SAARLAND UNIVERSITY

Faculty of Natural Sciences and Technology I
Department of Computer Science

Bachelor's Thesis

Processing Semantic Information from Procedural Modelling rules for Driving Simulation

Daniel Braun

Supervisor:
Dr. Christian Müller, German Research Center for Artificial Intelligence

Advisor:
Rafael Math, M.Sc., German Research Center for Artificial Intelligence

Reviewers:
Dr. Christian Müller, German Research Center for Artificial Intelligence
Prof. Dr. Dr. hc. mult. Wolfgang Wahlster, Saarland University

Submitted
May 05, 2014

Saarland University
Department 6.2 - Computer Science
Campus - Building E 1.1
66123 Saarbrücken

Abstract

Driving simulators are a safe and cheap way to evaluate automotive user interfaces, which is otherwise often not only very expensive but also dangerous and restricted by laws. But the creation of realistic simulation environments, especially for urban areas, can be very expensive too. So called *procedural modelling* is an approach to solve this problem by rule-based generation of models. Although there is a lot of information in these rules that could be useful for simulations, this information usually gets lost during the export process. This thesis compares different driving simulators and procedural modelling applications, presents an approach to make use of the information contained in the procedural modelling rules and shows how this information could be processed using *CityEngine*, a tool for procedural modelling, and *OpenDS*, an open-source driving simulator.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation

Road tests of new automotive systems are often very problematic: Not only that they are expensive, but also they often can only be conducted on cordoned-off test tracks, due to legal restrictions (cf. figure 1.1, which shows the legal situation using the example of automated driving in the US in February 2014) or safety reasons. For many experiments, like reaction time tests or lane keeping tasks, it is not possible to provide realistic conditions on cordoned-off test tracks, like oncoming traffic or visual distraction. Even if testing on public roads is possible, it is not possible to control all parameters there, like weather, traffic situation, etc. and reproduce them, what may be necessary to get reliable results. For these and more reasons, driving simulators became very popular in automotive research.

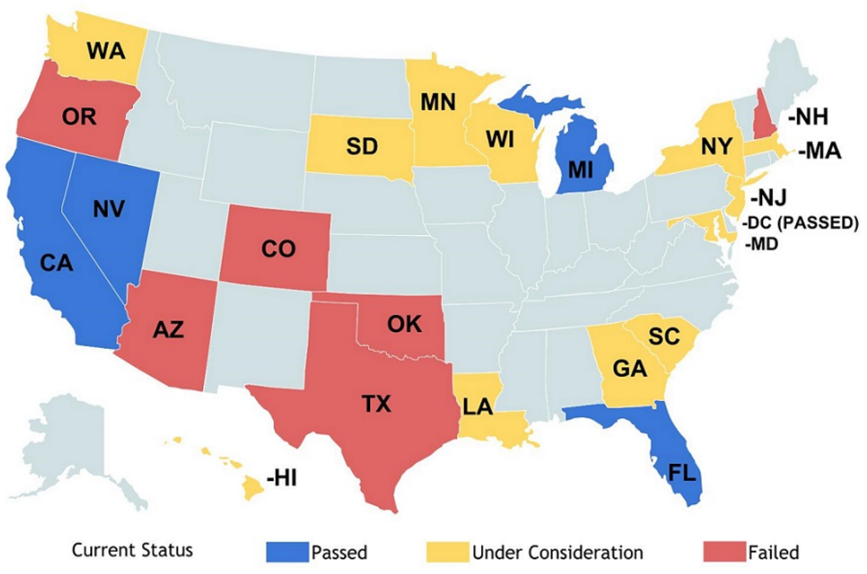


Figure 1.1: US-states that have (or are considering) bills related to automated driving (source: [26])

In order to conduct realistic driving experiments in such simulators, the environmental modelling is very important. The creation of complex models, like urban areas (cf. figure 1.2), can be very time-consuming and therefore expensive. Procedural modelling can help to reduce this effort. Instead of modelling each and every single building and street by hand, streets and buildings are generated randomly and automatically, according to given rules. Most often these models are exported to a common 3D format in order to use them in driving simulators.

During this export process, information contained in the procedural modelling rules can be lost. E.g. there can be additional (so called *semantic*) information about the number of lanes of a street during the process of procedural modelling, but after the export this information will be only available hidden in the texture of a 3D-object and not as structured information, although this could be very useful. Additional semantic information about the 3D-objects can help to improve

Figure 1.2: Parisian like city, created by procedural modeling using CityEngine [1]

driving simulations, as it could enable features like "in-game" navigation and could help to better interpret results, as it could e.g. help to determine parameters like the visual distraction.

For *OpenDS*[2], a modular open-source driving simulator developed at the German Research Center for Artificial Intelligence (DFKI)[14] (see also section 2.1.4), this thesis will present an approach to process the semantic information of the procedural modelling process in a way that it can be used in the driving simulation.

1.2 Goal Description

The major goal of this thesis is to create a 3D-model export plugin, which allows us to create semantically enriched 3D-models for driving simulations, with a procedural modelling application.

Therefore, the following objectives should be achieved:

- **Find an appropriate procedural modelling application**

  Find and compare existing procedural modelling applications with regard to the suitability for the goals of this thesis.

- **Examine the available semantic information**

  Depending on the chosen procedural modelling application the available semantic information has to be examined and rated according to the utility for a driving simulation.

[1]Example project available at `http://www.arcgis.com/home/item.html?id=a6bfebeec4664a00a1ff1eacaadac961`

[2]`http://www.opends.eu`

- **Find a suitable export format**

  Get an overview of existing file formats for 3D-models in simulators and check if these formats can store semantic information. If there is no appropriate format, define a new format or extend an existing format.

- **Implement an export plugin**

  Implement a plugin or program that can export a 3D-model enriched with semantic information from the chosen procedural modelling application.

- **Evaluate the export plugin**

  Evaluate the export plugin and the created models with regard to appearance and performance as well as usability.

2 Background

2.1 Driving Simulation

Because of the different areas of application for driving simulators, requirements vary greatly and there are a lot of specialized solutions for exactly one task. Although we will only consider simulators for research purposes here, there are of course many simulators for other purposes, like driver's education or entertainment. An example for a specialized driving simulation for research purposes is the *LCT Kit* developed at the Leibniz Research Centre for Working Environment and Human Factors[4] which implements the popular lane-change-task (LCT), introduced by [15]. Another example is a simulator developed by Kim et al., which simulates a multi-vehicle platoon[12]. Both of these examples are very specialized and each consist of only one straight road without any additional environment models. The advantages of such specialized systems are obvious: They are easier to implement and therefore cheaper than universal systems, that can handle different tasks.

But this is only true as long as no other task has to be simulated, since every task has to be built from scratch again. In the long term it is therefore better to develop and use an universal driving simulator that can easily be adapted to new tasks. Because different tasks need different environments, driving simulators with an universal approach have to provide a possibility to load different environment models. Most simulators therefore support ordinary 3D-models in different formats. Sections 2.1.1 to 2.1.4 will present some of the most popular available driving simulators for research purposes.

While the user can easily visually distinguish between streets, buildings, trees and other 3D-objects, the simulator can not. The common 3D formats only store information about the shape and the texture of an object (cf. section 2.3.1 for a closer look at 3D-formats). This means, the simulator that loads these objects does not know anything about the objects except for the shape and the texture. Although this can be satisfactory for some simple tasks, in general it is necessary, or at least desirable, to have more information about an object, so called semantic information. Since there is no consensus about the term "semantic information" and its exact meaning, neither in general nor in this special case (and this is a rather philosophical question[5]), we will interpret the term generously in this thesis and call everything semantic information that describes an object, which is represented by a 3D-model, in more detail.

The most basic example for semantic information in this context is maybe the information which sort of object a 3D-model represents (e.g. a street, a car, a tree, a building, a pedestrian etc.). Depending on the object there are a lot of other semantic information. For streets e.g.:

- number of lanes
- track width
- speed limit and other traffic rules
- street name
- ...

For buildings e.g.:

- number of floors
- number of windows
- façade colour
- house number
- number of residents
- ...

Of course, these lists are not complete and rather random. This raises the question which semantic information is of interest for driving simulations. As mentioned before, this depends on the observed object, but it also depends on the task that should be simulated. If we want to examine if a driver can obey the road rules while he is texting, we need accurate knowledge about speed limits and traffic rules, whereas information about surrounding buildings is not that important for this task. But for other tasks, like calculating the visual distraction of a driver, this information can be of great importance. Since all this information is not included in ordinary 3D-formats, there are some existing approaches to store this information. Some of these approaches will be introduced later.

In the following, we present a selection of driving simulators:

2.1.1 STISIM Drive

STISIM Drive[1] is a commercial driving simulator developed by System Technology Inc. from Hawthoren, California. STISIM Drive is offered as software and as whole system with hardware (cf. figure 2.1). It is advertised for research, rehabilitation, assessment and driver education. The heart of the software is the *STISIM Drive Scenario Definition Language (SDL).* The SDL is used to define the environment and place models (including streets, vehicles, pedestrians, signs, buildings, flora and fauna and other objects), to control traffic and pedestrians and to activate built in standard tasks like car following, divided attention and simple pedal reaction times.[11] STISIM Drive comes with a bunch of different ready-made models but also supports (non-animated) user generated models in OGRE-XML format (for more information about the format see section 2.3.1). Moreover, STISIM Drive offers a module interface, that allows the user to write own modules for STISIM Drive in C, VB and other languages that support Microsoft's Component Object Model. STISIM Drive is very popular and is used for many research projects, like "The impact of cell phones on simulated driving performance"[1], "Measuring distraction potential of operating in-vehicle devices"[20] or "Following slower drivers: Lead driver status moderates driver's anger and behavioural responses and exonerates culpability"[25].

2.1.2 SCANeR

Another very popular[23] commercial driving simulator is *SCANeR*[2], from the french company OKTAL. SCANeR is offered in two different packages, *SCANeR studio*, dedicated to engineering and research, and *SCANeR DT*, dedicated to training and safety awareness.[22] SCANeR uses the RoadXML format for saving 3D information, as well as semantic information. We will take a closer look on RoadXML in section 2.3.2. SCANeR also offers a powerful tool for scenario

[1]`http://www.stisimdrive.com/`
[2]`http://www.scanersimulation.com/`

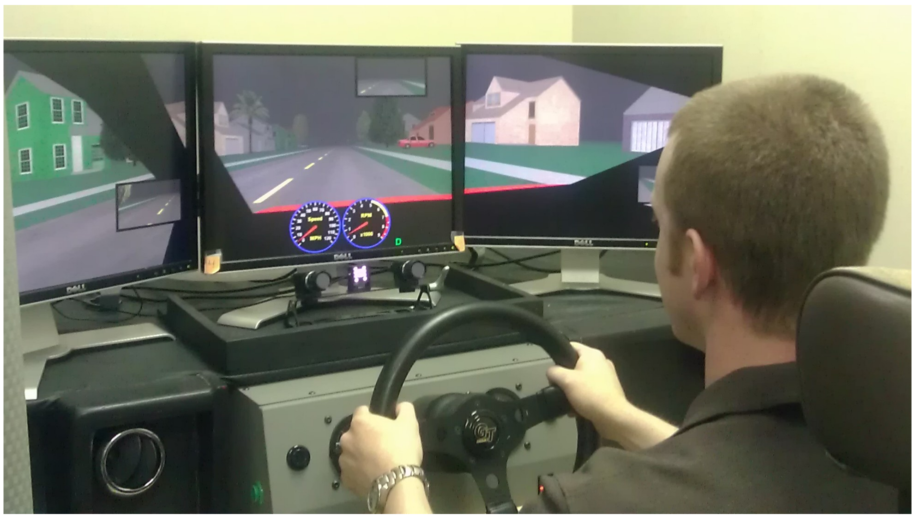


Figure 2.1: STISIM (M500) setup (source: [6])

generation, called *SCANeR studio Terrain*, which offers automatic road network creation from a GIS representation.[21] In contrast to the later presented procedural modelling applications, SCANeR studio Terrain does not offer automatic generation of buildings.

2.1.3 TORCS

TORCS[3] is one of the few open source driving simulators. Although TORCS was originally made for racing simulation, it is often used for non-racing research, due to its adaptability. TORCS also comes with pre-programmed AI cars. Though TORCS misses a lot of comfort functions that are offered by commercial systems, e.g. there is no ready-to-use interface for plugins or data exchange with other programs. TORCS uses its own format ("acc", an extension of the "ac"-format) for 3D-objects.

2.1.4 OpenDS

OpenDS (cf. figure 2.2) is a modular, cross-platform, open source driving simulation software for research, developed at the DFKI, fostered by the EU-project *GetHomeSafe*. OpenDS is based on the open source game engine *jMonkeyEngine*. Beside the advantages of an open source software, OpenDS also offers a lot of comfort function that are usually only known from commercial systems, like a graphical driving task editor, that allows the user to place objects (like road signs or traffic lights) on the map while driving around, and an automatic drive analyser, which can be used to record and visualize information like position, speed or fuel consumption.[14]

Although the jMonkeyEngine supports 3D-models in Wavefront OBJ and OGRE-XML format (both will be introduced in section 2.3.1), the OGRE-XML support is considered to be more reliable. Beneath the models, each OpenDS driving task consist of five additional XML files:

- a **main** file, from where the driving task is started and all files are linked
- an **interaction** file, where triggers and activities are defined

[3]`http://torcs.sourceforge.net/`

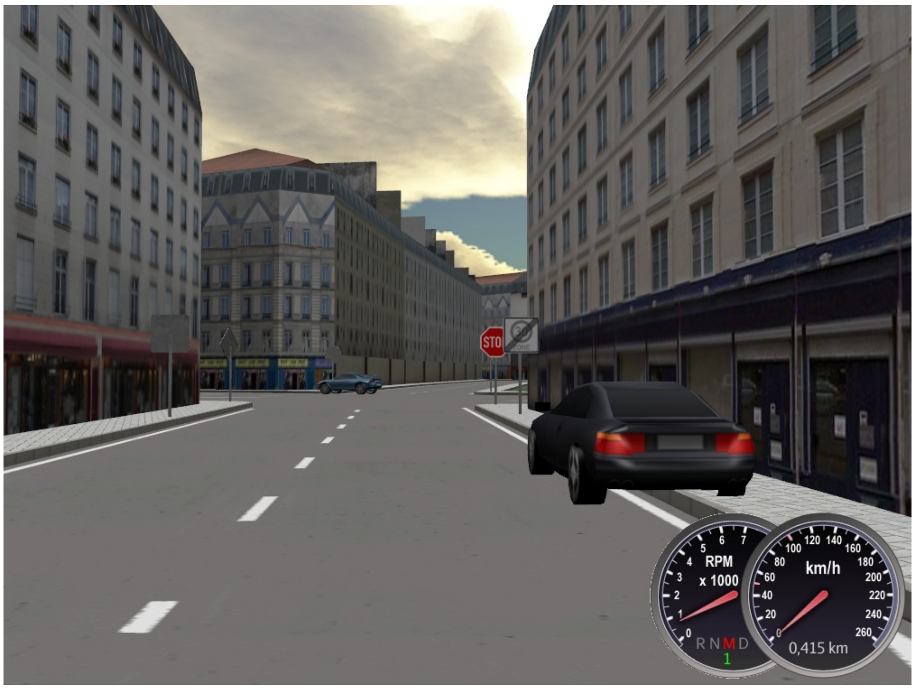


Figure 2.2: CityEngine-scene in OpenDS (source: [27])

- a **scenario** file, where environment variables (e.g. weather), driver and car variables (e.g. tires, max. speed and transmission) and traffic variables can be set
- a **scene** file, where the 3D-models are included
- a **settings** file, where additional simulation parameters (e.g. controller and connection to network services) can be set

At the moment OpenDS does not support any file format for semantic information.

2.2 Procedural Modelling

The main idea behind so called procedural modelling is the (semi-) automatic generation of complex 3D-models based on a given set of rules. The aim is to create a model that is rich in variety and realistic. Today, procedural modelling is mostly used in the game industry, where it is often necessary to create huge realistic landscapes with vegetation. While there are a lot of procedural modelling applications on the market for trees or landscapes, the market for procedural modelling applications for urban areas is quite small, probably due to the complexity of the task. Another reason may be that there are many tools for (semi-) automatic creation of 3D city models from real data, like satellite images or maps. Pascal Müller (who developed the first version of CityEngine) introduced some methods that are now very popular for the procedural modelling of cities in his papers [18], [16] and [17].

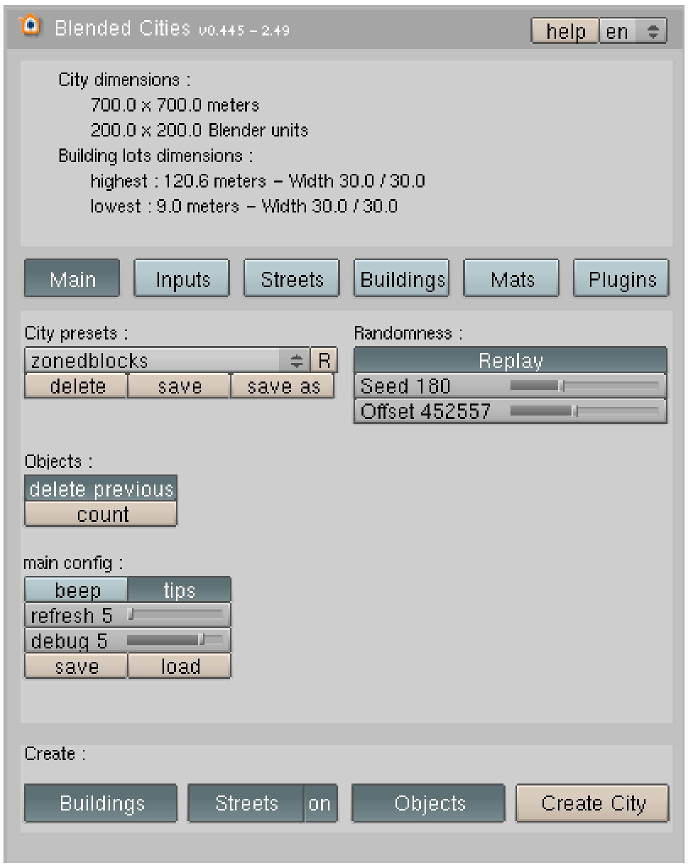
For this thesis, we are looking for a procedural modelling application that is compatible with OpenDS. Moreover we are looking for a powerful way to describe rules and a possibility to export semantic information, e.g. with a built in scripting engine.

2.2.1 City Kit

City Kit[4] is a plugin for the 3D modelling application *CINEMA 4D*. City Kit only consists of two texture sets, one for day and one for night. Moreover it does not support the usage of custom rules and only very basic parameters (like the city size or maximal building height) can be manipulated. CINEMA 4D and therefore City Kit can not export OGRE-XML files.

2.2.2 Blended Cities

Blended Cities[5] is another plugin, which works with the popular open source 3D graphics software *Blender* and is open source itself. In contrast to City Kit, Blended Cities offers a lot of manipulatable parameters (cf. figure 2.3), like street width, number of lanes, footway width etc. Moreover Blended Cities allows the usage of own textures and, due to the fact that it is open source, it is completely adaptable. Another advantage is that Blender, and therefore Blended Cities, supports a lot of different export formats, including OGRE-XML.


Blended Cities v0.445 - 2.49

City dimensions :
700.0 x 700.0 meters
200.0 x 200.0 Blender units
Building lots dimensions :
highest : 120.6 meters - Width 30.0 / 30.0
lowest : 9.0 meters - Width 30.0 / 30.0

City presets : zonedblocks
Randomness : Seed 180, Offset 452557
main config : refresh 5, debug 5

Figure 2.3: Blended Cities options screen

Nevertheless Blended Cities has some negative aspects. As figure 2.4 shows the level of detail of the generated scenes is very low and the variety rate of the buildings is low. Additionally there is no easy way to change the road network manually.

[4]`http://greyscalegorilla.com/blog/store/city-kit/?page=training`
[5]`http://jerome.le.chat.free.fr/index.php/en/download/blender/city-engine-download.html`

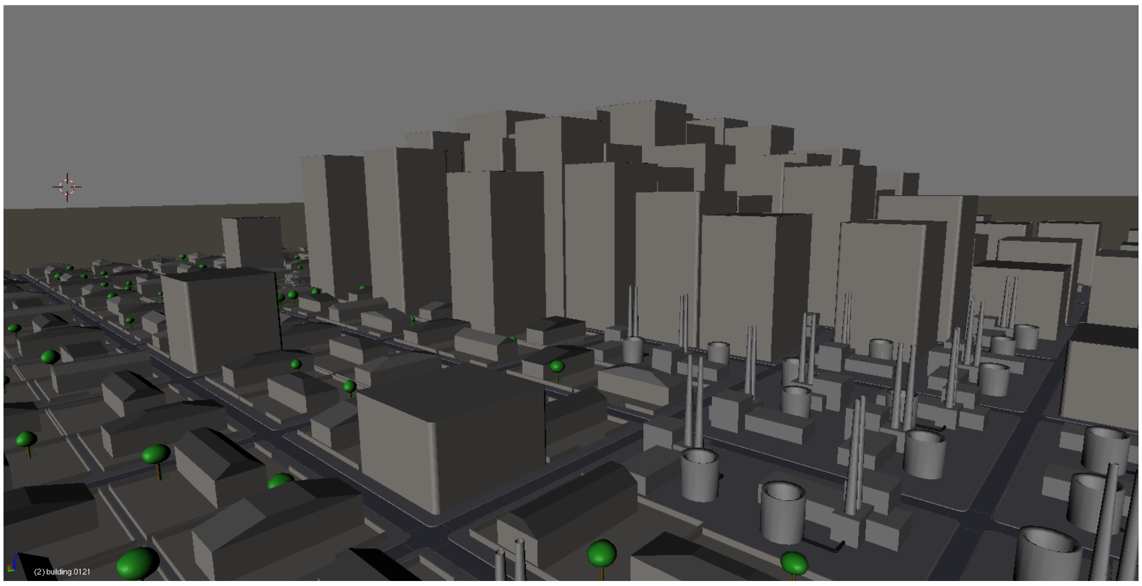


Figure 2.4: City created with Blended Cities

2.2.3 CityEngine

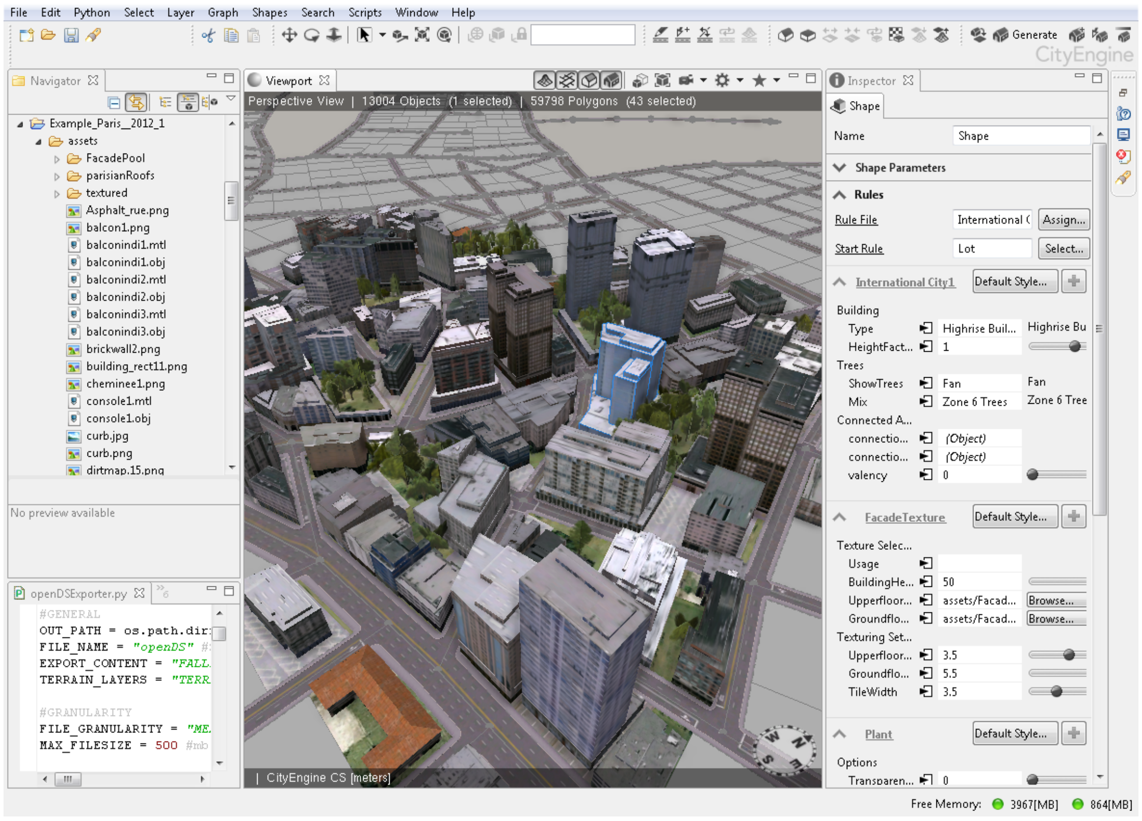
CityEngine[6] is a commercial procedural modelling application formerly developed by *Procedural Inc.*, now *Esri*. At the moment CityEngine is most likely the most powerful application for the procedural modelling of urban areas. It comes with different ready-to-use assets, an easy to use WYSIWYG[7] editor (cf. figure 2.5) and a powerful rule description language called *CGA*[8]. Moreover CityEngine supports many different output formats and has an integrated Python scripting engine.

CGA-Rules

The CGA rule language is a powerful scripting language, therefore it is not possible to cover the whole functionality here, nonetheless we want to give a short insight how CityEngine implements the procedural modelling concept, based on the minimal example from listing 2.1.

```
randomHeight = rand(10,100)

attr facadeColor = 60%: "#ff7777"
                   else: "#58FA58"


Lot  --> extrude(randomHeight) Mass
Mass --> comp(f) { top: Top | all: Facade }
Top  --> roofShed(10, 2) Roof

Roof -->
          color("#C0C0C0")

Facade -->
```

[6]http://www.esri.com/software/cityengine
[7]what you see is what you get
[8]Computer Generated Architecture


File Edit Python Select Layer Graph Shapes Search Scripts Window Help
Generate
CityEngine
Navigator
Viewport
Perspective View | 13004 Objects (1 selected) | 59798 Polygons (43 selected)
| CityEngine CS [meters]
Inspector
No preview available
Free Memory: 3967[MB] 864[MB]

```
#GENERAL
OUT_PATH = os.path.dir
FILE_NAME = "openDS" #
EXPORT_CONTENT = "FALL
TERRAIN_LAYERS = "TERR

#GRANULARITY
FILE_GRANULARITY = "ME
MAX_FILESIZE = 500 #mb
```

Shape

Name	Shape	
Rule File	International (	Assign...
Start Rule	Lot	Select...
International City1	Default Style...	
Building		
Type	Highrise Buil...	Highrise Bu
HeightFact...	1	
Trees		
ShowTrees	Fan	Fan
Mix	Zone 6 Trees	Zone 6 Tree
Connected A...		
connectio...	(Object)	
connectio...	(Object)	
valency	0	
FacadeTexture	Default Style...	
Texture Selec...		
Usage		
BuildingHe...	50	
Upperfloor...	assets/Facad...	Browse...
Groundflo...	assets/Facad...	Browse...
Texturing Set...		
Upperfloor...	3.5	
Groundflo...	5.5	
TileWidth	3.5	
Plant	Default Style...	
Options		
Transparen...	0	

Figure 2.5: CityEngine user interface

```
15          color(facadeColor)
```

Listing 2.1: CGA rule example

As mentioned before, one of the key concepts of procedural modelling is the random variation of values within given rules. In the short example above we have two different manifestations of this concept. In line 1 we set the variable “randomHeight” to a random value between 10 and 100, a concept that is available in all common scripting languages. In line 3 and 4, there is a different, a probabilistic, concept. The attribute “facadeColor” is set to the colour “#ff7777” (red) with a probability of 60%. It follows, that it is set to colour “#58FA58” (green) with a probability of 40%.

Semantic Information

The available semantic information about buildings strongly depends on the given rules. In the example above a red façade colour could describe a residential house and a green could describe commercial properties. We could also rewrite the generation of the random height to something like this:

```
floors = rand(4,40)

Lot  --> extrude(2.5 * floors) Mass
```

Listing 2.2: CGA rule with semantic information about the number of floors

With this change our buildings will still have a height between 10 and 100 meters, but we also get additional semantic information, the number of floors. In general, any information can be attached to a model in CityEngine using the `report(identifier, value)` command. It can store a string identifier and a string, boolean or float value.

The situation for streets is a bit different, since we always have some guaranteed minimal amount of semantic information about streets in CityEngine. This information is not contained in the CGA rules, but in the street network.

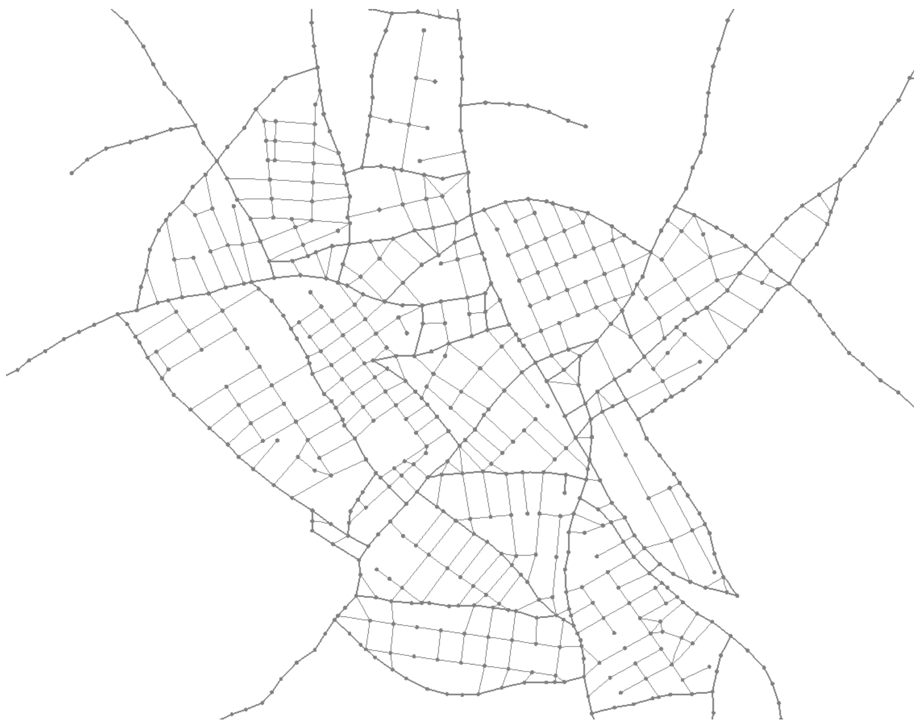


Figure 2.6: Street network in CityEngine

Figure 2.6 shows such a street network in CityEngine. It is represented by a graph with nodes, for junctions, and edges for stretches of road. Moreover, each edge has some attributes:

- street width
- sidewalk width left
- sidewalk width right
- street type (minor or major street)

Of course there can be more semantic information about a road (e.g. street names, a speed limit or a one-way street), this additional information could again be modelled with CGA rules. Especially detailed lane information (width, positioning, direction) is not available from the street network and has to be modelled with CGA rules.

Supported Export Formats

CityEngine supports eight different exports format:

- Keyhole Markup Language (KML)
- CityEngine WebScene
- COLLADA
- Wavefront OBJ
- Autodesk FBX
- E-On Software Vue
- Picar RenderMan
- Esri FileGDB

Although none of these formats is particularly suitable to be used with OpenDS, the OBJ format can be converted easily in OGRE-XML. Furthermore, none of these formats can store additional semantic information. Both problems can be solved using the built in scripting engine of CityEngine.

Scripting Engine

CityEngine has a built in Python scripting engine that uses *Jython*[9]. This scripting engine can be used during the creation of models, but also during the export of models. Scripts can be executed automatically before the export, after the export and during the export, after the preparation of each model. We could execute a script after the export that converts the generated OBJ files to OGRE-XML and we could execute a script during the export that saves the semantic information of a model stored with the `report` command.

2.2.4 Procedural Modelling Application Comparison

If we compare the procedural modelling applications introduced in this section, it is pretty obvious that CityEngine is the only application that is suitable for the goals of this thesis, especially due to its powerful rule language and its scripting engine. A detailed comparison of the introduced applications (and the desired functionality) can be found in table 2.1.

Application	**Comp. Export Format**	**User-Friendly**	**Rule Language**	**Scripting Engine**
City Kit	-	+	-	-
Blended Cities	+	-	-	+
CityEngine	o	+	+	+
Desired	+	+	+	+

Table 2.1: Comparison matrix for procedural modelling applications

2.3 File Formats

In this section, we will introduce some common file formats for 3D-models and semantic information that could be useful for the goals of this thesis.

[9] `http://www.jython.org/`

2.3.1 3D Formats

Wavefront OBJ

Wavefront OBJ is a geometry definition file format, that was developed by *Wavefront Technologies* in the 1980s. OBJ uses *MTL* auxiliary files to define material information for OBJ files. The OBJ definition itself is very open (e.g. the faces definition), but many programs only implement a subset of the OBJ definition[3]. Therefore, we focus on the syntax of the subset of OBJ at this point that is implemented by CityEngine:

Geometry definition syntax

- **Vertices**

  A vertex is defined by a line starting with a `v`, followed by x, y and z coordinates.

  ```
  v 10.123 -10.000 1.337
  ```

- **Texture coordinates**

  Lines that define texture coordinates start with `vt` and contain a u and a v coordinate.

  ```
  vt 1.2818 0.1667
  ```

- **Normals**

  Normals are defined by `vn` and x, y and z coordinates.

  ```
  vn 0.33 -1.00 0.66
  ```

- **Polygons**

  As already indicated, the definition of polygons in OBJ is very loose. The simplest possible definition of a polygon starts with a `f`, followed by three vertices indexes. Each vertex, texture coordinate and normal, has a index, starting with one (for each group) for the first entry. So a polygon could look like this:

  ```
  f 1 13 5
  ```

  The number of vertices a polygon could consist of is not limited by the OBJ definition. A polygon doesn't even have to be coplanar. But due to restrictions from CityEngine and the jMonkeyEngine, we only work with coplanar triangles. Moreover, CityEngine uses a more complex way to define polygons, using not only vertices, but also texture coordinates and normals. The definition schema is: `f v/vt/vn v/vt/vn v/vt/vn`, where `v` is the index of a vertex, `vt` is the index of a texture coordinate and `vn` is the index of a normal.

  ```
  f 1/3/5 13/1/6 5/120/137
  ```

- **Groups**

  In OBJ, groups are used to group polygons, unlike OGRE-XML, OBJ does not group vertices. Every polygon below the definition of a group belongs to this group. The definition of a group starts with a `g`, followed by the name of the group.

  ```
  g Floor_3
  ```

- **Reference Materials**

  Two commands are used to refer to materials defined in MTL files. First we need to integrate the MTL file that contains the material. This can be done with `mtllib` and the name of the

MTL file.

```
mtllib new_scene.mtl
```

Second we need to choose a material from the MTL file. This can be done with "usemtl" and the name of the material. The choosen material will be used for the polygons that follow.

```
usemtl FacadeSets_1
```

Material definition syntax

- **Material Name**

  Every material needs a name to be addressed from the OBJ file. The name of a material can be defined using "newmtl":

  ```
  newmtl FacadeSets_1
  ```

- **Material colour**

  There are three material colour options: the ambient material colour, the diffuse material colour and the specular material colour. All three colours can be controlled by giving a rgb-value (as usual, each value has to be between 0 and 256). The command for ambient is `Ka`, for diffuse `Kd` and for specular `Ks`.

  ```
  Ka 236 30 209
  Kd 236 30 209
  Ks 236 30 209
  ```

- **Transparency**

  The transparency of a material can be set using the letter `d`. The default value is 1.0 (no transparency), 0.0 means completely transparent.

  ```
  d 1.0
  ```

- **Shininess**

  The shininess can be set to a value between 0 (default) and 1000 using `Ns`.

  ```
  Ns 0.0
  ```

- **Illumination**

  With `illum` the illumination model can be chosen. There are 11 illumination models available with numbers from 0 to 10 (cf. table 2.2).

  ```
  illum 3
  ```

- **Optical Density**

  The optical density (or index of refraction) can be a value between 0.001 and 10, where 1.0 means that light is not bended by the material.

  ```
  Ni 1.0
  ```

- **Textures**

  Textures can be applied with `map_Kd` and the name of the corresponding image file.

  ```
  map_Kd genCourInterieure1.png
  ```

Model number	Properties
0	Colour on and Ambient off
1	Colour on and Ambient on
2	Highlight on
3	Reflection on and Ray trace on
4	Transparency: Glass on; Reflection: Ray trace on
5	Reflection: Fresnel on and Ray trace on
6	Transparency: Refraction on; Reflection: Fresnel off and Ray trace on
7	Transparency: Refraction on; Reflection: Fresnel on and Ray trace on
8	Reflection on and Ray trace off
9	Transparency: Glass on; Reflection: Ray trace off
10	Casts shadows onto invisible surfaces

Table 2.2: Illumination models in OBJ[2]

OGRE-XML

OGRE-XML is a geometry definition file format developed for the *Object-Oriented Graphics Rendering Engine* (OGRE), a multi platform open source rendering engine. OGRE-XML uses `.mesh` files to save the geometry definition and *MATERIAL* auxiliary files to define material information. Because OGRE-XML is a XML based format, it is easy to read and edit, but large in file size. Therefore OGRE-XML and MATERIAL files can be converted to a binary format, using the *OgreXMLConverter*. Again, we focus on the subset of the OGRE-XML definition that is relevant for our task.

Geometry definition syntax

- **Vertices**

  In OGRE-XML, each vertex is defined using not only a x, y and z coordinate, but also a texture coordinate and a normal. A vertex in OGRE-XML looks like listing 2.3.

```
<vertex>
        <position x="-44.280" y="0.250" z="-186.694"/>
        <normal x="-0.914" y="0.000" z="0.405"/>
        <texcoord u="0.0000" v="1.0"/>
</vertex>
```

Listing 2.3: Vertex in OGRE-XML

  Vertices are grouped in a `vertexbuffer`, which are grouped in `geometry`, cf. listing 2.4.

```
<geometry vertexcount="2">
        <vertexbuffer positions="true" normals="true"  texture_coords="1">
                <vertex>
                        <position x="-44.280" y="0.250" z="-186.694"/>
                        <normal x="-0.914" y="0.000" z="0.405"/>
                        <texcoord u="0.0000" v="1.0"/>
                </vertex>
                <vertex>
                        <position x="-35.967" y="0.250" z="-167.945"/>
                        <normal x="-0.914" y="0.000" z="0.405"/>
                        <texcoord u="1.2818" v="1.0"/>
                </vertex>
        </vertexbuffer>
```

```
</geometry>
```

Listing 2.4: Group of vertices in OGRE-XML

- **Polygons**

  In OGRE-XML polygons only consist of vertices, because each vertex has his own normal and texture coordinate. Another difference is, that OGRE-XML starts counting with 0, not with 1. Polygons are grouped in `faces`, cf. listing 2.5.

```
<faces count="2">
        <face v1="0" v2="1" v3="2"/>
        <face v1="0" v2="2" v3="3"/>
</faces>
```

Listing 2.5: Polygon in OGRE-XML

- **Groups**

  Groups are expressed in OGRE-XML as `submeshes` (cf. listing 2.6). Each submesh has a name and information about its material. Unlike OBJ, OGRE-XML also groups vertices. That means, each submesh starts counting vertices with 0.

```
<mesh>
        <submeshes>
                <submesh material="FacadeGF" usesharedvertices="false">
                        <faces/>
                        <geometry/>
                </submesh>
        </submeshes>
</mesh>
```

Listing 2.6: Submesh in OGRE-XML

Material definition syntax

- **Material name**

  As in OBJ, every material needs a name to be addressed:

  `material FacadeSets_2{}`

- **Material colour**

  As for OBJ files, we have three colour options, one for ambient, one for diffuse and one for specular, each as rgb value. Additionally, in OGRE-XML, a fourth value is added to each colour. For ambient and diffuse a alpha value is added (equal to `d` in OBJ), for specular a shininess value is added (corresponds to `Ns`).

```
ambient 1.0 1.0 1.0 1.0
diffuse 0.0 0.0 0.0 1.0
specular 0.5 0.5 0.5 1.0
```

- **Textures**

  Texture files can be used equivalent to OBJ, only the keyword is replaced by `texture`.

  `texture genCourInterieure1.png`

2.3.2 Semantic Information Formats

CityGML

The *City Geography Markup Language* (CityGML) was developed by the *Special Interest Group 3D* and is a file format for the representation and exchange of virtual 3D cities, containing both, 3D and semantic information. The latest version of the standard (2.0.0) has been released in 2012. CityGML can represent very detailed 3D information for buildings and streets, even interiors of buildings can be represented with CityGML. Therefore CityGML can also store a lot of semantic information about buildings, like windows, number of floors etc. Unfortunately CityGML can not save semantic information about streets, like speed limits or traffic rules and can not be extended about these aspects.[8] Furthermore the 3D representation of CityGML is not compatible with OpenDS.

RoadXML

RoadXML is a XML file format for the description of road networks developed by *OKTAL*, e.g. used by SCANeR, as mentioned before. In contrast to CityGML, which allows the very detailed description of buildings, RoadXML does not represent buildings, neither as 3D information nor as semantic information. Instead it supports very detailed 3D and semantic information about roads and road networks.

OpenDRIVE

Another XML format is *OpenDRIVE*, developed by *VIRES Simulationstechnologie* and supported by companies like *Rheinmetall*, *BMW* and *Daimler*. According to his developers, "OpenDRIVE is the leading open format and de-facto standard for the description of road networks in driving simulation applications"[7]. OpenDRIVE saves very detailed semantic information about road networks and road objects, like traffic lights or pylons. While the OpenDRIVE definition does not define a schema for buildings, it is very extensible and can store arbitrary semantic information about buildings or other objects.

Listing 2.7 is the representation of the street network from figure 2.7 in OpenDRIVE.

```
<?xml version="1.0" standalone="yes"?>
<OpenDRIVE>
        <header revMajor="1" revMinor="3" name="test" version="1.00
           " date="Tue␣Mar␣11␣08:53:30␣2014" north="0.00" south="
           0.00" east="0.00" west="0.00" maxRoad="3" maxJunc="0"
           maxPrg="0" vendor="Daniel␣Braun">
        </header>
        <road name="" length="10.00" id="1" junction="-1">
                <link>
                        <successor elementType="road" elementId="2"
                            contactPoint="start" />
                </link>
                <planView>
                        <geometry s="0.00" x="0.00" y="0.00" hdg="
                           45.00" length="10.00">
                                <line/>
                        </geometry>
                </planView>
```

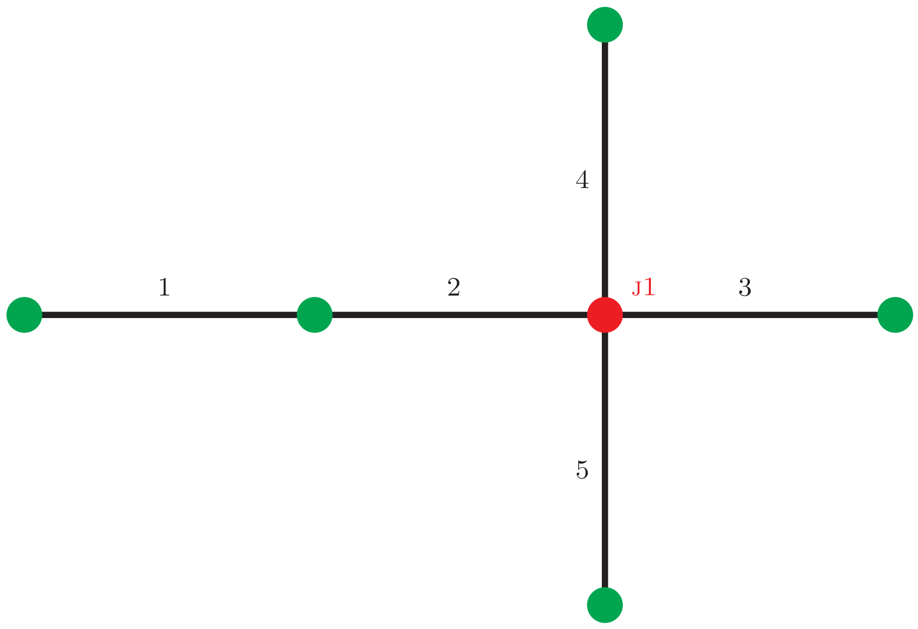


Figure 2.7: Small street network with intersection

```
        </road>
        <road name="" length="10.00" id="2" junction="1">
                <link>
                        <predecessor elementType="road" elementId="
                           2" contactPoint="end" />
                        <successor elementType="junction" elementId
                           ="1" />
                </link>
                <planView>
                        <geometry s="0.00" x="10.00" y="0.00" hdg="
                           45.00" length="10.00">
                                <line/>
                        </geometry>
                </planView>
        </road>
        <road name="" length="10.00" id="3" junction="1">
                <link>
                        <predecessor elementType="junction"
                           elementId="1" />
                </link>
                <planView>
                        <geometry s="0.00" x="20.00" y="0.00" hdg="
                           45.00" length="10.00">
                                <line/>
                        </geometry>
                </planView>
        </road>
        <road name="" length="10.00" id="4" junction="1">
                <link>
                        <predecessor elementType="junction"
                           elementId="1" />
                </link>
                <planView>
                        <geometry s="0.00" x="20.00" y="0.00" hdg="0.00" length="10.00">
                                <line/>
                        </geometry>
                </planView>
        </road>
        <road name="" length="10.00" id="5" junction="1">
                <link>
                        <predecessor elementType="junction" elementId="1" />
                </link>
                <planView>
                        <geometry s="0.00" x="20.00" y="0.00" hdg="180.00" length="10.00">
                                <line/>
                        </geometry>
                </planView>
        </road>
        <junction name="" id="1">
                <connection id="0" incomingRoad="2" connectingRoad="3" contactPoint="start"></connection>
                <connection id="1" incomingRoad="2" connectingRoad="4" contactPoint="start"></connection>
                <connection id="2" incomingRoad="2" connectingRoad="5" contactPoint="start"></connection>

                <connection id="3" incomingRoad="3" connectingRoad="2" contactPoint="end"></connection>
                <connection id="4" incomingRoad="3" connectingRoad="4" contactPoint="start"></connection>
                <connection id="5" incomingRoad="3" connectingRoad="5" contactPoint="start"></connection>

                <connection id="6" incomingRoad="4" connectingRoad="2" contactPoint="end"></connection>
                <connection id="7" incomingRoad="4" connectingRoad="3" contactPoint="start"></connection>
                <connection id="8" incomingRoad="4" connectingRoad="5" contactPoint="start"></connection>

                <connection id="9" incomingRoad="5" connectingRoad="2" contactPoint="end"></connection>
                <connection id="10" incomingRoad="5" connectingRoad="3" contactPoint="start"></connection>
                <connection id="11" incomingRoad="5" connectingRoad="4" contactPoint="start"></connection>
        </junction>
</OpenDRIVE>
```

Listing 2.7: OpenDRIVE representation of the street network from figure 2.7

Buildings or other objects can be described in the road element with the structure presented in listing 2.8.

```
<objects>
        <object id="1" type="type" name="some␣object" s="10.00" t="
           10.00" length="5.00" width="5.00" height="5.00">
                <userData>
                </userData>
        </object>
</objects>
```

Listing 2.8: Representation of an object in OpenDRIVE

Where “s” and “t” describe the position of the object, relative to the road it is attached to and “user data” can store arbitrary semantic information.

2.3.3 File Format Comparison

As the best results with the jMonkeyEngine can only be achieved with OGRE-XML and OGRE-XML can not be extended by semantic information, we will need to use two file formats, one for 3D information and one for semantic information. OpenDRIVE fits our needs perfect: We do not need 3D information but we do need a very flexible solution for semantic information, because our available semantic information varies, depending on the rules we use to create our models. Table 2.3 shows a detailed comparison of the different file formats.

Format	**3D info.**	**Sem. info. (Bldg. / St.)**	**Editable**	**Extendable**	**jME comp.**
OBJ	+	-/-	+	-	o
OGRE	+	-/-	+	-	+
CityGML	+	+/-	+	-	-
RoadXML	o[10]	-/+	+	-	-
OpenDRIVE	-	-/+	+	+	n.a.
Desired	+	+/+	+	+	+

Table 2.3: Comparison matrix for file formats

2.4 Related Work

2.4.1 A declarative approach to procedural modeling of virtual worlds

In their paper “A declarative approach to procedural modeling of virtual worlds”[24], from 2011, Smelik et al. presented an approach to make procedural modelling more easy and intuitive. In their opinion, today’s procedural modelling applications “are complex and unintuitive to use, hard to control”[24].

To solve these problems, they combined manual modelling and procedural modelling in a new way in an application called *SketchaWorld*: In a first step, the user draws rivers, lakes, streets, and boundaries for forests and cites on a 2D map. In the next step trees and houses will be automatically generated in between these boundaries. In contrast to “classic” procedural modelling applications, SketchaWorld does not need user rules to create cities. SketachWorld derives its own rules from the landscape. E.g. in the centre of a city, SketchaWorld will place high-class residential and commercial buildings, while heavy industries will be placed at the borders of the city and near

[10]RoadXML files do not contain 3D information about buildings.

rivers. Each building also influences the attractivity of an area and therefore its neighbourhood. That way, SketachWorld creates a lot of semantic information, without any user rules.

SketachWorld supports export as 3D-models in OpenSceneGraph-, COLLADA- or GIS-format.

2.4.2 Pro-SiVIC and Roads, a software suite for sensors simulation and virtual prototyping of adas

In 2010, Hiblot et al. (from the company *Civitec*) presented *ROADS* in their paper "Pro-SiVIC and Roads, a software suite for sensors simulation and virtual prototyping of adas"[10]. ROADS is a procedural modelling application for road networks. Like SketchaWorld, ROADS offers the possibility to draw roads on a 2D map, but in contrast to SketchaWorld, it does not support any additional objects, like buildings. In exchange, ROADS does not only support the export of 3D-models (in OBJ format), but also the export of semantic information in OpenDRIVE format. Moreover ROADS offers the automatic creation of roads along a path, defined by points (e.g. GPS points), and the automatic connection of road segments.

While ROADS is only dedicated to the creation of road networks, Civitec offers another software, dedicated to scenario modelling and simulation, called *Pro-SiVIC*[11]. Pro-SiVIC can import road networks created with ROADS, but does not offer any procedural modelling functions itself.

2.4.3 Semantic Road Network Models for Rapid 3D Traffic Scenario Generation

A highly related work, called "Semantic Road Network Models for Rapid 3D Traffic Scenario Generation"[9], was published in 2013 by Haubrich et al. They presented an approach to convert real world information (from *OpenStreetMap*) into both, semantic and 3D information. The aim was the creation of a road network that can be used for traffic simulation, as part of the AVeSi project ("Agentenbasierte Verkehrssimulation").

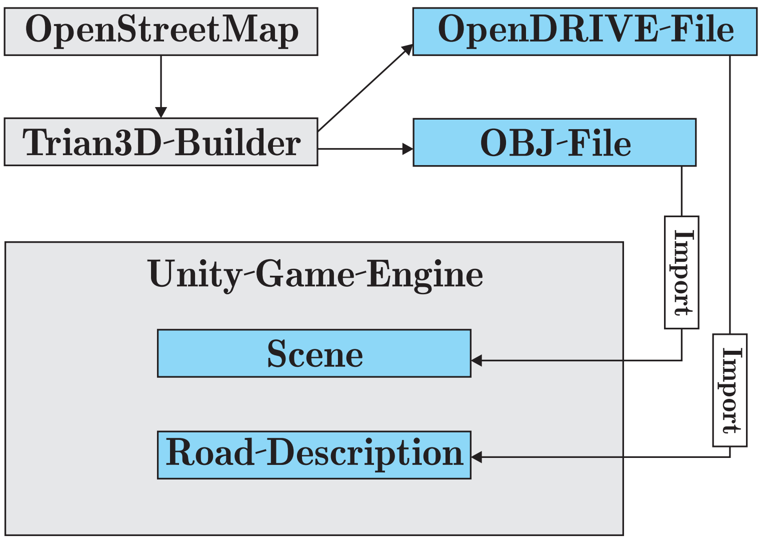


Figure 2.8: Illustration of the workflow for rapid scenario generation using Trian3DBuilder[9]

[11]http://www.civitec.com/solutions/pro-sivic-base-edition.html

Figure 2.8 illustrates the workflow of Haubrich et al. They imported the data from OpenStreetMap in *Trian3D Builder*[12]. From there, they exported 3D information in OBJ format and semantic information in OpenDRIVE format. For the actual simulation, they used the Unity Engine[13].

2.4.4 Related Work Comparison

If we compare the related work presented in this section with the goals of this thesis (cf. table 2.4), we see that all these works use procedural modelling (in different ways). But none of these papers allows completely user generated rules for buildings and streets, what is absolutely necessary, if we want to create highly controlled environments for simulations. Moreover none of the presented approaches uses OGRE-XML for saving 3D information, the format we want to use because of the jMonkeyEngine support, but [10] and [9] use OBJ, which can easily be converted to OGRE-XML. Although all three have additional semantic information, only [10] and [9] can export this semantic information. The fact that both tools use OpenDRIVE endorse our decision to use this format too.

Work	Procedural Modelling	User rules	jME comp.	3D inf.	Semantic inf.
[24]	+	-	-	+	o
[10]	+	-	o	+	+
[9]	+	-	o	+	+
This thesis	+	+	+	+	+

Table 2.4: Comparison matrix for related work

[12]http://www.triangraphics.de/?q=en/produkte/Trian3d-Builder
[13]http://www.unity3d.com

3 Practical Implementation

After we decided which procedural modelling application (CityEngine) and which file formats we will use (OGRE-XML and OpenDRIVE), we will present the concrete implementation of a tool that automatically exports 3D and semantic information from CityEngine for OpenDS. As CityEngine has a built in Python scripting interface, we will implement our tool as a Python script that can be executed in CityEngine.

3.1 Concept

We will develop the export tool in three steps: First we are going to develop a Python script that exports models from CityEngine in OBJ format and converts them to OGRE-XML. Second we will develop a script that exports only semantic information from CityEngine and saves the information in OpenDRIVE format. Finally we will create an integrated solution that exports 3D and semantic information from CityEngine and bring it in a format that can easily be shared with the OpenDS community.

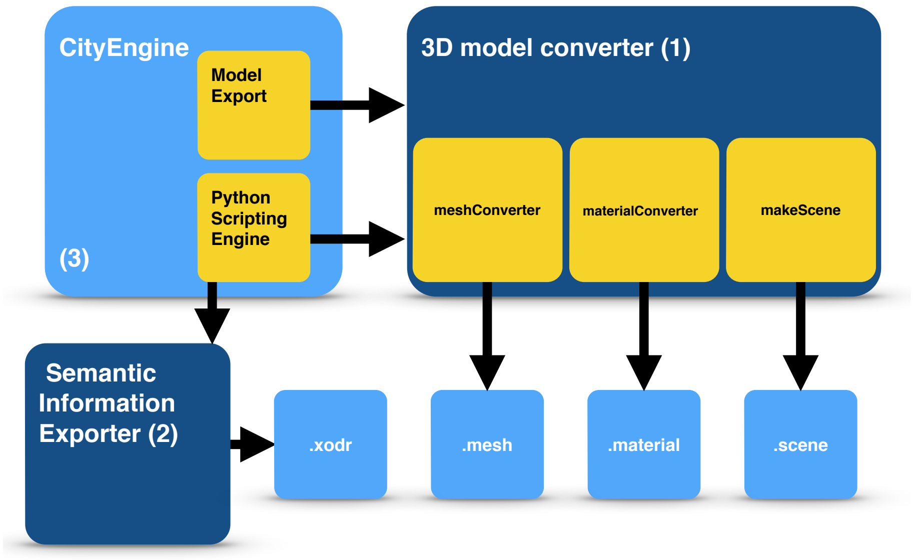


Figure 3.1: Implementation schema

Figure 3.1 shows the schema of the implementation. As we can see, the first part, the model converter, consists of three components: The “meshConverter”, that converts the OBJ geometry file to an OGRE-XML geometry file, the “materialConverter”, that converts the OBJ material file to an OGRE-XML material file and “makeScene”, that creates a scene file for OpenDS. This scene file gathers all exporter models and materials and make it easier to import them into OpenDS, because just one file has to be imported, instead of importing each model and material file on its own.

3.2 Model Export

As just mentioned, the model converter consists of three components. Two of these components are working with the OBJ files exported from CityEngine, one with the converted OGRE-XML export. Before something can be converted, we need to export the models in OBJ format. Fortunately the CityEngine Python-API has a command that allows us to easily export objects in OBJ format. While there are some options that have to be fixed in order to create models that are compatible with OpenDS (e.g. the usage of texture coordinates) other options can be set by the user in the script directly (cf. listing 3.1).

```
"""
///////////////////////////////////
SETTINGS
///////////////////////////////////
"""

#GENERAL
OUT_PATH = os.path.dirname(os.path.dirname(__file__))+'\models' #
    folder
FILE_NAME = "openDS" #String
EXPORT_CONTENT = "FALLBACK" #"FALLBACK", "MODEL"
TERRAIN_LAYERS = "TERRAIN_NONE" #"TERRAIN_ALL_VISIBLE", "
    TERRAIN_ALL_SELECTED", "TERRAIN_ALL", "TERRAIN_NONE"

#GRANULARITY
FILE_GRANULARITY = "MEMORY_BUDGET" #"MEMORY_BUDGET", "START_SHAPE"
MAX_FILESIZE = 500 #mb

#GEOMETRY
OFFSET = [0,0,0]#x,y,z

#DEBUG
KEEP_OBJ = False #True;False

"""
///////////////////////////////////
"""
```

Listing 3.1: Settings section in the model converter

More information about these settings can be found in the manual of the script (see section A.1 in the appendix). After the export finished, the script iterates over all exported OBJ geometry files and calls the meshConverter for each file.

The meshConverter reads each file line by line and stores all information in lists of vertices (for vertices, texture coordinates, and normals) or polygons (cf. section 2.3.1), which are defined in the geometry class that comes with the script. After reading a whole group, the meshConverter converts the content of the group in OGRE-XML and saves it. While the conversion in general is very simple and does not need many changes, there are two things that have to be minded: While OBJ starts counting with 1, OGRE-XML starts with 0, and while OBJ counts vertex numbers global, OGRE-XML counts them per group.

After all geometry files are converted, the script iterates over all OBJ material files and calls the materialConverter for each. Again each file is read line by line. The conversion of the material files is pretty straight forward, because only keywords have to be exchanged and no further conversion has to be done (cf. section 2.3.1).

Finally the script starts makeScene and creates a `.scene` file containing all mesh and material files. Overall the script for the model export consist of about 520 lines of code. Figure 3.2 shows the rough structure of the code.

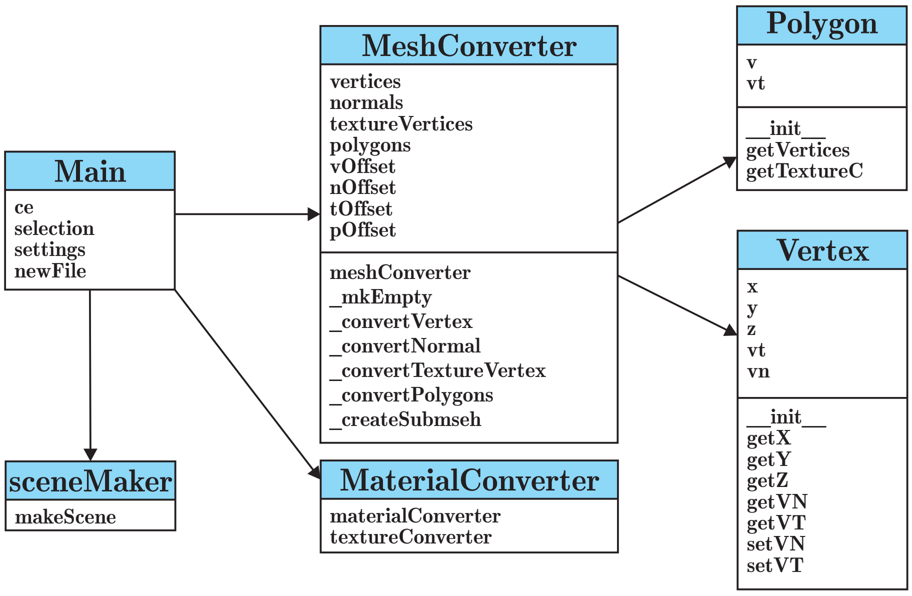


Figure 3.2: Structure of the model exporter implementation

For sharing the script with users, all code can be put in one file. Section A.1 in the appendix describes how the exporter can be used with CityEngine.

3.3 Semantic Information Export

First, we will take a look at the implementation of the semantic information export for the street network. Before we can export the information, we have to choose a starting point for our street network. Although we could choose any section to be the first it can be useful to not choose a random section. We distinguish three types of street networks:

1. circular networks: each street segment adjoin to at least two other street segments
2. non circular networks: at least one street is a dead end
3. independent networks: two or more street segments are not connected, i.e. we have two or more independent road networks

For case one it obviously does not matter where we start our street network and we can choose a random segment. For case two it is advisable to choose a dead end street as start point as it makes it easier to iterate through the graph. For the third case we can handle each independent network according to the first two rules.

To do so, we first create a list of all marked nodes and search for a node, that has only one outgoing edge and make it the start node (cf. listing 3.2).

```
ce = CE()

#get selected nodes
selectedNodes = ce.getObjectsFrom(ce.selection(), ce.isGraphNode)

startNode = selectedNodes[0] #random node

#search for node with one outgoing edge
for node in selectedNodes:
    edges = ce.getObjectsFrom(node, ce.isGraphSegment)

    if(len(edges) < 2):
        startNode = node
        break
```

Listing 3.2: Search for the start node

Starting there, we have to go through the whole graph. To do so, we do a breadth-first search (BFS) over all edges that iterates trough the graph. If the script reaches a node that should not be exported or was already visited, this branch of the BFS stops. For each node with more than two outgoing edges, we create an intersection entry in the OpenDRIVE file, for each edge we create a road entry. While CityEngine describes a road segment with a start and an end point, OpenDRIVE uses a starting point, the length of the segment and its angle. Given the start point $A = (x_1, y_1)$ and the end point $B = (x_2, y_2)$, we can easily calculate the length l and the angle α:

$l = |\overrightarrow{AB}| = |(x_2 - x_1, y_2 - y_1)| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$

$m = \frac{\Delta y}{\Delta x} = \frac{y_2 - y_1}{x_2 - x_1}$

$\alpha = arctan(m)$

The advantage of this method is, that we only use information from the CityEngine street network and this information does not depend on the CGA rules (cf. section 2.2.3). On the other hand we have some disadvantages: We only have one reference line in the middle of the street, we do not have detailed information about specific lanes or if there is a central reservation. Furthermore the reference line does not describe the accurate angle of curves, but rather an approximation.

To solve this problems we would have to take the CGA rules into account. But this would means that we have to set up name conventions for CGA rules and our script would not work with rules that do not obey these rules. Therefore we decided to not take the CGA rules into account, for the time being.

The situation for buildings is a bit easier, as we do not have any guaranteed information (except for the outlines) from CityEngine and no fixed structure in OpenDRIVE. We can just iterate over all selected buildings and write an `<object>`-entry for each of them in the road element next to them. For each parameter defined in the CGA file (with the report command), we add an `<userData>`-element to the object.

3.4 Integrated Solution

The aim of our integrated solution is an export plugin for CityEngine that is easy to use, easy to install and allows the user to export semantic and 3D information for OpenDS in one step. As we already implemented both exporters, we now just have to put them together into one file and add some feedback for the user.

```
#export selected models as obj
print "Step␣1/6:␣Exporting␣models"
ce.export(selection, settings)
print "Done!"

#convert meshes
print "Step␣2/6:␣Convert␣meshes"
for filename in glob.glob(os.path.join(settings.getOutputPath(), settings.getBaseName()+'*.obj')):
        meshFiles.append(os.path.splitext(os.path.basename(filename))[0])
        newFile = os.path.splitext(os.path.basename(filename))[0]+".mesh.xml"
        meshConverter(filename ,settings.getOutputPath()+"/"+newFile)
        if(not KEEP_OBJ):
            os.remove(filename)
print "Done!"

#convert materials
print "Step␣3/6:␣Convert␣materials"
for filename in glob.glob(os.path.join(settings.getOutputPath(), settings.getBaseName()+'*.mtl')):
        materialFiles.append(os.path.splitext(os.path.basename(filename))[0])
        newFile = os.path.splitext(os.path.basename(filename))[0]+".material"
        materialConverter(filename ,settings.getOutputPath()+"/"+newFile)
        if(not KEEP_OBJ):
            os.remove(filename)
print "Done!"

#check for tif textures
print "Step␣4/6:␣Check␣textures"
for filename in glob.glob(os.path.join(settings.getOutputPath(), '*.tif')):
   print "WARNING:␣TIF-textures␣found,␣pleas␣convert␣them␣to␣png"
   break
print "Done!"

#generate scene-file
print "Step␣5/6:␣Generate␣scene-file"
makeScene(meshFiles, materialFiles, settings.getOutputPath()+"/"+settings.getBaseName()+".scene")
print ("Done!")

#export semantic information
print "Step␣6/6:␣Export␣Semantic␣information"
makeOpenDrive(selection, settings.getOutputPath()+"/"+settings.getBaseName()+".xodr")
print ("Done!␣Files␣saved␣at␣" + settings.getOutputPath() + "")

print "Stop␣openDS-Exporter"
```

Listing 3.3: Integrated solution with user feedback

The integrated solution in listing 3.3 now just consists of one file and is therefore easy to share and can be used just like the script that is described in section A.1 of the appendix. The files exported by the integrated solution could now be used in OpenDS. In order to connect 3D and semantic information, OpenDS could use the position information from both formats and match them. This is necessary because the 3D-models do not have unique identifiers that could be matched with the OpenDRIVE identifiers.

4 Evaluation

Since semantic information from OpenDRIVE files can not be used in OpenDS yet, the evaluation focuses on the model export. We used two different methods to evaluate our results. First we evaluated the models that were exported mathematically and compared them, second we did a user survey and asked users how satisfied they are with the usability and the output of our export tool. The results of the usability questions are also applicable for the export of semantic information (more or less), because the workflow is the same.

For both methods, we wanted to compare our export tool with OGRE-XML files that were generated with Blender from the CityEngine OBJ files, as this was the best practice so far.

4.1 Mathematical Evaluation

To evaluate the exported models, we compared 10 different models, with regard to the number of vertices and polygons of the models, each in three versions:

1. OBJ files generated from CityEngine
2. OGRE-XML files generated with Blender from OBJ files
3. OGRE-XML files generated with our export script

We expected that the number of vertices and polygons are equal for all three versions, what means that we converted the models one-to-one. A lower number of vertices or polygons would mean a loss of details. Surprisingly, we found out that the OGRE-XML export plugin for Blender adds additional vertices during the export. Here are three examples from the evaluated models:

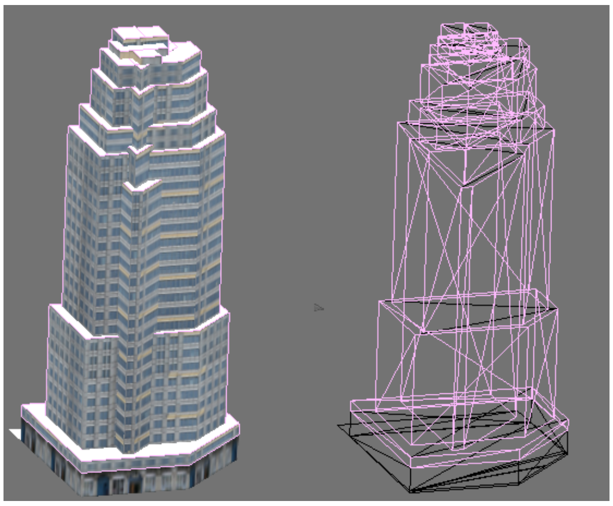


Figure 4.1: First model: skyscraper (textured and wireframe)

Export Method	**#vertices**	**#polygons**
CityEngine	353	183
Blender	363	183
Export script	353	183

Table 4.1: Evaluation of the first model

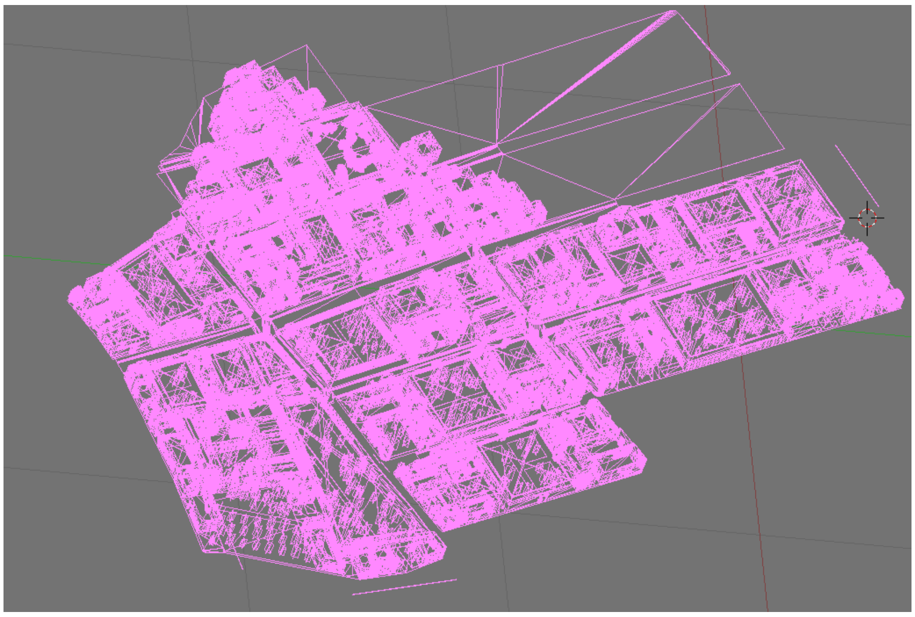


Figure 4.2: Second model: District of an antique city (wireframe)

Export Method	**#vertices**	**#polygons**
CityEngine	596981	402039
Blender	1007995	402039
Export script	596981	402039

Table 4.2: Evaluation of the second model

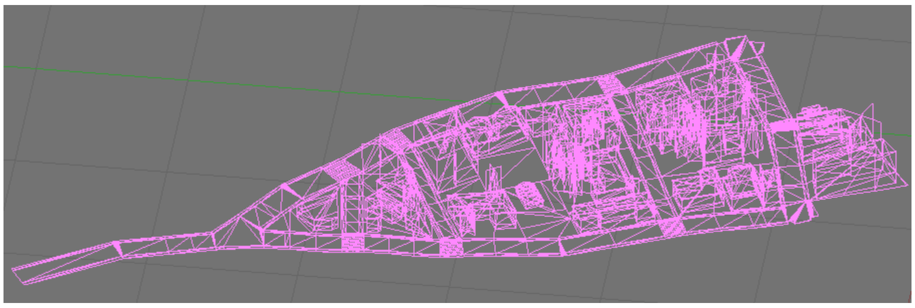


Figure 4.3: Third model: District of an modern city (wireframe)

Export Method	**#vertices**	**#polygons**
CityEngine	7904	5436
Blender	10916	5436
Export script	7904	5436

Table 4.3: Evaluation of the third model

These examples show, that, while the models converted with our script are one-to-one copies, the models exported with Blender have up to 50% more vertices compared to the original models. The smaller the models are, the smaller the is the difference between the original model and the Blender export.

This was unexpected, because the models do not benefit from these additional vertices (you can not add additional information that was not contained in the original model) but we can expect that the additional vertices lead to lower performance in the simulator. We tried to measure the performance difference in the form of fps[1] rates but although we saw a tendency to lower fps rates (up to 10%), the values were too inconstant and vague to draw a final conclusion.

4.2 User Survey Evaluation

The user survey was realized using the open source on-line survey tool LimeSurvey[2]. A print version of the questionnaire can be found in section A.2. The questionnaire consists of four parts and a text box for additional feedback. The first part consists of questions about the knowledge of the user about CityEngine, OpenDS and Blender. The second part is about the usability of the export tool and is based on ISONORM 9241/10[19] and the Computer System Usability Questionnaire (CSUQ)[13]. The third part is about the appearance and performance of the generated models, while the last part asks about the age, the sex and the profession of the user.

As we wanted to compare our export tool with the Blender export, we were looking for participants that are familiar with OpenDS, CityEngine and Blender. Unfortunately, we found only five participants for our survey, but four of them were familiar with CityEngine (cf. figure 4.4), four with OpenDS (cf. figure 4.5) and five with Blender (cf. figure 4.6) and all of them already exported models from CityEngine to OpenDS using Blender.

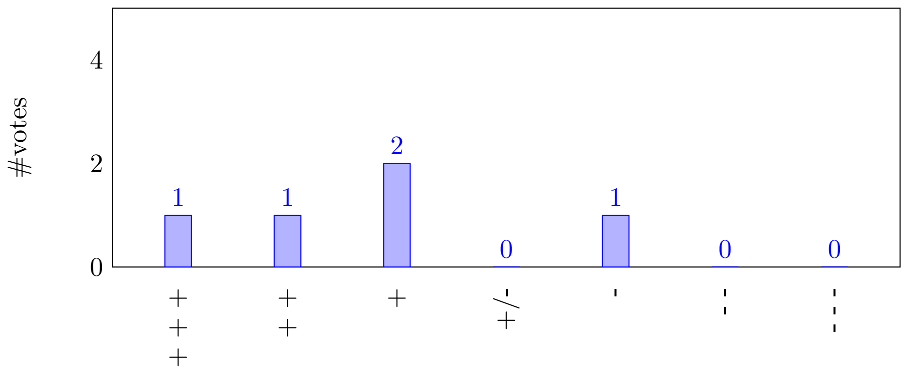


Figure 4.4: Results for: "I am familiar with CityEngine"

[1]frames per second
[2]`http://www.limesurvey.org/`

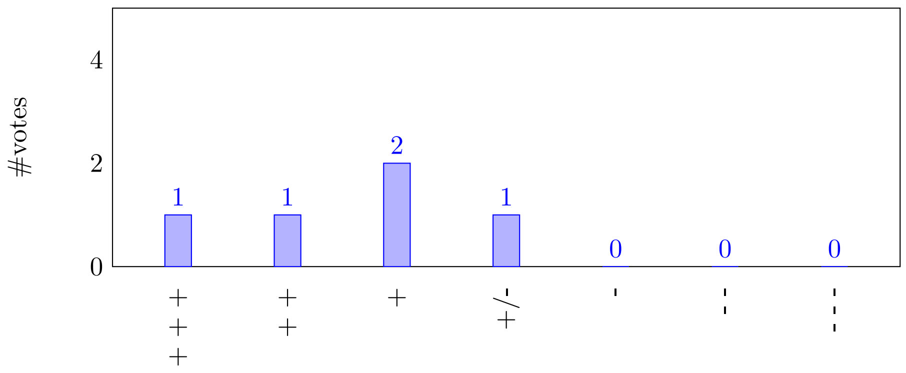


Figure 4.5: Results for: “I am familiar with OpenDS”

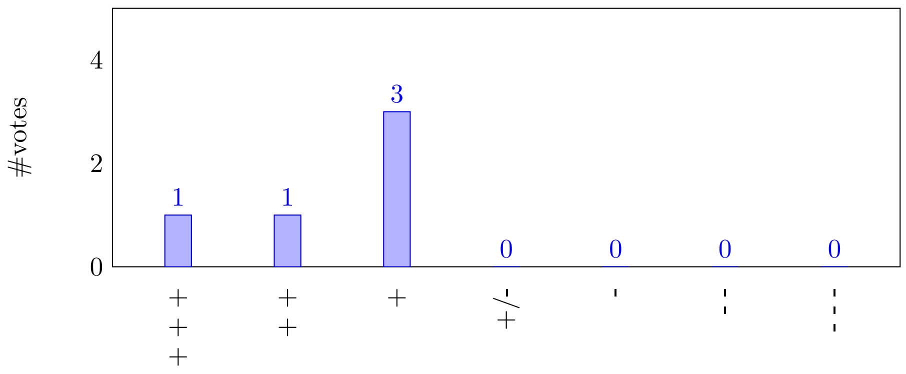


Figure 4.6: Results for: “I am familiar with Blender”

While all participants were satisfied with how easy it was to use the system (cf. figure 4.7) and thought that the tool made the export easier (cf. figure 4.8), only one of them thought that the system gave error messages that clearly told him how to fix problems (cf. figure 4.9). The reason might be, that CityEngine only allows script output on the Python console (cf. figure A.2), which is automatically hidden during export.

Another negative point was the performance of the exported models in OpenDS. Two participants thought the performance of the models exported with Blender was better, compared to the export with the script, three participants could not see a difference (cf. figure 4.7). As our mathematical evaluation showed that this is objectively not true, the reason may be, that Blender offers an optimization function during the export, that causes better performance and less details. The results of the survey show that this function is used often and could be added to future releases of the export script.

Nonetheless, overall all participants were more (or equal) satisfied with the models generated by the export script (cf. figure 4.11). All results of the survey can be found in section A.3.

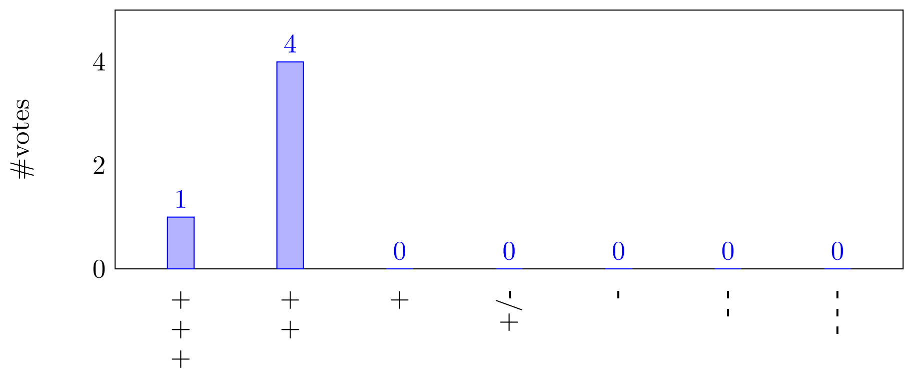


Figure 4.7: Results for: “Overall, I am satisfied with how easy it is to use this system.”

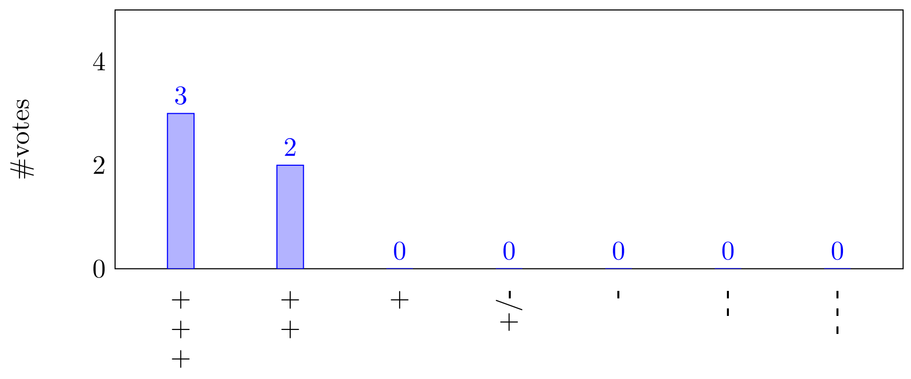


Figure 4.8: Results for: “Overall, this system made the export easier.”

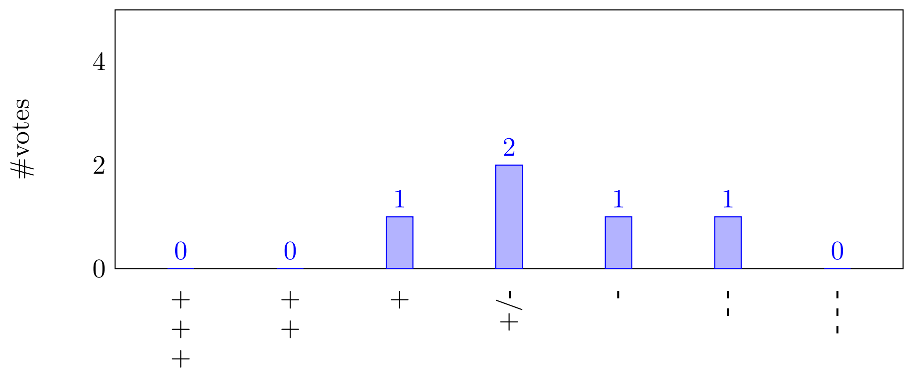


Figure 4.9: Results for: “The system gave error messages that clearly told me how to fix problems.”

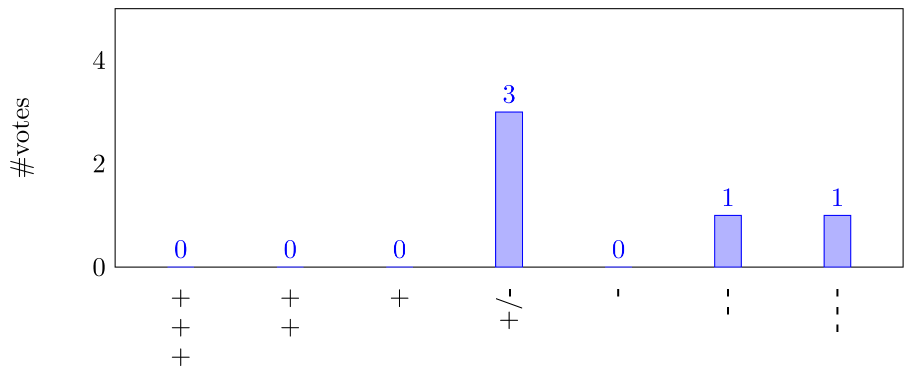


Figure 4.10: Results for: “The performance of the exported models in openDS is better.”

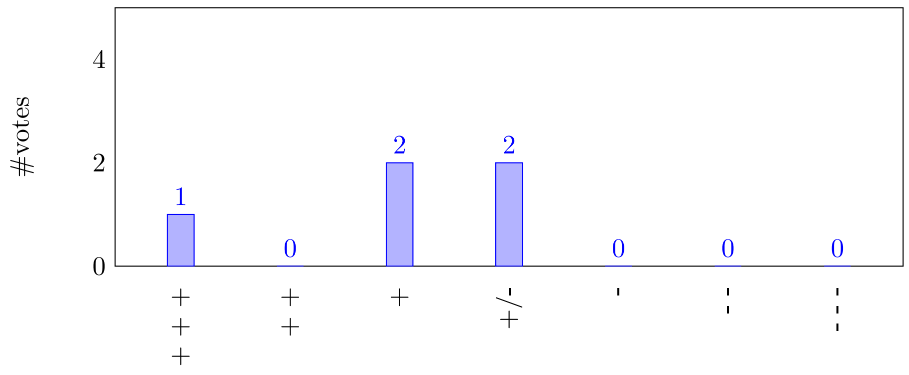


Figure 4.11: Results for: “Overall, I was more satisfied with the generated models.”

5 Summary and Outlook

5.1 Conclusion

In this thesis we examined the possibilities of procedural modelling for driving simulations in urban areas with regard to the creation and usage of semantic information. We have found out that semantic information can help to improve driving simulations, but there is no fixed set of semantic information that is always interesting or available for models generated with procedural modelling.

Moreover, we compared three procedural modelling applications and showed that none of them can export semantic information, although this information is available. As we compared different file formats, we concluded that the best way of saving semantic information for driving simulations is the OpenDRIVE format and we should save 3D and semantic information in different formats for compatibility reasons.

Finally we implemented a prototype that processes and saves semantic information from the procedural modelling rules of CityEngine and exports them, together with 3D information, for OpenDS. We evaluated the export script mathematically and based on an user survey. The evaluation of the script and its models showed that the script works, is easy to use and the models are one-to-one copies, as we expected it.

5.2 Future Work

The next logical step is the integration of OpenDRIVE support in OpenDS. After the integration, the semantic exporter should be evaluated with OpenDS and OpenDS could be extended by new features, like navigation. The 3D model exporter could be extended by an optimization function for complex 3D-models. Moreover the prototype of the semantic information exporter could be extended in the future, with regard to the support of different lanes and more detailed road networks.

Although we found out that there is no fixed set of semantic information that works for all cases, it could be useful to declare a minimal set of semantic information for buildings and streets that should always be available for OpenDS. In order to do this with CityEngine, it would be necessary to create a common standard for CGA rule files.

A Appendix

A.1 openDS-Exporter Manual

This manual provides you with a short introduction into the OpenDS-exporter for CityEngine. For more information about OpenDS please visit `www.opends.eu`, for more information about CityEngine please visit `www.esri.com`.

A.1.1 Installation

Open CityEngine and copy the "openDSExporter.py" file to the "scripts" folder of an existing CityEngine project (see figure A.1).

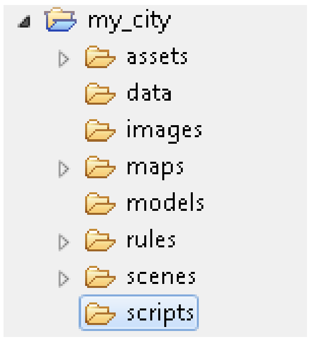


Figure A.1: Project folder in CityEngine

A.1.2 Usage

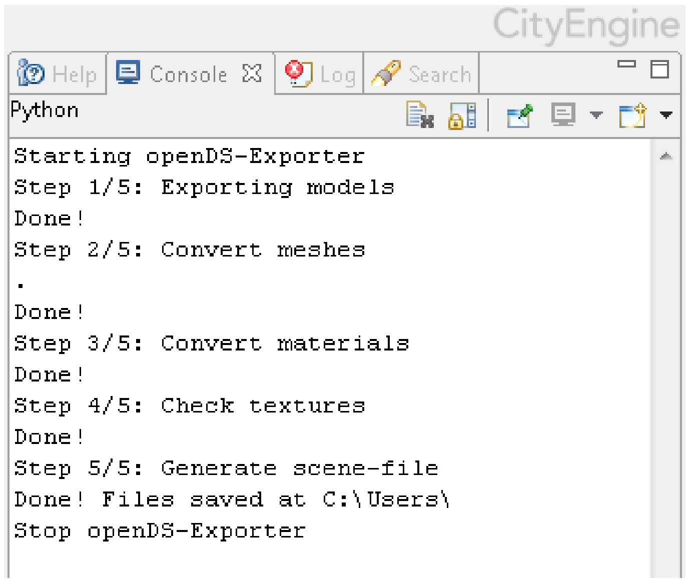
After you select the objects you want to export from your scene, there are two ways of using the openDS-exporter:

1. Open the model export dialogue from File > Export Models... (or press Ctrl + E) and select "Script Based Export (Python)". After that, click on Browse and select the "openDSExporter.py" file.
2. Double click on the "openDSExporter.py" in the "scripts" folder. Press F9 or choose Python > Run Script from the menu.

The export status is printed out in the python console on the right side of CityEngine (see figure A.2).

A.1.3 Settings

Some export parameters can be manipulated in the python file directly:



```
Starting openDS-Exporter
Step 1/5: Exporting models
Done!
Step 2/5: Convert meshes
.
Done!
Step 3/5: Convert materials
Done!
Step 4/5: Check textures
Done!
Step 5/5: Generate scene-file
Done! Files saved at C:\Users\
Stop openDS-Exporter
```

Figure A.2: Python console in CityEngine

OUT_PATH
Possible values: any folder path
Default value: `os.path.dirname(os.path.dirname(__file__))+'\models'`
Description: Defines the output folder.

FILE_NAME
Possible values: any string with valid characters for file names
Default value: `openDS`
Description: Defines the file names of the exported files.

EXPORT_CONTENT
Possible values: "FALLBACK", "MODEL"
Default value: `FALLBACK`
Description: Defines if export falls back to shapes if model can't be generated.

TERRAIN_LAYERS
Possible values: "TERRAIN_ALL_VISIBLE", "TERRAIN_ALL_SELECTED", "TERRAIN_ALL", "TERRAIN_NONE"
Default value: `TERRAIN_NONE`
Description: Defines which terrain layers will be exported.

FILE_GRANULARITY
Possible values: "MEMORY_BUDGET", "START_SHAPE"
Default value: `MEMORY_BUDGET`
Description: Defines the partition of the generated export into files.

MAX_FILESIZE
Possible values: file size in mb
Default value: `500`
Description: If `FILE_GRANULARITY` is set to `MEMORY_BUDGET`, maximum file size is defined here.

OFFSET
Possible values: float triple
Default value: `[0,0,0]`

Description: Defines global model offset (x, y, z).

KEEP_OBJ
Possible values: True or False
Default value: `False`
Description: Defines if OBJ files should be kept after conversion to OGRE-XML.

A.2 CityEngine OpenDS-Exporter User Survey: Questionnaire

Welcome to the CityEngine openDS-Exporter user survey.

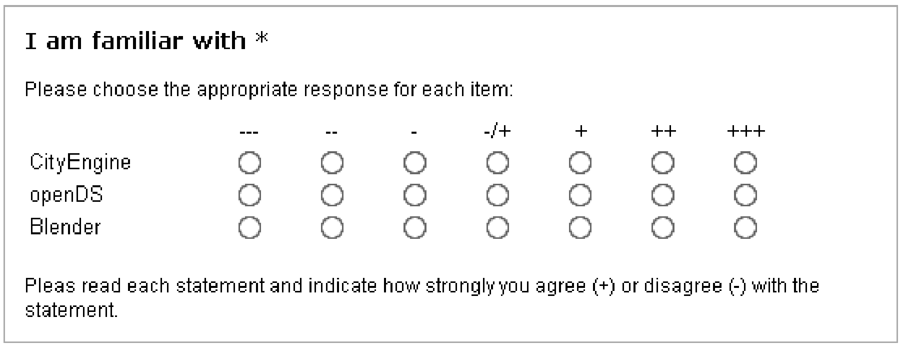
Please fill out this quick survey to help us improving our tool.

If you have any questions, please mail to: daniel.braun@dfki.de

There are 11 questions in this survey.

Background

Please give us some information about your technical background.

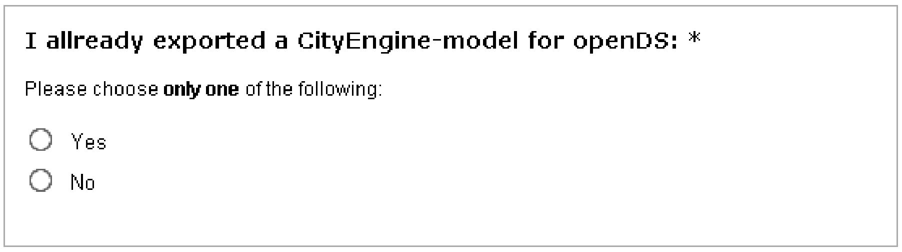


I am familiar with *

Please choose the appropriate response for each item:

	---	--	-	-/+	+	++	+++
CityEngine	○	○	○	○	○	○	○
openDS	○	○	○	○	○	○	○
Blender	○	○	○	○	○	○	○

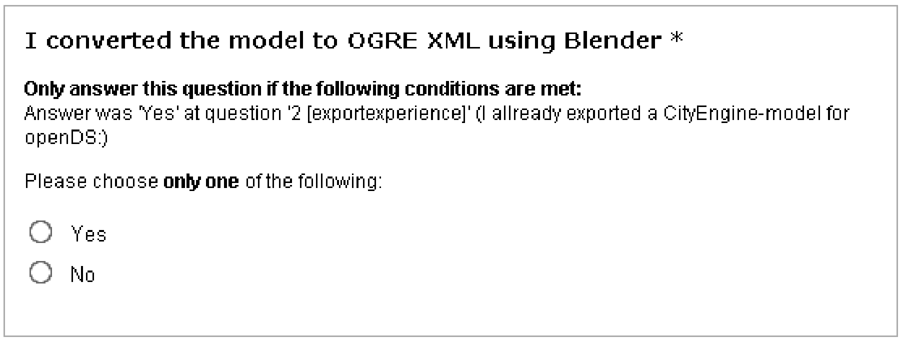
Pleas read each statement and indicate how strongly you agree (+) or disagree (-) with the statement.



I allready exported a CityEngine-model for openDS: *

Please choose **only one** of the following:

○ Yes
○ No



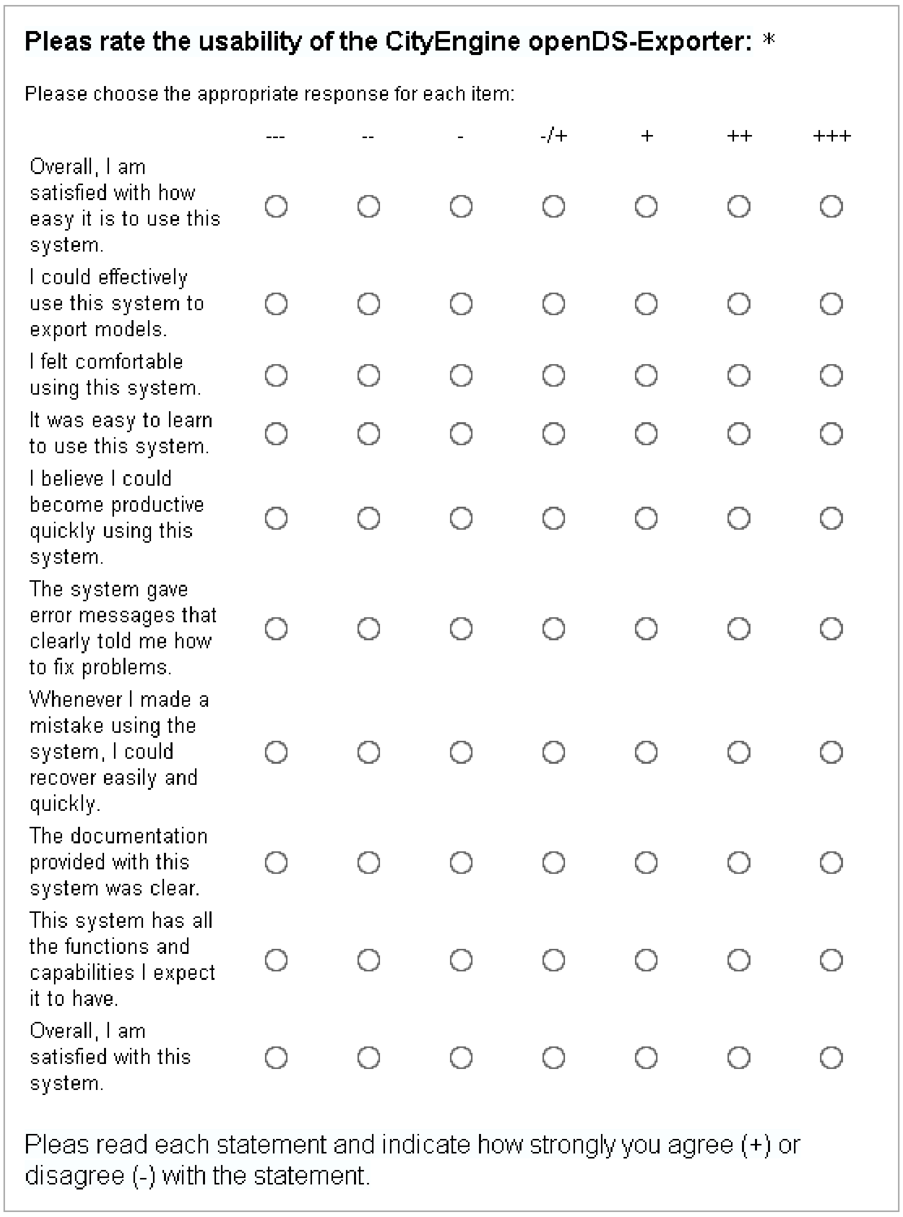
I converted the model to OGRE XML using Blender *

Only answer this question if the following conditions are met:
Answer was 'Yes' at question '2 [exportexperience]' (I allready exported a CityEngine-model for openDS:)

Please choose **only one** of the following:

○ Yes
○ No

Usability

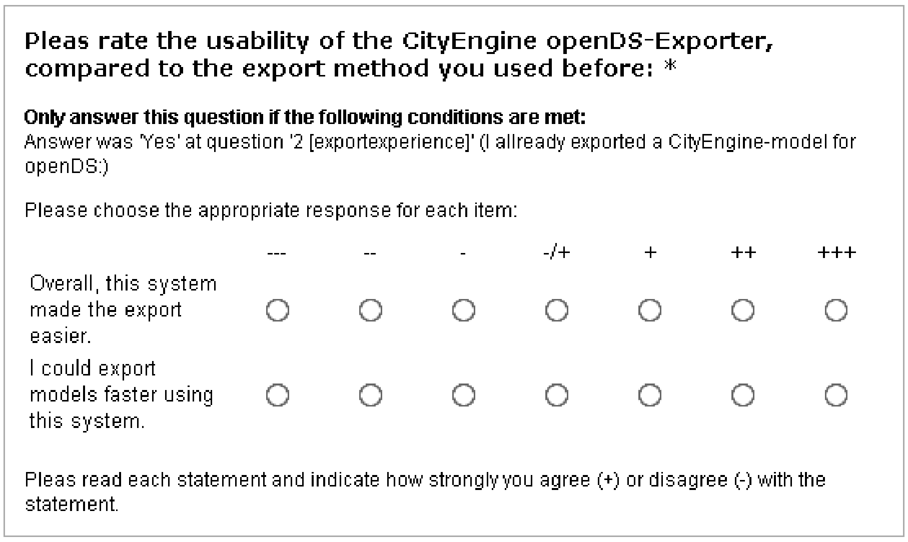


Pleas rate the usability of the CityEngine openDS-Exporter: *

Please choose the appropriate response for each item:

	---	--	-	-/+	+	++	+++
Overall, I am satisfied with how easy it is to use this system.	○	○	○	○	○	○	○
I could effectively use this system to export models.	○	○	○	○	○	○	○
I felt comfortable using this system.	○	○	○	○	○	○	○
It was easy to learn to use this system.	○	○	○	○	○	○	○
I believe I could become productive quickly using this system.	○	○	○	○	○	○	○
The system gave error messages that clearly told me how to fix problems.	○	○	○	○	○	○	○
Whenever I made a mistake using the system, I could recover easily and quickly.	○	○	○	○	○	○	○
The documentation provided with this system was clear.	○	○	○	○	○	○	○
This system has all the functions and capabilities I expect it to have.	○	○	○	○	○	○	○
Overall, I am satisfied with this system.	○	○	○	○	○	○	○

Pleas read each statement and indicate how strongly you agree (+) or disagree (-) with the statement.



Pleas rate the usability of the CityEngine openDS-Exporter, compared to the export method you used before: *

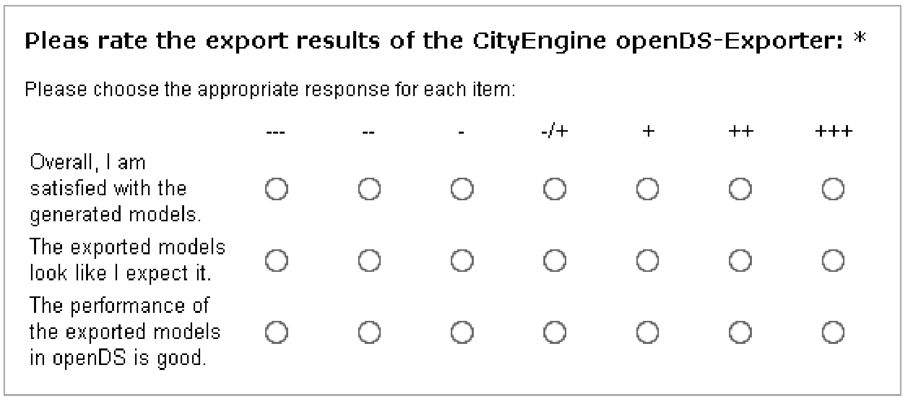
Only answer this question if the following conditions are met:
Answer was 'Yes' at question '2 [exportexperience]' (I allready exported a CityEngine-model for openDS:)

Please choose the appropriate response for each item:

	---	--	-	-/+	+	++	+++
Overall, this system made the export easier.	○	○	○	○	○	○	○
I could export models faster using this system.	○	○	○	○	○	○	○

Pleas read each statement and indicate how strongly you agree (+) or disagree (-) with the statement.

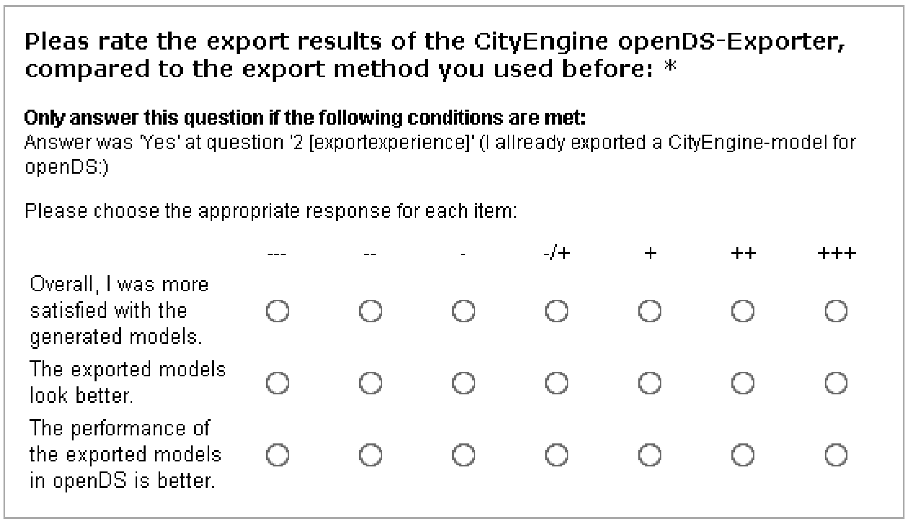
Export Results



Pleas rate the export results of the CityEngine openDS-Exporter: *

Please choose the appropriate response for each item:

	---	--	-	-/+	+	++	+++
Overall, I am satisfied with the generated models.	○	○	○	○	○	○	○
The exported models look like I expect it.	○	○	○	○	○	○	○
The performance of the exported models in openDS is good.	○	○	○	○	○	○	○

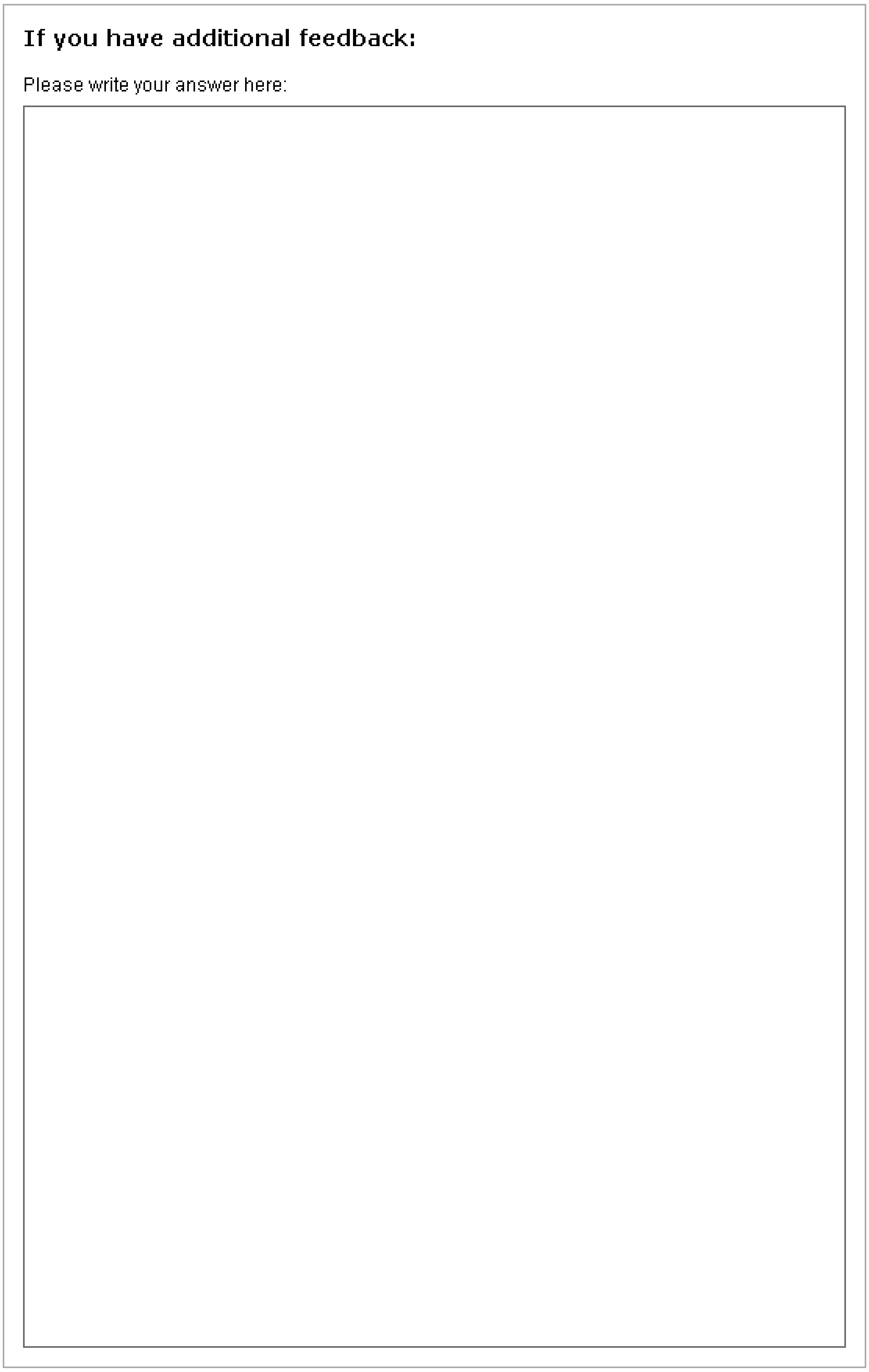


Pleas rate the export results of the CityEngine openDS-Exporter, compared to the export method you used before: *

Only answer this question if the following conditions are met:
Answer was 'Yes' at question '2 [exportexperience]' (I allready exported a CityEngine-model for openDS:)

Please choose the appropriate response for each item:

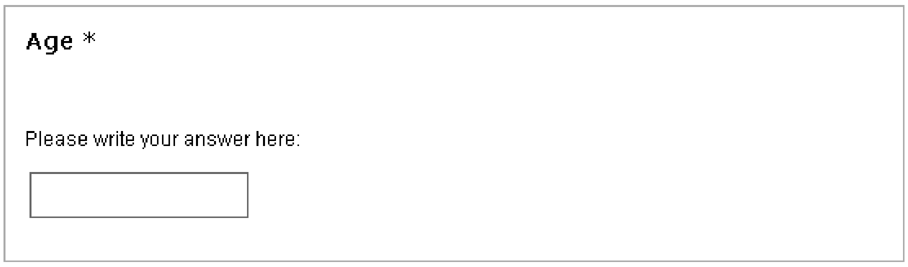
	---	--	-	-/+	+	++	+++
Overall, I was more satisfied with the generated models.	○	○	○	○	○	○	○
The exported models look better.	○	○	○	○	○	○	○
The performance of the exported models in openDS is better.	○	○	○	○	○	○	○



If you have additional feedback:

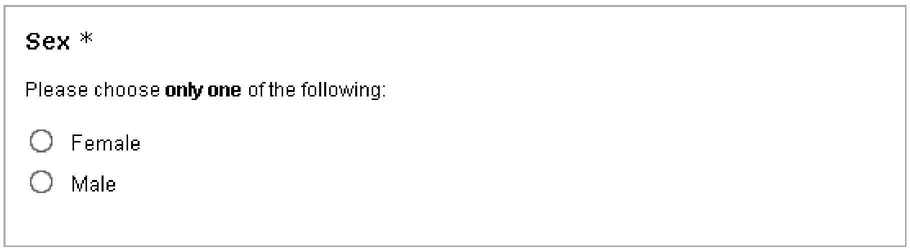
Please write your answer here:

Personal Information



Age *

Please write your answer here:

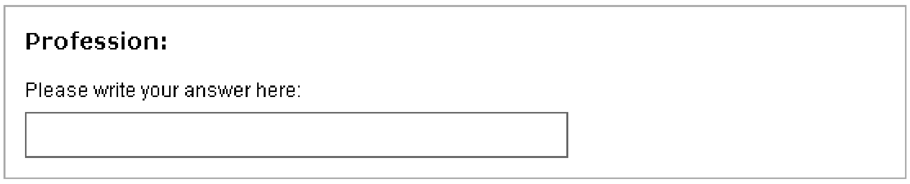


Sex *

Please choose **only one** of the following:

○ Female

○ Male



Profession:

Please write your answer here:

A.3 CityEngine OpenDS-Exporter User Survey: Results

Question ID	**Question**
Q1	I am familiar with [CityEngine]
Q2	I am familiar with [openDS]
Q3	I am familiar with [Blender]
Q4	I allready exported a CityEngine-model for openDS:
Q5	I converted the model to OGRE XML using Blender
Q6	Pleas rate the usability of the CityEngine openDS-Exporter: [Overall, I am satisfied with how easy it is to use this system.]
Q7	Pleas rate the usability of the CityEngine openDS-Exporter: [I could effectively use this system to export models.]
Q8	Pleas rate the usability of the CityEngine openDS-Exporter: [I felt comfortable using this system.]
Q9	Pleas rate the usability of the CityEngine openDS-Exporter: [It was easy to learn to use this system.]
Q10	Pleas rate the usability of the CityEngine openDS-Exporter: [I believe I could become productive quickly using this system.]
Q11	Pleas rate the usability of the CityEngine openDS-Exporter: [The system gave error messages that clearly told me how to fix problems.]
Q12	Pleas rate the usability of the CityEngine openDS-Exporter: [Whenever I made a mistake using the system, I could recover easily and quickly.]
Q13	Pleas rate the usability of the CityEngine openDS-Exporter: [The documentation provided with this system was clear.]
Q14	Pleas rate the usability of the CityEngine openDS-Exporter: [This system has all the functions and capabilities I expect it to have.]
Q15	Pleas rate the usability of the CityEngine openDS-Exporter: [Overall, I am satisfied with this system.]
Q16	Pleas rate the usability of the CityEngine openDS-Exporter, compared to the export method you used before: [Overall, this system made the export easier.]
Q17	Pleas rate the usability of the CityEngine openDS-Exporter, compared to the export method you used before: [I could export models faster using this system.]
Q18	Pleas rate the export results of the CityEngine openDS-Exporter: [Overall, I am satisfied with the generated models.]
Q19	Pleas rate the export results of the CityEngine openDS-Exporter: [The exported models look like I expect it.]
Q20	Pleas rate the export results of the CityEngine openDS-Exporter: [The performance of the exported models in openDS is good.]
Q21	Pleas rate the export results of the CityEngine openDS-Exporter, compared to the export method you used before: [Overall, I was more satisfied with the generated models.]
Q22	Pleas rate the export results of the CityEngine openDS-Exporter, compared to the export method you used before: [The exported models look better.]
Q23	Pleas rate the export results of the CityEngine openDS-Exporter, compared to the export method you used before: [The performance of the exported models in openDS is better.]

Table A.1: User survey results: Question IDs

User	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Q9	Q10	Q11
1	+++	+++	+	Yes	Yes	+++	+++	+++	+++	+++	+
2	+	+	++	Yes	Yes	++	++	+	+++	++	-/+
3	-	-/+	+	Yes	Yes	++	++	+	+++	+++	-/+
4	++	++	+	Yes	Yes	++	++	++	++	++	- -
5	+	+	+++	Yes	Yes	++	++	++	++	++	-

Table A.2: User survey results: Q1 - Q11

User	Q12	Q13	Q14	Q15	Q16	Q17	Q18	Q19	Q20	Q21	Q22	Q23
1	+	+++	++	+++	+++	+++	+++	+++	+	+++	+++	-/+
2	-/+	++	-	++	++	+++	++	++	-	+	+	-/+
3	-/+	+++	-/+	+	+++	+++	+++	+++	-/+	-/+	-/+	-/+
4	-	++	+	+	++	++	+	+	- -	+	+	- - -
5	-	++	-/+	+	+++	+++	++	++	- -	-/+	++	- -

Table A.3: User survey results: Q12 - Q23

Bibliography

[1] Kristen E Beede and Steven J Kass. Engrossed in conversation: The impact of cell phones on simulated driving performance. *Accident Analysis & Prevention*, 38(2):415–421, 2006.

[2] Linda Rose Diane Ramey and Lisa Tyerman. Mtl material format (lightwave, obj). Technical report, Alias—Wavefront, Inc., 1995.

[3] Johannes Diemke. Wavefront obj format. In *Übung im Modul OpenGL mit Java*. Carl von Ossietzky Universität Oldenburg, 2010.

[4] Christoph Endres, Rafael Math, and Daniel Braun. Simulator-based evaluation on the impact of visual complexity and speed on driver's cognitive load. In *Adjunct Proceedings of the 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI 2012)*, pages 30–31, 2012.

[5] Luciano Floridi. Is semantic information meaningful data? *Philosophy and Phenomenological Research*, 70(2):351–370, 2005.

[6] UAB Center for Research on Applied Gerontology. Uab driving simulator core. `http://crag.uab.edu/drivesim/index.asp`, 2013. Accessed: 2014-05-01.

[7] VIRES Simulationstechnologie GmbH. Opendrive® / opencrg® product data sheet. Technical report, VIRES Simulationstechnologie GmbH, 2011.

[8] G Gröger, TH Kolbe, C Nagel, and K Häfele. Opengis city geography markup language (citygml) encoding standard v2. 0.0. *Open Geospatial Consortium Standard. Open Geospatial Consortium*, 2012.

[9] Tobias Haubrich, Sven Seele, Rainer Herpers, Martin E. Müller, and Peter Becker. Semantic road network models for rapid 3d traffic scenario generation. In *Tagungsband ASIM/GI-Fachgruppentreffen STS/GMMS, Workshop Simulation technischer Systeme - Grundlagen und Methoden in Modellbildung und Simulation*, Simulation technischer Systeme und Grundlagen und Methoden in Modellbildung und Simulation, pages 51–55. Arbeitsgemeinschaft Simulation ASIM, February 2013.

[10] Nicolas Hiblot, Dominic Gruyer, Jean-Sébastien Barreiro, and Bertrand Monnier. Pro-sivic and roads. a software suite for sensors simulation and virtual prototyping of adas. In *Proceedings of DSC*, 2010.

[11] System Technology Inc. *M100K STISIM Drive Software Datasheet*, 2012.

[12] Kwangsoo Kim and Dong-il Cho. A multi-vehicle platoon simulator. In *FISITA World Automotive Congress*, 2000.

[13] James R Lewis. Ibm computer usability satisfaction questionnaires: psychometric evaluation and instructions for use. *International Journal of Human-Computer Interaction*, 7(1):57–78, 1995.

[14] Rafael Math, Angela Mahr, Mohammad Mehdi Moniri, and Dr. Christian Müller. Opends: A new open-source driving simulator for research. In Andrew L. Kun, Linda Ng Boyle, Bryan Reimer, and Andreas Riener, editors, *Adjunct Proceedings of the 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications. International Conference on Automotive User Interfaces and Interactive Vehicular Applications (AutomotiveUI-12), October 17-19, Portsmouth, New Hampshire, USA*. ACM, ACM Digital Library, 2012.

[15] S. Mattes. The lane-change-task as a tool for driver distraction evaluation. In *Quality of Work and Products in Enterprises of the Future (Proceedings of the GfA/17th Annual Conference of the International Society for Occupational Ergonomics ans Safety, ISOES)*, pages 57–60, 2003.

[16] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. In *Acm Transactions On Graphics (Tog)*, volume 25, pages 614–623. ACM, 2006.

[17] Pascal Müller, Gang Zeng, Peter Wonka, and Luc Van Gool. Image-based procedural modeling of facades. In *ACM Transactions on Graphics (TOG)*, volume 26, page 85. ACM, 2007.

[18] Yoav IH Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 301–308. ACM, 2001.

[19] J Prümper and M Anft. Isonorm 9241/10–beurteilung von software auf grundlage der internationalen ergonomie-norm iso 9241/10, 1993.

[20] Thomas A Ranney, GH Baldwin, Scott M Vasko, and Elizabeth N Mazzae. Measuring distraction potential of operating in-vehicle devices. Technical report, National Highway Traffic Safety Administration, 2009.

[21] OKTAL S.A. Scaner^{TM} studio terrain - the unique road database creation tool. Technical report, OKTAL S.A., 2010.

[22] OKTAL S.A. Scaner^{TM} studio v1.0 - feature overview. Technical report, OKTAL S.A., 2010.

[23] OKTAL S.A. Driving simulator scientific references. `http://www.scanersimulation.com/scientific-papers.html`, 2012. Accessed: 2014-03-14.

[24] Ruben Michaël Smelik, Tim Tutenel, Klaas Jan de Kraker, and Rafael Bidarra. A declarative approach to procedural modeling of virtual worlds. *Computers & Graphics*, 35(2):352–363, 2011.

[25] Amanda N. Stephens and John A. Groeger. Following slower drivers: Lead driver status moderates driver's anger and behavioural responses and exonerates culpability. *Transportation Research Part F: Traffic Psychology and Behaviour*, 22(0):140 – 149, 2014.

[26] Gabriel Weiner and Bryant Walker Smith. Automated driving: Legislative and regulatory action. `http://cyberlaw.stanford.edu/wiki/index.php/Automated_Driving:_Legislative_and_Regulatory_Action`, 2014. Accessed: 2014-05-01.

[27] white c. Opends website - about. `http://opends.eu/index.php/introduction/about`, 2013. Accessed: 2014-03-14.