00, lane:0, partition:0.3\}, travel at 20mph till it reaches the endpoint \{roadway:Beacon_00, lane:1, partition:1.0\}. Instead of going straight from endpoint 1 to endpoint 2, “V05” has to conduct a lane change maneuver to resemble real-life driving behaviors since the vehicle is switching lanes.

Figure 4.26 shows the lateral position profiles for lane changes in both directions for both (a) human data and (b) model simulations (Hetrick, 1997).

![Lateral position profiles for lane changes in both directions](image)

Figure 4.26 Lateral position profiles for lane changes in both directions (from Hetrick, 1997)

In this study, the lane change path between endpoint 1 and endpoint 2 is approximated by four connected sub-paths, as shown in Figure 4.27. Three lane change points are computed to guide the vehicle to make a lane change. Linear distance (L) can be calculated with equation

\[
L = \sqrt{\frac{4V^2D_m}{\mu g}} - D_m^2
\]

Equation 4.3
Where $V$ is the speed of the vehicle, $D_m$ is the lateral distance, $\mu$ is the coefficient of friction, and $g$ is the gravitational acceleration.

The first sub-path is defined by endpoint 1 and lane change point 1. The vehicle does not switch lane yet. It remains in its current lane and travels on the straight path segment till it reaches lane change point 1. The first sub-path exists only when the linear distance between two endpoints is longer than the linear distance ($L$) of lane change.

The actual lane change path is formulated by the second and third sub-paths. The second sub-path is a curve from lane change point 1 to lane change point 2, while the third sub-path is a curve from lane change point 2 to endpoint 2.

A closer observation of sub-path 2 and subpath-3 is given in Figure 4.28.

Figure 4.27 Lane change path between endpoint 1 and endpoint 2

Figure 4.28 Two curves that approximate the lane change swerve
To guide the vehicle to switch from lane 0 to lane 1, the program interpolates along sub-path 2 by smoothly rotate from vector 1 to vector 2. Similarly, once the vehicle hits lane change point 2, it moves on to sub-path 3, and the program interpolates sub-path 3 by smoothly rotate from vector 3 to vector 4.

4.2.3 Intersection Paths

Intersection is a junction where multiple roads meet. There are a number of possible routes that a vehicle can follow to cross the intersection. The choice of the route is dependent on the vehicle’s incoming road/lane and outgoing road/lane.

Take “V05” as an example. The second path it travels on is defined by endpoint 2 and endpoint 3. “V05” is guided to move from endpoint \{roadway:Beacon_00, lane:0, partition:1.3\} to endpoint \{roadway:massave_01, lane:0, partition:0.0\}. After converting \{GN,LN,P\} to \{x,y,z\}, two endpoints are positioned in the global coordinate system as shown in Figure 4.29. It’s quite obvious that “V05” needs to make a right turn to cross the intersection. However, it’s not obvious to the program. The vehicle could go straight from endpoint 2 to endpoint 3, or it could make a lane change, or make a left turn or a right turn. Studies done before often resolve this issue by superimposing an invisible route on the intersection to guide the vehicle. This study however eliminates the involvement of human supervision and resolves the problem by implementing a heading vector which gets associated to each endpoint.

It’s quite easy to the compute the angle from the start heading vector to the end heading vector. A positive angle indicates a right turn, while a negative angle indicates a right turn.
4.3 Autonomous Route Planning

As described in 4.2, designers can manually define the path for an autonomous vehicle to follow by adding records for that vehicle in the database. This is an important feature in terms of designing scenarios. A major purpose of having scenarios in which autonomous vehicles behave as desired in a repeatable mode is to monitor and study the driver’s performance or test driver assistance systems [add references].

On the other hand, when building a virtual driving environment, in order to provide a sense of realism, designers may also want to have some random traffic on the roadways. Those autonomous vehicles don’t have to behave strictly as defined in a repeatable mode. They are allowed, or actually preferred, to travel on different paths in different program runs in order to make the virtual environment appear more realistic. In that case, there is no need to define a detailed route for those vehicles to follow in the database. To lower workloads of designers, a method that autonomously plans routes for these vehicles is proposed in this research.

Different from the path shown in Figure 4.20, designers only specify the roads that vehicles start and stop as shown in Figure 4.30. The program has to automatically find the sequence of roadways and intersections that the vehicle has to travel through between “startRoad” and “endRoad”.

Figure 4.29 endpoints of “V05”
As introduced in 3.1, the topological network is stored as a tree. Therefore, the Breadth-First Search algorithm is adopted to find the path between the start node and the end node.

Implementation of the algorithm in this research is described as below:

1) Create an empty queue and push the starting node in queue.
2) Repeat step 3, step 4 and step 5 while the queue is not empty.
3) Push all the neighbors of node n in queue.
4) If any of the neighbors is the ending node, exit.
5) Pop the first node n of queue.
6) Output the search path as the planned path for the autonomous vehicle.

The breadth-first search algorithm actually returns the shortest-path between the starting node and ending node. Here, by shortest, it really means least number of roadways and intersections instead of shortest travel distance in the virtual environment.

With the sequence of roadways and intersections planned, navigating autonomous vehicles is no different as described in 4.2.

4.4 Colliders & Triggers

In Unity, colliders are important components of objects which allow objects to collide with each other. When a collision between two colliders occurs, three collision messages are sent out to the objects attached to them. These three collision messages are: OnCollisionEnter(), OnCollisionExit(), OnCollisionStay(). Events can be embedded in collision messages to be triggered when a collider enters/exits/touches another.
Sometimes, it’s desired to trigger events without really having collisions between objects. Unity provides an alternative way of using colliders, which is to mark them as triggers. Once marked as triggers, colliders will be ignored by the Unity physics engine and will not possess any physics properties. However, they can still trigger events because three similar trigger messages are sent out when a collider collides with a trigger. These triggers messages are OnTriggerEnter(), OnTriggerExit() and OnTriggerStay().

In this study, colliders are widely used to trigger events. For the sake of clarity, colliders that are marked as triggers will be referred to as “Triggers” in this paper, while collider that are not marked as triggers will be referred to as “Colliders”.

4.4.1 Colliders
Every object presented in the virtual environment has a collider attached. For example, a box collider is attached to a lane, and a terrain collider is attached to the terrain. Lane colliders, intersection colliders, and vehicle colliders will be described in this section since they are indispensable to vehicle perception and communication.

As a vehicle traverses the virtual environment, its location changes continuously. It travels in different lanes of different roadways, crosses intersections, etc. As described in 4.1, each topological node has a few driving related objects such as traffic signs, traffic lights, etc. attached. The vehicle has to be aware of its current topological nodes in order to know what objects to perceive.

A tedious way to provide real-time information about the vehicle’s current location in the topological structure is to have a supervision system which monitors all vehicles’ positions continuously throughout the simulation run. This would become quite resource-consuming as the scale of the road network grows and the number of vehicle increases.

In this project, colliders are used to update the location of vehicles in the road network.

As stated earlier in this section, each lane, intersection, and vehicles has a box collider attached. When a collision occurs between two colliders, collision messages are sent out to the objects. And any events can be embedded in these collision messages to be triggered.
When a vehicle enters a lane, the vehicle’s collider also enters the lane’s collider. This collision will send out a OnCollisionEnter() message. The program therefore can embed a few steps into the message body to perform an update to the vehicle’s current lane, current road. More details on these operations will be discussed in 5.1.

Similarly when a vehicle exits a lane, the vehicle’s collider also exits the lane’s collider. A OnCollisionExit() message is sent out as a results. Again a few operations can be performed inside the message body to update the vehicle’s location.

Both OnCollisionEnter() and OnCollisionExit() are sent out only once when the collision occurs. The program needs not to keep track of the vehicle’s location continuously. Hence, it’s tremendously more efficient and faster.

4.4.2 Triggers

Triggers are merely colliders which don’t possess physics properties. They’re adopted in this study to trigger events without affecting movements of objects. A very important use of triggers in this project is the endpoint triggers.

As described in 4.2, the global path of an autonomous vehicle is normally divided into a number of connected path segments $Sp[n]$. Each path segment has two endpoints.

When the autonomous vehicle reaches the end of path segment $Sp[n]$, it has to immediately move onto the next path segment $Sp[n+1]$ in the sequence in order to continue traversing its global path. This is done by move onto the next pair of endpoints and start interpolating again, as illustrated in Figure 4.31.

![Figure 4.31 Move onto the next pair of endpoints](image-url)
The program has to increase the path segment number from \( n \) to \( n+1 \) once the autonomous vehicle “V08” reaches endpoint[8][n+1], in order to move “V08” onto the next path segment without a delay or a jump, because either of them would cause the autonomous vehicle to move unnaturally.

The question is how does the program know that “V08” has reached endpoint[8][n+1]. The program can always calculate the distance between the autonomous vehicles and the endpoint[8][n] in real-time. But this is costly. As the number of vehicles increases, considerable more memory resources are taken. Instead, an Endpoint Trigger is planted at the position of each endpoint. When the autonomous vehicle hits the trigger, the collider of the autonomous vehicle enters the Endpoint trigger, a OnTriggerEnter() message is sent out. To increase the path segment of that vehicle from \( n \) to \( n+1 \), the program just has to embed a few lines of code in the message body. No operation is needed till the moment the vehicle reaches the endpoint. This is much more efficient than keep tracking of the distance between the vehicle and the endpoint.

![Figure 4.32 vehicle hits an endpoint](image)

Similar to endpoint triggers, lane change triggers are introduced to initialize the lane change path. A lane change trigger is placed at the position of each lane change point. When an autonomous vehicle hits a lane change trigger, the program initiates the lane change maneuver, which is described in details in Chapter 5.
Most lane change triggers are planted when the program initializes. However, new lane change triggers are planted in real-time when the program runs with respect to the instant traffic condition.

For example, when the vehicle perceives a road construction ahead in its current lane, it will make a decision to switch to another safe lane. New lane change points are therefore planted in real-time to guide the vehicle to follow a lane change path. The process of decision making will be described in details in Chapter 5.

As a conclusion, colliders and triggers can be generally used to trigger any events. They’re not only used in updating vehicle’s location, increasing path segment, initializing lane change, but also used in designing repeatable scenarios. For example, triggers can be very helpful in designing a scenario in which a parked vehicle always pulls up in front of the driver’s car when the distance is less than 30 feet,
5 Perception, Communication and Decision Making

As described in Chapter 4, both geometric and topological information about the virtual environment is stored into the database and loaded to in-memory arrays as the program initializes. Routes of vehicles can be planned both manually and automatically. However, the vehicle is not yet an autonomous vehicle as defined in this research. The problem with how vehicles perceive the information loaded into in-memory arrays is not addressed yet. Furthermore, a vehicle doesn’t communicate with other vehicle, and hence is incapable of making decisions and taking actions accordingly. With a considerable number of vehicles traveling on different routes in the virtual environment, it’s impossible to form an organized traffic without vehicles behaving autonomously.

In order to have simulated vehicles behave as if humans controlled them, past researchers (Carles & Espie, 1999) (Willemsen, Kearney, & Wang, 2006) have used databases to store properties about the environment. Examples include 1) providing road speed limit information to autonomous vehicles, and 2) letting autonomous people know the state of traffic signals. Supplying such information gives simulated vehicles a low level of intelligence i.e., an autonomous vehicle knows how fast it should be going. This concept has been expanded to include storing topological and semantic information of roadways in databases (Yan & Weijian, 2003). This has enabled autonomous vehicles to follow global routes while conforming to traffic rules (e.g. stop at stop signs and red traffic signals).

Furthermore, interactions between autonomous vehicles and roadways are not sufficient in terms of having a complete understanding of the driving environment. On real roadways, even if every single traffic rule is followed, there is still no guarantee of safe travel and efficiency. Drivers in the real world pay attentions to other drivers on the roadway and react in a timely manner to avoid unsafe situations. Drivers act/react by observing others’ behaviors in the real world. Thus, inter-vehicle communications are required for simulated vehicles in virtual environments.
Some virtual environments include live human participants as well as autonomous objects (Edward, Loudeaux, Barthes, Lenne, & Burkhardt, 2008). Thus there is also a need for communications between humans and autonomous objects. Humans can be modeled as a special type of very intelligent autonomous objects (Liu, Zhong, & Zhan, 2008). This would enable them to be included in a communications model for all objects in virtual environments.

To model human-like behaviors of simulated vehicles on highway systems, (Al-Shihabi & Mourant, 2003) proposed an abstract framework that consists of four units: the perception unit, the emotions unit, the decision-making unit, and the decision-implementation unit. Furthermore, four driving models: 1) a generic driving model, 2) an aggressive driving model, 3) an alcoholic driving model, and 4) an elderly driving model were implemented to assemble a variety of characteristics of drivers. Figure 5.1 shows the architecture of the proposed framework.

![Figure 5.1 Architecture of the driver behavior framework (from (Al-Shihabi & Mourant, 2003))](image)

In an earlier research (Mourant, Yin, Lin, & Kamarthi, 2010), a combination of a perception model, a communication model and a decision-making model was proposed to enable autonomy in simulated
vehicles. All models are attached to autonomous vehicles in the virtual environment. This enables such an autonomous vehicle to become an independent entity that is self-motivated and self-controlled. An autonomous vehicle perceives its environment with the perception model, and communicates with other autonomous vehicles via the communication model. The decision-making model relies on the perception model and the communication model to investigate possible outcomes before making decisions. The perception model perceives current roadways, current lane, traffic signs, traffic signals, obstacles, etc. to give the vehicle a comprehensive understanding of its surrounding driving environment, including traffic rules applied at the moment. The communication model defines senders, receivers, contents, and channels (media through which content is transferred) in real-time to gather desired information from other vehicles. The decision-making model is divided into two levels, the global level and the local level. These work, respectively, with environmental information perceived by the perception model and the traffic condition received by the communications model. A group of logic rules are formulated as decision trees to model the process of decision making in real-time.

This research is a great extension of the earlier research on following aspects: 1) improved perception speed and reduced computational costs; 2) better interaction among the perception, the communication, and the decision making model; 3) increased complexity in the decision-making model with more complicated logic rules formulated; 4) higher level of autonomy in complicated driving scenarios.

5.1 The Perception Model

5.1.1 In-Memory Arrays

As introduced in Chapter 4, all environmental information is stored into a relational database. However, it’s not possible for vehicles to retrieve information from the database in real-time. Going back and forward between the frontend program and backend database would consume a considerable amount of time. Especially when number of queries sent to the database increases, a great amount of load would be generated on both the frontend and the backend. The simulation typically needs to render tens of frames per second in order to eliminate any human perceivable discontinuity. Having the program getting
information back from the database during the simulation run is not acceptable. For the purpose of vehicles perceiving the driving environment in real-time efficiently, in-memory arrays are utilized in this project.

A JavaScript program is ran to load all database records into corresponding arrays when the program initializes. It’s understandable that these arrays are normally not full. All arrays that load information about static objects are static arrays. They’re populated when the program starts and remained unchanged throughout the simulation run.

However, static arrays are not suitable to store data of dynamic objects whose properties change during the simulation run. A most commonly seen example of dynamic objects is the autonomous vehicle. To store data of autonomous vehicles, dynamic arrays are implemented.

In this research, each roadway and intersection is transformed into a 2-D dynamic array. For example, a 2D array $R_i[..][..]$ represents the $i_{th}$ roadway. Each lane of the roadway is a 1-D array. The $i_{th}$ lane of the $R_{i}[..][..]$ roadways is represented as $R_{i}[i][..]$, and the $i_{th}$ autonomous vehicle on that lane is denoted as $R_{i}[i][j]$. Elements of $R_{i}[..][..]$ are IDs of autonomous vehicles that are currently on the $i_{th}$ roadway: $R_{i}[i][j] = "$\text{vehicle ID}"$. Therefore, to refer to an autonomous vehicle, the program can use either its vehicle ID or its index in the roadway array: $R_{i}[i][j]$.

![Figure 5.2](image.png)

**Figure 5.2** Refer to vehicles on roadways as elements of 2D arrays

Lanes are sorted from the most right to the most left. Elements (autonomous vehicles) in $R_{i}[i][..]$ are sorted by their relative distances to the end of the $i_{th}$ lane. Figure 5.2 shows how vehicles are referred to
as elements of 2-D arrays, based on its current lane and its distance to the end of the lane. Operations enabled on the one-dimensional array \( a[i] \) are: 1) insert a new element into the array, 2) remove an element from the array and 3) sort elements in the array. Therefore, when the \( i \)-th autonomous vehicle in the \( i \)-th lane of roadway \( i \), represented as \( a[i][i] \), shifts to the adjacent \((i-1)\) lane, it will be removed from the array \( a[i][..] \) and inserted into the array \( a[i-1][..] \). An autonomous vehicle is considered as an element in a lane even if only part of its 3D visual model is in that lane, e.g. an autonomous vehicle that is currently shifting from the adjacent lane.

With the 2-D roadway dynamic array implemented, it’s quite easy to locate an autonomous vehicle in the road network. For example, if \( x_2[0][3] = "\square 08" \), then vehicle “\( \square 08 " \) is currently in the most right lane on the 2\(^{nd} \) roadway and it has 3 vehicles driving ahead of it in the same lane. As an autonomous vehicle travels in the road network, its location changes continuously. The program needs to update 2-D roadway arrays in real-time whenever a change occurs. As shown in Figure 5.3.a, \( \square 06 \) can be referred to as \( a_0[0][0] \) since it’s in the most right lane and there is no vehicle ahead of it. In another word, it’s the first element in the 1-D array \( a_0[0] \). Figure 5.3.b shows that \( \square 06 \) starts to make a lane change from the most right (lane 0) to the adjacent lane (lane 1). Since its 3D visual model is not yet in lane 1, \( \square 06 \) is still referred as \( a_0[0][0] \). When \( \square 06 \) enters lane 1 as shown in Figure 5.3.c, it can no longer be referred to as \( a_0[0][0] \) since it’s not in the most right lane anymore. Instead, it should be referred to as \( a_0[1][0] \). In order to determine “\( \square \)”, the program has to sort elements in \( a_0[1] \) again since \( \square 06 \) has been added to it. After resorting \( a_0[1] \), \( \square 06 \) is referred to as \( a_0[1][1] \) and consequently \( \square 05 \) is referred to as \( a_0[1][2] \).

As a conclusion, a single lane change behavior requires 3 array operations: 1) remove the element from its current array; 2) insert the element into the new array and 3) re-sort the new array.
It’s not acceptable to have any delays in updating dynamic arrays. The program has to update dynamic arrays timely at the very moment when changes happen. “Colliders” are used to achieve this goal.

As described in 4.4.1, when a collision between two colliders occurs, three collision messages are sent out to the objects attached to them. These three collision messages are: OnCollisionEnter(), OnCollisionExit(), OnCollisionStay(). Events can be embedded in collision messages to be triggered when a collider enters/exits/touches another collider. In this project, each vehicle has a box collider and each lane has a box collider attached to it as well. Figure 5.4 shows the moment when the collider of □06 begins touching the collider of lane 1.

At this moment, □06 begins touching the collider of lane 1 and leaving the collider of lane 0. Therefore, two collision messages are invoked: OnCollisionEnter and OnCollisionExit. Two operations on the array □₀[₁] are embedded in the OnCollisionEnter message, and the pseudocode is given as below:

![Image of three frames showing operations on arrays in a lane-change scenario]

Figure 5.3 Operations on Arrays in a lane-change scenario
function OnCollisionEnter()
{
    Insert an element to the array $R_0[1]$
    Sort the array $R_0[1]$
}

Another two operations are embedded in the OnCollisionExit message, the pseudocode is given as below:
function OnCollisionExit()
{
    Remove the element from $R_0[0]$
    Sort the array $R_0[0]$
}

Figure 5.4 The collider of □06 enters the collider of lane 1

5.1.2 Perceiving the Driving Environment

In this research, “perception” is defined as the process of autonomous vehicles retrieving information from in-memory arrays on an as-needed basis. During the simulation run, the autonomous vehicle is provided information such as: 1) its location in world coordinates, 2) its location in the road network, 3) its heading information, 4) traffic rules applied at the moment (such as speed limits, lane markings, traffic
lights), and 5) presence of any obstacles. By means of perception (acquisition of information from in-memory arrays) autonomous vehicles have comprehensive state knowledge of their near-by surroundings.

Each autonomous vehicle is a GameObject in Unity 3D. A transform that determines its Position, Rotation, and Scale is attached to the GameObject. Therefore, to perceive its location and heading in world coordinates, the autonomous vehicle just has to refer to its attached transform and gets the Position and Rotation properties. Figure 5.5 shows a simulated vehicle in Unity and its attached transform.

![Figure 5.5 A simulated vehicle and its attached transform](image)

As introduced in 5.1.1, roadway arrays are dynamically updated during the simulation run. $\text{Roadway}[\text{id}][\text{lane}] = $ "roadway", therefore to get knowledge about its current roadway and current lane, the program just has to return $\text{id}$ and $\text{lane}$.

To perceive DR environmental objects, however, an autonomous vehicle has to get information from other static arrays that were populated when the simulation loaded. Recall that each DR environmental object is associated to a topological node, which is the topological representation of a road network component.

For example, autonomous vehicle $\text{Vehicle}[\text{id}][\text{lane}]$, which currently travels in the $\text{lane}$ on the $\text{roadway}$, it needs to perceive DR environmental objects that are associated with its current lane only.

When program initializes, all DR environmental objects are loaded into static Arrays. For example, traffic lights are loaded into the array: $\text{TrafficLight}[$[$\text{id}$]$][[$\text{lane}$]$][[$\text{sequence}$]%]. traffic signs are loaded into the array: $\text{TrafficSign}[$[$\text{id}$]$][[$\text{lane}$]$][[$\text{sequence}$]%], and obstacles are loaded into the array: $\text{Obstacles}[$[$\text{id}$]$][[$\text{lane}$]$][[$\text{sequence}$]%]. Each Array is 3 dimensional. The first dimension is the id of the associated roadway, the second dimension is the id of the associated lane, while the third dimension specifies the sequence of that DR object. There can be a number of traffic signs, for example, placed down a roadway. In the database, traffic signs are sorted
down the traffic direction. And therefore, when loaded into arrays, signs are sorted down the traffic direction as well.

As shown in Figure 5.6, autonomous vehicle $\Box_{15}[I][0]$ needs to perceive DR objects that are associated with its current lane only. It retrieves elements from all static arrays with "15" as the index in the first dimension, "1" as the index in the second dimension. As illustrated below:

$$\Box_{15}[1][0]$$

In this case, the program found that $\Box_{15}[1][0]$ was not empty and $\Box_{15}[1][0]$ represents a construction sign. As a result, such information is perceived by the autonomous vehicle $\Box_{15}[I][0]$ and it becomes aware of the existence of an obstacle ahead along its traveling direction.

Retrieving records from in-memory arrays is fast and little computer resource is consumed.

So far, autonomous vehicles are capable of getting all information of static objects in the driving environment. According to the definition of autonomous in this study, autonomous vehicles have to not only perceive the driving environment but also communicate with other moving autonomous vehicles.
For this purpose, a communications model is introduced.

5.2 The Communications Model

The communications model simulates communicating behaviors between autonomous vehicles (Berder, Quemerais, Sentieys, & al., 2008). This model includes every autonomous vehicle in the environment. Each autonomous vehicle is an independent entity that possesses the ability to communicate. Usually, communication is described as a process of transferring information from one entity to another. Four indispensible elements of communication are defined as: sender, receiver, channel, information to transfer.

The communication model implemented in this research is a Multiple-Sender-Multiple-Receiver (MSMR) model. Each autonomous vehicle is considered as a receiver by default. An autonomous vehicle becomes a sender upon requests from receivers. The information pool, which is the combination of in-memory arrays, serves as the channel to convey information from the senders to the receivers. Information to transfer is elements of in-memory arrays.

Information is transferred from senders to receivers upon requests. As an autonomous vehicle travels through the virtual environment, it sends different requests to the information pool with respect to its instantaneous circumstances. For example, an autonomous vehicle that’s traveling on a roadway would send a request to get the position of its preceding vehicle, in order to keep a safe following distance. When the vehicle intends to start a lane change, it sends two more requests to get the positions of the preceding vehicle and following vehicle in the target lane, in order to find a safe time gap to finish the lane change maneuver.
Take the intersection scenario presented in Figure 5.7 as an example. The process of communicating with other vehicles in the scene consists of following steps:

a) Initialization. By default □01 is a receiver.

b) Selection of potential senders. Based on □01’s current circumstance, a group of autonomous vehicles that may be affecting □01 is selected. The procedure of selecting the group of vehicles to communicate with in various scenarios is covered later in 5.2.1 where different driving behaviors are studied. The rational for this is that a driver in the real world would only want to pay attention to other vehicles whose behaviors may potentially affect his/her travel safety and efficiency. Similarly, an autonomous object should only communicate with vehicles that are worth communicating with. Communicating with all other autonomous objects in the virtual environment is neither necessary nor human-like. In the example presented in Figure 5.7, it’s only necessary for □01 to communicate with □02.
c) Determination of the information to be received from the sender. The receiver determines the information about the senders (vehicles) it needs to get, in order to have a real-time and comprehensive understanding of the current traffic condition, and hence make decisions on its next-step action.

d) A request which contains the “potential senders” and “information to transfer” is composed, and sent from the receiver to the information pool.

e) Process of the request. It’s important to understand that $\square \text{01}$ doesn’t refer to its senders by vehicle IDs. It would only refer to its senders as “the preceding vehicle in my lane” or “the following vehicle in the target lane”, etc. Therefore, the request received by the information tool needs to be further processed, in order to determine the vehicle IDs of senders. As described in 5.1.2, a vehicle can be allocated by referring to the dynamic 2D array $\square_1[\square_{\text{VehiclePosition}}][\square_{\text{VehicleLane}}]$. Thus, the information pool first allocates $\square \text{01}$ and finds out that it’s currently traveling in the 3rd lane from the most right on the $17_{\text{th}}$ roadway, and it’s the $0_{\text{th}}$ vehicle to the roadway end. Therefore, $\square \text{01}$, which should be referred to as $\square_1[7][0]$, refers to its sender as “vehicles whose intersection paths intersect with mine”. The information then tries to allocate those senders by referring to the dynamic 2D array $\square_0[\square_{\text{VehiclePosition}}][\square_{\text{VehicleLane}}]$.

f) When the sender receives the request from the information pool, the requested information is sent to the information pool by updating the corresponding elements of in-memory arrays. For example, the information pool receives the current position of $\square \text{02}$ by updating the 2nd element of the array $\text{VehiclePosition[]}$. Its mathematical representation is: $\square_0[\square_{\text{VehiclePosition}}][2] = \square \text{02. VehiclePosition}[]$.

g) At last, the information tool sends the information received from the sender to the receiver.

5.2.1 Selection of Vehicles to Communicate with

Before an autonomous vehicle initializes the communication, it has to know whom to talk to. There is no need to communicate with all other vehicles in the environment, thus it’s necessary to define the group of
its possible senders (PSG). Since an autonomous vehicle considers itself a receiver by default, it has to
define some other autonomous vehicles as senders at initialization time. A driver on real roads
experiences communications with different vehicles as he/she travels along the route. Our
communications model mirrors this process by adding and removing senders from an autonomous
vehicle’s PSG array in real time.
To determine senders, an autonomous vehicle perceives the driving environment around it. Three driving
scenarios are considered in this study: 1) following, 2) lane changing, and 3) intersection. The model
filters out non-relevant autonomous vehicles based on its perception of the driving environment.
It’s worth mentioning that the process of defining senders is executed in real-time so that the PSG array of
an autonomous vehicle gets updated in real-time.
We use vehicle □01 as an example.

1) Following.

When on a roadway and is still far from the upcoming intersection, □01 would mostly be affected
by the autonomous vehicle ahead if it intends to remain in the same lane. □01 has to keep a safe
distance from the preceding vehicle as well to maintain the desired speed. This is described as the
following behavior, one of the most universal situations in daily driving. In this case, the vehicle
selects the preceding vehicle in its lane to communicate with and sends a request to the
information pool.

As shown in Figure 5.8, □01 is found to be the 0_h autonomous vehicle in the 1_h lane of
roadway 10, represented as □10[l][0], it would only have to access the autonomous vehicle ahead,
which doesn’t exist since □01 is the first element in the array. However, for □07, which could
also be referred to as □10[l][1], it has to check the status of the vehicle ahead, denoted as
□10[l][0]. The program associate □10[l][0] with □01, therefore □01 is one of the PSG for □07.
A communication is then established between these two vehicles.
2) Lane Changing

A vehicle may be unsatisfied with its speed in the current lane or need to get prepared for the upcoming left/right turn. As a result, it intends to make a lane change instead of following the preceding vehicle in the current lane. The decision of starting a lane change is made by the decision-making model, which is described later in this paper. However, the vehicle doesn’t initialize a lane change right after it makes the decision. It has to communicate with other vehicles in order to find the appropriate time gap. In the case of a lane-changing behavior, the autonomous vehicle ahead in the same lane is still worth communicating with. More importantly, the vehicle has to communicate with autonomous vehicles in the target lane. The following vehicle and the leading vehicle in the target lane are two vehicles that determine whether it’s safe to start a lane change. Therefore, in this case, the vehicle defines the preceding vehicle in the current lane, the preceding vehicle in the target lane and the following vehicle in the target lane as its senders.

Figure 5.9 illustrates the communication between □08 and its senders in a lane change scenario where □08 tries to make a right lane change.

□08 is still the $l_{th}$ autonomous vehicle in the $l_{th}$ lane of roadway 18, represented as $l_{18}[I][I]$ and it intends to shift to the $0_{th}$ lane. The program has to determine which vehicles in the $0_{th}$
lane are the preceding vehicle and the following vehicle. It scans through the array \( I_8[0][\cdot] \), and computes the vectors with three components \((x, y, z)\) as:

\[
\vec{\mathbf{v}}[\cdot] = I_8[0][\cdot].
\]

If the angle between \( \vec{\mathbf{v}}[\cdot] \) and the forward direction of \( 08 \) is smaller than \( \pi/2 \), then vehicle \( I_8[0][\cdot] \) is traveling ahead of \( 08 \) in the target lane. Otherwise, vehicle \( I_8[0][\cdot] \) is traveling behind \( 01 \) in the target lane.

At last, the program finds the vehicle that’s the closest one to \( 08 \) among vehicles traveling ahead of \( 08 \), and defines it as the preceding vehicle in the target lane. Similarly, the closet vehicle among vehicles traveling behind \( 08 \) is defined as the following vehicle in the target lane.

In Figure 5.9, \( 12 \) is the preceding vehicle in the target lane, and \( 13 \) is the following vehicle in the target lane, \( 05 \) is the leading vehicle in the current lane of \( 08 \). As described earlier, in order to make a safe lane change, \( 08 \) selects \( 05, 12, 13 \) as its senders, and request information about these 3 vehicles from the information pool.

![Figure 5.9 Communicatio in a lane change scenario](image)

3) At an Intersection

Another frequently encountered circumstance in driving simulations is when autonomous vehicles come to an intersection. Autonomous vehicles’ intended paths often intersect at
intersections. Other than checking traffic signs and signals, autonomous vehicles must as well communicate with each other to avoid collisions and to continue on their planned routes. For example, at a stop sign before a “T” intersection, a driver should stop until the way ahead is clear. To realize the way ahead is clear, the driver has to check for other vehicles and/or pedestrians.

We require all autonomous vehicles coming to an intersection to strictly follow traffic signals and signs. Then, it’s only necessary to consider vehicles whose intended courses of travel interfere.

Figure 5.10 Communication in an intersection scenario

Figure 5.10 shows an intersection scenario that involves 4 vehicles.

Autonomous vehicle □/15 intends to turn left while autonomous vehicle □/17 intends to go straight. Once the traffic light for them turns green, there is an available time period for them to cross the intersection. The problem is that autonomous vehicle □/15 and □/17 share the same time window. Their intended routes physically intersect at point B. If communication is missing between them, they may come into a conflict and cause a collision. Also, autonomous vehicle □/16 on the right roadway plans to drive straight. Its intended route physically intersects with
\( I7 \)'s planned route at point \( A \) and \( I4 \)'s planned route at point \( C \). However, the traffic light for \( I6 \) is still on red, its time window to cross the intersection does not overlap with that of \( I7 \) or \( I4 \). Therefore, even though \( I7 \) and \( I4 \) doesn't communicate with \( I6 \), there would be no collision expected. As a conclusion, autonomous vehicle \( I7 \) is a sender of autonomous vehicle \( I5 \), and it's also true the other way around.

It takes a bit more works for the program to find the group of vehicles that a vehicle wishes to communicate with at an intersection.

As described in 4.1.2, each roadway is modeled as a 2D array, each lane of the roadway is modeled as a 1-D array, and each autonomous vehicle on the roadway can be referred as an array element. In the case of an intersection, although it is not divided into visible lanes, there are a number of possible routes on it. Each route can be considered as a “lane”. Hence, in order to get an autonomous vehicle’s path when it’s on an intersection, each intersection is transformed to a 2D array as well.

Four 4-lane roadways connect to the intersection in the model shown in Figure 5.11. All possible routes for autonomous vehicles to follow are marked as arrowed lines. When coming to the end of the entry lane, a path will be formulated based on its desired exit roadway and exit lane. Given the entry point and exit point, an invisible path is computed to connect these two points. The invisible path then steers the autonomous object to cross the intersection. As shown in Figure 5.11, each lane has two possible paths connected to the end of the lane axis. Notice that, the invisible intersection in the intersection model is a bit bigger than the actual 4-way junction. This is because drivers always get prepared to cross the junction a bit before they arrive at the entry point of the junction. In order to have autonomous vehicles plan routes as live beings, a bigger invisible collider is attached to the geometric junction surface.
It’s easy to reconfigure the general model built in Figure 5.11 to be adapted to a 3-way junction. Designers just have to erase all routes directed to or originated from the roadway that is not there anymore.

Similar to the transformation of a roadway into a 2-D array, an intersection is transformed into a 2-D array, represented as \( \Box_{\square}[...][]\) as well. Each invisible path that guides vehicles to cross the intersection is transformed into a 1-D array, represented as \( \Box_{\square}[] \). Each autonomous vehicle on that route is then an array element, represented as \( \Box_{\square}[\square][] \). Elements (autonomous vehicles) in \( \Box_{\square}[] \) are sorted by their relative distances to the exit point of the \( \Box_{\square} \) invisible path of intersection \( i \).

When an autonomous vehicle, e.g. \( \Box I5 \), comes to the entry point of an intersection, it sends requests to the information pool to communicate with vehicles on “intersected path”. The information pool then first finds out the invisible path that \( \Box I5 \) is on, e.g. the \( \Box_{\square} \) path. According the intersection model, it’s easy to determine the paths that intersect with the \( \Box_{\square} \) path in the same time period, say the \( \Box_{\square} \) path and the \( \Box_{\square} \) path. If a vehicle, e.g. \( \Box I7 \), is within certain distance to the intersection and its intended path on the intersection is the \( \Box_{\square} \) path or the
If 17’s intended path is the path or the path, but it’s still “far” away from the intersection, it’s not considered as a sender of 15. By “far”, it means that there is enough time left for 15 to cross the intersection before 17 comes to the intersected point of their paths. The information finds all senders and send request to them.

5.3 The Decision-Making Model

The decision-making model is responsible for making the best decision based on perceptions of the virtual environment and information received from other vehicles via the communication model. Decisions made by an autonomous vehicle’s decision-making model are carried out by a dynamics model associated with the vehicle.

The decision-making model does not only make decisions but also decides when and where to carry out these decisions. With information received from other vehicles, the decision-model is capable of evaluating the instant driving condition to see whether it’s safe to implement these decisions.

To achieve the goal of making proper decisions at the proper time and location, the decision-making model is divided into two levels: 1) the global level and 2) the local level. At the global level, the model reaches a decision D. At the local level, the model has to reach a decision whether to carry out the decision D. The global level is named global, because the model makes a decision based on perceptions of the global environment via the perception model. At the local level, the model reaches a decision on the basis of local information received from the communication model.

A “tree” data structure was used to model the process of decision-making [1]. Various trees are implemented with respect to various driving conditions.

Figure 5.12 shows a sample tree that indicates how the decision-making model works at the global level and the local level. By stepping through the nodes of the tree, a leaf node is reached. A leaf node, which
has no child node, represents a decision. Therefore, by reaching a leaf node, a decision is made. Once a
decision is made at the global level, the decision-making model proceeds to the local level and relies on
the communication model to gather the needed information. An autonomous vehicle that intends to make
a decision communicates with its senders defined by the communication model. With information sent
from senders, the decision model steps through the tree at the local level, and finally reaches a leaf node
that either permits the decision or provides an alternative decision.

Figure 5.12 presents the logic rules that are constructed into the decision-making tree which handles lane
changing scenarios.

5.3.1 Lane Changing Behavior

Lane changing is a common seen driving behavior. In driving simulations, lane changes made by
autonomous vehicles help to produce more realistic and immersive scenarios. Below are logic rules
constructed into the decision making model at the global level:

a) If an autonomous vehicle is traveling on a roadway with more than one lane in the same
   traffic direction and is far from the next intersection, and its speed is unsatisfactory, then the
decision is to make a lane change.

b) If the autonomous vehicle is traveling on a roadway with more than one lane in the same
   traffic direction and is close to the next intersection, and is in the right lane for the upcoming turn,
then the decision is not to make a lane change.

c) If the autonomous vehicle is traveling on a roadway with more than one lane in the same
   traffic direction and is close to the next intersection, and is not in the right lane for the upcoming
   turn, then the decision is to make a lane change.

d) If the autonomous vehicle is not traveling on a roadway with more than one lane in the
   same traffic direction, then the decision is to not make a lane change.
Figure 5.12 An example of the decision-making tree at the global level and the local level
The meaning of “far” and “close” to the next intersection varies on each roadway for each autonomous vehicle. A method was proposed to divide the roadway into three zones [9]. In the zone that is “close” to the intersection, the autonomous vehicle has to get into the right lane as soon as possible to follow its global route. In the zone that is “far” from the intersection, the autonomous vehicle has the freedom to change its lane to seek a better driving condition. This paper adopts this method and defines the term “close” based on the time to complete a lane change. For example, if an autonomous vehicle has to make \( n \) discrete lane changes to get into the correct lane, to get prepared for the upcoming turn, then the term “close” can be expressed mathematically as:

\[
\text{Equation 5.1}
\]

\( v \) is the speed of the autonomous vehicle, \( n \) is the number of single lane changes to make, \( T \) is the time needed to finish a single lane change and \( c \) is a constant factor. The constant factor \( c \) is always greater than 1. This would allow a vehicle to be in the proper lane for the turn before reaching the intersection.

With the four logic rules introduced above, the decision-making model has reached a decision on whether to make a lane change. It’s not yet finished because the model has to decide which lane is the target lane [9]. If the autonomous vehicle only tries to get into the correct lane to get prepared for the upcoming turn, it would be easy to decide the target lane because it’s defined by the global route. For example, if the autonomous vehicle wants to turn right at the intersection, it has no other choice but to get into the most right lane. However, if the autonomous vehicle is seeking to travel at a faster speed, it would be impossible to select the target lane rely solely on the perception model. Assistance from the communication model is needed. By communicating with other autonomous vehicles in adjacent lanes, the autonomous vehicle gets knowledge about the traveling speed in other lanes. The lane with the highest traveling speed is then chosen as the target lane. Logic rules to be implemented are:

\[\text{e) If a lane has the highest traveling speed among all three adjacent lanes in the same direction, select this lane as the target lane.}\]
f) If none of the lanes has a higher traveling speed, then abandon the decision to make a lane change.

Until now, the decision to make a lane change or not has been reached at the global level. If the decision is to not make a lane change, the decision-making model stops at this level. Otherwise, the model continues onto the local level. With the decision to make a lane change to the target lane (the lane) in mind, the autonomous vehicle starts to communicate via the communication model with its senders in the adjacent lane (the lane) towards the target lane. Notice that the target lane may not be one of the adjacent lanes. As described in the communication model, both position and speed of the leading vehicle and following vehicle in the lane are sent to the autonomous vehicle. The minimum acceptable distances to the leading vehicle and the following vehicle are defined as follows:

$$D_f = (v_{fp} - v_p) \ast T + D_s;$$

Equation 5.2

$D_f$ is the vehicle’s speed projected onto the traffic direction vector, $v_{fp}$ is the leading vehicle’s speed projected onto the traffic direction vector, $v_p$ is the following vehicle’s speed projected onto the traffic direction vector, $T$ is the time to finish a lane change and $D_s$ is the safe distance between vehicles. If both $D_f$ and $D_s$ are satisfied, then it’s safe to start changing the lane. The logic rule to be implemented at the local level is:

$$g) \text{ If both the distances } D_f \text{ and } D_s \text{ are satisfied, then starts to make a lane change, else, do not start}$$

The final decision is reached after applying this logic rule.

5.3.2 Following Behavior

To ensure the safety of travel, one should keep a safe distance to its preceding vehicle. It should decelerate if the vehicle ahead is within a certain distance $D_s$. To ensure the efficiency of travel, one
should also accelerate if the vehicle ahead is beyond a certain distance $h$. The logic rule implemented at the global level is:

h) If the autonomous vehicle is traveling on a roadway and its traveling speed is satisfactory, then the decision is to follow.

Once the decision to follow is made at the global level, the decision-making model goes deeper into the local level. The logic rule to implement at the local level is:

i) If the distance to the vehicle ahead is smaller than $h$, then decelerate, else if the distance is larger than $h$, then accelerate, else, maintain the current speed ”.

The communication model returns the speed and position of the autonomous vehicle ahead and hence the distance to it is easily computed. Apply the logic rule and a final decision will be achieved.

5.3.3 Intersection Behavior

At intersections, the decision-making model plays an important role in resolving conflicts among autonomous vehicles. The final decision made by the decision-making model determines whether it’s legal and safe for the autonomous vehicle to cross the intersection.

First, the autonomous vehicle perceives the driving environment to check the status of the traffic light.

j) If the autonomous vehicle is coming to an intersection and the traffic light is on red, then decelerate to stop and wait

k) If the autonomous vehicle is coming to an intersection and the traffic light is on green, then cross the intersection

l) If the autonomous vehicle is coming to an intersection and the traffic light is on yellow and the autonomous vehicle is close enough to the intersection entry point, then cross the intersection.
m) If the autonomous vehicle is coming to an intersection and the traffic light is on yellow and the autonomous vehicle is not close enough to the intersection entry point, then decelerate to stop and wait.

By saying “the autonomous vehicle is close enough to the intersection entry point”, it means there is enough time for the autonomous vehicle to reach the entry point before the traffic light turns from yellow to red. On basis of its speed and current distance to the entry point, the decision-making model is capable of determining whether it’s close enough. “Stop” is a driving behavior that slows the autonomous vehicle down to a complete stop. By reaching the leaf node of the decision-making tree at the global level, the model already drew a conclusion about whether it’s legal to cross the intersection. If it’s not legal, the decision-making model stops at the global level and lets the decision of “stop to wait” be carried out. If the decision is to cross the intersection, the decision-making model starts to step through the decision-making tree at the local level to see whether it’s safe to cross. As described in the communication model, an autonomous vehicle communicates with other vehicles whose intended paths interfere with its planned routes and are within certain distance to the intersection. Here, these vehicles are named as “conflicted vehicles” for convenience.

The concept of priority is introduced to determine which vehicle gets to cross the intersection first when there is a conflict. By default, vehicles that go straight across the intersection have the priority over vehicles that turn left/right. Vehicles with no priority should wait till there is no conflicted vehicle with higher priority crossing the intersection. Since the communication model has already determined the group of conflicted vehicles, it’s easy for the decision-making model to reach decisions based on logic rules presented below at the local level:

n) If the autonomous vehicle has the priority over conflicted vehicles, then cross the intersection

o) If the autonomous vehicle does not have the priority over conflicted objects and the group of conflicted vehicles is not empty, then stop to wait.
p) If the autonomous vehicle does not have the priority, but the group of conflicted vehicles is empty, then cross the intersection.
6 Conclusions and Future Work

6.1 Contributions and Conclusions

This thesis, motivated by the need of designing realistic virtual environment efficiently for driving simulations, began by examining state of the art of Virtual Reality. Advancement in computer hardware and software fields has helped tremendously in enhancing the reality of virtual environments as well as lowering the workload of designers. However, the ability of designers to construct and update the virtual environment efficiently still differs greatly depend on the algorithm proposed by designers. Also, this study chooses to focus on an emerging fact: Intelligent Virtual Agents (e.g. autonomous vehicles) which helps enhancing the feeling of user presence greatly. Enabling autonomy in virtual agents is considered as a major contribution this study can make.

After reviewing previous researches on geometrical modeling, topological modeling of virtual environments, this study concluded that the concept of multilevel modeling of virtual environments helped tremendously in standardizing the construction of virtual environments as well providing abundant information to virtual agents to behave autonomously. Aerial images are considered as good resources in reproducing geometrical structures of the road network concisely and efficiently in virtual environments. However, this study needs to propose a simplified and more reliable approach to reproduce real-world network with aerial images since none of the previous studies have achieved both efficiency and high-reality. Previous studies on topological representations of road network have provided foundations of this study. A review of previously proposed topological models has shown great advantages of having multiple levels of topological structures which however, none of the previous studies has actually implemented universally when constructing virtual environments. Last, the review of different driving behavior models provides deep understandings of three mostly common seen daily driving behaviors. This study, as a result, demonstrates proposed algorithms and techniques with these three behaviors.
To geometrically model a real-world road network, this study chooses Skeleton Google MAP image as the dataset. A semi-automatic extraction process is proposed to extract road centerlines of the road network from the Google MAP image. Road elevations are added later to generate 3-D centerlines. In order to convert these 3-D centerlines to 3-D geometric models, curve fitting was applied to construct a smooth spline to approximate the series of scattered points. As a result, reproducing geometrical structures of roadways is facilitated greatly. Much less human interferences were needed.

Furthermore, this study implements two levels of topological structures to represent topological information about the virtual environment comprehensively. The global level topological structure is a topological representation of the road network. Nodes of the global topological structure are components of the road network: road segments, intersections, ramps, bridges, etc. However, agents cannot be guided specifically from one point in the virtual environment to another due to the missing of local details. As a result, the local level topological structure is proposed. Each global topological node has a local topological structure attached. Local topological nodes are components of the global topological node. Two levels of topological structures are therefore inter-connected.

The need of storing geometrical and topological information about the environment results in the implementation of the database. All objects presented in the virtual environment have corresponding records in the database. Different tables are created to store different groups of objects. Inserting, removing and updating the table are completely automated by a JavaScript program.

With both geometrical and topological information available, this study proceeded with enabling autonomy in dynamic agents. A combination of a perception model, a communication model and a decision-making model was proposed to achieve the goal of modeling autonomous objects that behave intelligently. Models are attached to autonomous objects that represent people and vehicles in a virtual environment. This enables such an autonomous object to be an independent entity that is self-motivated and self-controlled.
The perception model enables the ability of vehicles to perceive their surrounding environments in real-time. When the program initializes, all information stored in the database was loaded to in-memory arrays. Information about static objects is loaded into static arrays, while information about dynamic objects such as vehicles is loaded into dynamic arrays which get updated in real-time. The process of “perceiving” is thus the process of autonomous vehicles receiving information from in-memory arrays on an as-needed basis. Autonomous vehicles are capable of getting all information of static objects in the driving environment.

The communication model improves the ability of autonomous vehicles to understand driving conditions by enabling effective inter-vehicle communications. It builds communications between autonomous vehicles by referring to the process of real communications. Autonomous vehicles know clearly about local traffic conditions as a result. Working in connection with the communication model, the decision-making model guarantees that autonomous vehicles act/react properly and quickly to changing circumstances. The decision-making trees at the global level and local level are a beginning to modeling the process of human decision making by incorporating a group of logic rules. The communication model and the decision-making model both can be revised easily to be adapted to new environments. By establishing new logic rules, the decision-making model is capable of handling new driving conditions.

With the implementation of these two models, the following can be achieved:

1) Simplified reproduction of real traffic in urban areas. Nowadays, traffic in cities is heavy. A great number of vehicles/pedestrians are involved. To reproduce even part of these vehicles and pedestrians in a driving simulator could be quite difficult. Without inter-vehicle communications and self decision-making, designers may have to organize the traffic manually. Other than designing global routes for autonomous vehicles to follow, designers must also take local conditions into account. Workloads increase considerably as the complexity of traffic increases.
2) Enhanced immersion for the human operator in a driving simulator. Thanks to the improvements of intelligence and autonomy, autonomous vehicles can behave more naturalistic and flexible. Also, because each autonomous vehicle acts and reacts with regard to its instantaneous environment, it would not behave in the exactly same way (unless wanted by the designer) every time the simulator runs. Users who drive in the driving simulator for multiple times would be less likely to get bored in repeated scenarios.

3) More reliable testing results. Because of the enhanced immersion of the human operator and improved reality of the simulator, driving tests done with driving simulators would be expected to be more reliable and convincing.

6.2 Future work

Future research could consider the following unresolved/neglected questions in this thesis.

First, geo-specific modeling of other environmental objects presented in virtual environments such as buildings is needed. This study tries to reproduce the road network concisely, while spend less efforts on reproducing other environmental objects. To increase the level of visual fidelity, more data about environmental objects has to be introduced in the modeling process (Coelho, Bessa, & Augusto, 2007).

Second, all autonomous vehicles in the above models behave at the same skill level. In other words, varieties of human drivers are not reflected in our models. Aggressive drivers should behave more aggressively as compared to moderate drivers. Skilled drivers surely have better controls over vehicle as compared to novice drivers. Also, autonomous vehicles perform ideally. For example, their deviations to the center of the lane do not vary. When cross intersections, vehicles follow possible routes only. Also, the conformation to traffic rules is a hundred percent in the current model, which is not necessarily true in the real world. More unexpected scenarios would be generated if autonomous vehicles were allowed to violate traffic rules occasionally.
Third, a human operator can be added to the virtual application to interact with autonomous vehicles. Since skill levels, emotional characteristics, driving habits of human operators vary greatly, the introduction of a human operator is a great way of testing the level of autonomy of simulated vehicles. Human operators could add both variety and unpredictability to the virtual application, and therefore help fine tuning the perception, communication and decision-making models.
REFERENCE


60. Willemsen, P. J. (2000). *Behavior and Scenario Modeling for Real-time Virtual Environments*. The University of Iowa.


