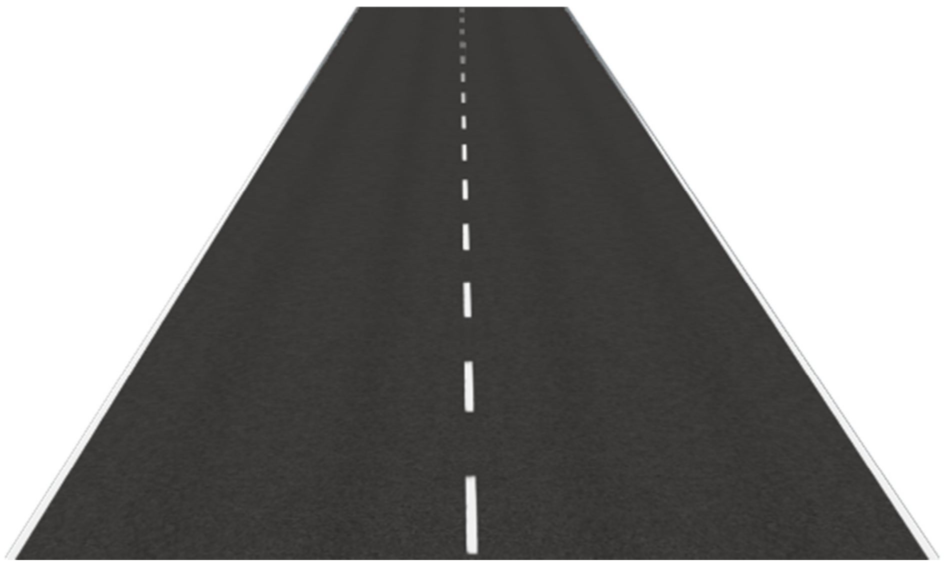
CHALMERS | UNIVERSITY OF GOTHENBURG



Open-source road generation and editing software

Master of Science Thesis in Computer Science

Dmitri Kurteanu
Egor Kurteanu

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
Göteborg, Sweden, June 2010

Open-source road generation and editing software

Dmitri Kurteanu
Egor Kurteanu

Examiner: Fang Chen

Chalmers University of Technology
University of Gothenburg
Department of Computer Science and Engineering
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Department of Computer Science and Engineering
Göteborg, Sweden June 2010

This thesis is the result of collaboration of two students:

Dmitri Kurteanu - student at the University of Gothenburg, Computer Science master's programme. Dmitri focused on the communication between the application's core systems, graphical user interface, road structure functionality and xml writing.

Egor Kurteanu - student at the Linköping University, Advanced Computer Graphics master's programme. Egor focused on the xml parsing, road structure, road and terrain geometry generation and helpers.

Abstract

The increasing use of simulators in the industry and various scientific facilities stressed the lack of simple and accessible content creation tools for these simulators. The goal of this thesis was to study the domain of driving simulators and produce an application, which can be used to generate logical and geometrical road data. This paper presents the details of the standardized format used to store the logical road description as well as the process and the problems encountered during the development of the application.

Keywords: **driving simulator, road generation, OpenDRIVE**

Acknowledgements

We would like to express our gratitude to our supervisor, Henrik Lind, who advanced the idea for this thesis and offered invaluable support and guidance during the work on the project.
We would like to thank Martin Fischer from VTI and Per Nordqvist from Volvo Technology Corporation for their constructive feedback and suggestions, as well as for helping us with testing and evaluation of the application.
We thank our relatives, friends and colleagues for their support, numerous suggestions and for devoting their time to help us test and improve the application during its development.
Lastly, we would like thank our examiners, Fang Chen and Björn Gudmundsson for their support and for the help they provided to improve this paper.

Contents

List of Figures

Chapter 1

Introduction

This thesis work will be focusing on the development of an application, designed to help engineers create a logical description of a road structure as well as the corresponding visualization data, to be used in driving simulators.

When we say “logical road description”, we mean a dataset that describes the road in a way that can be read and interpreted by a driving simulator. This description covers all the properties of a road, such as the number of lanes, elevation, traffic direction, but does not include the visual description of the road.

By “visualization data” we mean the data, such as the 3D-geometry and textures that is used by the visualization system of the simulator to display the road and its environment.

1.1 Background

With the increasing use of computer technology in the industry and research facilities, various simulators started to gain popularity for their undisputable ability to provide a relatively cheap and safe way to verify and test a multitude of scenarios.

Simulator - a device that enables the operator to reproduce or represent under test conditions phenomena likely to occur in actual performance [5].

A multitude of simulators exist nowadays. Each simulator is designed for one or multiple specific purposes, which include civilian and military personnel training, entertainment, engineering and manufacturing testing and verification, financial planning, etc.

For this specific thesis we will be focusing on driving simulators, and more specifically, on input data generation tools for driving simulators.

Driving simulators are most commonly used to test the driver's behavior in various scenarios and study the different factors that might influence this behavior, as otherwise those tests would be dangerous to perform in real environments.

To be able to function, driving simulators require various input data, such as logical road description, 3D-environments, road surface definition, etc. To create these, special tools are used. They generate and store the required information in various formats, which are then loaded into simulators.

1.2 Problem

The problem is the lack of accessible and easy to use solutions that are targeted, specifically, towards the generation of logical road description and visualization data, used in simulators.

1.3 Goal

The goal for this thesis is to design and develop a road generation application that could produce logical road description and provide options to export 3D-visualization data that could be used in simulators with no support for on-the-fly geometry generation.

The application should:

- be accessible
- have an easy and intuitive user interface
- provide exporting options for the 3D-visualization data
- be released under Open Source license

The later would allow a continuous development and patching of the application by third party developer teams. That, in its turn allows the application to:

- support features required by the majority of users
- provide automation to some of the raw functionality and processes of the initial release
- apply performance improvements for both geometry and logical data computation

The targeted OS is Microsoft Windows, although future ports to Linux might be possible.

The application is targeted for user that has a good knowledge in the field of road planning and building. User should have a good understanding of terms used in this field as well as have knowledge of the rules applicable in specific road planning scenarios.

Chapter 2

Literature study

Before starting implementing the application, we had to study the related fields, as well as find the appropriate tools.

2.1 Logical road description standard

Due to a high number of companies requiring a similar set of input data for their simulators, most notably the logical description of the road network and road objects, several standardized formats were developed. The major examples of these standards are OpenDRIVE [19], developed by VIRES and RoadXML [24], developed by OKTAL.

With the introduction of these road description formats, road creation and evaluation tools, as well as geometry generation procedures, used by the companies that decided to switch to the new formats, became obsolete. This led to the necessity to develop new road generation and evaluation tools. Some companies provided their own commercial solution to cover the niche, though, not everyone accepted the offer and some of the companies had to implement their own tools to export and use the logical road data in the standardized format.

We have chosen OpenDRIVE format, as VTI [28], one of the companies we cooperated with during the thesis work is using the OpenDRIVE format for their simulator, and Volvo Technology [13], the other company we cooperated with, will be switching to OpenDRIVE soon.

According to the OpenDRIVE website [19], OpenDRIVE is a standard for logical description of the single roads and road networks developed by VIRES Simulationstechnologie GmbH in cooperation with Daimler Driving Simulator, Stuttgart. The idea behind this standard was to facilitate the exchange of data between different companies and simulators. The first public version was released in the first half of 2006. Since then, the standard was recognized by a large number of companies. The list can be viewed at the “Users” page of the OpenDRIVE website [19]. Today, these companies are not just users, but also contributors to the standard’s development.

OpenDRIVE implies an open file format that stores the data in a XML [14] format. The nodes of the XML file represent various logical features of the road, like geometry, various lane parameters, signals and objects. The complete specification of the OpenDRIVE format can be accessed from the official website’s “Downloads” section [20]. We present a simplified and hierarchically organized example of an OpenDRIVE structure showing some of the nodes (omitting others, which are not fully covered by the current thesis work) in Figure 2.1. A short description of the presented structure will follow afterwards.

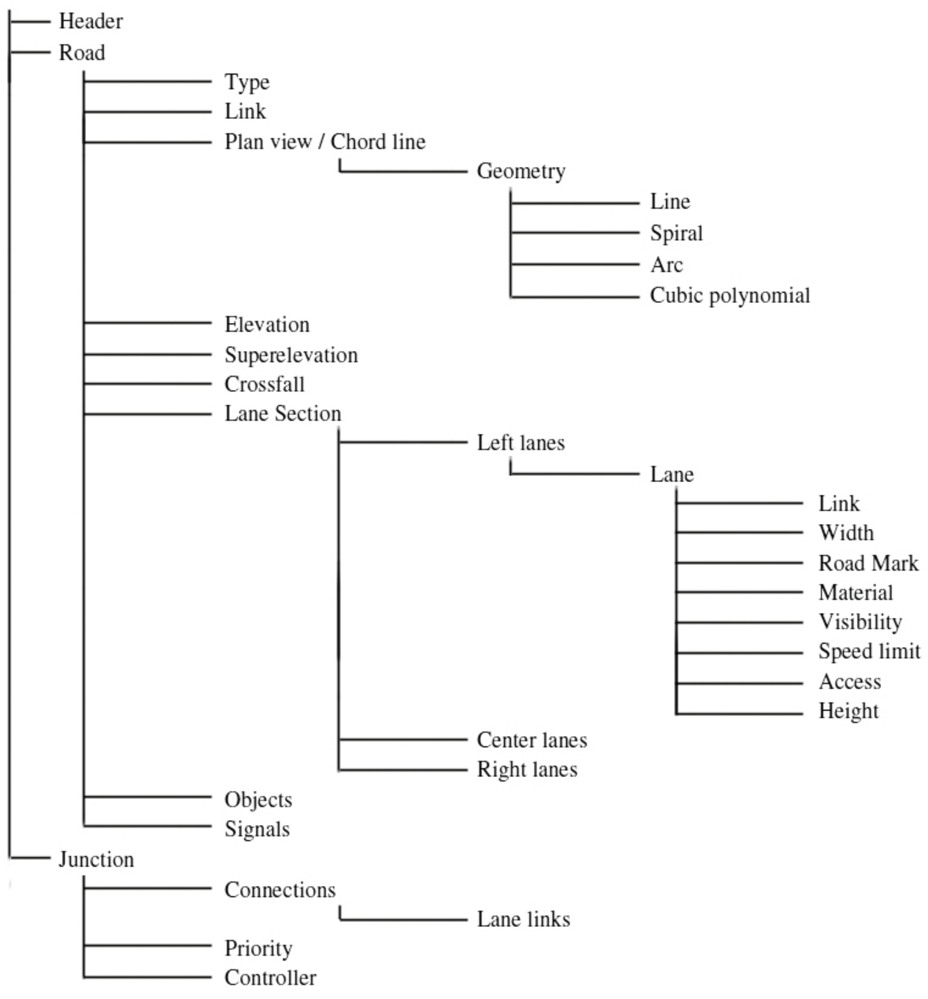


Figure 2.1: OpenDRIVE structure

At first, it would be useful to mention that the standard uses two coordinate systems in the specification: an Inertial Coordinate System [20, p. 8] and a *Track Coordinate System* [20, p. 9], Figure 2.2. Most of the road parameter records function in relation to the chord line or in the abovementioned Track Coordinate System by specifying an S-offset parameter in meters from the beginning of the road or lane section.

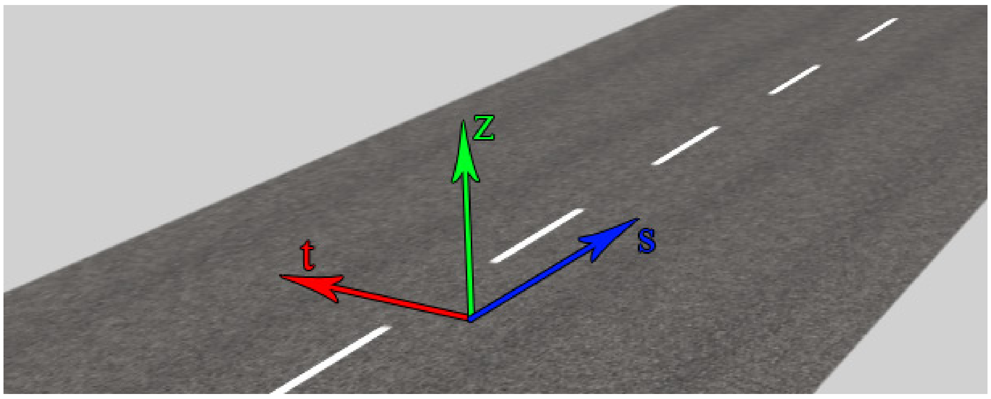


Figure 2.2: Track Coordinate System

Speaking about the key-nodes of the structure, there are two main nodes on the highest hierarchical level: *Road* and *Junction*.

As can be seen from the diagram above, a Road is a group-node that contains a number of different records that define various parameters needed to logically describe a road segment.

In order to logically connect several Road segments together, a *Link* record is required. It defines the *Successor* and the *Predecessor* of each road. In case this simplified linkage form is not enough, a *Junction* record should be used (described later).

One of the most important children of the Road node is the *Plan view* or the *Chord line* node with its *Geometry nodes*. Geometry records, define the layout of the road segment's chord line in the plan view. The Geometry node can be of different types: a *Line*, defined just by the heading and length; an *Arc*, defined by its curvature; a *Spiral*, which is a transition curve with linear curvature change, defined by its start and end curvatures. The above mentioned elements are described in detail in the "Design and Development" chapter.

In order to set the elevation at a certain point along the chord line, the *Elevation* node is used. There are also two parameters that define the lateral elevation, or the elevation at the cross section of the road, used for drainage or in the case of curves. These are called *Superelevation* and *Crossfall* [20, p. 13].

Another record with increased importance is the *Lane Section* group-node. It contains the records for the left, center and right lanes of the road. There might be multiple Lane sections along the chord line for a single road. Besides the explicit grouping of the lanes under a Lane section into left, right and center, the lanes have a numerical index, with the zero at the central lane (the chord line itself), ascending to the left and descending to the right. As a result, all the left lanes along the chord line have a positive index and all the right lanes have a negative index. A more detailed description can be found in the specification [20, p. 11,12].

The *Lane* record, under the Lane section, is a group that might contain a number of parameters represented by various records like *Width, Road mark, Material, Access, Height*, etc. One other parameter of the lane is the *level record* that can be used to exclude the lane from the application of Superelevation and Crossfall.

A Width record can be applied to any lane besides the central one and has a form of a cubic polynomial. Multiple width record can be provided in order, for example, to define a smooth transition from zero to its full width for a newly created lane [20, p. 38].

A Road Mark record is used to define a line road marking type for the lane's outer border. Multiple records can be used along a lane section if a road mark type change is needed for a certain lane.

Material, Access, Height, etc. define respective parameters of the lane and can be added multiple times if a change of the parameter is needed for the same Lane Section.

The *Object* and *Signal* nodes are not fully covered by the current thesis work, but behave the same way as most of the road parameter records do.

Junction record is used to remove the ambiguities in the road linkage. The case when a simple Predecessor and Successor record is not enough to fully define the connection, appears in case of any most simple road intersection. A Junction is used to define the paths between all the in-coming roads on a per-lane level [20, p. 14,15]. The paths of a Junction record are actually records of Road type and possess all the properties that a Road record does.

The sample of an OpenDRIVE file can be found in Appendix A.1.

2.2 License

We decided to release the application under an Open Source license, as this would allow opening up the source code for anyone that would want to participate in the development and improvement of the application after its initial release.

Releasing this application under an open source license is beneficial in a multitude of ways:

1. Makes the application widely available .
2. Gives third party developers the right to fix and improve the initial functionality.
3. Gives third party developers the rights to add functionality for their specific needs.
4. Allows for a wide range of available third party libraries, released under open source licenses, to be used in the development, reducing significantly the development time.

General Public License (GPL) being the most used free software license [29] is the one we are aiming for, although "Lesser" General Public License (LGPL) is also a viable option.

2.3 Tools and libraries

As the initial project plan implied that the application should be developed for Microsoft Windows platform in the first place, the tools and libraries have been selected accordingly. Yet, we tried to choose them in a way that will permit the future developers and contributors to port the code to other platforms as well. At the moment, all the code and all the libraries support porting to Linux, Mac OS and other platforms.

2.3.1 Tools

- **Integrated development environment**

  There is a large number of various IDEs available on the market.

  We have chosen Microsoft Visual C++ Express [6] as the IDE for the project. One of the main reasons was our previous experience with this IDE. It is a free and lightweight environment with all the needed functionality for a project of a scope like ours.

- **Version control**

  In order to have full control over the code, collaborate efficiently and have the possibility to store and access the full list of changes for every step in the development, a version control system was required. From the two most recognized version control systems: SVN [1] and GIT [3], we chose GIT, as it implies that each user has a full copy of the entire repository which makes it easier and much faster to track and commit changes. This feature might sound as a negative aspect in terms of space requirements, but due to the extremely small size of the GIT repositories, no such issue arises.

2.3.2 Libraries

- **Graphical User Interface (GUI)**

  For the GUI, we looked for libraries with support for multiple platforms, which would allow a later porting of the application to Linux. The two major GUI libraries we explored where GTK+ [4] and Qt [9].

  Qt is a cross-platform application and user interface (UI) development framework, produced by Nokia's Qt Development Framework unit. Compared to other GUI frameworks, and specifically GTK+, Qt was chosen for a number of reasons, and mainly: its simplicity, rich and intuitive class library, readable and easily maintainable code, well-structured documentation, and a great community that could provide solutions and suggestions for most of the problems that could arise during the development.

  The provided documentation covers every single class, with good description and examples. Multiple tutorials are installed with the SDK, which also help in getting acquainted with the library.

  Readability of the produced code was an important factor, as the application was initially planned as open source, which implied that it would be developed by various development teams during its life-cycle, which would have to study the code before applying any changes.

  Qt also provided a number of licenses, under which the final application could be released. That included the GPL and, what is more important, LGPL license, which offered more freedom and the ability to build commercial applications, which, at some point might be a viable option.

- **Graphics engine**

  We spent less time looking for a graphics engine / toolkit than we did for the other libraries. After even a short research it becomes clear that OpenSceneGraph, being one of the world's most used and developed graphics toolkits, completely fulfills all our requirements. As it is stated on the official website [8], the toolkit is written in standard C++ and OpenGL, thus supporting the full list of platforms that it can run on. Besides that, the companies which we cooperated with during the thesis work use OpenSceneGraph as the graphical framework for their simulators.

OpenSceneGraph has a history of more than 20 years of development and a large base of users who contribute and develop the toolkit. Even though not every single class and method is documented properly, open source code, a Quick Start Guide book, lots of community created tutorials; a mailing list and numerous forums, all these sources present a good database of information that provides a solution to almost any problem that might come up.

OpenSceneGraph has a large number of Input / Output plugins which allows for easy loading of textures and objects, as well as easy exporting of created roads in various formats. It also has a large set of tools that simplify the procedural generation of the geometry for both: roads and scenery. The toolkit additionally provides a number of customizable solutions to interact and navigate the 3D world. What is most important, even if OpenSceneGraph does not fully support Qt at the moment, it has all the necessary modules to manually integrate OSG with Qt.

- **XML parser**
  In order to read and write the OpenDRIVE files, a XML parser was required. There are a lot of various parsers on the market. We chose TinyXML [11] as it is simple, small and easy to use. Besides that it is distributed under the zlib license [15] which means that it can be used with licenses that our application might get released under.

- **Math libraries**
  For some of the calculations that are described in detail in the Design and development chapter, an implementation of the Fresnel integrals calculation was required. We used the code from the Cephes Mathematical Library [2], being almost the only freely available option.

Chapter 3

Design and development

3.1 Methodology

3.1.1 Priority

After a study on available tools, having a well-defined goal and being able to assess an approximate amount of work that has to be done, we used the "Project management triangle" [27] to establish a priority, which we were trying to adhere to:

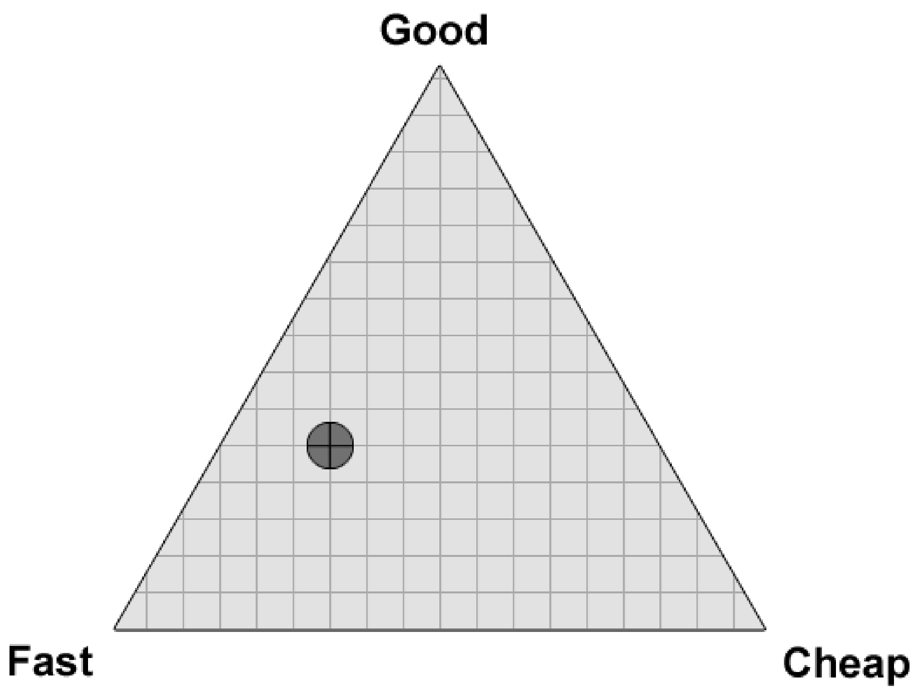


Figure 3.1: Project management triangle

In a project, the terms in the figure represent the quality of the product, the calendar time and the resources [27]. For our specific project the "Good" extreme represents the high quality and implementation of all the necessary functionality, "Fast" is our ability to finalize the project in time, and "Cheap" would be the low amount of work time allocated.

As can be seen in the figure, we could allocate a fair amount of work time. That allowed focusing on delivering the application by deadline, striving for a reasonable quality and functionality.

3.1.2 Development model

We used Rapid Application Development [23] model. We decided to go with minimal planning, focusing on developing a working prototype as soon as possible. After each iteration we would reassess the requirements, determining which functionality has priority, implementing it in the next iteration.

3.1.3 User Interface testing

Due to the strict time-frame available for the application development - we decided to test every major decision we make in relation to the graphical user interface. That would help us determine major interface flaws at early stages and avoid time-consuming changes later in development.

For each test we used up to four people. Testers had different backgrounds, from professional 3D modelers, to coders and casual users. That gave us, to some extent, and idea about a wider audience.

Every person was asked to provide freeform feedback about a specific interface element or change, and the overall experience. Although most of the testers were running the application at their own place, providing feedback remotely, for some of the tests, such as for 3D navigation, we had the opportunity to observe the testers locally.

3.1.4 Result evaluation

Before the final release, a test version had been evaluated in different conditions. A more detailed description of the process can be found in section "Evaluation".

3.2 Application structure

Working with multiple libraries at the same time, we tried to keep things separate, so, in case someone would decide to switch to a different library, it won't be difficult to do.

To achieve this level of separation, we decided to divide the application in 3 major layers:

1. Road structure
2. User interface
3. Geometry generation and rendering

Each layer communicates with other layers and with an OpenDRIVE file as shown in 3.2:

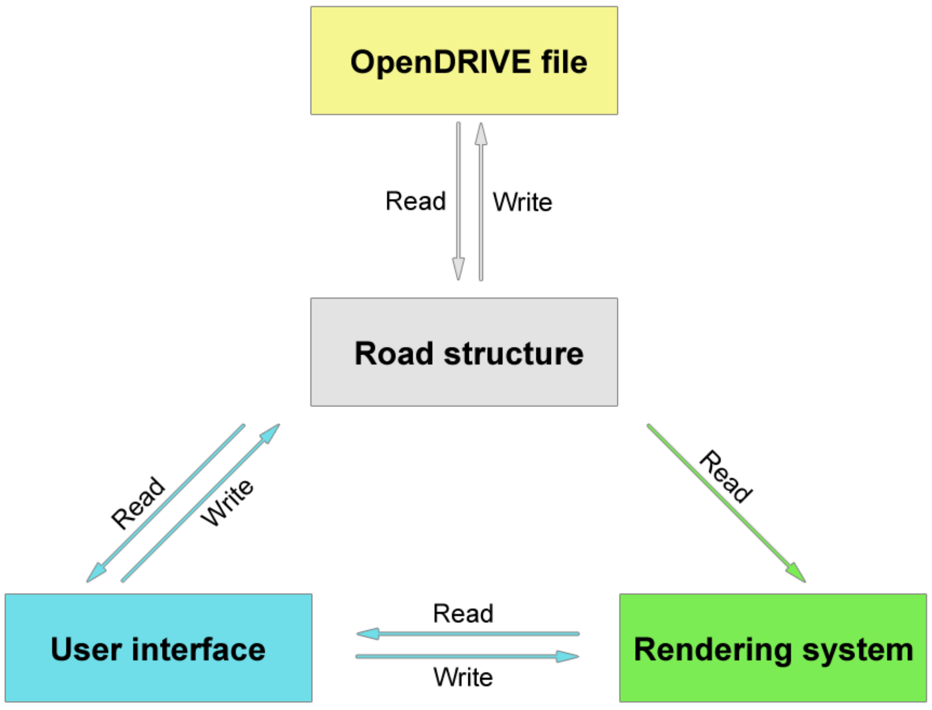


Figure 3.2: Layers communication

Each of the objects responsible for each layer is created on the same level. Due to the fact that some of the objects keep and use references of the other layers, it is important to create those objects in a specific order.

This way the road structure object, accessed and used by both user interface and rendering system is created first, followed by the rendering system and finally, the user interface application, which uses both road structure and rendering system.

3.3 Road structure

The road structure layer is represented by an object, which serves as a container for the logical description of the road. It holds the data in a hierarchical structure, similar to the one described by the OpenDRIVE format.

Aside from keeping the actual data about the road, the road structure object also holds several sets of methods used for the following purposes:

1. Loading data from a file
2. Saving data to a file
3. Retrieving raw data
4. Setting raw data
5. Retrieving evaluation data
6. Creating new records

Loading data from a file

When opening an OpenDRIVE file, an XML parser goes through the content, creating and filling in the objects for every record in the file.

Saving data to a file

When the user creates the road from scratch (by adding and filling in the records), or modifies the previously loaded file, the road structure provides ways to save the data, using the OpenDRIVE format. During this process, the application goes through the structure and writes its content, record by record, to the file.

It is also possible to save the road geometry to a file in the Open Scene Graph format, which could be later used by applications which are not able to generate the geometry on-the-fly, and would rather use a predefined scenery data.

Retrieving raw data

The structure provides methods to retrieve raw data, which is the actual information about the road records stored internally.

Setting raw data

Raw data retrieving methods are supplemented with methods used to set new values for the records in the structure.

Retrieving evaluation data

The structure also provides methods that return evaluated data. This kind of data includes information which is not directly provided by the structure, but is rather computed on-the-fly using the raw record data available internally.

An example of such data could be the space coordinates of the end-point of the road segment, which is not given by default, but could be computed using the geometry definition of the road segment and the road length.

This kind of information is used by the rendering engine to generate geometrical road data used to draw the road to the screen. Evaluation data could also be used by systems that require information about sections of the road which are not directly described in the structure and could be later used in road connection algorithms, junction generation and other.

Creating new records

Road structure contains all the necessary methods used to add new blank records, as well as clone the already existent records, inheriting all or most of their properties.

3.3.1 Geometry blocks

One of the decisions that we've made during the research of the OpenDRIVE format and the algorithms aimed at geometry generation was to combine some of the atomic OpenDRIVE geometry records into "blocks".

That decision was basically reasoned by the road building practice of using clothoids (special type of spirals) as transition links between straight lines and arcs [12]. According to this practice, every arc in the road geometry was preceded and succeeded by a clothoid, which shared the arc's curvature.

We designed the XML parser in such a way that it would read specific sequence of geometry records into a single geometry block.

Having a single geometry block gave us the ability to create, edit and delete road turns as unitary objects, rather than sequences of separate records with shared properties, thus reducing the time and effort required to define turns.

The use of geometry blocks is basically the only major distinction of the way the data is stored in the road structure of the application in comparison to the structure of the OpenDRIVE format.

3.4 Interface

While working on the interface, a lot of inspiration was taken from the design patterns, described in “Designing interfaces” by Jenifer Tidwell [26]. In the following chapters, those patterns will be highlighted, giving a short explanation on how they affect the user experience.

From the initial requirements for the application, it was clear that there is a set of interface elements that have to be incorporated into the application:

1. 3D viewport
2. Road record properties
3. Road record creation

3D viewport

The viewport would be used to visualize the road geometry, reflecting all the changes the user does to the road records.

Road record properties

The space allocated for the record properties would be used to visualize and modify the properties of the various road records from the structure.

Road record creation

Record creation section would provide all the tools necessary to add or remove new records to/from the road structure.

We used the “Center stage” pattern [26], giving the most important role in the application to the 3D viewport. This decision was basically dictated by a number of reasons:

- 3D viewport would be used to visualize the road, thus being the only reasonable mean to determine if the result is meeting the user requirements and is overall satisfactory.
- 3D viewport would be one of the main ways to navigate through the elements of the road structure, and select items for their modification.

- 3D viewport could be used to actually modify some of the road structure elements.

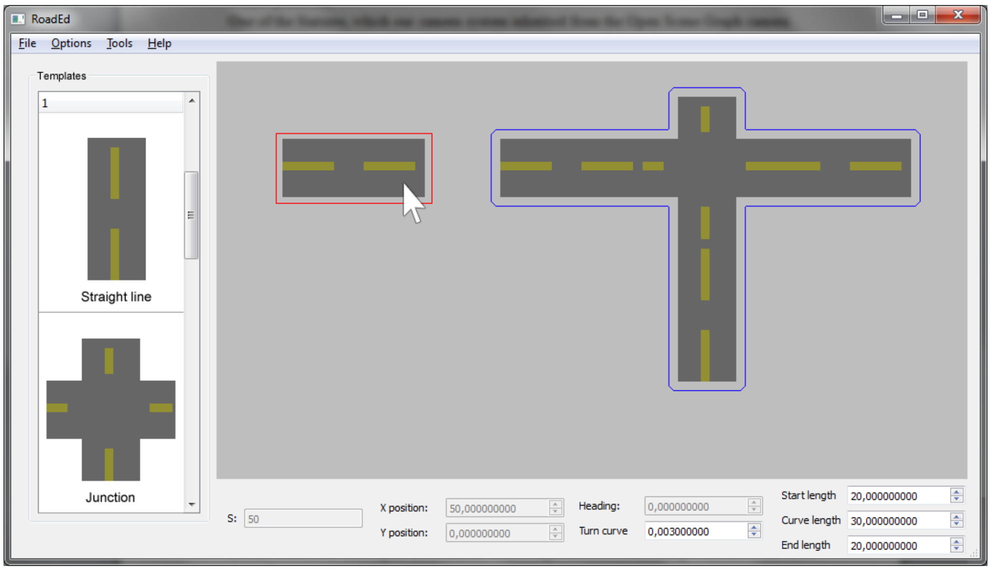
3.4.1 Initial ideas and mock-ups

We started by sketching some overall layouts, trying to think of additional interface elements we might require.

At first we had the idea of having a separate panel that would hold a library of road segment templates, part of which could be defined by default, and the other part could be set by the user.

This would allow operating with blocks of road geometry, quickly using the template panel and the 3D viewport to create turns and junctions. While on first launch of the application the library would contain only a limited selection of templates, later when the user would start building the road, custom templates could be added, based on the default ones with modified parameters. This would allow creating libraries of templates that could be used for different scenarios and could be transferred between users. A mock-up for that interface idea could be seen in Figure 3.3.

Though, while still working on the OpenDRIVE file format parsing, and researching the specification in detail, it became obvious that defining blocks won't be as easy as it seemed from the beginning. Some of the elements had a really high number of parameters or sub elements, which would differ for every single situation, thus creating ambiguity and complicating their use in templates.



S: 50
X position: 50,000000000
Y position: 0,000000000
Heading: 0,000000000
Turn curve 0,003000000
Start length 20,000000000
Curve length 30,000000000
End length 20,000000000

Figure 3.3: First interface idea

After gathering all the details about the way a road is defined, we started focusing on ways to represent the road structure on the lowest possible level, as this would allow the user to quickly access any single records or property stored in the structure. We thought that additional interface improvements could be added later. These improvements would make the interaction easier by removing some of the rarely used properties or automating some of the processes. Initially, we wanted to provide access to the smallest detail possible, even if that wasn't the best solution.

3.4.2 Road-tree and its problems

As the road structure was basically a hierarchy, we thought of ways to represent that. The tree-view widget, provided by the Qt library, seemed the best option available as it was intended specifically for that type of use. Visually it represents a hierarchical list, with items that can have one or multiple sub elements. Each item can be collapsed or expanded to hide or show its sub items.

Initially, while providing access to every single record stored in the structure, the road tree was hard to navigate, due to the high number of records. When more than one record was expanded, the amount of text in the panel, mixed with various levels of indentation, created a really confusing view, where it was difficult to distinguish and focus on the right item. This problem was reported by the majority of the testers, so we tried to find solutions as soon as possible.

One of the things we tried first was to add icons for every selectable item in the tree. Though, as it was stated by some of the people testing the changes - it actually made it worse. The addition of icons led to the shifting to the right of the items' names, which would line them up with the sub-items below, thus making it difficult to distinguish quickly between high and low level elements.

We decided to try to color code the tree. We defined a distinctive color for each type of item represented in the tree and set the background for each item to that color. It quickly became clear that this type of coloring would not work, as the resulted tree was too vivid and distracting.

Next try was focused on differentiating various hierarchy levels of items. This time we tried to choose a single color and modify its intensity for different levels of the items. Using this technique - the items on the first level would have the color with the highest intensity, while items that go lower in the hierarchy - would be "washed out", having a desaturated look. All three road-tree states could be seen in Figure 3.4.

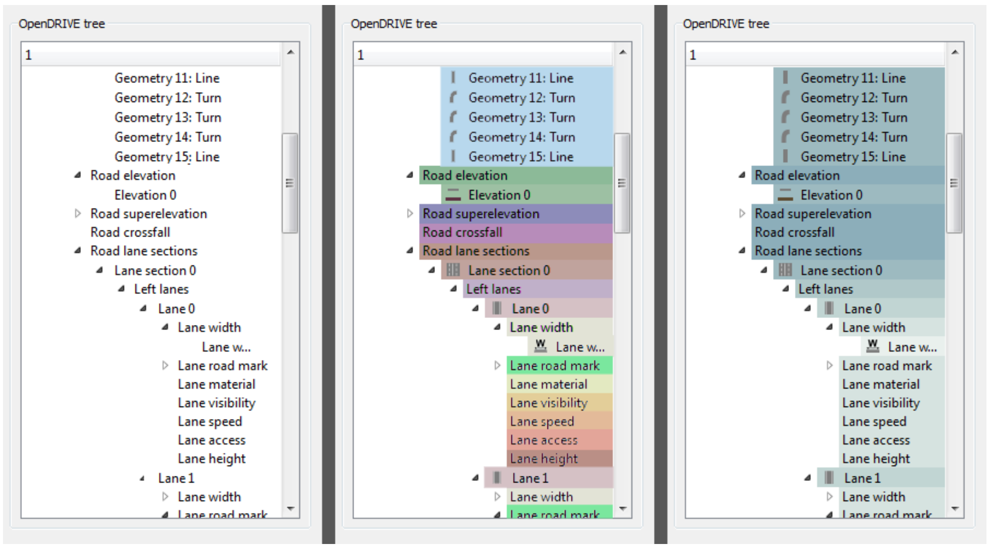


Figure 3.4: Road tree coloring

We received positive feedback from the test session, so we decided to keep the colors on. As an additional option, we included an action into the settings menu, which would toggle the tree coloring, in case someone would not like or would not find the color scheme appropriate.

3.4.3 Settings panel

To be able to modify the properties of a record from the road structure, a settings panel was defined. The panel contained multiple groups of properties, for each type of the road structure record.

When the user would select one of the items in the road-tree or in the 3D viewport, the system would determine which type of record was selected, displaying the appropriate set of properties, and filling them in with the data from the record.

Layout

At once - only one of the property groups would be shown. Every group was labeled and contained a form-layout, with property names on the left side, and the corresponding data controllers on the right side. This type of layout, also called a “property sheet” [26] was a good way to help the user build a mental model about a specific record, as it tells the user about the properties each record has and what are the appropriate values for these properties.

Complex properties

For some of the records that could contain more complex properties, made of a number of fields, we used a pattern called “Responsive enabling” [26]. The pattern would help the user determine which low-level fields are related to which properties, by enabling or disabling them together with the property they belong to.

Value limitations

To avoid erroneous input from the user for some of the properties, we either provided a dropdown list with available options, or limited the range of the input by defining the minimum and the maximum value. This helped making the application more stable by removing any ambiguity from inputting incorrect data.

3.4.4 3D-navigation

Navigation in the 3D viewport was on a priority list for us. We considered that the 3D viewport would be the most used interface element of the application, as it allows doing a number of important actions.

We thought that the 3D camera movement was the core for easy and intuitive interaction so it was important to find the best solution.

To be able to decide how the system should work, we studied several 3D editing applications: Gmax, Side Effects Houdini [10] and Milkshape 3D [18]. Any 3D creation suite could be a good example, as the main interaction between the user and the application is done through one or a series of 3D viewports.

In every 3D editing application we tested, we could distinguish 3 modes of operation of the 3D viewport:

1. Selection – clicking on a 3D object with the mouse would select it for future manipulation.

2. Object manipulation – dragging the object with the mouse would move the object in 3D space, change its shape, size or other properties.

3. Camera manipulation – moving the mouse would move the camera - changing the perspective from which the user watches the scene.

Each of the applications had its own way to switch between the modes. Some of them had different icons on the toolbar or keyboard shortcuts which would toggle one of the modes; others had those modes mixed and available, to some extent, at any time.

We liked the way SideFX Houdini [10] managed the modes: initially the viewport works in selection/object manipulation mode, but when the Space button is pressed and until it is released - camera mode is active. That seemed to be a very smart solution, as it provided a very quick way to activate and deactivate the camera mode, which is constantly used when working with the application. The Space button also seemed to be the easiest shortcut to remember, which in its turn removed the necessity for additional icons and buttons on the toolbar.

Camera movement

The next step was to decide on the way for the camera to be controlled. It was harder to research this aspect, as most of the 3D creation suites are intended to work with objects, thus their camera movement is centered on objects. What this actually means is that the camera will pivot around an object when being rotated.

Camera 1 The first camera system we implemented was based on this model. It was defined by position and a target point, around which it rotates. Moving along the ground plane implied the movement of both camera position and camera target. Zooming was implemented by increasing or decreasing the distance between the camera position and the target. Rotation was done by projecting the camera on the surface of a sphere, with the radius equal to the distance from the target.

The user would use the left mouse button while holding space to rotate around the target, right mouse button to move along the ground plane and middle mouse button (or both left and right buttons together) to move along the vertical axis. A simplified version of this camera model is presented in Figure 3.5.

In the implementation stages it seemed to be quite intuitive and handy. Later though, through testing, it became clear that this model of camera movement is inconvenient for the tasks we had. Testers wanted the ability to quickly zoom on a specific point, or move along the road curve. With the model we had at that moment - it was practically impossible or was very awkward.

The problem was basically explained by the inability to freely “look around”, as the camera wouldn’t turn left or right, but would rather rotate

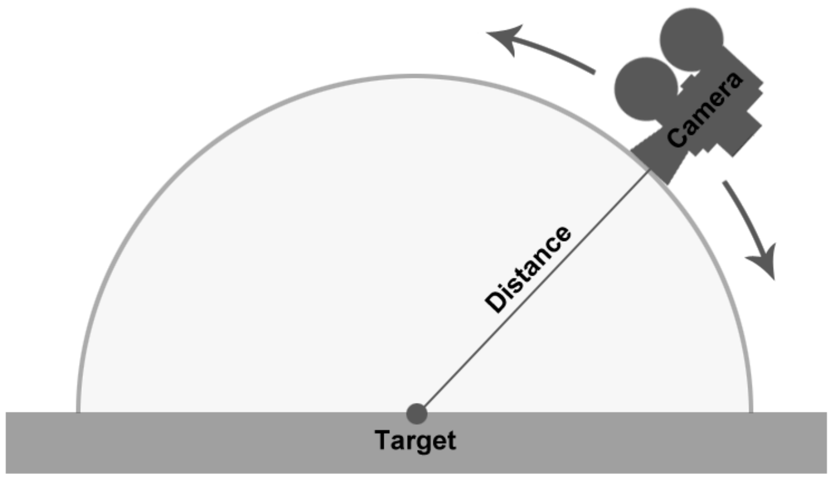


Figure 3.5: Camera 1

around a point in space, which was invisible and unintuitive to move. It became clear that we had to get rid of the camera target, moving the pivot-point back to the origin of the camera. This would allow rotating the camera around its center, rather than around some point in space, making it easier to move to a specific, well defined, point.

Camera 2 The new camera model, which we were aiming for, was in a way similar to the camera models implemented in First Person Shooter games (FPS), where the user is able to rotate the camera around the local horizontal and vertical axis and move in the specified direction along the ground plane. This type of controls allowed moving along a path, such as the curve of the road, by just "looking" in the direction one wanted to move, and "going forward".

In addition to these controls we added the ability to move along the vertical axis, increasing the height above the ground. The distance from the camera to the ground was used as a parameter for the movement speed along the ground plane. This way the user could "cover more ground" while being at a high altitude. Being close to the ground reduced the speed, which could be used for more accurate positioning.

The controls were distributed as follows: left mouse button would rotate the camera around, right mouse button would move the camera forward, backward and sideways, and middle mouse button would increase the altitude.

While holding both left and right mouse buttons, the user could move and rotate around at the same time. The new camera model is presented in Figure 3.6.

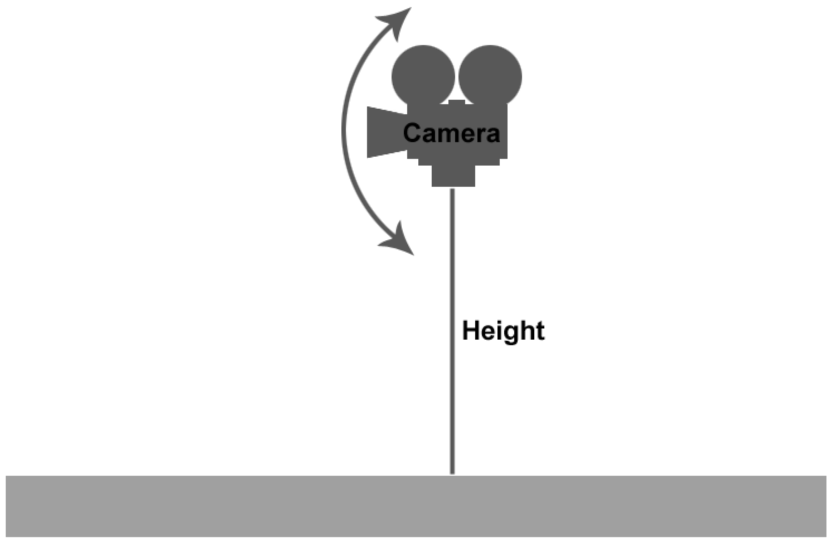


Figure 3.6: Camera 2

The tester liked the change, as the new camera model allowed more precise positioning and movement controls which were good for “road inspection”.

Dragging the terrain One of the issues pointed out by some of the users was the mouse movement direction in relation to the movement of the camera. To move to a specific point in space, some of the users preferred to move the mouse down, “dragging” the terrain backward, rather than actually moving the mouse forward to move the camera forward. This question remained unsolved, as different users had different experience with 3D applications, and it was important to keep the consistency between applications they use, thus keeping their habituation intact [26]. A solution was to add a flag to the options menu, which would toggle the mouse movement direction for user that would not prefer the default mode.

Home position One of the features, which our camera system inherited from the Open Scene Graph camera implementation, was the “Home” position. By pressing a special combination of keys, the camera would be moved to the default position in space, which was defined by the bounding box of the road network. This way, if the user got lost flying in 3D space, or wanted to go to the starting position, he could press “Space + H” to move the camera to a position above the ground, determined by the size of the road network, and rotate the camera to face the ground.

Selection

Another important feature of the 3D viewport was the ability to select objects by clicking on them in 3D space. This feature was added on one of the later stages and is described in more detail in section *Helpers*.

3.4.5 Item creation

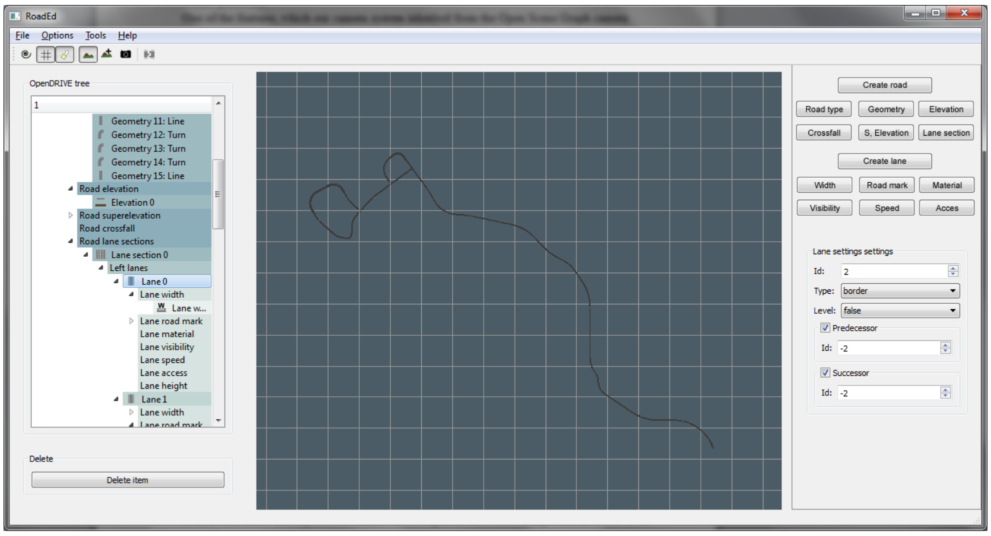
While at this point the user could successfully edit already existing roads, he wasn’t able to create or add new records. To address that - we created a set of buttons, each of which would create a record of some type.

One of the problems with record creation that we felt, was the number of various types of records that could be added to the road structure. We had to find a spot on the screen, where all these buttons would fit and would be easy to access.

In the settings panel

The first idea was to position the creation buttons into the settings panel, along with all the properties for each type of the record. This way, when the user would select, for example, a “crossfall” element - along with its properties, a “create” and a “delete” button would appear in the settings panel. “Create” button would add a new crossfall to the list and respectively “delete” - would remove the current item from the list. A preview of this layout is given in Figure 3.7.

The problem with such a layout is that it forces the user to select an already existing item to create a new one of the same type (as the settings panel shows item’s properties only when it is selected). Another problem is that we wanted to allow the user to create some types of records even when nothing specific is selected, thus it would require to add creation buttons even for empty items such as item containers and higher level items, which might have no relationship with the items to be created.

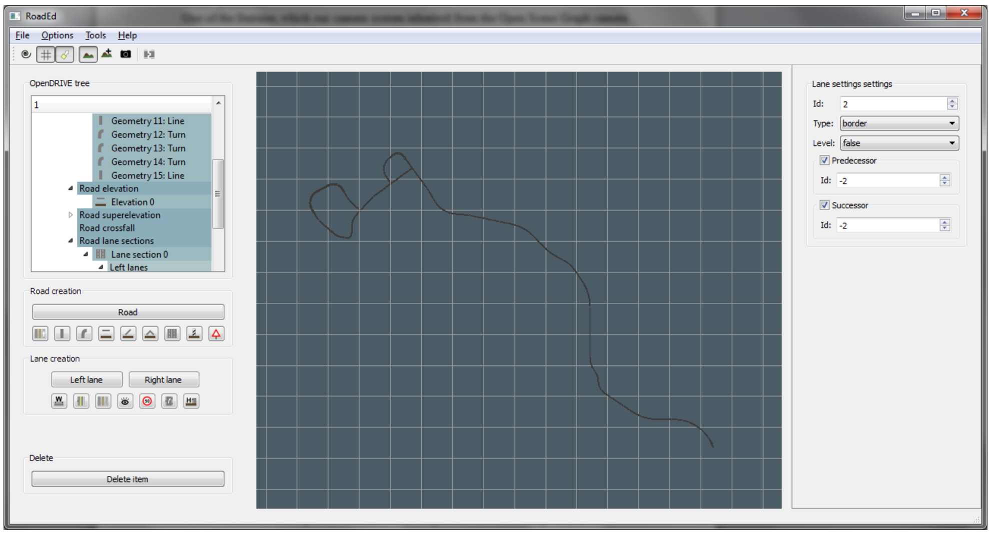


Id: 2
Type: border
Level: false
☑ Predecessor
Id: -2
☑ Successor
Id: -2

Figure 3.7: Creation panel 1

In the road-tree panel

We couldn't find a good solution for these problems, so we tried to group all the items of the same level, label the groups and add them sequentially to the panel that contained the road tree (Figure 3.8).



Id: 2
Type: border
Level: false
☑ Predecessor
Id: -2
☑ Successor
Id: -2

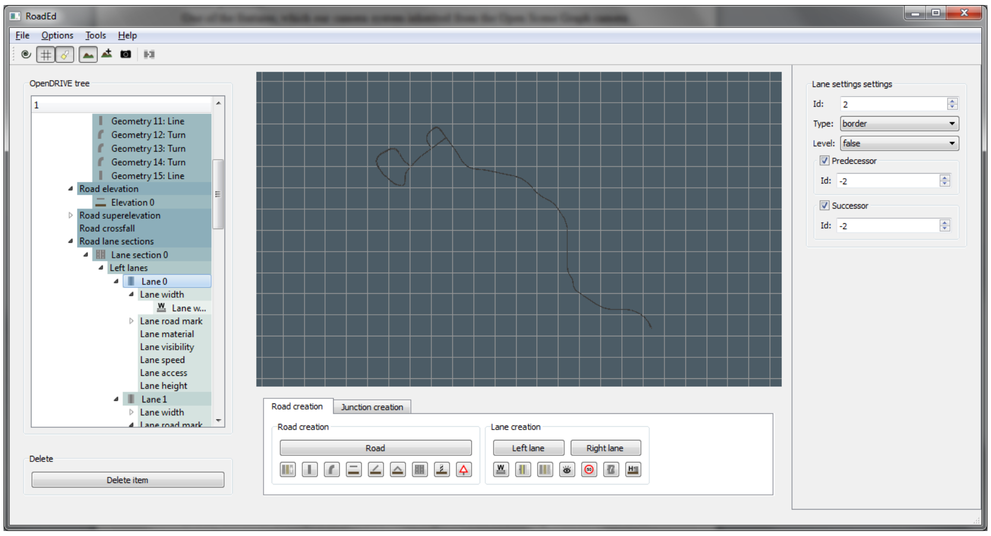
Figure 3.8: Creation panel 2

The new position instantly showed the flaw of the decision: due to high number of buttons and a high space requirements for the groups, the road tree area was reduced considerably, which made it almost impossible to work with the tree, as only a small portion of the records could be visualized at once.

A separate panel

We tried to analyze which portion of screen would be able to hold a high number of buttons and not impede the work. The solution that we found reasonable was to use a tabbed widget [26] at the lower part of the 3D-viewport (Figure 3.9). This part of the screen reduced the area for the 3D viewport, but it was a good compromise, as the 3D window could be scaled insignificantly, still keeping its functionality. Another aspect which was noted in testing feedback is that the viewport changed from vertical aspect ratio to horizontal, which seemed more convenient for the user.

The tabbed panel allowed dividing the button groups according to the hierarchy level of the items they create, as well as the relationship of the items. Two tabs were created, one for all the items related to road definition, and another one for the definition of junctions. At the same time, the road item tab had two groups, one for all the direct children of the "road" record and the road record itself, and another one for all the children of the "lane" record and the lane record itself. Buttons, responsible for creation of items with high number of children, like the mentioned "road" and "lane" were larger in size and labeled. Buttons for smaller objects, such as "elevation" and "crossfall" were smaller in size, and defined by an icon.



1
- Geometry 11: Line
- Geometry 12: Turn
- Geometry 13: Turn
- Geometry 14: Turn
- Geometry 15: Line
- Road elevation
  - Elevation 0
- Road superelevation
- Road crossfall
- Road lane sections
  - Lane section 0
    - Left lanes
      - Lane 0
        - Lane width
          - Lane w...
        - Lane road mark
        - Lane material
        - Lane visibility
        - Lane speed
        - Lane access
        - Lane height
      - Lane 1
        - Lane width
        - Lane road mark

Id: 2
Type: border
Level: false
[✓] Predecessor
Id: -2
[✓] Successor
Id: -2

Figure 3.9: Creation panel 3

An idea that we thought would allow to reduce human confusion - was to disable buttons for items that cannot be created at a specific moment (when an item is selected, which does not support records of a given type) [26]. A good example for this feature is the situation when the "lane section" button is disabled until the user creates a road. When a road is created and selected, the “lane section” button becomes active, and when clicked - creates a new lane section record, which in its turn - enables the second group of buttons, responsible for creating lanes and their properties.

Another positive aspect of using the tabbed panel is the additional space which was made available. It could be used for additional groups of items, or libraries of templates that could be implemented in the future, as well as for the special tools, such as the road connection tool.

3.5 Geometry generation

In the current chapter we present various aspects of the road geometry generation process, used algorithms, encountered problems and found solutions. The chapter is divided into several sections, reflecting the entity types that have to be visualized. The list below displays the topics of the abovementioned sections.

- Road Geometry
  - Chord line and general road shape
    - Geometry types
    - Detailed review of the transition curve
  - Road properties
  - Lanes
  - Triangulation
  - Junctions
- Helpers
  - Road record helpers
  - Road helpers and selection
  - Grid and reference plane
- Scenery
  - Heightmap based landscape generation
  - Road and landscape blending

3.5.1 Road geometry

The generation of the geometry of the road is a high-priority feature of the developed application as it is used to visualize the creation progress of the logical road description. Besides, it might be used together with the logical data in the simulators as it guarantees that the geometry will exactly match the road description.

As it was stated previously in the OpenDRIVE description section, the standard defines all the road's parameters based on the chord line and in the track coordinate system. In order to draw anything on the screen, its geometry has to be created relative to the center of the 3D world. This means that a conversion between the two coordinate systems is required.

Because the roads in OpenDRIVE are not described explicitly along their length, the entire geometry generation system is based on samples. This means that in order to draw a road segment, the road structure is sampled a number of times, starting from the beginning of the road, till its end, with a predefined, configurable step. This also means that the number of samples (or the size of the step) defines the geometric resolution of the road and is directly proportional to the number of triangles that are being generated. To conclude the points mentioned above, each sample is a road's cross-section line of vertices with the properties (coordinates, color, normal, texture coordinates) derived from the road's parameters that apply to the current sample position along the chord line. As a result, these lines of vertices are then triangulated to form the roadbed.

Below is an overview-list of actions that are executed in order to generate the road geometry:

1. Fill in the geometry arrays:

   a Sample the chord line and get world-coordinates for current step.

   b Sample the various road parameters.

   c Sample the current lane section.

   d Calculate the offset angles of the road if the superelevation and crossfall are applied.

   e Go through all of the lanes on both sides of the road and for each lane of the current sample, set a pair of vertices at the left and tight edge of the lane. Besides the position, the normal, color and texture coordinates are also set at this step.

   f Apply lane specific properties.

2. Triangulate the road.
3. Triangulate the road marks.
4. Fill in the road mask vertex array assigned to current road. This is needed for the scenery generation (described later in this chapter).
5. Assign the texture to the roadbed.
6. Assign the texture to the road marks.

The sections that follow describe in detail the problems and solutions for the points above.

Chord line and general road shape

The standard specifies that the road's chord line is described by a number of geometry records. In the terms of visualization, each of these records has:

- An s offset, which defines where the current chord line segment starts in relation to the full road length.
- A world coordinate that defines the starting position of the chord line segment described by the record.
- A heading that describes the initial orientation of the chord line segment.
- The length of the chord line segment.

Another important feature of each geometry record is a child node that describes the type of the current geometry. There are several types of geometries. The list includes:

- **Lines** – have no additional parameters.
- **Arcs** – have a constant curvature parameter.
- **Spirals** – have two curvature parameters for the start and the end of the segment.
- **Third order polynomials** – have a,b,c,d parameters that define the polynomial (Not covered in this thesis).

To conclude the above mentioned, a chord line is combination of alternating geometries with different lengths. Each such geometry derives its initial position and heading from the end position and heading of the preceding geometry.

In order to sample the chord line and get the coordinates and the heading that are later used for the rest of the road geometry computations, a number of evaluation methods is used. These methods might also be considered *coordinate system conversion methods*, as they are used to convert the track coordinate system into the world coordinate system. The track's s-offset parameter is provided to the road's geometry sample method which is then used to find the appropriate geometry record that the sample position belongs to. If the record is found, depending on its geometry type, a corresponding coordinate system conversion method is called.

Line Geometry In case of a line geometry (Figure3.10), its sample coordinates are found using the following expressions:

$$x_{Sample} = x_{Geom} + \cos(hdg) * (s_{Sample} - s_{Geom}) \quad (3.1)$$

$$y_{Sample} = y_{Geom} + \sin(hdg) * (s_{Sample} - s_{Geom}) \quad (3.2)$$

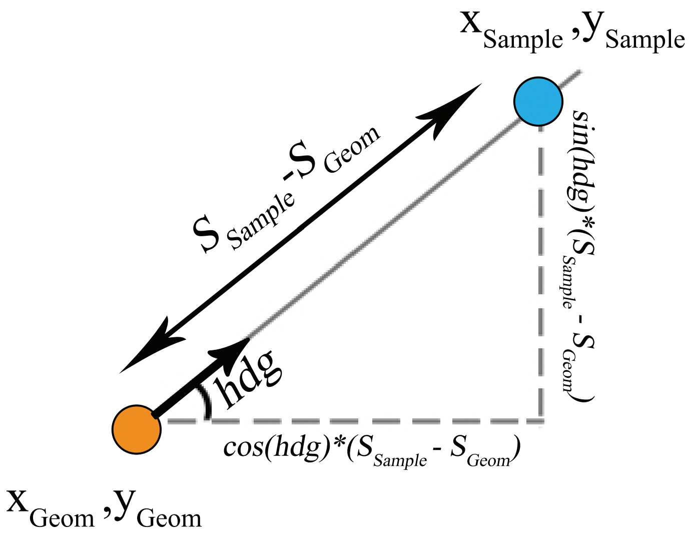


Figure 3.10: Line Geometry

Where x_{Sample} and y_{Sample} are the returned x, y coordinates; x_{Geom}, y_{Geom} - the x, y coordinates of the geometry / chord line; *hdg* is the geometry's initial heading; s_{Sample} - the method's input parameter that defines the current sample position; s_{Geom} - the chord line's s-offset.

The sample heading, in case of a line geometry is the same as the initial geometry heading.

Arc Geometry An arc chord line type is bit more complicated than the line (Figure3.11). Its sample method has an input curvature parameter. Knowing the curvature, we compute the radius of the arc as:

$$radius = |\frac{1}{curvature}| \tag{3.3}$$

Depending on the curvature sign, and knowing the initial heading and arc radius, we compute the center of the arc:

$$x_{Arc} = x_{Geom} + \cos(hdg \pm \frac{\pi}{2} - \pi) * radius \tag{3.4}$$

$$y_{Arc} = y_{Geom} + \sin(hdg \pm \frac{\pi}{2} - \pi) * radius \tag{3.5}$$

We then use the following equation to compute the central angle of the arc:

$$\Theta_{central} = \frac{(length_{arc})}{radius} \tag{3.6}$$

$length_{arc}$- position of the current sample relative to the initial offset of the geometry in track coordinate system.

In order to compute the angle of the sample point (*sample angle*), we add the found central angle to the initial position angle. Then, the sample angle and arc's center position are used to compute the coordinates of the current sample:

$$x_{Sample} = x_{Arc} + \cos(\Theta_{central} \pm hdg \pm \frac{\pi}{2}) * radius \tag{3.7}$$

$$y_{Sample} = y_{Arc} + \sin(\Theta_{central} \pm hdg \pm \frac{\pi}{2}) * radius \tag{3.8}$$

The heading is found using the following equation:

$$hdg_{Sample} = \Theta_{central} \pm hdg \tag{3.9}$$

Spiral Geometry (Transition curves / Clothoids) A spiral record has two parameters that hold the initial and the end curvatures of the spiral. The description in OpenDRIVE specification states that the present spiral implies a linear curvature change between its start and end. The present spiral, being an *Euler Spiral* (or a Cornu spiral, a Clothoid or a spiros) [16], is used as a transition curve (a transition spiral or a spiral easement) [25] to reduce the sharp changes in centripetal acceleration when approaching a segment of the road with constant curvature.

In the terms of OpenDRIVE, the Euler spiral is used between a line and an arc with one of the curvature parameters corresponding to the arc's curvature, and the other parameter corresponding to the line's curvature, which is zero.

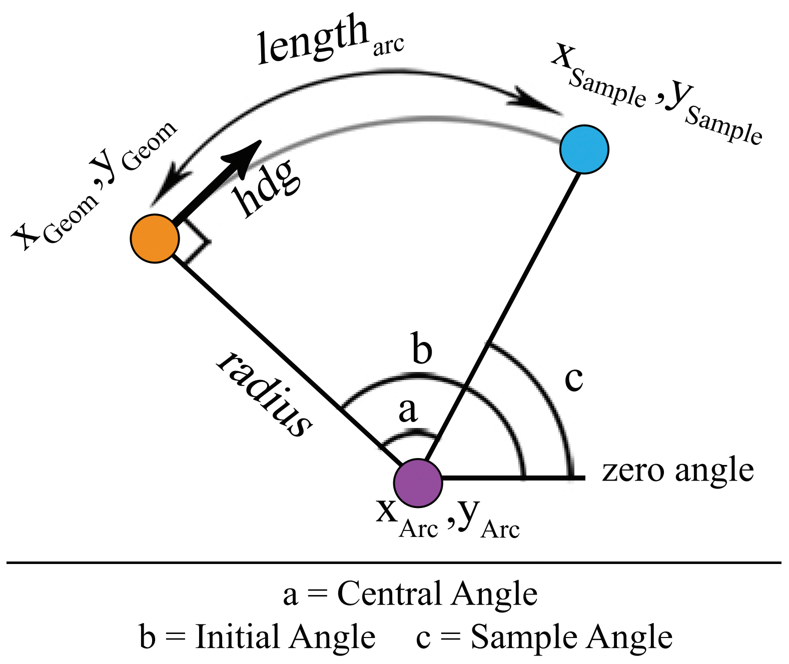


Figure 3.11: Arc Geometry

In order to find the sample coordinates and heading at a certain s-offset, there is a need to solve the Fresnel Integrals that describe the normalized spiral:

$$x_{Sample} = \int_0^L \cos(L^2)\,dL \tag{3.10}$$

$$y_{Sample} = \int_0^L \sin(L^2)\,dL \tag{3.11}$$

Where L is the s-offset along the spiral.

In order to get a normalized spiral, L should be multiplied by a normalization factor which is calculated as in the next equation:

$$L_{Normalized} = L * \frac{1}{\sqrt{2R_{Curve} * L_{Curve}}} \tag{3.12}$$

R_{Curve} is the final radius of the spiral (or the radius of the arc that the spiral is connected to). L_{Curve} – the length of the spiral.

The equations above are enough to find the coordinates for a point at any distance between zero and the length of the spiral. According to a paper written by Raph Levien [22], Cephes library [2] provides an implemented solution which is accurate and fast (its time is comparable to the time which is needed to compute the standard trigonometric functions). The computation is divided based on the function's argument value range. For values below 1, a

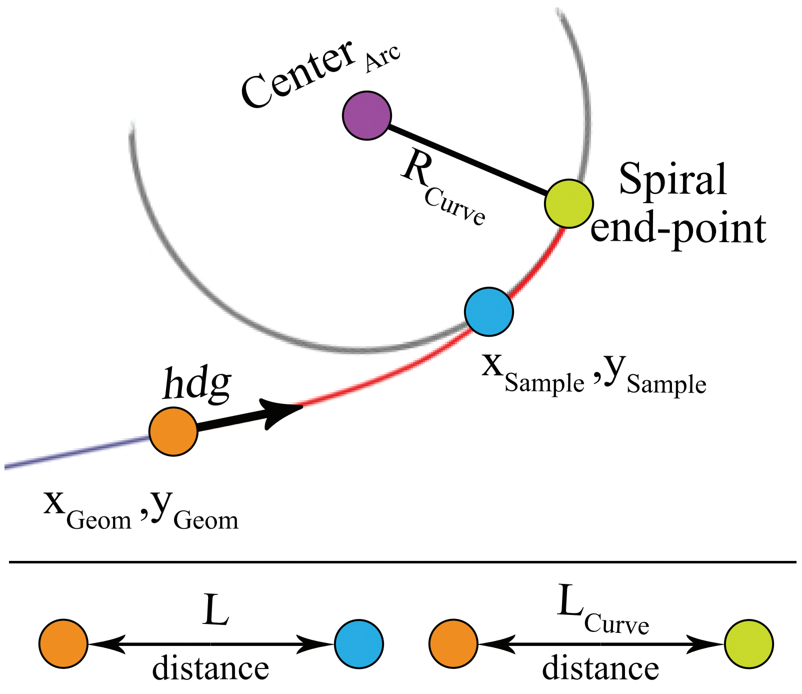


Figure 3.12: Spiral Geometry

power series is used to evaluate the integrals and for the values above 1, some auxiliary functions $f(x)$ and $g(x)$ are used, changing the equations into:

$$x_{Sample} = \frac{1}{2} + f(L) * \sin(L^2) - g(L)\cos(L^2) \tag{3.13}$$

$$y_{Sample} = \frac{1}{2} - f(L) * \cos(L^2) - g(L)\sin(L^2) \tag{3.14}$$

We calculate the sample heading using the following equation. We can use the equation as the curvature of the spiral is linearly related to its curve length.

$$hdg_{Sample} = L^2_{Normalized} \tag{3.15}$$

All the statements above apply to the situation when we have a curvature change from zero to a certain value, or if the spiral is used as a transition curve between a line and an arc (line to arc direction). A problem arises when there is a need to calculate the coordinates in the inverse direction, from a non-zero curvature to a zero curvature, which is used to ease the turn exits (arc to line direction).

In order to apply a global transform for the spiral (so that it is possible to place it between an arc and a line), there is a need to translate the spiral and rotate it around its origin. In case of an inverse spiral (a spiral that provides a transition from an arc to a line), the center of rotation is at the end of the normalized spiral. This implies an additional step, a pre-computation of the Fresnel integrals with the parameter of the full length of the spiral. The result is used to compute the translation and orientation difference parameters. These, in their own turn, are used to offset and rotate the normalized Fresnel integral before computing the actual sample coordinates and heading.

The methods presented by the Cephes library proved to be accurate enough as the average error is rarely higher than $3.2e^{-16}$.

Road Parameters

Having the chord line coordinates and heading from the geometry sample, it becomes much easier to apply road-specific parameters. A number of evaluation methods are used to get the elevation, superelevation and crossfall values at the sample position. These methods operate in the same fashion as the geometry evaluation methods - they use the s-offset parameter to find the corresponding record and afterwards process the record to get its value. The required values are calculated according to the OpenDRIVE specification [20, p. 30,32,33].

In order to optimize the process, the sine and cosine values for the superelevation and crossfall angles are pre-calculated and used on necessity.

Lanes

As it was stated previously, the entire road geometry generation system is based on the sampling technique. Each step, both the chord line and the road parameters are sampled. This information is enough to visualize the chord line (which is actually used as a helper to display the road layout and the geometries of different types), but is insufficient to compute the actual geometry of the road, as there is no information on lanes, thus, most importantly, there is no information on the width of the road.

We sequentially tried two different techniques to combine the road and lane parameters in order to compute the road geometry.

The idea behind the first method was to optimize the face count by creating pairs of faces (a quad) just where they are required. This means that for each sample line, vertices are created just at the positions that define changes in the lane parameters or the road edges. For example, there might be just two faces (one quad) per a segment of the road defined by two sample cross-section lines, even if the road has 3 lanes for each of the road directions. To conclude, 6 lanes are defined by just 2 faces. If a crossfall record is added, the road triangles have to be split in two, having the origin in the middle of the road (as the crossfall is applied per road side). This doubles the number of faces. If some of the lanes, for example the sidewalks, ignore the crossfall parameter, as their keep level option is set to true, then an additional pair of quads is added for each of the sidewalks, increasing the road segment's face count to 8 faces: a pair of faces per each sidewalk and a pair for the two internal lanes for each side of the road. An example (showing less lanes) is presented in Figure3.13. The next situations are displayed (left to right): No special parameters; Crossfal record that divides the quad into two; Crossfall and a lane that keeps the level.

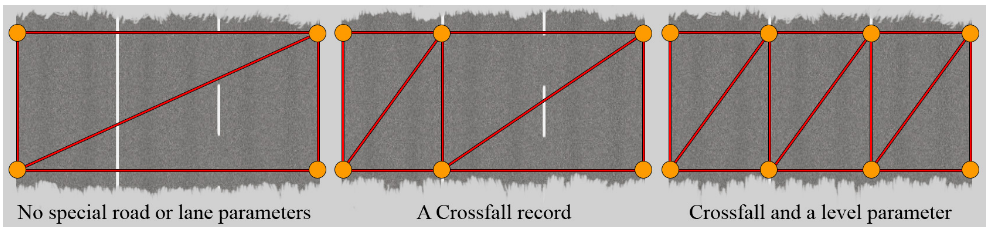

No special road or lane parameters
A Crossfall record
Crossfall and a level parameter

Figure 3.13: Optimized Technique

This method has a good optimization use, but brings a number of substantial flaws:

- There is no possibility to define different surface textures to the specific lanes of a group of lanes defined by a pair of triangles.
- There is no possibility to define different widths for a group of lanes described by a pair of faces.
- It is impossible to properly use texture coordinates, as for example, it is impossible to tile the texture over the faces if the texture has a number of types of entities on the same image.
- The worst case scenario brings in no face-count optimization but introduces a large number of additional checks and operations.

Because of these inevitable problems, we developed a different method that might seem to be also a more straightforward one. The second technique implies that each lane is explicitly defined by a pair of vertices (and thus, a pair of triangles per section at the triangulation step). This change instantly reduces all of the problems of the first method:

- By providing a pair of vertices for each lane, it is easy to define the appropriate texture by using multiple-entity road textures (that have multiple texture-objects on the same image) and shifting the texture coordinates to the right offset.

- Having the width record for a lane, it is extremely easy to define a width offset for a specific lane vertex pair.
- Each vertex at the edges of the lane has its own texture-coordinate. This allows us to tile just a small portion of the entire texture over the road in both directions.

The method's main flaw is a rather high number of faces (compared to the possible results of the first method) that is directly proportional to the number of lanes and the size of the sample step. Yet, this is an appropriate sacrifice as it allows for a precise graphical representation of the created road, which is the most important property of a road editor.

Lane sampling During the geometry arrays filling step, a very large number of lane specific parameters had to be operated on at the same time. A useful decision was to create a special structure that will store all of the lane parameters for all the lanes of the current sample - a *LaneSectionSample*. The feature of this structure is that it holds organized arrays for all the parameters of all the lanes of the lane section at the current sample position. These arrays are divided into the arrays for the left and right sides of the road. This makes it much easier to fill in the geometry arrays as it divides the process into two separate steps: from the central lane to the left and then from the central lane to the right.

The LaneSectionSample also holds the properties of the road marks. These are stated separately as there is a separate vertex and triangle array for the road marks. The process of road mark vertex and triangle generation is exactly the same as the one for the road: each road mark can be considered a separate lane of the road section.

Triangulation

In order to triangulate the road, an additional helper array was introduced. It is filled with indices of the vertices during the geometry arrays population. For each cross-section line, three indices are stored, the first one pointing to the outer vertex of the left side of the road, one index pointing to the vertex of the center of the road and the last one, pointing to the outer vertex of the right side of the road. This makes it easier to move through all of the vertices and create groups of triangles for each lane for both sides of the road.

Per each side of the road, there might be three cases that define how the side will be triangulated.

1. First sample line has less lanes then the second one.

2. Both samples have the same number of lanes.

3. Second sample has less lanes then the first sample.

These cases are shown in Figure 3.14.

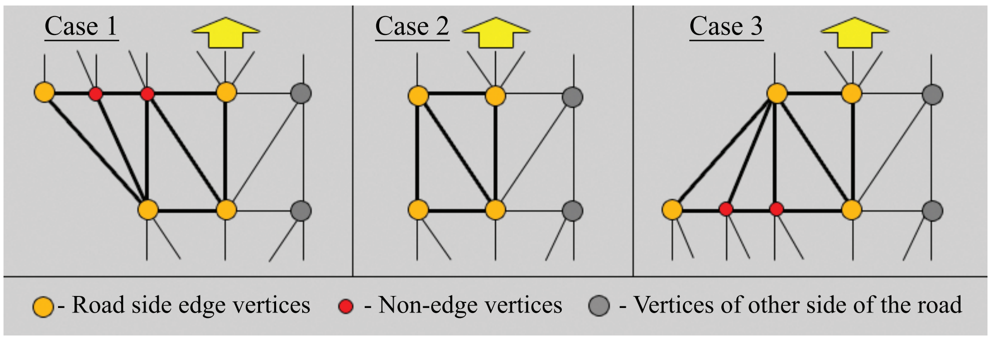

Case 1
Case 2
Case 3
- Road side edge vertices
- Non-edge vertices
- Vertices of other side of the road

Figure 3.14: Road triangulation cases

Triangulation is done in two steps, first the left side, then the right side. Using the indices array, populated before, the difference in the number of lanes between the first and the second sample is found. The triangulation of each side is then divided into three steps:

1. Compute the indices of the current side of the road for the first and second cross-section.

2. Triangulate the part of the road that has the equal number of lanes by creating pairs of triangles.

3. Create the *connection triangles* that create a connection to cover the difference in lane number between the two cross-sections.

The triangulation algorithm is straightforward by itself. The main problem was to correctly compute the indices during the vertex array population.

Junctions

In order to visualize the junctions in the most efficient way, we tried two methods:

1. Double road method

2. Triangulation method

As it was stated previously in the OpenDRIVE section, a junction consists of a number of roads or paths that connect the junction's incoming roads. The first technique is based on the triangulation of the area occupied by the junction's roads. The steps of this technique are as follows:

1. Go through all the roads that belong to a junction and add the vertices computed on from the first sample (at the $s = 0$) to an array.

2. Use the Delaunay triangulation methods provided by OpenSceneGraph to compute the triangles that will cover the junction area.

3. Compute the texture coordinates and apply a texture.

Yet this method, in its raw form, creates a visible flaw, as it does not permit for precise junction geometry (Left of Figure 3.15). Besides that, it does not permit for wheel tracks texture or road marks to be displayed. It also has a performance impact, as it requires the use of a triangulator.

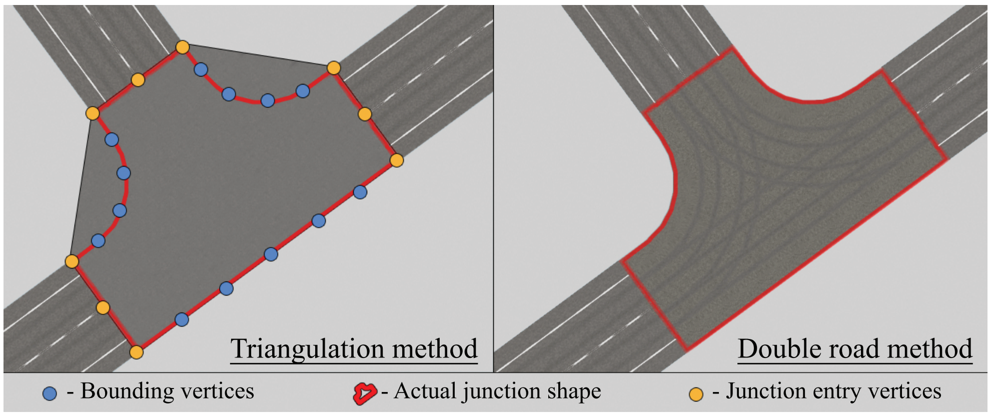


Figure 3.15: Two junction methods

To fix some of the problems the switch from Delaunay triangulation, to Constrained Delaunay triangulation [17] is required. Additional computations have to be done in order to find all the bounding vertices of the junction besides the previously computed junction entry vertices (Figure 3.15). This, in its own case, is a problem: the number of roads of the junction is the double of the junction's connecting paths (there is a separate junction road for each side - left and right, of the connecting path). This makes it difficult to find which road (or connecting path side) is the junction's outer / bounding road and vertices from which side of that bounding road have to be taken into consideration. Even if this is feasible, in relation to computational costs, there is another problem that has to be dealt with - having a multitude of vertex groups for each such bounding road; there is a requirement to create a line-loop consisting from the named vertex groups that should not self-intersect (as a condition of the Constrained Delaunay triangulation).

As a result, the method became too expensive to use and work on it further, from both development and computational points of view. Though, its research resulted in a few ideas that helped to develop the second, *double road method.*

The double road technique is based on the fact that OpenDRIVE specification uses simple roads to describe the connecting paths of a junction. This fact can be exploited to use the road generation methods to draw the junction's roads. It also increases the consistency of the road geometry group; as it treats with no difference both the original roads and junction paths. There is still a need for a minor modification that fixes two problems that arose during the test-runs of the drawing methods:

1. There is a need to remove the face-flickering caused by a mixture of events:
    - Lots of faces having the same elevation and overlapping as a result of the natural overlaps of the connecting paths.
    - Insufficient depth precision, resulting in the depth-test, failing to produce constant depth value for the same faces, which is the main cause of the flickering.
2. Because of the face overlaps, there is a need to use a different texture for the driving area, as the default one has wheel tracks. This means that by default, a blank asphalt texture is applied to the roads of the junction.

These modifications permit to generate an exact junction, as shown in the right image of Figure 3.15. The method was called *Double Road method* as it adds another roads layer on top of the road layer described previously, but in this case, uses just the wheel track texture with alpha channel to blend in the wheel tracks into the asphalt texture, thus creating a seamless transition between the incoming roads and the junction (Figure 3.15). Even though the example from the figure does not have any road-marks for the junction, these are fully supported the same way as for the simple roads.

3.5.2 Helpers

Using the *Road tree* alone to select and define various parameters of the road becomes more difficult in an exponential manner with the growth of the number of roads and parameters that are added to the system. At a certain moment it might become rather difficult to manage all the available entities as the information provided by the tree itself is insufficient for a complete overview of the situation.

In order to improve the usability of the application, we decided to develop a number of helper objects that will be displayed in the 3D View and thus, make the user aware of the current situation.

Road chord line helper

One type of helpers that we made available at the earliest phase of development was a dotted line of different colors that represented the various types of chord line geometry. It was first developed as a debug feature, but apparently, it still has its use. Three colors represent three types of supported geometry: *red* for spirals, *green* for arcs and *blue* for lines. This type of helper might become useful at early stages of the road creation process as it helps to distinguish the beginning and the end of each chord line segment and also its type.

Road records helpers

The first type of helper objects we decided to add is a group of glyphs that will represent each parameter along the road. This option does not only show where, along the road, a specific parameter is situated, but also reflects the currently selected record. Having a helper object for each road parameter record is a good addition for the basic tree structure, but it makes much more sense if the user is able to select the helper object and thus select the record, represented by the that helper.

The statements above resulted in a series of problems that had to be solved:

1. Having a large number of helper types (15 for the moment), how to position and organize them in a user friendly fashion, making them easy to be seen and selected?

2. What shape should the helper have in order to make it easy for the user to distinguish the type of the record represented by the helper?

3. How to distinguish the helpers that belong to a specific lane?

A number of methods were researched to answer the questions. We chose a method that permits for the complete list of helpers to be displayed at the same time without interfering with each other. Below is a list of needed actions, implied by the selected method that would answer the first question:

a Rotate the helper according to the heading of the road at property's s-offset.

b Position each helper according to the property's s-offset along the chord line.

c Translate the helper along the lateral t vector of the track coordinate system to a given lateral offset, depending on the record type. The record types are sorted in the same order as in the road tree and the creation region in order to make it more intuitive for the user. This means that if viewed from top, the helpers have the same ordering as in the road tree.

d In case it is a lane parameter (a lane is a group-record on its own, so its child-record need a special treatment), translate the helper vertically along the z vector of the track coordinate system to an offset that depends on the lane id in a "left lanes at bottom" fashion.

We tested various shapes and colors for the helper glyphs, but in order to keep it as simple as possible yet intuitive and at the same time to avoid inconsistency, we decided to add a billboard to the glyph. The billboard, depending on the record type, would have the same image as the record has in the road tree and creation section. This makes it much easier for the user to mentally associate a helper with a specific record type. Besides that, a billboard will always point to the camera, which makes it easier to notice independently on the current point of view.

This brought another problem at first, as a rotating icon would make it difficult to understand the direction to which the record applies: Figure 3.16.

To solve this, we added another shape to the glyph - a 3D arrow that makes the direction noticeable from any angle. The final example of the glyph is shown in Figure 3.17.

It was stated previously that the helper is translated vertically according to the lane id. That statement partially answers the third question regarding the helpers belonging to specific lanes. Having a vertical translation might help you find out which lane the record belongs to, but another small modification increases the understanding much more: apply different colors to the helpers of different lanes. The lanes with a positive id, being left lanes, get a blue color. Lanes with negative id, being right lanes, get red color. The central lane has a

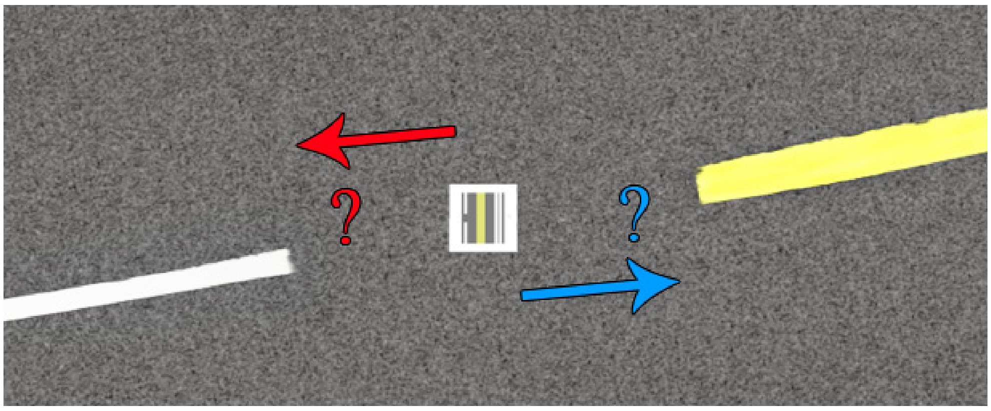


Figure 3.16: Record direction problem

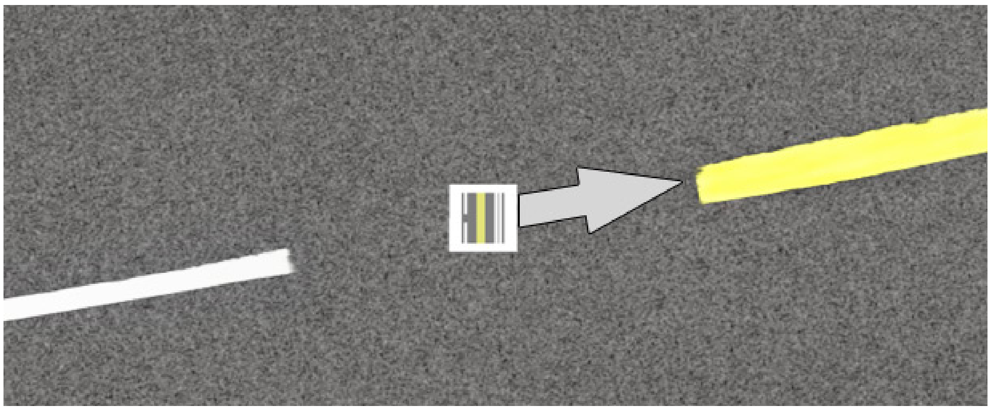


Figure 3.17: Record direction - Solved!

standard grey color. As can be seen from Figure 3.18, this modification makes the helpers much more distinguishable on the lane level.

Figure 3.18 shows that the chosen method permits for a large number of helpers to be effectively displayed. It also provides the necessary comfort when operating with lots of road parameter records simultaneously.

The mentioned glyphs fully support picking which means that a click on either the icon or the arrow selects the corresponding record in the road tree and displays its properties in the properties panel. The glyph is also partially colored to show that it has been selected.

Road helpers and selection

The specification of OpenDRIVE permits the users to connect the roads to each other in various manners:

- Start to end
- Start to start
- End to start
- End to end

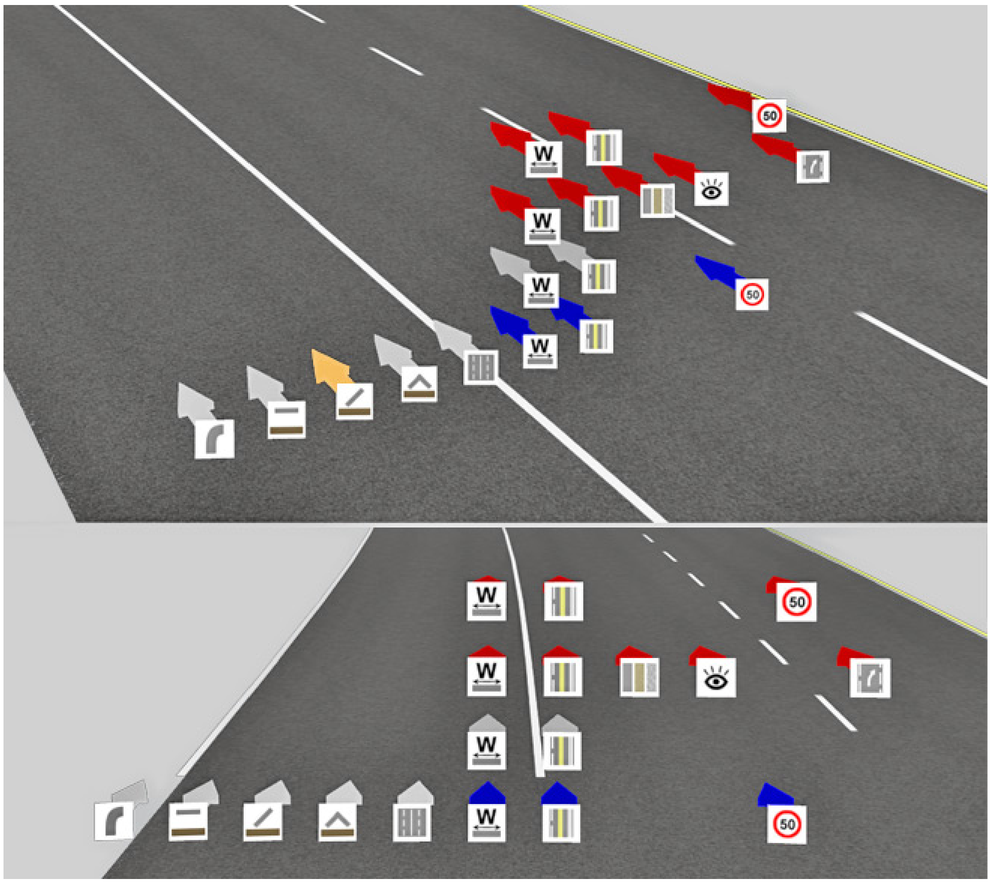


Figure 3.18: Record helpers example

This brings in a problem when it is not always clear what is the direction of the road and at what end the next geometry will be created. The simplest, yet a very convenient method is to create an arrow helper at the end of each road segment, pointing to the heading of the road at that position. In case a OpenDRIVE geometry record is added, the arrow's position and direction is recalculated to conform to a new road end. An example displaying a helper arrow is presented in Figure 3.19.

Among the features of the 3D Viewport are the possibilities not only to display the results of the road generation, but also to allow the user to select and manipulate the objects in 3D space. As it was stated previously, helper objects should be made available for selection. The road object should also be selectable, even if it does not have any helpers assigned to it.

In order to provide the selection functionality, we created a custom geometry node class that inherits both a geometry node and also a special "Information" class called *OSGObjectNode*. *OSGObjectNode* stores the type of the object and also an array of indices to the corresponding arrays in the OpenDRIVE structure. When the desired object is picked via

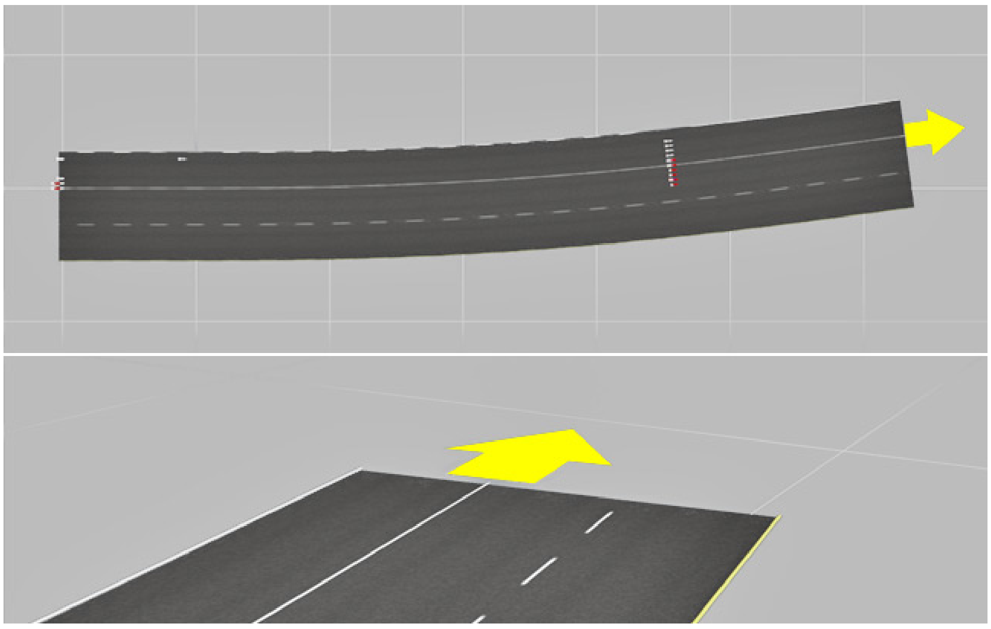


Figure 3.19: Arrow and Grid helpers

the OpenSceneGraph *LineSegmentIntersector* methods, all the additional information of the geometry node stored in *OSGObjectNode*, is sent to the user interface modules and the corresponding object is selected, displaying its parameters, etc. All the objects that can be selected from the 3D-viewport are objects of this custom geometry node class.

Reference plane and grid

In order to increase the user's positional awareness in the 3D View, some reference is required. For our application we developed two types of references: a *Grid* and a *Reference plane*. A grid on the ground plane, on a negative z-axis level is enabled by default (Figure 3.19). Below is a list of advantages of grid as a reference helper:

- It is visible 90% of the time.
- Its orthogonal lines provide a good orientation reference.
- It does not cover any important objects.
- It has a customizable position, size and resolution.

The second available awareness helper option is a reference plane consisting of a quad with a custom texture applied to its faces. In the context of our application this is an important option (which has been requested by one of the cooperating companies). It permits the user to load a reference map or a satellite image, define the size/scale and the position of the plane and then build the road on top of it.

3.5.3 Scenery

One of the initial ideas of the project was to generate some scenery along the road in addition to the road itself. This would permit for the use of the generated geometry not just to display the real-time progress of the road creation / editing process, but also to export and use the resulted geometry in simulators together with the logical description, or even for some different purposes. The later implies a wider application of the program as it would allow the users to generate road and landscape geometry that could be used in any 3D engine including game-engines. This last additional feature does not require much effort to be implemented as OpenSceneGraph supports enough export formats to fulfill any such needs.

The work on road scenery in the context of our application consists of a few steps:

1. Landscape generation:

   a Generate a 3d landscape from a given heightmap

   b Blend the road into the generated landscape

2. Landscape population:

   a Add vegetation

   b Construct buildings and various structures

   c Spread other objects through the landscape

   d Add a skybox and atmospherics support

The current thesis focuses just on the first step as it provides the base for everything else. The points listed under the second step can be easily implemented using the data resulted from the landscape generation step, but are not included in the scope of the current thesis project.

Heightmap based landscape generation

In order to create a basic landscape for the road scenery, a heightmap is used. A heightmap is a greyscale image that stores the elevation value as the intensity of each of its pixels. One channel of a standard 8-bit image can contain a maximum of 256 values/levels so the height value is limited. In order to avoid the issue, the pixel's intensity is normalized and multiplied by a custom height-factor to result in the required height value.

The process of the heightmap based landscape generation consists of two main steps:

1. **Depending on the resolution of the landscape, create a grid of vertices.**
   There is no dependency or restrictions related to the resolution of the grid or the resolution of the image, as the height values obtained from the heightmap are linearly interpolated in accordance to the landscape resolution. This means that a 64 by 64 pixel heightmap can be used on a 100 by 100 segment landscape grid and vice versa: a large image (2048x2048 px) can be used on a 10 by 10 segment grid. This step generates a field of points, positioned at the intersections of the grid cells in the ground plane and translated vertically according to the height values derived from the heightmap.

2. **Triangulate the vertices to generate the geometry.**
   The triangulation step uses the vertices obtained from step 1. The most efficient way to store connectivity information is to use triangle strips, as this not only allows for a decrease of the memory, used to store the triangle information, but as a result, reduces the time needed to send the data to the hardware [7]. Besides that, triangle strips are optimized on most graphics cards. In order to effectively build the triangles, a snake-like motion is used as the vertices are triangulated column by column (Figure 3.20). When a column changes, an additional vertex index is added to the triangle-strip (blue circles and x2 symbol in Figure 3.20) in order to create a deprecated triangle (with two vertices at the exactly same position, thus making it is not visible) and start the new column with a "fresh" triangle. The result of this step is a landscape mesh ready to be rendered.

3. **Assign texture coordinates in order to apply diffuse textures to the landscape.**
   Having an organized grid, it is not difficult to calculate texture coordinates as a relation of the vertex position to the landscape grid size.

We created a special tool that presents a range of options related to the creation of the landscape (Figure 3.21). Below is a list of parameters used in the landscape generation:

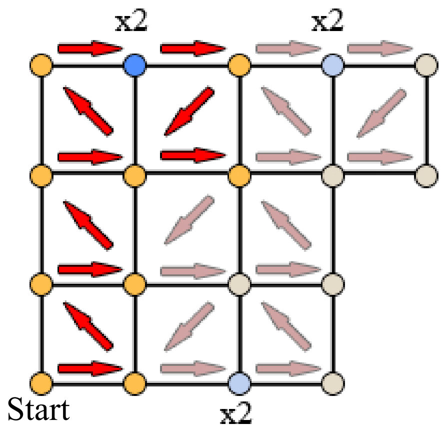
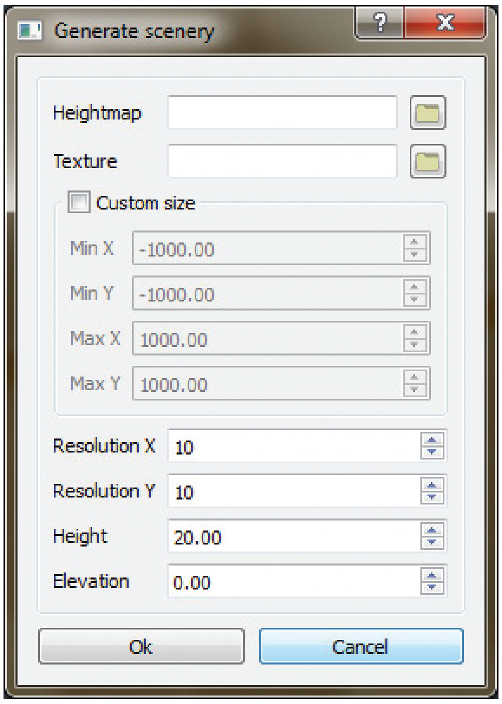


Figure 3.20: Triangulation of the landscape



Heightmap

Texture

Custom size

Min X -1000.00

Min Y -1000.00

Max X 1000.00

Max Y 1000.00

Resolution X 10

Resolution Y 10

Height 20.00

Elevation 0.00

Figure 3.21: Landscape generation dialog

- The user has the possibility to define the position and size of the landscape manually or let the application set these parameters automatically, based on the size of the bounding box that contains all the roads present in the scene.

- Users can also define the resolution of the grid that is being displaced by the heightmap.

- Depending on the heightmap contrast, its intensity values need to be scaled by a height-factor, as stated previously. This is done via the *Height* parameter. Besides that, the user might want to use the *Elevation* parameter to shift the landscape vertically, relative to the ground-level.
- Besides the heightmap itself, a texture map (a satellite photo, for example) might be assigned to the landscape. In order to subjectively increase the detail at close-ups, while providing a low resolution texture map, an additional noise level is added through the use of multitexturing. An example of the noise use is displayed in Figure 3.22.

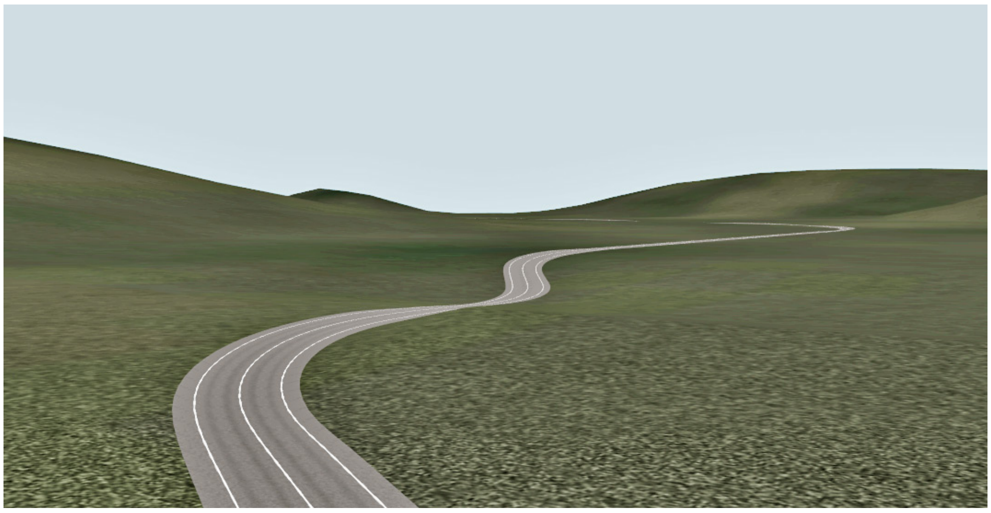


Figure 3.22: Basic scenery

Road and landscape blending

It is not that much of a problem just to generate a landscape from a heightmap, the problem instead is to make the landscape seamlessly blend with the road surface geometry. Basically there might be three different types of blending between the two:

1. Modify the scenery so it follows the road.
2. Modify the road according to the scenery elevation.
3. Apply modifications to the scenery and the road in order to achieve an average transformation of both geometries.

For our system we decided to stick with the first option. As the project under development is a road editor in the first place and the scenery is just an addition, the landscape should not affect the road parameters in any way.

From the other point of view, the second and the third option should not be completely excluded from the research. In that case, the scenery would not be just a decoration for the generated road, but instead be a part of the road generation process, as it would permit the use of the satellite elevation maps to define the elevation of the road. Though, this is a topic for another thesis project.

The first method to blend the landscape and the road, keeping the road elevation and other parameters intact, is to create additional faces on both sides of the road oriented downwards to a certain degree relative to the road surface. This will permit to hide small irregularities of the landscape geometry that come in contact with the roadbed. The method is very efficient as it requires almost no additional computation and adds just 4 additional triangles per each sampling-step related segment of the road. A texture of gravel or any other kind of road boarder can be used for the blending faces. The method is visually explained in Figure 3.23.

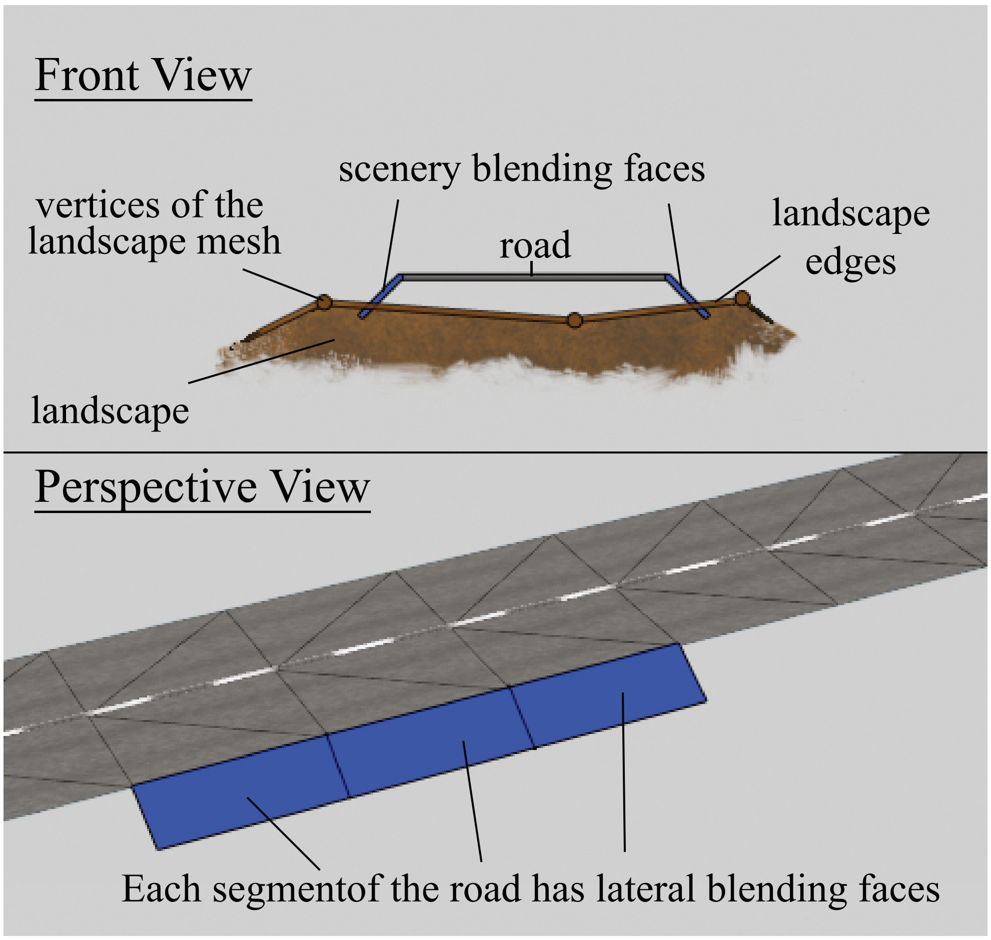

Front View
scenery blending faces
vertices of the landscape mesh
road
landscape edges
landscape
Perspective View
Each segmentof the road has lateral blending faces

Figure 3.23: First scenery blending method

Despite that the method seems computationally inexpensive and extremely easy to implement, it has a serious flaw: the landscape should be as flat as possible along the road, with a specially designed heightmap, to exactly follow the road elevation. And even if these conditions are fulfilled, there might still be enough problems to make the method unusable for certain situations (Figure 3.24). In other words, the method requires plenty of extra manual work and planning in the process of heightmap/scenery design which does not fulfill the goal of the tool as it distributes and even shifts the focus from the road design, to the scenery design.

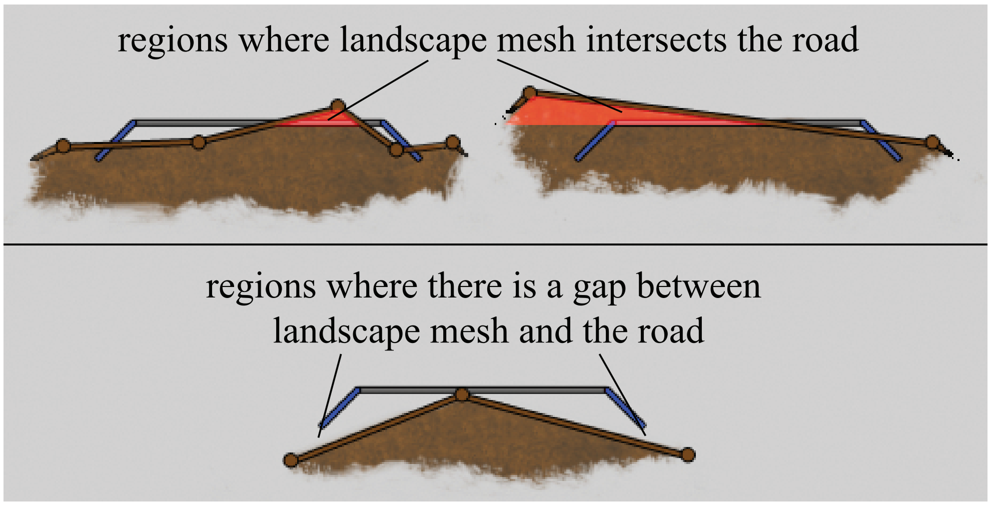

regions where landscape mesh intersects the road
regions where there is a gap between landscape mesh and the road

Figure 3.24: Problems of the first blending method

Due to the issues presented above, we decided to implement a different method. The technique implies a much more precise blending of the road with the scenery but it also requires additional implementation and computation efforts.

If the described technique is used, there is no need to manually triangulate the landscape as described in the second step of the *Heightmap based landscape generation* section and the third, texture-coordinates step also becomes redundant at that phase. Instead, a constrained Delaunay triangulation [17] is used to create the surface of the landscape. OpenSceneGraph has an implementation of the named triangulator which significantly simplified the development process.

The preparation and the post-process procedure consists of a few steps that are divided between the various methods of the road and scenery generation:

1. For each road, on its calculation or re-calculation, fill a corresponding array of the *OSGScenery* class with the bounding vertex-pairs for each

sample along the road. This creates a "mask" for each of the roads, as it stores the bound vertices of the road surface.

2. Create an *osgUtil::DelaunayConstraint* object for each road.

3. Re-arrange the vertices for each of the roads, computed in the first step, to form a line loop and add them to the objects from step 2. This is needed, because the initial structure, described in the road generation chapter, holds the vertices in a different order (displayed in Figure 3.25).

4. Compute the vertex field of the landscape from a given heightmap (step 1 from Heightmap based landscape generation) or use the previously computed vertices.

5. Add the vertices from step 4 and the constraints form step 3 to an *osgUtil::DelaunayTriangulator* object and triangulate.

6. Remove the triangles inside the constraints as these are the triangles that are covered by the roadbed.

7. Compute the vertex normals as the average of the normals of the vertices' surrounding faces.

8. Compute texture-coordinates using a top projection, which is easy, knowing the size of the landscape, its origin and the position of each vertex.

9. Apply a texture and a noise map.

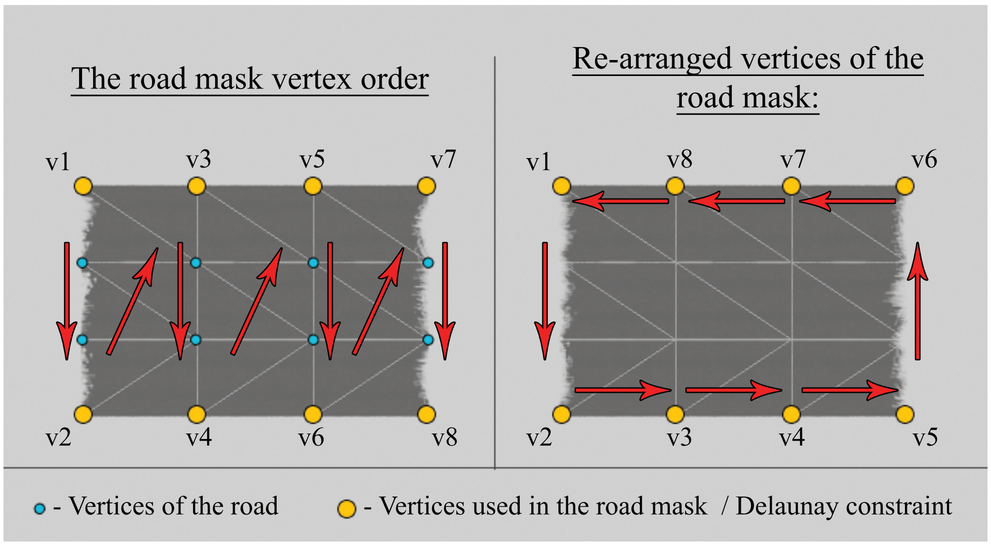

The road mask vertex order
v1
v3
v5
v7
v2
v4
v6
v8
Re-arranged vertices of the road mask:
v1
v8
v7
v6
v2
v3
v4
v5
● - Vertices of the road
● - Vertices used in the road mask / Delaunay constraint

Figure 3.25: A road mask transformation to a Delaunay constraint

As a result, a perfect blending of the road with the scenery is achieved. Some additional work might be done in order to smoothen the geometry of the road sides, but as long as the elevation difference between the road and the scenery is reasonable, there should be no visible issues. Figure 3.26 displays the blending of the scenery with the road that has a changing elevation slope on each subimage.

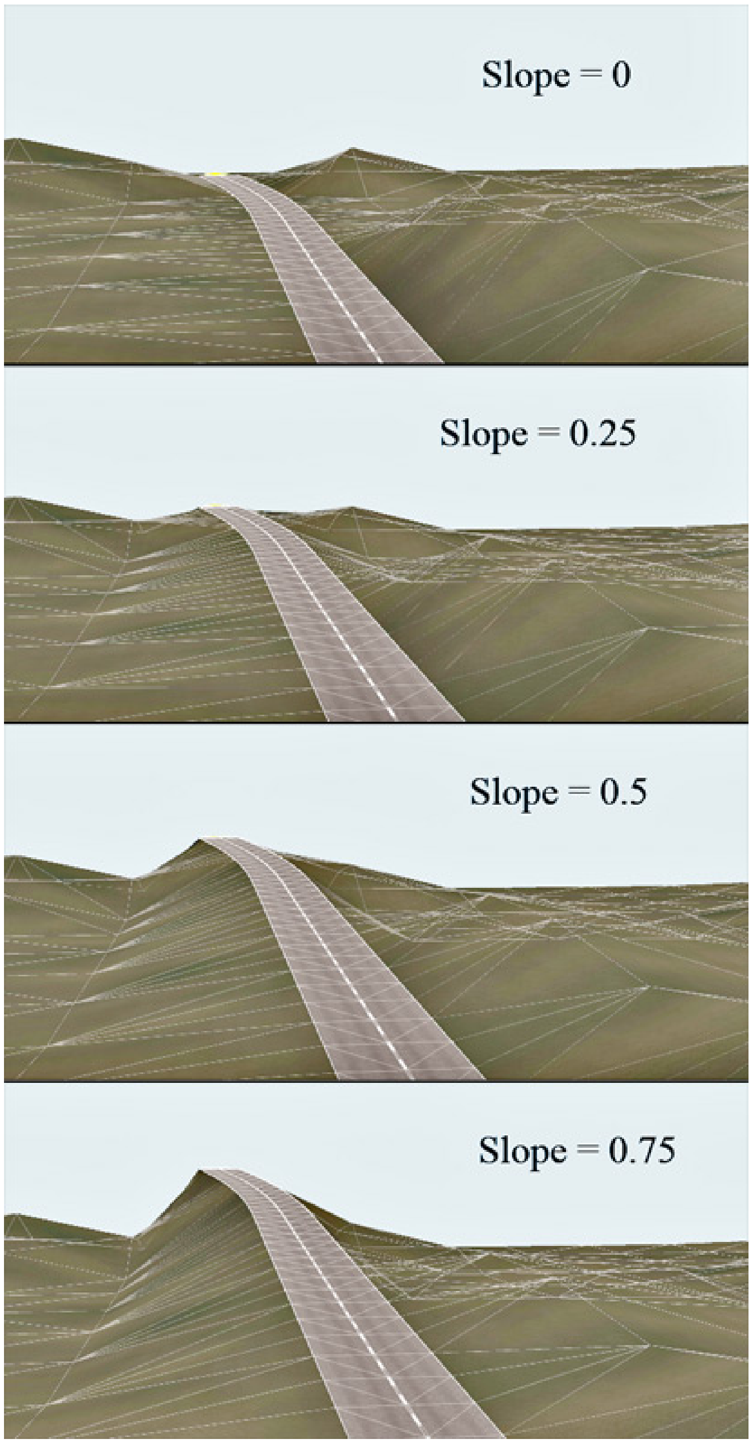


Figure 3.26: Example of the second blending method

Chapter 4

Thesis work results and discussion

In this paper we focused on the problem of logical road data generation tools, used to create content for driving simulators. With the introduction of standardized formats for logical road description, simulator companies had to rely on the tools, provided by the developers of the format. The goal for this thesis was the development of an accessible open source solution, which could compete with the commercial applications on the market.

After choosing and studying the standard, we managed to produce an application that is capable of generating and exporting logical road data in OpenDRIVE format, as well as 3D geometry of the resulting road. Both sets of data could be used in driving simulators, supporting OpenDRIVE format and Open Scene Graph rendering system.

Even though, due to time constraints, not all of the functions and records were implemented, the result has all the basic functionality, required for normal operation. Except the “Signal” and “Object” records, the current version supports every other type of element, used for road description in OpenDRIVE. As additional functionality the application has methods to connect roads and generate terrain.

4.1 Evaluation

To identify the problems with the application, two evaluations were performed.

4.1.1 Heuristic evaluation

First evaluation was based on heuristic evaluation [21]. We explained the user interface in detail, as well as the available actions and some use scenarios. The evaluators provided feedback on the overall state of the application, as well as the potential problems in user interface and available functions.

During this session, we gathered a lot of useful information and suggestions. Some of the points mentioned during the evaluation include the addition of:

- Descriptive labels and help on the record purpose into the empty space under the property group in the settings panel
- Import/Export function for separate road records
- A Splash screen listing the open-source libraries used in development
- Mouse cursor position in the local X Y coordinates and as an S offset from the road origin
- Road linking function, which will generate a new road segment between two roads that need to be connected
- Road geometry adjustment for the height-map based scenery
- Custom road marks which could be set up with signals or objects

4.1.2 User testing

The second evaluation was a remote user testing. This evaluation implied the use of the application in live conditions and a later report of the problems found during the test.

The test users had to fill in a form, answering the questions and giving details on problems and issues they encountered. The sample form is given in Appendix B.

As a conclusion from this evaluation we can mention that a lot of potential issues determined at the heuristic evaluation showed themselves during testing. Testers asked for:

- A way to import/export single road records
- A way to automatically link roads by generating a new segment between the two roads
- An explanation for the records and their properties

- Additional instructions and guides, hints and tips, an example project
- Changeable measurement units for record properties
- A way to focus the camera on the selected item
- Easier ways to define junctions

4.2 Future Work

After analyzing the results of the evaluation and the suggestions made, we compiled a list of features which would either improve the usability of the application or would add new key functionality.

Some of the requests, such as the splash screen, additional help and guides, record description and selection focusing will be implemented before the release, while the following items are more complicated and are left for future development iterations:

- Add the ability to connect two roads with a new road section. This would allow for quick assembling of complex road networks, making it easier to interconnect junctions and separated roads with a road section, generated on-the-fly.
- Simplify and automate the process of junction definition, by providing the ability to use and create junction templates.
- Add support for the OpenDRIVE elements that are missing from current release.
- Add the possibility to export and import the road sections or separate roads.

In relation to geometry generation, there are several things that could be improved:

- Road and landscape blending should be improved in order to smooth the vertex peaks that might appear along the edges of road (as a result of a large difference in height of the road and the landscape vertex defined by the heightmap).
- Add a different road-landscape blending mode that will fit the road to a defined landscape.

- As a separate step, create a road border with a different sample step size (larger) in order to decrease the number of edges that emerge from the road after blending with the landscape.
- Check the road type record and assign the corresponding texture to the road.
- Add scenery population with trees and buildings.
- Add various on-screen gadgets that will help users orient in 3D space. An example of such a helper is a gizmo that will always (independently on the camera position and orientation) point to the X, Y, Z directions.
- Improve the junction geometry generation solution. For the moment, there are face overlaps, and even if it causes no visible issues, there is a little waste of face count.

Bibliography

[1] Apache subversion. http://subversion.apache.org/.

[2] Cephes mathematical library. http://www.netlib.org/cephes/.

[3] Git - fast version control system. http://git-scm.com/.

[4] Gtk+. http://www.gtk.org/.

[5] Merriam-webster. http://www.merriam-webster.com/dictionary/simulator.

[6] Microsoft visual studio express. http://www.microsoft.com/express/windows/.

[7] Msdn - triangle strips. http://msdn.microsoft.com/en-us/library/bb206274(VS.85).aspx.

[8] Openscenegraph. http://www.openscenegraph.org/projects/osg.

[9] Qt. http://qt.nokia.com/products.

[10] Side effects houdini. www.sidefx.com/.

[11] Tinyxml. http://www.grinninglizard.com/tinyxml/.

[12] Vejgeometri. http://www.matematiksider.dk/vejgeometri.html.

[13] Volvo technology. http://www.volvogroup.com/group/global/en-gb/volvo%20group/our%20companies/volvotechnologycorporation/Pages/volvo_technology.aspx.

[14] World wide web consortium (w3c). http://www.w3.org/TR/REC-xml/.

[15] zlib license. http://www.gzip.org/zlib/zlib_license.html.

[16] Raymond Clare Archibald. Euler integrals and euler's spiral–sometimes called fresnel integrals and the clothoide or cornu's spiral. *American Mathematical Monthly*, 25, 1918.

[17] L. P. Chew. Constrained delaunay triangulations. Waterloo, Ontario, Canada, 1987. ACM New York, NY, USA. 0-89791-231-4.

[18] Mete Ciragan. Milskhape 3d. `chumbalum.swissquake.ch/ms3d/`.

[19] VIRES Simulationstechnologie GmbH. Opendrive, 2008-2010. `www.opendrive.org`.

[20] VIRES Simulationstechnologie GmbH. Opendrive format specification, 2008-2010. `http://www.opendrive.org/docs/OpenDRIVEFormatSpecRev1.2A.pdf`.

[21] Soren Lauesen. *User Interface Design: A Software Engineering Perspective.* Addison-Wesley, 2004. Chpater: Heuristic Evaluation.

[22] Raph Levien. The euler spiral: a mathematical history, 2008.

[23] Walter Maner. Rapid application development. `http://www.cs.bgsu.edu/maner/domains/RAD.htm`.

[24] OKTAL. Roadxml. `http://www.road-xml.org/`.

[25] Arthur Newell Talbot. The railway transition spiral. 1899. A revision and extension of the article in Technograph No. 13 reprinted in handbook form.

[26] Jenifer Tidwell. *Designing Interfaces.* O'Reilly, November 2005.

[27] Bo Tonnquist. *Project Management.* Bonnier Utbildning, 2008.

[28] VTI. Vti. `http://www.vti.se/`.

[29] David A. Wheeler. Make your open source software gpl-compatible. or else. `http://www.dwheeler.com/essays/gpl-compatible.html`.

Appendix A

Application output samples

A.1 OpenDRIVE file listing

```
<?xml version="1.0" ?>
<OpenDRIVE>
    <header revMajor="1" revMinor="1" name="Testfile" version="1" date="Thu Feb 8
         14:24:06 2007" north="2.0000000000000000e+003"
         south="-2.0000000000000000e+003" east="2.0000000000000000e+003"
         west="-2.0000000000000000e+003" />
    <road name="" length="1.9000000000000000e+002" id="" junction="-1">
        <link />
        <planView>
            <geometry s="0.0000000000000000e+000" x="0.0000000000000000e+000"
                 y="0.0000000000000000e+000" hdg="2.0000000000000000e+000"
                 length="5.0000000000000000e+001">
                <line />
            </geometry>
            <geometry s="5.0000000000000000e+001" x="-2.0807341827357121e+001"
                 y="4.5464871341284088e+001" hdg="2.0000000000000000e+000"
                 length="2.0000000000000000e+001">
                <spiral curvStart="0.0000000000000000e+000"
                     curvEnd="1.0000000000000002e-002" />
            </geometry>
            <geometry s="7.0000000000000000e+001" x="-2.9727724897415456e+001"
                 y="6.3355409224493101e+001" hdg="2.1000000000000001e+000"
                 length="3.0000000000000000e+001">
                <arc curvature="1.0000000000000002e-002" />
            </geometry>
            <geometry s="1.0000000000000000e+002" x="-4.8502343507187732e+001"
                 y="8.6610170318631901e+001" hdg="2.4000000000000004e+000"
                 length="2.0000000000000000e+001">
                <spiral curvStart="1.0000000000000002e-002"
                     curvEnd="0.0000000000000000e+000" />
            </geometry>
            <geometry s="1.2000000000000000e+002" x="-6.4110503828771186e+001"
                 y="9.9101363666061445e+001" hdg="2.5000000000000004e+000"
                 length="2.0000000000000000e+001">
                <spiral curvStart="0.0000000000000000e+000"
                     curvEnd="-1.6000000000000007e-002" />
            </geometry>
            <geometry s="1.4000000000000000e+002" x="-7.9455202240690255e+001"
                 y="1.1189319291971061e+002" hdg="2.3400000000000003e+000"
                 length="3.0000000000000000e+001">
                <arc curvature="-1.6000000000000007e-002" />
            </geometry>
            <geometry s="1.7000000000000000e+002" x="-9.4455607866302017e+001"
                 y="1.3754158462082819e+002" hdg="1.8600000000000003e+000"
                 length="2.0000000000000000e+001">
                <spiral curvStart="-1.6000000000000007e-002"
                     curvEnd="0.0000000000000000e+000" />
            </geometry>
```

```
</planView>
<elevationProfile>
    <elevation s="0.0000000000000000e+000" a="0.0000000000000000e+000"
         b="0.0000000000000000e+000" c="0.0000000000000000e+000"
         d="0.0000000000000000e+000" />
    <elevation s="5.0000000000000000e+001" a="0.0000000000000000e+000"
         b="4.8000000000000036e-002" c="0.0000000000000000e+000"
         d="0.0000000000000000e+000" />
    <elevation s="1.5000000000000000e+002" a="5.0000000000000000e+000"
         b="0.0000000000000000e+000" c="0.0000000000000000e+000"
         d="0.0000000000000000e+000" />
</elevationProfile>
<lateralProfile />
<lanes>
    <laneSection s="0.0000000000000000e+000">
        <left>
            <lane id="1" type="driving" level="false">
                <link />
                <width sOffset="0.0000000000000000e+000"
                     a="3.0000000000000000e+000"
                     b="0.0000000000000000e+000"
                     c="0.0000000000000000e+000"
                     d="0.0000000000000000e+000" />
                <roadMark sOffset="0.0000000000000000e+000" type="solid"
                     weight="standard" color="yellow"
                     width="0.0000000000000000e+000" laneChange="both" />
            </lane>
        </left>
        <center>
            <lane id="0" type="driving" level="false">
                <link />
                <width sOffset="0.0000000000000000e+000"
                     a="0.0000000000000000e+000"
                     b="0.0000000000000000e+000"
                     c="0.0000000000000000e+000"
                     d="0.0000000000000000e+000" />
                <roadMark sOffset="0.0000000000000000e+000" type="broken"
                     weight="standard" color="standard"
                     width="0.0000000000000000e+000" laneChange="both" />
            </lane>
        </center>
        <right>
            <lane id="-1" type="driving" level="false">
                <link />
                <width sOffset="0.0000000000000000e+000"
                     a="3.0000000000000000e+000"
                     b="0.0000000000000000e+000"
                     c="0.0000000000000000e+000"
                     d="0.0000000000000000e+000" />
                <roadMark sOffset="0.0000000000000000e+000" type="solid"
                     weight="standard" color="yellow"
                     width="0.0000000000000000e+000" laneChange="both" />
            </lane>
        </right>
    </laneSection>
    <laneSection s="9.0000000000000000e+001">
        <left>
            <lane id="1" type="driving" level="false">
                <link />
                <width sOffset="0.0000000000000000e+000"
                     a="3.0000000000000000e+000"
                     b="0.0000000000000000e+000"
                     c="0.0000000000000000e+000"
                     d="0.0000000000000000e+000" />
                <roadMark sOffset="0.0000000000000000e+000" type="solid"
                     weight="standard" color="yellow"
                     width="0.0000000000000000e+000" laneChange="both" />
            </lane>
        </left>
        <center>
            <lane id="0" type="driving" level="false">
                <link />
```

```
                        <width sOffset="0.0000000000000000e+000"
                              a="0.0000000000000000e+000"
                              b="0.0000000000000000e+000"
                              c="0.0000000000000000e+000"
                              d="0.0000000000000000e+000" />
                        <roadMark sOffset="0.0000000000000000e+000" type="broken"
                              weight="standard" color="standard"
                              width="0.0000000000000000e+000" laneChange="both" />
                    </lane>
                </center>
                <right>
                    <lane id="-1" type="driving" level="false">
                        <link />
                        <width sOffset="0.0000000000000000e+000"
                              a="3.0000000000000000e+000"
                              b="0.0000000000000000e+000"
                              c="0.0000000000000000e+000"
                              d="0.0000000000000000e+000" />
                        <roadMark sOffset="0.0000000000000000e+000" type="broken"
                              weight="standard" color="standard"
                              width="0.0000000000000000e+000" laneChange="both" />
                    </lane>
                    <lane id="-2" type="driving" level="false">
                        <link />
                        <width sOffset="0.0000000000000000e+000"
                              a="0.0000000000000000e+000"
                              b="2.2800000000000004e-001"
                              c="-1.3000000000000005e-002"
                              d="1.0000000000000000e-003" />
                        <width sOffset="1.0000000000000000e+001"
                              a="2.2999999999999998e+000"
                              b="0.0000000000000000e+000"
                              c="0.0000000000000000e+000"
                              d="0.0000000000000000e+000" />
                        <roadMark sOffset="0.0000000000000000e+000" type="solid"
                              weight="standard" color="yellow"
                              width="0.0000000000000000e+000" laneChange="both" />
                    </lane>
                </right>
            </laneSection>
        </lanes>
        <objects />
        <signals />
    </road>
</OpenDRIVE>
```

A.2 Generated geometry screenshots

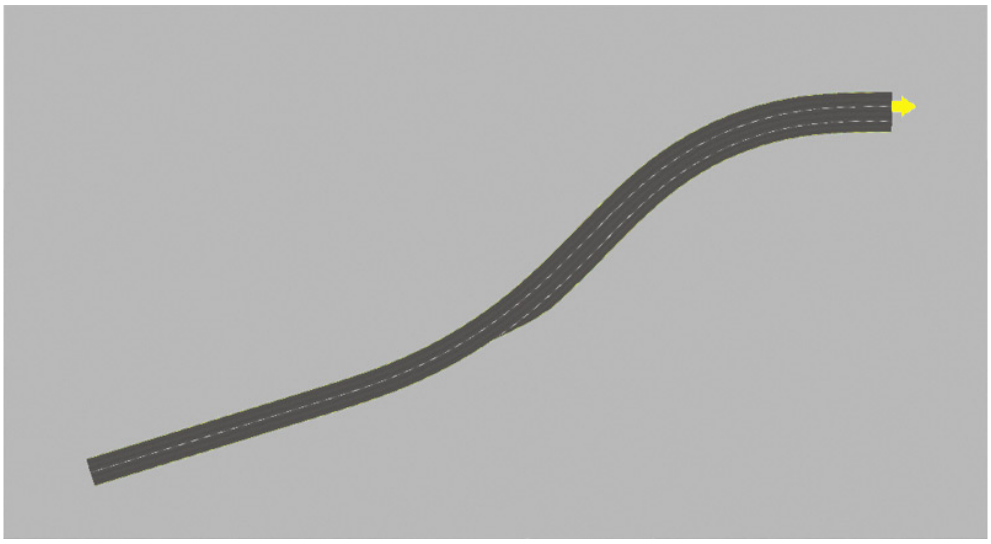


Figure A.1: Screenshot 1 of the sample road

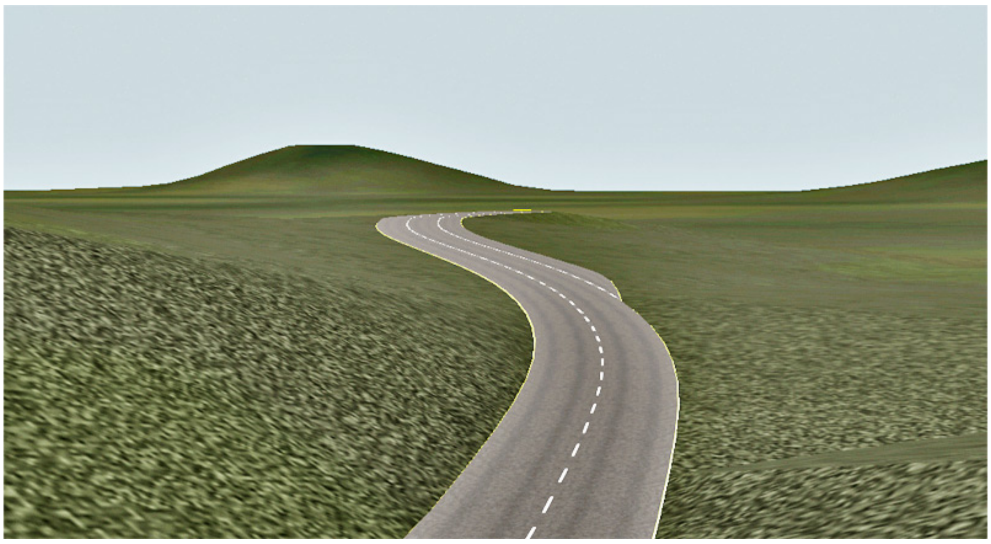


Figure A.2: Screenshot 2 of the sample road with a generated landscape

Appendix B

Road Editor Feedback Form

Please describe your experience with the application by filling this form:

1. **How easy was it to get started?**
   1 (Hard) - 5 (Easy)
2. **Is the application easy to navigate?**
   1 (Hard) - 5 (Easy)
3. **Are icons that are used for record types clear and intelligible?**
   1 (Not clear) - 5 (Clear)
4. **Rate the overall user interface?**
   1 (Poor) - 5 (Fine)
5. **Relative number of task you were unable to accomplish?**
   1 (Low) - 5 (High)
6. **Did you find yourself in a situation where you didn't know what to do next?**
   Box
7. **Did you find yourself in a situation where you didn't know how to do a specific task?**
   Box
8. **Rate the performance of the application:**
   1 (Low) - 5 (High)
9. **Rate the overall quality of the application:**
   1 (Low) - 5 (High)

10. **Rate the overall quality of the application output:**
    1 (Low) - 5 (High)

11. **Name the good and the bad points about the application:**
    Box

12. **Suggestions:**
    Box