ViP City

Authors

Carl Johan Andhill, Dynagraph
David Orebäck, Dynagraph
Pontus Holmertz, HiQ
Richard Leandertz, Scania
Matteo Manelli, Scania
Fredrik Persson, Volvo Cars
Anders Ödbloom, Volvo Cars
Dennis Saluääär, AB Volvo (GTT)
ViPCity

Authors
Carl Johan Andhill, Dynagraph
David Orebäck, Dynagraph
Pontus Holmertz, HiQ
Richard Leandertz, Scania
Matteo Manelli, Scania
Fredrik Persson, Volvo Cars
Anders Ödbloom, Volvo Cars
Dennis Saluääär, AB Volvo (GTT)

www.vipsimulation.se
Preface

The ViPCity project was a collaborative activity between Dynagraph, HiQ, Volvo Cars, Scania, AB Volvo (GTT) and the Swedish National Road and Transport Research Institute (VTI). The work was carried out within ViP Driving Simulation Centre (www.vipsimulation.se), a Swedish competence centre, which is financed by Vinnova (the Swedish Governmental Agency for Innovation Systems; grant number 2011-03994) and the centre partners.

ViPCity builds upon the previous ViP projects DeDT, DeDT2 and Greit, and was co-financed by Test Site Sweden.

The purpose of the ViPCity project does not include any planned dissemination outside the ViP community, neither of this report nor the technical (software) outcome. The project results and a demonstration have been presented at a ViP workshop, and the technical assets are accessible at ViPForge. The source files can be found on ViPForge/ViPCity. For information contact Carl Johan Andhill, Dynagraph (can@dynagraph.se), Pontus Holmertz, HiQ (pontus.holmertz@hiq.se) or the ViPForge administrator Jonas Andersson Hultgren, VTI (jonas.andersson.hultgren@vti.se).

Gothenburg, May 2015

Carl Johan Andhill
Project manager
Quality review

Peer review was performed to 28 August 2015 by Emil Knabe, Volvo Cars and to 31 August 2015 by Jonas Andersson Hultgren VTI. The first author Carl Johan Andhill has made alterations to the final manuscript of the report. The ViP Director Lena Nilsson examined and approved the report for publication on 1 July 2016.
## Table of contents

Executive summary ............................................................................................................. 9

1. Problem description ..................................................................................................... 11

2. Goal ................................................................................................................................. 12

3. Method ............................................................................................................................ 13

4. Project activities ............................................................................................................. 14
  4.1. Gathering of requirements ....................................................................................... 14
  4.2. DeDT and DeDT2 evaluation .................................................................................. 16
  4.3. State of the art .......................................................................................................... 17
    4.3.1. Software ........................................................................................................... 17
    4.3.2. International organisations .............................................................................. 20
    4.3.3. Conclusion ........................................................................................................ 22
    4.3.4. Inspiration ........................................................................................................ 22
  4.4. Developing ViPEdit ................................................................................................... 22
    4.4.1. Design/editing tool ......................................................................................... 22
    4.4.2. Metafile format ............................................................................................... 23
    4.4.3. OpenDRIVE export ....................................................................................... 24
    4.4.4. Usability ........................................................................................................... 24
    4.4.5. Curves ............................................................................................................. 25
    4.4.6. Representing the environment ........................................................................ 26
    4.4.7. Junctions .......................................................................................................... 26
    4.4.8. Save format ...................................................................................................... 27
  4.5. Development of 3D world generation (Greit) ......................................................... 28
  4.6. Integration, testing and verification ......................................................................... 29

5. Comments from the manufacturers ............................................................................... 31
  5.1. Volvo Cars ............................................................................................................... 31
  5.2. AB Volvo (GTT) ..................................................................................................... 31
  5.3. Scania ........................................................................................................................ 32

6. Results ............................................................................................................................ 34

7. Conclusions .................................................................................................................... 36

8. Future activities .............................................................................................................. 37

References ........................................................................................................................ 39
## Abbreviations and definitions

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arc</td>
<td>A part of a circle with constant radius.</td>
</tr>
<tr>
<td>Bezier curve</td>
<td>A mathematically defined curve used in two-dimensional graphic applications.</td>
</tr>
<tr>
<td>Clothoid</td>
<td>Curve whose curvature changes linearly.</td>
</tr>
<tr>
<td>DeDT/DeDT2</td>
<td>Previous ViP projects which led to an editor for creating road logics.</td>
</tr>
<tr>
<td>Greit</td>
<td>Visualization component based on the game engine Unity3D.</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical user interface.</td>
</tr>
<tr>
<td>HMI</td>
<td>Human Machine Interaction.</td>
</tr>
<tr>
<td>Metamodel</td>
<td>File format containing the environment description.</td>
</tr>
<tr>
<td>NVDB</td>
<td>Nationell vägdatabas (National Road Database).</td>
</tr>
<tr>
<td>OpenDRIVE</td>
<td>Open format specification to describe the logic of a road network.</td>
</tr>
<tr>
<td>OpenSceneGraph</td>
<td>Open source 3D graphics application programming interface.</td>
</tr>
<tr>
<td>Polygon</td>
<td>A two-dimensional shape formed with straight lines.</td>
</tr>
<tr>
<td>Qt</td>
<td>Cross-platform application framework for developing application software.</td>
</tr>
<tr>
<td>UDP</td>
<td>User Datagram Protocol.</td>
</tr>
<tr>
<td>ViPCore</td>
<td>The core part of the ViP simulation platform.</td>
</tr>
<tr>
<td>ViPEdit</td>
<td>Editor to create road logics and environment descriptions.</td>
</tr>
<tr>
<td>VISIR</td>
<td>Visualization component used in the ViP platform.</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language (XML) is a markup language to encode documents in a format which is both human-readable and machine-readable.</td>
</tr>
<tr>
<td>xodr</td>
<td>File extension used for the OpenDRIVE format.</td>
</tr>
<tr>
<td>2D</td>
<td>Two-dimensional space.</td>
</tr>
<tr>
<td>3D</td>
<td>Three-dimensional space.</td>
</tr>
</tbody>
</table>
List of figures

Figure 1. Schematic view of how ViPCity (in orange) integrates with the ViP platform....................... 13
Figure 2. Screenshot of VIRES. ........................................................................................................ 19
Figure 3. Screenshot of PreScan......................................................................................................... 20
Figure 4. Scenarios where arcs will (green) and will not (red) work to connect line segments......... 26
Figure 5. The strength of Bezier segment connection......................................................................... 26
Figure 6. Two four-way junctions before (top) and after (bottom) connection................................. 27
Figure 7. Four inter-connected junctions capsuling an area................................................................. 27
Figure 8. View from the generated city............................................................................................... 34
Figure 9. View from the generated city............................................................................................... 35
Figure 10. View from the generated city........................................................................................... 35
List of tables

Table 1. Example of ace file layout................................................................................................. 24
Table 2. Example of save file layout............................................................................................... 28
Executive summary

Today simulation studies in ViP are mainly carried out in countryside driving environments. There is a lack of city environments. This is probably due to the fact that creating and running countryside environments in some aspects are easier than creating and running city environments. Another reason might be that countryside driving is very relevant in Swedish studies.

As projects, and markets, become more international the need for city simulator studies becomes more important. Many drivers around the world do most of their driving in cities.

In the ViPCity project software has been developed which facilitates the generation of driving environments for city simulations on the ViP platform. The project result is a number of assets (software, file formats and 3D components) which integrates well with the ViP platform. These assets together give simulator users the possibility to design city environments in a fast and easy way. The software has been implemented and tested successfully in Scania’s truck simulator.
1. Problem description

Most ViP simulation studies today are carried out in countryside driving environments. There is a lack of, and need for, city driving environments to cover the full spectra of driving situations. Vehicles are produced for an international market and the Swedish countryside is not the typical driving environment for most drivers.

Simulated city driving environments are harder to produce than countryside environments. The city environments need more details and more inhabitants to become realistic. The logical description of the road network, necessary to the simulation software, is much more complex. It is very challenging, and time consuming, to produce both the 3D environment and the logical road network. Today this has to be done by skilled programmers.
2. **Goal**

The goal of the ViPCity project was to make it easier to create simulated city driving environments. To reach the goal, a software that helps the user to build cities, including the 3D scene and the logical road network, should be developed. An important requirement was that the software should be user-friendly and accessible to non-programmers.
3. Method

The ViPCity project will make city driving environments available to the ViP platform. It will be done by creating a design tool where non-programmers, with little training, can draw their own city in a drag-and-drop fashion. The output from the design tool will be both the 3D environment and the logical road network. These two “assets” can then be used with the current ViP platform.

ViPCity will be integrated with the ViP platform like this:

- **ViPCity**
  ViPEdit is developed where the user can design a city. The result is a logical description of the road network (xodr) and a metamodel. The metamodel, which is a 2D map, is imported by a second software, Greit (Andhill, Blissing and Källgren, 2014), where the 3D world is generated by combining a set of already made 3D components. The 3D world and the logical description is then used by the ViP platform.

- **ViP platform**
  The ViPCore is the “brain” of the simulator. It reads the logical road description (xodr), the scenario and the input from the driver's cabin. ViPCore continuously calculates new positions for all actors and sends them to the 3D world. The 3D world updates all actors, and displays them on the simulator screen.

Figure 1 below shows the different components of ViPCity and how they will integrate with the ViP platform. Objects in orange (to the left of ViPCore) belong to ViPCity.

![Figure 1. Schematic view of how ViPCity (in orange) integrates with the ViP platform.](image-url)
4. **Project activities**

**Work packages**

The ViPCity project was divided into the seven work packages below.

**WP1 – Gathering of requirements**

In this work package internal and external stakeholders were interviewed to find out their requirements.

**WP2 - DeDT and DeDT2 evaluation**

Since the ViPCity project is a continuation of the previous ViP projects DeDT (Alm, Hagemann and Andhill, 2012) and DeDT2 (Stenmarck, Leandertz and Blissing, in preparation), the software developed in these two projects was evaluated. DeDT is a design tool for creating countryside road crossings. What works well, what does not work well and what parts can be re-used were analysed.

**WP3 – State of the art**

The purpose of this work package was to find out how similar challenges have been approached by others. The work included contacting other simulator sites/researchers around the world.

**WP4 – Development of the design tool**

In this work package the design tool, ViPEdit, was developed.

**WP5 – Development of 3D world generation software (Greit)**

In this work package the functionality to turn the 2D city map into a 3D world was developed.

**WP6 – Integration and test**

In this work package the software developed in ViPCity was implemented and tested in Scania’s driving simulator in Södertälje.

**WP7 - Demonstration and report**

The developed software was demonstrated at a ViP workshop and the report was written.

**Meetings**

Throughout the project we have held about 20 meetings with internal and external parties. At the meetings we have discussed requirements, problems and solutions. The meetings have also been a good opportunity to keep everyone involved and informed. Furthermore, it has given us the possibility to have an agile work process.

**4.1. Gathering of requirements**

This project started with gathering requirements from internal and external parties – Volvo Cars, AB Volvo (GTT), Scania, Swedish Road Marking Association (SVMF) and VTI. We sent out a set of questions and also held a meeting.
The questions and the most notable answers are listed below.

**How can your organization use and benefit from city environments?**

We can use it for:

- Product development, HMI design and research.
- Testing of road markings and how they affect drivers' behaviour.
- Studying complex driving environments.
- Demonstrating the ViP platform capabilities.

**What shall be in the city? Both in terms of static objects and actors.**

- Streets of different kinds – main roads, small streets. Crossings with right of way, crossings with priority, pedestrian crossings, traffic lights and crossings with bike lanes.
- City buildings, pedestrians, bicyclists, parks, sidewalks and other props to make the city lively.

**What kind of tests and studies do you want to do?**

- One example was studying and comparing how drivers behave both when road markings exist and when they do not exist.

**Which is your target group?**

- A wide range of drivers in terms of age, gender and professional/non-professional. Also drivers with disabilities like limited vision as well as drug and alcohol affected drivers.

**Which type of simulators will be used?**

- The simulator at Scania is a fixed-based truck cabin with three projectors covering ~180 degrees of the view from the driver’s seat. The cabin is also equipped with two monitors as rear-view mirrors. Most of the software is from ViPForge\(^1\). Some is developed at Scania.
- High fidelity simulators as well as low fidelity (desktop) simulators.

**Which requirements do you have for the design tool?**

- Today the roads and environments are made by the design tool DeDT (developed in ViP) and ‘manual’ coding. To create a junction and making it work in the simulator is hard work. So in order to design a city with lots of junctions some automation would be desirable. This could be done by using template junctions with the possibility to change number of lanes etc.
- The tool needs to be easy and fast to use. The developer should not need to bother about too many details or the work will be too extensive. One example could be blocks of roads and buildings, kind of like LEGO, or template roads and template city blocks. Another example could be to auto-generate roads from Google maps or other open source maps.
- Functions to generate random city environments would speed up the city development a lot.
- To be able to add static and dynamic actors to the scene, and to add autonomous traffic via the design tool. As a ready-made scenario or as a complement to the Kernel scenario.
- To be able to choose brightness of sunlight, fog, night and day etc. Lights, shadows and overall more visual interaction between actors and the environment.

---

\(^1\) [https://www.vipforge.se](https://www.vipforge.se)
• Exporting functions of xml documents with roads, actors, objects and scenarios.
• To be flexible, the tool should not be locked to xodr, rather having the functionality to generate xodr when ready to build.
• It would be great if you easily could design 50 meters of business area, 50 meters of cafés and restaurants, and then a traffic light with emergency vehicles.
• Drag and drop design.
• Automatically generate/randomize a city.
• We would like to have exact knowledge about all objects and their positions in the generated world.

Which graphical level is necessary?
• The closer we get to real life without compromising functionality and compatibility is nice. We do not have a specific demand at this point.
• Not to detailed graphics. It must be possible to add a lot of actors.

What driving environments do you have today? Do you have any city environments?
• Today the roads are created using DeDT and “manual” work such as hard coding junctions and making calculations for, for example, elevation curvature. The possibility to create a city, or even a network of roads, is very limited and would take too much time and effort to be considerable.

Do you see any risks in developing a city design tool?
• Junctions might not be fully functional in VISIR\(^2\), this needs to be tested. No easy desktop testing environment for ViP simulations.

Other?
• Can the city be generated without Unity\(^3\)?

Since ViPCity was quite well specified in the project application and its approval, not all these requirements could be met. But it is very useful to know the requirements even if everything could not be implemented.

4.2. DeDT and DeDT2 evaluation
DeDT (Alm et al., 2012) was a project within ViP which investigated and implemented functionality to generate complex driving environments by adding things such as pedestrian crossings with animated agents and road junctions. The functionality was based on the software OpenRoadEd, which is the result of a master thesis project by two students at Chalmers (Kurteanu and Kurteanu, 2010). The overall goal of DeDT was to enhance the realism of simulator environments, and make it easier and more cost effective to create advanced driving environments. The project was carried out between January and June 2011. It was led by HiQ and the following partners participated; HiQ, VTI and Dynagraph. The project was carried out in co-operation with Test Site Sweden who was also co-funding it. As a proof of concept one of Test Site Sweden’s facilities was created as a simulator environment. In the ViP project DeDT2 (Stenmarck et al., in preparation) more functionality was

---

\(^2\) Graphical engine in ViP simulation platform.
\(^3\) Commercial graphical engine http://unity3d.com/
added to the DeDT tool and by that extending it from a junction/segment editor to a road network editor.

DeDT is using Qt\(^4\) for window handling and internal software architecture. For the graphical view it uses OpenSceneGraph\(^5\). The internal structure of the road network data is based on the structure used by OpenDRIVE\(^6\), which is an XML-based storage format used for storing logical road networks based on parametric data. OpenDRIVE defines the road networks by connecting road segments where each segment is defined by a reference line. The reference line consists of one or more parts of varying geometric form such as lines, arcs and clothoids.

When evaluating DeDT2, the ViP partners were asked if they use the tool and what they like and dislike with it. Today only Scania is using DeDT2 for designing new simulation environments. According to Scania the output from DeDT2 cannot be used directly without refining the junction to work properly within their simulator. All partners said that they believe DeDT2 to be too hard and time consuming to use, and that is why they have not integrated it as part of their tool chains.

Within the bounds of ViPCity the choice was to either build upon DeDT2 or to create a new editor from scratch. The decision made was to build a new editor. During the DeDT development the partners involved experienced some problems with using OpenDRIVE as the internal format. Based on that experience, the ViPCity partners decided that it would be better to create a new editor as the intention was to create a flexible internal format which is built upon how the user wants to interact with the objects rather than how they a best represented mathematically. This also gives the benefit of not being bound to the OpenDRIVE format if it is later decided that another format should be used. With this in mind it was decided to create a new tool based on what was learnt from the development of DeDT2. The fundamental idea for the new tool was, per request of ViP partners, a tool that prioritizes user-friendliness over customizability. The tool was created with how the user wants to work in focus, and making the internal representation resemble that.

When developing the new tool, we re-used the parts of the DeDT2 source code that handle windowing, setting up OpenSceneGraph rendering area in a Qt, and setting up a Qt application.

4.3. State of the art

In this work package we evaluated other software used for building cities and driving environments. We also contacted some other organizations which are facing similar challenges as the ViP community.

4.3.1. Software

We tried to look at all software we could find that is used for generation of roads or urban environments. We also reached out to the participating partners to get ideas of software to look into.

Ghost town

When looking at Ghost Town\(^7\) we used the free lite version for the evaluation. It is a plugin for the 3D-modeling software 3Ds Max from Autodesk. To efficiently use Ghost Town prior experience of either 3Ds Max or other 3D-modeling software is necessary. Ghost town is a great tool for fast creation of big areas of a city, but to get a great result both time and experience are needed.

\(^4\) http://www.qt.io/
\(^5\) http://trac.openscenegraph.org/projects/osg/
\(^6\) http://www.opendrive.org/
\(^7\) http://www.scriptspot.com/3ds-max/scripts/ghosttown-lite
Ghost Town city generator is a tool for creating urban environments mostly for architectural purposes. It is able to create low-poly and high-poly buildings, keeping parts of the buildings separate if a higher or lower poly-count is desired.

Ghost Town can also randomize trees, benches, streetlight into the scene for a more advanced environment.

When Ghost Town generates urban environments it operates in the following way:

1. Blocks of houses are created based on a polygon area.
2. Each block is considered a “group”.
3. Each “block” is then divided into different sub-groups:
   3.1 ” Level 0” – street level.
   3.2 Facades (4 sides).
   3.3 Roof.
4. All these polygons-areas can be made into a more detailed area with easy clicks.

It is also possible to just texture the polygon-faces if the high-poly is unwanted. There are some textures provided but textures are easily added.

Problems in the software:

- Not all types of houses are easy to create “high-poly” parts to.
- Some street-props are placed incorrectly – with the “back” facing out.
- It has a default way in which it” divides” the areas.
- The default way can be influenced by the user dividing the area before generating.
- It is not possible to specify where a specific area should be, it is always done randomly.

CityEngine

CityEngine is a tool for urban planning. Fairly basic for adding volumes – and if there is information from cities (that has been done previously) it seems to be able to fetch and build the area – the ArcGIS holds this information. Other positive things are the dynamic of the interface, i.e. lots of “drag able” panels to change thickness, heights, volumes etc.

We experienced problems with the 30-days trial, so we were not able to test the trial.

City Gen

The City Gen software supports some other 3D programs “to some degree”. This software is mainly for inspiration purposes. The program had no realistic road networks and it did not focus on what we need. Inspiring because of its easy “mouse-drag interface” although it seems that it has not been worked on for many years.

Greeble plugin for Max

Greeble plugin for Max is a free modifier for 3Ds Max to create fast “building shapes” that could for instance resemble a city. This is a plugin that has been used in “older sci-fi” movies like Star Wars etc. It is a quick and easy way to create the “Basic” shape of buildings and to generate a reasonable height difference.

---

9 http://www.citygen.net/
10 Quick video – https://www.youtube.com/watch?v=pHJ1-64VA6U
OpenDRIVE - Road Designer

RoadDesigner is a design tool that comes with the visualization engine VIRES\(^\text{11}\). In RoadDesigner you can build new or modify existing OpenDRIVE files. Since VIRES has based its software completely on OpenDRIVE, and also is one of the companies behind the development of OpenDRIVE, their Road Designer is tightly tied to the format and its structure. It was clear that the program is aimed for only handling small and limited road networks since the build area is very limited. It was also difficult to modify the road as any change was likely to break the road network in multiple places, which in turn could be hard to fix without breaking the road again. This makes the RoadDesigner more fitting for creating small test environments for specific tests rather than a tool intended for creation of generic simulator road environments. A VIRES screenshot is shown in Figure 2. The figure shows a test track opened in RoadDesigner with a couple of different speed limited road segments.

![Figure 2. Screenshot of VIRES.](image)

PreScan

PreScan\(^\text{12}\) is a complete simulator solution targeted towards active safety functionality testing. PreScan uses a number of internal formats to describe scenarios, logical roads and graphics. The scenes or test environments are built in PreScan’s own design tool called "PreScan Experiment Editor". It uses a type of finished building blocks where the settings are trying to copy the ones from the block it gets attached to. This means that it is quick and relatively easy to build roads from A to B going forward. However, the approach gets a lot less user-friendly when it comes to building road networks since it is very hard to make changes without breaking the network in multiple places.

\(^{11}\) www.vires.com/
\(^{12}\) https://www.tassinternational.com/prescan
Since PreScan is a complete suit directed towards active safety it is no surprise that the design tools are focused on creating environments for such testing. Just like VIRES the product feels directed towards creating small specific test environments rather than the design of general purpose simulator environments. PreScan is shown in Figure 3. In the illustration the user is building a test environment junction for testing an active safety function.

![Figure 3. Screenshot of PreScan.](image)

**VTI software suit**

VTI has a number of different tools used for creating OpenDRIVE environments, mostly for creating road networks from real world data. These tools mostly use different kinds of measured data to generate OpenDRIVE files. In the ViP project Known Roads (Näbo et al, 2015; Hjort and Källgren, 2015) VTI has also investigated and implemented ways to use existing databases (e.g. NVDB and Lantmäteriet) to generate OpenDRIVE files. Which tools that are used are chosen depending on what kind of raw data that is available for the given road network.

**E on – Vue**

Vue is a tool for creating massive landscapes. It is used in movies and other CGI projects. Knowledge is needed to use Vue and it is an expansive software. We looked at this mainly for inspiration.

**4.3.2. International organisations**

To get a better idea of how the creation and use of urban environments look in the world, we sent out a questionnaire that we asked some simulator sites to answer. The questionnaire consisted of four questions which they answered in their own words. The results are presented here in question and answer form per part.

**University of Wisconsin, US**

- What kind of experience do you have of creating/running city simulations?

  When I was at University of Iowa/NADS we developed a city component to a relatively long, naturalistic drive. It was quite an effortful, long, and expensive process. The end result was quite
impressive, but the process was not something that would fit into most simulator use situations. A faster and easier development process is very much needed.

- What are you using your simulations for? E.g. studies, product development?

My use of driving simulators is focused in two areas: driver distraction and vehicle automation and support systems.

- How do you create your simulation environments and road descriptions today?

Using the tile-based system provided by the simulator vendor (RTI).

- If you are using a 3rd party tool, do you think it is good or is there any changes you would like to do?

The software has substantial flexibility but is a challenge to use. A more seamless and simple process for developing generic worlds is needed.

University of Iowa, US

- What kind of experience do you have of creating/running city simulations?

We have created city simulations that range in size from a few hundred square feet to areas that are 6 square miles. We have experience in creating cities that are based in the United States as well as Europe. More recently, we have added the capability to have dozens of pedestrians (as well as vehicles) in our driving simulations. Our cities have realistic road markings, signs, traffic signals. It takes a big effort to create a realistic city—for example, it can take anywhere between 1-4 person months of effort to create a 1 square mile city area. We are currently exploring ways to make this process more efficient.

- What are you using your simulations for? E.g. studies, product development?

Our simulations are primarily used for human factors research studies and product testing. Product development has not seen prominent use on our simulations. One recent exception is a project that will produce data for the sponsor that plans to use it to improve collision avoidance systems.

- How do you create your simulation environments and road descriptions today?

The simulation environments and road descriptions are primarily created by importing publicly available road/scene data into a tool like Presagis Creator. This 3rd party allows us to create correlated visual and logical road definitions. We currently use our own road definition language called LRI (Logical Road Information). In the future, we may move towards a more universally used road definition language.

- If you are using a 3rd party tool, do you think it is good or is there any changes you would like to do?

In general, we prefer using tools that we have developed internally as it gives us greater flexibility. However, we are open to using 3rd party tools when those tools are able to provide functionality that would be too expensive to develop internally. For example, we use a 3rd party tool for road creation (Presagis Creator). Recently we also started using a 3rd party tool for pedestrians (DI-Guy). In both cases, these 3rd party tools have been effective for us despite the license fees.

DLR, Germany

DLR did not answer the questions directly but instead we had a phone conference where we discussed the questions above. Here we list our impressions from this discussion.
What kind of experience do you have of creating/running city simulations?

At DLR they have been working on projects aimed at creating vehicle simulation in urban environments for the last 5-6 years. Their projects are mostly focused on creating roads and environments based on real world data. They are currently working with a project which aims to create a digital representation of a German city.

What are you using your simulations for? E.g. studies, product development?

The projects currently in progress are mostly aimed at creating test environments for autonomous vehicle development.

How do you create your simulation environments and road descriptions today?

DLR is currently working with developing a tool-chain for creating urban driving environments based on existing database data. They use many different sources depending on what they are building. For the surrounding environments, such as buildings, they use OpenStreetMap while for roads they are mostly using data from municipal sources.

If you are using a 3rd party tool, do you think it is good or is there any changes you would like to do?

At DLR they use commercial products for some parts of the driving environment creation tool-chain. The commercial products are then combined with their own proprietary software to create the full suit environment tool-chain.

4.3.3. Conclusion

In general, the software we have looked at, for inspiration, ideas and to get a glimpse of what is already on the market, is expensive and has a learning level before the user can get use out of it.

Many of the solutions looked at involved a “3Ds Max” license or a “stand-alone” program license which make these solutions expensive.

4.3.4. Inspiration

Many of the looked at software gave ideas and inspiration regarding what we actually strive to deliver.

In general, the focus of those software was not to make an interactive environment, but more to create a town for picture purposes or for urban planning.

The complexity is not to create the building areas of a “basic city”, it lies in creating a complex city-road-network. The surrounding is secondary and easier to update in future scenarios and that is why the interface and its updating compatibility is our focus.

This research gave us a ”proof of concept” that automation is a good thing to have in our user interface. It also gave us inspiration of what the interface should be able to handle. This will be explained more in the next section.

4.4. Developing ViPEdit

4.4.1. Design/editing tool

There are many challenges when it comes to developing a graphical editing tool. The tool needs to be both sufficiently powerful and flexible to be useful but also easy and intuitive to use. These things are not necessarily mutually exclusive but can often be difficult to combine, especially since it often requires a lot of experience to determine what is easy and intuitive. Another potential problem area we
see is how a graphical tool can be combined with OpenDRIVE exports in an easy manner. A common practice of many OpenDRIVE road design tools is to allow the user to describe objects with the same parameters as OpenDRIVE uses internally. This leads to that it often takes a lot of work to get road segments to match with each other which can be a very tedious work.

The development of the editing tool was conducted in an agile spirit with sprint demonstration every two weeks. All participating ViP partners were invited to these demonstrations. The development was done in three stages. In the first stage we began by investigating the needs of the various partners in ViP and what requirements they have on the functionality and use of the tool. More about the survey can be read in chapter 4.1 (Gathering of requirements). Based on the input we got from the survey we did the evaluation of DeDT and DeDT2, which can be read about in chapter 4.2 (DeDT and DeDT2 evaluation). Finally, based on the conclusions from the evaluation we started developing the new software, which we call ViPEdit.

During the development, we have successively evaluated and selected functionality that we have implemented in ViPEdit to meet the requirements in ViP. During this process we have tried to continuously get feedback and input from other simulation knowledgeable people. We have also held several meetings and demonstrations where we have tested and evaluated the entire pipeline.

4.4.2. Metafile format

When choosing a new file format there are many things to take into consideration so that the format remains useful. The format must be flexible enough to cope with many different use cases, powerful so it will be useful in the long run, and expandable so it can be adapted to the users’ exact needs. To make sure that the file format is easy to use and integrate it is also important that it is easily read by both computers and humans. One must also consider size and parsing speed of the format so these features will not be too bulky or slow when large areas are described.

When it came to selecting the format for the graphical representation of the world, we started by looking into existing formats. The most promising one we examined was the shape format developed by ESRI\(^{13}\) (ESRI, 1998). Shape is a vector based format used to store geographical data. It describes objects using points, lines and polygons. Shape consists of three required files; one with all the vector objects, one index file used for search, and an attribute file in which attributes for each object are stored. There is also a plethora of optional additional files to store different things in. Since searching for individual items in the file is irrelevant in our implementation and there is a specific list of required attributes for each object, we decided that it would be easier and more convenient to implement a new format. As with shape, we based our format on vectors of polygons, lines and points. Each type has a specific subset of attributes associated with it which describes what the vector is used for. The format is defined in such a way that it can be saved in both plain text and binary data. See Table 1 for the specification of the metafile format. The format we defined is called "abstract city environment" and uses the file ending ".ace".

The .ace format is specified to store a graphical representation of the world. It stores the world using areas, lines and points. The format can be imported by Greit (Andhill et al., 2014) and used for generating the 3D-world. Table 1 below provides a breakdown of how the format is structured.

\(^{13}\) www.esri.com
Table 1. Example of ace file layout.

<table>
<thead>
<tr>
<th>Input type</th>
<th>Input example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectType</td>
<td>Polygon</td>
<td>The input is describing a Polygon entry.</td>
</tr>
<tr>
<td>Property</td>
<td>Verts 4</td>
<td>The property data will contain 4 rows of vertex data.</td>
</tr>
<tr>
<td>PropertyData</td>
<td>10, 0, 10</td>
<td>The X, Y, Z cords for the first Vertex.</td>
</tr>
<tr>
<td>PropertyData</td>
<td>10, 0, 0</td>
<td>The X, Y, Z cords for the second Vertex.</td>
</tr>
<tr>
<td>PropertyData</td>
<td>0, 0, 0</td>
<td>The X, Y, Z cords for the third Vertex.</td>
</tr>
<tr>
<td>PropertyData</td>
<td>0, 0, 0</td>
<td>The X, Y, Z cords for the fourth Vertex.</td>
</tr>
<tr>
<td>ObjectProperty</td>
<td>Type Road</td>
<td>The poly is a Road with asphalt.</td>
</tr>
<tr>
<td>Property</td>
<td>Verts 4</td>
<td>The property data will contain 4 rows of vertex data.</td>
</tr>
<tr>
<td>PropertyData</td>
<td>10, 0, 10</td>
<td>The X, Y, Z cords for the first Vertex.</td>
</tr>
<tr>
<td>PropertyData</td>
<td>10, 0, 0</td>
<td>The X, Y, Z cords for the second Vertex.</td>
</tr>
<tr>
<td>PropertyData</td>
<td>0, 0, 0</td>
<td>The X, Y, Z cords for the third Vertex.</td>
</tr>
<tr>
<td>ObjectProperty</td>
<td>Type CentreLine ShortGap</td>
<td>The line is a road centerline with normal short gaps and lines.</td>
</tr>
<tr>
<td>ObjectProperty</td>
<td>Verts 1</td>
<td>The input is describing a Point entry.</td>
</tr>
<tr>
<td>Property</td>
<td>Verts 1</td>
<td>The property data will contain center position of object.</td>
</tr>
<tr>
<td>PropertyData</td>
<td>0, 0, 0</td>
<td>The X, Y, Z cords for the position.</td>
</tr>
<tr>
<td>ObjectProperty</td>
<td>Quat</td>
<td>The property data will contain orientation of object.</td>
</tr>
<tr>
<td>PropertyData</td>
<td>0, 1, 2, 3</td>
<td>The X, Y, Z, W cords for the quaternion.</td>
</tr>
<tr>
<td>ObjectProperty</td>
<td>Type 3DObject Trafficlight</td>
<td>Point is a 3DObject that should be loaded representing a traffic light.</td>
</tr>
</tbody>
</table>

4.4.3. OpenDRIVE export

OpenDRIVE is a flexible and well-structured format that handles logical descriptions of the road networks. The format is based on XML and describes the road network through various parameter-based parts. It is described in such a way that e.g. a curve is described by; a starting position, a length, and a constant curvature. In order to have flexible junctions OpenDRIVE is designed so that a junction is a container containing references to many segments making up every possible connection in the junction. This makes it both difficult and time consuming to manually build junctions in an OpenDRIVE environment.

4.4.4. Usability

To achieve that the project will lead to something useful for the various parties within ViP it is important that the software and pipeline produced are not only easily integrated and user-friendly but also flexible and powerful enough to solve the problems of the partners. There are several difficulties in creating software that is user-friendly because there are several things which contribute to the usability. Here, we describe some of those things that we think are important for creating useable
software. We believe that in a user-friendly tool the user should be able to switch between creating details and automatically generated content. Another aspect of the usability is how the difference in complexity between performing different tasks vary. Dealing with such aspects there are no simple answers, because in some cases it may be better to have tasks that are not done often become complicated so tasks that are repeated often can be easy.

To create a user-friendly tool, we have both turned to ViP’s partners, to get their input of how they want to use the tool, as well as looked at other tools and talked to the users of these about what they perceive as good and bad. From this feedback an interaction designer was put on the task to design a user-friendly GUI.

The development of ViPEdit was conducted in three stages. In the first stage, we began to examine the needs and desires of functionality we could find in ViP and which were compatible with the project application. In that study, it was considered that the first priority is to create a tool that should be very easy to use. All ViP partners agreed that the tool must be simple, even if it is at the expense of flexibility and advanced functionality. The problem with the tools available in ViP is that they place high demands on the user. Instead, ViP partners desired a tool that is simple enough to be used by holiday workers or the like. Another wish held in high regard was that the tool and the pipeline should provide a way of working where it is relatively fast and easy to create usable environments.

In the second stage, we investigated whether it would be possible to combine these requests with the architecture developed in DeDT or if it would be easier to develop something new. We came to the conclusion that it would be easier if we developed a new tool where the basic architecture better reflected the expressed wishes.

In the third and final stage, we developed ViPEdit. We decided early on to carry out the development in an agile spirit. We tried to develop the tool from a user perspective where the architecture reflects a graphical user approach so that objects are represented in a way that is logical for modifying them. We decided that this was only feasible if we chose to move away from using an OpenDRIVE representation of data internally in the editor, and instead created our own structure with the ability to export as an OpenDRIVE structure. The architecture we chose was to base the program around a prefab architecture. A prefab is a finished component that can be paired with other components to create the road network and environment. The prefab structure we have developed is based on the idea that there are components of two main types, active ones and passive ones. An active component is one that the user has full control over and directly can modify all parts of, while a passive component is modified when the active component is attached to it. In our implementation straight roads and junctions are active components while curves are passive.

4.4.5. Curves

The first choice we faced with using the prefab architecture was how curves should be represented. The two candidates we chose between were constant-radius curves (arc) and Bezier representation. Figure 4 shows examples of cases where it is not possible to connect two lines with an arc (in red), while possible scenarios (connections) are shown in green. As Figure 4 shows, the limitation with arcs is that the lines that are created by the line segments must be crossed on the same side of both segments. Figure 5 shows an example of two ways in which the connection is made with a Bezier segment. The advantage of Bezier is that it can easily and flexibly be used to connect any two segments. But, this flexibility comes at the cost of two important things that arcs give us. The first one is that with Bezier it can be hard to programmatically check for things such as curves that do not meet the requirements of a minimum radius. The other problem we saw is that it is very difficult to convert Bezier to arcs and clothoid segments, which is what OpenDRIVE uses for road representation. Another solution to this, that we looked into, was to make our own extensions for Bezier in OpenDRIVE but this would result in leaving the standard. In the end we decided to use arcs as they
are easier to convert to OpenDRIVE and allow us to better aid the users to construct more realistic road networks.

![Figure 4. Scenarios where arcs will (green) and will not (red) work to connect line segments.](image)

Figure 4. Scenarios where arcs will (green) and will not (red) work to connect line segments.

![Figure 5. The strength of Bezier segment connection.](image)

Figure 5. The strength of Bezier segment connection.

### 4.4.6. Representing the environment

The next issue we faced was how to deal with the creation of the surroundings. With the goal to create a tool that is as easy as possible to use, we began investigating the possibilities to generate the environment. Our first question was whether the generation should occur in the editor or in the visualization engine. We chose to add the generation on the visualization side as it seemed to be the easier option. The next question we had to answer was how we would give the user control over how the environment is generated. We chose to solve this by letting the user draw surfaces that can then be populated with the type of props that the user selects. In our implementation, we opted for letting the user populate with building or park props.

### 4.4.7. Junctions

The last major issue we faced during the development was how to deal with junctions in our architecture. The question then was how we could integrate junctions in our prefab architecture. To continue with our previous mind-set of simplicity, we chose to do different junction types as prefabs and have limited our implementation to one prefab for four-way and one for three-way junctions. These prefabs can then be customized with both pedestrian crossings and traffic lights.

In the editing tool, functionality to construct urban road networks has been developed. For each created road there are a number of attributes that the user can modify. These attributes are; number of lanes, lane width, shoulder width and type of road lines. A road network is created by the user putting out straight roads and junctions and linking them together with curves. Figure 6 shows two four-way junctions on the top half and how they are linked together in the lower part.
In addition to the attributes that can be modified for the road, the user has the possibility to modify if the junction has traffic lights and pedestrian crossings. For the areas that are populated in the visualization, the user can choose what the area should be populated with and if the area should be surrounded by a sidewalk. Figure 7 shows an example of four interconnected junctions enclosing a surface for populating a block.

4.4.8. Save format

In addition to the format for the graphical representation, we also developed a format for saving projects in ViPEdit. The format is loosely specified since it is only intended to be used to store information internally for the editor. The format is structured so that each item saves the information it needs to re-build itself. For couplings it is required that each object has a unique ID which, after all objects have been loaded, can be used to connect the objects together. Table 2 below shows an example of how the format is structured. In the current implementation of the editor the order in which an object’s data is stored is important since we only use the identifier of readability. This is because we opted for that the format should use the same structure in both text and binary implementations.
### Table 2. Example of save file layout.

<table>
<thead>
<tr>
<th>Input type</th>
<th>Input example</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ObjectType</td>
<td>RoadLine</td>
<td>The type identifier of the object, usually the class name.</td>
</tr>
<tr>
<td>ObjectData</td>
<td>ID 13</td>
<td>This is an ID field and it stores this object’s ID as 13.</td>
</tr>
<tr>
<td>ObjectData</td>
<td>Lanes 1</td>
<td>This is a number of lanes field and this object has 1 lane.</td>
</tr>
<tr>
<td>ObjectType</td>
<td>RoadCurve</td>
<td>The type identifier of the object, usually the class name.</td>
</tr>
<tr>
<td>ObjectData</td>
<td>ID 14</td>
<td>This is an ID field and it stores this object’s ID as 14.</td>
</tr>
<tr>
<td>ObjectData</td>
<td>Lanes 1</td>
<td>This is a number of lanes field and this object has 1 lane.</td>
</tr>
</tbody>
</table>

#### 4.5. Development of 3D world generation (Greit)

This chapter describes the 3D world generation where the actual 3D driving environment is created. The world generation is made by a software which is a further development or extension of the Greit software written in the ViP project Greit (Andhill et al., 2014).

Greit had functionality for displaying, but not generating, a 3D world in the ViP simulator. Our work in ViPCity added generation functionality.

Greit was created using the Unity3D\(^{14}\) computer game development tool. The most powerful benefit of Unity 3D is that it makes the developer very productive. Many desired functions for game realism and game functionality are available to the developer as components.

The Greit software has three main responsibilities:

1. It shall generate a 3D world based on the 2D map designed by ViPEdit. This 3D world must be lively and realistic. This happens when Greit is started.
2. It shall listen to the information coming from the ViPCore. This information includes positions of the Ego Vehicle and other actors. This happens continuously during the simulation.
3. It shall display the world to the simulator screen. This also happens continuously during the simulation.

**Generating the 3D world**

The 3D world generation happens when the Greit software is started. The software reads the map and generates 3D.

The map can basically contain three types of geometry:

- Points describing for example the position of a road sign.
- Lines describing for example a road line.
- Polygons describing for example a road surface, a park area or a building area.

Greit parses (reads) the map and creates 3D objects. In case a point describing a sign is found in the map, Greit places a sign at the corresponding position in the 3D world. In case the map contains a road line, Greit draws a road line in the 3D world.

Areas in the map are a little more interesting. Areas are populated by Greit and here there is room for some creativity.

\(^{14}\) http://unity3d.com/
• In park areas Greit will put grass, trees, playground toys and people moving around.
• In pavement areas Greit will put trees, dustbins, bus shelters and people walking. Along the side of the pavement Greit can put parked cars.
• In building areas houses are put along its border.

See examples of these in Figures 8, 9 and 10 in the Results chapter.

The areas are filled with objects in a random manor, although they will look the same each time since we use the same seed. The purpose is to get a lively and realistic city without the designer having to put every little object by itself. A lot of effort was put into making the city come alive. The randomization could be a drawback for studies which need exact control of every little object, but this was a design choice we made based on gathered requirements. The parties wanted it to be easy and quick to make a realistic city.

Most objects placed by Greit, like traffic signs, trees, dustbins, people, cars, buildings and furniture, are pre-created by a 3D artist. They are manually made and just placed by Greit according to the map. The 3D assets library is available to all ViP parties. There are buildings, vehicles, people, proppings, vegetation, road signs etc. In case somebody would like to create a different look and feel of the city, all they have to do is to replace the 3D assets. Put Chinese buildings, signs etc. and the city will look Chinese.

Listen to the ViPCore

Greit will listen to the ViPCore which sends information about the Ego Vehicle and other actors like cars, trucks, busses 200 times per second. The actors repeatedly are placed according to this information. Most of this functionality was re-used from the Greit project.

Display the world to the simulator

Greit is responsible for producing the simulator’s display continuously during the simulation. Most of this functionality was also re-used from the Greit project.

4.6. Integration, testing and verification

To verify that the developed software works as planned, it was decided to install and run it in Scania’s simulator in Södertälje. This simulator is made up by a truck cabin and five displays (three forward for a 180 degrees’ view and two rear mirrors). Each display is connected to a high end PC. The simulator uses the ViPCore simulation software and normally VISIR as visual system.

The test was performed as follows:

1. Greit was installed to replace VISIR as visual system.
2. ViPEdit was used to draw a city of 10x10 blocks. A map was exported for Greit and an xodr road network for the ViPCore.
3. Scania engineers created a simple scenario with traffic and pedestrians.
4. Greit was started on all graphics clients.
5. ViPCore was started with the road network and the scenario.
6. We took a drive in the city, verifying correct functionality.

We were very pleased with the result. In very little time, about 15 minutes, we could draw a small city (10x10 blocks) and then drive in it. The city looked good too in our opinion. Traffic and the random people on the sidewalks added a lot to the realism.

We discovered that the version of ViPCore run at Scania had some problems with xodr crossings. But according to VTI this has been solved in later versions.
Greit is dependent of knowing the absolute heading of actors. This information is normally not sent from ViPCore. It sends only the heading relative to the road. To solve this, we added the absolute heading to the actor UDP package sent from ViPCore.

When developing the software, we often needed to do quick tests in the simulator. To avoid traveling to Scania, we set up a procedure for remote testing. Scania would run a simulation and record the information coming from the ViPCore with the Wireshark\(^\text{15}\) software. This recording could be re-played with the software Playcap\(^\text{16}\) on a desktop computer allowing us to test Greit with “real” traffic.

\(^{15}\) https://www.wireshark.org/
\(^{16}\) http://www.signal11.us/oss/playcap/
5. Comments from the manufacturers

In this chapter the vehicle manufacturers in ViP give their opinions about ViPCity. Volvo Cars had reviewed the code and we also had separate meetings with them to show, test and discuss the software. Scania was an active participant when implementing and testing the developed software in their simulator. AB Volvo (GTT) has based their opinion on the project meetings and the demonstrations.

5.1. Volvo Cars

There are several potential use cases at Volvo Cars for the outcome of the ViPCity project. They relate to different parts of the tool chain at Volvo Cars for development and verification of active safety functions.

The use cases also relate to different sub-deliveries in the ViPCity project which are:

A. City editor.
B. City format.
C. 3D visualization.

Specifically, we believe that the ViPCity project will contribute to the following needs at Volvo Cars:

1. A user-friendly editor for designing road networks and city environments.
2. Scripts for generating roads in the OpenDRIVE format.
3. An open format that allows to represent real world variations which are not covered by the OpenDRIVE format.
4. License free 3D visualization of traffic scenes for early function testing, for sensor and system testing and for real time driving simulators.

From initial tests it is clear that the road and city editor is smooth and easy to use to quickly set up road networks as well as to create park and building areas primarily for visualization. The graphical visualization of the editor works well. The 3D world is generated quickly from the editor design with random population of parks and buildings. There are two logical files that describe the designed city; one for the editor and one for the graphics engine that uses Unity3D.

Further extensions may include:

- Skew and distort intersections.
- Copy and paste of road elements.
- Merging lanes, exits and entrances.
- Add new points to existing park area polygon.
- Mark all objects in the editor.
- Import of OpenDRIVE roads.
- Convert from graphics engine input file (.ace) for Greit to editor format (.vea).
- Transfer (or import) random population to ViPEdit.
- Import Greit scene back into ViPEdit.

5.2. AB Volvo (GTT)

At AB Volvo (GTT-ATR) there is a driving simulator that has been developed over the years to perform research and development within HMI and driver environment. This simulator is implemented in a different way from the ViP/VTI system. E.g. the logical road networks are based on road descriptions using sampled points in space (x,y,z), sequences of such points form a path, the paths are connected to form a road network. The visualization of the road and traffic environment is implemented using OpenSceneGraph.
Traditionally, the creation of road models at Volvo have been performed with dedicated modelling tools, this step is time consuming and costly. Furthermore, the needed competence to perform this activity is not common within the staff, so there is dependence on certain expertise. A more user-friendly tool can be suited to a wider user group with reduced need for tool experts.

It is of interest to test the ViPCity approach in the simulator at Volvo with the main interest focused on an alternative visualization solution compared to the existing one. There is no doubt that the gaming technology is progressing very fast and we should consider taking more advantage of this development.

However, in order to do that, there are a couple of steps needed:

- Availability of road descriptions as needed for the Volvo simulator.
- Create a modified protocol from the Volvo “core” system to Greit visualization engine.
- Set up a test environment and demonstrate concept.

Technically, the ViPCity editor could be designed to export the road descriptions needed in the Volvo simulator. If time and resources allow, this functionality will be realized within the project. The alternative solution is to have a data conversion step from the ViPCity editor export to the road and scenario description in the Volvo simulator. Previous investigation shows that this is realistic with limited effort, at least for simple road networks.

The Volvo system has a master-slave architecture, meaning that the scenario is executed in one master system and the scenario state is then sent and replicated at the slave systems. This is quite similar to the ViP/VTICcore, although the protocols are different. We need to create a modified protocol from the Volvo master system compatible to the Greit visualization engine.

So, ultimately, the objective for Volvo is to get an increased understanding of available solutions for visualizing environment models in a driving simulator.

5.3. Scania

ViPCity is relevant to Scania in many ways. Scania trucks are used in cities as well as on highways and the objectives vary a lot. Since the Scania simulator, at the time, is mainly used for HMI research and tests, notifying the driver of his/her surroundings is one of the focus points. This is something that differs a lot depending if you are driving on a highway or driving in the city. An interest for interacting with pedestrians and bicyclists has been shown from some projects and is one of the things ViPCity will improve, both simplify and make a more credible appearance.

ViPCity will also be a great chance to look into Greit and Unity3D as the demand for more detailed and natural obstacles and scenarios are increasing. And we hope that in the future this will make it easier to make landscapes and other environments. Switching to Unity3D might also give us an opportunity to add more functions and intelligence directly to the environment rather than adding more code to the simulator Kernel.
Further development:

- The possibility to add specific objects to specific locations, and even further ahead add actors with functionality. This would have to respond to a protocol defined in the Kernel.
- ViP City could also become a more general road editing tool by adding some of the old functions from DeDT, such as making longer road segments, and new functions like junctions with a specified angel for road exits and entrances. This could be supplemented by auto-generating landscapes.
- The city environment demands a large number of objects, and if the objects are standard for all ViP simulators there could be a drag and drop function in the ViPCity editor for adding objects.
6. Results

The result of this ViPCity project is a number of assets (software, formats and 3D components) which integrates well with the ViP platform. These assets together give a user the possibility to design his/her own city in a fast and easy way. The result was implemented and tested successfully in Scania’s simulator in Södertälje.

ViPEdit

ViPEdit is the design tool where a non-programmer can design a city. The output is a 2D map and a logical road network.

Greit

Greit is the software which generates the 3D world from the 2D map. It is also active during the entire simulation providing the 3D graphics for the simulator screen.

Metamodel (2D map)

The metamodel is a file format describing the city.

3D components

The 3D world is populated with a large number of already made 3D components like buildings, vehicles, pedestrians, trees, playgrounds, road signs, bus shelters, waste bins, street lights, surface types etc.

Examples of the output 3D world are shown in Figures 8, 9 and 10 below. We believe they show the resulting city in a representative way.

![Figure 8. View from the generated city.](image-url)
Figure 9. View from the generated city.

Figure 10. View from the generated city.
7. Conclusions

The work in ViPCity went surprisingly well. Although the tasks were sometimes very challenging we did not step into any major pitfalls. We had concerns about the possibility to make the cities come alive, with details and actors, but we managed to solve all problems. Another concern was the usability of ViPEdit, but our efforts there led to what we believe is a user-friendly software. Algorithms for creating road networks, populating cities and simulating sidewalk pedestrians could be created without having to give up on our set goals. We developed a simple but yet powerful and flexible file format for the metamodel. Even the integration and testing at Scania worked out well. Our conclusion must be that it is possible for users without programming skills to create city driving environments (of course there still have to be technicians to run the simulator).

During the development of ViPEdit we have learned many things about how to design an editing tool but also how it can be integrated with the existing platform. We are very pleased with the results we achieved with the editor and feel that we chose the right path. The road networks exported from the editor have been tested, and are working very well, at Scania in their ViP-based simulator.

The choice of developing our own format for the graphical representation was, in our opinion, the right way to go. By defining a custom format, we have been able to adapt it to the development of the project and therefore work more effectively. For the internal saving we feel that a simple plain text format has worked very well. One advantage of having an internal format, which is only intended for use by the editor, is that it was easy to store the data in a way which resembles how it is represented in the editor.

Another conclusion we can draw from the project is that it was possible to build an easy to use road editor that can be used to export OpenDRIVE roads. One of the drawbacks of simplicity over flexibility was that we found that it might not always be possible to re-create real world road networks.

To this project we also see several opportunities for further development of the editor in different areas. We see an increased interest and need to be able to create scenarios using some kind of editor. Here, we believe that ViPEdit is an excellent platform to build upon to create this scenario editor. This would be extra useful in conjunction with being able to import road networks, from both OpenDRIVE and other sources, to be able to add environment and scenarios to them. We believe this feature would be very useful even if the imported roads are not modifiable.

We also recommend to evaluate the possibilities of implementing Bezier-based connection of roads in the editor. There are several different options for how it could work but we believe that with a well thought out implementation it would make the editor both more flexible and easy to use.

In the future we also would like to see the random area population to be a part of the editor. This would be extra powerful if implemented together with functionality to position houses and items by themselves. The generation could then be used both to generate a base for the user to build upon when details are important, and to add bulk to the environment when details are not important.
8. Future activities

During our work and during discussions with the project participants we have identified three main possible future activities:

**Scenario development in ViPEdit**

Creating scenarios was not the scope of ViPCity. But a powerful addition to ViPEdit would be to allow for scenario creation.

**Detailed city design**

As a design choice we decided to let the user draw streets and blocks, while adding all the details is made randomly by the program. The result is that it is quick and easy to draw a city. The drawback is that the user is not in control of details like exactly where to place a waste bin or a tree. Adding this control for advanced users is a possible future activity.

**Real world map data**

A possible next activity would be to generate the city from real world map data, or at least to import that kind of data into ViPEdit to support the user.
References


Stenmarck, M., Leandertz, R., & Blissing, B. (In preparation). Driving Environment Design Tool 2 – DeDT2: Enhanced capacity to produce complex and dynamic traffic environments. (ViP PM.)
ViP is a joint initiative for development and application of driving simulator methodology with a focus on the interaction between humans and technology (driver and vehicle and/or traffic environment). ViP aims at unifying the extended but distributed Swedish competence in the field of transport related real-time simulation by building and using a common simulator platform for extended co-operation, competence development and knowledge transfer. Thereby strengthen Swedish competitiveness and support prospective and efficient (costs, lead times) innovation and product development by enabling to explore and assess future vehicle and infrastructure solutions already today.

Centre of Excellence at VTI funded by Vinnova and ViP partners

VTI, Scania, Volvo Trucks, Volvo Cars, Swedish Transport Administration, Dynagraph, Empir, HiQ, SmartEye, Swedish Road Marking Association

www.vipsimulation.se
Olaus Magnus väg 35, SE-581 95 Linköping, Sweden – Phone +46 13 204000